MF1295-01

EPSON

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

TS33 MIDDLEWARE MANUAL

ENERGY SAVING EPSON

SEIKO EPSON CORPORATION

PREFACE

This manual describes the configuration and functions of Text-to-Speech Conversion Middleware TS33 for the E0C33 Family, and explains methods for using this middleware. It is targeted to developers of applications for the E0C33 Family of microcomputers.

CONTENTS

1 Outline of the TS33 Middleware

As voice output middleware for the E0C33 Family, Text to Speech for the E0C33 Family (TS33) converts text data into voice data using an E0C33 Family chip, then outputs voice by expanding word data compressed in VSX2 format. Each function is provided as a library function that can be used after linking with the target program.
The product also includes PC tools for TS/VSX2 ROM data generation and evaluation.
Although the sound quality falls short of that of general recorded voice data, the voice data format offers the advantage of requiring relatively little memory space. TS33 middleware is best suited to applications such as portable devices with voice output functions and automatic vending machines.

Its main features are listed below.

- Support for 16-bit programmable timer built-in E0C33 models (including the E0C33208) operating at CPU clock rates of 40 MHz or higher.
  On-chip voice VSX2 compression requires an A/D converter.
- Support for E0C33 models (E0C33A104, E0C33208, etc.) operating at 20 MHz, if VSX format (8-kHz sampling) data is used (installer option).
- Text conversion for each registered word
- Allows adjustment of voice pitch and length.
- VSX2 compression and expansion technologies
  An exclusive Seiko Epson technology, VSX2 is ADPCM-based time-base and silent compression. With 11-kHz, 16-kHz, and 22-kHz sampling options, it provides a compact voice data form and supports reproduction at altered talking speeds. It also allows the independent use of VSX2 compression and expansion technologies for voice input and output.
  Note) The VSX2 format data is incompatible with the VOX33 library VSX format (8-kHz sampling).

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 TS33 Package

The contents of the TS33 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 TS33 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. No microphones or A/D converters are required if only the text-voice conversion output function is used, without the voice VSX2 compression input function.
Note that the TS33 library uses two channels of the 16-bit programmable timer on the E0C33 chip. For voice VSX2 compression input, it also uses an A/D converter channel on the E0C33 chip and an additional 16-bit programmable timer channel. It also uses some of the internal RAM to accelerate operation.

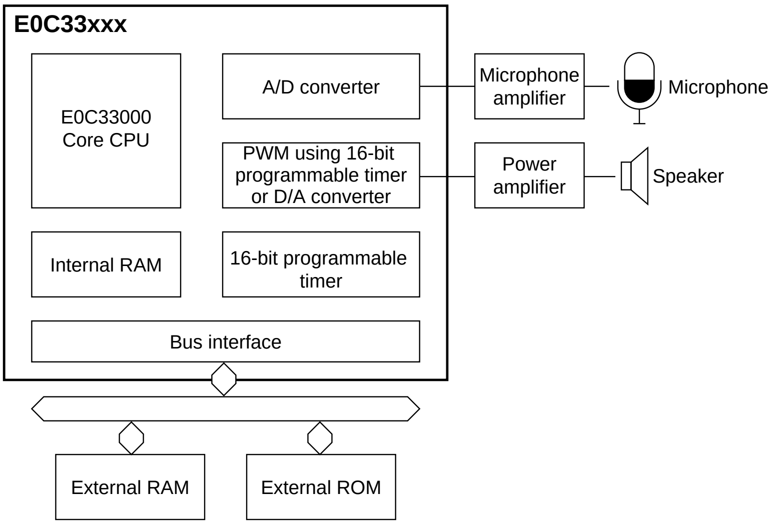


Figure 1.2.1 Hardware Configuration of Voice Input/Output System

The TS33 library consists of TS functions for text-voice conversion and output and VSX2 functions for voice VSX2 compression and expansion. Although the TS function uses the VSX2 function to output voice data, the VSX2 function may be used independently. 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 lower library functions directly from the user program.

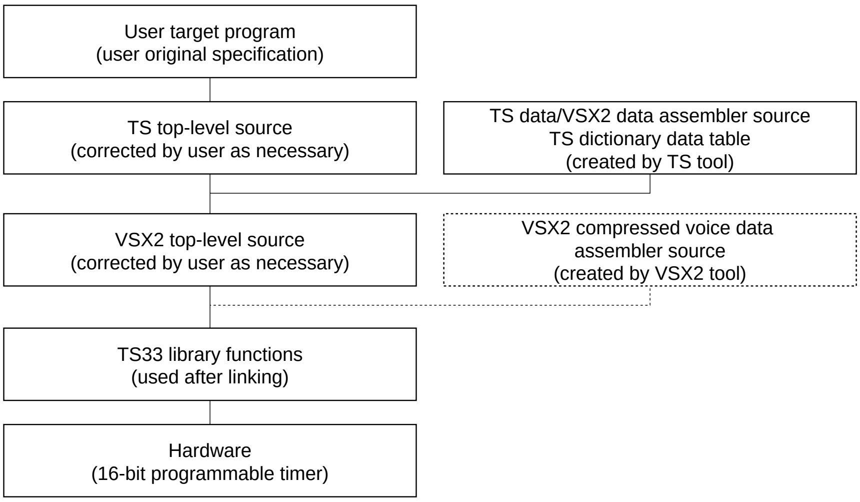

User target program (user original specification)
TS top-level source (corrected by user as necessary)
TS data/VSX2 data assembler source
TS dictionary data table
(created by TS tool)
VSX2 top-level source (corrected by user as necessary)
VSX2 compressed voice data assembler source (created by VSX2 tool)
TS33 library functions (used after linking)
Hardware (16-bit programmable timer)

Figure 1.2.2 Software Configuration of Voice Input/Output System

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

1.3 TS33 Tool

The TS33 tools can be run on a personal computer to generate TS ROM and VSX2 voice ROM data to be stored on the E0C33 Family chip as well as to evaluate text-voice conversion output and VSX2 voice compression and voice processing performance. All of these tools can be run under Windows 95, Windows NT 4.0, or higher versions. Refer to Section 3, "Software Development Procedure" for information on generating ROM data. Refer to Section 4, "Tool Reference" for details on TS33 tools.

2 Installation

This section explains the operating environment for the TS33 tools and how to install the TS33 middleware.

2.1 Operating Environment

Software development and ROM data generation/evaluation using TS33 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 32MB or more of RAM is recommended. Installation requires a CD-ROM drive.

Display

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

Sound card, sound editor

DAT and a digital sound card are recommended for the generation of VSX2 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 that supports an 8- to 22-kHz sampling 16-bit monaural sound. Choose a sound editor that can handle the preceding PCM data and save to a file.

System software

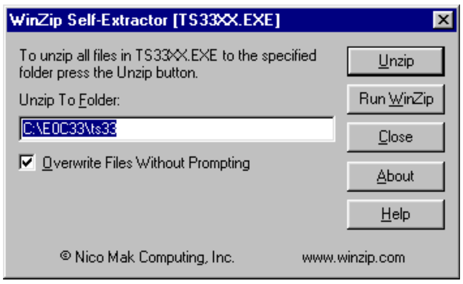
TS33 tools run under Microsoft® Windows®95, Windows NT®4.0, or higher versions (in English or Japanese).

Other requirements

"E0C33 Family C Compiler Package" is required for software development.

2.2 Method of Installation

The TS33 library and TS33 tools are supplied on CD-ROM. Open the self-extracting file on the CD-ROM named "ts33vXX.exe" to install the TS33 library and TS33 tools in your computer. (The XX in this file name denotes a version number. For Version 1.0, for example, the file is named "ts33v10.exe".)
Double-click on "ts33vXX.exe" to start installation. The dialog box shown below appears.


WinZip Self-Extractor [TS33XX.EXE]
Unzip
Run WinZip
Close
About
Help
© Nico Mak Computing, Inc.
www.winzip.com

To unzip all files in TS33XX.EXE to the specified folder press the Unzip button.

Unzip To Folder: C:\E0C33\ts33

☑ Overwrite Files Without Prompting

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)

tstool ..... TS33 tool directory

readme.txt	TS33 tool supplementary explanation, etc. (in English)
readmeja.txt	TS33 tool supplementary explanation, etc. (in Japanese)
param.txt	vsx2param.exe supplementary explanation, etc. (in English)
paramja.txt	vsx2param.exe supplementary explanation, etc. (in Japanese)

bin ..... TS33 tools

tb33.exe	Work bench for evaluation of TS-converted voice data
jtxt2pts.exe	TXT-to-PTS file conversion program
pts2ts.exe	PTS-to-TS file conversion program
ts2pcm.exe	TS-to-PCM file conversion program
ts2bin.exe	TS-to-binary file conversion program
tstbl.exe	Installed dictionary data table-generation program
vsx2param.exe	VSX2 compression/expansion evaluation program
vsx2cmprs.exe	VSX2 compression program
vsx2dec.exe	VSX2 decoding program
vsxcmprs.exe	VSX compression program *1
vsxdec.exe	VSX decoding program *1
ppccmprs.exe	Packed PCM compression program
ppcdec.exe	Packed PCM decoding program
bin2s.exe	Binary-to-assembly source conversion program
bdmp.exe	Binary file dumping program
dct_cnv.exe	Sampling rate conversion program
pcm_norm.exe	PCM normalization program
voxflt.exe	High-pass filter program
ccap.exe	Tool message filing tool

dict ..... Dictionary directory *1
Dictionary voice 16-bit PCM file

data ..... Evaluation dictionary data directory *1
This file is used to evaluate VSX2-compressed, expanded, and processed voice data in the dict directory.

sample ..... TS sample directory *2
Sample of text and batch files

smplvsx2 ..... VSX2 sample directory *2
Voice, batch file, and make file samples

src ..... Source directory
Published tool source files

tslib\ ..... TS33 library-related directory

readme.txt	TS33 library supplementary explanation, etc. (in English)
readmeja.txt	TS33 library supplementary explanation, etc. (in Japanese)

lib208\ ..... TS33 library for E0C33208 directory

vox208.lib	TS33 library for E0C33208
sl208.lib	Voice input / output library for E0C33208
vsx2.o	VSX2 object
fadpcm16.o, fadpcm24.o, fadpcm32.o, fadpcm40.o, vsxgcp.o	Objects retrieved from vox208.lib to accelerate operation

include\ ..... TS33 library function header file directory

ts.h	TS header file
vsx2.h	VSX2 header file
voxcomn.h	Library common 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

ts2top.c	TS top-level function
vsx2top.c	VSX2 top-level function
voxcomn.c	Library common function
slutil.c	SPEAK and LISTEN utility functions

smpl208\ ..... DMT33005 sample program directory *2

hardsrc\ ..... Hardware dependent source directory *2

*1 The dict and data directories are installed with 16-kHz sampling data as standard specification. You can also install 8-kHz sampling (tsOP1vXX.exe) and 22-kHz sampling (tsOP2vXX.exe) files. Run the "tsOP1vXX.exe" file to install tools for generating VSX format data. Except for the English samples, the dictionary data installed as part of the standard specification is for Japanese voice output and can be used only for Japanese output.

*2 Refer to "readme.txt" or "readmeja.txt" in the "tstool\\" or "tslib\\" directory for the composition and the method for using sample programs.

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. The basic development flow is shown below.

