

CMOS 32-BIT SINGLE CHIP MICROCOMPUTER **E0C33 Family**

VOX33 MIDDLEWARE MANUAL



NOTICE

No part of this material may be reproduced or duplicated in any form or by any means without the written permission of Seiko Epson. Seiko Epson reserves the right to make changes to this material without notice. Seiko Epson does not assume any liability of any kind arising out of any inaccuracies contained in this material or due to its application or use in any product or circuit and, further, there is no representation that this material is applicable to products requiring high level reliability, such as medical products. Moreover, no license to any intellectual property rights is granted by implication or otherwise, and there is no representation or warranty that anything made in accordance with this material will be free from any patent or copyright infringement of a third party. This material or portions thereof may contain technology or the subject relating to strategic products under the control of the Foreign Exchange and Foreign Trade Control Law of Japan and may require an export license from the Ministry of International Trade and Industry or other approval from another government agency.

Windows95 and Windows NT are registered trademarks of Microsoft Corporation, U.S.A.

PC/AT and IBM are registered trademarks of International Business Machines Corporation, U.S.A.

All other product names mentioned herein are trademarks and/or registered trademarks of their respective owners.

PREFACE

This manual is intended for those who develop application systems using the E0C33 Family of microcomputers. It explains the configuration, functions, and usage method of VOX33 as voice compression/expansion middleware for the E0C33 Family.

CONTENTS

1 Outline of the VOX33 Middleware.....	1
1.1 Contents of the VOX33 Package.....	1
1.2 Basic Configuration of Voice Input/Output System.....	2
1.3 VOX33 Tools.....	3
2 Installation	5
2.1 Operating Environment	5
2.2 Method of Installation.....	6
3 Software Development Procedure.....	8
3.1 Creating Voice ROM Data using VOX33 Tools	9
3.1.1 Preparing Voice Data	10
3.1.2 Preprocessing 16-bit PCM Data	10
3.1.3 Evaluating Compression and Talking Speed/Tone Pitch Conversion	11
3.1.4 Converting Voice Data into an Assembly Source File.....	17
3.1.5 Precautions Concerning Voice ROM Data Creation.....	20
3.2 Creating a User Program and Linking the VOX33 Library.....	21
4 VOX33 Tool Reference	22
4.1 Outline of VOX33 Tools	22
4.2 Voice ROM Data Generation Tools	25
4.2.1 cnv48_8.exe.....	25
4.2.2 dct_cnv.exe	26
4.2.3 voxlvl.exe	27
4.2.4 voxflt.exe	28
4.2.5 pcm_norm.exe	29
4.2.6 vox2cmp.exe.....	30
4.2.7 voxcmps.exe	31
4.2.8 adpcmps.exe.....	32
4.2.9 vsxcmps.exe.....	33
4.2.10 ppccmps.exe.....	34
4.2.11 bin2s.exe	35
4.2.12 pcm2s.exe.....	36
4.2.13 bdmp.exe.....	37
4.2.14 vox2dec.exe	38
4.2.15 voxdec.exe.....	38
4.2.16 adpdec.exe	38
4.2.17 ppccdec.exe.....	38
4.2.18 vsxdec.exe.....	39
4.2.19 ampchk.exe	40
4.2.20 addslnt.exe.....	40
4.2.21 pcm2wav.exe	41
4.2.22 wav2pcm.exe	41
4.2.23 Executing Tools from a Batch File.....	42
4.2.24 Executing Tools from a Make File.....	47

CONTENTS

4.3 Voice Compression/Processing Evaluation Tools.....	50
4.3.1 vox2parm.exe	51
4.3.2 voxparam.exe	55
4.3.3 adpparam.exe.....	59
4.3.4 vsxparam.exe.....	62
4.3.5 vscparam.exe.....	66
4.4 VOX Parameters.....	70
4.4.1 Function of Each VOX Parameter.....	70
4.4.2 VOX Parameter Samples.....	72
5 VOX33 Library Reference.....	73
5.1 Outline of VOX33 Library.....	73
5.2 Hardware Resources and Initialization.....	75
5.3 Top-Level Functions.....	77
5.3.1 Compile Options.....	78
5.3.2 External Variables.....	79
5.3.3 Data Structure	80
5.3.4 Error Codes Returned by Top-Level Functions	80
5.3.5 VSX Data Processing Functions (vsxtop.c)	81
5.3.6 ADPCM Data Processing Functions (adptop.c)	84
5.3.7 VOX2 Data Processing Functions (vox2top.c).....	87
5.3.8 VOX Data Processing Functions (voxtop.c)	89
5.3.9 PCM Data Processing Functions (ppctop.c).....	92
5.3.10 Common Functions (voxcomn.c)	94
5.3.11 Input/Output Data Convert Functions (slutil.c).....	95
5.4 VOX33 Library Functions.....	97
5.4.1 VSX Processing Functions	99
5.4.2 ADPCM Processing Functions.....	102
5.4.3 VOX2 Processing Functions.....	104
5.4.4 VOX Processing Functions.....	106
5.4.5 VSC Processing Functions	108
5.4.6 PCM Processing Functions.....	110
5.4.7 Output (Speak) Functions.....	112
5.4.8 Input (Listen) Functions.....	116
5.4.9 High-Pass Filter Functions	120
5.5 Techniques for Speeding Up Operation.....	121
5.6 Library Performance and Memory Size.....	122
5.6.1 CPU Occupancy of VOX33 Library.....	122
5.6.2 Memory Sizes Used.....	124
5.7 Precautions	125
Appendix Verifying Operation with DMT33 Boards	126
A.1 System Configuration Using DMT33004.....	126
A.1.1 Hardware Configuration.....	126
A.1.2 Software	128
A.2 Program Execution Procedure	129
A.3 Program Examples	131
A.3.1 Explanation Concerning Files	131
A.3.2 make	136
A.4 When Using the DMT33005 Board.....	137

1 Outline of the VOX33 Middleware

VOX33 is voice compression/expansion middleware for the E0C33 Family. It is capable of performing voice compression/recording, expansion/reproduction, and talking speed/tone pitch conversion in real time on the E0C33 Family chip. Each function is offered as a library function which can be used after being linked with the target program. Also, a top-level function that performs the necessary voice processing after calling up these functions is provided as the C source. This helps to significantly reduce the programming burden involved in voice processing. In addition, the VOX33 package includes voice ROM generation tools that run on a PC and tools for evaluating compression and talking speed/tone pitch conversion performance.

The VOX33 middleware is suitable for developing such applications as voice memos, databanks with voice function, PDAs, and electronic stationery and toys.

Its main features are listed below:

- Can be used with the E0C33 Family chip that contains an A/D converter and 16-bit programmable timer
- Supports various voice compression formats
 1. Seiko Epson's original voice compression technology "VOX"
This technology uses speech analysis and synthesis to achieve a high compression ratio (8 kbps typ.).
 2. Seiko Epson's original voice compression technology "VSX"
Based on ADPCM, this technology accomplishes timebase compression and voice compression (12 kbps typ.).
 3. ADPCM (40 kbps, 32 kbps, 24 kbps, 16 kbps)
 4. Playback-only "VOX2" (high-sound-quality version of VOX)
Note: Not data compatible with the VOX format.

Thus, a wide selection of formats are available, depending on the desired compression ratio and sound quality.
- Voice processing technology "VSC" that allows the talking speed to be changed in the range of 1/2 to 2 times and the pitch (tone of voice) to be changed in the range of 1/2 (low) to 2 times (high)
- Capable of evaluating and verifying the compression and talking speed/tone pitch conversion functions using Windows GUI tools on a PC

Precautions

- Be sure to fully evaluate the operation of your application system before shipping. Seiko Epson will not assume any responsibility for problems arising from the use of this middleware in your commercial products.
- The rights to sell this middleware are owned solely by Seiko Epson. The resale rights are not transferable to any third party.
- All program files included in this package, except sample programs, are copyrighted by Seiko Epson. These files may not be reproduced, distributed, modified, or reverse-engineered without the written consent of Seiko Epson.

1.1 Contents of the VOX33 Package

The contents of the VOX33 package are listed below. After unpacking, check to see that all items are included with your package.

- | | |
|--|-------------------------------------|
| (1) Tool disk (CD-ROM) | 1 disk |
| (2) E0C33 Family VOX33 Middleware Manual (this manual) | 1 copy each in English and Japanese |
| (3) Warranty card | 1 card each in English and Japanese |

1.2 Basic Configuration of Voice Input/Output System

The basic hardware configuration of a voice input/output system is shown in Figure 1.2.1. This system is based on the E0C33 chip and incorporates external memory, amplifiers, a microphone, and a speaker.

Note that the VOX33 library uses one channel of the A/D converter and one or two channels of the 16-bit programmable timer on the E0C33 chip. It also uses some of the internal RAM to accelerate operation.

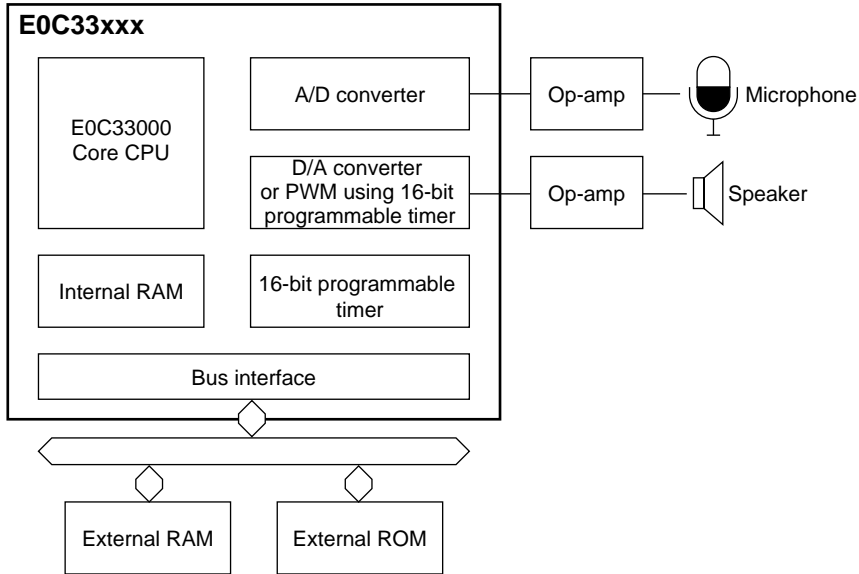


Figure 1.2.1 Hardware Configuration of Voice Input/Output System

The VOX33 library is a type of middleware positioned between the E0C33 hardware and the user program in order to assume the role of hardware control over voice processing. By incorporating or linking the top-level functions supplied in the C source file into or with the user program, voice processing can be accomplished easily without having to call up the VOX33 library functions directly from the user program.

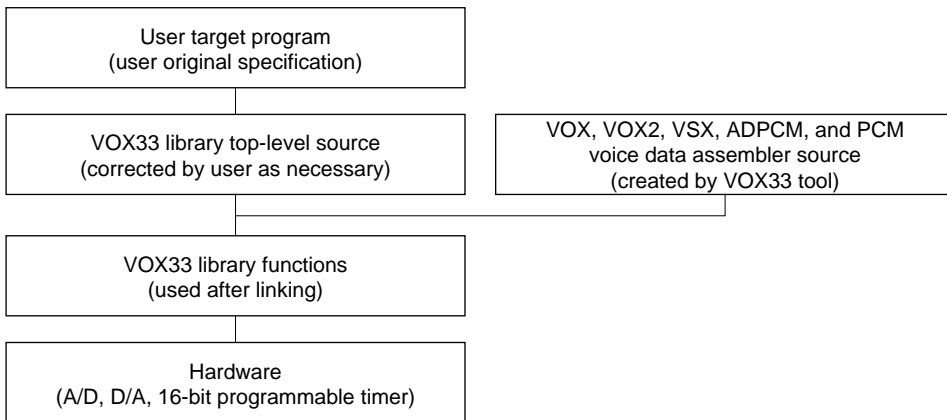


Figure 1.2.2 Software Configuration of Voice Input/Output System

For details on the VOX33 library functions and top-level functions, refer to Section 5, "VOX33 Library Reference".

1.3 VOX33 Tools

The VOX33 tools can be run on a personal computer to create the voice ROM data to be stored on the E0C33 Family chip, as well as to evaluate the voice compression and voice processing performance. All of these tools can be run under Windows 95, Windows NT 4.0, or higher versions.

Voice ROM data generation tools

The voice ROM data generation tools consist of a series of programs that convert, process, and compress the voice file (wav, pcm) to create an assembly source file for the E0C33. These tools are used to create the playback-only voice data to be written into ROM. Each program is a 32-bit application that can be executed from the DOS prompt.

Table 1.3.1 below lists the voice ROM data generation tools.

Table 1.3.1 Voice ROM Data Generation Tools

Tool	Function
cnv48_8.exe	A down-sampler used to convert a WAV file (48 kHz) into a 16-bit PCM file (8 kHz). Commercially available sound editors can be used, but may degrade sound quality.
dct_cnv.exe	A down-sampler used to convert a WAV or PCM file into any sampling rate. Commercially available sound editors can be used, but may degrade sound quality.
voxlvl.exe	Adjusts the level of 16-bit PCM data. It amplifies the low-level parts of voice data by a factor of 1.5 to 2 and attenuates high-level peaks by a factor of 3.
voxfilt.exe	Increases the clarity of sound by passing 16-bit PCM data through a high-pass filter.
addsint.exe	Appends a specified number of silent data to the 16-bit PCM data file.
pcm_norm.exe	Normalizes 16-bit PCM data to a 90% (default) amplitude to make it suitable for input to the voice compression tool. Also, if amplitude readjustment is required based on amplitude inspection results obtained by "ampchk.exe" after VOX compression, it makes the necessary adjustment.
ampchk.exe	Calculates the ratio between two PCM files (before and after VOX compression) and writes it to a file. This result can be input to "pcm_norm.exe" for amplitude readjustment in the PCM file.
pcm2wav.exe	Converts a PCM file into a WAV file.
wav2pcm.exe	Converts a WAV file into a PCM file.
voxcmprs.exe	Compresses 16-bit PCM data into VOX format based on VPM file (.vpm).
vox2cmp.exe	Compresses 16-bit PCM data into VOX2 format based on VPM file (.vpm).
adpcmprs.exe	Compresses 16-bit PCM data into ADPCM format based on the compression ratio specified by an option.
vsxcmprs.exe	Compresses 16-bit PCM data into VSX format based on the compression ratio specified by an option.
ppccmprs.exe	Compresses 16-bit PCM data into packed PCM format.
bin2s.exe	Converts a binary data file (VOX/VOX2 file, VSX file, VPM file, ADPCM file, or PPC file) into an assembly source file.
pcm2s.exe	Converts a 16-bit PCM file into a 10-bit amplitude assembly source file.
bdmp.exe	A utility used to dump binary data.
voxdec.exe	Decodes the voice data that has been compressed by "voxcmprs.exe" to save it as PCM data.
vox2dec.exe	Decodes the voice data that has been compressed by "vox2cmp.exe" to save it as PCM data.
adpdec.exe	Decodes the voice data that has been compressed by "adpcmprs.exe" to save it as PCM data.
vsxdec.exe	Decodes the voice data that has been compressed by "vsxcmprs.exe" to save it as PCM data.
ppcdec.exe	Decodes the voice data that has been compressed by "ppccmprs.exe" to save it as PCM data.

Voice compression/processing evaluation tools

Voice compression/processing evaluation tools are programs used to evaluate the sound quality after the input voice file has been VOX2-, VOX-, VSX-, ADPCM-, or VSC-compressed. Depending on the operating environment, they also support voice input from microphones. These tools allow you to examine the voice data compression ratio or talking speed/tone pitch conversion parameters. All these programs are 32-bit Windows GUI applications, and can display voice waveforms as you evaluate the sound quality.

Table 1.3.2 lists the voice compression/processing evaluation tools.

Table 1.3.2 Voice Compression/Processing Evaluation Tools

Tool	Function
voxparam.exe	After adjusting VOX parameters, it evaluates the quality of VOX-compressed sound and generates a VPM file.
vox2parm.exe	After adjusting VOX parameters, it evaluates the quality of VOX2-compressed sound and generates a VPM file.
vsxparam.exe	After adjusting VSX parameters, it evaluates the quality of VSX-compressed sound.
adpparam.exe	After adjusting ADPCM parameters, it evaluates the quality of ADPCM-compressed sound.
vscparam.exe	After adjusting tone pitch and talking speed, it evaluates the quality of VSC-compressed sound.

For details on the VOX33 tools, refer to Section 4, "VOX33 Tool Reference".

2 Installation

This section explains the operating environment for the VOX33 tools and how to install the VOX33 middleware.

2.1 Operating Environment

Software development and voice ROM data generation/evaluation using VOX33 require the following operating environment.

Personal computer

An IBM PC/AT or compatible is required. A model with Pentium 90 MHz or faster CPU and 32 MB or more of RAM is recommended.

Display

A display with a resolution of 800 × 600 pixels or more is required. For display, choose "small fonts" from the control panel.

Hard disk

Although the VOX33 tools and VOX33 library themselves require only about 7 MB of space, the hard disk must have sufficient space available for voice data. Consider the approximate data sizes shown below:

- 48 kHz, 16-bit monaural data (WAV file): Approx. 100KB/S, approx. 6MB/M, approx. 360MB/H
- 8 kHz, 16-bit monaural data (PCM file): Approx. 16KB/S, approx. 1MB/M, approx. 60MB/H
- 8 kbps, VOX-compressed data (VOX file): Approx. 1KB/S, approx. 60KB/M, approx. 3.6MB/H
- 8 kbps, VOX data assembly source file: Approx. 6KB/S, approx. 360KB/M, approx. 20MB/H (6 characters/byte on average)

CD-ROM drive

One CD-ROM drive is required for installing the software from CD-ROM.

Mouse

A mouse is required for operating the compression and processing evaluation tools.

Sound card, sound editor

DAT and a digital sound card are recommended for the creation of voice ROM data. When using an analog sound card, choose one with the highest possible quality.

The compression and processing evaluation tools require a sound card or on-board chip (compatible with SoundBlaster 16) that supports 8-kHz sampling and 16-bit monaural sound.

Choose a sound editor that can handle 48K WAV and 8K PCM data and can edit sound and save it a file.

System software

VOX33 tools run under Microsoft® Windows®95, Windows NT®, or higher versions (in Japanese or English).

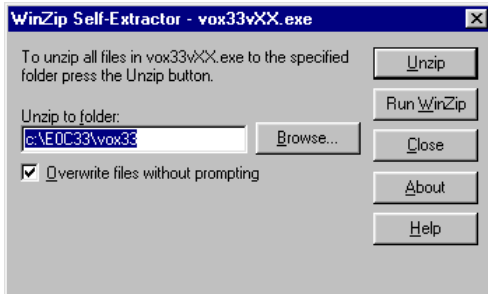
Other requirements

E0C33 Family C Compiler Package is required for software development.

2.2 Method of Installation

The VOX33 library and VOX33 tools are supplied on CD-ROM. Open the self-extracting file on the CD-ROM named "vox33vXX.exe" to install the VOX33 library and VOX33 tools in your computer. (The XX in this file name denotes a version number. For Version 1.0, for example, the file is named "vox33v10.exe".)

Double-click on "vox33vXX.exe" to start installation. The dialog box shown below appears.



Enter the path and folder name under which you want to install the files in the text box and click on the [Unzip] button. The specified folder is created and all files are copied into it. If the specified folder already exists in the specified path and [Overwrite Files Without Prompting] is checked (turned on), the files in the folder are overwritten without asking for your confirmation.

The following shows the directories and file configuration after the program files have been copied:

(root)\

readme.txt	Supplementary explanation, etc. (in English)
readmeja.txt	Supplementary explanation, etc. (in Japanese)
voxtool\ VOX33 tool directory
readme.txt	VOX33 tool supplementary explanation, etc. (in English)
readmeja.txt	VOX33 tool supplementary explanation, etc. (in Japanese)
param.txt	Compression parameter supplementary explanation, etc. (in English)
paramja.txt	Compression parameter supplementary explanation, etc. (in Japanese)
bin\VOX33 tools
addsInt.exe	Silent data addition program
adpcmprs.exe	ADPCM compression program
adpdec.exe	ADPCM decoding program
adpparam.exe	ADPCM compression/expansion evaluation program
ampchk.exe	Amplitude checking program
bdmp.exe	Binary file dumping program
bin2s.exe	Binary-to-assembly source conversion program
cnv48_8.exe	WAV-to-PCM conversion program
dct_cnv.exe	Sampling rate conversion program
pcm2s.exe	PCM-to-assembly source conversion program
pcm2wav.exe	PCM-to-WAV conversion program
pcm_norm.exe	PCM normalization program
ppccmprs.exe	Packed PCM compression program
ppcdec.exe	Packed PCM decoding program
vox2cmp.exe	VOX2 compression program
vox2dec.exe	VOX2 decoding program
vox2parm.exe	VOX2 compression/expansion evaluation program
voxcmprs.exe	VOX compression program
voxdec.exe	VOX decoding program
voxflt.exe	High-pass filter program
voxlvl.exe	Level adjustment program
voxpath.exe	VOX compression/expansion evaluation program
voxcmprs.exe	VOX compression program
vscparam.exe	VSC conversion evaluation program
vsxcmprs.exe	VSX compression program
vsxdec.exe	VSX decoding program
vsxpath.exe	VSX compression/expansion evaluation program
wav2pcm.exe	WAV-to-PCM conversion program
param\ Parameter directory
	Various sample files of VOX parameters

sample\ Sample directory Voice, batch file, and make file samples
src\ Source directory Published tool source files
voxlib\ VOX33 library-related directory
readme.txt	VOX33 library supplementary explanation, etc. (in English)
readmeja.txt	VOX33 library supplementary explanation, etc. (in Japanese)
lib\ VOX33 library for E0C33A104 directory
vox.lib	VOX33 library for E0C33A104
sl104.lib	Voice input/output library for E0C33A104 (D/A version)
vox33asm.o, vox2asm.o, mesa.o, cpclrdat.o, fadpcm16.o, fadpcm24.o, fadpcm32.o, fadpcm40.o	Objects retrieved from vox.lib to accelerate operation
lib208\ VOX33 library for E0C33208 directory
vox208.lib	VOX33 library for E0C33208
sl208.lib	Voice input/output library for E0C33208 (PWM version)
vox33asm.o, vox2asm.o, mesa.o, cpclrdat.o, fadpcm16.o, fadpcm24.o, fadpcm32.o, fadpcm40.o	Objects retrieved from vox208.lib to accelerate operation
include\ VOX33 library function header file directory
voxcomn.h	Library common header file
adpcm.h	ADPCM header file
vsx.h	VSX header file
vox.h	VOX compression/expansion function header file
vsc.h	Talking speed/tone pitch conversion header file
packpcm.h	Packed PCM header file
speak.h	Output function header file
listen.h	Input function header file
lksym.h	Linker symbol header file
src\ Library source directory
voxcomn.c	Library common functions
slutil.c	SPEAK and LISTEN utility functions
vsxtop.c	VSX top-level functions
adptop.c	ADPCM top-level functions
voxtop.c	VOX top-level functions
vox2top.c	VOX2 top-level functions
ppctop.c	PCM top-level functions
hardsrc\ Hardware dependent source directory
Listen.s	Listen.o source (E0C33A104)
LisAD.s	LisAD.o source (E0C33A104)
Speak.s	Speak.o source (E0C33A104)
SpkDA.s	SpkDA.o source (E0C33A104)
Lis208.s	Lis208.o source (E0C33208)
Lis208AD.s	Lis208AD.o source (E0C33208)
Spk208.s	Spk208.o source (E0C33208)
Spk208PW.s	Spk208PW.o source (E0C33208)
slintr.def	
	(Use these sources as references when you want to modify timer A/D or D/A channels and ports.)
sample\ DMT33004 sample program directory
smpl208\ DMT33005 sample program directory
	(For details on the configuration of sample programs and how to use them, refer to "readme.txt" or "readmeja.txt" in "voxlib".)

Although the directory structure in your computer can be changed as desired, the explanations on the following pages assume that each file has been copied from CD-ROM in the above directory structure.

3 Software Development Procedure

This section describes the procedure for developing software to process voice data on the E0C33 Family chip. The basic development procedure is shown below.

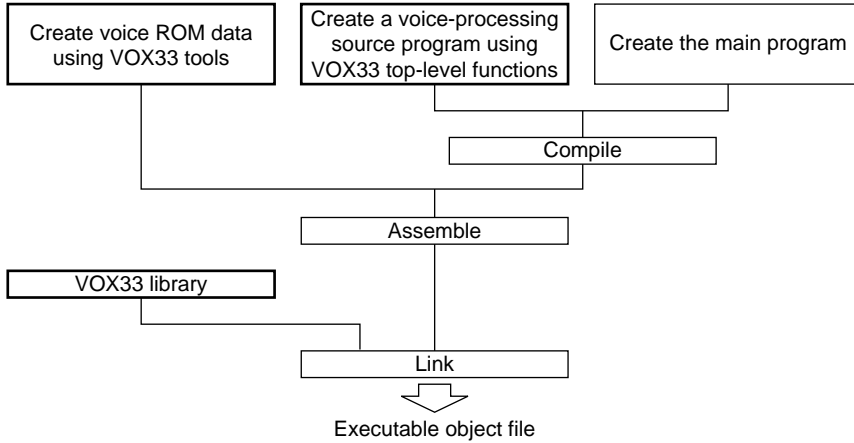
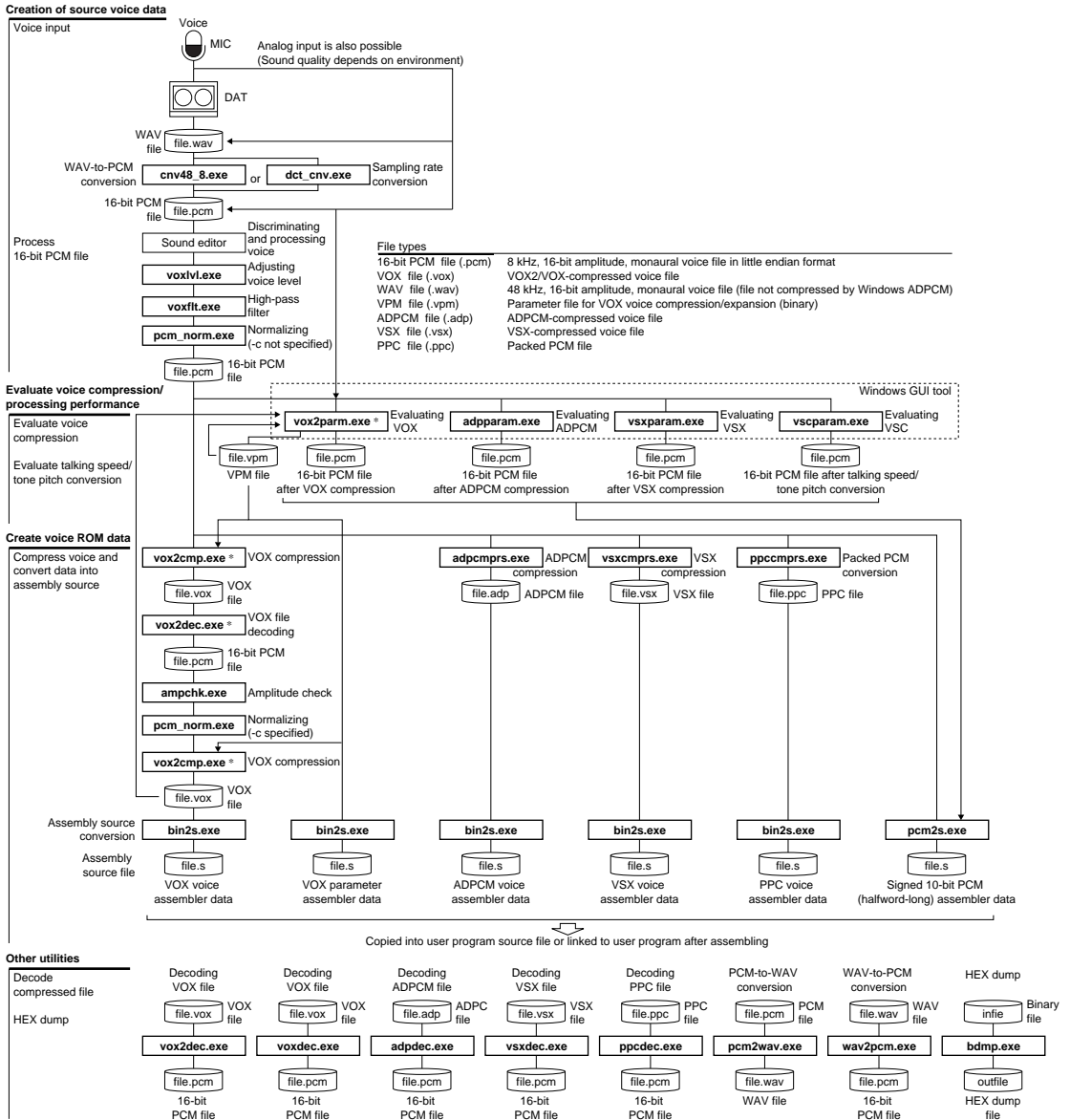


Figure 3.1 E0C33 Voice-Processing Software Development Procedure

- 1) To write playback-only voice data to ROM, use VOX33 tools to create a voice ROM data assembly source file. Even when creation of this data is unnecessary, VOX parameter data is required for sounds to be recorded and reproduced on the E0C33 chip using VOX compression/expansion functions.
- 2) Create a user program. For voice processing, use the top-level functions provided in the VOX33 library. The voice ROM data source file created in step 1 can be included in the user program source file.
- 3) Compile and assemble the source program.
- 4) Link the object files generated in step 3 with the VOX33 library. This generates the object program file in executable form.

3.1 Creating Voice ROM Data using VOX33 Tools

If you want to prepare the voice data to be reproduced on the E0C33 chip in advance, create such data using VOX33 tools. Also, even when not writing voice data to ROM, if you want to use VOX or VOX2 compression/expansion functions on the E0C33 chip, you need to create VOX parameters using VOX33 tools and include them in the program. Figure 3.1.1 shows the procedure for creating voice ROM data and the configuration of VOX33 tools.



* Although only VOX2 tool names (vox2xxx) are shown for VOX compression in the above diagram, VOX tools ("voxparam.exe", "voxcmprs.exe", and "voxdec.exe") also are available. Note that the files compressed by VOX2 tools and those compressed by VOX tools are not compatible.

Figure 3.1.1 Flow Chart for Creating Voice ROM Data

Only the methods for using VOX33 tools are outlined here. For details, refer to Section 4, "VOX33 Tool Reference". In the explanation that follows, "se.pcm" in the "voxtool\sample" directory is used as the source voice file. Also, the explanation below assumes that "voxtool\sample" is the current directory, and that PATH is set in the "voxtool\bin\" directory.

```
DOS>CD e0c33\vox33\voxtool\sample
DOS>PATH c:\e0c33\vox33\voxtool\bin
```

3.1.1 Preparing Voice Data

Using a microphone, create the source voice data (a 16-bit PCM file containing 8-kHz-sampling, 16-bit monaural voice). Prepare data with the highest sound quality possible. Digital sampling using a DAT (digital audio tape recorder) is recommended.

Down-sampling a WAV file

When you have created a 48-kHz-sampling WAV file, use VOX33 tool "cnv48_8.exe" or "dct_cnv.exe" to down-sample it to 8 kHz and convert it into a 16-bit PCM file. Execute these tools from the DOS prompt.

Example: "cnv48_8.exe"

```
DOS>cnv48_8 sample.wav sample.pcm
```

In this example, "sample.wav" is down-sampled to generate "sample.pcm".

The sample "se.pcm" is a 16-bit PCM file that has been created in the above way from 48-kHz digital sampling data. The WAV files that can be converted by "cnv48_8.exe" are limited to 16-bit amplitude, monaural voice data sampled at 48 kHz that has not been compressed into Windows ADPCM format. Commercially available sound editors can also be used for this processing, but care must be taken not to degrade sound quality.

3.1.2 Preprocessing 16-bit PCM Data

Next, separate the actually used part from the sampled voice data and preprocess it by level adjustment and filtration. Use a commercially available sound editor for separation processing. Although such a sound editor can also be used for level adjustment and filtration, care must be taken to avoid degradation of sound quality.

Adjusting voice level

Adjust the low-and high-level parts of voice data to the appropriate level. The VOX33 tool "voxlvl.exe" amplifies the low-level parts of voice data by a factor of 1.5 to 2 and attenuates high-level peaks by a factor of 3.

```
DOS>voxlvl 200 6000 10000 20000 0 0 se.pcm sel.pcm
```

In this example, "se.pcm" is adjusted for level to generate "sel.pcm". Parts of voice data with a level (signed 16-bit value) of 200 or less are regarded as silent parts and are not processed. For sound parts of data, the amplitude is increased two-fold when the maximum amplitude level in one block of data is 6,000 or less or 1.5-fold when the maximum amplitude level is 10,000 or less. Furthermore, parts of data with a maximum amplitude level exceeding 20,000 (i.e., peaks) are attenuated by a factor of 3. For details on parameters, refer to Section 4, "VOX33 Tool Reference".

In view of sound quality, try to prepare source voice data that will not require such level adjustment.

The sample "se.pcm" does not require such level adjustment.

High-pass filtering

VOX33 tools have a high-pass filter program "voxflt.exe" that allows you to specify the cut-off frequency (120 Hz by default). By passing voice data through this filter, the clarity of speech can be improved. Normally, Seiko Epson recommends filtering voice data with a 120 Hz cut-off frequency before using it in the next processing step.

```
DOS>voxflt -l 120 se.pcm seH.pcm
```

In this example, "se.pcm" is filtered with cut-off frequency of a 120 Hz (-l 120) to generate "seH.pcm".

Normalizing

If the maximum amplitude of the source voice data exceeds 90% of the maximum value of 16-bit PCM data when the data is compressed, sound quality after compression may be degraded. For this reason, adjust the amplitude of the source voice to below 90% of the maximum value. Use "pcm_norm.exe" for this processing.

```
DOS>pcm_norm seH.pcm seN.pcm
```

In this example, "seH.pcm" is adjusted so that the amplitude is below 90% of the maximum value of 16-bit PCM data and saved to "seN.pcm" after being adjusted.

The voice data must always undergo this processing before it can be compressed by VOX33 tools.

3.1.3 Evaluating Compression and Talking Speed/Tone Pitch Conversion

Before creating voice ROM data, voice compression/expansion and the voice data after talking speed/tone pitch conversion can be evaluated on a PC. Explained here is the method of voice evaluation using VOX2 compression evaluation tool "vox2param.exe", VSX compression evaluation tool "vsxparam.exe", and VSC processing evaluation tool "vscparam.exe". Although other tools such as VOX compression evaluation tool "voxparam.exe" and ADPCM compression evaluation tool "adpparam.exe" also are available, the method for using these tools is almost the same as that for "vox2param.exe" and "vsxparam.exe". For details on each tool, refer to Section 4.3, "Voice Compression and Processing Evaluation Tools".

Evaluating VOX2 compression and creating a VOX parameter file

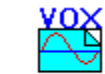
VOX2 and VOX are Seiko Epson's original voice data formats, which offer a sophisticated way to set compression parameters. VOX supports both compression recording and expansion playback on the E0C33 chip. VOX2 is a high-sound-quality version of VOX that allows for only expansion playback on the E0C33 chip.

When using the VOX2 expansion function on the E0C33 chip, evaluate VOX2-compressed voice data and create a VOX parameter file using a 16-bit PCM file that has been normalized in the preprocess.

A Windows GUI tool "vox2param.exe" is available for this processing.

Note that "voxparam.exe" is available for evaluating VOX compression, and the method for using this tool is the same as that for "vox2param.exe". However, because the VOX2- and VOX-compressed data are not compatible with each other, be sure to use the dedicated tool for each compression format. The basic procedure for using "vox2param.exe" is described below.

(1) Starting vox2param.exe



vox2param.exe

Double-click on the "vox2param.exe" icon to start the tool. To quit the tool, click on the [Close] button on the title bar.

When "vox2param.exe" starts, the [VOX2Param] window appears.



[VOX2Param] window

(2) Entering voice data

Loading a 16-bit PCM data file

Click on the [Open] button to call up a file selection dialog box. Use this dialog box to choose the 16-bit PCM file that was normalized in the preprocess.

Entering data from a microphone

To enter voice data from a microphone, set the recording time (seconds) in [Time] and the input level in [Gain] and click on the [Listen] button.