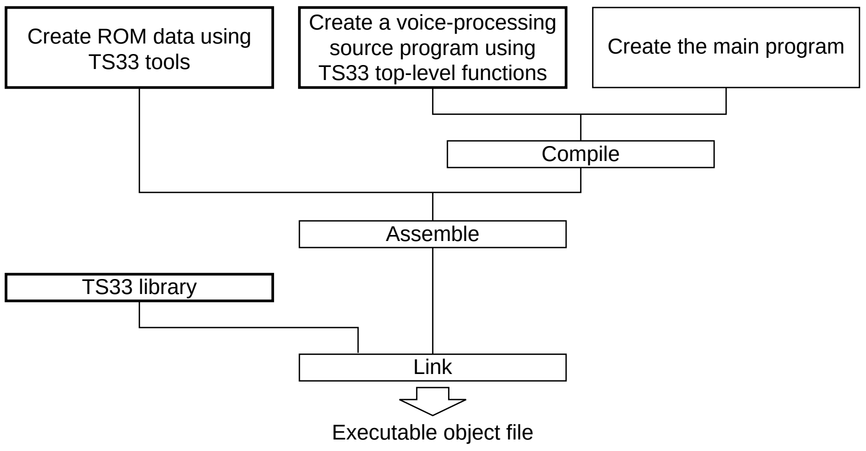

Create ROM data using TS33 tools
Create a voice-processing source program using TS33 top-level functions
Create the main program
Compile
Assemble
TS33 library
Link
Executable object file

Figure 3.1 E0C33 Voice-Processing Software Development Procedure

1) Use TS33 tools to generate the assembly source file for the reproduction text data and voice dictionary data. To write playback-only VSX2 voice data not derived from text-voice to ROM, use VSX2 tools to generate a voice ROM data assembly source file.
2) Create a user program. For voice processing, use the top-level function provided in the TS library. The source of the ROM data generated in step 1 can be included in the user program source.
3) Compile and assemble the source program.
4) Link the objects generated in step 3 with the TS33 library. This generates the object in executable form.

3.1 Generating TS ROM Data

To perform TS data conversion output on the E0C33 chip, use the TS33 tools to generate text, dictionary data, and other required data.
Figure 3.1.1 shows the procedure for generating ROM data and the configuration of TS33 tools.

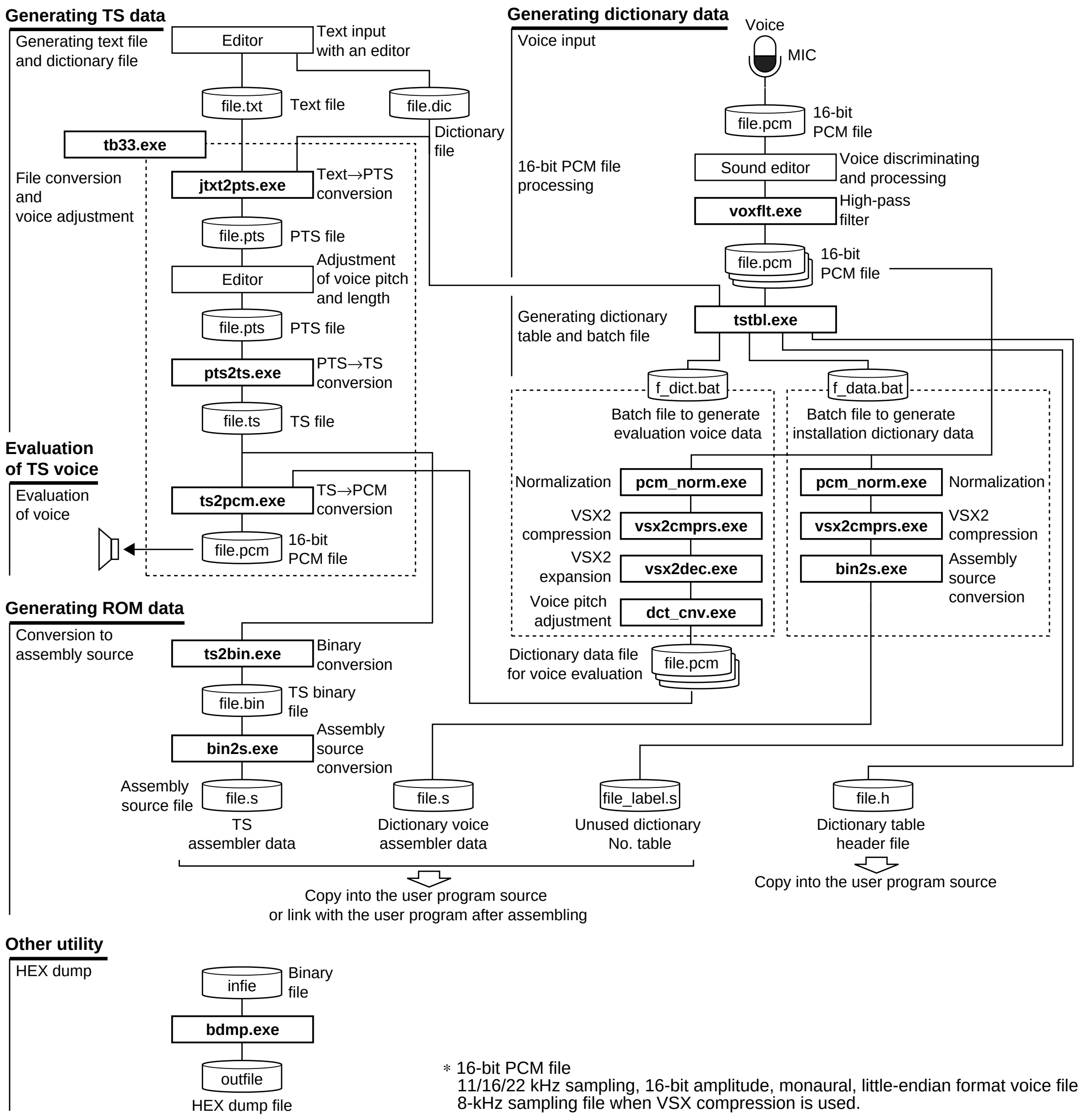


Figure 3.1.1 Flow Chart for Generating TS ROM Data

Only the methods for using TS33 tools are outlined here. For details, refer to Section 4, "Tool Reference".
In the explanation that follows, assume that "tstool\sample\" is set as the current directory, and that PATH is set in the "tstool\bin\" directory.

Example:
```
DOS>CD e0c33\ts33\tstool\sample
DOS>PATH c:\e0c33\ts33\tstool\bin
```

Text-voice conversion output works from a dictionary file wherein the correspondence between each word and voice data is mapped, as explained further below. For text for voice output, each word is converted into voice data with the dictionary file.
As voice data occupies large memory space, it is compressed in VSX2 (or VSX) format and expanded for reproduction in real-time.

3.1.1 Input and Processing of TS Voice Data

List all the words required for voice output. We recommend recording even phonemes and words not currently used for the text being generated, but that may be used in the future in the same environment.

Input voice through a microphone to generate the source voice data (16-bit PCM file: 16-kHz sampling*, 16-bit monaural voice file). Prepare data to ensure the best possible quality. Use digital sampling with DAT, if possible.
* VSX2 supports sampled at 11.025, 16, and 22.050 kHz. The optional VSX format supports only 8-kHz sampling.

Next, discriminate words from the sampled voice data and process the data through a filter. Use a commercial sound editor for data discriminating. Some of the sound editors are also capable of performing filter processing, but may degrade sound quality.

High-pass filtering

TS33 tools have a high-pass filter program "voxflt.exe" that allows you to specify the cut-off frequency. 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 60 se.pcm seH.pcm
```

In this example, the 16-kHz sampling "se.pcm" is filtered with cut-off frequency of a 120Hz (-l 60) to generate "seH.pcm".

3.1.2 Generating a Dictionary File

Using a general-purpose editor, generate a dictionary file to match each word to the corresponding voice data. The dictionary file is in the following text format.
Example: english.dic

```
[user]  Please add user dictuonary from the dictionary No.0xc8 to No.0xfe.
c8 ..\dict\the.pcm  [the]
c9 ..\dict\fami.pcm  [family]
ca ..\dict\txt2sp.pcm  [texttospeach]
cb ..\dict\demo.pcm  [demonstration]
cc ..\dict\three.pcm  [three]
cd ..\dict\this.pcm  [this]
ce ..\dict\is.pcm  [is]
cf ..\dict\eoc.pcm  [e0c]
```

The contents of each line are as follows.
<Dictionary No.> <PCM file name> <[Word]>

The <Dictionary No.> is one of 54 hexadecimal numbers from "c8" to "fe", which must be in ascending order, though not necessarily continuous. You cannot use the values "00" to "c7", since they are reserved for output in Japanese.
Specify the <PCM file name> including the path, absolute or relative. For "english.dic", specify the relative path from the "tstool\samle\" directory. This assumes that a series of jobs are performed in the directory.
<[Word]> is the word to be converted into the PCM file specified on the line. Specify by sandwiching it between brackets [ ].

3.1.3 Generating Installation Dictionary Data

From the dictionary file generated as a text file, use "tstbl.exe" to generate the data table to be included in the program, then perform VSX2 compression of the registered PCM file to generate a batch file for inclusion in the product.

Example: `DOS>tstbl -c24 -t2 -s 20 english.dic vsxdataE`

In this example, the following files are generated from the dictionary file "sample.dic".
The symbols -c24, -t2, and -s20 are parameters that determine the compression ratio of PCM files registered in the dictionary file into VSX2/VSX format. The symbol -c24 specifies a compression corresponding to 24 kbps; -t2 compresses data 50% in the timebase direction; and -s20 specifies the threshold value for silence judgement. For other settings, refer to "4.2.5 tstbl.exe".

<file>.h

This is a header file that defines the externally referenced dictionary files, 16-bit timer setting and dictionary data table. Include this file in the user program.

Example: vsxdataE.h

```
#define VSX_CMP VSX_COMPRESS_24K      Definition of the constant specifying the compression ratio

extern unsigned char ts00[];          Statement of the externally referenced dictionary file
extern unsigned char ts01[];
extern unsigned char ts02[];
extern unsigned char ts03[];
            :
extern unsigned char tscc[];
extern unsigned char tscd[];
extern unsigned char tsce[];
extern unsigned char tscf[];

const short tsFtbl[] = {              16-bit timer setting table
    0x3133,
    0x4,
    0x119,
    0x10d,
    0x100,
    0xe7,
    0x0
};

const int tsPtbl[] = {                Dictionary data table
    0x3233,
    0xd0,
    (int)&ts00[0],
    (int)&ts01[0],
    (int)&ts02[0],
    (int)&ts03[0],
         :
    (int)&tscc[0],
    (int)&tscd[0],
    (int)&tsce[0],
    (int)&tscf[0],
    0x0
};
```

<file>_data.bat

A batch file that normalizes and compresses all PCM files registered in the dictionary file in VSX2/VSX format, and then generates the assembly source file to install these files into the program.

Example: vsxdataE_data.bat

```
set tspath=..\bin\
%tspath%pcm_norm -r 65 ..\dict\the.pcm tmp.pcm
%tspath%vsx2cmprs  -c24  -t2 -s 20  tmp.pcm tmp.vsx
%tspath%bin2s -l tsc8 tmp.vsx > vsxdataE.s
%tspath%pcm_norm -r 65 ..\dict\fami.pcm tmp.pcm
%tspath%vsx2cmprs  -c24  -t2 -s 20  tmp.pcm tmp.vsx
%tspath%bin2s -l tsc9 tmp.vsx >> vsxdataE.s
                        :
%tspath%pcm_norm -r 65 ..\dict\eoc.pcm tmp.pcm
%tspath%vsx2cmprs  -c24  -t2 -s 20  tmp.pcm tmp.vsx
%tspath%bin2s -l tscf tmp.vsx >> vsxdataE.s
del tmp.vsx
del tmp.pcm
```

Modify the path (..\bin\) to the TS33 tool if necessary. For details on tools run by the batch file, refer to Section 4, "Tool Reference".

This batch file generates the following assembly source file.

Example: vsxdataE.s

```
            .global tsc8
            .align  2
tsc8:
            .byte   0x53 0x22 0x2e 0x9f 0xff 0xff 0xdb 0x5b
            .byte   0xb5 0x92 0xc9 0x64 0x30 0x14 0xc8 0x24
                                 :
            .byte   0x92 0x40 0x01 0x01 0x01 0x01 0x00
; total 799 bytes data

            .global tsc9
            .align  2
tsc9:
            .byte   0x53 0x22 0x55 0x1f 0xff 0xfe 0xfb 0x6b
            .byte   0x6d 0xb6 0xed 0x6d 0xb4 0x4b 0xa9 0x00
                                 :
            .global tscf
            .align  2
tscf:
            .byte   0x53 0x22 0x49 0x8d 0x6b 0xea 0xb4 0x79
            .byte   0x69 0x93 0x6c 0x40 0xde 0xc3 0x2e 0xb2
                                 :
            .byte   0x01 0x00
; total 1746 bytes data
```

The label attached to each data is "ts" + dictionary No., the title used by the header file for the statement of external reference. Copy this file in the user program, or link after assembling.

<file>_label.s

This file describes only the labels of unused dictionary numbers in the dictionary file to generate a table that includes unused dictionary numbers in the header file. This file is used to prevent linking errors. Copy this file in the user program along with the above assembly source, or link after assembling.

Example: vsxdataE_label.s

```
            .global ts00
ts00:
            .global ts01
ts01:
            .global ts02
            :
            .global tsc6
tsc6:
            .global tsc7
tsc7:
```

<file>_dict.bat

A batch file for generating a voice file to evaluate text-voice conversion output on a PC.

Example: vsxdataE_dict.bat

```
set tspath=..\bin\
%tspath%pcm_norm -r 65 ..\dict\the.pcm ..\data\tmp.pcm
%tspath%vsx2cmprs  -c24  -t2 -s 20  ..\data\tmp.pcm ..\data\tmp.vsx
%tspath%vsx2dec ..\data\tmp.vsx  ..\data\tsc8_02_norm.pcm
%tspath%vsx2dec -s15 ..\data\tmp.vsx  ..\data\tsc8_02_s15.pcm
%tspath%vsx2dec -s20 ..\data\tmp.vsx  ..\data\tsc8_02_s20.pcm
%tspath%vsx2dec -f15 ..\data\tmp.vsx  ..\data\tsc8_02_f15.pcm
%tspath%dct_cnv 100 111 ..\data\tsc8_02_norm.pcm  ..\data\tsc8_00_norm.pcm
%tspath%dct_cnv 100 111 ..\data\tsc8_02_s15.pcm  ..\data\tsc8_00_s15.pcm
%tspath%dct_cnv 100 111 ..\data\tsc8_02_s20.pcm  ..\data\tsc8_00_s20.pcm
%tspath%dct_cnv 100 111 ..\data\tsc8_02_f15.pcm  ..\data\tsc8_00_f15.pcm
%tspath%dct_cnv 100 105 ..\data\tsc8_02_norm.pcm  ..\data\tsc8_01_norm.pcm
%tspath%dct_cnv 100 105 ..\data\tsc8_02_s15.pcm  ..\data\tsc8_01_s15.pcm
%tspath%dct_cnv 100 105 ..\data\tsc8_02_s20.pcm  ..\data\tsc8_01_s20.pcm
%tspath%dct_cnv 100 105 ..\data\tsc8_02_f15.pcm  ..\data\tsc8_01_f15.pcm
                                     :
%tspath%dct_cnv 100 91 ..\data\tscf_02_norm.pcm  ..\data\tscf_03_norm.pcm
%tspath%dct_cnv 100 91 ..\data\tscf_02_s15.pcm  ..\data\tscf_03_s15.pcm
%tspath%dct_cnv 100 91 ..\data\tscf_02_s20.pcm  ..\data\tscf_03_s20.pcm
%tspath%dct_cnv 100 91 ..\data\tscf_02_f15.pcm  ..\data\tscf_03_f15.pcm
del ..\data\tmp.vsx
del ..\data\tmp.pcm
```

Modify the path (..\bin\) to the TS33 tool if necessary. For details on the tools run by the batch file, refer to Section 4, "Tool Reference".

This batch file performs VSX2/VSX-compression/expansion of all dictionary data (individual PCM data), and generates a file of voices of different pitches and lengths. The file thus generated is needed to evaluate voice converted from text on a PC with "ts2pcm.exe" and "tb33.exe".

3.1.4 Generating Text Data

To generate a text format file storing text to be converted into voice, input the registered words enclosed with brackets [ ], as shown below.

Example: sampleE.txt

```
   [this] [is] [the] [e0c] [three][three] [family] [texttospeach] [demonstration]
```

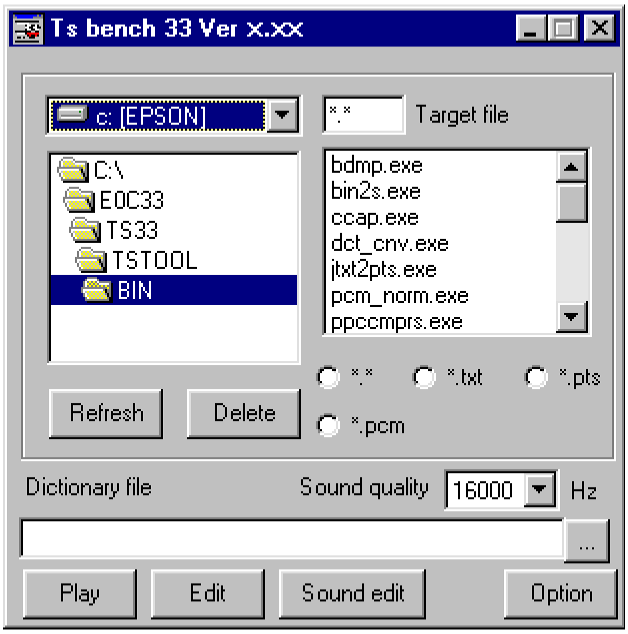
Spaces are converted into silent data.
Words to be described in the text file should be registered in the dictionary file.
If you describe a word that is not in the dictionary, a warning will be issued when the text file is converted into a PTS file with "jtxt2pts.exe", and the word will be ignored.

3.1.5 Voice Evaluation and Adjustment

Use "tb33.exe" to convert the text data and to reproduce the generated voice file on a PC. This is the simplest method of evaluation. "tb33.exe" is a Windows GUI application. Double-click its icon to launch it.

tb33.exe

The following window is displayed.



Target file: *.*

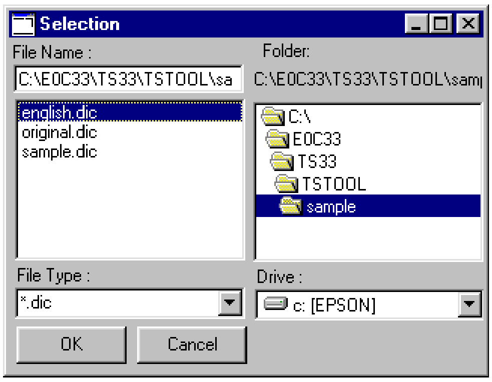
C:\
E0C33
TS33
TSTOOL
BIN

bdmp.exe
bin2s.exe
ccap.exe
dct_cnv.exe
jtxt2pts.exe
pcm_norm.exe
ppccmprs.exe

Sound quality: 16000 Hz

[Main] window

Follow the steps given below to reproduce the voice of the generated "sampleE.txt".



Selection

File Name : C:\E0C33\TS33\TSTOOL\sa

Folder: C:\E0C33\TS33\TSTOOL\sam

english.dic
original.dic
sample.dic

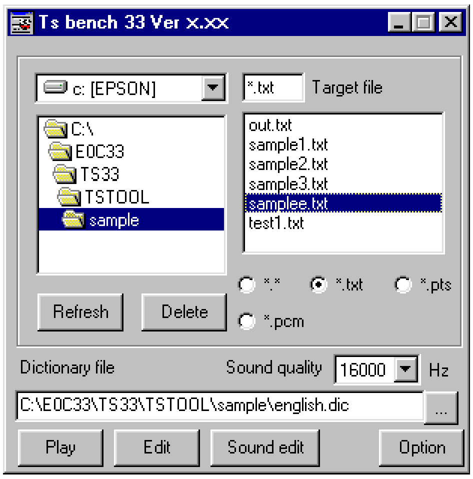
C:\
E0C33
TS33
TSTOOL
sample

File Type : *.dic

Drive : c: [EPSON]

OK Cancel

1) Click the [...] button to the right of the [Dictionary file] text box. In the dialog box that appears, select the dictionary file "english.dic".



Ts bench 33 Ver x.xx

c: [EPSON]

*.txt Target file

C:\
E0C33
TS33
TSTOOL
sample

out.txt
sample1.txt
sample2.txt
sample3.txt
samplee.txt
test1.txt

. [✓ *.txt] *.pts *.pcm

Refresh Delete

Dictionary file

Sound quality 16000 Hz

C:\E0C33\TS33\TSTOOL\sample\english.dic ...

Play Edit Sound edit Option

2) Select the sample folder from the directory list box and "sampleE.txt" from the file list box.

3) Click the [Play] button.

"tb33.exe" invokes and runs the following tools and reproduces the final PCM file.

1) jtxt2pts.exe	Text file-to-PTS file conversion
2) pts2ts.exe	PTS file-to-TS file conversion
3) ts2pcm.exe	TS file-to-PCM file conversion

If you are using a dictionary to which data is added or newly-generated data, execute the batch file ("xxx_dict.bat" generated by "tstbl.exe") shown in Section 3.1.3 to generate PCM data for evaluation before the operation described above.

"jtxt2pts.exe" converts the text file into a PTS file. You can also run this tool from the DOS prompt.
Example: `DOS>jtxt2pts 2 english.dic sampleE.txt sampleE.pts`

In this example, "sampleE.txt" is converted into "sampleE.pts". As the other parameter, specify the dictionary "english.dic" to be used. The parameter value "2" specifies a silent period of 32 msec to be inserted between letters or words to delimit each letter during conversion into Japanese. If you run the tool from the DOS-prompt, you can also specify "0" (no silent period), "1" (16 msec), or "3" (48 msec).

The following PTS files are generated through the operation described above.

Example: sampleE.pts

```
cd   norm   norm   norm   s2     //[this]
ff   10                          //
ce   norm   norm   norm   s2     //[is]
ff   10                          //
c8   norm   norm   norm   s2     //[the]
ff   10                          //
cf   norm   norm   norm   s2     //[e0c]
ff   10                          //
cc   norm   norm   norm   s2     //[three]
cc   norm   norm   norm   s2     //[three]
ff   10                          //
c9   norm   norm   norm   s2     //[family]
ff   10                          //
ca   norm   norm   norm   s2     //[texttospeach]
ff   10                          //
cb   norm   norm   norm   s2     //[demonstration]
```

Each line of the PTS file contains the following.

<Dictionary No.> <Loudness> <Length of sound> <Pitch of sound> <Length of silent period> //<Original text>

<Dictionary No.>

Dictionary data No. corresponding to the <Original text> in the specified dictionary file ("english.dic" in this example)

* ff is the reservation code denoting the silence inserted with the space key. The following numeric value indicates the length of the silent period in 16 msec units, up to a maximum of 127 (2,032 msec). "ff 10" (default) indicates a silent period of 160 msec.

<Loudness>

Reproduces sound at:

hl4	1.4 times the standard volume
norm	The standard volume
l07	0.7 times the standard volume
l05	0.5 times the standard volume

<Length of sound>

Reproduces sound with length:

s20	2 times the standard length (twice slower)
s15	1.5 times standard length (1.5 times slower)
norm	The standard length
f15	1/1.5 times standard length (1.5 times faster)

<Pitch of sound>

Reproduces sound at a pitch:

l2	10% lower than standard
l1	5% lower than standard
norm	The standard pitch
h1	10% higher than standard

<Length of silent period>

Length of the silent period to delimit sound on the next line

Insert a silent period of:

s0	0 msec (no silent period)
s1	16 msec
s2	32 msec
s3	48 msec

//<Original text>

Words described in the text file before conversion

When a text file is converted with "jtxt2pts.exe", <Loudness>, <Pitch of sound> and <Length of sound> are set at "norm", and <length of silent period> is set to the length specified by the parameter on the command line.
The PTS file thus generated is used to manually modify the preceding parameters.

Modify the parameters of each word in the PTS file when necessary, and reproduce the sound again to check the results.
To modify the PTS file (text file), double-click the "sampleE.pts" in the file list box, or select the file and click the [Edit] button. The Windows "Notepad" starts to open the file, enabling easier modification.
The default editor is set in the "Notepad". You can set a preferred application as the default editor in the [Option] window. (Click the [Option] button to display the window.)
To reproduce a sound from a PTS file, select "sampleE.pts" from the file list box and click the [Play] button.

For processing with "tb33.exe" and details on other tools, refer to Section 4, "Tool Reference".

3.1.6 Conversion of TS Data into an Assembly Source File

Use the batch file "tstool\sample\ts.bat" to convert the PTS file thus generated into an assembly source file.
Example: `DOS>ts sampleE`

<ts.bat>