When you have finished entering voice data from a file or microphone, the input waveform is displayed in the full-waveform display area in the upper part of the window.



Full-waveform display area (example for se.pcm)

The input voice can be reproduced by clicking on the [Speak] button.

(3) Loading a VPM file

Although VOX parameters that determine the compression ratio and sound quality can be set directly from the [VOX2Param] window, they also can be loaded from an existing VPM file as a template. Use one of the various sample parameter files that are provided in the "voxtool\param\" directory. Use the [Open VPM] button to load a VPM file. The parameter setup contents in the window are updated with the parameters loaded from the file. For details on the contents of each parameter file, refer to Section 4.4.2, "VOX Parameter Samples".

(4) Adjusting VOX parameters

For the parameters that determine the VOX2 voice compression ratio and sound quality, the following items must be set:

Depth, Width, Height, Weight, Pre, Mid, Post, Level-2, Level-3, NS Filter

The following shows approximate values of standard settings for each parameter:

Compression ratio = 4 to 8kbps: Depth = 2, Width = 4 to 8, Pre = Off, Mid = Off, Post = 75

Compression ratio = 8 to 15kbps: Depth = 3, Width = 4 to 8, Pre = Off, Mid = Off, Post = 75

Compression ratio = 10 to 20kbps: Depth = 4, Width = 4 to 8, Pre = Off, Mid = Off, Post = 75

For other parameters, the default values should normally be used.

Height = 10, Weight = 100, Level-2 = 100, Level-3 = 0

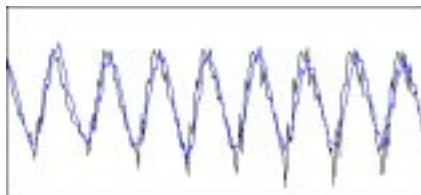
NS Filter is used to reduce noise at high frequencies. It should normally be left on.

When using "voxparam.exe" to set VOX voice parameters, note that its setting ranges differ from those of VOX2. For details on each parameter, refer to Section 4.4, "VOX Parameters".

After setting each parameter in the [VOX2Param] window, click on the [SyntheSpk] button. The input voice data is compressed according to the parameters set, and the compressed voice data is reproduced by expansion. Reproduce the source voice using the [Speak] button and compare it with the compressed voice. The sound quality deteriorates as the compression ratio increases, so adjust the parameters within the allowable range of voice data to attain the best sound quality possible.

The [Avr.] box shows the average voice data rate after compression.

The lower waveform display area is used to display part of the voice waveform along the time axis as an enlarged view. By choosing the [Synthe.] and [Source] check boxes, you can display the source voice waveform in black and the compressed voice waveform in blue. This helps to visually verify the effects of applied parameters. The partial-waveform you want to check can be displayed by scrolling the screen using the upper scroll bar.



Partial-waveform display area (example for se.pcm)

(5) Saving VOX parameters

After determining each VOX parameter, save the parameters you have set to a VPM file. Use the [Save VPM] button to save.

The VPM file created here is always required for VOX2 compression of voice data by "vox2parm.exe", as well as for VOX2 expansion on the E0C33 chip. Therefore, do not forget to save the parameters to a VPM file.

(6) Saving compressed voice data

To use the result of compression as voice ROM data, save it to a PCM file using the [SavPCM] button.

When loading a 16-bit PCM file to evaluate compression, you do not need to save compression results here, because a VOX2 voice file can be generated from the original data using "vox2cmp.exe".

Evaluating VSX compression

VSX, an extended version of the ADPCM format, is Seiko Epson's original voice data format. It allows compression in the timebase direction and compression of silent parts of data, thereby providing a higher compression ratio than ADPCM.

To use the VSX compression/expansion function on the E0C33 chip, evaluate VSX compression by using a 16-bit PCM file that has been normalized in the preprocess or by entering voice data from a microphone.

Windows GUI tool "vsxparam.exe" is available for this processing.

The following describes the basic procedure for using "vsxparam.exe".

(1) Starting vsxparam.exe



vsxparam.exe

Double-click on the "vsxparam.exe" icon to start the tool. To quit the tool, click on the [Close] button on the title bar.

When "vsxparam.exe" starts, the [VSXParam] window appears.



[VSXParam] window

(2) Entering voice data

Loading a 16-bit PCM data file

Click on the [Open] button to call up a file selection dialog box. Use this dialog box to choose the 16-bit PCM file that has been normalized in the preprocess.

Entering data from a microphone

To enter voice data from a microphone, set the recording time (seconds) in [Time] and the input level in [Gain] and click on the [Listen] button.

When you have finished entering voice data from a file or microphone, the input waveform is displayed in the full-waveform display area in the upper part of the window.



Full-waveform display area (example for se.pcm)

The input voice can be reproduced by clicking on the [Speak] button.

(3) Choosing compression ratio

Using the [Compress] combo box, choose the desired compression ratio from the following four:

- 16 kbps
- 24 kbps (default)
- 32 kbps
- 40 kbps

Using the [Time Cmprs] combo box, choose a compression ratio in the timebase direction.

- ×1.0 (same as the source voice; default)
- ×2.0 (same effect as recording at 2 times normal speed)
- ×3.0 (same effect as recording at 3 times normal speed)
- ×4.0 (same effect as recording at 4 times normal speed)

Using the [Speed] combo box, choose a playback speed for voice data.

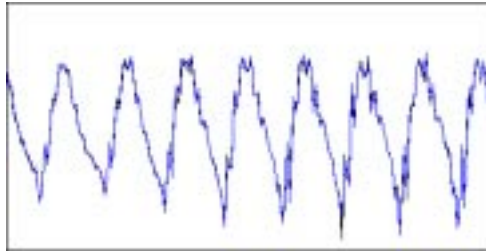
- ×1.0 (same as the source voice; default)*
- ×1.5 (speed converted to 1.5 times that of source voice)*
- ×2.0 (speed converted to 2 times that of source voice)*
- ×3.0 (speed converted to 3 times that of source voice)
- ×4.0 (speed converted to 4 times that of source voice)
- ×6.0 (speed converted to 6 times that of source voice)
- ×8.0 (speed converted to 8 times that of source voice)
- ×16.0 (speed converted to 16 times that of source voice)
- ×1/1.5 (speed converted to 1/1.5 times that of source voice)*
- ×1/2.0 (speed converted to 1/2 times that of source voice)

Note: Conversion on the E0C33 chip is subject to limitations on the parameters that can be selected. (Seiko Epson recommends using only the parameters marked by *.)

Since silent parts of data are compressed further, use the [Silent thresh] edit box to set the threshold level at which you want the data to be treated as silent. The greater the threshold, the higher the compression ratio, but the lower the sound quality. Normally, set the threshold in the range of 0 to 50.

After selecting each parameter, click on the [SyntheSpk] button. The input voice data is compressed according to the selected parameters, and the compressed voice data is reproduced by expansion.

The lower waveform display area is used to display part of the voice waveform along the time axis as an enlarged view. By choosing the [Synthe.] and [Source] check boxes, you can display the source voice waveform in black and the compressed voice waveform in blue. The partial-waveform you want to check can be displayed by scrolling the screen using the upper scroll bar.



Partial-waveform display area (example for se.pcm)

(4) Saving compressed voice data

To use the result of compression as voice ROM data, save it to a PCM file using the [SavPCM] button.

When loading a 16-bit PCM file to evaluate compression, you do not need to save compression results here, because a VSX voice file can be generated from the source data using "vsxcmprs.exe".

Evaluating talking speed/tone pitch conversion (VSC conversion)

The VOX33 library offers a function to convert talking speed or tone pitch on the E0C33 chip based on Seiko Epson's exclusive VSC voice processing technology. This VSC conversion function can be evaluated using Windows GUI tool "vscparam.exe". Also, the VSC-converted voice data can be saved to a file for use as voice ROM data. The following describes the basic procedure for using "vscparam.exe".

(1) Starting vscparam.exe



vscparam.exe

Double-click on the "vscparam.exe" icon to start the tool. To quit the tool, click on the [Close] button on the title bar.

When "vscparam.exe" starts, the [VSCParam] window appears.



[VSCParam] window

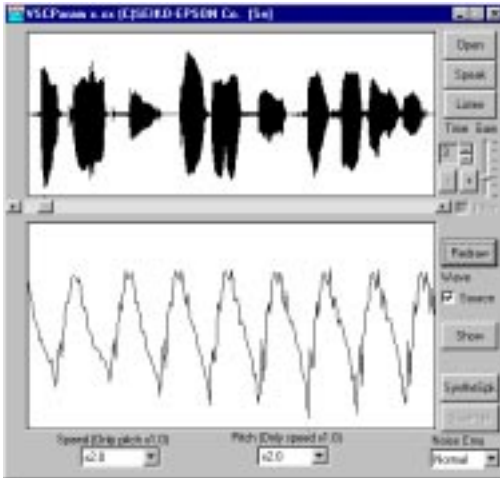
(2) Entering voice data

Loading a 16-bit PCM data file

Click on the [Open] button to call up a file selection dialog box. Use this dialog box to choose the 16-bit PCM file to be VSC-converted.

Entering data from a microphone

To enter voice data from a microphone, set the recording time (seconds) in [Time] and the input level in [Gain] and click on the [Listen] button.



Waveform display (example for se.pcm)

When you have finished entering voice data from a file or microphone, the input waveform is displayed in the full-waveform display area in the upper part of the window. If the [Source] check box is selected (turned on), an enlarged waveform is displayed along the time axis in the partial-waveform display area below the full-waveform display area.

The input voice can be reproduced by clicking on the [Speak] button.

(3) Setting speed and pitch

Using the [Speed] or [Pitch] combo boxes, choose speed or pitch.

Use the [Speed] combo box to choose a playback speed as a multiplying factor relative to the speed of the source voice. The speed can be selected from the following five options:

- ×1.0 (same as the source voice; default)
- ×1.5 (speed converted to 1.5 times that of the source voice)
- ×2.0 (speed converted to 2 times that of the source voice)
- ×1/1.5 (speed converted to 1/1.5 times that of the source voice)
- ×1/2.0 (speed converted to 1/2 times that of the source voice)

Use the [Pitch] combo box to choose a pitch (tone of voice) during playback as a multiplying factor relative to that of the source voice. The pitch can be selected from the following five options:

- ×1.0 (same as the source voice; default)
- ×1.5 (pitch converted to 1.5 times that of the source voice)
- ×2.0 (pitch converted to 2 times that of the source voice)
- ×1/1.5 (pitch converted to 1/1.5 times that of the source voice)
- ×1/2.0 (pitch converted to 1/2 times that of the source voice)

Note: Real-time conversion on the E0C33 chip is subject to limitations on the parameters that can be selected. This limitation does not apply when the conversion result of "vscparam.exe" is saved to a PCM file for use as voice ROM data.

(4) Reproducing VSC-converted voice data

Click on the [SyntheSpk] button. The speed- or pitch-converted voice data will be reproduced.

(5) Saving VSC-converted data

To use the result of conversion as voice ROM data, you can save it to a PCM file using the [SavPCM] button.

3.1.4 Converting Voice Data into an Assembly Source File

To enable the generated voice data to be included in or linked to the user program, generate an assembly source file for the EOC33 Assembler.

To determine whether to compress voice data and which compression format to use, examine the evaluation results regarding the compression ratio and sound quality obtained by using the evaluation tools described above, as well as the system's memory capacity and other factors.

When using VOX2/VOX-compressed voice data

To write VOX2/VOX-compressed voice data to ROM, follow the procedure described below to generate the assembly source file. VOX2 tools are used in the explanation below. When using VOX voice data, substitute "vox..." tools for the "vox2..." tools.

1. Using "vox2cmp.exe", compress the source voice data (normalized 16-bit PCM file) in VOX2 format to create a VOX2 voice file. The VOX parameter file created by "vox2parm.exe" is required for this operation.

Example: DOS>vox2cmp seN.pcm se0.vox ..\param\d3w7N.vpm

In this example, the voice data "seN.pcm" that has been normalized by "pcm_norm.exe" is VOX2-compressed according to settings in the VOX parameter file "d3w7N.vpm" (located in the "voxtool\param\" directory), thereby creating "se0.vox".

2. Using "vox2dec.exe", decode the VOX2 voice file into a PCM file.

Example: DOS>vox2dec se0.vox se1.pcm

3. Using "ampchk.exe", check the amplitude of the VOX2-converted voice data.

Example: DOS>ampchk seN.pcm se1.pcm

The amplitude ratio between the source voice and the converted voice is obtained, and the result is written to the file "amp.rto".

4. Load "amp.rto" into "pcm_norm.exe" and readjust the amplitude. To load "amp.rto", specify the -c option in "pcm_norm.exe".

Example: DOS>pcm_norm -c se1.pcm se2.pcm

The amplitude is adjusted based on the data in "amp.rto".

5. Using "vox2cmp.exe" again, create a VOX2 voice file.

Example: DOS>vox2cmp se2.pcm se.vox ..\param\d3w7N.vpm

Thus, the final VOX2 voice data "se.vox" is generated.

Steps 2 to 5 must be executed when the amplitude of the VOX-compressed voice data exceeds the designated level. They are unnecessary for operations other than VOX2/VOX compression.

6. Using "bin2s.exe", convert the VOX2 voice file (binary file) into an assembly source file.

Example: DOS>bin2s se.vox > se.voxs (The redirect function of DOS is used)

In this example, the VOX2 file "se.vox" is converted into the assembly source file "se.voxs". This file "se.voxs" is generated using the input file name "se" as a global symbol as shown below. (The symbol name can be changed using the "-l *symbol*" option of "bin2s.exe".)

Contents of "se.voxs"

```
.global    se
.align    2
se:
.byte     0x83 0x95 0x03 0xfe 0x28 0x43 0x4b 0x4b
.byte     0x4b 0x64 0x64 0x00 0x07 0x4a 0x8f 0x86
;
; total 3705 bytes data
```

- Using "bin2s.exe", convert the VPM file (VOX parameter file) into an assembly source file.

Example: DOS>bin2s ..\param\d3w7N.vpm > d3w7N.s

In this example, the VOX parameter file "d3w7N.vpm" that was used to create "se.vox" is converted into the assembly source file "d3w7N.s".

Contents of "d3w7N.s"

```
.global    d3w7N
.align    2
d3w7N:
.byte     0x83 0x95 0x03 0xfe 0x28 0x43 0x4b 0x4b
.byte     0x4b 0x64 0x64 0x00
; total 12 bytes data
```

If you want to use other parameter settings on the E0C33 chip in addition to VOX2 expansion and VOX compression/expansion, convert the VPM files for those parameters into assembly source files too.

When using VSX-compressed voice data

To write VSX-compressed voice data to ROM, follow the procedure described below to generate the assembly source file.

- Using "vsxcmprs.exe", compress the source voice data (normalized 16-bit PCM file) in VSX format to create a VSX voice file.

Example: DOS>vsxcmprs -c24 -t2 -s 20 seN.pcm se.vsx

In this example, the voice data "seN.pcm" that has been normalized by "pcm_norm.exe" is compressed by a factor of 2 in the timebase direction with a compression ratio of 24 kbps and a silent packet level of 50, thereby creating "se.vsx". For the compression ratio option to be specified here, use the parameter determined during evaluation by "vsxparam.exe".

- Using "bin2s.exe", convert the VSX voice file (binary file) into an assembly source file.

Example: DOS>bin2s -l sevsx se.vsx > se.vxsx (The redirect function of DOS is used)

In this example, the VSX file "se.vsx" is converted into the assembly source file "se.vxsx". This file "se.vxsx" is generated using "sevsx" as a global symbol as shown below. (If the "-l *symbol*" option is omitted, the symbol name becomes the same as the input file name "se".)

Contents of "se.vxsx"

```
.global    sevsx
.align    2
sevsx:
.byte     0x53 0x22 0x01 0x01 0x1b 0xd6 0xdd 0x61
.byte     0x92 0x50 0x05 0x06 0x49 0x27 0x26 0x00
.byte     0x3d 0xb2 0x53 0x48 0x00 0xda 0x69 0x14
:
; total 4733 bytes data
```

When using ADPCM-compressed voice data

To write ADPCM-compressed voice data to ROM, follow the procedure described below to generate the assembly source file.

1. Using "adpcmprs.exe", compress the source voice data (normalized 16-bit PCM file) in ADPCM format to create an ADPCM voice file.

Example: DOS>adpcmprs -24 seN.pcm se.adp

In this example, the voice data "seN.pcm" that has been normalized by "pcm_norm.exe" is ADPCM-compressed with a 24 kbps compression ratio, thereby creating "se.adp". For the compression ratio option to be specified here, use the compression ratio determined during evaluation by "adpparam.exe".

2. Using "bin2s.exe", convert the ADPCM voice file (binary file) into an assembly source file.

Example: DOS>bin2s -l seadp se.adp > se.adps (The redirect function of DOS is used)

In this example, the ADPCM file "se.adp" is converted into the assembly source file "se.adps". This file "se.adps" is generated using "seadp" as a global symbol as shown below. (If the "-l *symbol*" option is omitted, the symbol name becomes the same as the input file name "se".)

Contents of "se.adps"

```
.global    seadp
.align    2
seadp:
.byte     0x41 0x70 0x26 0x00 0x00 0x00 0x00 0x00
.byte     0x00 0x02 0xae 0xd7 0x19 0x24 0xe2 0x71
.byte     0x34 0x93 0x80 0x12 0x49 0x24 0x91 0x5b
:
; total 9850 bytes data
```

When writing to ROM in packed PCM format

To write voice data to ROM after converting it into packed PCM format, follow the procedure described below to generate the assembly source file.

1. Using "ppcmprs.exe", convert the source voice data (normalized 16-bit PCM file) into packed PCM format to create a PPC file.

Example: DOS>ppcmprs seN.pcm se.ppc

In this example, the voice data "seN.pcm" that has been normalized by "pcm_norm.exe" is converted to create "se.ppc".

2. Using "bin2s.exe", convert the PPC file (binary file) into an assembly source file.

Example: DOS>bin2s -l seppc se.ppc > se.ppcs (The redirect function of DOS is used)

In this example, the PPC file "se.ppc" is converted into the assembly source file "se.ppcs". This file "se.ppcs" is generated using "seppc" as a global symbol as shown below. (If the "-l *symbol*" option is omitted, the symbol name becomes the same as the input file name "se".)

Contents of "se.ppcs"

```
.global    seppc
.align    2
seppc:
.byte     0x50 0x80 0x66 0x00 0x00 0x00 0x00 0x00
.byte     0x00 0x01 0x00 0x00 0x00 0xff 0x00 0xff
.byte     0x00 0x00 0xff 0x00 0xff 0xff 0xff 0xff
:
; total 32436 bytes data
```

When using uncompressed PCM voice data

To write the source 16-bit PCM data directly to ROM without compressing it, use "pcm2s.exe" to create the assembly source file.

Example: DOS>pcm2s -l sepcm seN.pcm > se.pcms (The redirect function of DOS is used)

In this example, the 16-bit PCM file "seN.pcm" is converted into the assembly source file "se.pcms". This file "se.pcms" is generated using "sepcm" as a global symbol as shown below. (If the "-l symbol" option is omitted, the symbol name becomes the same as the input file name "seN".)

Contents of "se.adps"

```
.global      sepcm
.align      2
sepcm:
.word 0x667a

        .half 0x0001 0x0001 0x0001 0x0001 0x0000 0x0001 0x0000 0x0001
        .half 0x0001 0x0000 0x0001 0x0000 0x0000 0x0000 0x0000 0x0000
        .half 0x0001 0x0000 0x0000 0x0001 0x0000 0x0000 0x0001 0x0000
        :
; total 26234 short data + 4byte header
```

3.1.5 Precautions Concerning Voice ROM Data Creation

- The voice compression/processing evaluation tools ("vox2parm.exe", "voxparam.exe", "vsxparam.exe", "adpparam.exe", and "vscparam.exe") use algorithms that are similar to, but not identical to, the libraries implemented on the E0C33 chip. Use these tools for the preliminary evaluation of compression parameters, etc. Also, because sound quality depends on analog components such as the speaker, microphone, and op-amp, the sound quality as evaluated on a PC may differ from that actually obtained in the application system. The final sound quality, operation, etc. must be evaluated using the actual application system that contains the E0C33 chip.
- Before any voice compression/processing evaluation tool can be used, a sound card (SoundBlaster 16 or compatible) that supports 8 kHz-sampling and 16-bit monaural voice input/output is required. The evaluation tools may not work with a sound card that only supports 8-bit data.
- VOX2 voice data and VOX voice data are not compatible with each other. For data processing, use the dedicated tools provided for each type of voice data.
- When VSX data or VSC-converted voice data is reproduced on the actual application system using the VOX33 library, the number of data samples output may differ by several packets with respect to the source voice data. If the source voice does not have enough silent data appended, this error may cause gaps in the reproduced voice. In such a case, add silent data to the last part of the source voice using "addsInt.exe".

3.2 Creating a User Program and Linking the VOX33 Library

A range of operations from voice compression and recording to voice expansion, talking speed/tone pitch conversion, and playback on the E0C33 chip can be implemented by calling up VOX33 library functions. In addition to low-level library objects, this middleware package contains the source file of the functionally classified top-level functions created in C language. By installing these functions into the user program, a voice-processing routine can be created easily.

For details on the top-level functions and VOX33 library, refer to Section 5, "VOX33 Library Reference".

In addition, sample programs are provided in the "voxlib\sample\" directory for your reference.

The voice ROM data and VOX parameter source files you have created can be incorporated into the user program or linked to the user program along with the VOX33 library after assembling.

When creating and linking programs, note the following:

- (1) The VOX33 library functions use the CPU's R8 register. Therefore, when linking VOX33 library functions, including the top-level functions, you cannot use the -gp option (optimization using global pointer/R8) of the instruction extender ext33.
- (2) Make sure all of the BSS sections used by the VOX33 library are mapped into the internal RAM. Also, be sure to use the internal RAM for the stack.
- (3) When mapping VOX33 library program code into an external memory area, make sure this area is accessed in 2 wait cycles or less, if possible. Also, be sure to use a 16-bit-wide memory area for this external area.
- (4) Several objects in the VOX33 library need to be mapped into the internal RAM in order to increase the operation speed. For details, refer to Section 5.5, "Techniques for Speeding Up Operation".

Procedures for executing a sample program using the DMT33004 and DMT33AMP boards are listed in the Appendix for your reference.

4 VOX33 Tool Reference

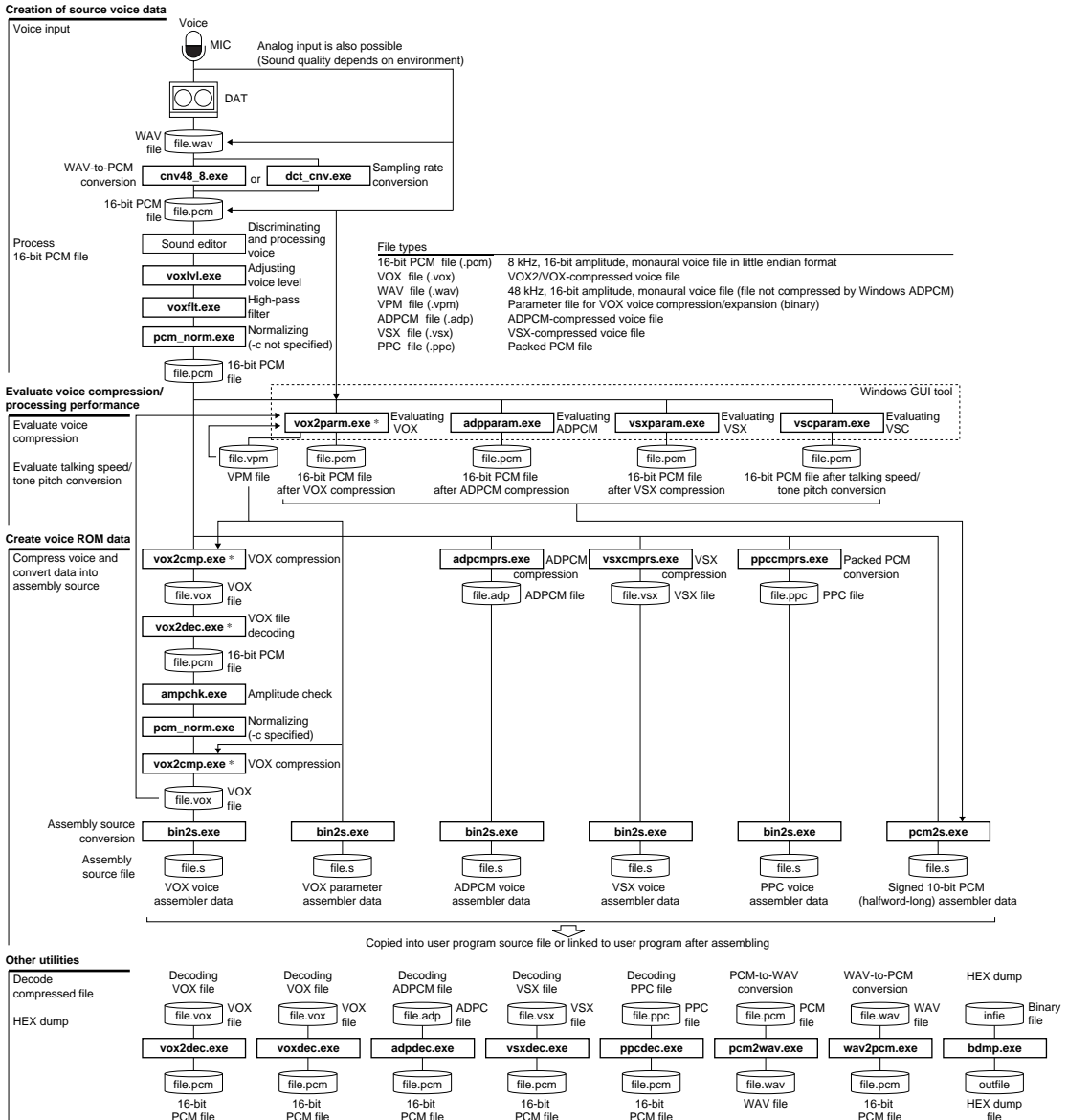
This section describes the functions of each VOX33 tool and how to use them.

4.1 Outline of VOX33 Tools

VOX33 tools are software tools that are run on a personal computer to create the voice ROM data to be stored on the E0C33 Family chip, as well as to evaluate the voice compression and voice processing performance. All of these tools can be run under Windows 95, Windows NT 4.0, or higher versions. (For details on the operating environment, refer to Section 2.1, "Operating Environment".)

The VOX33 library and related files are all provided in the "voxtool" folder (directory).

The configuration of VOX33 tools and the procedure for creating voice ROM data are shown in Figure 4.1.1 below.



* Although only VOX2 tool names (vox2xxx) are shown for VOX compression in the above diagram, VOX tools (voxparm.exe, voxcmp.exe, and voxdec.exe) are also available. Note that the files compressed by VOX2 tools and those compressed by VOX tools are not compatible.

Figure 4.1.1 Flow Chart for Creating Voice ROM Data

Voice ROM data generation tools

The voice ROM data generation tools consist of a series of programs that convert, process, and compress a voice file (wav, pcm) to create an assembly source file for the E0C33. Each program is a 32-bit application that can be executed from the DOS prompt, and can be used from a batch file or a make file.

Table 4.1.1 below lists the voice ROM data generation tools.

Table 4.1.1 Voice ROM Data Generation Tools

Tool	Function
cnv48_8.exe	A down-sampler used to convert a WAV file (48 kHz) into a 16-bit PCM file (8 kHz). Commercially available sound editors can be used, but may degrade sound quality.
dct_cnv.exe	A down-sampler used to convert a WAV or PCM file into any sampling rate. Commercially available sound editors can be used, but may degrade sound quality.
voxlvl.exe *	Adjusts the level of 16-bit PCM data. It amplifies the low-level parts of voice data by a factor of 1.5 to 2 and attenuates high-level peaks by a factor of 3.
voxfilt.exe	Increases the clarity of sound by passing 16-bit PCM data through a high-pass filter.
addsint.exe	Appends a specified number of silent data to the 16-bit PCM data file.
pcm_norm.exe	Normalizes 16-bit PCM data to a 90% (default) amplitude to make it suitable for input to the voice compression tool. Also, if amplitude readjustment is required based on amplitude inspection results obtained by "ampchk.exe" after VOX compression, it makes the necessary adjustment.
ampchk.exe	Calculates the ratio between two PCM files (before and after VOX compression) and writes it to a file. This result can be input to "pcm_norm.exe" for amplitude readjustment in the PCM file.
pcm2wav.exe	Converts a PCM file into a WAV file.
wav2pcm.exe	Converts a WAV file into a PCM file.
voxcmprs.exe	Compresses 16-bit PCM data into VOX format based on VPM file (.vpm).
vox2cmp.exe	Compresses 16-bit PCM data into VOX2 format based on VPM file (.vpm).
adpcmprs.exe	Compresses 16-bit PCM data into ADPCM format based on the compression ratio specified by an option.
vsxcmprs.exe	Compresses 16-bit PCM data into VSX format based on the compression ratio specified by an option.
ppccmprs.exe	Compresses 16-bit PCM data into packed PCM format.
bin2s.exe *	Converts a binary data file (VOX/VOX2 file, VSX file, VPM file, ADPCM file, or PPC file) into an assembly source file.
pcm2s.exe	Converts a 16-bit PCM file into a 10-bit amplitude assembly source file.
bdmp.exe	A utility used to dump binary data.
voxdec.exe *	Decodes the voice data that has been compressed by "voxcmprs.exe" to save it as PCM data.
vox2dec.exe *	Decodes the voice data that has been compressed by "vox2cmp.exe" to save it as PCM data.
adpdec.exe *	Decodes the voice data that has been compressed by "adpcmprs.exe" to save it as PCM data.
vsxdec.exe *	Decodes the voice data that has been compressed by "vsxcmprs.exe" to save it as PCM data.
ppcdec.exe *	Decodes the voice data that has been compressed by "ppccmprs.exe" to save it as PCM data.

* The source codes of these tools are included in the "voxtool\src" directory. Use these source codes as necessary when developing applications with VOX33.

Voice compression/processing evaluation tools

Voice compression/processing evaluation tools are programs used to evaluate the sound quality of a compressed and processed 16-bit PCM voice file. Depending on the operating environment, they also support voice input from microphones. These tools allow you to examine the voice data compression ratio or talking speed/tone pitch conversion parameters. Also, the voice data can be saved to a 16-bit PCM file after being evaluated and examined, for use in the creation of voice ROM data. All these programs are 32-bit Windows GUI applications, and can display voice waveforms to help you evaluate the sound quality.

Table 4.1.2 lists the voice compression/processing evaluation tools.

Table 4.1.2 Voice Compression/Processing Evaluation Tools

Tool	Function
voxparam.exe	After adjusting VOX parameters, it evaluates the quality of VOX-compressed sound and generates a VPM file.
vox2param.exe	After adjusting VOX parameters, it evaluates the quality of VOX2-compressed sound and generates a VPM file.
vsxparam.exe	After adjusting VSX parameters, it evaluates the quality of VSX-compressed sound.
adpparam.exe	After adjusting ADPCM parameters, it evaluates the quality of ADPCM-compressed sound.
vscparam.exe	After adjusting tone pitch and talking speed, it evaluates the quality of VSC-compressed sound.

Note: These evaluation tools use algorithms that are similar to, but not identical to, the libraries implemented on the E0C33 chip. Use these tools for the preliminary evaluation of compression parameters, etc.

Also, because sound quality depends on analog components such as the speaker, microphone, and op-amp, the sound quality as evaluated on a PC may differ from that actually obtained in the application system. The final sound quality, operation, etc. must be evaluated using the actual application system that contains the E0C33 chip.

4.2 Voice ROM Data Generation Tools

This section describes the functions of each voice ROM data generation tool and how to use them.

Start each tool from the DOS prompt. When a tool is started without specifying command line parameters, Usage is displayed. In the explanation of command lines below, the items enclosed in brackets [] can be omitted. The parameters in *italics* represent the appropriate values or file names to be specified.

Note: The file names that can be specified for each tool are subject to the limitations described below.

- File name: Maximum of 32 characters
- Usable characters: a to z, A to Z, 0 to 9, _, .

4.2.1 *cnv48_8.exe*

Function: Converts a WAV file into a 16-bit PCM file. This tool is used exclusively for 48 kHz to 8 kHz conversion.

Usage: DOS>*cnv48_8 infile.wav outfile.pcm*↵

Arguments: *infile.wav* Input file name (WAV file)
outfile.pcm Output file name (16-bit PCM file)

Example: DOS>*cnv48_8 sample1.wav sample1.pcm*

Note: This tool allows only WAV files in the following format to be input.

- 16-bit amplitude
- Monaural
- Sampling rate = 48 kHz
- Not compressed in Windows ADPCM format

Reference: The tool "*cnv48_8.exe*" is used only for 48 kHz to 8 kHz conversion. For conversion to another sampling rate, use the tool "*dct_cnv.exe*" described in Section 4.2.2.
 If neither of these tools is appropriate, use a commercially available sound editor for conversion.

4.2.2 *dct_cnv.exe*

Function: Converts the input voice file to a file with any desired sampling rate.

Usage: DOS>*dct_cnv DctFrom DctTo infile.(wav/pcm) outfile.pcm*↵

Arguments: *DctFrom* Number of input data to be converted
 DctTo Number of corresponding output data
 infile.wav Input file name (WAV file)
 infile.pcm Input file name (PCM file)
 outfile.pcm Output file name (16-bit PCM file)

Example: For "DctFrom" and "DctTo", Seiko Epson recommends specifying a value that is an integral multiple of the source sampling rate. For example, to down-sample a 48 kHz WAV file to 8 kHz, specify the arguments as shown below.

```
DOS>dct_cnv 48 8 sample1.wav sample1.pcm       (x1)
DOS>dct_cnv 96 16 sample1.wav sample1.pcm      (x2)
DOS>dct_cnv 144 24 sample1.wav sample1.pcm     (x3)
DOS>dct_cnv 140 40 sample1.wav sample1.pcm    (x5)
DOS>dct_cnv 480 80 sample1.wav sample1.pcm   (x10)
```

The greater the values specified for DctFrom and DctTo, the better the sound quality, but the lower the processing speed. When small values are specified for DctFrom and DctTo, the processing speed increases but the sound quality deteriorates. To avoid deterioration in sound quality, Seiko Epson recommends using a value of ×5 or larger for this conversion.

Reference: To convert a 48 kHz WAV file into an 8 kHz PCM file, use the tool "cnv48_8.exe" described above in Section 4.2.1. If tools "cnv48_8.exe" and "dct_cnv.exe" are both inappropriate, use a commercially available sound editor for conversion.

4.2.3 voxlv.exe

Function: Accepts as its input a specified 16-bit PCM file and automatically adjusts it for level. This tool amplifies the low-level parts of sound data by a factor of 1.5 to 2 and attenuates high-level peaks by a factor of 3.

Level adjustment is performed for each occurrence of consecutive blocks of sound according to the maximum amplitude of each block and the specified argument values.

Usage: DOS>voxlv1 *N B0 B1 B2 P D infile.pcm outfile.pcm*␣

Arguments: To specify *N*, *B0*, *B1*, and *B2*, use a signed 16-bit value (in the range of 1 to 32,767).

N If |(previous data) – (current data)| is below this specified value, the input data is treated as a silent part and not adjusted for level.

B0 Blocks whose maximum amplitude level parts of sound is below this specified value are adjusted to twice that amplitude.

B1 Blocks whose maximum amplitude level parts of sound is below this specified value are adjusted to 1.5 times that amplitude.

B2 Blocks whose maximum amplitude level parts of sound is above this specified value have their peaks that exceed this specified value (i.e., *B2* to maximum amplitude) attenuated to 1/3 of that amplitude.

P *P* = 0: Blocks with large power (The value *N*) is not amplified even if the amplitude is small (below the values specified by *B0* and *B1*). (This corresponds to cases where the maximum power is larger than the maximum amplitude.)
P = 1: When the maximum amplitude is between *B0* and *B1*, the amplitude is adjusted by a factor of 1/1.5.

D *D* = 0: Normal mode
D = 1: Debug mode. The data size is displayed on the screen and the following debugging files are output. The debugging files can be displayed as 16-bit PCM data using a sound editor, etc.

lpower.pcm	Power of each data item
larea.pcm	Maximum amplitude of each 100-item block of data
lmax.pcm	Maximum amplitude of each voice block
lpmx.pcm	Maximum power of each voice block

infile.pcm Input file name (16-bit PCM file)

outfile.pcm Output file name (16-bit PCM file)

Example: DOS>voxlv1 200 6000 10000 20000 0 0 se.pcm sel.pcm

Reference: The source code of this tool "voxlv.c" is provided in the "voxtool\src\voxlv\" directory.

4.2.4 *voxflt.exe*

Function: Filters a 16-bit PCM file using a high-pass filter. Such filtration produces the following effects:

- The sound pressure level is attenuated by 40 dB at half the specified cut-off frequency. Generally, when the sound pressure level decreases by 6 dB, the sound volume is halved.
- The sound pressure drops slowly starting from peaks slightly above the cut-off frequency and begins to drop rapidly near the cut-off frequency. (Attenuated by about 3 dB at the cut-off frequency)

Normally, Seiko Epson recommends specifying cut-off at about 120 Hz (default). However, because the sound quality of some data is degraded by filtering, the sound quality must be checked on the user's system.

Usage: `DOS>voxflt [-l CutOff] infile.pcm outfile.pcm↵`

Arguments: *infile.pcm* Input file name (16-bit PCM file)

outfile.pcm Output file name (16-bit PCM file)

`-l CutOff` Cut-off frequency (optional)

The effective values (Hz) for *CutOff* are shown below.

60, 120, 180, 240, 250, 300, 360, 420, 480, 500, 540, 600, 720, 1000, 1440, 2000

When the option is omitted, the input data is filtered at the default cut-off frequency (120 Hz).

Example: `DOS>voxflt -l 180 samp1.pcm samp2.pcm ... Filtered at 180 Hz cut-off frequency`
`DOS>voxflt samp1.pcm samp2.pcm ... Filtered at 120 Hz cut-off frequency`

4.2.5 pcm_norm.exe

Function: Converts the voice data amplitude of the input 16-bit PCM file to a specified amplitude. Also, this tool reads the file "amp.rto" (output by "ampchk.exe") from the current directory for use in amplitude adjustment. The values with 16 signed bits range from -32,768 (SHORT_MIN) to +32,767 (SHORT_MAX). In this program, use a percentage of SHORT_MAX to specify the target amplitude to which you want the maximum amplitude of the input voice data to be reduced as the amplitude of the input voice data is converted.

Usage: DOS>pcm_norm [-r XXX] [-c] input.pcm output.pcm

Arguments:

<i>input.pcm</i>	Input file name (16-bit PCM file)
<i>output.pcm</i>	Output file name (16-bit PCM file)
-r XXX	Coefficient of normalization (optional) Specify the amplitude of the 16-bit PCM voice data as a percentage of the maximum amplitude. For XXX, enter a positive value between 0.0 and 100.0. When this option is omitted, the maximum amplitude of the output voice is set to 90%. Always insert a space between -r and XXX.
-c	Read the file "amp.rto" (optional) The file "amp.rto" is read from the current directory for use in amplitude adjustment. If the value written in "amp.rto" exceeds 1.00, it is automatically corrected to 1.00. The file "amp.rto" is generated as a result of inspection of the VOX-compressed data by "ampchk.exe". This option is used when the amplitude of VOX-compressed voice data exceeds that of the source PCM data in order to readjust the amplitude before reexecuting VOX compression. This option need not be specified for operations other than VOX2 or VOX compression.

Example:

```
DOS>pcm_norm -r 80 input.pcm output.pcm    ... Converted to 80%
DOS>pcm_norm input.pcm output.pcm        ... Converted to 90% (default)
DOS>pcm_norm -c -r 80 input.pcm output.pcm ... Converted to 80% (+ amp.rto)
```

Note: If the maximum amplitude of the voice data used for input to the voice compression program exceeds 90% of the effective value of 16-bit PCM data, the quality of compressed voice data may deteriorate.

4.2.6 *vox2cmp.exe*

Function: Compresses the 16-bit PCM data with the compression ratio specified by the compression parameter file and saves the compressed data to a VOX file (VOX2 format).

Usage: DOS>`vox2cmp infile.pcm outfile.vox params.vpm`␣

Arguments: *infile.pcm* Input file name (16-bit PCM file)
outfile.vox Output file name (VOX file)
params.vpm Compression parameter file name (VPM file)
 This compression parameter file is output by the VOX2 evaluation tool "vox2parm.exe".

Example: DOS>`vox2cmp se.pcm se.vox d3w7N.vpm`
 ... Compressed to 9 kbps equivalent (in the case of se.pcm)

- Note:**
- If the maximum amplitude of the voice data input to this program exceeds 90% of the effective value of 16-bit PCM data, the quality of compressed voice data may deteriorate. To avoid this problem, use "pcm_norm.exe" to normalize the amplitude of the 16-bit PCM file. Also, if the amplitude of VOX2-compressed data exceeds that of the source PCM data, execute VOX2 compression twice, following the procedure below.
 1. Normalize the source PCM data using "pcm_norm.exe" (-c option not specified).
 2. Execute VOX2 compression using "vox2cmp.exe" (first compression).
 3. Convert the compressed VOX file into a PCM file using "vox2dec.exe".
 4. Compare the source PCM file and the PCM file generated in step 3 using "ampchk.exe".
 5. Readjust the PCM file generated in step 3 using "pcm_norm.exe" (-c option specified).
 6. Execute VOX2 compression using "vox2cmp.exe" (second compression).
 - Always be sure to use "vox2parm.exe" to generate and edit the VPM file.
 - The VOX files generated by "vox2cmp.exe" are not compatible with the VOX format files generated by "voxcmps.exe".

Reference: Several VOX parameter files are provided in the "voxtool\param\" directory. For details on VOX parameters, refer to Section 4.4, "VOX Parameters".

4.2.7 *voxcmprs.exe*

Function: Compresses the 16-bit PCM data with the compression ratio specified by the compression parameter file and saves the compressed data to a VOX file.

Usage: `DOS>voxcmprs infile.pcm outfile.vox params.vpm.↓`

Arguments:

<i>infile.pcm</i>	Input file name (16-bit PCM file)
<i>outfile.vox</i>	Output file name (VOX file)
<i>params.vpm</i>	Compression parameter file name (VPM file) This compression parameter file is output by the VOX evaluation tool "voxparam.exe".

Example: `DOS>voxcmprs se.pcm se.vox d3w8N.vpm`
... Compressed to 8 kbps equivalent (in the case of se.pcm)

- Note:**
- If the maximum amplitude of the voice data input to this program exceeds 90% of the effective value of 16-bit PCM data, the quality of compressed voice data may deteriorate. To avoid this problem, use "pcm_norm.exe" to normalize the amplitude of the 16-bit PCM file. Also, if the amplitude of VOX-compressed data exceeds that of the source PCM data, execute VOX compression twice, following the procedure below.
 1. Normalize the source PCM data using "pcm_norm.exe" (-c option not specified).
 2. Execute VOX compression using "voxcmprs.exe" (first compression).
 3. Convert the compressed VOX file into a PCM file using "voxdec.exe".
 4. Compare the source PCM file and the PCM file generated in step 3 using "ampchk.exe".
 5. Readjust the PCM file generated in step 3 using "pcm_norm.exe" (-c option specified).
 6. Execute VOX compression using "voxcmprs.exe" (second compression).
 - Always be sure to use "voxparam.exe" to generate and edit the VPM file.
 - The VOX files generated by "voxcmprs.exe" are not compatible with the VOX2 format files generated by "vox2cmp.exe".

Reference: Several VOX parameter files are provided in the "voxtool\param\" directory. For details on VOX parameters, refer to Section 4.4, "VOX Parameters".

4.2.8 *adpcmprs.exe*

Function: Compresses the 16-bit PCM data with the compression ratio specified by an option and saves the compressed data to an ADPCM file.
The ADPCM file thus output can be loaded using the ADPCM evaluation tool "adpparam.exe".

Usage: DOS>**adpcmprs** [*option*] *infile.pcm outfile.adp*↵

Arguments: *infile.pcm* Input file name (16-bit PCM file)
outfile.adp Output file name (ADPCM file)
option Specify a compression ratio (optional)
 Choose one of the following switches for this specification:
 -16 Compress voice to 16 kbps equivalent.
 -24 Compress voice to 24 kbps equivalent (default).
 -32 Compress voice to 32 kbps equivalent.
 -40 Compress voice to 40 kbps equivalent.

Example: DOS>adpcmprs -16 sample1.pcm sample2.adp ... Compressed to 16 kbps equivalent
 DOS>adpcmprs sample1.pcm sample2.adp ... Compressed to 24 kbps equivalent

Note: If the maximum amplitude of the voice data input to this program exceeds 90% of the effective value of 16-bit PCM data, the quality of compressed voice data may deteriorate. To avoid this problem, use "pcm_norm.exe" to normalize the amplitude of the 16-bit PCM file.

4.2.9 vsxcmprs.exe

Function: Compresses the 16-bit PCM data with the compression ratio specified by an option and saves the compressed data to a VSX file. The VSX file thus output can be loaded using the VSX evaluation tool "vsxparam.exe".

Usage: DOS>**vsxcmprs** [-cXX] [-tX] [-s level] *infile.pcm* *outfile.vsx*↓

Arguments: *infile.pcm* Input file name (16-bit PCM file)
outfile.vsx Output file name (VSX file)
-cXX Specify a compression ratio (optional)
-c16 Compress voice to 16 kbps equivalent.
-c24 Compress voice to 24 kbps equivalent (default).
-c32 Compress voice to 32 kbps equivalent.
-c40 Compress voice to 40 kbps equivalent.
-tX Specify a compression ratio in the timebase direction (optional)
-t1 Not compressed (default)
-t2 Compressed by a factor of 1/2.
-t3 Compressed by a factor of 1/3.
-t4 Compressed by a factor of 1/4.
-s level Silent packet level (optional)
 If the difference in voice level from the preceding data exceeds the value specified by *level*, the data is handled as a silent packet. The effective specification range of *level* is 0 to 5,000. Be sure to insert a space between -s and *level*. If this option is omitted, the default value 0 is assumed.

Example: DOS>vsxcmprs -c16 -t2 -s 50 sample1.pcm sample2.vsx
 ... Compressed to 16 kbps equivalent, compressed by a factor of 1/2 in the timebase direction, with silent packet level = 50
 DOS>vsxcmprs sample1.pcm sample2.vsx
 ... Compressed to 24 kbps equivalent, not compressed in the timebase direction, with silent packet level = 0

Note: If the maximum amplitude of the voice data input to this program exceeds 90% of the effective value of 16-bit PCM data, the quality of compressed voice data may deteriorate. To avoid this problem, use "pcm_norm.exe" to normalize the amplitude of the 16-bit PCM file.

4.2.10 *ppccmprs.exe*

Function: Converts the 16-bit PCM data into a packed PCM format file.

Usage: DOS>**ppccmprs** *infile.pcm outfile.ppc*↵

Arguments: *infile.pcm* Input file name (16-bit PCM file)
outfile.ppc Output file name (packed PCM file)

Example: DOS>ppccmprs sample1.pcm sample2.ppc

Note: If the maximum amplitude of the voice data input to this program exceeds 90% of the effective value of 16-bit PCM data, the quality of compressed voice data may deteriorate. To avoid this problem, use "pcm_norm.exe" to normalize the amplitude of the 16-bit PCM file.

4.2.11 bin2s.exe

Function: Converts the binary file (VOX, VPM, ADPCM, VSX, or PPC file) into a text file in E0C33 assembly source format.
 Since results are output to the standard output device (stdout), use the redirect function of DOS to save the converted data to a file.

Usage: DOS>bin2s [-l *symbol*] *infile.bin* > *outfile.s*␣

Arguments: *infile.bin* Input file name (binary file)
outfile.bin Output file name (assembly source file)
 -l *symbol* Define assembler symbol name (optional)
 If this option is omitted, the input file name is used as the symbol name.

Example: 1) -If the -l option is omitted, the input file name is assumed to be the assembler symbol name.

```
DOS>bin2s sample1.vox > sample1.s
DOS>type sample1.s
.global sample1
.align 2
sample1:
.byte 0xab 0xcd 0xef 0x00 0x11 0x22 0x33 0x44
.byte 0x10 0x29 0x38 0x47 0xab 0x34 0x45 0x88
:
```

DOS>

2) To use a symbol name that is different from the file name, use the -l option to specify the symbol name.

```
DOS>bin2s -l Smp01 sample1.vox > sample1.s
DOS>type sample1.s
.global Smp01
.align 2
Smp01:
.byte 0xab 0xcd 0xef 0x00 0x11 0x22 0x33 0x44
.byte 0x10 0x29 0x38 0x47 0xab 0x34 0x45 0x88
:
```

DOS>

Note: Symbol name specification is subject to the limitations below.

- Symbol length: Maximum of 32 characters
- Usable characters: a to z, A to Z, 0 to 9, _

4.2.12 pcm2s.exe

Function: Converts the 16-bit PCM file into a text file in E0C33 assembly source format. The output file contains the 10-bit amplitude PCM data that has been output in units of halfword length. Since results are output to the standard output device (stdout), use the redirect function of DOS to save the converted data to a file.

Usage: DOS>**pcm2s** [-1 *symbol*] *infile.bin* > *outfile.s*␣

Arguments: *infile.bin* Input file name (16-bit PCM file)
outfile.bin Output file name (assembly source file)
 -1 *symbol* Define assembler symbol name (optional)
 If this option is omitted, the input file name is used as the symbol name.

Example: 1) -If the -l option is omitted, the input file name is assumed to be the assembler symbol name.

```
DOS>pcm2s sample1.pcm > sample1.s
DOS>type sample1.s
.global sample1
.align 2
sample1:
.word 0x???
.half 0x0001 0x0001 0x0001 0x0001 0x0000 0x0001 0x0000 0x0001
.half 0x0001 0x0000 0x0001 0x0000 0x0000 0x0000 0x0000 0x0000
:
.half 0xFF33 0xFFBF 0xFF40 0xFFC6 0xFFDB 0xFFC8 0xFFFF 0x0035
.half 0x005C 0x0071 0x00EC 0x00EF 0x0164 0x01AC 0x0045 0x012E
:
DOS>
```

The ??? in data shows the total number of data calculated in units of halfword length.

2) To use a symbol name that is different from the file name, use the -l option to specify the symbol name.

```
DOS>pcm2s -l Smp01 sample1.pcm > sample1.s
DOS>type sample1.s
.global Smp01
.align 2
Smp01:
.word 0x???
.half 0x0001 0x0001 0x0001 0x0001 0x0000 0x0001 0x0000 0x0001
.half 0x0001 0x0000 0x0001 0x0000 0x0000 0x0000 0x0000 0x0000
:
.half 0xFF33 0xFFBF 0xFF40 0xFFC6 0xFFDB 0xFFC8 0xFFFF 0x0035
.half 0x005C 0x0071 0x00EC 0x00EF 0x0164 0x01AC 0x0045 0x012E
:
DOS>
```

The ??? in data shows the total number of data calculated in units of halfword length.

Note: Symbol name specification is subject to the limitations below.

- Symbol length: Maximum of 32 characters
- Usable characters: a to z, A to Z, 0 to 9, _

4.2.13 *bdmp.exe*

Function: Dumps the input binary file in a specified format.
 Since results are output to the standard output device (stdout), use the redirect function of DOS to save the dumped data to a file.

Usage: DOS>*bdmp option infile > outfile*␣

Arguments: *infile* Input file name (binary file)
outfile Output file name (text file)
option Specify output format (optional)
 Choose one of the following switches for this specification:
-b Output in byte format
-l Output in little endian short format
-m Output in big endian short format

Example: DOS>*bdmp -b se.vox*
 00000000 83 95 03 FE 78 42 4B 4B 4B 64 64 05 09 78 84 4C
 00000010 EE 9B 00 00 00 00 FF 01 00 00 E7 03 91 1C 9C 34
 00000020 A8 57 7D 67 67 DA E4 E3 5E 72 48 E2 62 A0 71 E0
 :
 00000D90 21 94 15 81 1D 54 3D 3A 1B CD 03 FB 05 1E 16 6F
 00000DA0 46 04 FE A6 03 2D 91 19 23 24 8E 0C CE 06 A6 06
 00000DB0 CC 05 89 06 0A 00
 DOS>*bdmp -l se.vox*
 00000000 9583 FE03 4278 4B4B 644B 0564 7809 4C84
 00000010 9BEE 0000 0000 01FF 0000 03E7 1C91 349C
 00000020 57A8 677D DA67 E3E4 725E E248 A062 E071
 :
 00000D90 9421 8115 541D 3A3D CD1B FB03 1E05 6F16
 00000DA0 0446 A6FE 2D03 1991 2423 0C8E 06CE 06A6
 00000DB0 05CC 0689 000A
 DOS>*bdmp -m se.vox*
 00000000 8395 03FE 7842 4B4B 4B64 6405 0978 844C
 00000010 EE9B 0000 0000 FF01 0000 E703 911C 9C34
 00000020 A857 7D67 67DA E4E3 5E72 48E2 62A0 71E0
 :
 00000D90 2194 1581 1D54 3D3A 1BCD 03FB 051E 166F
 00000DA0 4604 FEA6 032D 9119 2324 8E0C CE06 A606
 00000DB0 CC05 8906 0A00

4.2.14 *vox2dec.exe*

Function: Decodes the VOX file compressed by "vox2cmp.exe" and saves the decoded data to a PCM file.

Usage: DOS>**vox2dec** *infile.vox outfile.pcm*↵

Arguments: *infile.vox* Input file name (VOX file)
outfile.pcm Output file name (PCM file)

Example: DOS>vox2dec sample1.vox sample1.pcm

Note: This tool cannot decode VOX files compressed by "voxcmprs.exe".

Reference: The source code of this tool is provided in the "voxtool\src\vox2dec\" directory.

4.2.15 *voxdec.exe*

Function: Decodes the VOX file compressed by "voxcmprs.exe" and saves the decoded data to a PCM file.

Usage: DOS>**voxdec** *infile.vox outfile.pcm*↵

Arguments: *infile.vox* Input file name (VOX file)
outfile.pcm Output file name (PCM file)

Example: DOS>voxdec sample1.vox sample1.pcm

Note: This tool cannot decode VOX files compressed by "vox2cmp.exe".

Reference: The source code of this tool is provided in the "voxtool\src\voxdec\" directory.

4.2.16 *adpdec.exe*

Function: Decodes the ADPCM file compressed by "adpcmprs.exe" and saves the decoded data to a PCM file.

Usage: DOS>**adpdec** *infile.adp outfile.pcm*↵

Arguments: *infile.adp* Input file name (ADPCM file)
outfile.pcm Output file name (PCM file)

Example: DOS>adpdec sample1.adp sample1.pcm

Reference: The source code of this tool is provided in the "voxtool\src\adpdec\" directory.

4.2.17 *ppcdec.exe*

Function: Decodes the packed PCM file compressed by "ppccmprs.exe" and expands the decoded data into a PCM file.

Usage: DOS>**ppcdec** *infile.ppc outfile.pcm*↵

Arguments: *infile.ppc* Input file name (PPC file)
outfile.pcm Output file name (PCM file)

Example: DOS>ppcdec sample1.ppc sample1.pcm

Reference: The source code of this tool is provided in the "voxtool\src\ppcdec\" directory.

4.2.18 vsxdec.exe

Function: Decodes the VSX file compressed by "vsxcmprs.exe" and saves the decoded data to a PCM file.

Usage: DOS>**vsxdec** [*option*] *infile.vsx outfile.pcm*↵

Arguments: *infile.vsx* Input file name (VSX file)
outfile.pcm Output file name (PCM file)
option Specify playback speed (optional)
 Choose one of the switches below for this specification. The saved data is reproduced at the speed specified here.

- norm Normal speed (default)
- f15 1.5 times normal speed
- f20 2 times normal speed
- f30 3 times normal speed
- f40 4 times normal speed
- f60 6 times normal speed
- f80 8 times normal speed
- f120 12 times normal speed
- f160 16 times normal speed
- s15 1/1.5 times normal speed
- s20 1/2 times normal speed

Example: DOS>vsxdec -f20 sample1.vsx sample1.pcm ... Saved with ×2 playback speed
 DOS>vsxdec sample1.vsx sample1.pcm ... Saved with normal playback speed

Reference: The source code of this tool is provided in the "voxtool\src\vsxdec\" directory.

4.2.19 *ampchk.exe*

Function: Compares the maximum amplitudes of two input PCM files and outputs the ratio between the two to a file "amp.rto" and the screen.

Use this tool after VOX2/VOX compression. The file "amp.rto" contains the ratio of maximum amplitudes between the two input files (afterfile and original). This output data can be loaded for use in amplitude readjustment after VOX2/VOX compression by specifying the -c option in "pcm_norm.exe".

Usage: DOS>**ampchk** *original.pcm* *afterfile.pcm*↵

Arguments: *original.pcm* Input file name 1 (PCM file before VOX2/VOX compression)

afterfile.pcm Input file name 2 (PCM file after VOX2/VOX compression)

Example: (1) DOS>pcm_norm sample.pcm temp1.pcm ... Normalize amplitude
 (2) DOS>vox2cmp temp1.pcm temp2.vox para.vpm ... Compress in VOX2 format
 (3) DOS>vox2dec temp2.vox temp2.pcm ... Decode VOX2
 (4) DOS>**ampchk temp1.pcm temp2.pcm**
 ... Compare amplitudes before and after compression
 (5) DOS>pcm_norm -c temp2.pcm temp3.pcm
 ... Readjust amplitude (-c must be specified)
 (6) DOS>vox2cmp temp3.pcm final.vox para.vpm ... Compress in VOX2 format

4.2.20 *addslnt.exe*

Function: Adds a specified number of silent data to the last part of the input PCM file.

Usage: DOS>**addslnt** *infile.pcm* *outfile.pcm* [*NumOfSilent*]↵

Arguments: *infile.pcm* Input file name (16-bit PCM file)

outfile.pcm Output file name (16-bit PCM file)

NumOfSilent Number of silent data (optional)

If this option is omitted, the default value "short 128" (256 bytes) is assumed.

Example: DOS>addslnt sample1.pcm sample2.pcm 512
 ... 512 (short) entries of silent data are added to the last part of "sample1" to create sample2.

4.2.21 pcm2wav.exe

Function: Converts the input PCM file into a WAV file of the same frequency.

Usage: `DOS>pcm2wav [SamplingRate] infile.pcm outfile.wav`

Arguments: *infile.pcm* Input file name (16-bit PCM file)
outfile.wav Output file name (WAV file)
SamplingRate Input PCM file sampling rate (optional)
 For 8 kHz, specify 8; for 44.1 kHz, specify 44.1. If this option is omitted, the default value 8 (8 kHz) is assumed.

Example: `DOS>pcm2wav 44.1 sample1.pcm sample1.wav`
 ... Convert a PCM file with 44.1 kHz sampling into a WAV file.
`DOS>pcm2wav sample2.pcm sample2.wav`
 ... Convert a PCM file with 8 kHz sampling into a WAV file.

4.2.22 wav2pcm.exe

Function: Converts the input WAV file into a PCM file of the same frequency.

Usage: `DOS>wav2pcm infile.wav outfile.pcm`

Arguments: *infile.wav* Input file name (WAV file)
outfile.pcm Output file name (PCM file)

Example: `DOS>wav2pcm sample1.wav sample1.pcm`

4.2.23 Executing Tools from a Batch File

Voice ROM generation tools are all 32-bit applications that can be executed from the DOS prompt. Therefore, a series of processing steps can be executed after creating a batch file.

The following shows examples of processing executed using the batch files provided in the "voxtool\sample\" directory.

Each batch file was created to execute voice ROM generation tools in "voxtool\bin\" from "voxtool\sample\" as the current directory. Correct any batch file as necessary before using.

pcm2pcms.bat

Converts PCM voice data into an assembly source.

Processing: 1) High-pass filtering (120 Hz cut-off)
 2) Normalization (90% amplitude)
 3) Conversion into assembly source file

Input file: *file_name.pcm* 16-bit PCM file

Output file: *file_name.pcms* Assembly source file

Contents of file: @echo off
 if "%1"==" " goto ERR
 echo voxflt %1
 ..\bin\voxflt -l 120 %1.pcm %1N.pcm
 echo pcm_norm %1
 ..\bin\pcm_norm %1.pcm %1N.pcm
 echo pcm2s %1
 ..\bin\pcm2s -l %1 %1N.pcm > %1.pcms
 goto END
 :ERR
 echo Please input filename
 :END

Example: >pcm2pcms se
 Create an assembly source file "se.pcms" from the 16-bit PCM file "se.pcm". The output file "se.pcms" has the global label (se), which is the same as the input file name.

```
.global se
.align 2
se:
.word 0x667a

.half 0x0001 0x0001 0x0001 0x0001 0x0000 0x0001 0x0000 0x0001
.half 0x0001 0x0000 0x0001 0x0000 0x0000 0x0000 0x0000 0x0000
.half 0x0001 0x0000 0x0000 0x0001 0x0000 0x0000 0x0001 0x0000
:
; total 26234 short data + 4byte header
```

Reference: "4.2.4 voxflt.exe", "4.2.5 pcm_norm.exe", "4.2.12 pcm2s.exe"

pcm2vox2s.bat

Converts PCM voice data into an assembly source after compressing it in VOX2 format.

Processing:

- 1) High-pass filtering (120 Hz cut-off)
- 2) Normalization (50% amplitude)
- 3) VOX2 compression (using VOX parameter "d3w7N.vpm")
- 4) Amplitude check after compression, then amplitude adjustment and recompression
- 5) Conversion into assembly source file

Input file: *file_name.pcm* 16-bit PCM file

Output file: *file_name.voxs37* Assembly source file

```
Contents of file: @echo off
if "%1"==" " goto ERR
echo voxflt %1
..\bin\voxflt -l 120 %1.pcm %1H.pcm
rem PASS 1
echo PASS 1
echo pcm_norm %1
..\bin\pcm_norm -r 50 %1H.pcm %1N.pcm
echo vox2cmp %1
..\bin\vox2cmp %1N.pcm %1.vox ..\param\d3w7N.vpm
echo vox2dec %1.vox
..\bin\vox2dec %1.vox %1vox.pcm
echo ampchk %1
..\bin\ampchk %1N.pcm %1vox.pcm
rem PASS 2
echo PASS 2
echo pcm_norm %1
..\bin\pcm_norm -c %1H.pcm %1N.pcm
echo vox2cmp %1
..\bin\vox2cmp %1N.pcm %1.vox ..\param\d3w7N.vpm
echo bin2s %1
..\bin\bin2s -l %137N %1.vox > %1.voxs37
goto END
:ERR
echo Please input filename
:END
```

Example: >pcm2vox2s se

Create an assembly source file "se.voxs37" from the 16-bit PCM file "se.pcm". The VOX compression parameter file "d3w7N.vpm" is used for this processing. To create the assembly source file with another compression ratio, change the "..\param\d3w7N.vpm" part to another parameter file. The batch file in this example was created for compression in VOX2 format. If VOX compression is needed, change the tool name in the file name from "vox2..." to "vox...". The output file "se.voxs37" has a global label (se37N) derived from the input file name by adding "37N".

```
.global se37N
.align 2
se37N:
.byte 0x83 0x95 0x03 0xfe 0x28 0x43 0x4b 0x4b
.byte 0x4b 0x64 0x64 0x00 0x07 0x4a 0x8f 0x86
:
; total 3705 bytes data
```

Reference: "4.2.4 voxflt.exe", "4.2.5 pcm_norm.exe", "4.2.6 vox2cmp.exe", "4.2.11 bin2s.exe", "4.2.19 ampchk.exe"

pcm2adps.bat

Converts PCM voice data into an assembly source after compressing it in ADPCM format.

Processing: 1) High-pass filtering (120 Hz cut-off)
 2) Normalization (90% amplitude)
 3) ADPCM compression (compressed to 24 kbps equivalent)
 4) Conversion into assembly source file

Input file: *file_name.pcm* 16-bit PCM file

Output file: *file_name.adps24* Assembly source file

Contents of file: @echo off
 if "%1"==" " goto ERR
 echo voxflt %1
 ..\bin\voxflt -l 120 %1.pcm %1H.pcm
 echo pcm_norm %1
 ..\bin\pcm_norm -r 90 %1H.pcm %1N.pcm
 echo adpcmprs %1
 ..\bin\adpcmprs -24 %1N.pcm %1.adp
 echo bin2s %1
 ..\bin\bin2s -l %1a24 %1.adp > %1.adps24
 goto END
 :ERR
 echo Please input filename
 :END

Example: >pcm2adps se

Create an assembly source file "se.adps24" from the 16-bit PCM file "se.pcm". The ADPCM compression option used for "adpcmprs.exe" is "-24" (compression to 24 kbps equivalent). To create the assembly source file with another compression ratio, change "-24" to another option. The output file "se.adps24" has a global label (sea24) derived from the input file name by adding "a24".

```

.global sea24
.align 2
sea24:
.byte 0x41 0x70 0x26 0x00 0x00 0x00 0x00 0x00
.byte 0x00 0x02 0xae 0xd7 0x19 0x24 0xe2 0x71
.byte 0x34 0x93 0x80 0x12 0x49 0x24 0x91 0x5b
:
; total 9850 bytes data

```

Reference: "4.2.4 voxflt.exe", "4.2.5 pcm_norm.exe", "4.2.8 adpcmprs.exe", "4.2.11 bin2s.exe"

pcm2vsxs.bat

Converts PCM voice data into an assembly source after compressing it in VSX format.

Processing: 1) High-pass filtering (120 Hz cut-off)
 2) Normalization (90% amplitude)
 3) VSX compression (compressed to 24 kbps equivalent, timebase direction = 1/2, silent packet level = 20)
 4) Conversion into assembly source file

Input file: *file_name*.pcm 16-bit PCM file

Output file: *file_name*.vsxs24 Assembly source file

Contents of file: @echo off
 if "%1"==" " goto ERR
 echo voxflt %1
 ..\bin\voxflt -l 120 %1.pcm %1H.pcm
 echo pcm_norm %1
 ..\bin\pcm_norm -r 90 %1H.pcm %1N.pcm
 echo vsxcmprs %1
 ..\bin\vsxcmprs -c24 -t2 -s 20 %1N.pcm %1.vsx
 echo bin2s %1
 ..\bin\bin2s -l %1v24t2 %1.vsx > %1.vxsxs24
 goto END
 :ERR
 echo Please input filename
 :END

Example: >pcm2vsxs se

Create an assembly source file "se.vxsxs24" from the 16-bit PCM file "se.pcm". The VSX compression options used for "vsxcmprs.exe" are "-c24" (compression to 24 kbps equivalent), "-t2" (compression by a factor of 1/2 in the timebase direction), and "-s 20" (silent packet level = 20). To create the assembly source file under other conditions, change these options to those desired. The output file "se.vxsxs24" has a global label (sev24t2) derived from the input file name by adding "v24t2".

```

.global sev24t2
.align 2
sev24t2:
.byte 0x53 0x22 0x01 0x01 0x1b 0xd6 0xdd 0x61
.byte 0x92 0x50 0x05 0x06 0x49 0x27 0x26 0x00
.byte 0x3d 0xb2 0x53 0x48 0x00 0xda 0x69 0x14
:
; total 4733 bytes data

```

Reference: "4.2.4 voxflt.exe", "4.2.5 pcm_norm.exe", "4.2.9 vsxcmprs.exe", "4.2.11 bin2s.exe"

pcm2ppcs.bat

Converts PCM voice data into an assembly source after compressing it in packed PCM format.

Processing: 1) High-pass filtering (120 Hz cut-off)
 2) Normalization (90% amplitude)
 3) PPC compression
 4) Conversion into assembly source file

Input file: *file_name.pcm* 16-bit PCM file

Output file: *file_name.vxsx24* Assembly source file

Contents of file: @echo off
 if "%1"==" " goto ERR
 echo voxflt %1
 ..\bin\voxflt -l 120 %1.pcm %1H.pcm
 echo pcm_norm %1
 ..\bin\pcm_norm %1H.pcm %1N.pcm
 echo ppccmprs %1
 ..\bin\ppccmprs %1N.pcm %1.ppc
 echo bin2s %1
 ..\bin\bin2s -l %1p %1.ppc > %1.ppcs
 goto END
 :ERR
 echo Please input filename
 :END

Example: >pcm2ppcs se
 Create an assembly source file "se.ppcs" from the 16-bit PCM file "se.pcm". The output file "se.ppcs" has a global label (sep) derived from the input file name by adding "p".

```

.global sep
.align 2
sep:
.byte 0x50 0x80 0x66 0x00 0x00 0x00 0x00 0x00
.byte 0x00 0x01 0x00 0x00 0x00 0xff 0x00 0xff
.byte 0x00 0x00 0xff 0x00 0xff 0xff 0xff 0xff
:
; total 32436 bytes data

```

Reference: "4.2.4 voxflt.exe", "4.2.5 pcm_norm.exe", "4.2.10 ppccmprs.exe", "4.2.11 bin2s.exe"

4.2.24 Executing Tools from a Make File

Use of a make file allows various types of voice data with different compression ratios, etc. to be managed collectively. The following make files are provided in the "voxtool\sample\" directory:

```
adpdata.mak    Creates ADPCM voice ROM data
vox2data.mak   Creates VOX2 voice ROM data
vsxdata.mak    Creates VSX voice ROM data
ppcdata.mak    Creates packed PCM voice ROM data
```

Shown below is an execution example involving "vsxdata.mak".

For details on the make file format and the functions of make, refer to the "E0C33 Family C Compiler Package Manual".

vsxdata.mak

This make file executes the following processing using the sample PCM file "se.pcm" as the voice source data:

- 1) High-pass filtering (120 Hz cut-off)
- 2) Normalization (90% amplitude)
- 3) Creation of PPC file
- 4) Creation of a VSX file with each compression ratio
- 5) Conversion into assembly source file

Finally, the above processing produces the assembly source files shown below and a file "vsxdata.vs" in which these assembly source files have been combined.

Packed PCM voice ROM data file: se.ppc

VSX-compressed voice ROM data files: se.x16t2, se.x24t1, se.x24t3, se.x24t4, se.x32t2, se.x40t2

The value ".x??" denotes the numeric values representing the compression ratio options specified in "vsxcmprs.exe". The value "t?" indicates that data is compressed by a factor of 1/? in the timebase direction. All silent packet levels are 20.

To generate voice ROM data files, start make by entering the following:

```
DOS>make -f vsxdata.mak
```

The make file "vsxdata.mak" is created to be run from "voxtool\sample\" as the current directory. Therefore, you need to specify the "make.exe" directory by the PATH command or copy "make.exe" into the "voxtool\sample\" directory before opening the make file.