```
set tspath=..\bin\
..\bin\pts2ts %1.pts %1.ts
..\bin\ts2bin %1.ts %1.bin
..\bin\bin2s -l %1 %1.bin > %1.s
```

"ts.bat" converts the specified PTS file as required and generates the following assembly source file. For details on tools run in the batch file, refer to Section 4, "Tool Reference".
Example: sampleE.s

```
             .global sampleE
             .align  2
sampleE:
             .byte   0x33 0x30 0xcd 0x0a 0xff 0x10 0xce 0x0a
             .byte   0xff 0x10 0xc8 0x0a 0xff 0x10 0xcf 0x0a
             .byte   0xff 0x10 0xcc 0x0a 0xcc 0x0a 0xff 0x10
             .byte   0xc9 0x0a 0xff 0x10 0xca 0x0a 0xff 0x10
             .byte   0xcb 0x0a 0xff 0xff 0xff 0xff
; total 38 bytes data
```

The source file will be generated with an input file name as a global symbol. You can change the symbol name with option "-l *symbol*" in "bin2s.exe". Use this symbol to access data from the user program.
Copy this file into the user program, together with the assembly source generated with "tstbl.exe" (refer to Section 3.1.3), or link after assembling.

3.2 Generating Voice ROM Data Using VSX2 Tools

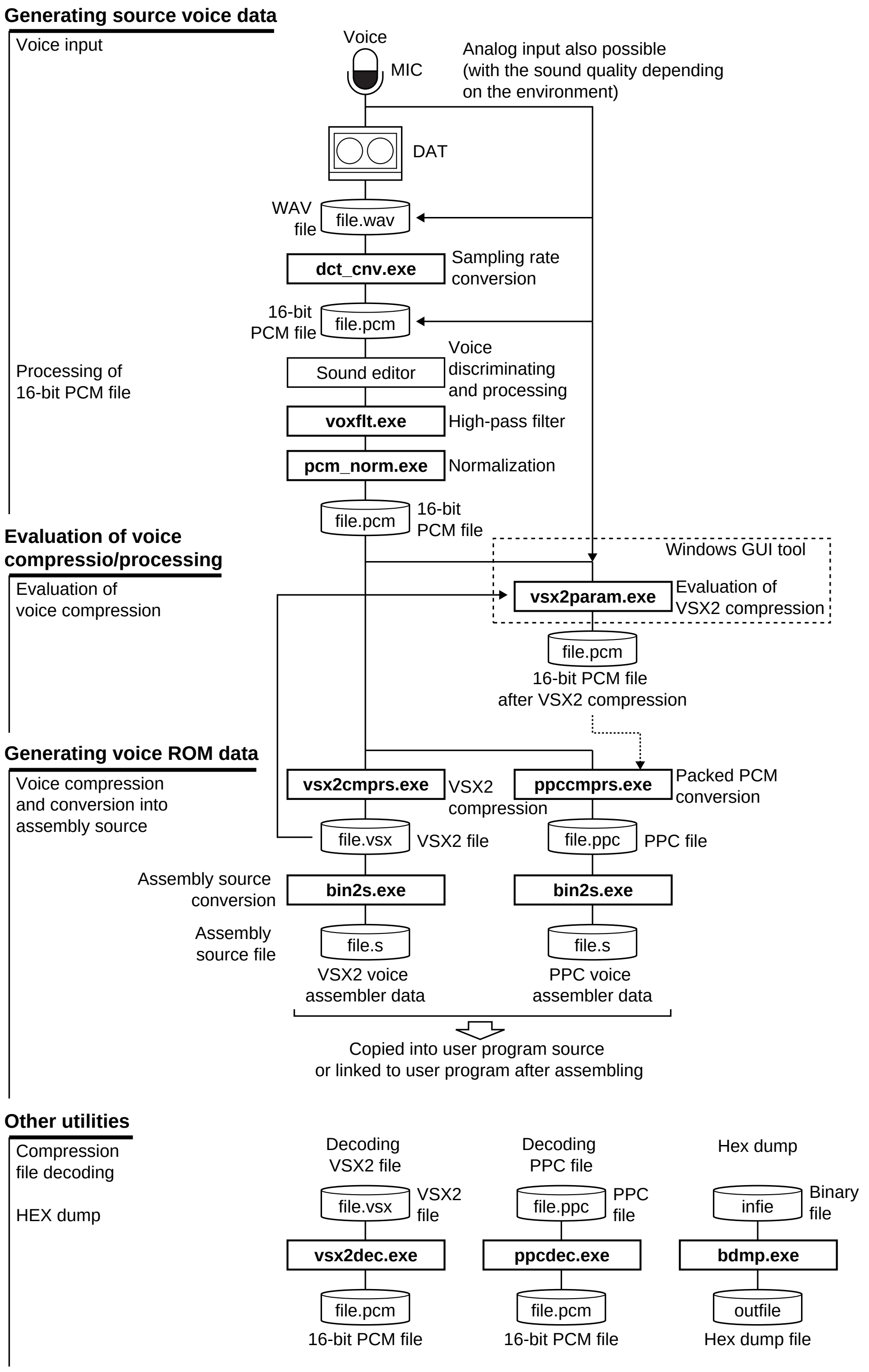
VSX2 is Seiko Epson's exclusive voice data format based on the ADPCM format and features high compression ratio using timebase compression and silent part compression. It supports sampling at 11.025, 16, and 22.050 kHz. In contrast, the conventional VSX format supports 8-kHz sampling only.

Note: The VSX2 compression format is incompatible with the VSX format (also included in the VOX33 middleware), which is a TS33 tool option.

A method for generating VSX2 voice data to be reproduced on the E0C33 chip is given below. This procedure is not required for voice data used for text-voice conversion output.
Figure 3.2.1 shows the procedure for generating voice ROM data and the configuration of VSX2 tools.

Only the methods for using VSX2 tools are outlined here. For details, refer to Section 4,"Tool Reference".
In the following explanation, "se.pcm" in the "tstool\smplvsx2\" directory is used as the source voice file. In the explanation that follows assumes that "tstool\smplvsx2\" is set as the current directory, and that PATH is set in the "tstool\bin\" directory.

Example:
```
DOS>CD e0c33\ts33\tstool\smplvsx2
DOS>PATH c:\e0c33\ts33\tstool\bin
```



File types	
16-bit PCM file (.pcm)	11/16/22 kHz, 16-bit amplitude, monaural, little endian format voice file 8-kHz sampling data when VSX compression is used
WAV file (.wav)	48 kHz, 16-bit amplitude, monaural voice file (file not compressed by Windows ADPCM)
VSX2 file (.vsx)	VSX2-compressed voice file
PPC file (.ppc)	Packed PCM file

Figure 3.2.1 Flow Chart for Generating Voice ROM Data

3.2.1 Preparing Voice Data

Using a microphone, generate the source voice data (16-bit PCM file: 16-kHz sampling*, 16-bit monaural voice file). Prepare data with the highest sound quality possible. Digital sampling using a DAT is recommended.
* VSX2 also supports data sampled at 11.025 and 22.050 kHz.

Down-sampling a WAV file

When you have generated a 48-kHz-sampling WAV file, use "dct_cnv.exe" to down-sample it to the preceding sampling rate and convert it into a 16-bit PCM file. Execute this tool from the DOS prompt.
Example: `DOS>dct_cnv 140 80 sample.wav sample.pcm`

In this example, "sample.wav" is down-sampled at 16 kHz ($140 = 48 \times 5 \rightarrow 80 = 16 \times 5$) to generate "sample.pcm".
Commercially available sound editors can also be used for this processing, but care must be taken not to degrade sound quality.

3.2.2 Preprocessing 16-bit PCM Data

Next, discriminate the actually used part from the sampled voice data, and preprocess it by level adjustment and filtering. Use a commercially available sound editor for discrimination processing and level adjustments.

High-pass filtering

TS33 tools have a high-pass filter program "voxflt.exe" that allows you to specify the cut-off frequency. 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.
Example: `DOS>voxflt -l 60 se.pcm seH.pcm`

In this example, the 16-kHz sampling "se.pcm" is filtered with cut-off frequency of a 120 Hz (-l 60) 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 source voice is compressed, sound quality after compression may be degraded. For this reason, adjust the amplitude of the source voice to or below 90% of the maximum value. Use "pcm_norm.exe" for this processing.
Example: `DOS>pcm_norm seH.pcm seN.pcm`

In this example, "seH.pcm" is adjusted so that the amplitude is 90% or below 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 VSX2 tools.

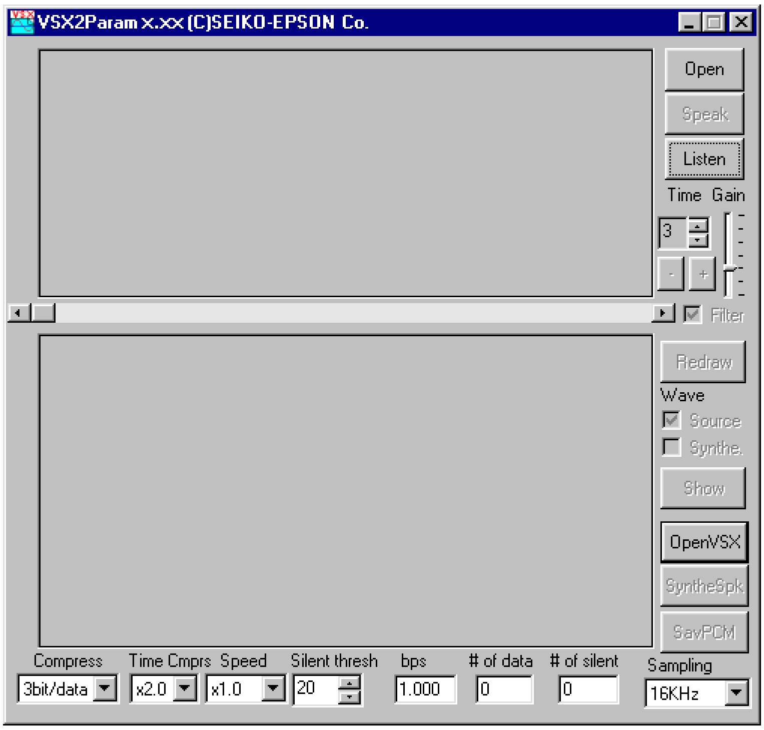
3.2.3 Evaluating Compression and Talking Speed/Tone Pitch Conversion

Before generating voice ROM data, voice compression/expansion can be evaluated on a PC with the VSX2 compression evaluation tool "vsx2param.exe". For details on the tool, refer to Section 4.3.13, "vsx2param.exe".

(1) Starting vsx2param.exe

vsx2param.exe

Double-click on the "vsx2param.exe" icon to start the tool. To quit the tool, click on the [Close] button on the title bar.
When "vsx2param.exe" starts, the [VSX2Param] window appears.



[VSX2Param] window

(2) Selecting the sampling rate

In the [Sampling] combo box, select a sampling rate from the following four categories. This parameter determines the sound quality. When loading a PCM data, use the sampling rate of the file.

8kHz	(for PCM data reproduction)
11.025kHz	(for VSX2 compression)
16kHz	(for VSX2 compression)
22.05kHz	(for VSX2 compression)

(3) Entering voice data

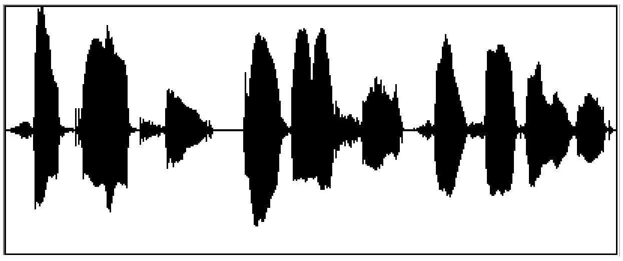
Loading a 16-bit PCM data (sampled at 11.025, 16, and 22.05 kHz)

Click 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 then 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.

(4) Choosing compression ratio

Using the [Compress] combo box, choose the desired compression ratio from the following four.
2bit/data (Compresses equivalent to 16 kbps.)
3bit/data (Compresses equivalent to 24 kbps; default)
4bit/data (Compresses equivalent to 32 kbps)
5bit/data (Compresses equivalent to 40 kbps)

Using the [Time Cmprs] combo box, choose a compression ratio in the timebase direction.
×1.0 (same effect as 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)

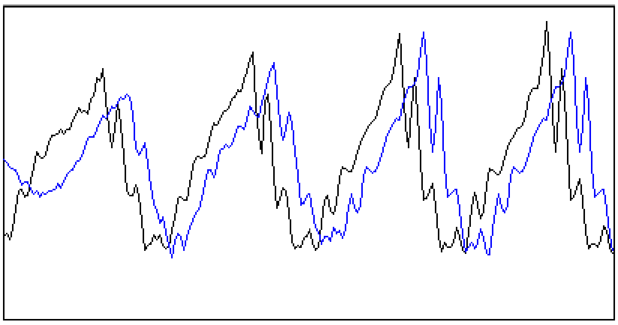
Using the [Speed] combo box, choose a playback speed for voice data.
×1.0 (same as source voice; default) *
×1.5 (speed converted to 1.5 times that of source voice) *
×2.0 (speed converted to 2 times that of source voice) *
×3.0 (speed converted to 3 times that of source voice)
×4.0 (speed converted 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 ∗.)

To compress the silent part 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, and the compressed voice data is reproduced by expansion according to the selected parameters.

The lower waveform display area is used to display part of the voice waveform in the timebase direction 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)

(5) 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 VSX2 file can be generated from source data using "vsx2cmprs.exe".

3.2.4 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 E0C33 assembler.

When writing to ROM in VSX2 format

Determine the quality and compression ratio of the voice data based on the evaluation results of the quality and compression ratio with the "vsx2param.exe" and system memory capacity.

1. Using "vsx2cmprs.exe", compress the source voice data (normalized 16-bit PCM file) in VSX2 format to generate a VSX2 file.

   Example: `DOS>vsx2cmprs -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 1/2 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 "vsx2param.exe".

2. Using "bin2s.exe", convert the VSX2 voice file (binary file) into an assembly source file.

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

   In this example, the VSX2 file "se.vsx" is converted into the assembly source file "se.vsxs". This file "se.vsxs" 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.vsxs"