The make file "sample.mak" contains a command description that allows all generated files except the original (se.pcm) to be deleted from the hard disk. To execute this function, start make by entering the following:

```
DOS>make -f vsxdata.mak clean
```

The contents of files are shown below.

```
# macro definitions for tools & dir
```

```
TOOL_DIR    = ..\bin
PCM_NORM    = $(TOOL_DIR)\pcm_norm.exe
VOXFLT      = $(TOOL_DIR)\voxflt.exe
VSXCMPRS    = $(TOOL_DIR)\vsxcmprs.exe
PPCCMPRS    = $(TOOL_DIR)\ppccmprs.exe
BIN2S       = $(TOOL_DIR)\bin2s.exe
```

```
# suffix & rule definitions
```

```
.SUFFIXES : .pcm .pcmn .pcmh .x16t2 .x24t2 .x24t1 .x24t3 .x24t4 .x32t2
.x40t2 .ppc .pp
```

4 VOX33 TOOL REFERENCE

```
.pcm.x16t2 :  
$(VOXFLT) -l 120 $*.pcm $*.pcmh  
$(PCM_NORM) -r 90 $*.pcmh $*.pcmn  
$(VSXCMPRS) -c16 -t2 -s 20 $*.pcmn $*.vsx  
$(BIN2S) -l $*v16t2 $*.vsx > $*.x16t2
```

```
.pcm.x24t1 :  
$(VOXFLT) -l 120 $*.pcm $*.pcmh  
$(PCM_NORM) -r 90 $*.pcmh $*.pcmn  
$(VSXCMPRS) -c24 -t1 -s 20 $*.pcmn $*.vsx  
$(BIN2S) -l $*v24t1 $*.vsx > $*.x24t1
```

```
.pcm.x24t2 :  
$(VOXFLT) -l 120 $*.pcm $*.pcmh  
$(PCM_NORM) -r 90 $*.pcmh $*.pcmn  
$(VSXCMPRS) -c24 -t2 -s 20 $*.pcmn $*.vsx  
$(BIN2S) -l $*v24t2 $*.vsx > $*.x24t2
```

```
.pcm.x24t3 :  
$(VOXFLT) -l 120 $*.pcm $*.pcmh  
$(PCM_NORM) -r 90 $*.pcmh $*.pcmn  
$(VSXCMPRS) -c24 -t3 -s 20 $*.pcmn $*.vsx  
$(BIN2S) -l $*v24t3 $*.vsx > $*.x24t3
```

```
.pcm.x24t4 :  
$(VOXFLT) -l 120 $*.pcm $*.pcmh  
$(PCM_NORM) -r 90 $*.pcmh $*.pcmn  
$(VSXCMPRS) -c24 -t4 -s 20 $*.pcmn $*.vsx  
$(BIN2S) -l $*v24t4 $*.vsx > $*.x24t4
```

```
.pcm.x32t2 :  
$(VOXFLT) -l 120 $*.pcm $*.pcmh  
$(PCM_NORM) -r 90 $*.pcmh $*.pcmn  
$(VSXCMPRS) -c32 -t2 -s 20 $*.pcmn $*.vsx  
$(BIN2S) -l $*v32t2 $*.vsx > $*.x32t2
```

```
.pcm.x40t2 :  
$(VOXFLT) -l 120 $*.pcm $*.pcmh  
$(PCM_NORM) -r 90 $*.pcmh $*.pcmn  
$(VSXCMPRS) -c40 -t2 -s 20 $*.pcmn $*.vsx  
$(BIN2S) -l $*v40t2 $*.vsx > $*.x40t2
```

```
.pcm.pp :  
$(VOXFLT) -l 120 $*.pcm $*.pcmh  
$(PCM_NORM) $*.pcmh $*.pcmn  
$(PPCCMPRS) $*.pcmn $*.ppc  
$(BIN2S) -l $*p $*.ppc > $*.pp
```

dependency list

ALL_S = se.x16t2 se.x24t1 se.x24t2 se.x24t3 se.x24t4 se.x32t2 se.x40t2 se.pp

```
vsxdata.vs : $(ALL_S)
type se.pp > vsxdata.vs
type se.x16t2 >> vsxdata.vs
type se.x24t1 >> vsxdata.vs
type se.x24t2 >> vsxdata.vs
type se.x24t3 >> vsxdata.vs
type se.x24t4 >> vsxdata.vs
type se.x32t2 >> vsxdata.vs
type se.x40t2 >> vsxdata.vs
```

```
se.x16t2 : se.pcm
se.x24t1 : se.pcm
se.x24t2 : se.pcm
se.x24t3 : se.pcm
se.x24t4 : se.pcm
se.x32t2 : se.pcm
se.x40t2 : se.pcm
se.pp : se.pcm
```

```
# clean files except source
```

```
clean:
```

```
del *.vs
del *.x16t2
del *.x24t1
del *.x24t2
del *.x24t3
del *.x24t4
del *.x32t2
del *.x40t2
del *.pcmn
del *.pcmh
del *.vsx
del *.pp
del *.ppc
```

4.3 Voice Compression/Processing Evaluation Tools

This section describes the functions of each voice compression/processing evaluation tool and how to use them.

- Note:**
- The voice compression/processing evaluation tools ("vox2parm.exe", "voxparam.exe", "adpparam.exe", "vsxparam.exe", and "vscparam.exe") use algorithms that are similar to, but not identical to the libraries implemented on the E0C33 chip. Use these tools for the preliminary evaluation of compression parameters, etc. Also, because sound quality depends on analog components such as the speaker, microphone, and op-amp, the sound quality as evaluated on a PC may differ from that actually obtained in the application system. The final sound quality, operation, etc. must be evaluated using the actual application system that contains the E0C33 chip.
 - The size of files that can be loaded by each voice compression/processing evaluation tool varies with the operating environment. The upper limit for PCM data is the size that takes approximately 7 to 8 minutes to load.
 - Before any voice compression/processing evaluation tool can be used, a sound card (SoundBlaster 16 or compatible) that supports 8 kHz-sampling and 16-bit monaural voice input/output is required. The evaluation tools may not work with a sound card that only supports 8-bit data.

4.3.1 vox2parm.exe

This tool is used to evaluate VOX2-compressed voice data after entering voice data from a 16-bit PCM file or a microphone.

It can reproduce the original data and the compressed voice data, allowing you to change the compression ratio while listening. The compression ratio data thus obtained can be saved to a VPM file in order to specify the compression ratio when executing "vox2cmp.exe" or for use as the VOX parameters that are embedded into the user program. Also, the compressed voice data can be saved to a 16-bit PCM file for use as voice ROM data by "pcm2s.exe". The VOX2-compressed data can be loaded for expansion and playback.

Starting and quitting



vox2parm.exe

Double-click on the "vox2parm.exe" icon to start the tool.

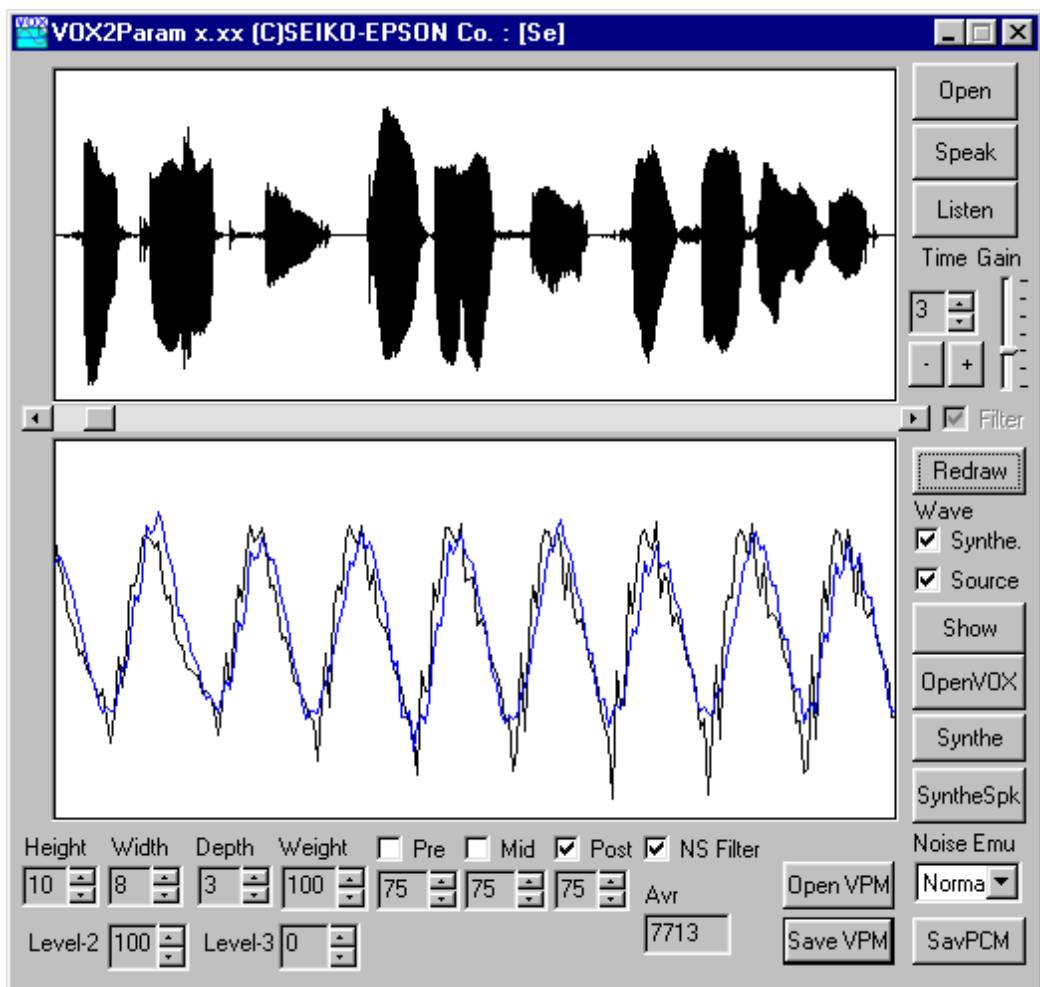
To quit "vox2parm.exe", click on the [Close] button at the upper right corner of the [VOX2Param] window.

Windows and the function of each part

[VOX2Param] window

When "vox2parm.exe" starts up, the [VOX2Param] window appears.


To perform any operation from this window, click on the appropriate control button using the mouse.




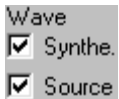
Controls for waveform display

Upper window: **Full-waveform display area**
 Displays the full waveform of the source voice data loaded from a file or entered from a microphone.

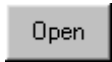
Lower window: **Partial-waveform display area**
 Displays a partial waveform of the source voice or compressed/expanded voice. Use the [Wave] check box to choose to display one or both of the source and compressed/expanded waveforms. The source voice waveform is displayed in black and the compressed/expanded data is displayed in blue.
 The waveforms displayed in this window can be scrolled using the scroll bar located at the top of the window.


 **[Redraw] button**
 Redraws the waveforms in both waveform display areas.


 **[Show] button**
 Redraws the waveform in the partial-waveform display area.


 **[Wave] check box**
 Used to choose the waveform to be displayed in the partial-waveform display area. If you choose [Synthe.], the compressed data waveform is displayed in blue. If you choose [Source], the source voice waveform is displayed in black. You can also choose both [Synthe.] and [Source].
 Note that the partial-waveform display area is not updated by this operation alone. After selecting or deselecting [Synthe.] or [Source], click on the [Redraw] or [Show] button or scroll the partial-waveform display area to redisplay the waveform.


Controls for source voice


 **[Open] button**
 Loads a 16-bit PCM file.
 The waveform of the loaded voice data is displayed in the waveform display area.

 **[Speak] button**
 Outputs the source voice data to the speaker via a sound card.
 If a VOX2 voice file has been loaded, the voice data is reproduced by expansion.

 **[Listen] button**
 Enters voice data from a microphone.
 The waveform of the entered voice data is displayed in the waveform display area.

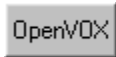
 **[Time]**
 When entering voice data from a microphone, set the input time here in units of seconds. The input time can be set in the range of 1 to 60 seconds. When this set time has elapsed after the [Listen] button is clicked, the voice input stops. The voice input cannot be stopped before the set time has elapsed.

 **[Gain]**
 When entering voice data from a microphone, set the input level here.

 **[+] button, [-] button**
 Increments ([+] button) or decrements ([-] button) the amplitude of the input data waveform in steps of 1/8. The waveforms displayed in the full and partial-waveform display areas are updated when you click on these buttons. The sound volume during playback also changes.

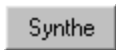
**[Filter] check box**

Turns on or off the filter function that removes noise when voice data is entered from a microphone. In the current version, however, it is set to on and you cannot choose to turn it off.

**[OpenVOX] button**

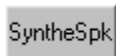
Loads a VOX2-compressed voice file. Always be sure to choose a VOX2 voice file that has been generated by "vox2cmp.exe". The waveform of the loaded voice data is displayed in the waveform display area.

When a VOX2 voice file is loaded, the expanded waveform is displayed in the waveform display area, and the waveform to be displayed in the partial-waveform display area cannot be selected. In the [Wave] check box, you can only choose [Synthe.].

Controls for compression and sound quality adjustment**[Synthe] button**

Compresses the source voice according to the parameters set. It does not reproduce voice data. After processing is completed, an average voice data rate (bps) is displayed in the [Avr] box located at the bottom of the window.

Note that the compressed data waveform in the partial-waveform display area is not updated by this operation alone. Click on the [Redraw] or [Show] button or scroll the partial-waveform display area to redisplay the waveform.

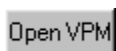
**[SyntheSpk] button**

Compresses the source voice according to the parameters set and outputs the result from a speaker. After processing is completed, an average voice data rate (bps) is displayed in the [Avr] box located at the bottom of the window.

Note that the compressed data waveform in the partial-waveform display area is not updated by this operation alone. Click on the [Redraw] or [Show] button or scroll the partial-waveform display area to redisplay the waveform.

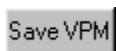
**[SavPCM] button**

Saves the compressed data to a 16-bit PCM file.

**[Open VPM] button**

Loads compression parameters from a specified VPM file.

The loaded parameter values are displayed on each VOX parameter control.

**[Save VPM] button**

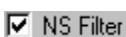
Saves the compression parameters that have been set by VOX parameter controls to a VPM file.

**[Noise EMU] combo box**

The default selection of this combo box is "Normal", which processes the unconverted 8-kHz-sampling source voice.

Choices "8bit", "Light", and "Heavy" are provided to produce sound quality approximately equal to that of the E0C33 chip by emulating the 8-bit D/A converter output on the E0C33 chip. If you choose "8bit", 10-bit data is converted into 8-bit form before being output.

If you choose "Light" or "Heavy", the quantization noise in the D/A converter output from the E0C33 chip is emulated. As a result, high tones are enhanced when sound is output. "Light" and "Heavy" respectively reduce or increase this effect. Note that these functions require a large amount of memory.

**[NS Filter] check box**

If you choose this option, the noise-shaving filter is enabled. This filter helps to reduce noise at high frequencies.

For details, refer to Section 4.4, "VOX Parameters".

**[Avr] box**

The average voice data rate after compression/expansion is displayed here.

Height	Width	Depth	Weight	<input type="checkbox"/> Pre	<input type="checkbox"/> Mid	<input checked="" type="checkbox"/> Post
10	8	3	100	75	75	75
Level-2	100	Level-3	0			

VOX parameters

Use these controls to set each VOX parameter. For details, refer to Section 4.4, "VOX Parameters".

Basic operation procedure

- Using the [Open] button, enter the 16-bit PCM file (.pcm) to be compressed.
For microphone input, set the recording time (seconds) in [Time] and the input level in [Gain] and click on the [Listen] button to enter voice data from a microphone.
The entered voice can be reproduced using the [Speak] button.
- Using the [Open VPM] button, enter the VOX parameter file (.vpm) to be referenced during compression.
The "voxtool\param\" directory contains a VOX parameter file for which various parameters have already been set.
- Correct each parameter as necessary while referring to Section 4.4, "VOX Parameters".
- Use the [SyntheSpk] button to reproduce the compressed voice.
- Repeat steps 3 and 4 until you achieve a good balance between compression ratio and sound quality. For the compression ratio, refer to the average voice data rate shown in the [Avr] box.
- Use the [Save VPM] button to save the desired VOX parameters to a file.
- Save the compressed voice data using the [SavPCM] button.

Note:

- Evaluate the VOX parameters using the actual application system before making the final decision.
- The VOX2 voice files handled by "vox2parm.exe" are not compatible with the VOX voice files handled by "voxparam.exe" and "voxcmprs.exe". The VOX2 voice data has better sound quality, but on the E0C33 chip it can only be reproduced. VOX voice data can be compressed and recorded, as well as reproduced, on the E0C33 chip.

4.3.2 voxparam.exe

This tool is used to evaluate VOX-compressed voice data after entering voice data from a 16-bit PCM file or a microphone.

It can reproduce the original data and the compressed voice data, allowing you to change the compression ratio while listening. The compression ratio data thus obtained can be saved to a VPM file in order to specify the compression ratio when executing "voxcmprs.exe" or for use as the VOX parameters that are embedded into the user program. Also, the compressed voice data can be saved to a 16-bit PCM file for use as voice ROM data by "pcm2s.exe". The VOX-compressed data can be loaded for expansion and playback.

Starting and quitting



Voxparam.exe

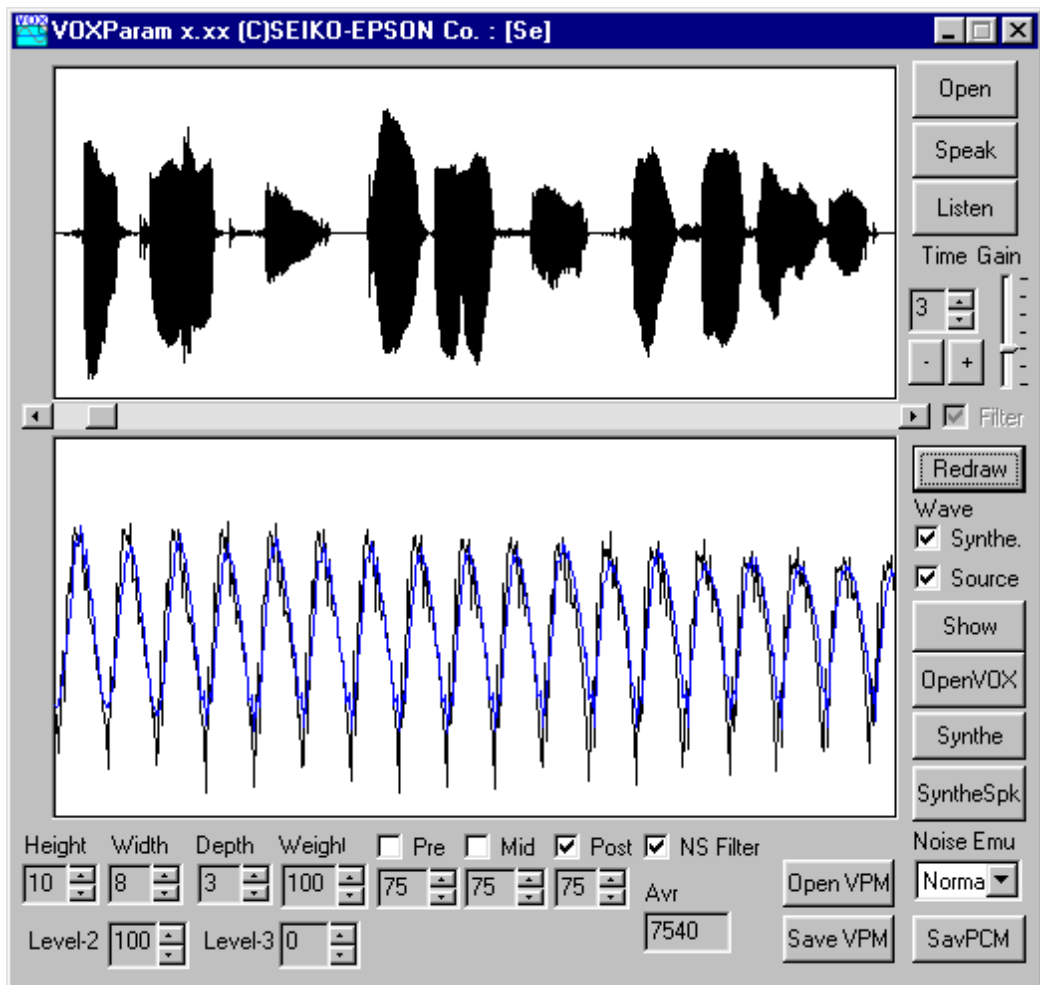
Double-click on the "voxparam.exe" icon to start the tool.

To quit "voxparam.exe", click on the [Close] button at the upper right corner of the [VOXParam] window.

Windows and the function of each part

[VOXParam] window


When "voxparam.exe" starts up, the [VOXParam] window appears. To perform any operation from this window, click on the appropriate control button using the mouse.




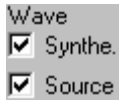
Controls for waveform display

Upper window: **Full-waveform display area**
 Displays the full waveform of the source voice data loaded from a file or entered from a microphone.

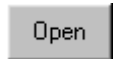
Lower window: **Partial-waveform display area**
 Displays a partial-waveform of the source voice or compressed/expanded voice. Use the [Wave] check box to choose to display one or both of the source and compressed/expanded waveforms. The source voice waveform is displayed in black and the compressed/expanded data is displayed in blue. The waveforms displayed in this window can be scrolled using the scroll bar located above the window.

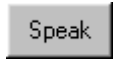
 **[Redraw] button**
 Redraws the waveforms in both waveform display areas.


 **[Show] button**
 Redraws the waveform in the partial-waveform display area.


 **[Wave] check box**
 Used to choose the waveform to be displayed in the partial-waveform display area. If you choose [Synthe.], the compressed data waveform is displayed in blue. If you choose [Source], the source voice waveform is displayed in black. You also can choose both [Synthe.] and [Source].
 Note that the partial-waveform display area is not updated by this operation alone. After selecting or deselecting [Synthe.] or [Source], click on the [Redraw] or [Show] button or scroll the partial-waveform display area to redisplay the waveform.


Controls for source voice


 **[Open] button**
 Loads a 16-bit PCM file. The waveform of the loaded voice data is displayed in the waveform display area.

 **[Speak] button**
 Outputs the source voice data to the speaker via a sound card.
 If a VOX voice file has been loaded, the voice data is reproduced by expansion.

 **[Listen] button**
 Enters voice data from a microphone.
 The waveform of the entered voice data is displayed in the waveform display area.

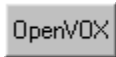
 **[Time]**
 When entering voice data from a microphone, set the input time here in units of seconds. The input time can be set in the range of 1 to 60 seconds. When this set time has elapsed after the [Listen] button is clicked, the voice input stops. The voice input cannot be stopped until the set time has elapsed.

 **[Gain]**
 When entering voice data from a microphone, set the input level here.

 **[+] button, [-] button**
 Increments ([+] button) or decrements ([-] button) the amplitude of the input data waveform in steps of 1/8. The waveforms displayed in the full-and partial-waveform display areas are updated when you click on these buttons. The sound volume during playback also changes.

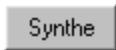
**[Filter] check box**

Turns on or off the filter function that cuts noise when entering voice from a microphone. In the current version, however, it is set to on and you cannot choose to turn it off.

**[OpenVOX] button**

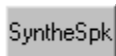
Loads a VOX-compressed voice file. Always be sure to choose a VOX voice file that has been generated by "voxcmps.exe". The waveform of the loaded voice data is displayed in the waveform display area.

When a VOX voice file is loaded, the expanded waveform is displayed in the waveform display area, and the waveform to be displayed in the partial-waveform display area cannot be selected. In the [Wave] check box, you only can choose [Synthe].

Controls for compression and sound quality adjustment**[Synthe] button**

Compresses the source voice according to the parameters set. It does not reproduce voice data. After processing is completed, the average voice data rate (bps) is displayed in the [Avr] button located at the bottom of the window.

Note that the compressed data waveform in the partial-waveform display area is not updated by this operation alone. Click on the [Redraw] or [Show] button or scroll the partial-waveform display area to redisplay the waveform.

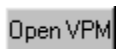
**[SyntheSpk] button**

Compresses the source voice according to the parameters set and outputs the result from a speaker. After processing is completed, the average voice data rate (bps) is displayed in the [Avr] button located at the bottom of the window.

Note that the compressed data waveform in the partial-waveform display area is not updated by this operation alone. Click on the [Redraw] or [Show] button or scroll the partial-waveform display area to redisplay the waveform.

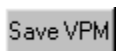
**[SavPCM] button**

Saves the compressed data to a 16-bit PCM file.

**[Open VPM] button**

Loads compression parameters from a specified VPM file.

The loaded parameter values are displayed on each VOX parameter control.

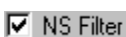
**[Save VPM] button**

Saves the compression parameters that have been set by VOX parameter controls to a VPM file.

**[Noise EMU] combo box**

The default selection of this combo box is "Normal", which processes the unconverted 8-kHz-sampling source voice data. Choices "8bit", "Light", and "Heavy" are provided to produce sound quality approximately equal to that of the E0C33 chip by emulating the 8-bit D/A converter output from the E0C33 chip.

If you choose "8bit", 10-bit data is converted into 8-bit form before being output. If you choose "Light" or "Heavy", the quantization noise in D/A converter output from the E0C33 chip is emulated. As a result, high tones are enhanced when sound is output. "Light" and "Heavy" respectively reduce or increase this effect. Note that these functions require a large amount of memory.

**[NS Filter] check box**

If you choose this option, the noise-shaving filter is enabled. This filter helps to reduce noise at high frequencies.

For details, refer to Section 4.4, "VOX Parameters".

**[Avr] box**

The average voice data rate after compression/expansion is displayed here.



VOX parameters

Use these controls to set each VOX parameter. For details, refer to Section 4.4, "VOX Parameters".

Basic operation procedure

1. Using the [Open] button, enter the 16-bit PCM file (.pcm) to be compressed.
For microphone input, set the recording time (seconds) in [Time] and the input level in [Gain] and click on the [Listen] button to enter voice data from a microphone.
The entered voice data can be reproduced using the [Speak] button.
2. Using the [Open VPM] button, enter the VOX parameter file (.vpm) to be referenced during compression.
The "voxtool\param\" directory contains a VOX parameter file for which various parameters have already been set.
3. Correct each parameter as necessary while referring to Section 4.4, "VOX Parameters".
4. Use the [SynthSpk] button to reproduce the compressed voice.
5. Repeat steps 3 and 4 until you reach a good balance between compression ratio and sound quality.
For the compression ratio, refer to the average voice data rate shown in the [Avr] box.
6. Use the [Save VPM] button to save the required VOX parameters to a file.
7. Save the compressed voice data using the [SavPCM] button.

Note:

- Evaluate the VOX parameters using the actual application system before making the final decision.
- The VOX voice files handled by "voxparam.exe" are not compatible with the VOX2 voice files handled by "vox2parm.exe" and "vox2cmp.exe". The VOX2 voice data has better sound quality, but on the E0C33 chip it can only be reproduced. VOX voice data can be compressed and recorded, as well as reproduced, on the E0C33 chip.

4.3.3 adpparam.exe

This tool is used to evaluate ADPCM-compressed voice data after entering voice data from a 16-bit PCM file or a microphone.

It can reproduce the original data and the compressed voice data, allowing you to change the compression ratio while listening. The compressed voice data can be saved to a 16-bit PCM file for use as voice ROM data by "pcm2s.exe".

Starting and quitting



Adpparam.exe

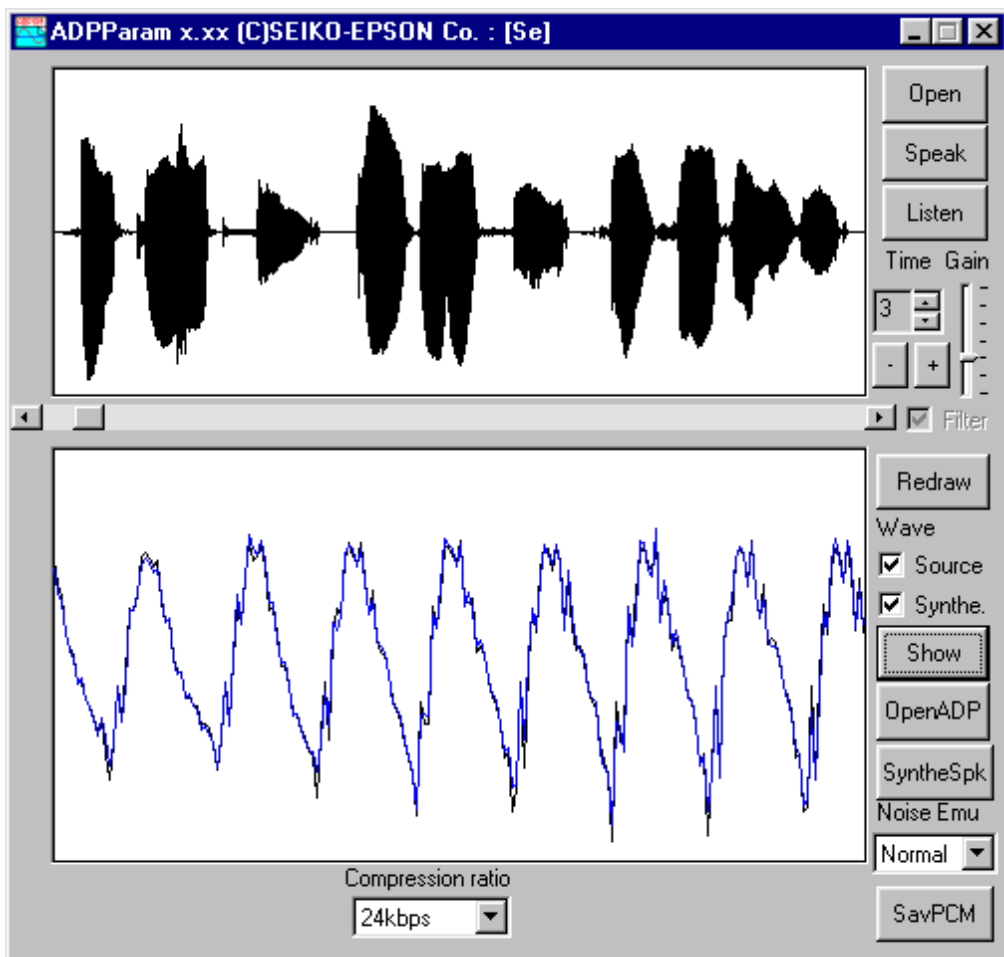
Double-click on the "adpparam.exe" icon to start the tool.

To quit "adpparam.exe", click on the [Close] button at the upper right corner of the [ADPParam] window.

Windows and the function of each part

[ADPParam] window

When "adpparam.exe" starts up, the [ADPParam] window appears. To perform any operation from this window, click on the appropriate control button using the mouse.



Controls for waveform display

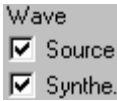
- Upper window:** **Full-waveform display area**
 Displays the full waveform of the source voice data loaded from a file or entered from a microphone.
- Lower window:** **Partial-waveform display area**
 Displays a partial-waveform of the source voice data or compressed voice data. Use the [Wave] check box to choose to display one or both of the source and compressed waveforms. The source voice waveform is displayed in black and the compressed data is displayed in blue. The waveforms displayed in this window can be scrolled using the scroll bar located above the window.



[Redraw] button
 Redraws the waveforms in both waveform display areas.

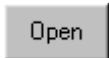


[Show] button
 Redraws the waveform in the partial-waveform display area.

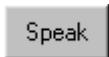


[Wave] check box
 Used to choose the waveform to be displayed in the partial-waveform display area. If you choose [Source], the source voice waveform is displayed in black. If you choose [Synthe.], the compressed data waveform is displayed in blue. You also can choose both [Synthe.] and [Source]. Note that the partial-waveform display area is not updated by this operation alone. After selecting or deselecting [Synthe.] or [Source], click on the [Redraw] or [Show] button or scroll the partial-waveform display area to redisplay the waveform.

Controls for source voice



[Open] button
 Loads a 16-bit PCM file. The waveform of the loaded voice data is displayed in the waveform display area.



[Speak] button
 Outputs the source voice data to the speaker via a sound card.



[Listen] button
 Enters voice data from a microphone. The waveform of the entered voice data is displayed in the waveform display area.



[Time]
 When entering voice data from a microphone, set the input time here in units of seconds. The input time can be set in the range of 1 to 60 seconds. When this set time has elapsed after the [Listen] button is clicked, the voice input stops. The voice input cannot be stopped before the set time has elapsed.



[Gain]
 When entering voice data from a microphone, set the input level here.



[+] button, [-] button
 Increments ([+] button) or decrements ([-] button) the amplitude of the input data waveform in steps of 1/8. The waveforms displayed in the full-and partial-waveform display areas are updated when you click on these buttons. The sound volume during playback also changes.




[Filter] check box
 Turns on or off the filter function that cuts noise when entering voice from a microphone. In the current version, however, it is set to on and you cannot choose to turn it off.


[OpenADP] button

Loads an ADPCM-compressed voice file. Always be sure to choose an ADPCM voice file that has been generated by "adpcmprs.exe".

The waveform of the loaded voice data is displayed in the waveform display area.

When an ADPCM file is loaded, the expanded waveform is displayed in the waveform display area, and the waveform to be displayed in the partial-waveform display area cannot be selected. In the [Wave] check box, you only can choose [Synthe.].

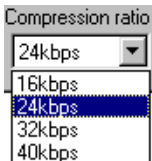
Controls for compression

[SyntheSpk] button

Compresses the source voice according to the parameters set and outputs the result from a speaker.

Note that the compressed data waveform in the partial-waveform display area is not updated by this operation alone. Click on the [Redraw] or [Show] button or scroll the partial-waveform display area to redisplay the waveform.

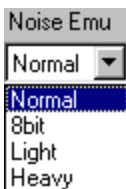

[SavPCM] button

Saves the compressed data to a 16-bit PCM file.

**[Compression ratio] combo box**

Used to choose an ADPCM compression ratio. Here, one of the following compression ratios can be selected:

- 16 kbps
- 24 kbps (default)
- 32 kbps
- 40 kbps

**[Noise EMU] combo box**

The default selection of this combo box is "Normal", which processes the unconverted 8-kHz-sampling source voice data.

Choices "8bit", "Light", and "Heavy" are provided to produce sound quality approximately equal to that of the E0C33 chip by emulating the 8-bit D/A converter output from the E0C33 chip.

If you choose "8bit", 10-bit data is converted into 8-bit form before being output.

If you choose "Light" or "Heavy", the quantization noise in D/A converter output from the E0C33 chip is emulated. As a result, high tones are enhanced when sound is output. "Light" and "Heavy" respectively reduce or increase this effect. Note that these functions require a large amount of memory.

Basic operation procedure

1. Using the [Open] button, enter the 16-bit PCM file (.pcm) to be compressed.
For microphone input, set the recording time (seconds) in [Time] and the input level in [Gain] and click on the [Listen] button to enter voice data from a microphone.
The entered voice data can be reproduced using the [Speak] button.
2. Choose the desired compression ratio from the [Compression ratio] combo box.
3. Use the [SyntheSpk] button to reproduce the compressed voice data.
4. The compressed voice data can be saved using the [SavPCM] button.

4.3.4 vsxparam.exe

This tool is used to evaluate VSX-compressed voice data after entering voice data from a 16-bit PCM file or a microphone.

It can reproduce the original data and the compressed voice data, allowing you to change the compression ratio while listening. The compressed voice data can be saved to a 16-bit PCM file for use as voice ROM data by "pcm2s.exe".

Starting and quitting



Vsxparam.exe

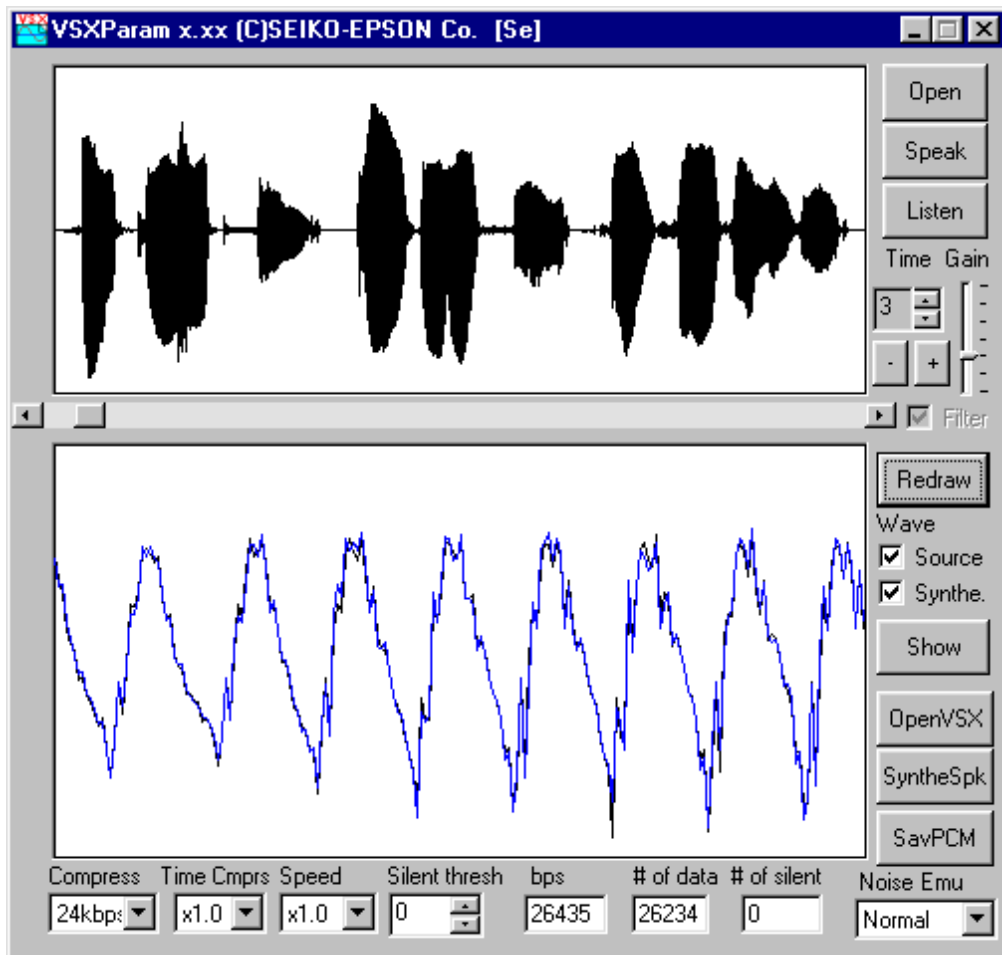
Double-click on the "vsxparam.exe" icon to start the tool.

To quit "vsxparam.exe", click on the [Close] button at the upper right corner of the [VSXParam] window.

Windows and the function of each part

[VSXParam] window

When "vsxparam.exe" starts up, the [VSXParam] window appears. To perform any operation from this window, click on the appropriate control button using the mouse.

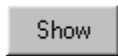


Controls for waveform display

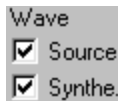
- Upper window:** **Full-waveform display area**
Displays the full waveform of the source voice data loaded from a file or entered from a microphone.
- Lower window:** **Partial-waveform display area**
Displays a partial-waveform of the source voice data or compressed voice data. Use the [Wave] check box to choose to display one or both of the source and compressed waveforms. The source voice waveform is displayed in black and the compressed data is displayed in blue. The waveforms displayed in this window can be scrolled using the scroll bar located above the window.

**[Redraw] button**

Redraws the waveforms in both waveform display areas.

**[Show] button**

Redraws the waveform in the partial-waveform display area.

**[Wave] check box**

Used to choose the waveform to be displayed in the partial-waveform display area.

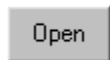
If you choose [Source], the source voice waveform is displayed in black.

If you choose [Synthe.], the compressed data waveform is displayed in blue.

You also can choose both [Synthe.] and [Source].

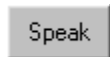
Note that the partial-waveform display area is not updated by this operation alone. After

selecting or deselecting [Synthe.] or [Source], click on the [Redraw] or [Show] button or scroll the partial-waveform display area to redisplay the waveform.

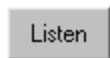
Controls for source voice**[Open] button**

Loads a 16-bit PCM file.

The waveform of the loaded voice data is displayed in the waveform display area.

**[Speak] button**

Outputs the source voice data to the speaker via a sound card.

**[Listen] button**

Enters voice data from a microphone.

The waveform of the entered voice data is displayed in the waveform display area.

**[Time]**

When entering voice data from a microphone, set the input time in units of seconds here.

The input time can be set in the range of 1 to 60 seconds. When this set time has elapsed after the [Listen] button is clicked, the voice input stops. The voice input cannot be stopped before the set time has elapsed.

**[Gain]**

When entering voice data from a microphone, set the input level here.

**[+] button, [-] button**

Increments ([+] button) or decrements ([-] button) the amplitude of the input data waveform in steps of 1/8. The waveforms displayed in the full-and partial-waveform display areas are updated when you click on these buttons. The sound volume during playback also changes.

**[Filter] check box**

Turns on or off the filter function that cuts noise when entering voice from a microphone.

In the current version, however, it is set to on and you cannot choose to turn it off.



[OpenVSX] button

Loads a VSX-compressed voice file. Always be sure to choose a VSX voice file that has been generated by "vsxcmprs.exe".

The waveform of the loaded voice data is displayed in the waveform display area.

When a VSX voice file is loaded, the expanded waveform is displayed in the waveform display area, and the waveform to be displayed in the partial-waveform display area cannot be selected.

In the [Wave] check box, you only can choose [Synthe.].

Controls for compression



[SyntheSpk] button

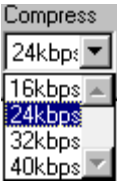
Compresses the source voice data according to the parameters set and outputs the result from a speaker.

Note that the compressed data waveform in the partial-waveform display area is not updated by this operation alone. Click on the [Redraw] or [Show] button or scroll the partial-waveform display area to redisplay the waveform.



[SavPCM] button

Saves the compressed data to a 16-bit PCM file.



[Compress] combo box

Used to choose an ADPCM-compatible compression ratio.

Here, one of the following compression ratios can be selected:

- 16 kbps
- 24 kbps (default)
- 32 kbps
- 40 kbps

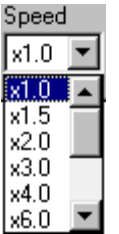


[Time Cmprs] combo box

Used to choose a compression ratio in the timebase direction.

Here, one of the following compression ratios can be selected:

- ×1.0 (same as the source voice; default)
- ×2.0 (same effect as recording at 2 times normal speed)
- ×3.0 (same effect as recording at 3 times normal speed)
- ×4.0 (same effect as recording at 4 times normal speed)



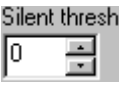
[Speed] combo box

Chooses the speed of speech.

Here, one of the following speeds can be selected:

- ×1.0 (same as the source voice; default)*
- ×1.5 (speed converted to 1.5 times that of source voice)*
- ×2.0 (speed converted to 2 times that of source voice)*
- ×3.0 (speed converted to 3 times that of source voice)
- ×4.0 (speed converted to 4 times that of source voice)
- ×6.0 (speed converted to 6 times that of source voice)
- ×8.0 (speed converted to 8 times that of source voice)
- ×16.0 (speed converted to 16 times that of source voice)
- ×1/1.5 (speed converted to 1/1.5 times that of source voice)*
- ×1/2.0 (speed converted to 1/2 times that of source voice)

Note: Conversion on the E0C33 chip is subject to limitations on the parameters that can be selected. (Seiko Epson recommends using only the parameters marked by *.)



[Silent thresh] edit box

To further compress silent parts of voice data, set the threshold level at which it is judged that there is no sound. The greater the value of this threshold, the higher the compression ratio, but the lower the sound quality. Normally, set this threshold in the range of 0 to 50.

[bps] text box

Shows average voice data rates after compression/expansion.

[# of data] text box

Shows the number of data loaded from the source data in units of short size.

[# of silent] text box

Shows the number of data judged to be silent when any value other than 0 was set in [Silent thresh].

Noise Emu

- Normal
- 8bit
- Light
- Heavy

[Noise EMU] combo box

The default selection of this combo box is "Normal", which processes the unconverted 8-kHz-sampling source voice data.

Choices "8bit", "Light", and "Heavy" are provided to produce sound quality approximately equal to that of the E0C33 chip by emulating the 8-bit D/A converter output from the E0C33 chip.

If you choose "8bit", 10-bit data is converted into 8-bit form before being output.

If you choose "Light" or "Heavy", the quantization noise in D/A converter output from the E0C33 chip is emulated. As a result, high tones are enhanced when sound is output. "Light" and "Heavy" respectively reduce or increase this effect.

Note that these functions require a large amount of memory.

Basic operation procedure

1. Using the [Open] button, enter the 16-bit PCM file (.pcm) to be compressed.
For microphone input, set the recording time (seconds) in [Time] and the input level in [Gain] and click on the [Listen] button to enter voice data from a microphone.
The entered voice data can be reproduced using the [Speak] button.
2. Choose compression parameters and playback speed from the corresponding boxes: [Compress], [Time cmprs], [Silent Thresh], and [Speed].
3. Use the [SyntheSpk] button to reproduce the compressed voice data.
4. The compressed voice data can be saved using the [SavPCM] button.

4.3.5 vsctparam.exe

This tool is used to evaluate the talking speed and tone pitch conversions by VSC after entering voice data from a 16-bit PCM file or a microphone.

It can reproduce the original data and the compressed voice data, allowing you to change the talking speed and tone pitch while listening. The converted voice data can be saved to a 16-bit PCM file for use as voice ROM data by "pcm2s.exe".

Starting and quitting



Vscparam.exe

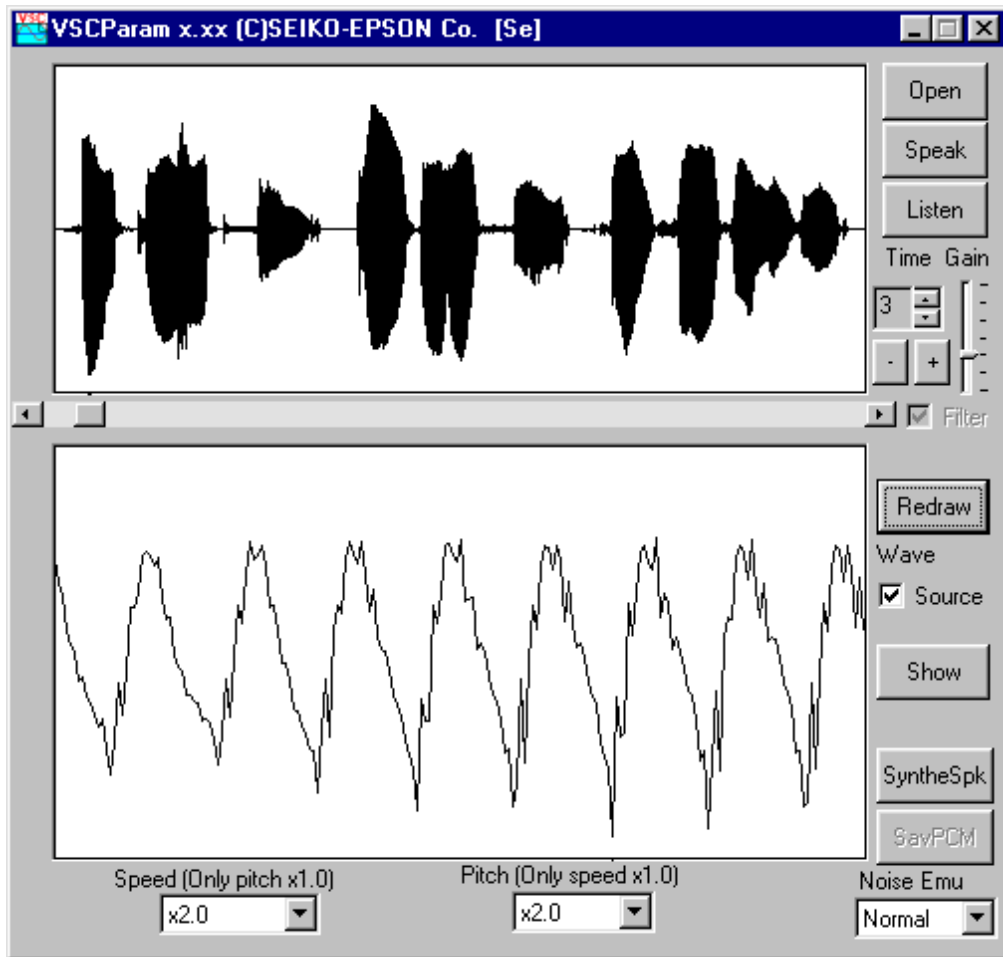
Double-click on the "vsctparam.exe" icon to start the tool.

To quit "vsctparam.exe", click on the [Close] button at the upper right corner of the [VSCParam] window.

Windows and the function of each part

[VSCParam] window

When "vsctparam.exe" starts up, the [VSCParam] window appears. To perform any operation from this window, click on the appropriate control button using the mouse.

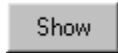


Controls for waveform display

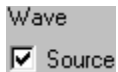
- Upper window:** **Full-waveform display area**
Displays the full waveform of the source voice data loaded from a file or entered from a microphone.
- Lower window:** **Partial-waveform display area**
Displays the waveform of the source voice data loaded from a file or entered from a microphone after enlarging it along the time base.
The waveform in this window can be scrolled using the scroll bar located above the window.

**[Redraw] button**

Redraws the waveforms in both waveform display areas.

**[Show] button**

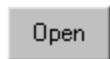
Redraws the waveform in the partial-waveform display area.

**[Wave] check box**

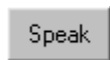
Used to turn the partial-waveform display area on or off.

If you choose [Source], the source voice waveform is displayed in black.

Note that the partial-waveform display area is not updated by this operation alone. After selecting or deselecting [Source], click on the [Redraw] or [Show] button or scroll the partial-waveform display area to redisplay the waveform.

Controls for source voice**[Open] button**

Loads a 16-bit PCM file. The waveform of the loaded voice data is displayed in the waveform display area.

**[Speak] button**

Outputs the source voice data to the speaker via a sound card.

**[Listen] button**

Enters voice data from a microphone.

The waveform of the entered voice data is displayed in the waveform display area.

**[Time]**

When entering voice data from a microphone, set the input time in units of seconds here.

The input time can be set in the range of 1 to 60 seconds. When this set time has elapsed after the [Listen] button is clicked, the voice input stops. The voice input cannot be stopped before the set time has elapsed.

**[Gain]**

When entering voice data from a microphone, set the input level here.

**[+] button, [-] button**

Increments ([+] button) or decrements ([-] button) the amplitude of the input data waveform in steps of 1/8. The waveforms displayed in the full-and partial-waveform display areas are updated when you click on these buttons.

The sound volume during playback also changes.

**[Filter] check box**

Turns on or off the filter function that cuts noise when entering voice from a microphone.

In the current version, however, it is set to on and you cannot choose to turn it off.

Controls for talking speed/tone pitch adjustments



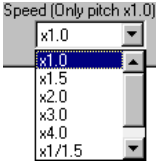
[SyntheSpk] button

Converts the source voice according to the talking speed and tone pitch that have been set and outputs the result from a speaker.



[SavPCM] button

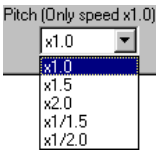
Saves the VSC-converted data to a 16-bit PCM file.



[Speed] combo box

Chooses the speed of speech. Here, one of the following speeds can be selected:

- ×1.0 (same as the source voice; default)
- ×1.5 (speed converted to 1.5 times that of source voice)
- ×2.0 (speed converted to 2 times that of source voice)
- ×1/1.5 (speed converted to 1/1.5 times that of source voice)
- ×1/2.0 (speed converted to 1/2 times that of source voice)

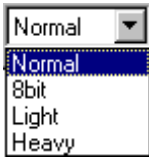


[Pitch] combo box

Used to set the parameter according to which the tone pitch is converted. One of the following parameters can be selected:

- ×1.0 (same as the source voice; default)
- ×1.5 (pitch converted to 1.5 times that of source voice)
- ×2.0 (pitch converted to 2 times that of source voice)
- ×1/1.5 (pitch converted to 1/1.5 times that of source voice)
- ×1/2.0 (pitch converted to 1/2 times that of source voice)

Note: If any value other than "×1.0" is selected for [Speed], [Pitch] is automatically reset to "×1.0".



[Noise] combo box

The default selection of this combo box is "Normal", which processes the unconverted 8-kHz-sampling source voice data.

Choices "8bit", "Light", and "Heavy" are provided to produce sound quality approximately equal to that of the E0C33 chip by emulating the 8-bit D/A converter output from the E0C33 chip.

If you choose "8bit", 10-bit data is converted into 8-bit form before being output.

If you choose "Light" or "Heavy", the quantization noise in D/A converter output from the E0C33 chip is emulated. As a result, high tones are enhanced when sound is output. "Light" and "Heavy" respectively reduce or increase this effect. Note that these functions require a large amount of memory.

Basic operation procedure

1. Using the [Open] button, enter the 16-bit PCM file (.pcm) in which you are going to convert the talking speed and/or tone pitch.
For microphone input, set the recording time (seconds) in [Time] and the input level in [Gain] and click on the [Listen] button to enter voice data from a microphone.
The entered voice can be reproduced using the [Speak] button.
2. Choose the desired speed or pitch from the [Speed] or [Pitch and Speed] combo box.
3. Use the [SyntheSpk] button to reproduce the speed/pitch-converted voice.
4. The converted voice data can be saved to a 16-bit PCM file using the [SavPCM] button.

4.4 VOX Parameters

The VOX parameter file (.vpm) is a binary file that contains a record of various settings required for VOX2 expansion and VOX compression/expansion, and is used for voice data compression by "vox2cmp.exe" and "voxcmprs.exe".

Also, this file is used for VOX2 expansion and VOX compression/expansion on the E0C33 chip. In this case, the necessary parameter file must be converted into an assembly resource file by "bin2s.exe" and linked to the user program before it can be used.

The VOX parameter file is created by "vox2parm.exe" or "boxparam.exe". The "voxtool\param\" directory includes parameter files that have had various settings already input, so you can use them unchanged or by correcting them after loading into "vox2parm.exe" or "voxparam.exe".

4.4.1 Function of Each VOX Parameter

VOX parameters must be set in the "vox2parm.exe" or "voxparam.exe" window.

When a VOX parameter is changed, the sound quality and compression ratio change. The content of each parameter is explained below.

Depth

This parameter affects the sound quality and compression ratio more than any other parameter.

When the value of Depth increases, although the sound quality does not deteriorate, the amount of created data increases significantly and the compression ratio drops. In VOX compression, when the value of Depth increments by 1, the amount of data is nearly doubled and the sound quality improves by one rank. In VOX2 compression, the amount of data increases in proportion to the value of Depth.

Choose the desired value of Depth with respect to the compression ratio.

For VOX

Depth = 4 when compression ratio = 16 kbps or more

Depth = 3 when compression ratio = 7 to 16 kbps

Depth = 2 when compression ratio = 3 to 7 kbps

Depth = 1 when compression ratio = 3 kbps or less

Note: Although in "voxparam.exe" Depth can be selected in the range of 1 to 4, set Depth in the range of 1 to 3 if you are going to perform real-time compression on the E0C33 chip. If used for expansion only, Depth can also be set to 4.

For VOX2

Depth = 4 when compression ratio = 10 to 20 kbps

Depth = 3 when compression ratio = 8 to 15 kbps

Depth = 2 when compression ratio = 4 to 8 kbps

Note: In "vox2parm.exe" you cannot choose 1 for Depth.

Width

This parameter affects the sound quality and compression ratio more than any other parameter apart from Depth.

As the value of Width increases, so does the compression ratio. However, the number of processing steps per unit time decreases and the sound quality deteriorates accordingly. Normally, when the value of Width doubles, processing becomes twice as rough and the amount of data is nearly halved.

Since compression ratio changes due to small changes in width finely, adjust the compression ratio using Width after setting Depth. If the compression ratio cannot be fully adjusted using Width, reset Depth. Width can be set in the range of 4 to 8.

Height

Normally set this parameter to 10 (= default). Do not set a value greater than 10. When the value of Height is reduced, the compression ratio increases at a rate of approximately 1%. The compression ratio changes more rapidly (at a rate of about 2 to 3%) if Height = 7 to 8.

Note that for extremely high tones, the sound quality may increase if the value of Height is 7 rather than 10. For low-pitched sounds (male voice), the sound quality does not deteriorate significantly if the value of Height is reduced.

Weight

Normally set this parameter to 100 (= default).

Post

This parameter is used to reduce noise at high frequencies with a low-pass filter when using a high compression ratio. The value for Post can be set in the range of 1 to 99. The smaller the value, the greater the effect. However, the clarity of sound deteriorates accordingly.

This parameter has no effect on the compression ratio. By default, this parameter is turned on.

Mid

This parameter enables smooth transitions between processing steps. Use this parameter to reduce mechanical noise when using a high compression ratio.

The value for Mid can be set in the range of 1 to 99. The smaller the value, the greater the effect.

However, the clarity of sound deteriorates accordingly.

This parameter has no effect on the compression ratio. By default, this parameter is turned off.

Pre

This parameter emphasizes rapidly rising voices. When pronunciations of S-based vowel sounds are obscured by Mid and Post when Depth = 2 or so, use this parameter to increase the clarity of pronunciation.

The value for Pre can be set in the range of 1 to 99. The smaller the value, the greater the effect.

This parameter affects the compression ratio. When Pre is turned on, the amount of data decreases by approximately 10%. By default, this parameter is turned off.

Standard ranges of settings for Pre, Mid, and Post are shown below.

Depth = 4: Pre = Off, Mid = Off to 75, Post = Off

Depth = 3: Pre = Off, Mid = 75, Post = 75

Depth = 2: Pre = Off, Mid = 75, Post = 75 (Set Pre to 75 to improve the clarity of S-based vowel sounds.)

Depth = 1: Pre = Off, Mid = 75, Post = 75 to 50 (Set Pre to 75 to improve the clarity of S-based vowel sounds.)

NS Filter

This filter reduces noise at high frequencies by shifting the noise component in the high frequency region to the mid-frequency region. This may result in increased clarity of the compressed voice. Determine whether the clarity is increased or not by listening. When applying this filtering, turn Mid off. Seiko Epson recommends turning Post off too, or setting it to a value between 50 and 75.

This parameter has no effect on the compression ratio. By default, this parameter is turned on.

Level-2

Normally set this parameter to 100 (= default). The smaller this value, the higher the compression ratio, but the rate of increase is only a few percent. As the value of this parameter decreases, so does the sound quality.

Level-3

Normally set this parameter to 0 (= default). This parameter controls the threshold at which sound parts of voice data are discriminated from silent parts. The greater this value, the higher the compression ratio, but the speech becomes increasingly truncated towards the end. Although this truncation effect is weakened by lowering Level-3, the compression ratio also drops by about 10 to 20%. In VOX compression, low-power parts of voice data may be inadvertently treated as silent parts. To avoid this problem, lowering Level-3 may prove effective.

4.4.2 VOX Parameter Samples

The "voxtool\param\" directory contains sample VOX parameter files.

Load one of these sample files into "vox2parm.exe" or "voxparam.exe" and use it as a template for compression evaluation.

Note that all VOX parameters have been set for use in the voice sample file "se.pcm" stored in the "voxtool\sample\" directory. When these parameters are applied directly to the voice data used in your application system, the intended effect may not be obtained. Therefore, correct them as necessary. When you have corrected parameters, save them under another name using the [SavPCM] button.

The table below shows how VOX parameters in each sample file are set. All parameters not shown here are set to the default values. The compression ratios shown here are for "se.pcm". Use them for reference purposes only.

Table 4.4.1 Setup Contents of Sample VOX Parameter Files (*.vpm)

File name	Depth	Widht	Pre	Mid	Post	Approx. compression rate
d4w4N.vpm	4	4	Off	Off	75	20.9 kbps
d4w7N.vpm	4	7	Off	Off	75	12.4 kbps
d3w7N.vpm	3	7	Off	Off	75	8.9 kbps
d2w5N.vpm	2	5	Off	Off	75	6.3 kbps

5 VOX33 Library Reference

This section describes the precautions to be observed when using VOX33 library functions and explains each function in detail.

5.1 Outline of VOX33 Library

Functional outline

The VOX33 library consists of a set of voice-processing functions in srf33 library format, and is used by linking it to the target program. By calling up the necessary functions from the target program, the following functions can be executed in real time:

- VSX data compression/recording and expansion/playback functions
- VOX data compression/recording and expansion/playback functions
- VOX2 data expansion/playback function
- ADPCM data compression/recording and expansion/playback functions
- PCM data recording and playback functions
- PCM, VOX, and VOX2 data VSC conversion (talking speed/tone pitch conversion) and playback functions

* The VSX, VOX, VOX2, and VSC conversions are Seiko Epson's original voice processing technologies.

Note: VOX data can be both recorded and reproduced. VOX2 surpasses VOX in sound quality, but is a playback-only format. Because VOX and VOX2 formats are not compatible, data compressed in one format cannot be reproduced by expanding it in the other format. When writing data in either format to the chip for playback-only purposes, Seiko Epson recommends using VOX2.

This software package also contains the C source of the top-level VOX33 library functions and the assembly source, which can be used for initialization purposes. These sources can be used by copying them, in whole or part, into the target program.

This set of functions helps you to easily implement voice-processing functions in your application system.

Program structure

The structure of a voice application program is shown in Figure 5.1.1.

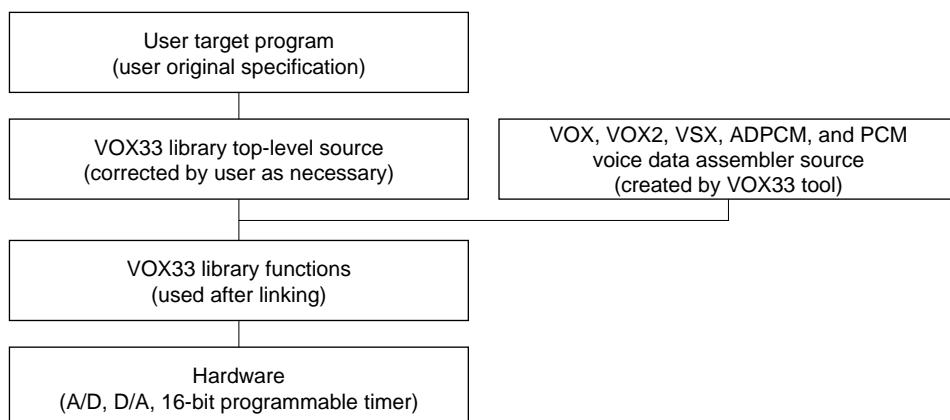


Figure 5.1.1 Program Structure

VOX33 library structure

The VOX33 library and all related files are provided in the "voxlib" folder (directory). The contents of the "voxlib" folder are listed below.

voxlib\ VOX33 library-related directory
readme.txt	VOX33 library supplementary explanation, etc. (in English)
readmeja.txt	VOX33 library supplementary explanation, etc. (in Japanese)
lib\ VOX33 library for E0C33A104 directory
vox.lib	VOX33 library for E0C33A104
sl104.lib	Voice input/output library for E0C33A104 (D/A version)
vox33asm.o, vox2asm.o, mesa.o, cpclrdat.o, fadpcm16.o, fadpcm24.o, fadpcm32.o, fadpcm40.o	Objects retrieved from vox.lib to accelerate operation
lib208\ VOX33 library for E0C33208 directory
vox208.lib	VOX33 library for E0C33208
sl208.lib	Voice input/output library for E0C33208 (PWM version)
vox33asm.o, vox2asm.o, mesa.o, cpclrdat.o, fadpcm16.o, fadpcm24.o, fadpcm32.o, fadpcm40.o	Objects retrieved from vox208.lib to accelerate operation
include\ VOX33 library function header file directory
voxcomn.h	Library common header file
adpcm.h	ADPCM header file
vsx.h	VSX header file
vox.h	VOX compression/expansion function header file
vsc.h	Talking speed/tone pitch conversion header file
packpcm.h	Packed PCM header file
speak.h	Output function header file
listen.h	Input function header file
lksym.h	Linker symbol header file
src\ Library source directory
voxcomn.c	Library common functions
slutil.c	SPEAK and LISTEN utility functions
vsxtop.c	VSX top-level functions
adptop.c	ADPCM top-level functions
voxtop.c	VOX top-level functions
vox2top.c	VOX2 top-level functions
ppctop.c	PCM top-level functions
hardsrc\ Hardware dependent source directory
Listen.s	Listen.o source (E0C33A104)
LisAD.s	LisAD.o source (E0C33A104)
Speak.s	Speak.o source E0C33A104)
SpkDA.s	SpkDA.o source (E0C33A104)
Lis208.s	Lis208.o source (E0C33208)
Lis208AD.s	Lis208AD.o source (E0C33208)
Spk208.s	Spk208.o source (E0C33208)
Spk208PW.s	Spk208PW.o source (E0C33208)
slintr.def	(Use these sources as references when you want to modify timer A/D or D/A channels and ports.)
sample\ DMT33004 sample program directory
smpl208\ DMT33005 sample program directory

(For details on the configuration of sample programs and how to use them, refer to "readme.txt" or "readmeja.txt" in "voxlib".)

* The structures of top-level functions and library functions are described later.

5.2 Hardware Resources and Initialization

Note: If you want to use other channels or ports for the playback/recording hardware resources described below, modify the relevant sources in the "hardsrc\" directory. Use the modified sources by linking them directly to, not by copying them into, the user program. For details on each hardware resource, refer to the Technical Manual for the model used in your application system.

Hardware resources used the by the VOX33 library (vox.lib) for the E0C33A104

The VOX33 library for the E0C33A104 uses the internal hardware resources listed below. Therefore, these resources cannot be utilized by the user target program.

Hardware resources used for voice reproduction (Speak)

- D/A converter (channel 0) and all control registers associated with it
- K53 port and all control registers associated with it
- 16-bit programmable timer (timer 5) and all control registers associated with it
- 16-bit programmable timer (timer 5-1) underflow interrupt

Make sure the SpkIntr0() function is set in the underflow interrupt vector address of the 16-bit programmable timer (timer 5-1). The interrupt level is set to 4 by the SpkOpen() function.

Example: `.word SpkIntr0 ; Vector No. 50 (16-bit timer #5-1 underflow)`

Hardware resources used for voice recording (Listen)

- A/D converter (channel 0) and all control registers associated with it
- K60 port and all control registers associated with it
- 16-bit programmable timer (timer 0) and all control registers associated with it
- A/D conversion-completed interrupt

Make sure the LisIntr0() function is set in the A/D conversion-completed interrupt vector address. The interrupt level is set to 4 by the LisOpen() function.

Example: `.word LisIntr0 ; Vector No.64 (ADC)`

Operating clock

The VOX33 library assumes that the high-speed (OSC3) clock frequency used for the E0C33A104 is 20 MHz (typ.).

Hardware resources used by the VOX33 library (vox208.lib) for the E0C33208

The VOX33 library for the E0C33208 uses the internal hardware resources listed below. Therefore, these resources cannot be utilized by the user target program.

Hardware resources used for voice reproduction (Speak)

- 16-bit programmable timer (timer 1) and all control registers associated with it (used for PWM output)
- P23 port and all control registers associated with it
- 16-bit programmable timer (timer 5) and all control registers associated with it
- 16-bit programmable timer (timer 5) compare B interrupt

Make sure the SpkIntr0() function is set in the compare B interrupt vector address for the 16-bit programmable timer (timer 5). The interrupt level is set to 4 by the SpkOpen() function.

Example: `.word SpkIntr0 ; Vector No. 50 (16-bit timer #5 compare B)`

Hardware resources used for voice recording (Listen)

- A/D converter (channel 0) and all control registers associated with it
- K60 port and all control registers associated with it
- 16-bit programmable timer (timer 0) and all control registers associated with it
- A/D conversion-completed interrupt

Make sure the LisIntr0() function is set in the A/D conversion-completed interrupt vector address. The interrupt level is set to 4 by the LisOpen() function.

Example: `.word LisIntr0 ; Vector No. 64 (ADC)`

Operating clock

The VOX33 library assumes that the high-speed (OSC3) clock frequency used for the E0C33208 is 20 MHz (typ.), and that PLL is in x2 mode.

Memory

The memory requirements for real-time voice processing are as follows:

- Make sure all of the BSS sections used by the VOX33 library are mapped into the internal RAM.
- Be sure to use the internal RAM for the stack.
- When mapping VOX33 library program code into an external memory area, make sure this area is accessed in 2 wait cycles or less, if possible. Also, be sure to use a memory area 16 bits wide for this external area.

5.3 Top-Level Functions

The top-level functions are C sources that are provided to realize each required capability easily. They are implemented using VOX33 library functions. Table 5.3.1 below lists the functions in each source.

Table 5.3.1 Top-Level Functions

Source file	Function name	Description
vsxtop.c (VSX processing)	unsigned char *vsxSpeak()	Expands, speed-converts, and reproduces VSX data
	unsigned char *vsxListen()	Compresses and records VSX data
	void vsxTopDecode()	Call-back function for reproducing
	void vsxTopEncode()	Call-back function for recording
	void vsxTopEncodeEnd()	Call-back function for completion of recording
adptop.c (ADPCM processing)	unsigned char *adpcmSpeak()	Expands and reproduces ADPCM data
	unsigned char *adpcmListen()	Compresses and records ADPCM data
	void adpTopDecode()	Call-back function for reproducing
	void adpTopEncode()	Call-back function for recording
	void adpTopEncodeEnd()	Call-back function for completion of recording
vox2top.c (VOX2 processing)	unsigned char *vox2Speak()	Expands, VSC-converts, and reproduces VOX2 data
	void vox2TopDecode()	Call-back function for reproducing
voxtop.c (VOX processing)	unsigned char *voxSpeak()	Expands, VSC-converts, and reproduces VOX data
	unsigned char *voxListen()	Compresses and records VOX data
	void voxTopDecode()	Call-back function for reproducing
	void voxTopEncode()	Call-back function for recording
	void voxTopEncodeEnd()	Call-back function for completion of recording
ppctop.c (PCM processing)	unsigned char *ppc2Speak()	VSC-converts and reproduces PCM data
	unsigned char ppcListen()	Records PCM data
	void ppcTopDecode()	Call-back function for reproducing
	void ppcTopEncode()	Call-back function for recording
	void ppcTopEncodeEnd()	Call-back function for completion of recording
voxcomn.c (Common functions)	void voxCodecpy()	Copies code section
	void adpcmCodecpy()	Copies ADPCM code section
slutil.c (Input/output data conversion)	void setSpeakVolume()	Sets output volume
	void slPcm2Spk()	Converts output data
	void slIs2Pcm()	Converts input data

These functions can be used by copying the sources of the necessary functions from the above files and pasting them into the user program source. At this time, be sure to copy parts defined by external variables from the source files along with said functions.

When using the source files directly by linking them to the user program, obtain the header file stored in the "include\" directory and include it in the user program.

Note: The VOX33 library functions use the CPU's R8 register. Therefore, when linking the VOX33 library, including the top-level functions to the user program, you cannot use the -gp option (optimization using global pointer/R8) of the instruction extender ext33.

5.3.1 Compile Options

When compiling the top-level function source files, the following compile options can be specified. Define the names of your desired options when compiling (using the `-D` option of `gcc33`).

HIGH_PASS_FILTER

Recording (Listen) functions allow the input voice data to be fed through a high-pass filter. To use this filter, define `HIGH_PASS_FILTER` when compiling.

Normally, Seiko Epson recommends using the high-pass filter.

IRAM_CACHE

To run the program at high speed by mapping it into the internal RAM, define `IRAM_CACHE` when compiling.

In this case, you also need to define the necessary commands in the linker command file. For details, refer to Section 5.5, "Techniques for Speeding Up Operation".

NO_VSC

This option eliminates the VSC function, in order to increase the processing speed and reduce the amount of memory used. This option is effective when compiling "voxtop.c", "vox2top.c", and "ppctop.c".

ADPCM

To use the ADPCM or VSX function, specify this option when compiling "voxcomm.c".

SPK_8BIT

Specify this option when using an 8-bit D/A converter to output voice data.

CLOCK40

Specify this option for 10-bit PWM output on the E0C33208. This type of output requires that the PLL be placed in x2 mode.

In all sample programs, the compile options are defined as shown below.

Sample programs for DMT33004 (E0C33A104)

- `HIGH_PASS_FILTER` Defined
- `IRAM_CACHE` Defined
- `NO_VSC` Undefined
- `ADPCM` Defined only in ADPCM and VSX sample programs
- `SPK_8BIT` Defined
- `CLOCK40` Undefined

Sample programs for DMT33005 (E0C33208)

- `HIGH_PASS_FILTER` Defined
- `IRAM_CACHE` Defined
- `NO_VSC` Undefined
- `ADPCM` Defined only in ADPCM and VSX sample programs
- `SPK_8BIT` Undefined
- `CLOCK40` Defined

5.3.2 External Variables

When using the top-level functions by copying each one into the user program individually, be sure to also copy the external variable definitions given at the beginning of each source file. These external variables are also required when calling up library functions directly without using the top-level functions.

Note: These external variables always need to be mapped into the internal RAM because they greatly affect the processing speed.

The main external variables are outlined below.

short SplisBuf[SPLIS_BUF_SIZE];

This buffer stores the input/output data used by library functions. Although the buffer size varies with the function used, do not change it to any value other than that defined in the source.

short SpkDecBuf[PACKET_SIZE];

This buffer stores the PCM data derived from decoding of compressed data. Do not change the buffer size to any value other than that defined in the source.

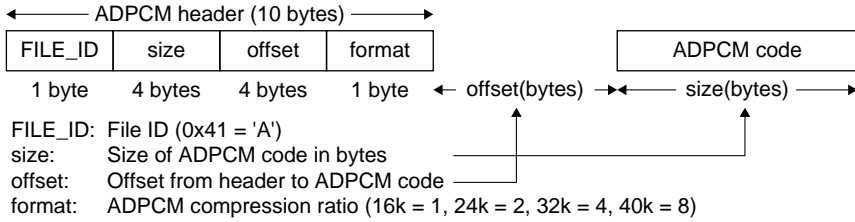
SiParam SIParam;

This is the data conversion parameter used for input/output devices.

5.3.3 Data Structure

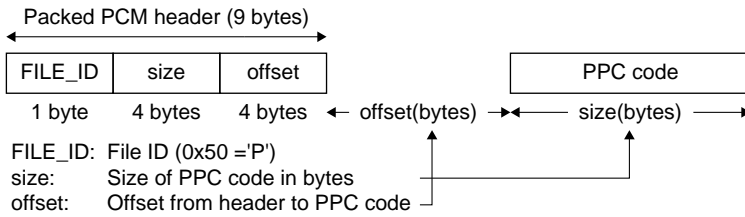
The following shows the data structure in each compression format.