```
     .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 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. Use this method to write PCM data without VSX2 compression, or write PCM data processed by "vsx2param.exe" to ROM.

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

   Example: `DOS>ppccmprs seN.pcm se.ppc`

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

3.2.5 Precautions Concerning Voice ROM Data Creation

- "vsx2param.exe" uses an algorithm that is 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 "vsx2param.exe" can be used, a sound card (Sound Blaster 16-compatible) that supports 8- to 22-kHz sampling 16-bit monaural voice input/output is required. The program may not work with a sound card that only supports 8-bit data.
- VSX2 voice data and VSX voice data are not compatible with each other.

3.3 Generating a User Program and Linking the TS33 Library

A range of operations from text-speech conversion, VSX2 voice compression and recording to voice expansion, and playback on the E0C33 chip can be implemented by calling up TS33 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 library functions and examples of programs, refer to Section 5, "Library Reference".

The TS ROM data and VSX2 voice ROM data you have generated can be incorporated into the user program or linked to the user program along with the TS33 library after assembling.
When creating and linking programs, note the following:

(1) The top-level functions (ts2top.c) for text-voice conversion calls the top-level functions (vsx2top.c, voxcomn.c and slutil.c) of the VSX2 library. Even if you have not called up the VSX2 functions from the user program directly, link the VSX2 top-level functions and the library.

(2) The TS33 library functions use the CPU’s R8 register. Therefore, when linking TS33 library functions, including the top-level functions, you cannot use the -gp option (optimization using global pointer/R8) of the instruction extender ext33.

(3) Make sure all of the BSS sections used by the TS33 library are mapped into the internal RAM. Also, be sure to use the internal RAM for the stack.

(4) When mapping TS33 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.

(5) Several objects in the 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".

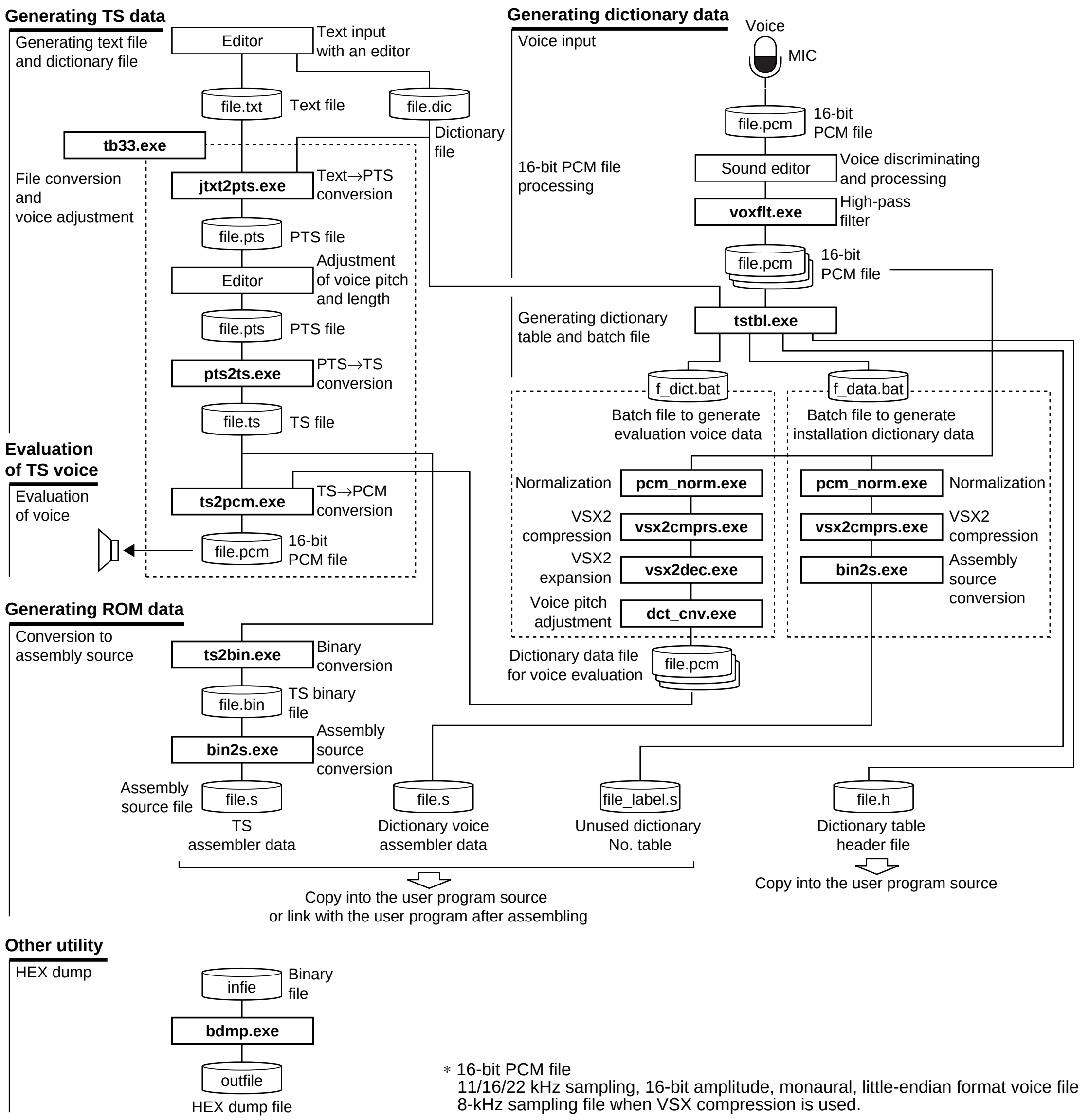
Procedures for executing a sample program using the DMT33005 and DMT33AMP boards are listed in the Appendix for your reference.

4 Tool Reference

This section describes the function of each TS33 tool and how to use them.

4.1 Outline of TS33 Tools

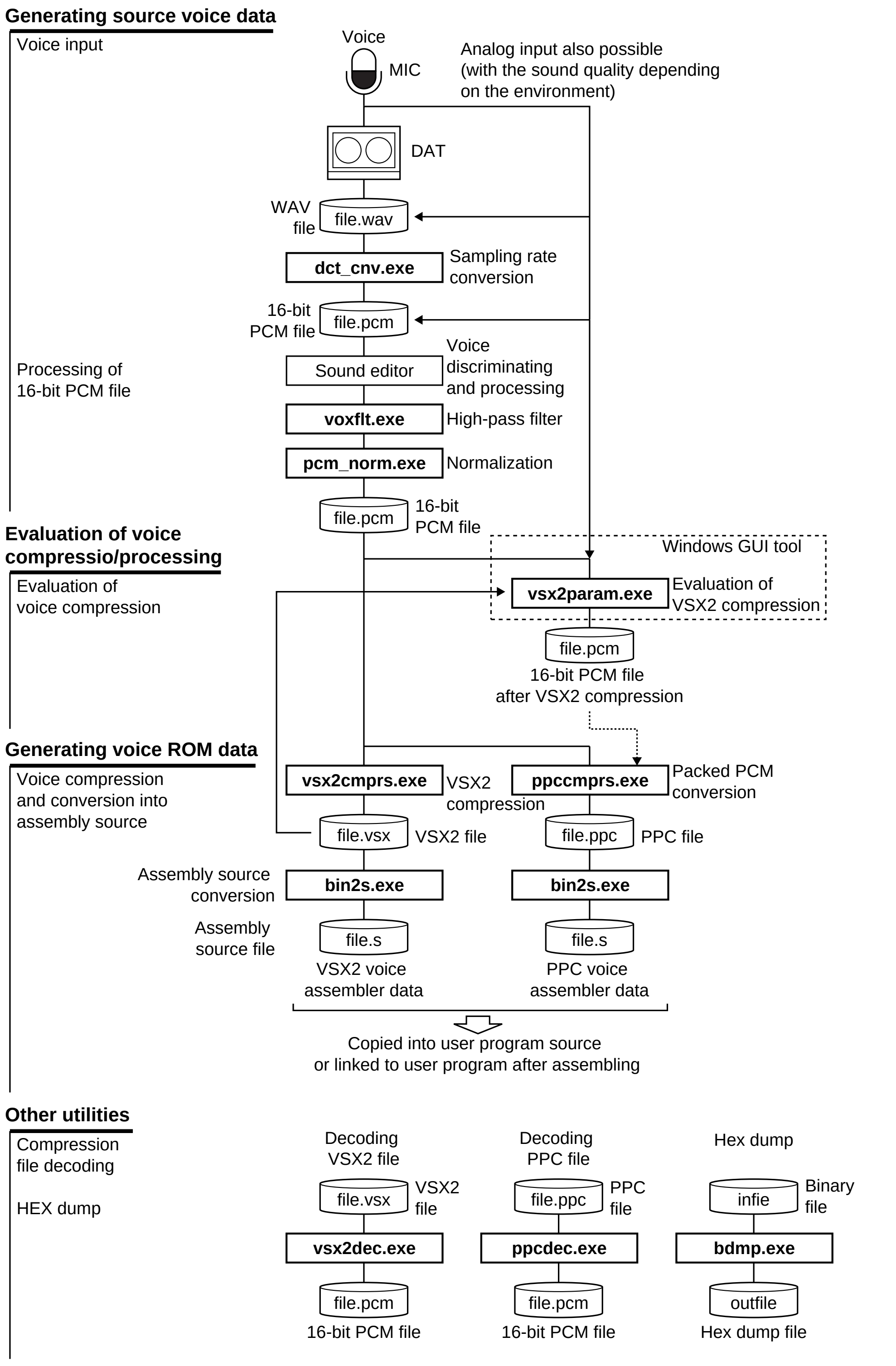
TS33 tools are software tools that are run on a personal computer to generate TS ROM data and VSX2 voice ROM data to be stored on the E0C33 Family chip, as well as to evaluate the voice. All of these tools can be run under Windows 95, Windows NT 4.0, or higher versions. (For details on the operating environment, refer to Section 2.1, "Operating Environment".)
The TS33 tools are located in the "\tstool\bin\" directory.
Figure 4.1.1 and Figure 4.1.2 show the respective procedures for generating TS ROM data and VSX2 voice ROM data.



∗ 16-bit PCM file
11/16/22 kHz sampling, 16-bit amplitude, monaural, little-endian format voice file
8-kHz sampling file when VSX compression is used.