ADPCM data



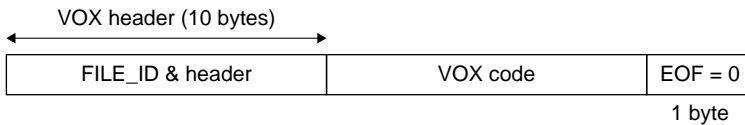
Note: For data exchanged with a PC, set *offset* to 0 so that the code follows immediately after the header.

Packed PCM data

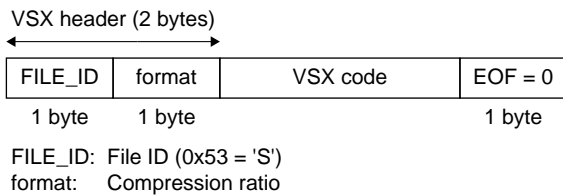


Note: For data exchanged with a PC, set *offset* to 0 so that the code follows immediately after the header.

VOX, VOX2 data



VSX data



5.3.4 Error Codes Returned by Top-Level Functions

Whether playback or recording executed using the top-level functions has been terminated normally or not can be determined by checking error codes that are stored in the error variables defined for each function.

- Error variables**
- vsxTopError
 - adpTopError
 - voxTopError
 - vox2TopError
 - ppcTopError

- Error codes**
- VOX_SUCCESS Terminated normally
 - VOX_BUF_FULL Specified data storage buffer filled when recording
 - VSC_ERROR Playback by VSC conversion failed

5.3.5 VSX Data Processing Functions (*vsxtop.c*)

Note: Because VSX data processing uses much CPU power, the program must be run at high speed by mapping part of it into the internal memory. The object required for internal memory mapping varies with the compression ratio involved, but is normally one of *fadpcm16.o*, *fadpcm24.o*, *fadpcm32.o*, or *fadpcm40.o*. For details, refer to Section 5.5, "Techniques for Speeding Up Operation".

vsxSpeak()

Function: Expands, speed-converts, and reproduces VSX data

Format: `unsigned char *vsxSpeak(unsigned char *Src, int Speed);`

Arguments: `unsigned char *Src` Playback VSX data pointer
`int Speed` Talking speed

Return values: Terminated normal.....SpkParams pointer
 Terminated abnormally0

Description: After loading VSX-format data from *Src*, this function expands and reproduces it. The talking speed can be changed by specifying any value for *Speed*. The values defined in "vsx.h" can be used. Normally, choose one of the following four:

```
/* play speed */
#define VSX_SPEED_SLOW15 (-1)   Slowed down by a factor of 1.5
#define VSX_SPEED_NORMAL (0)   Reproduced at the speed of the original data
#define VSX_SPEED_FAST15 (1)   Speeded up by a factor of 1.5
#define VSX_SPEED_FAST20 (2)   Speeded up by a factor of 2
```

Example: Reproducing the data input by *vsxListen()*

```
unsigned char *SpkParams, *LisParams, CmpData[8000*3];
LisParams = vsxListen(8000*3,8000*3,CmpData,
VSX_TIME_CMP_10|VSX_COMPRESS_24K,50);
:
SpkParams = vsxSpeak(CmpData, VSX_SPEED_FAST20);
```

The VSX data in *CmpData* is reproduced at 2 times the normal speed after expansion.

Reproducing the VSX data created by the user

Create the compressed voice assembly source data that is output by the VOX33 tool "bin2s.exe". For example, when you create the data labeled "sev24t1", specify the C source as shown below.

```
extern unsigned char sev24t1[];
:
unsigned char *SpkParams;
SpkParams = vsxSpeak(sev24t1, VSX_SPEED_NORMAL);
```

vsxListen()

Function: Compresses and records voice data in VSX format

Format: unsigned char *vsxListen(int Samples, int MaxBytes, unsigned char *Dst, unsigned char format, short silent_level);

Arguments:

int	Samples	Maximum number of input samples (sampling rate × seconds)
int	MaxBytes	Maximum number of written data (bytes)
unsigned char	*Dst	Pointer to the beginning of the data write buffer
unsigned char	format	Compression ratio
short	silent_level	Silent threshold

Return values: Terminated normally.....LisParams pointer
Terminated abnormally0

Description: This function compresses the input voice data in real time and writes the compressed voice data to the specified data buffer (Dst).

To specify the fourth argument (format), use logical OR of the "compression rate" and "time compression rate" defined in "vsx.h".

```

/* compression rate */           ADPCM-compatible compression ratio
#define VSX_COMPRESS_16K (1)     Compressed to 16 kbps equivalent
#define VSX_COMPRESS_24K (2)     Compressed to 24 kbps equivalent
#define VSX_COMPRESS_32K (4)     Compressed to 32 kbps equivalent
#define VSX_COMPRESS_40K (8)     Compressed to 40 kbps equivalent

/* time compression rate */      Compression ratio in timebase direction
#define VSX_TIME_CMP_10 (0x10)   Not compressed
#define VSX_TIME_CMP_20 (0x20)   Compressed as in ×2 recording
#define VSX_TIME_CMP_30 (0x30)   Compressed as in ×3 recording
#define VSX_TIME_CMP_40 (0x40)   Compressed as in ×4 recording
    
```

The greater the value of the fifth argument (silent_level), the higher the compression ratio, but the sound quality deteriorates accordingly. Normally, specify this argument in the range of 0 to 50.

Example:

```

unsigned char *LisParams, CmpData[8000*3];
LisParams = vsxListen(8000*3,8000*3,CmpData,
VSX_TIME_CMP_10|VSX_COMPRESS_24K,50);
    
```

Voice data is input for about 3 seconds while sampling at 8 kHz. The input voice data is compressed to 24 kbps equivalent and then stored in the buffer CmpData.

vsxTopDecode()

Function: Call-back function for reproducing

Format: `void vsxTopDecode(unsigned char *SpkParams, short *Buffer, int Length);`

Description: This function is called back from the library when free space is available in the playback data queue. It decodes data and adds Buffer to the queue. Enter this function using the SpkOnDone() function in TopSpeakStart(). (Refer to "vsxtop.c")

vsxTopEncode()

Function: Call-back function for recording

Format: `void vsxTopEncode(unsigned char *SpkParams, short *Buffer, int Length);`

Description: This function is called back from the library when data is stored in the record data buffer. It encodes the data stored in Buffer and adds Buffer to the queue. Enter this function using the LisOnDone() function in TopLisStart(). (Refer to "vsxtop.c")

vsxTopEncodeEnd()

Function: Call-back function for completion of recording

Format: `void vsxTopEncodeEnd(unsigned char *LisParams);`

Description: This function is called back from the library when recording is completed. Enter this function using the LisOnEmpty() function in TopLisStart(). (Refer to "vsxtop.c")

5.3.6 ADPCM Data Processing Functions (adptop.c)

Note: Because ADPCM data processing uses much CPU power, the program must be run at high speed by mapping part of it into the internal memory. The object required for internal memory mapping varies with the compression ratio involved, but is normally one of fadpcm16.o, fadpcm24.o, fadpcm32.o, or fadpcm40.o. For details, refer to Section 5.5, "Techniques for Speeding Up Operation".

adpcmSpeak()

Function: Expands and reproduces ADPCM data

Format: unsigned char *adpcmSpeak(unsigned char *adpcmData);

Argument: unsigned char *adpcmData Playback ADPCM data pointer

Return values: Terminated normally.....SpkParams pointer
Terminated abnormally0

Description: After loading ADPCM format data from adpcmData, this function expands and reproduces it.

Example: Reproducing the data input by adpcmListen()

```
unsigned char *SpkParams, *LisParams
unsigned char CmpData[8000*3+ADPCM_HEADER_SIZE];
LisParams = adpcmListen(8000*3, 8000*3, CmpData+ADPCM_HEADER_SIZE,
ADPCM_24K, CmpData);
:
SpkParams = adpcmSpeak(CmpData);
```

The ADPCM data in CmpData is reproduced after expansion.

Reproducing the PCM data created by the user

Create the compressed voice assembly source data that is output by the VOX33 tool "bin2s.exe". For example, when you have created the data labeled "sea16", specify the C source as shown below.

```
extern unsigned char sea16[];
:
unsigned char *SpkParams;
SpkParams = adpcmSpeak(sea16);
```


adpcmListen()

Function: Compresses and records voice data in ADPCM format

Format: unsigned char *adpcmListen(int Samples, int MaxBytes,
unsigned char *Dst, unsigned char format, unsigned char *Header);

Arguments:

int	Samples	Maximum number of input samples (sampling rate × seconds)
int	MaxBytes	Maximum number of written data (bytes)
unsigned char	*Dst	Pointer to the beginning of the data write buffer
unsigned char	format	Compression ratio
unsigned char	*Header	Buffer in which to write header information (10 bytes or more)

Return values: Terminated normally.....LisParams pointer
Terminated abnormally0

Description: This function compresses the input voice data in real time and writes the compressed voice data to a specified data buffer (Dst). The header information and ADPCM data can be saved separately.

For the fourth argument (format), specify one of the values defined in "adpcm.h".

```
#define ADPCM_16K (1)    Compressed to 16 kbps equivalent
#define ADPCM_24K (2)    Compressed to 24 kbps equivalent
#define ADPCM_32K (4)    Compressed to 32 kbps equivalent
#define ADPCM_40K (8)    Compressed to 40 kbps equivalent
```

Example:

```
unsigned char *LisParams, CmpData[8000*3+ADPCM_HEADER_SIZE];
LisParams = adpcmListen(8000*3, 8000*3, CmpData+ADPCM_HEADER_SIZE,
ADPCM_24K, CmpData);
```

Voice data is input for about 3 seconds while sampling at 8 kHz. The input voice is compressed to 24 kbps equivalent and then stored in the buffer CmpData.

adpTopDecode()

Function: Call-back function for reproducing

Format: `void adpTopDecode(unsigned char *SpkParams, short *Buffer, int Length);`

Description: This function is called back from the library when free space is available in the playback data queue. It decodes data and adds Buffer to the queue. Enter this function using the SpkOnDone() function in TopSpeakStart(). (Refer to "adptop.c")

adpTopEncode()

Function: Call-back function for recording

Format: `void adpTopEncode(unsigned char *SpkParams, short *Buffer, int Length);`

Description: This function is called back from the library when data is stored in the record data buffer. It encodes the data stored in Buffer and adds Buffer to the queue. Enter this function using the LisOnDone() function in TopLisStart(). (Refer to "adptop.c")

adpTopEncodeEnd()

Function: Call-back function for completion of recording

Format: `void adpTopEncodeEnd(unsigned char *LisParams);`

Description: This function is called back from the library when recording is completed. Enter this function using the LisOnEmpty() function in TopLisStart(). (Refer to "adptop.c")

5.3.7 VOX2 Data Processing Functions (*vox2top.c*)

Note: VOX2 data is playback-only and cannot be recorded. VOX format data cannot be reproduced by this function.

vox2Speak()

Function: Expands, VSC-converts, and reproduces VOX2 data

Format: `unsigned char *vox2Speak(unsigned char *vox2data, int pitch_speed, int real_time);`

Arguments:

<code>unsigned char *vox2data</code>	Playback VOX2 data pointer
<code>int pitch_speed</code>	Tone pitch and talking speed
<code>int real_time</code>	Real-time parameter

Return values: Terminated normally.....SpkParams pointer
Terminated abnormally0

Description: After loading VOX2-format compressed data from `vox2data`, this function expands and VSC-converts it in real time before outputting. The data created by "bin2s.exe" can be reproduced after changing the tone pitch and talking speed.

To specify the second argument (`pitch_speed`) and third argument (`real_time`), use the following macros defined in "vsc.h".

<code>VSC_NORMAL</code>	Standard
<code>VSC_SLOW20</code>	Talking speed twice as slow as standard
<code>VSC_SLOW15</code>	Talking speed 1.5 times as slow as standard
<code>VSC_FAST15</code>	Talking speed 1.5 times as fast as standard
<code>VSC_FAST20</code>	Talking speed twice as fast as standard
<code>VSC_LOW20</code>	Tone pitch twice as low as standard
<code>VSC_LOW15</code>	Tone pitch 1.5 times as low as standard
<code>VSC_HIGH15</code>	Tone pitch 1.5 times as high as standard
<code>VSC_HIGH20</code>	Tone pitch twice as high as standard

Example: Create the compressed voice assembly source data that is output by the VOX33 tool "bin2s.exe". For example, when you have created the data labeled "se37N", specify the C source as shown below.

```
extern unsigned char se37[];
:
unsigned char *SpkParams;
SpkParams = vox2Speak(se37N, VSC_SLOW20);
```

The data "se37N" is expanded and VSC-converted (talking speed is halved) before being reproduced.

vox2TopDecode()

Function: Call-back function for reproducing

Format: void vox2TopDecode(unsigned char *SpkParams, short *Buffer,
int Length);

Description: This function is called back from the library when free space is available in the playback data queue. It decodes data and adds Buffer to the queue. Enter this function using the SpkOnDone() function in TopSpeakStart(). (Refer to "vox2top.c".)

5.3.8 VOX Data Processing Functions (voxtop.c)

voxSpeak()

Function: Expands, VSC-converts, and reproduces VOX data

Format: `unsigned char* voxSpeak(unsigned char *voxData, int pitch_speed, int real_time);`

Arguments:

<code>unsigned char *voxData</code>	Playback VOX data pointer
<code>int pitch_speed</code>	Tone pitch and talking speed
<code>int real_time</code>	Real-time parameter

Return values: Terminated normally..... SpkParams pointer
Terminated abnormally..... 0

Description: After loading VOX-format compressed data from `voxData`, this function expands and VSC-converts it in real time before outputting. The data created by "bin2s.exe" or input by `voxListen()` can be reproduced after changing the tone pitch and talking speed.

To specify the second argument (`pitch_speed`) and third argument (`real_time`), use the following macros defined in "vsc.h".

<code>VSC_NORMAL</code>	Standard
<code>VSC_SLOW20</code>	Talking speed twice as slow as standard
<code>VSC_SLOW15</code>	Talking speed 1.5 times as slow as standard
<code>VSC_FAST15</code>	Talking speed 1.5 times as fast as standard
<code>VSC_FAST20</code>	Talking speed twice as fast as standard
<code>VSC_LOW20</code>	Tone pitch twice as low as standard
<code>VSC_LOW15</code>	Tone pitch 1.5 times as low as standard
<code>VSC_HIGH15</code>	Tone pitch 1.5 times as high as standard
<code>VSC_HIGH20</code>	Tone pitch twice as high as standard

Example: Reproducing the data input by `voxListen()`

```
unsigned char *SpkParams, *LisParams, CmpData[8000*3];
LisParams = voxListen(8000*3, 8000*3, CmpData, d3w8N);
:
SpkParams = voxSpeak(CmpData, VSC_NORMAL);
```

The VOX data in `CmpData` is expanded and reproduced without changing the tone pitch and talking speed.

Reproducing the PCM data created by the user

Create the compressed voice assembly source data that is output by the VOX33 tool "bin2s.exe". For example, when you have created the data labeled "se_v8", specify the C source as shown below.

```
extern unsigned char se_v8[];
:
unsigned char *SpkParams;
SpkParams = voxSpeak(se_v8, VSC_FAST20);
```

The data "se_v8" is expanded and VSC-converted (talking speed is doubled) before being reproduced.

Note: Data in the VOX2 format cannot be reproduced by this function.

voxListen()

Function: Compresses and records voice data in VOX format

Format: unsigned char *voxListen(int Samples, int MaxBytes,
unsigned char *Dst, unsigned char *Params);

Arguments:

int	Samples	Maximum number of input samples (sampling rate × seconds)
int	MaxBytes	Maximum number of written data (bytes)
unsigned char	*Dst	Pointer to the beginning of the data write buffer
unsigned char	*Params	VOX compression parameter pointer

Return values: Terminated normally.....LisParams pointer
Terminated abnormally.....0

Description: This function compresses the input voice data in real time and writes the compressed voice data to a specified data buffer (Dst).

To specify the fourth argument (Params), use the VPM file created by "voxparam.exe" that has been converted into an assembly source by "bin2s.exe". The "sample\vox\vpm.s" directory contains sample parameters, which can also be used. The usable parameter (Depth and Width) values are as follows:

Depth = 1	Width = 8	Approx. 2.3 kbps	(d1w8N)
Depth = 2	Width = 8	Approx. 4 kbps	(d2w8N)
Depth = 3	Width = 8	Approx. 7.5 kbps	(d3w8N)
Depth = 4	Width = 8	(Approx. 14 kbps,	d4w8N)

When operating at 20 MHz, do not use this parameter setting because it will prevent the CPU from processing data in real time.

For details on these parameters, refer to Section 4.4, "VOX Parameters".

Example:

```
unsigned char *LisParams, CmpData[8000*3];
LisParams = voxListen(8000*3, 8000*3, CmpData, d3w8N);
```

Voice data is input for about 3 seconds while sampling at 8 kHz. The input voice data is compressed using the VOX parameters specified by d3w8N and then stored in the buffer CmpData.

Note:

- VOX compression is significantly affected by DC level of the input signal, power supply noise, and the S/N ratio. These effects must be eliminated by input circuits on the board.
 1. Shield signal lines to prevent them picking up environmental noise in the range of 50–60 Hz.
 2. Prevent picking up wind noise from the microphone.
 3. Adjust the input level so that it will not be excessively low or exceed the tolerance level.
- Because this function uses much CPU power, the program must be run at high speed by mapping it into the internal memory. The objects required for internal memory mapping are "vox33asm.o", "mesa.o", and "cpclrdat.o".
For details, refer to Section 5.5, "Techniques for Speeding Up Operation".

voxTopDecode()

Function: Call-back function for reproducing

Format: `void voxTopDecode(unsigned char *SpkParams, short *Buffer, int Length);`

Description: This function is called back from the library when free space is available in the playback data queue. It decodes data and adds Buffer to the queue. Enter this function using the SpkOnDone() function in TopSpeakStart(). (Refer to "voxtop.c".)

voxTopEncode()

Function: Call-back function for recording

Format: `void voxTopEncode(unsigned char *SpkParams, short *Buffer, int Length);`

Description: This function is called back from the library when data is stored in the record data buffer. It encodes the data stored in Buffer and adds Buffer to the queue. Enter this function using the LisOnDone() function in TopLisStart(). (Refer to "voxtop.c".)

voxTopEncodeEnd()

Function: Call-back function for completion of recording

Format: `void voxTopEncodeEnd(unsigned char *LisParams);`

Description: This function is called back from the library when recording is completed. Enter this function using the LisOnEmpty() function in TopLisStart(). (Refer to "voxtop.c".)

5.3.9 PCM Data Processing Functions (ppctop.c)

ppcSpeak()

Function: VSC-converts and reproduces PCM data

Format: `unsigned char* ppcSpeak(unsigned char *Data, int pitch_speed, int real_time);`

Arguments: `unsigned char *Data` Playback PCM data pointer
`int pitch_speed` Tone pitch and talking speed
`int real_time` Real-time parameter

Return values: Terminated normally.....SpkParams pointer
 Terminated abnormally0

Description: After loading PCM-format data (10-bit amplitude, 8K sampling) from Data, this function VSC-converts it in real time before outputting. The data created by "pcm2s.exe" or input by ppcListen() can be reproduced after changing the tone pitch and talking speed.

To specify the second argument (pitch_speed) and third argument (real_time), use the following macros defined in "vsc.h".

VSC_NORMAL Standard
 VSC_SLOW20 Talking speed twice as slow as standard
 VSC_SLOW15 Talking speed 1.5 times as slow as standard
 VSC_FAST15 Talking speed 1.5 times as fast as standard
 VSC_FAST20 Talking speed twice as fast as standard
 VSC_LOW20 Tone pitch twice as low as standard
 VSC_LOW15 Tone pitch 1.5 times as low as standard
 VSC_HIGH15 Tone pitch 1.5 times as high as standard
 VSC_HIGH20 Tone pitch twice as high as standard

Example: Reproducing the data input by pcmListen()

```
unsigned char *SpkParams, *LisParams, pcmData[8000*3];
unsigned char pcmData[8000*3+PACKPCM_HEADER_SIZE];
LisParams = ppcListen(8000*3, 8000*3, pcmData+PACKPCM_HEADER_SIZE,
pcmData);
```

:

```
SpkParams = ppcSpeak(pcmData, VSC_NORMAL);
```

The PCM data in pcmData is reproduced without changing the tone pitch and talking speed.

Reproducing the PCM data created by the user

Create the compressed voice assembly source data that is output by the VOX33 tool "pcm2s.exe".

For example, when you have created the data labeled "se", specify the C source as shown below.

```
extern unsigned char s[];
:
unsigned char *SpkParams;
SpkParams = ppcSpeak(se, VSC_LOW20);
```

The data "se" is reproduced after being VSC-converted (tone pitch is halved).

ppcListen()

Function: Records PCM data

Format: `unsigned char *ppcListen(int Samples, int MaxBytes, unsigned char *Dst, unsigned char *Header);`

Arguments: `int Samples` Maximum number of input samples (sampling rate × seconds)
`int MaxBytes` Maximum number of written data (bytes)
`unsigned char *Dst` Pointer to the beginning of the data write buffer
`unsigned char *Header` Buffer in which to write header information (9 bytes or more)

Return values: Terminated normally.....LisParams pointer
 Terminated abnormally0

Description: This function inputs 10-bit amplitude, 8K-sampling PCM data.

Example: `unsigned char *LisParams, pcmData[8000*3+PACKPCM_HEADER_SIZE];
 LisParams = ppcListen(8000*3, 8000*3, pcmData+PACKPCM_HEADER_SIZE, pcmData);`

Voice data is input for about 3 seconds while sampling at 8 kHz.

ppcTopDecode()

Function: Call-back function for reproducing

Format: `void ppcTopDecode(unsigned char *SpkParams, short *Buffer, int Length);`

Description: This function is called back from the library when free space is available in the playback data queue. It decodes data and adds Buffer to the queue. Enter this function using the SpkOnDone() function in TopSpeakStart(). (Refer to "ppctop.c".)

ppcTopEncode()

Function: Call-back function for recording

Format: `void ppcTopEncode(unsigned char *SpkParams, short *Buffer, int Length);`

Description: This function is called back from the library when data is stored in the record data buffer. It encodes the data stored in Buffer and adds Buffer to the queue. Enter this function using the LisOnDone() function in TopLisStart(). (Refer to "ppctop.c".)

ppcTopEncodeEnd()

Function: Call-back function for completion of recording

Format: `void ppcTopEncodeEnd(unsigned char *LisParams);`

Description: This function is called back from the library when recording is completed. Enter this function using the LisOnEmpty() function in TopLisStart(). (Refer to "ppctop.c".)

5.3.10 Common Functions (voxcomn.c)

Note: When handling data in ADPCM or VSX format, define "ADPCM" when compiling this source.

voxCodecpy()

Function: Copies code section

Format: void voxCodecpy(int *dst, int *src, int *size);

Arguments: int *dst Destination address of transfer (internal RAM)
 int *src Source address of transfer (external ROM)
 int *size Size of transfer code (byte)

Return value: None

Description: This function transfers code from external ROM to internal RAM.

Example: voxCodecpy(&__START_CACHE1, &__START_cpclrdat_code,
 &__SIZEOF_cpclrdat_code);

The code of "cpclrdat.o" is copied into the &__START_CACHE1 position of the internal RAM.

In this case, the following must be written in the linker command file:

```
-section CASHE1
-ucode CACHE1 {(pass)\cpclrdat.o}
```

adpcmCodecpy()

Function: Copies ADPCM or VSX code section

Format: void adpcmCodecpy(unsigned char format);

Argument: unsigned char format Compression ratio

Return value: None

Description: This function transfers ADPCM or VSX code from external ROM to internal RAM (&__START_ADPCODE). For the argument (format), use one of the values defined in "adpcm.h":

```
#define ADPCM_16K (1) Copy object to be compressed at 16 kbps
#define ADPCM_24K (2) Copy object to be compressed at 24 kbps
#define ADPCM_32K (4) Copy object to be compressed at 32 kbps
#define ADPCM_40K (8) Copy object to be compressed at 40 kbps
```

Example: adpcmCodecpy(ADPCM_24k);

The code of "fadpcm24.o" is copied into the &__START_ADPCODE position of the internal RAM. In this case, the following must be written in the linker command file:

```
-section ADPCODE
-ucode ADPCODE {(pass)\fadpcm16.o (pass)\fadpcm24.o (pass)\fadpcm32.o
(pass)\fadpcm40.o}
```

5.3.11 Input/Output Data Convert Functions (*slutil.c*)

The following shows the format of each type of data handled by input/output data functions.

ADPCM data: Signed 14-bit data
 VSX data: Signed 12-bit data
 VOX data: Signed 10-bit data
 PCM data: Signed 10-bit data

setSpeakVolume()

Function: Sets output volume

Format: `void setSpeakVolume(unsigned short spkv);`

Argument: `unsigned short spkv` Playback sound volume

Return value: None

Description: This function sets the playback sound volume. Always be sure to set this volume before calling up the following playback output functions:
`vsxSpeak(), adpcmSpeak(), vox2Speak(), voxSpeak(), ppcSpeak()`

The argument is a relative value defined with reference to 0x100 (= 1 fold). This specified value is multiplied by an internal voice data value to determine the playback sound volume. When the value exceeds the tolerate level, it is rounded off. Since the value can be specified in increments of 1, sophisticated settings are possible.

Example:

```
setSpeakVolume(0x100);    ... 1.0-fold sound volume
setSpeakVolume(0x80);    ... 0.5-fold sound volume
setSpeakVolume(0x200);    ... 2.0-fold sound volume
```

slPcm2Spk()

Function: Converts output data

Format: `void slPcm2Spk(short *Src, short *Dst, int Length, Slparam *slParam);`

Arguments:

<code>short</code>	<code>*Src</code>	Pointer to source data array
<code>short</code>	<code>*Dst</code>	Pointer to write array
<code>int</code>	<code>Length</code>	Number of data to be converted (short)
<code>Slparam</code>	<code>*slParam</code>	Conversion parameter

Return value: None

Description: This function converts the PCM data obtained by expansion of compressed data by offsetting, shifting, or clipping it according to the parameters defined in `slParam`. Use separate arrays for `Src` and `Dst`.

Example: To expand ADPCM (signed 14-bit) data and output the result to a device used for 8-bit signed data, define `slParam` as follows:

```
slParam->offset = 0x80;    Adds offset of 0x80
slParam->shift  = 2 | SPLIS_RSHIFT  Shifts to the right by 2 bits
slParam->limit  = 0xfe;    Clips data with upper limit 0xfe and lower
                           limit 0x0
```

sLis2Pcm()

Function: Converts input data

Format: `void sLis2Pcm(short *Src, short *Dst, int Length, Slparam *slParam);`

Arguments:

<code>short</code>	<code>*Src</code>	Pointer to source data array
<code>short</code>	<code>*Dst</code>	Pointer to write arra
<code>int</code>	<code>Length</code>	Number of data to be converted (short)
<code>Slparam</code>	<code>*slParam</code>	Conversion parameter

Return value: None

Description: This function converts the A/D-converted data by offsetting and shifting it according to the parameters defined in `slParam`. Use separate arrays for `Src` and `Dst`.

Example: To convert A/D-converted (signed 10-bit) data into the signed 12-bit data used for VSX compression, define `slParam` as follows:

```
slParam->offset = -512;           Adds offset of -512
slParam->shift  = 2 | SPLIS_LSHIFT Shifts to the left by 2 bits
```

5.4 VOX33 Library Functions

The VOX33 libraries "vox.lib" and "vox208.lib" contain object files that include the functions required for expansion, compression, and VSC conversion. The VOX33 libraries "sl104.lib" and "sl208.lib" contain object files that include the functions required for voice input/output. By linking these object files to the user program, any desired voice function can be implemented. Note that in order for voice data to be compression-recorded or expansion-reproduced in real time, some objects need to be retrieved from the library and mapped into the internal memory. For details, refer to Section 5.5, "Techniques for Speeding Up Operation".

Table 5.4.1 below lists the VOX33 library functions.

Table 5.4.1 VOX33 Library Functions

vox.lib, vox208.lib

Type	Function name	Description
VSX processing	vsxReadHeader()	Read VSX header (parameter)
	vsxDecodeInit()	Initialize for VSX expansion
	vsxGetDecodePacketSize()	Get VSX expansion packet size
	vsxDecode()	Expand 1 VSX packet
	vsxIsEOF()	Check for VSX end data
	vsxWriteHeader()	Write VSX header (parameter)
	vsxEncodeInit()	Initialize for VSX compression
	vsxSetEncodeData()	Enter VSX compression data
	vsxGetEncodePacket()	Get VSX-compressed packet
	vsxEncodeFlush()	Finish VSX compression
ADPCM processing	vSxWriteEOF()	Write VSX end data
	adpcmReadHeader()	Read ADPCM header (parameter)
	adpcmInit()	Initialize ADPCM
	adpcmDecode()	Expand 1 ADPCM packet
	adpcmEncode()	Compress 1 ADPCM packet
VOX2 processing	adpcmWriteHeader()	Write ADPCM header (parameter)
	vox2ReadHeader()	Read VOX2 header (parameter)
	vox2GetPacketSize()	Get VOX2 packet size
	vox2Init()	Initialize VOX2
	vox2Decode()	Expand 1 VOX2 packet
VOX processing	vox2EOF()	Check for VOX2 end data
	voxReadHeader()	Read VOX header (parameter)
	voxGetPacketSize()	Get VOX packet size
	voxInit()	Initialize VOX
	voxDecode()	Expand 1 VOX packet
	voxEOF()	Check for VOX end data
	voxEncode()	Compress 1 VOX packet
	voxWriteHeader()	Write VOX header (parameter)
voxWriteEOF()	Write VOX end data	
VSC conversion	vsclnit()	Initialize VSC
	vscSetData()	Enter VSC conversion data
	vscGetData()	VSC convert
PCM processing	packpcmReadHeader()	Read packed PCM header (parameter)
	packpcmInit()	Initialize packed PCM
	packpcmDecode()	Decode 1 packed PCM packet
	packpcmEncode()	Encode 1 packed PCM packet
	packpcmWriteHeader()	Write packed PCM header (parameter)
High-pass filter	fltInit()	Initialize cut-off frequency
	fltFiltering()	Filtering

sl104.lib, sl208.lib

Type	Function name	Description
Voice output	SpkSoftening()	Soften start volume
	SpkSampleRate()	Change sampling rate
	SPK_SAMPLING()	Get 16-bit timer reload value (macro)
	SpkInit()	Initialize internal library variables
	SpkOpen()	Open output channel
	SpkClose()	Close output channel
	SpkStart()	Start voice output
	SpkHalt()	Halt voice output
	SpkAppend()	Append the voice to output data queue
	SpkRoom()	Get number of remaining entries in queue
	SpkQueue()	Get number of entries waiting for output
	SpkIsRunning()	Check output status
	SpkOnDone()	Enter reproduction call-back function
	SpkOnEmpty()	Enter reproduction-complete call-back function
	SpkOnNotInTime()	Enter non-realtime operating call-back function
	SpkIntr0()	Process voice output by interrupt
Voice input	LIS_SAMPLING()	Get 16-bit timer reload value (macro)
	LisInit()	Initialize internal library variables
	LisOpen()	Open input channel
	LisClose()	Close input channel
	LisStart()	Start voice input
	LisHalt()	Halt voice input
	LisAppend()	Append the voice to input data queue
	LisRoom()	Get number of entries in queue
	LisQueue()	Get number of entries waiting for input
	LisIsRunning()	Check input status
	LisOnDone()	Enter recording call-back function
	LisOnEmpty()	Enter recording-complete call-back function
	LisOnNotInTime()	Enter non-realtime operating call-back function
	LisIntr0()	Process voice input by interrupt

Note: The VOX33 library functions use the CPU's R8 register. Therefore, when linking the VOX33 library, including the top-level functions to the user program, you cannot use the -gp option (optimization using global pointer/R8) of the instruction extender ext33. Also, make sure the BSS sections used by VOX33 library functions are mapped into the internal RAM.

The following explains the specification of each function. For details on how to use these functions, refer to the top-level function sources.

5.4.1 VSX Processing Functions

vsxReadHeader()

Function: Reads VSX header (parameter)

Format: `int vsxReadHeader(unsigned char *Src, vsxParams *Params);`

Arguments: `unsigned char *Src` Byte array for data input
`vsxParams *Params` Pointer to the structure to which to assign vsxParams value

Return value: Number of bytes read from Src

Description: This function reads VSX parameters (vsxParams) from Src and writes them into a specified structure. If Src does not have the vsxParams structure, the value 0 or a negative value is returned, in which case the content of the structure is not changed.

vsxDecodeInit()

Function: Initializes VSX expansion

Format: `int vsxDecodeInit(vsxParams *Params, int Speed);`

Arguments: `vsxParams *Params` Pointer to vsxParams
`int *Speed` Playback speed

Return value: Terminated normally.....Other than 0
 Terminated abnormall.....0

Description: This function initializes settings for VSX expansion processing with vsxParams and sets the playback speed using the second argument (Speed).

Reference: Defined values of talking speed in vsx.h (Recommended values in bold face)

```
/* play speed */
#define VSX_SPEED_SLOW20 (-2)
#define VSX_SPEED_SLOW15 (-1)
#define VSX_SPEED_NORMAL (0)
#define VSX_SPEED_FAST15 (1)
#define VSX_SPEED_FAST20 (2)
#define VSX_SPEED_FAST30 (3)
#define VSX_SPEED_FAST40 (4)
#define VSX_SPEED_FAST60 (5)
#define VSX_SPEED_FAST80 (6)
#define VSX_SPEED_FAST120 (7)
#define VSX_SPEED_FAST160 (8)
```

vsxGetDecodePacketSize()

Function: Gets VSX-expansion data packet size

Format: `int vsxGetDecodePacketSize(unsigned char *Src);`

Argument: `unsigned char *Src` Byte array for data input

Return value: Number of data (bytes) in one packet

Description: This function returns the number of bytes that comprise one packet beginning with Src. When the data is EOF, the value 1 is returned.

vsxDecode()

Function: Expands one packet of VSX data

Format: `int vsxDecode(unsigned char *Src, int *Cont, short *Dst, int DstSize);`

Arguments:

<code>unsigned char</code>	<code>*Src</code>	Byte array for data input
<code>int</code>	<code>*Cont</code>	Number of repetitions
<code>short</code>	<code>*Dst</code>	short-type data array at output destination
<code>int</code>	<code>DstSize</code>	Output buffer size

Return value: Number of data (bytes) written to Dst

Description: This function decodes one packet of data beginning with Src and writes the decoded data to Dst by appropriately overlapping the basic waveform on it so that the preset playback speed is obtained. If the data cannot be written to Dst in one operation, temporarily store data in the internal buffer, set the number of repetitions in Cont, and call up the function repeatedly beginning with high-order data, until the return value is 0.

Note:

- dstSize must be 120 or more.
- When the packet is skipped vsxDecode() returns 0.
- For repeated calls, set 0 in Cont when first calling up the function for high-order data.

vsxIsEOF()

Function: Checks for VSX end data

Format: `int vsxIsEOF(unsigned char *Src);`

Argument: `unsigned char *Src` Byte array for data input

Return values: When data is end data.....1
When data is not end data....0

Description: This function determines whether Src is the end data of VSX.

vsxWriteHeader()

Function: Writes VSX header (parameter)

Format: `int vsxWriteHeader(vsxParams *Params, int MaxBytes, unsigned char *Dst);`

Arguments:

<code>vsxParams</code>	<code>*Params</code>	Pointer to vsxParams
<code>int</code>	<code>MaxBytes</code>	Maximum number of output bytes
<code>unsigned char</code>	<code>*Dst</code>	unsigned char-type data array at output destination

Return value: Number of data bytes actually written to Dst

Description: This function writes vsxParams to Dst. If the number of bytes to be written is greater than MaxBytes, no parameters are written to Dst. In this case, the value 0 or a negative value is returned.

vsxEncodeInit()

Function: Initializes VSX compression

Format: `int vsxEncodeInit(vsxParams *Params);`

Argument: `vsxParams *Params` Pointer to vsxParams

Return values: Terminated normally.....Other than 0
Terminated abnormally.....0

Description: This function initializes settings for VSX compression processing using vsxParams.

vsxSetEncodeData()

Function: Enters data for VSX compression

Format: `int vsxSetEncodeData(short *Src, int Length);`

Arguments: `short *Src` short-type data array at input source
`int Length` Input data size

Return value: Number of samples actually entered

Description: This function enters the source voice data to be VSX-compressed.

Note: The second argument (Length) and the return value are the number of voice samples, not the number of bytes.

vsxGetEncodePacket()

Function: Gets VSX-compressed packet

Format: `int vsxGetEncodePacket(unsigned char *Dst, int MaxBytes);`

Arguments: `unsigned char *Dst` unsigned char-type data array at output destination
`int MaxBytes` Maximum number of output bytes

Return value: Number of data bytes actually written to Dst

Description: This function writes one packet of VSX-compressed data to Dst. Before using this function, enter the data to be compressed using `vsxSetEncodeData()`. Then call `vsxGetEncodePacket()` repeatedly until the return value is 0. After this function returns 0, the next data to be compressed can be entered using `vsxSetEncodeData()`.

vsxEncodeFlush()

Function: Carry out processes to complete VSX compression

Format: `int vsxEncodeFlush();`

Argument: None

Return value: Terminated normally..... 1

Description: After entering all data to be compressed using `vsxSetEncodeData()`, call `vsxEncodeFlush()` to close the internal buffers. Then call `vsxGetEncodePacket()` repeatedly until the return value is 0. When `vsxGetEncodePacket()` returns 0, compression is terminated.

vsxWriteEOF()

Function: Writes VSX end data

Format: `int vsxWriteEOF(unsigned char *Dst);`

Arguments: `unsigned char *Dst` unsigned char-type data array at output destination

Return values: Number of data bytes actually written to Dst

Description: This function writes the data that indicates the end of VSX data to Dst.

5.4.2 ADPCM Processing Functions

adpcmReadHeader()

Function: Reads ADPCM header (parameter)

Format: `int adpcmReadHeader(unsigned char *Src, adpcmParams *Params);`

Arguments: `unsigned char *Src` Byte array for data input
`adpcmParams *Params` Pointer to the structure to which to assign `adpcmParams` value

Return value: Number of bytes read from `Src`

Description: This function reads ADPCM parameters (`adpcmParams`) from `Src` and writes them into a specified structure.
 If `Src` does not have the `adpcmParams` structure, the value -1 is returned, in which case the content of the structure is not changed.

adpcmInit()

Function: Initializes ADPCM

Format: `int adpcmInit(adpcmParams *Params, int PacketSize);`

Arguments: `adpcmParams *Params` Pointer to `adpcmParams`
`int PacketSize` Packet size (normally 128)

Return values: Terminated normally.....1
 Terminated abnormally.....-1

Description: This function initializes settings for ADPCM processing with `adpcmParams`.

adpcmDecode()

Function: Expands one packet of ADPCM data

Format: `int adpcmDecode(unsigned char *Src, short *Dst);`

Arguments: `unsigned char *Src` unsigned char-type data array at input source
`short *Dst` short-type data array at output destination

Return value: Number of data bytes read from `Src`

Description: This function reads one packet of ADPCM format data from `Src` and converts it into short-type data before writing it to `Dst`. In the output destination array `Dst`, memory space for more than one packet of converted short-type data must be allocated. Also, before using this function, ADPCM processing must be initialized by `adpcmInit()`.

adpcmEOF()

Function: Checks for ADPCM end data

Format: `int adpcmEOF(unsigned char *Src);`

Argument: `unsigned char *Src` Byte array for data input

Return values: When data is not end data.....1
 When data is not end data.....0

Description: This function determines whether `Src` is the end data of ADPCM data.

adpcmEncode()

Function: Compresses one packet of ADPCM data

Format: `int adpcmEncode(short *Src, int MaxBytes, unsigned char *Dst);`

Arguments: `short *Src` short-type data array at input source
`int MaxBytes` Maximum number of output bytes
`unsigned char *Dst` unsigned char-type data array at output destination

Return values: Terminated normally.....Number of data bytes actually written to Dst
 Terminated abnormally-1

Description: This function reads one packet of short-type data from Src and converts it into ADPCM format before writing it to Dst.
 Before using this function, ADPCM processing must be initialized by `adpcmInit()`.
 The number of input data is equivalent to one packet specified by `adpcmParams`.
 If the number of bytes of compressed data is greater than `MaxBytes`, no data is written to Dst. In this case, the value -1 is returned.

adpcmWriteHeader()

Function: Writes the ADPCM header (parameter)

Format: `int adpcmWriteHeader(adpcmParams *Params, int MaxBytes, unsigned char *Dst);`

Arguments: `adpcmParams *Params` Pointer to `adpcmParams`
`int MaxBytes` Maximum number of output bytes
`unsigned char *Dst` unsigned char-type data array at output destination

Return values: Terminated normally.....Number of data bytes actually written to Dst
 Terminated abnormally -1

Description: This function writes `adpcmParams` to Dst.
 If the number of bytes to be written is greater than `MaxBytes`, no parameters are written to Dst. In this case, the value -1 is returned.

5.4.3 VOX2 Processing Functions

vox2ReadHeader()

Function: Reads the VOX2 header (parameter)

Format: `int vox2ReadHeader(unsigned char *Src, voxParams *Params);`

Arguments: `unsigned char *Src` Byte array for data input
`voxParams *Params` Pointer to the structure to which to assign voxParams value

Return value: Number of bytes read from Src

Description: This function reads VOX parameters (voxParams) from Src and writes them into a specified structure.

If Src does not have the voxParams structure, the value 0 is returned, in which case the content of the structure is not changed.

vox2GetPacketSize()

Function: Gets VOX2 packet size

Format: `int vox2GetPacketSize(voxParams *Params);`

Argument: `voxParams *Params` Pointer to voxParams

Return values: Number of data (bytes) in one packet

Description: This function returns the size of packets used as units for processing VOX2 data.

vox2Init()

Function: Initializes VOX2

Format: `int vox2Init(voxParams *Params);`

Argument: `voxParams *Params` Pointer to voxParams

Return values: Terminated normally.....1
 Terminated abnormally.....0

Description: This function initializes settings for VOX2 processing using voxParams.

vox2Decode()

Function: Expands one packet of VOX2 data

Format: `int vox2Decode(unsigned char *Src, short *Dst);`

Arguments: `unsigned char *Src` unsigned char-type data array at input source
`short *Dst` short-type data array at output destination

Return values: Number of data bytes read from Src

Description: This function reads one packet of VOX2 format data from Src and converts it into short-type data before writing it to Dst.

In the output destination array Dst, memory space for more than one packet of converted short-type data must be allocated. Also, before using this function, VOX2 processing must be initialized by vox2Init().

vox2EOF ()

Function: Checks for VOX2 end data

Format: `int vox2EOF(unsigned char *Src);`

Argument: `unsigned char *Src` Byte arrayfor data input

Return values: When data is end data1
When data is not end data0

Description: This function determines whether Src is the end data of VOX2.

5.4.4 VOX Processing Functions

voxReadHeader ()

Function: Reads the VOX header (parameter)

Format: `int voxReadHeader(unsigned char *Src, voxParams *Params);`

Arguments: `unsigned char *Src` Byte array for data input
`voxParams *Params` Pointer to the structure to which to assign voxParams value

Return value: Number of bytes read from Src

Description: This function reads VOX parameters (voxParams) from Src and writes them into a specified structure.

If Src does not have the voxParams structure, the value 0 is returned, in which case the content of the structure is not changed.

voxGetPacketSize()

Function: Gets VOX packet size

Format: `int voxGetPacketSize(voxParams *Params);`

Argument: `voxParams *Params` Pointer to voxParams

Return value: Number of data (bytes) in one packet

Description: This function returns the size of packets used as units for processing VOX data.

voxInit()

Function: Initializes VOX

Format: `int voxInit(voxParams *Params);`

Argument: `voxParams *Params` Pointer to voxParams

Return values: Terminated normally.....1
 Terminated abnormally.....0

Description: This function initializes settings for VOX processing using voxParams.

voxDecode()

Function: Expands one packet of VOX data

Format: `int voxDecode(unsigned char *Src, short *Dst);`

Arguments: `unsigned char *Src` unsigned char-type data array at input source
`short *Dst` short-type data array at output destination

Return value: Number of data bytes read from Src

Description: This function reads one packet of VOX-format data from Src and converts it into short-type data before writing it to Dst. In the output destination array Dst, memory space for more than one packet of converted short-type data must be allocated. Also, before using this function, VOX processing must be initialized by voxInit().

voxEOF()

Function: Checks for VOX end data

Format: `int voxEOF(unsigned char *Src);`

Argument: `unsigned char *Src` Byte array for data input

Return values: When data is end data.....1
When data is not end data.....0

Description: This function determines whether Src is the end data of VOX.

voxEncode()

Function: Compresses one packet of VOX data.

Format: `int voxEncode(short *Src, int MaxBytes, unsigned char *Dst);`

Arguments: `short *Src` short-type data array at input source
`int MaxBytes` Maximum number of output bytes
`unsigned char *Dst` unsigned char-type data array at output destination

Return values: Terminated normally.....Number of data bytes actually written to Dst or 0
Terminated abnormally -1

Description: This function reads one packet of short-type data from Src and converts it into VOX format before writing it to Dst.
Before using this function, VOX processing must be initialized by `voxInit()`.
The number of input data is equivalent to one packet specified by `voxParams`.
If the number of bytes of compressed data is greater than `MaxBytes`, no data is written to Dst. In this case, the value 0 is returned.

voxWriteHeader()

Function: Writes the VOX header (parameter)

Format: `int voxWriteHeader(voxParams *Params, int MaxBytes, unsigned char *Dst);`

Arguments: `voxParams *Params` Pointer to `voxParams`
`int MaxBytes` Maximum number of output bytes
`unsigned char *Dst` unsigned char-type data array at output destination

Return value: Number of data bytes actually written to Dst or 0

Description: This function writes `voxParams` to Dst. If the number of bytes to be written is greater than `MaxBytes`, no parameters are written to Dst. In this case, the value 0 is returned.

voxWriteEOF()

Function: Writes VOX end data

Format: `int voxWriteEOF(unsigned char *Dst);`

Argument: `unsigned char *Dst` unsigned char-type data array at output destination

Return value: Number of data bytes actually written to Dst

Description: This function writes the data that indicates the end of VOX data to Dst.

5.4.5 VSC Processing Functions

vscInit()

Function: Initializes VSC

Format: `int vscInit(int PitchSpeed, int RealTime);`

Arguments: `int PitchSpeed` Tone pitch and talking speed
(specified by logical OR of values defined in vsc.h)
`int RealTime` Real-time processing parameter (defined in vsc.h)

Return values: Terminated normally.....1
Terminated abnormally.....0

Description: This function initializes VSC processing. To specify the first argument (PitchSpeed), use logical OR of the values defined in "vsc.h".

Reference: Tone pitch and talking speed values defined in vsc.h

```
// define pitch (also speed change)      // define speed
#define VSC_PITCH_NORMAL 1879048192     #define VSC_SPEED_NORMAL 100
#define VSC_PITCH_HIGH15 2113929216    #define VSC_SPEED_SLOW15 615
#define VSC_PITCH_LOW15 2063597568     #define VSC_SPEED_FAST15 1126
#define VSC_PITCH_HIGH20 1929379840    #define VSC_SPEED_SLOW20 84
#define VSC_PITCH_LOW20 2080374784     #define VSC_SPEED_FAST20 103
```

The second argument (RealTime) need to be set when the number of samples in output data must be the same before and after the tone pitch is changed.

Normally: `VSC_PITCH_NO_REAL`
When pitch is NORMAL, HIGH15, HIGH20: `VSC_PITCH_REAL_HIGH`
When pitch is LOW15, LOW20: `VSC_PITCH_REAL_LOW`

vscSetData()

Function: Enters VSC conversion data

Format: `int vscSetData(short *Src, int Samples);`

Arguments: `short *Src` short-type data array at input source
`int Samples` Number of data samples to be entered (normally 128)

Return value: Number of samples entered in internal buffer

Description: This function reads data for a specified number of samples from Src and enters it in the VSC internal buffer.

Before using this function, VSC processing must be initialized by vscInit().

Once data is entered in the buffer, VSC-converted data can be obtained by calling up vscGetData().

Note: Before using this function, check that vscGetData() returns 0. If data is entered during VSC processing (i.e., vscGetData() returns any value other than 0), the internal buffer may be corrupted, in which case program operation cannot be guaranteed.

vscGetData()

Function: VSC conversion

Format: `int vscGetData(int Samples, short *Dst);`

Arguments: `int Samples` Number of data samples output (normally 128)
`short *Dst` short-type data array at output destination

Return value: Number of data bytes actually written to Dst or 0

Description: This function VSC-converts the data that has been entered in the internal buffer and writes the converted data to Dst.

Before using this function, VSC processing must be initialized by `vscInit()`.

If data cannot be written to Dst (e.g., no data is entered in the internal buffer), the value 0 is returned. In this case, enter data in the internal buffer using `vscSetData()`.

Note: Do not enter new data using `vscSetData()` until this function returns 0. Otherwise, the internal buffer may be corrupted, in which case program operation cannot be guaranteed.

5.4.6 PCM Processing Functions

packpcmReadHeader()

Function: Reads the packed PCM header (parameter)

Format: `int packpcmReadHeader(unsigned char *Src, packpcmParams *Params);`

Arguments:

<code>unsigned char *Src</code>	Byte array for data input
<code>packpcmParams *Params</code>	Pointer to the structure to which to assign packpcmParams value

Return values: Number of bytes read from Src

Description: This function reads packed PCM parameters (packpcmParams) from Src and writes them into a specified structure.
If Src does not have the packpcmParams structure, the value -1 is returned, in which case the content of the structure is not changed.

packpcmInit()

Function: Initializes packed PCM processing

Format: `int packpcmInit(packpcmParams *Params, int PacketSize);`

Arguments:

<code>packpcmParams *Params</code>	Pointer to packpcmParams
<code>int PacketSize</code>	Packet size (normally 128)

Return values: Terminated normally.....1
Terminated abnormally.....-1

Description: This function initializes packed PCM processing using packpcmParams.

packpcmDecode()

Function: Decodes one packet of packed PCM data

Format: `int packpcmDecode(unsigned char *Src, short *Dst);`

Arguments:

<code>unsigned char *Src</code>	unsigned char-type data array at input source
<code>short *Dst</code>	short-type data array at output destination

Return value: Number of data bytes read from Src

Description: This function reads one packet of packed PCM-format data from Src and converts it into short-type data before writing it to Dst. In the output destination array Dst, memory space for more than one packet of converted short-type data must be allocated. Also, before using this function, packed PCM processing must be initialized by packpcmInit().

packpcmEncode()

Function: Converts one packet of data into packed PCM format

Format: `int packpcmEncode(short *Src, int MaxBytes, unsigned char *Dst);`

Arguments: `short *Src` short-type data array at input source
`int MaxBytes` Maximum number of output bytes
`unsigned char *Dst` unsigned char-type data array at output destination

Return values: Terminated normally.....Number of data bytes actually written to Dst
 Terminated abnormally-1

Description: This function reads one packet of short-type data from Src and converts it into packed PCM format before writing it to Dst. Before using this function, packed PCM processing must be initialized by `packpcmInit()`. The number of input data is equivalent to one packet specified by `packpcmParams`. If the number of bytes of converted data is greater than `MaxBytes`, no data is written to Dst. In this case, the value -1 is returned.

packpcmWriteHeader()

Function: Writes the packed PCM header (parameter)

Format: `int packpcmWriteHeader(adpcmParams *Params, int MaxBytes, unsigned char *Dst);`

Arguments: `packpcmParams *Params` Pointer to `packpcmParams`
`int MaxBytes` Maximum number of output bytes
`unsigned char *Dst` unsigned char-type data array at output destination

Return values: Terminated normally.....Number of data bytes actually written to Dst
 Terminated abnormally -1

Description: This function writes `packpcmParams` to Dst. If the number of bytes to be written is greater than `MaxBytes`, no parameters are written to Dst. In this case, the value -1 is returned.

5.4.7 Output (Speak) Functions

SpkSoftening()

Function: Soften start volume for output

Format: `void SpkSoftening(unsigned char SPK_SOFTENING);`

Argument: `unsigned char SPK_SOFTENING` Output ON/OFF delay time

Return value: None

Description: This function is use to reduce the switching noise that is generated at the start and end of reproduction output. Be sure to set this function before calling SpkStart().
For ×2 oversampled output, the output-ON delay time is obtained from the equation below:
$$1/16000 \times \text{SPK_SOFTENING} \times \text{CENTER_DATA} \text{ [msec]}$$

CENTER_DATA is a median value of output data bits (0x80 for 8 bits, 0x200 for 10 bits).
Check switching noise in the actual application system before determining the delay time.

SPK_SAMPLING()

Function: Gets 16-bit timer reload value (macro)

Format: `SPK_SAMPLING(CpuClock, SamplingRate)`

Arguments: `CpuClock` CPU clock frequency
`SamplingRate` Sampling rate

Return value: 16-bit timer reload value

Description: This is the macro used to acquire the 16-bit timer reload value from the specified CPU clock frequency and sampling rate.

SpkInit()

Function: Initializes internal library variables

Format: `void SpkInit(void);`

Argument: None

Return value: None

Description: This function clears the internal variables used by the library to 0.

SpkOpen()

Function: Opens the output channel

Format: `unsigned char *SpkOpen(int Channel, int ReloadValue);`

Arguments: `int Channel` Channel number
`int ReloadValue` 16-bit timer set value

Return values: Terminated normally.....SpkParams pointer corresponding to the opened channel
Terminated abnormally0

Description: This function opens the specified output channel with a specified sampling rate. The SpkParams value returned by this function is used as an argument for other output (Spk) functions.
For ReloadValue, specify the value acquired by the SPK_SAMPLING macro.
In the following cases, the function fails to open and returns 0.

- When the specified channel is already open
- When an unavailable channel is specified
- When the reload value exceeds 16 bits

To change the channels used, modify the source in the "hardsrc\" directory.

SpkClose()

Function: Closes the output channel

Format: `int SpkClose(unsigned char *SpkParams);`

Argument: `unsigned char *SpkParams` SpkParams pointer (return value of SpkOpen)

Return values: Terminated normally.....Other than 0
Terminated abnormally.....0

Description: This function closes the specified output channel. If the specified channel is not open, it returns 0.

SpkStart()

Function: Starts voice output

Format: `int SpkStart(unsigned char *SpkParams);`

Argument: `unsigned char *SpkParams` SpkParams pointer (return value of SpkOpen)

Return values: Terminated normally.....Other than 0
Terminated abnormally0

Description: This function starts the operation to output voice data in a specified channel. If the specified channel is not open, it returns 0.

SpkHalt()

Function: Halts voice output

Format: `int SpkHalt(unsigned char *SpkParams);`

Argument: `unsigned char *SpkParams` SpkParams pointer (return value of SpkOpen)

Return values: Terminated normally.....Other than 0
Terminated abnormally0

Description: This function halts the operation to output voice data in a specified channel. If output in the specified channel has not been started by SpkStart(), it returns 0.

SpkAppend()

Function: Appends data to output data queue

Format: `int SpkAppend(unsigned char *SpkParams, void *Buffer, int Length);`

Arguments: `unsigned char *SpkParams` SpkParams pointer (return value of SpkOpen)
`void *Buffer` Pointer to the data to be entered
`int Length` Data size

Return values: Terminated normally.....Other than 0
Terminated abnormally0

Description: This function appends the output data to the output queue of a channel specified by SpkParams. If output in the specified channel has not been started by SpkStart() or there is no free entry in the queue, no data is entered and 0 is returned.

SpkRoom()

Function: Gets the number of remaining entries in the queue

Format: `int SpkRoom(unsigned char *SpkParams);`

Argument: `unsigned char *SpkParams` SpkParams pointer (return value of SpkOpen)

Return value: Number of available entries

Description: This function returns the number of available remaining entries in the output queue. When this function is called up immediately after opening an output channel, it shows the maximum number of available entries.
The value returned during voice output operation is as follows:
(Maximum number of entries) - (Number of entries) - (Number of entries that are not called back)

SpkQueue()

Function: Gets the number of entries waiting for output

Format: `int SpkQueue(unsigned char *SpkParams);`

Argument: `unsigned char *SpkParams` SpkParams pointer (return value of SpkOpen)

Return value: Number of entries waiting for output

Description: This function returns the number of entries waiting for output in the output queue. The value returned during voice output operation is as follows:
(Number of queued entries) - (Number of entries that are not called back) - (Number of entries that are called back)

SpkIsRunning()

Function: Checks output status

Format: `int SpkIsRunning(unsigned char *SpkParams);`

Argument: `unsigned char *SpkParams` SpkParams pointer (return value of SpkOpen)

Return values: When output operation is under way.....Other than 0
When output operation has halted.....0

Description: This function returns a value indicating whether output operation in the specified output channel is under way.

SpkOnDone()

Function: Enters the reproduction call-back function

Format: `int SpkOnDone(unsigned char *SpkParams, void *Callback);`

Arguments: `unsigned char *SpkParams` SpkParams pointer (return value of SpkOpen)
`void *Callback` Pointer to the call-back function to be entered

Return value: Pointer to the original call-back function

Description: This function enters the function in a specified output channel that is called back when reproducing voice data. The call-back function has the following format:
`void Callback(unsigned char *SpkParams, void *Buffer, int Length)`

SpkOnEmpty()

Function: Enters the reproduction-complete call-back function

Format: `int SpkOnEmpty(unsigned char *SpkParams, void *Callback);`

Arguments: `unsigned char *SpkParams` SpkParams pointer (return value of SpkOpen)
`void *Callback` Pointer to the call-back function to be entered

Return value: Pointer to the original call-back function

Description: This function enters the function in a specified output channel that is called back upon completion of reproduction. The call-back function has the following format:
`void Callback(unsigned char *SpkParams)`

SpkOnNotInTime()

Function: Enters the non-realtime operating call-back function

Format: `int SpkOnNotInTime(unsigned char *SpkParams, void *Callback);`

Arguments: `unsigned char *SpkParams` SpkParams pointer (return value of SpkOpen)
`void *Callback` Pointer to the call-back function to be entered

Return value: Pointer to the original call-back function

Description: This function enters the function in a specified output channel that is called back if voice data cannot be reproduced in real time. The call-back function has the following format:
`void Callback(unsigned char *SpkParams, void *Buffer, int Length)`

SpkIntr0()

Function: Processes voice output by interrupt

Format: `void SpkIntr0(void);`

Argument: None

Return value: None

Description: This function processes voice output by an interrupt. Use this function only as an interrupt vector.

SpkSampleRate()

Function: Changes sampling rate

Format: `void SpkSampleRate(unsigned char *SpkParams, void *Buffer, int ReloadValue);`

Arguments: `unsigned char *SpkParams` SpkParams pointer (return value of SpkOpen)
`void *Buffer` Pointer to the output data
`int ReloadValue` 16-bit timer set value

Return value: None

Description: This function changes the sampling rate of a channel specified by spkParams according to ReloadValue, starting from the time at which the system outputs the data that begins with Buffer. For ReloadValue, specify the value obtained by the SPK_SAMPLING macro. Use this function if you want to change the sampling rate dynamically after calling SpkStart(). Buffer is the buffer specified by SpkAppend(). Use the buffer immediately before SpkAppend(). Normally, specify the sampling rate in SpkOpen().

5.4.8 Input (Listen) Functions

LIS_SAMPLING()

Function: Gets the 16-bit timer reload value (macro)

Format: LIS_SAMPLING(CpuClock, SamplingRate)

Arguments: CpuClock CPU clock frequency
SamplingRate Sampling rate

Return value: 16-bit timer reload value

Description: This is the macro used to acquire the 16-bit timer reload value from the specified CPU clock frequency and sampling rate.

LisInit()

Function: Initializes internal library variables

Format: void LisInit(void);

Argument: None

Return value: None

Description: This function clears the internal variables used by the library to 0.

LisOpen()

Function: Opens input channel

Format: unsigned char *LisOpen(int Channel, int ReloadValue);

Arguments: int Channel Channel number
int ReloadValue 16-bit timer set value

Return values: Terminated normally.....LisParams pointer corresponding to the opened channel
Terminated abnormally.....0

Description: This function opens the specified input channel with a specified sampling rate. The LisParams value returned by this function is used as an argument for other input (Lis) functions. For ReloadValue, specify the value acquired by the LIS_SAMPLING macro. In the following cases, the function fails to open and returns 0.

- When the specified channel is already open (For input, only channel 1 can be opened.)
- When an unavailable channel is specified
- When the reload value exceeds 16 bits

To change the channels used, modify the source in the "hardsrc\" directory.

LisClose()

Function: Closes input channel

Format: int LisClose(unsigned char *LisParams);

Argument: unsigned char *LisParams LisParams pointer (return value of LisOpen)

Return values: Terminated normally.....Other than 0
Terminated abnormally.....0

Description: This function closes the specified input channel. If the specified channel is not open, it returns 0.

LisStart()

Function: Starts voice input

Format: `int LisStart(unsigned char *LisParams);`

Argument: `unsigned char *LisParams` LisParams pointer (return value of LisOpen)

Return values: Terminated normally.....Other than 0
Terminated abnormally0

Description: This function starts the operation to input voice data in a specified channel. If the specified channel is not open, it returns 0.

LisHalt()

Function: Stops voice input

Format: `int LisHalt(unsigned char *LisParams);`

Argument: `unsigned char *LisParams` LisParams pointer (return value of LisOpen)

Return values: Terminated normally.....Other than 0
Terminated abnormally0

Description: This function stops the operation to input voice data in a specified channel. If input in the specified channel has not been started by LisStart(), it returns 0.

LisAppend()

Function: Appends data to input data queue

Format: `int LisAppend(unsigned char *LisParams, void *Buffer, int Length);`

Arguments: `unsigned char *LisParams` LisParams pointer (return value of LisOpen)
`void *Buffer` Pointer to the data to be entered
`int Length` Data size

Return values: Terminated normally.....Other than 0
Terminated abnormally0

Description: This function appends the input buffer to the input queue of a channel specified by LisParams. If input in the specified channel has not been started by LisStart() or there is no free entry in the queue, 0 is returned.

LisRoom()

Function: Gets the number of remaining entries in the queue

Format: `int LisRoom(unsigned char *LisParams);`

Argument: `unsigned char *LisParams` LisParams pointer (return value of LisOpen)

Return value: Number of available entries

Description: This function returns the number of available remaining entries in the input queue. When this function is called immediately after opening an input channel, it shows the maximum number of available entries. The value returned during voice input operation is as follows:
(Maximum number of entries) - (Number of queued entries) - (Number of entries that are not called back)

LisQueue()

Function: Gets the number of entries waiting for input

Format: `int LisQueue(unsigned char *LisParams);`

Argument: `unsigned char *LisParams` LisParams pointer (return value of LisOpen)

Return value: Number of entries waiting for input

Description: This function returns the number of entries waiting for input in the input queue. The value returned during voice input operation is as follows:
(Number of queued entries) - (Number of entries that are not called back) - (Number of entries that are called back)

LisIsRunning()

Function: Checks input status

Format: `int LisIsRunning(unsigned char *LisParams);`

Argument: `unsigned char *LisParams` LisParams pointer (return value of LisOpen)

Return values: When input operation is under way.....Other than 0
When input operation has halted.....0

Description: This function returns a value indicating whether input operation in the specified input channel is under way.

LisOnDone()

Function: Enters the recording call-back function

Format: `int LisOnDone(unsigned char *LisParams, void *Callback);`

Arguments: `unsigned char *LisParams` LisParams pointer (return value of LisOpen)
`void *Callback` Pointer to the call-back function to be entered

Return value: Pointer to the original call-back function

Description: This function enters the function in a specified input channel that is called back when recording voice data. The call-back function has the following format:
`void Callback(unsigned char *LisParams, void *Buffer, int Length)`

LisOnEmpty()

Function: Enters the recording-complete call-back function

Format: `int LisOnEmpty(unsigned char *LisParams, void *Callback);`

Arguments: `unsigned char *LisParams` LisParams pointer (return value of LisOpen)
`void *Callback` Pointer to the call-back function to be entered

Return value: Pointer to the original call-back function

Description: This function enters the function in a specified input channel that is called back upon completion of recording. The call-back function has the following format:
`void Callback(unsigned char *LisParams)`

LisOnNotInTime()

Function: Enters the non-realtime operating call-back function

Format: `int LisOnNotInTime(unsigned char *LisParams, void *Callback);`

Arguments: `unsigned char *LisParams` LisParams pointer (return value of LisOpen)
`void *Callback` Pointer to the call-back function to be entered

Return value: Pointer to the original call-back function

Description: This function enters the function in a specified input channel that is called back if voice data cannot be recorded in real time. The call-back function has the following format:
`void Callback(unsigned char *LisParams, void *Buffer, int Length)`

LisIntr0()

Function: Processes voice input by interrupt

Format: `void LisIntr0(void);`

Argument: None

Return value: None

Description: This function processes voice input by an interrupt. Use this function only as an interrupt vector.

5.4.9 High-Pass Filter Functions

About the high-pass filter

- This filter attenuates the sound pressure level by 40 dB at half the specified cut-off frequency. Generally speaking, when the sound pressure level decreases by 6 dB, the sound volume is halved.
- The characteristic of this filter is such that the sound pressure level attenuates gently, starting from a region slightly above the cut-off frequency.
- For the VOX33 library, Seiko Epson recommends always specifying a cut-off frequency of about 180 Hz. However, because the sound quality of some voice data deteriorates as a result of filtering, sound quality ultimately must be determined on the user application system.

fltInit()

Function: Initializes cut-off frequency

Format: `int fltInit(int CutOff);`

Argument: `int CutOff` Cut-off frequency

Return values: Terminated normally.....1
Terminated abnormally-1

Description: This function initializes high-pass filtering.
The cut-off frequency (Hz) can be selected from the values listed below. Specifying any other value results in an error.
60, 120, 180, 240, 300, 360, 420, 480, 540, 600, 720, 1440, 250, 500, 1000, 2000

fltFiltering()

Function: Filtering

Format: `int fltFiltering(short *Src, short *Dst, short PacketSize);`

Arguments: `short *Src` Pointer to the source data array to be filtered
`short *Dst` Pointer to the array to which to write filtered data
`short PacketSize` Number of data to be processed
Choose a value in the range of 0 to 128.

Return values: Terminated normally.....Number of filtered data
Terminated abnormally-1

Description: This function filters the input voice data with the cut-off frequency specified by `fltInit()`.
The same array may be specified for `Src` and `Dst`. In this case, data is overwritten.

5.5 Techniques for Speeding Up Operation

By executing several objects in "vox.lib" after mapping them into the internal memory, it is possible to increase the library processing speed by up to 30%. To map library objects into the internal memory, use the linker's U section function. The necessary processing is described below.

1. Retrieve the necessary objects from the library.

They can be restored into the object file using the -x option of librarian lib33.

Example: lib33 -x vox.lib cpclrdat.o

* The lib directory contains the sample objects listed below. Therefore, these objects can be used directly.
vox33asm.o, vox2asm.o, mesa.o, cpclrdat.o, fadpcm16.o, fadpcm24.o, fadpcm32.o, fadpcm40.o

2. Write the following in the linker command file.

```
-objsym                               ; Create object symbol
-section <name>                        ; Create section symbol
-ucode <name> { <object file> [<object file>....] } ; Map into U section
```

Example:

```
-objsym
-section CACHE3
-ucode CACHE3 {..\lib\cpclrdat.o}
```

Looking at the linked map file, you will find that the execution addresses of "cpclrdat.o" have been mapped into the internal memory.

Example: Map file

Address	Vaddress	Size	File
00c19c14	<u>00000f8c</u>	00000038	..\lib\cpclrdat.o

3. Modify the source statements so that when booting or before calling a library function, the relevant object code will be transferred into the internal memory.

Example:

```
xld.w %r12, __START_CACHE3
xld.w %r13, __START_cpclrdat_code
xld.w %r14, __SIZEOF_cpclrdat_code
call HCOPY_LOOP
```

Top-level functions can also be used for code transfer. Use voxCodeCpy() to transfer VOX and VOX2 codes or adpcmCodeCpy() to transfer VSX and ADPCM codes.

Example:

```
voxCodecpy(&__START_CACHE3, &__START_cpclrdat_code, &__SIZEOF_cpclrdat_code);
adpcmCodecpy(ADPCM_24k);
```

- Note:
- When handling data in VSX or ADPCM format, always be sure to map fadpcm16.o, fadpcm24.o, fadpcm32.o, or fadpcm40.o into the internal memory according to the compression ratio used. When using adpcmCodeCpy() for code transfer, define "ADPCM" in voxcomn.c before compiling.
 - When handling VOX2-format data, be sure to map two objects, vox2asm.o and cpclrdat.o, into the internal memory.
 - When handling VOX-format data, be sure to map three objects, voxasm.o, mesa.o, and cpclrdat.o, into the internal memory.

5.6 Library Performance and Memory Size

5.6.1 CPU Occupancy of VOX33 Library

The CPU occupancy rates shown below have been calculated assuming the following conditions as the standard environment:

Operating frequency:	20 MHz
CODE section:	External ROM (2 wait cycles)
BSS section and stack:	Internal RAM (no wait cycle)
VOX parameters:	Depth = 3, Width = 8
ADPCM parameters:	Compression ratio = 24 kbps (ADPCM_24K)
VSX parameters:	Compression ratio = 24 kbps & ×2 equivalent compression (VSX_TIME_CMP_20 VSX_COMPRESS_24K), Silent threshold = 20
VSC conversion parameters:	Talking speed = Twice normal level (VSC_FAST20)

The note "Internal RAM cache used" means that the objects mapped into external memory have been copied into the internal RAM before use. The following shows the relationship between VOX33 functions and the copied objects.

Function	Object
VOX expansion/compression	vox33asm.o, mesa.o, cpcldrdat.o
VOX2 expansion	vox2asm.o, cpcldrdat.o
ADPCM expansion/compression	fadpcm16.o, asdpcm24.o, fadpcm32.o, fadpcm40.o
VSX expansion/compression	fadpcm16.o, asdpcm24.o, fadpcm32.o, fadpcm40.o

CPU occupancy rates during voice expansion/output

Table 5.6.1 CPU Occupancy Rates During Voice Expansion/Output (Values Expressed in %)

Condition	VSX	ADPCM	VOX2	VOX	VSC	Multiplying factor relative to standard (= 1)
Standard environment	50	110	27	20	15	1.0 times
Internal RAM cache used	21	45	22	17	–	0.8 times (ADPCM, VSX: 0.5 times)
External ROM (3 wait cycles) used	66	143	34	26	19	1.2 times
External RAM (2 wait cycles) used for BSS and stack	59	130	39	28	23	1.4 times
External RAM (3 wait cycles) used for BSS and stack	62	137	42	32	26	1.6 times

The CPU occupancy rate for operations associated with voice output (SPEAK) is 5% in the standard environment.

CPU occupancy rates during voice compression/input

The voice compression/input function can only be used if the internal RAM cache is used.

Table 5.6.2 CPU Occupancy Rates During Voice Compression/Input (Values Expressed in %)

Condition	VSX	ADPCM	VOX
Standard environment	88	130	145
Internal RAM cache used	50	58	80

The CPU occupancy rate for operations associated with voice input (LISTEN) is 5% in the standard environment.

CPU occupancy rate during expansion with varying parameters

For VSX, the CPU occupancy rate decreases by about 50% when the timebase compression value is increased by 1. However, the CPU occupancy rate does not decrease further when the compression value is 3 or greater.
 For VOX, the CPU occupancy rate does not change even when parameters are changed.
 For ADPCM, the CPU occupancy rate does not change even when parameters are changed.
 For VSC, the CPU occupancy rate increases by about 30% when VSC_PITCH_REAL_HIGH or VSC_REAL_LOW is used.

CPU occupancy rate during compression with varying parameters

For VSX, the CPU occupancy rate decreases by about 50% when the timebase compression value is increased by 1. However, the CPU occupancy rate does not decrease further when the compression value is 3 or greater.
 For VOX, the CPU occupancy rate increases by about 50% when Depth is increased by 1. The CPU occupancy rate increases by about 10% when Width is reduced by 1.
 For ADPCM, the CPU occupancy rate does not change even when parameters are changed.

Determination of whether real-time execution is possible

If the sum total of the values shown in the above table is 90% or less, the program may be considered capable of running in real time in the corresponding environment.