∗ In addition to the VSX2 tools (vsx2cmprs.exe and vsx2dec.exe) for VSX2 compression shown in the Figure, tools for VSX (8K sampling) (vsxcmprs.exe and vsxdec.exe) are also prepared. Note that files compressed by VSX2 tools and files compressed by VSX tools are incompatible.

Figure 4.1.1 Flow Chart for Generating TS ROM Data



File types

16-bit PCM file (.pcm)	11/16/22 kHz, 16-bit amplitude, monaural, little endian format voice file 8-kHz sampling data when VSX compression is used
WAV file (.wav)	48 kHz, 16-bit amplitude, monaural voice file (file not compressed by Windows ADPCM)
VSX2 file (.vsx)	VSX2-compressed voice file
PPC file (.ppc)	Packed PCM file

Figure 4.1.2 Flow Chart for Generating VSX2 Voice ROM Data

TS ROM data generation tools

The TS ROM data generation tools consist of a series of programs that convert text data and compress dictionary data into VSX2 format to generate assembly source files. A Windows GUI tool, tb33.exe, is included for evaluating the quality of converted voice on a PC. Table 4.1.1 below lists these tools.

Table 4.1.1 TS ROM Data Generation Tools

Tool	Function
tb33.exe	A GUI tool for launching the required tools, converting text files, and evaluating voice data following conversion.
jtxt2pts.exe	Converts text files into PTS files to adjust speech parameters.
pts2ts.exe	Converts PTS files into TS files convertible into binary or PCM files.
ts2bin.exe	Converts TS files into binary files.
ts2pcm.exe	Converts TS files into PCM files for evaluating voice on a PC.
tstbl.exe	Load the dictionary file generated by the user and generates a header file for the dictionary table to be included into the user program, and a batch file for converting dictionary data into compressed data for installation.
bin2s.exe	Converts binary data files (TS binary file, VSX2 dictionary data) into assembly source files.
bdmp.exe	A utility for dumping binary data.
dct_cnv.exe	A down-sampler to convert WAV and PCM files into files at selected sampling rates. You may use a commercial sound editor, but degraded sound quality may result.
voxflt.exe	Applies a high-pass filter to 16-bit PCM data to improve sound clarity.
pcm_norm.exe	Normalizes 16-bit PCM data to the specified amplitude and matches it to the input rule of the voice compression tool.
vsx2cmprs.exe	Compresses 16-bit PCM data (16, 11 and 22-kHz sampling) into VSX2 format.
vsxcmprs.exe	Compresses 16-bit PCM data (8-kHz sampling) into VSX format.
ppccmprs.exe	Compresses 16-bit PCM data into packed PCM format.
vsx2dec.exe	Decodes the voice data that has been compressed by "vsx2cmprs.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.
ccap.exe	Files messages issued by the tool used by tb33 during operation. For details on this tool, refer to "E0C33 Family C Compiler Package Manual".

VSX2 voice ROM data generation tools

The VSX2 voice ROM data generation tools consist of a series of programs that compress voice files to generate assembly source files for E0C33 chips. A Windows GUI tool, (vsx2param,exe), is included for evaluating the quality of compressed voice on a PC. Table 4.1.2 below lists these tools. Most of these tools are listed among the TS ROM data generation tools.

Table 4.1.2 VSX2 Voice ROM Data Generation Tools

Tool	Function
dct_cnv.exe	A down-sampler to convert WAV and PCM files into files at selected sampling rates. You may use a commercial sound editor, but degraded sound quality may result.
voxflt.exe	Applies a high-pass filter to 16-bit PCM data to improve sound clarity.
pcm_norm.exe	Normalizes 16-bit PCM data to the 90% amplitude (default) and matches it to the input rule of the voice compression tool.
vsx2cmprs.exe	Compresses 16-bit PCM data (16, 11 and 22-kHz sampling) into VSX2 format.
ppccmprs.exe	Compresses 16-bit PCM data into packed PCM format.
bin2s.exe	Converts binary data files (VSX2 file, PPC file) into assembly source files.
bdmp.exe	A utility for dumping binary data.
vsx2dec.exe	Decodes the voice data that has been compressed by "vsx2cmprs.exe" to save it as PCM data.
vsx2param.exe	Adjust VSX2 parameters and evaluates compressed voice data in VSX2 format.

Note: The evaluation tool uses algorithms that are similar to, but not identical to, the libraries implemented on the E0C33 chip. Use the tool 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 TS ROM Data Generation and Evaluation Tools

This section describes the functions of the tools used to generate and evaluate TS ROM data, and how to use them. For details on VSX2 tools, refer to Section 4.3, "VSX2 Voice ROM Data Generation and Evaluation Tools".
Start the tools, except the evaluation tool "tb33.exe", 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 [ ] indicate options that 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 jtxt2pts.exe

Function: Converts text files for voice conversion into PTS files to adjust speech parameters.

Usage:
```
DOS>jtxt2pts [silent] dictfile.dic infile.txt outfile.pts↵
```

Arguments:

silent	Length of silent period inserted between phonemes/words (optional) 0: Not inserted 1: 16 msec 2: 32 msec (default) 3: 48 msec
dictfile.dic	Dictionary file
infile.txt	Input file name (text file)
outfile.pts	Output file name (PTS file)

Example:
```
DOS>jtxt2pts 2 sample.dic sample1.txt sample1.pts
```

In this example, "sample1.txt" is converted into "sample1.pts". A silent period of 32 msec is inserted between words.

Reference: For details on the contents of each file, refer to the following sections.

Dictionary file:	"3.1.2 Generating a Dictionary File"
Text file:	"3.1.4 Generating Text Data"
PTS file:	"3.1.5 Voice Evaluation and Adjustment"

4.2.2 pts2ts.exe

Function: Converts PTS files into TS files convertible into binary or PCM files. Use this tool after adjusting the parameters of the PTS file generated by "jtx2pts.exe".

Usage: `DOS>`**`pts2ts`** ***infile.pts outfile.ts***↵

Arguments:

infile.pts	Input file name (PTS file)
outfile.ts	Output file name (TS file)

Example: `DOS>pts2ts sample1.pts sample1.ts`

4.2.3 ts2bin.exe

Function: Converts TS files into a binary file that can be converted into an assembly source by "bin2s.exe".

Usage: `DOS>`**`ts2bin`** ***input.ts output.bin***↵

Arguments:

input.ts	Input file name (TS file)
output.bin	Output file name (binary file)

Example: `DOS>ts2bin input.ts output.bin`

4.2.4 ts2pcm.exe

Function: Converts TS files into PCM files, which can be used to evaluate voice data on a PC. A file thus generated is a 8- to 22-kHz sampling monaural PCM file in signed 16-bit little endian format.

Usage: `DOS>`**`ts2pcm`** [***sampling***] ***input.ts output.pcm***↵

Arguments:

sampling	PCM file sampling rate (optional) 8000: 8 kHz sampling 16000: 16 kHz sampling (default) 22050: 22 kHz sampling
input.ts	Input file name (TS file)
output.pcm	Output file name (PCM file)

Example: `DOS>ts2pcm 16000 input.ts output.pcm`

Note:

- PCM conversion with this tool requires evaluation dictionary (PCM files) containing the sound data of adjusted pitches and lengths. You can execute the batch file "*file*_dict.bat" generated by "tstbl.exe" to generate this dictionary. (Refer to Section 3.1.3, "Generating Installation Dictionary Data".)
- Use the 8-kHz sampling rate only when you install dictionary data in VSX format (option of "tstbl.exe"). Do not use 16 or 22 kHz for the VSX format. You cannot use the 8-kHz sampling data for the VSX2 format.
- This tool normally works only with the standard directory configuration. Do not move this tool to another directory or change the directory name.

4.2.5 tstbl.exe

Function: Generates the following files from the user-generated dictionary.

1) Header file (*outfile*.h) that defines the dictionary table to be included in the program
2) Batch file (*outfile*_data.bat) for VSX2 (VSX) compressing the PCM files registered in the dictionary for generating dictionary data (assembly source) to be installed in the product
3) Assembly source file (*outfile*_label.s) describing the labels of unused dictionary numbers
4) Batch file (*outfile*_dict.bat) for generating the voice file used to evaluate text-voice conversion results on a PC

For details on the contents of each file, refer to Section 3.1.3, "Generating Installation Dictionary Data".

Usage: `DOS>`**`tstbl`** `[-c`***XX***`] [-t`***X***`] [-s` ***level***`] [-u] [-v]` ***infile.dic outfile***↵

Arguments:

-c*XX*	Specify a VSX2/VSX compression ratio (optional)	
	-c16	Compresses voice to 16 kbps equivalent.
	-c24	Compresses voice to 24 kbps equivalent. (default)
	-c32	Compresses voice to 32 kbps equivalent.
	-c40	Compresses voice to 40 kbps equivalent.
-t*X*	Specify a VSX2/VSX 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*	VSX2/VSX compression 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 5000. Be sure to insert a space between -s and *level*. If this option is omitted, the default value 0 is assumed.	
-u	Separation of user dictionary (optional) Generates a batch file for generating the PC evaluation dictionary data in two parts, "before [user]" (*outfile*_dict.bat), and "after [user]" (*outfile*_udict.bat). If you omit this option, all data is output in "*outfile*_dic.bat".	
-v	Specify the VSX (8 kH sampling) format (optional) Generates a batch file to compress dictionary data in VSX format. The batch file normally specifies "vsc2cmprs.exe" for compression and "vsx2dec.exe" for expansion. If you specify this option, these tool names will be changed to "vsxcmprs.exe" and "vsxdec.exe," respectively. If you omit this option, a batch file will be generated to compress data in VSX2 format.	
infile.dic	Input file name (dictionary file, text file generated by the user)	
outfile	Output file name (with extension not specified)	

Example:

```
DOS>tstbl -c24 -t2 -s 20 sample.dic vsxdata
```

Output files are generated to generate dictionary data compressed to 24 kbps equivalent, compressed by a factor of 1/2 in the timebase direction, with silent packet level = 20.

4.2.6 bin2s.exe

Function: Converts the binary file (TS binary file, VSX2/VSX file or Packed PCM 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.s	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 sample1c.bin > sample1c.s
DOS>type sample1c.s
       .global sample1c
       .align  2
 sample1c:
       .byte   0x33 0x30 0x09 0x09 0x2d 0x0b 0x15 0x0b
       .byte   0x10 0x0b 0x2b 0x0b 0xff 0x20 0x0d 0x0a
       .byte   0x01 0x0b 0x09 0x2b 0xff 0x10 0x03 0xcb
       .byte   0x44 0x09 0x0e 0x0a 0x2d 0x08 0xff 0x10
       .byte   0x3b 0x0b 0x0c 0x09 0xff 0x20 0xff 0xff
       .byte   0xff 0xff
 ; total 42 bytes data
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 sample1c.bin > sample1c.s
DOS>type sample1c.s
       .global Smp01
       .align 2
Smp01:
       .byte   0x33 0x30 0x09 0x09 0x2d 0x0b 0x15 0x0b
       .byte   0x10 0x0b 0x2b 0x0b 0xff 0x20 0x0d 0x0a
                               :
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, _

4.2.7 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 sample1c.bin
00000000 33 30 09 09 2D 0B 15 0B 10 0B 2B 0B FF 20 0D 0A
00000010 01 0B 09 2B FF 10 03 CB 44 09 0E 0A 2D 08 FF 10
00000020 3B 0B 0C 09 FF 20 FF FF FF FF

DOS>bdmp -l sample1c.bin
00000000 3033 0909 0B2D 0B15 0B10 0B2B 20FF 0A0D
00000010 0B01 2B09 10FF CB03 0944 0A0E 082D 10FF
00000020 0B3B 090C 20FF FFFF FFFF

DOS>bdmp -m sample1c.bin
00000000 3330 0909 2D0B 150B 100B 2B0B FF20 0D0A
00000010 010B 092B FF10 03CB 4409 0E0A 2D08 FF10
00000020 3B0B 0C09 FF20 FFFF FFFF
```

4.2.8 Executing Tools from a Batch File

TS 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.
As an example, this section discusses the "ts.bat" batch file found in the "tstool\sample\" directory.

ts.bat

Generates a TS data assembly source from the PTS file modified by the user.
This batch file is generated to execute the TS ROM-generation tools in "tstool\bin\". The current directory should be "tstool\sample\". Correct when necessary for execution.

Processing:
1) Converts the PTS file into a TS file.
2) Converts the TS file into a binary file.
3) Converts the TS binary file into an assembly source file.

Input file:	*file_name*.pts	PTS file
Output file:	*file_name*.s	Assembly source file

Contents of file:
```
set tspath=..\bin\
%tspath%pts2ts %1.pts %1.ts
%tspath%ts2bin %1.ts %1.bin
%tspath%bin2s -l %1 %1.bin > %1.s
```

Example:
```
>ts sample1c
```
Generates an assembly source file "sample1c.s" from the PTS file "sample1c.pts".
The TS data has the global label (sample1c), which is the same as the input file name.