For "Multiplying factor relative to standard (= 1)" in Table 5.6.1, refer to the following information when combining conditions.

The CPU occupancy rate when vox2Speak() is executed using external ROM (3 wait cycles) and internal RAM cache, for example, can be calculated as follows:

$$(27\% + 15\% + 5\%) \times 0.8 \times 1.2 = 45\%$$

$$45\% / 0.9(\text{margin}) = 50\%$$

This value indicates that voice data can be output in real time even when the talking speed is doubled by VSC conversion. (Because the output time is halved when the talking speed is doubled, the CPU occupancy rate must be 50% or less.)

This value is given for reference purposes only. Before making the final decision, check the return values of SpkQueue() and LisQueue() in the call-back functions that have been entered by SpkOnDone() and LisOnDone().

The values returned by SpkQueue() and LisQueue() are as follows:

(Number of queued entries) - (Number of entries that are not called back) - (Number of entries that are called back)

The "Number of entries that are not called back" means the number of entries that cannot be processed by calling back the designated function. The "Number of entries processed by call back" means the number of entries being processed by a call-back function, and is normally 1. The maximum number of entries that can be queued is 16.

If SpkQueue() or LisQueue() returns 0, it means that voice data probably is not being processed in real time. The following shows adpTopDecode() as an example of a call-back function.

```

/*
 * Decode ADPCM data and output PCM data.
 * Call back function register to SpkOnDone().
 */
int adpTopDecode
(
  unsigned char *SpkParams,
  short *Buffer,
  int Length
)
{
  int len;

```

```

len = adpcmDecode(adpData, SpkDecBuf);
/* Decode end */
if(len < 0)
    return;
adpData += len;

/* SpkDecBuf is 14 bit PCM data */
slPcm2Spk(SpkDecBuf, Buffer, PACKET_SIZE, &SlParam);

if(SpkQueue(SpkParams) == 0)
    return; /* Not processed in real time */
SpkAppend(SpkParams, Buffer, PACKET_SIZE<<1); /* over sampling */

return len;
}

```

5.6.2 Memory Sizes Used

Table 5.6.3 lists the memory sizes used by each function. The values in this table were measured using the sample programs in the "sample\" directory. The internal RAM is used for the BSS section, stack, and program cache. For 10-bit PWM output, the BSS section requires a memory size about 770 bytes larger than the value shown in the table below.

Table 5.6.3 Memory Sizes Used (Values Expressed in Bytes)

Function	CODE	BSS	Stack	Program cache	Internal RAM
VSX playback/recording	17K	2.8K	320	2.2K	5.3K
ADPCM playback/recording	16K	2K	250	2.2K	4.5K
VOX2 playback	12K	4K	600	300	5.0K
VOX playback/recording	13K	4K	800	1K	5.8K

5.7 Precautions

- (1) The VOX33 library functions use the CPU's R8 register. Therefore, when linking the VOX33 library-including the top-level functions to the user program, you cannot use the `-gp` option (optimization using global pointer/R8) of the instruction extender `ext33`.
- (2) The requirements for real-time voice processing are as follows:
 - Make sure all of the BSS sections used by the VOX33 library are mapped into the internal RAM.
 - Be sure to use the internal RAM for the stack.
 - When mapping VOX33 library program code into an external memory area, make sure this area is accessed in 2 wait cycles or less, if possible. Also, be sure to use a memory area 16 bits wide for this external area.
- (3) When handling data in VSX or ADPCM format, always be sure to map `fadpcm16.o`, `fadpcm24.o`, `fadpcm32.o`, or `fadpcm40.o` into the internal memory according to the compression ratio used.

When handling data in VOX2 format, be sure to map two objects, `vox2asm.o` and `cpclrdat.o`, into the internal memory.

When handling data in VOX format, be sure to map three objects, `voxasm.o`, `mesa.o`, and `cpclrdat.o`, into the internal memory.
- (4) The number of samples that can actually be recorded differs by several packets from the maximum number of input samples (sample) specified by each compression/recording top-level function (`*Listen`).
- (5) When VSX data or VSC-converted voice data is reproduced, the number of data samples output differs by several packets from the source voice data.
- (6) Depending on the output amp circuit used, the `SpkSoftening()` function may cause quantization noise at the start and end of playback output. In such a case, process the playback data to make the noise less conspicuous by reducing the length of silent parts preceding and following the playback data.

Appendix Verifying Operation with DMT33 Boards

This section describes how to verify the operation of voice compression and expansion by executing a sample program using E0C33 Family demonstration tools, the DMT33004, DMT33MON, and DMT33AMP.

A.1 System Configuration Using DMT33004

A.1.1 Hardware Configuration

Configure the system shown in Figure A.1.1 using the DMT33004, DMT33MON, and DMT33AMP. This system allows you to compression-record voice data that has been input from a microphone using the DMT33AMP and expansion-reproduce voice data that has been inserted into the program and input from a microphone.

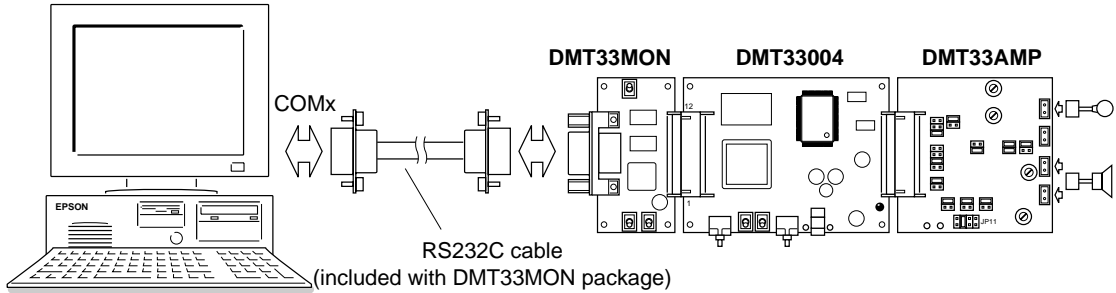


Figure A.1.1 System Configured with DMT33004, DMT33MON, and DMT33AMP

DMT33004 board

The DMT33004 is a demonstration tool for the E0C33A104, a 32-bit RISC-type microcomputer. Mounted on this board are the 128KB ROM, 1MB RAM, and 1MB flash memory, an interface connector for the DMT33MON board, and an interface connector for the DMT33AMP board and other voice input/output circuits. The ROM on this board contains a debugging monitor.

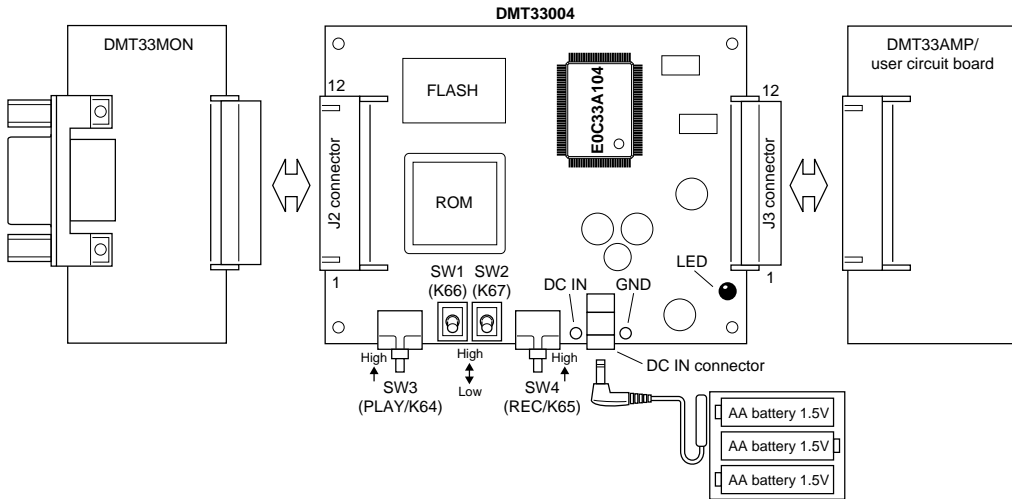


Figure A.1.2 DMT33004 Board

DMT33MON board

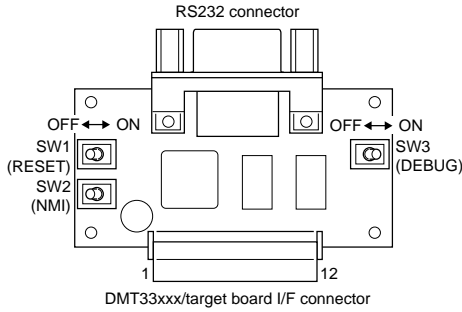


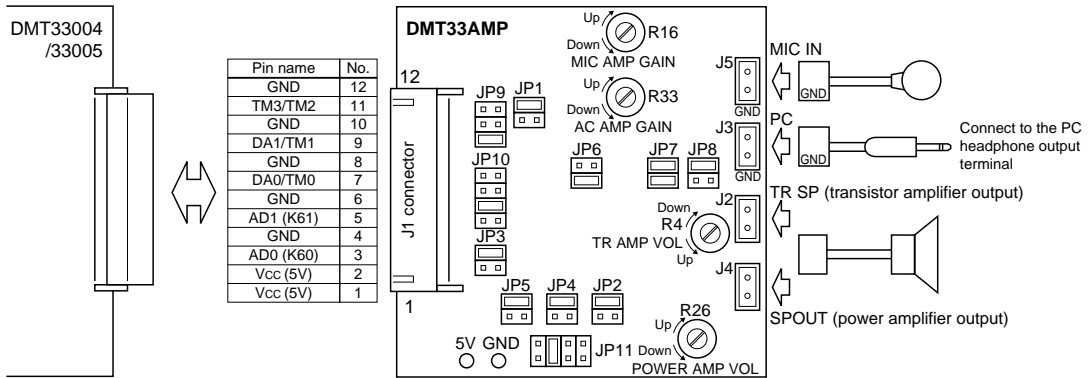
Figure A.1.3 DMT33MON Board

The DMT33MON interfaces with demonstration tools such as the DMT33004 and the user target board for a debugging monitor. By connecting the DMT33004 board to your personal computer via the DMT33MON board, you can debug programs on-board using the debugger (db33.exe) installed in your computer.

Note: For the DMT33004 board, always be sure to use the DMT33MON, which is designed to operate with a 5-V power supply. The DMT33MONLV board, which is designed to operate at 3.3 V, cannot be used.

DMT33AMP board

The DMT33AMP is an optional board that adds voice input/output functions to the DMT33004, etc. It allows voice input from a microphone and voice output from the amplifier mounted on the board. It also allows you to test the configuration (and effect) of speaker/microphone low-pass and high-pass filters, which determine sound quality.



Jumper switch

- JP1** DMT MIC
Selects the voice source to be output.
DMT: DMT33004/33005 output (default)
MIC: Microphone input of this board
- JP2** DA TM
Selects an input for the transistor amplifier circuit.
DA: DMT33004 D/A output (default)
TM: DMT33005 PWM output
- JP3** DA TM
Selects an input for the CR 2nd order filter circuit.
DA: DMT33004 D/A output (default)
TM: DMT33005 PWM output
- JP4** SP MIC
Selects a filter in the OP AMP 4th order filter circuit (for speaker and MIC).
SP: For speaker (default)
MIC: For microphone
- JP5** SP MIC
Selects an filter in the OP AMP 4th order filter circuit (for speaker and MIC).
SP: For speaker (default)
MIC: For microphone
- JP6** AD1 AD0
Selects the A/D channel on the DMT33004/33005 used to convert the MIC input.
AD0: Channel 0 (default)
AD1: Channel 1
- JP7** 3300pF 1500pF
Selects a cutoff frequency in the CR 1st order high-pass filter circuit.
Short 3300pF only: 300 Hz
Short 1500pF only: 500 Hz
Short both: 250 Hz (default)

- JP8** MIC MLP
Selects whether the OP AMP 4th order filter circuit for the MIC circuit is used or not.
MIC: Not used (default)
MLP: Used
- JP9** TM3/TM2 DA1/TM1 DA0/TM0
Selects a DMT33004/33005 output signal.
TM3/TM2: DMT33004 TM3 or DMT33005 TM2
DA1/TM1: DMT33004 DA1 or DMT33005 TM1
DA0/TM0: DMT33004 DA0 or DMT33005 TM0 (default)
- JP10** TR 2LP 4LP MLP
Selects the circuit to be used for voice output.
TR: Transistor amplifier circuit
2LP: CR 2nd order filter circuit
4LP: OP AMP 4th order filter circuit (for speaker) (default)
MLP: OP AMP 4th order filter circuit (for speaker and MIC)
- JP11** PC 2LP 4LP MLP
Selects a power amplifier input.
PC: PC headphone output
2LP: CR 2nd order filter circuit
4LP: OP AMP 4th order filter circuit (for speaker) (default)
MLP: OP AMP 4th order filter circuit (for speaker and MIC)

Note: The DMT33AMP is set for connecting the DMT33004 by default. When using with the DMT33005, select TM using JP2 and JP3, and DA1/TM1 using JP9.

Control

- R26** Volume adjustment for the power amplifier
- R4** Volume adjustment for the transistor amplifier
- R33** Gain adjustment for the AC amplifier (x2 to x12)
- R16** Gain adjustment for the microphone input (microphone amplifier) (x90 to x2000)

Figure A.1.4 DMT33AMP Board

In systems where the DMT33AMP is used along with the DMT33004 board, the jumper switches do not need to be changed; default settings suffice.

System connections

Note: Before setting up or dismantling the system, always be sure to turn off the power to all boards and equipment to be connected or disconnected. For handling precautions for each board, refer to the "E0C33 Family Demonstration Board Manual".

1. Attach the DMT33MON and DMT33AMP to the DMT33004.
2. Connect the microphone and speaker (included with the DMT33AMP package) to the DMT33AMP.
3. Connect the DMT33MON and the personal computer using the RS232C cable (included with the DMT33MON package).
4. Set the [DEBUG] switch (SW3) on the DMT33MON to the "ON" position.
5. Place the battery in the battery holder (included with the DMT33004) and connect it to the DMT33004.
6. Turn on the power to the personal computer.

A.1.2 Software

The personal computer serving as the host must have the development tool "E0C33 Family C Compiler Package" installed in it.

Note that when using the debug monitor to download the program into the DMT33004, a debugger (db33) of Ver. 1.72 or later is required.

A.2 Program Execution Procedure

Each sample program directory contains an absolute object file in executable format. Therefore, you do not need to compile or link the sample programs.

An explanation of how to download the program into the DMT33004 and verify its operation is given below. For details on make and other necessary files for the program, refer to Appendix A.3, "Program Examples".

Note that in the explanation below, the VSX sample program for the E0C33A104 that is stored in the "voxlib\sample\vsx\" directory is used as an example.

- (1) Connect the boards and personal computer, following the explanation of system connection in Appendix A.1.1, and turn on the power.
- (2) Before the program can be downloaded, the debug monitor must be active on the DMT33004. After reconfirming that the [DEBUG] switch (SW3) on the DMT33MON is set to the "ON" position, reset the system using the [RESET] switch (SW1).
- (3) After making "voxlib\sample\vsx\" the current directory, execute "vsxdemo.bat" from the DOS prompt.

```
C:\E0C33\VOX33\VOXLIB\SAMPLE\VSX>vsxdemo
```

This batch file starts the debugger in debug monitor mode (-mon), assuming that the debugger (db33) has been installed in the "c:\cc33\" directory.

Contents of "vsxdemo.bat"

```
@echo off
start c:\cc33\db33 -p 33104_v.par -b 115200 -c vsxdemo.cmd -mon
```

The debugger can also be started from the work bench (wb33). In this case, the debug monitor mode and the command file (vsxdemo.cmd) to be executed at startup must be selected on wb33.

- (4) When the debugger starts up, the sample program "vsxdemo.srf" is loaded into the RAM (beginning with 0x600000) of the DMT33004 by the command written in "vsxdemo.cmd" and the sample program starts running.
- (5) To stop executing the sample program, use the debugger's break function.

The following shows how the sample program is executed.

- (1) The sample voice data (PCM data), "voxtool\sample\se.pcm", is reproduced once.
- (2) The sample voice data "se.pcm" is expansion-reproduced using VSX-compressed data. This reproduction is executed in the following manner:
 1. The data compressed to 40 kbps and ×2 equivalent in the timebase direction is reproduced at normal speed.
 2. The data compressed to 32 kbps and ×2 equivalent in the timebase direction is reproduced at normal speed.
 3. The data compressed to 24 kbps and ×2 equivalent in the timebase direction is reproduced at normal speed.
 4. The data compressed to 16 kbps and ×2 equivalent in the timebase direction is reproduced at normal speed.
 5. The data compressed to 24 kbps and ×1 equivalent in the timebase direction is reproduced at normal speed.
 6. The data compressed to 24 kbps and ×2 equivalent in the timebase direction is reproduced at normal speed.
 7. The data compressed to 24 kbps and ×3 equivalent in the timebase direction is reproduced at normal speed.
 8. The data compressed to 24 kbps and ×4 equivalent in the timebase direction is reproduced at normal speed.
 9. The data compressed to 24 kbps and ×2 equivalent in the timebase direction is reproduced at twice normal speed.

(3) After the above reproduction is completed, the DMT board stands by waiting for switch input, allowing you to record or reproduce voice data using switches. The following shows how to use each switch.

NMI(SW2) on DMT33MON, SW1 and SW2 on DMT33004

These switches set VSX compression parameters.

Each time you press the NMI switch, the compression ratio changes between 24 kbps and 32 kbps.

SW1 and SW2 determine the compression ratio in the timebase direction.

Table A.2.1 Settings of SW1 and SW2

SW2	SW1	Compression ratio in the timebase direction
Low	Low	Compressed to ×2 equivalent
Low	High	Not compressed
High	Low	Compressed to ×3 equivalent
High	High	Compressed to ×4 equivalent

REC(SW4) on DMT33004

When you press the REC switch, the LED on the DMT33004 lights for about 3 seconds. The voice input from the microphone on the DMT33AMP is recorded in VSX format during this time.

PLAY(SW3) on DMT33004

When you press the PLAY switch, the recorded data is reproduced after expansion. This reproduction is performed at the original speed and then at twice the original speed.

A.3 Program Examples

Because the sample program can be modified before testing as necessary, the following explains the make procedure and the necessary files for your reference.

Note that as in Appendix A.2, the explanation is given using the VSX sample program in the "voxlib\sample\vsx\" directory as an example.

A.3.1 Explanation Concerning Files

Source files

The main program for VSX is "vsxdemo.c" in the "voxlib\sample\vsx\" directory.

In addition, the files "boot.s", "demoasm.s", and "table.s" provided in the "voxlib\sample\common\" directory are also used. These sources are also used in common by sample programs other than VSX. Furthermore, "vsxdata.vs" is used as voice data for playback purposes.

vsxdemo.c

This is the main routine of the VSX sample program. The contents executed by this routine are shown in Appendix A.2.

Since the top-level functions for VSX are used, "voxcomn.h" and "vsx.h" are included at the beginning of the file. Also, the label of the playback voice data is defined as an external variable.

The size (DATA_SIZE) of the voice data buffer (CmpData) is set to 8000*3. This setting is necessary to handle data equivalent to about three seconds of 8K sampling.

GetEvent(), which is called up from this routine, is the function used to get the status of the K6[7:4] input port (DMT33004 SW1 to SW4). This function is defined in "demoasm.s".

NMI_CNT is the NMI input count (NMI switch on DMT33MON), which shows the value counted by the NMI processing routine in "demoasm.s".

The top-level functions vsxSpeak(), vsxListen(), and ppcSpeak() and library functions SpkIsRunning() and setSpeakVolume() are used to reproduce and record voice data. For details on these functions, refer to Section 5, "VOX33 Library Reference".

```
#include "voxcomn.h"
#include "vsx.h"

extern unsigned char sep[];
extern unsigned char sev24t1[];
extern unsigned char sev24t2[];
extern unsigned char sev24t3[];
extern unsigned char sev24t4[];
extern unsigned char sev16t2[];
extern unsigned char sev32t2[];
extern unsigned char sev40t2[];

const char vsx_ratio[] = {
    VSX_TIME_CMP_20|VSX_COMPRESS_24K,
    VSX_TIME_CMP_10|VSX_COMPRESS_24K,
    VSX_TIME_CMP_30|VSX_COMPRESS_24K,
    VSX_TIME_CMP_40|VSX_COMPRESS_24K,
    VSX_TIME_CMP_20|VSX_COMPRESS_32K,
    VSX_TIME_CMP_10|VSX_COMPRESS_32K,
    VSX_TIME_CMP_30|VSX_COMPRESS_32K,
    VSX_TIME_CMP_40|VSX_COMPRESS_32K
};

#define DATA_SIZE (8000*3)
static unsigned char CmpData[DATA_SIZE];

vsxSpeakBatch(unsigned char *Data, int speed)
{
    unsigned char* SpkParams;
    SpkParams = vsxSpeak(Data, speed);
    if(SpkParams==0) return;
    do { } while(SpkIsRunning(SpkParams));
}
```

```

Listen(int mode)
{
    unsigned char* LisParams;

    LedON();
    Wait(200);
    LisParams = vsxListen(DATA_SIZE,DATA_SIZE,CmpData,vsx_ratio[mode],50);
    do { } while(LisIsRunning(LisParams));
    LedOFF();
}

void main(void)
{
    int mode;
    unsigned char* SpkParams;
    extern char NMI_CNT;

    *CmpData = 0;    /* init recording data */

    setSpeakVolume(0x100);

    SpkParams = ppcSpeak(sep, 0, 0);
    do { } while(SpkIsRunning(SpkParams));

    vsxSpeakBatch(sev40t2, VSX_SPEED_NORMAL);
    vsxSpeakBatch(sev32t2, VSX_SPEED_NORMAL);
    vsxSpeakBatch(sev24t2, VSX_SPEED_NORMAL);
    vsxSpeakBatch(sev16t2, VSX_SPEED_NORMAL);
    Wait(10000);
    vsxSpeakBatch(sev24t1, VSX_SPEED_NORMAL);
    vsxSpeakBatch(sev24t2, VSX_SPEED_NORMAL);
    vsxSpeakBatch(sev24t3, VSX_SPEED_NORMAL);
    vsxSpeakBatch(sev24t4, VSX_SPEED_NORMAL);
    Wait(10000);
    vsxSpeakBatch(sev24t2, VSX_SPEED_FAST20);

    /* record and play */
    for(;;) {
        /* mode is 4bit data [SW2 SW1 SW4 SW3] */
        mode = GetEvent();
        switch(mode & 3) {
            case 1:
                vsxSpeakBatch(CmpData, VSX_SPEED_NORMAL);
                vsxSpeakBatch(CmpData, VSX_SPEED_FAST20);
                break;
            case 2:
                /* mode change to compression ratio
                 by SW1, SW2 status and NMI count */
                mode >>= 2;
                mode |= (NMI_CNT&1)<<2;
                Listen(mode);
                break;
            default:
                break;
        }
    }
}

```


boot.s

This is the routine used to initialize system settings after a reset. After initializing the stack and bus, it calls the main routine.

```
#define STACK_INIT      0x00001800
#define PSR_INIT        0x00000110      ; InitIntr. Level 1, Intr. enable

.global Boot
Boot:
    xld.w    %r4,STACK_INIT
    ld.w     %sp,%r4                    ; set STACK
    xld.w    %r4,PSR_INIT
    ld.w     %psr,%r4                    ; set PSR
;
    xcall   InitBusCtrl
    xcall   InitCPUClock

    ld.w    %r4,0
    xld.w   [NMI_CNT],%r4

    xcall   main
;
```

demoasm.s

This source contains a description of the following routines:

GetEvent: Gets K6[7:4] input port data
InitBusCtrl: Sets bus conditions
1 wait cycle is set for areas 4 through 10; output disable delay time = 0.5 cycles
InitCPUClock: Sets the CPU clock
HLT2OP = 1, 8T1ON = 0, PF1ON = 1
Wait: Generates wait time (in units of 0.1 ms) using the 16-bit timer
LedON, LedOFF: Turns the LED on the DMT33004 board on or off
NMI: NMI processing routine

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; int GetEvent()
;;;
#define K6_INPUT_DATA  0x000402c6

.global GetEvent
    .align 1
GetEvent:
    xld.w    %r4,K6_INPUT_DATA
    ld.ub   %r10,[%r4]                ; read EVENT
    srl     %r10,0x04
    and     %r10,0xf
    ret

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; InitBusCtrl
;;;
#define BUS_CTRL_HEAD  0x00048126

.global InitBusCtrl
InitBusCtrl:
    xld.w    %r4,BUS_CTRL_HEAD
;
    xld.w    %r5,0x0001                ; dis : 0.5, wait : 1
    ld.h    [%r4]+,%r5                ; 0x048126 area 9,10
    xld.w    %r5,0x0001                ; dis : 0.5, wait : 1
    ld.h    [%r4]+,%r5                ; 0x048128 area 7,8
    xld.w    %r5,0x0101                ; dis : 0.5, wait : 1
    ld.h    [%r4]+,%r5                ; 0x04812a area 4,5,6
    ret
```

APPENDIX VERIFYING OPERATION WITH DMT33 BOARDS

```

////////////////////////////////////
;;; InitCPUClock
;;;
#define CPUPOWER_CTRL_HEAD      0x00040140
#define CPUCLOCK_CTRL_HEAD     0x00040150
#define CPUCLOCK_CTRL_PROTECT  0x0004015e

        .align 1
.global InitCPUClock
InitCPUClock:
        xld.w   %r4,CPUCLOCK_CTRL_PROTECT
        xld.w   %r5,0x96
        ld.b    [%r4],%r5
        xld.w   %r4,CPUCLOCK_CTRL_HEAD
        ld.w    %r5,0x05
        ld.b    [%r4],%r5
        ret

////////////////////////////////////
;;; SpkSoftOpen, SpkSoftClose
;;;
#define PRESC16_5              0x0004014c
#define TMCTRL16_5            0x000401a8
#define RELOAD16_5            0x000401aa
#define TIMER16_5_IMASK      0x00040274
#define TIMER16_5_IFLAG      0x00040284

.global Wait
Wait:
        ld.w    %r5,0x10          ; 16bit-mode, Stop
        ld.w    %r6,0x00          ; Prescaler Stop
        xld.b   [TMCTRL16_5],%r5
        xld.b   [PRESC16_5],%r6
        xld.h   [RELOAD16_5],%r12 ; set reload value
;
        xld.w   %r4,TIMER16_5_IFLAG
        bset   [%r4],0x06        ; clear Underflow
;
        xld.w   %r5,0xff          ; Run prescaler 1/2048 0.1 msec
        xld.b   [PRESC16_5],%r5
        ld.w    %r5,0x12          ; Preset(16-bit mode)
        xld.b   [TMCTRL16_5],%r5
        ld.w    %r5,0x11          ; Run(16-bit mode)
        xld.b   [TMCTRL16_5],%r5

Loop:
        btst   [%r4],0x06
        jreq   Loop

        ld.w    %r5,0x10          ; 16bit-mode, Stop
        ld.w    %r6,0x00          ; Prescaler Stop
        xld.b   [TMCTRL16_5],%r5
        xld.b   [PRESC16_5],%r6
        ret

////////////////////////////////////
;;; LED ON and OFF
;;;
#define P0IO 0x402d2          ; P0 I/O port control

.global LedON
LedON:
        xld.w   %r4,0x80
        xld.b   [P0IO],%r4      ; P07 port output
        ret

.global LedOFF
LedOFF:
        xld.w   %r4,0x00
        xld.b   [P0IO],%r4      ; P07 port input
        ret

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; int NMI routine
;;;
.comm NMI_CNT 1

.global NMI
.global ESC
NMI:
    pushn    %r1
ESC:
    ext     NMI_CNT@h
    ext     NMI_CNT@m
    ld.w    %r0,NMI_CNT@l
    ld.b    %r1,[%r0]
    add     %r1,1
    ld.b    [%r0],%r1

    popn    %r1
    reti

```

table.s

This source defines the trap table vectors. SpkIntr0() and LisIntr0() respectively are entered as interrupt vector for underflow of 16-bit timer 5 and the A/D converter interrupt vector.

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Interrupt Vectors
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    .word    Boot           ; 0 Reset
    .word    exception      ; 1 reserved
    .word    exception      ; 2 reserved
    .word    exception      ; 3 reserved
    .word    exception      ; 4 Zero Div.
    .word    exception      ; 5 reserved
    .word    exception      ; 6 Address Error
    .word    NMI            ; 7 NMI
    :
    .word    SpkIntr0       ; 50 16-bit Timer #5-1 underflow
    :
    .word    LisIntr0       ; 64 ADC
    :
exception:
    jp 0

```

vsxdata.vs

This is the assembly source of VSX-compressed voice data created by VOX33 tools. This data file has been created using the make file "vsxdata.mak" in the "voxtool\sample\" directory. For the content of this file, refer to Section 4.2.24, "Executing Tools from a Make File".

Linker command file

The following shows the content of the linker command file that is used to link the sample programs explained here. Because the sample programs are executed in the external RAM of the DMT33004, the start address of the CODE section is set to 0x600000.

When creating a linker command file, note the following:

- The BSS section of the VOX33 library must always be mapped into the internal RAM.
- To speed up operation, the objects executed in the internal memory must be mapped into the U section.

Note: The routine used to transfer code into the internal memory is written in the vsxtop.c top-level functions vsxSpeak() and vsxListen(). This routine is enabled by defining "IRAM_CACHE" when compiling vsxtop.c.

```
-objsym
-w ;
-d ;

; Library path
-l C:\CC33\lib
-l ..\..\lib

;Map set

-code 0x600000
-bss 0x20
-bss 0x6a0000 {vsxdemo.o} ; external RAM

-section ADPCODE
-ucode ADPCODE {..\..\lib\fadpcm16.o ..\..\lib\fadpcm24.o
..\..\lib\fadpcm32.o ..\..\lib\fadpcm40.o}

;Object files

table.o
boot.o
voxcomm.o
demoasm.o
vsxdemo.o
vsxtop.o
ppctop.o

slutil.o

..\..\lib\fadpcm16.o
..\..\lib\fadpcm24.o
..\..\lib\fadpcm32.o
..\..\lib\fadpcm40.o

vsxdata.o

;Voice33 library
vox.lib
sl104.lib
```

A.32 make

The make file "vsxdemo.mak" is provided for use with the VSX sample program. When you have corrected the source, create an executable object file "vsxdemo.srf" by using "vsxdemo.mak".

make execution procedure

1. Make "voxlib\sample\vsx\" the current directory.
2. Enter the command shown below from the DOS prompt:
C:\E0C33\VOX33\VOXLIB\SAMPLE\VSX>make -f vsxdemo.mak

You also can execute make from the work bench wb33. (Refer to the "E0C33 Family C Compiler Package Manual".)

Note: The VOX33 library functions use the CPU's R8 register. Therefore, when linking the VOX33 library to the program, do not use the -gp option (optimization using global pointer/R8) of the instruction extender ext33.

A.4 When Using the DMT33005 Board

The DMT33005 uses the E0C33208 chip, which does not have a D/A converter in the CPU. For this reason, the sample program for the DMT33004 (E0C33A104) described above cannot be used for the DMT33005.

A sample program for the DMT33005 is provided in the "voxlib\smp1208\" directory. Program operation can therefore be verified in the same way as explained for the DMT33004.

Use DSW1 on the DMT33005 in its default settings (PLL x2; 20 to 33 MHz input).

To use the DMT33005 in place of the DMT33004 in the system configuration of Figure A.1.1, choose TM with DMT33AMP jumper switches JP2 and JP3, and DA1/TM1 with JP9. Other jumper switch settings do not need to be changed between the DMT33004 and DMT33005.

Also note that, when creating a program for the E0C33208, the VOX33 library in the "voxlib\lib208\" directory must link to the program. For details, refer to the sample linker command file.

EPSON International Sales Operations

AMERICA

EPSON ELECTRONICS AMERICA, INC.

- HEADQUARTERS -

1960 E. Grand Avenue
El Segundo, CA 90245, U.S.A.
Phone: +1-310-955-5300 Fax: +1-310-955-5400

- SALES OFFICES -

West

150 River Oaks Parkway
San Jose, CA 95134, U.S.A.
Phone: +1-408-922-0200 Fax: +1-408-922-0238

Central

101 Virginia Street, Suite 290
Crystal Lake, IL 60014, U.S.A.
Phone: +1-815-455-7630 Fax: +1-815-455-7633

Northeast

301 Edgewater Place, Suite 120
Wakefield, MA 01880, U.S.A.
Phone: +1-781-246-3600 Fax: +1-781-246-5443

Southeast

3010 Royal Blvd. South, Suite 170
Alpharetta, GA 30005, U.S.A.
Phone: +1-877-EEA-0020 Fax: +1-770-777-2637

EUROPE

EPSON EUROPE ELECTRONICS GmbH

- HEADQUARTERS -

Riesstrasse 15
80992 Muenchen, GERMANY
Phone: +49-(0)89-14005-0 Fax: +49-(0)89-14005-110

- GERMANY -

SALES OFFICE

Altstadtstrasse 176
51379 Leverkusen, GERMANY
Phone: +49-(0)217-15045-0 Fax: +49-(0)217-15045-10

- UNITED KINGDOM -

UK BRANCH OFFICE

2.4 Doncastle House, Doncastle Road
Bracknell, Berkshire RG12 8PE, ENGLAND
Phone: +44-(0)1344-381700 Fax: +44-(0)1344-381701

- FRANCE -

FRENCH BRANCH OFFICE

1 Avenue de l'Atlantique, LP 915 Les Conquerants
Z.A. de Courtaboeuf 2, F-91976 Les Ulis Cedex, FRANCE
Phone: +33-(0)1-64862350 Fax: +33-(0)1-64862355

ASIA

- CHINA -

EPSON (CHINA) CO., LTD.

28F, Beijing Silver Tower 2# North RD DongSanHuan
ChaoYang District, Beijing, CHINA
Phone: 64106655 Fax: 64107320

SHANGHAI BRANCH

4F, Bldg., 27, No. 69, Gui Jing Road
Caohejing, Shanghai, CHINA
Phone: 21-6485-5552 Fax: 21-6485-0775

- HONG KONG, CHINA -

EPSON HONG KONG LTD.

20/F., Harbour Centre, 25 Harbour Road
Wanchai, HONG KONG
Phone: +852-2585-4600 Fax: +852-2827-4346
Telex: 65542 EPSCO HX

- TAIWAN, R.O.C. -

EPSON TAIWAN TECHNOLOGY & TRADING LTD.

10F, No. 287, Nanking East Road, Sec. 3
Taipei, TAIWAN, R.O.C.
Phone: 02-2717-7360 Fax: 02-2712-9164
Telex: 24444 EPSONTB

HSINCHU OFFICE

13F-3, No. 295, Kuang-Fu Road, Sec. 2
HsinChu 300, TAIWAN, R.O.C.
Phone: 03-573-9900 Fax: 03-573-9169

- SINGAPORE -

EPSON SINGAPORE PTE., LTD.

No. 1 Temasek Avenue, #36-00
Millenia Tower, SINGAPORE 039192
Phone: +65-337-7911 Fax: +65-334-2716

- KOREA -

SEIKO EPSON CORPORATION KOREA OFFICE

50F, KLI 63 Bldg., 60 Yoido-Dong
Youngdeungpo-Ku, Seoul, 150-010, KOREA
Phone: 02-784-6027 Fax: 02-767-3677

- JAPAN -

SEIKO EPSON CORPORATION

ELECTRONIC DEVICES MARKETING DIVISION

Electronic Device Marketing Department

IC Marketing & Engineering Group

421-8, Hino, Hino-shi, Tokyo 191-8501, JAPAN
Phone: +81-(0)42-587-5816 Fax: +81-(0)42-587-5624

ED International Marketing Department I (Europe & U.S.A.)

421-8, Hino, Hino-shi, Tokyo 191-8501, JAPAN
Phone: +81-(0)42-587-5812 Fax: +81-(0)42-587-5564

ED International Marketing Department II (Asia)

421-8, Hino, Hino-shi, Tokyo 191-8501, JAPAN
Phone: +81-(0)42-587-5814 Fax: +81-(0)42-587-5110



In pursuit of "**Saving**" Technology, Epson electronic devices.
Our lineup of semiconductors, liquid crystal displays and quartz devices
assists in creating the products of our customers' dreams.
Epson IS energy savings.

EPSON

SEIKO EPSON CORPORATION
ELECTRONIC DEVICES MARKETING DIVISION

■ Electronic devices information on Epson WWW server

<http://www.epson.co.jp>

Issue JULY 1999, Printed in Japan  A