```
            .global sample1c
            .align  2
sample1c:
            .byte   0x33 0x30 0x09 0x09 0x2d 0x0b 0x15 0x0b
            .byte   0x10 0x0b 0x2b 0x0b 0xff 0x20 0x0d 0x0a
            .byte   0x01 0x0b 0x09 0x2b 0xff 0x10 0x03 0xcb
            .byte   0x44 0x09 0x0e 0x0a 0x2d 0x08 0xff 0x10
            .byte   0x3b 0x0b 0x0c 0x09 0xff 0x20 0xff 0xff
            .byte   0xff 0xff
; total 42 bytes data
```

Reference: "4.2.2 pts2ts.exe", "4.2.3 ts2bin.exe", "4.2.6 bin2s.exe"

4.2.9 tb33.exe

"tb33.exe" is an application used to convert text files and PTS files into PCM files, using the command-line version of the TS33 tool, and then to perform playback of the file for evaluation.

Note: Sound evaluation with this tool requires a Sound Blaster 16-compatible sound card that supports 8 to 22-kHz sampling, 16-bit monaural sound output, and an evaluation dictionary data file (PCM file) containing sound data for processed pitches and lengths. You can generate this evaluation dictionary data by running the "xxxx_dict.bat" batch file (or "xxxx_udict.bat" when the -u option of "tstbl.exe" is specified) generated by "tstbl.exe". (refer to Section 3.1.3, "Generating Installation Dictionary Data").

Starting and quitting

tb33.exe

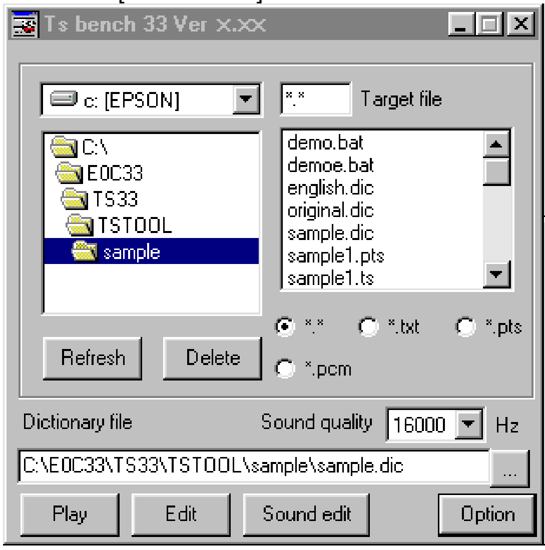
Double-click on the "tb33.exe" icon to start the tool.

To quit "tb33.exe," click on the [Close] button at the upper right corner of the [Ts bench 33] window.

Window

"tb33.exe" consists of the following four windows.

[Ts bench 33] window



c: [EPSON]

C:\
E0C33
TS33
TSTOOL
sample

.

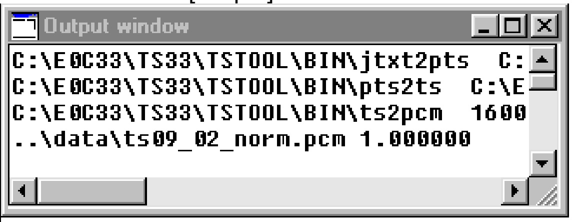
demo.bat
demoe.bat
english.dic
original.dic
sample.dic
sample1.pts
sample1.ts

● *.* ○ *.txt ○ *.pts ○ *.pcm

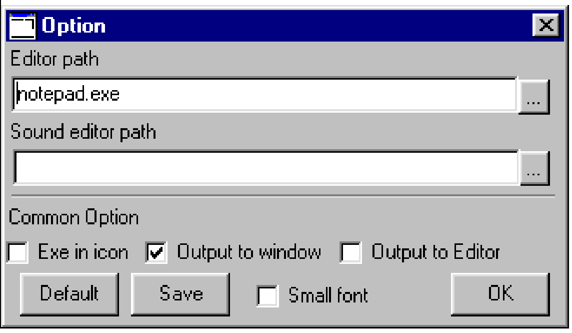
Dictionary file: C:\E0C33\TS33\TSTOOL\sample\sample.dic

Sound quality: 16000 Hz

[Output] window



```
C:\E0C33\TS33\TSTOOL\BIN\jtxt2pts  C:
C:\E0C33\TS33\TSTOOL\BIN\pts2ts  C:\E
C:\E0C33\TS33\TSTOOL\BIN\ts2pcm  1600
..\data\ts09_02_norm.pcm 1.000000
```



Editor path: notepad.exe

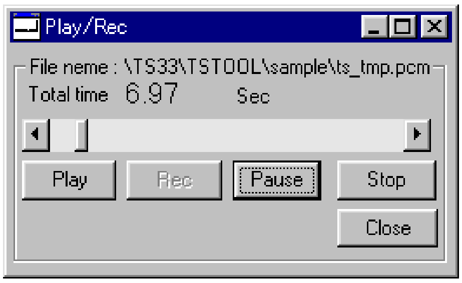
Sound editor path:

Common Option

☐ Exe in icon ☑ Output to window ☐ Output to Editor

☐ Small font

[Option] window



File neme : \TS33\TSTOOL\sample\ts_tmp.pcm

Total time 6.97 Sec

[Play/Rec] window

[Ts bench 33] window

When "tb33.exe" starts up, the [Ts bench 33] window appears. All operations involving text-voice conversion are performed in this window.

[Output] window

Displays the execute commands and results (output messages) of the tool invoked to convert data. Running the tool opens this window automatically.
You must have previously set the destination for the output results to this window in the [Option] window.

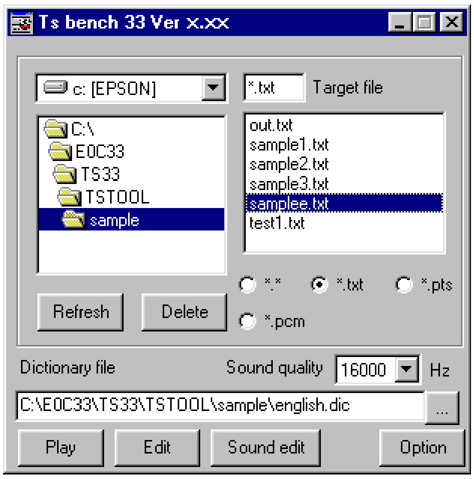
[Option] window

Click on the [Option] button in the [Ts bench 33] window to display this window. Use it to select the editor you want to use, as well as execution options.

[Play/Rec] window

This window is displayed when text conversion by the [Play] button in the [Ts bench 33] window finishes and playback starts. This window is used to control playback and stop of converted voice data.

File selection


Ts bench 33 Ver x.xx

c: [EPSON]

*.txt Target file

C:\
E0C33
TS33
TSTOOL
sample

out.txt
sample1.txt
sample2.txt
sample3.txt
samplee.txt
test1.txt

( ) *.* [✓ *.txt] ( ) *.pts ( ) *.pcm

Dictionary file

Sound quality 16000 Hz

C:\E0C33\TS33\TSTOOL\sample\english.dic

Select files to be converted or reproduced using the [Ts bench 33] window directory list and the file list box.
Use the radio button to select the file format to be displayed in the file list box.

[Refresh] button

The contents of the file list box are not updated automatically if files are added or deleted with any tool other than tb33. Click on the [Refresh] button to update the list.

[Delete] button

Deletes the selected files from the file list box.

[Edit] button

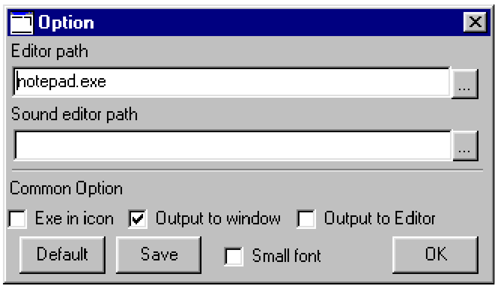
After selecting the text format file from the file list box, click on the [Edit] button to start the editor and open the selected file. Windows Notepad is default editor. You can select another editor from the [Option] window.

[Sound edit] button

After selecting the PCM file in the file list box, click on the [Sound edit] button to start the sound editor and open the selected file. This function is enabled only when a sound editor has been selected from the [Option] window.

Selecting options

Click on the [Option] button to open the [Option] window.



[Editor path] text box

Specify the editor to be launched when you click on the [Edit] button. Enter the start up command including the absolute path.

[Sound editor path] text box

Specify the sound editor to be launched when you click on the [Sound edit] button. Enter the start up command including the absolute path.

[Exe in icon] check box

When this option is selected, the tool started from tb33 is executed in iconized form.

[Output to window] check box

When this option is selected, the tool start command and output messages are displayed in the [Output] window. tb33 describes the start command and output messages for each tool in the file "tb33.err" using "ccap.exe". (Refer to the "E0C33 Family C Compiler Package Manual".)
The [Output] window displays the contents of the "tb33.err" file.

[Output to Editor] check box

When this option is selected, the tool start command and output messages are displayed with the specified editor. The editor opens "tb33.err" in the same way as does [Output to window].

[Small font] check box

Reduces the size of fonts in the [Output] window display.

[Default] button

Returns the option settings to their default values.

Editor program:	notepad.exe
Sound editor program:	None
Common options:	[Output to window] is selected.

[Save] button

Saves settings, including the editor command, in tb33.sav. The saved parameters apply for the next use.

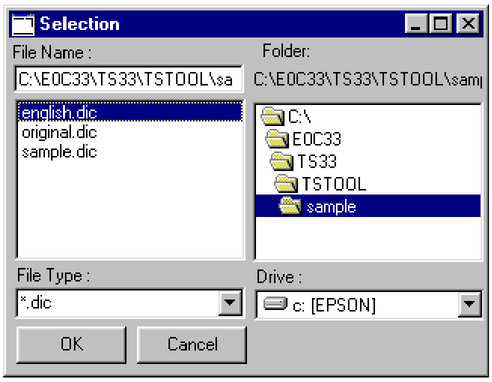
[OK] button

After selecting and setting options, click on this button to close the [Option] window.

Converting text files

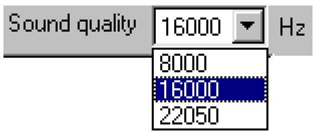
tb33 converts voice conversion text files into PCM files with the following procedure.

1) Display the [Selection] dialog box with the [...] button on the side of the [Dictionary file] text box. Select the dictionary file and click on the [OK] button.



The selected dictionary file name is displayed in the [Dictionary file] text box.

2) Select a sampling rate from the [Sound quality] combo box.



8000	8 kHz sampling (for reproducing only)
16000	16 kHz sampling (VSX2 format)
22050	22 kHz sampling (VSX2 format)

3) Select a text file to be converted from the file list box, then click on the [Play] button.

tb33 executes "jtxt2pts.exe", "pts2ts.exe" and "ts2pcm.exe" in succession, generating and reproducing a 16-bit PCM file at the specified sampling rate.
"jtxt2pts.exe" will be executed with a 32-msec silent period inserted between words. The PTS file generated has the same name as the text file, but with a an extension of ".pts". The TS file generated by "pts2ts.exe" also has the same name, but with a ".ts" extension. The PCM file is "ts_tmp.pcm".

Correction and reconversion of PTS file

Following text file conversion, a PTS file is generated with "jtxt2pts.exe".
Follow the steps given below to correct the PTS file parameters and to perform playback of the corrected voice data file.

1) Double-click on the PTS file in the file list box, or select the file and click on the [Edit] button.
The Notepad or editor selected from the [Option] window launches and opens the file. Correct and save the parameters in the PTS file.

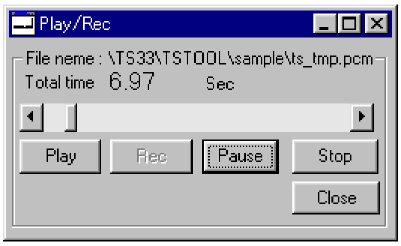
2) Select the corrected PTS file from the file list box and click on the [Play] button.

The specific action performed by the [Play] button depends on the extension of the selected file. If you select a PTS file, the button runs "pts2ts.exe" and "ts2pcm.exe" to generate a 16-bit PCM file at the specified sampling rate, and plays back the generated PCM file. The TS file generated has the same name as the PTS file, but with an extension of ".ts". The PCM file is "ts_tmp.pcm".

Reproducing PCM files

Select a PCM file from the file list box and click on the [Play] button to play the sound file.

[Play/Rec] window and playback control



The [Play/Rec] window is displayed when the PCM file following text/PTS file conversion by the [Play] button in the [Ts bench 33] is played back, or when a selected PCM file is played back.
The window closes automatically after completion of playback.
Press the [Pause] button to pause playback. The [Play/Rec] window remains open. Click on the [Play] button to resume playback from the point at which you pressed [Pause]. Use the scroll bar to move the restart point to any other point in the file.
Click on the [Stop] button during playback to stop playback. The [Play/Rec] window remains open. Click on the [Play] button to resume playback from the beginning.
The [Close] button is enabled during playback or a pause. Click on this button to close the [Play/Rec] window.
The [Rec] button is disabled for tb33.

After evaluation

The TS file (.ts) is generated by the "pts2ts.exe" which is activated with the [Play] button in the [Ts bench 33] window. tb33 does not perform TS file-binary conversion (ts2bin.exe) and binary file-assembly source conversion (bin2s.exe). After tb33 runs, run each tool to generate the assembly source file.

4.3 VSX2 Voice ROM Data Generation Tools

This section describes the functions of the tools used to generate and evaluate VSX2 voice ROM data, and how to use them. Some tools, including "bin2s.exe", are also described as TS ROM data generation tools. These tools are identical to those used for TS ROM generation.
Start each tool, except the evaluation tool "tvsx2param.exe", 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, _, .

4.3.1 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 16 sample1.wav sample1.pcm      (x1)
DOS>dct_cnv 96 32 sample1.wav sample1.pcm      (x2)
DOS>dct_cnv 144 48 sample1.wav sample1.pcm     (x3)
DOS>dct_cnv 140 80 sample1.wav sample1.pcm     (x5)
DOS>dct_cnv 480 160 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.

4.3.2 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. (for 8 kHz sampling PCM file) 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: 8 kHz sampling PCM file

```
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
```

Note: The cut-off frequency depends on the sampling frequency of the entered PCM file, as shown in the following Table.

Table 4.3.1 Specified CutOff Value and Cut-off Frequency

Sampling frequency	Cut-off frequency (Hz)
8 kHz	*CutOff* × 1
11.025 kHz	*CutOff* × 1.4
16 kHz	*CutOff* × 2
22.05 kHz	*CutOff* × 2.8

4.3.3 pcm_norm.exe

Function: Converts the voice data amplitude of the input 16-bit PCM file to a specified amplitude.
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:

input.pcm	Input file name (16-bit PCM file)
output.pcm	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. This option is used for VOX compression and is not used for VSX2/VSX compression.

Example:

```
DOS>pcm_norm -r 65 input.pcm output.pcm        ... Converted to 65%
DOS>pcm_norm input.pcm output.pcm              ... Converted to 90% (default)
```

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.

4.3.4 vsx2cmprs.exe

Function: Compresses the 16-bit PCM data with the compression ratio specified by an option and saves the compressed data to a VSX2 file. The VSX2 file thus output can be loaded using the VSX2 evaluation tool "vsx2param.exe".

Usage: `DOS>`**`vsx2cmprs [-c`*`XX`*`] [-t`*`X`*`] [-s`*`level`*`]`** ***infile.pcm outfile.vsx***↵

Arguments:

infile.pcm	Input file name (11 kHz/16 kHz/22 kHz sampling 16-bit PCM file)
outfile.vsx	Output file name (VSX2 file)
-c*XX*	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.
-t*X*	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>vsx2cmprs -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>vsx2cmprs 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.
- Files are output from this tool in VSX2 compression format, which is incompatible with the VSX format output by "vsxcmprs.exe" (8 kHz sampling).

4.3.5 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.

Usage: `DOS>`**`vsxcmprs [-c`*`XX`*`] [-t`*`X`*`] [-s `*`level`*`]`** ***infile.pcm outfile.vsx***↵

Arguments:

infile.pcm	Input file name (8 kHz sampling 16-bit PCM file)
outfile.vsx	Output file name (VSX file)
`-c`*`XX`*	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.
`-t`*`X`*	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.
- Files are output from this tool in VSX compression format, which is incompatible with the VSX2 format output by "vsx2cmprs.exe" (11/16/22 kHz sampling).

4.3.6 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.

4.3.7 bin2s.exe

Function: Converts the binary file (VSX2 file, VSX file 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.s	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.vsx > 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.vsx > 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, _

4.3.8 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.vsx
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.vsx
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.vsx
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
```

4.3.9 vsx2dec.exe

Function: Decodes the VSX2 file compressed by "vsx2cmprs.exe" and saves the decoded data to a PCM file.

Usage: `DOS>`**`vsx2dec`** `[`***option***`]` ***infile.vsx outfile.pcm***↵

Arguments:

infile.vsx	Input file name (VSX2 file)
outfile.pcm	Output file name (11/16/22 kHz sampling 16-bit 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>vsx2dec -f20 sample1.vsx sample1.pcm   ... Saved with ×2 playback speed
DOS>vsx2dec sample1.vsx sample1.pcm        ... Saved with normal playback speed
```

4.3.10 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 (8 kHz sampling 16-bit 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
```

4.3.11 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 a example of processing executed using the batch file provided in the "tstool\smplvsx2\" directory.
Each batch file was created to execute VSX2 voice ROM generation tools in "tstool\bin\" from "tstool\smplvsx2\" as the current directory. Correct any batch file as necessary before using.

pcm2vsx2s.bat

Converts PCM voice data into an assembly source after compressing it in VSX2 format.

Processing:
1) High-pass filtering (120 Hz cut-off for 16 kHz sampling)
2) Normalization (90% amplitude)
3) VSX2 compression (compressed to 24 kbps equivalent, timebase direction = 1/2, silent packet level = 10)
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 60 %1.pcm %1H.pcm
echo pcm_norm %1
..\bin\pcm_norm -r 90 %1H.pcm %1N.pcm
echo vsx2cmprs %1
..\bin\vsx2cmprs -c24 -t2 -s 10 %1N.pcm %1.vsx
echo bin2s %1
..\bin\bin2s -l %1v24t2 %1.vsx > %1.vsxs24
goto END
:ERR
echo Please input filename
:END
```

Example:

```
>pcm2vsx2s se
```

Create an assembly source file "se.vsxs24" from the 16-bit PCM file "se.pcm". The VSX2 compression options used for "vsx2cmprs.exe" are "-c24" (compression to 24 kbps equivalent), "-t2" (compression by a factor of 1/2 in the timebase direction), and "-s 10" (silent packet level = 10). To create the assembly source file under other conditions, change these options to those desired. The output file "se.vsxs24" has a global label (sev24t2) derived from the input file name by adding "v24t2".

```
    .global sev24t2
    .align  2
sev24t2:
    .byte   0x53 0x22 0x01 0x01 0x01 0x01 0x22 0x90
    .byte   0x89 0x24 0x92 0x41 0x25 0x90 0xc3 0x25
    .byte   0x92 0x49 0x24 0x90 0x41 0xb2 0x53 0x65
                                 :
; total 9271 bytes data
```

Reference: "4.3.2 voxflt.exe", "4.3.3 pcm_norm.exe", "4.3.4 vsx2cmprs.exe", "4.3.7 bin2s.exe"

4.3.12 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. "vsx2data.mak" is provided in the "tstool\smplvsx2\" directory as a samole make file for VSX2 compression.
For details on the make file format and the functions of make, refer to the "E0C33 Family C Compiler Package Manual".

vsx2data.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 for 16 kHz sampling)
2) Normalization (90% amplitude)
3) Creation of PPC file
4) Creation of a VSX2 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 "vsx2data.vs" in which these assembly source files have been combined.

Packed PCM voice ROM data file:	se.pp
VSX-compressed voice ROM data files:	se.x16t2, se.x24t1, se.x24t2, se.x24t3, se.x24t4, se.x32t2, se.x40t2 The value ".x??" denotes the numeric values representing the compression ratio options specified in "vsx2cmprs.exe". The value "t?" indicates that data is compressed by a factor of 1/? in the timebase direction. All silent packet levels are 10.

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

```
DOS>make -f vsx2data.mak
```

The make file "vsx2data.mak" is created to be run from "tstool\smplvsx2\" as the current directory. Therefore, you need to specify the "make.exe" directory by the PATH command or copy "make.exe" into the "tstool\smplvsx2\" directory before opening the make file.

The make file "vsx2data.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 vsx2data.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)\vsx2cmprs.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

.pcm.x16t2 :
         $(VOXFLT) -l 60 $*.pcm $*.pcmh
         $(PCM_NORM) -r 90 $*.pcmh $*.pcmn
         $(VSXCMPRS) -c16 -t2 -s 10 $*.pcmn $*.vsx
         $(BIN2S) -l $*v16t2 $*.vsx > $*.x16t2

.pcm.x24t1 :
         $(VOXFLT) -l 60 $*.pcm $*.pcmh
         $(PCM_NORM) -r 90 $*.pcmh $*.pcmn
         $(VSXCMPRS) -c24 -t1 -s 10 $*.pcmn $*.vsx
         $(BIN2S) -l $*v24t1 $*.vsx > $*.x24t1

.pcm.x24t2 :
         $(VOXFLT) -l 60 $*.pcm $*.pcmh
         $(PCM_NORM) -r 90 $*.pcmh $*.pcmn
         $(VSXCMPRS) -c24 -t2 -s 10 $*.pcmn $*.vsx
         $(BIN2S) -l $*v24t2 $*.vsx > $*.x24t2

.pcm.x24t3 :
         $(VOXFLT) -l 60 $*.pcm $*.pcmh
         $(PCM_NORM) -r 90 $*.pcmh $*.pcmn
         $(VSXCMPRS) -c24 -t3 -s 10 $*.pcmn $*.vsx
         $(BIN2S) -l $*v24t3 $*.vsx > $*.x24t3

.pcm.x24t4 :
         $(VOXFLT) -l 60 $*.pcm $*.pcmh
         $(PCM_NORM) -r 90 $*.pcmh $*.pcmn
         $(VSXCMPRS) -c24 -t4 -s 10 $*.pcmn $*.vsx
         $(BIN2S) -l $*v24t4 $*.vsx > $*.x24t4

.pcm.x32t2 :
         $(VOXFLT) -l 60 $*.pcm $*.pcmh
         $(PCM_NORM) -r 90 $*.pcmh $*.pcmn
         $(VSXCMPRS) -c32 -t2 -s 10 $*.pcmn $*.vsx
         $(BIN2S) -l $*v32t2 $*.vsx > $*.x32t2

.pcm.x40t2 :
         $(VOXFLT) -l 60 $*.pcm $*.pcmh
         $(PCM_NORM) -r 90 $*.pcmh $*.pcmn
         $(VSXCMPRS) -c40 -t2 -s 10 $*.pcmn $*.vsx
         $(BIN2S) -l $*v40t2 $*.vsx > $*.x40t2

.pcm.pp :
         $(VOXFLT) -l 60 $*.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

vsx2data.vs : $(ALL_S)
         type se.pp  > vsx2data.vs
         type se.x16t2 >> vsx2data.vs
         type se.x24t1 >> vsx2data.vs
         type se.x24t2 >> vsx2data.vs
         type se.x24t3 >> vsx2data.vs
         type se.x24t4 >> vsx2data.vs
         type se.x32t2 >> vsx2data.vs
         type se.x40t2 >> vsx2data.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
```

4.3.13 vsx2param.exe

This tool is used to evaluate VSX2-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 VSX2-compressed data can be loaded for expansion and playback.

Starting and quitting

vsx2param.exe

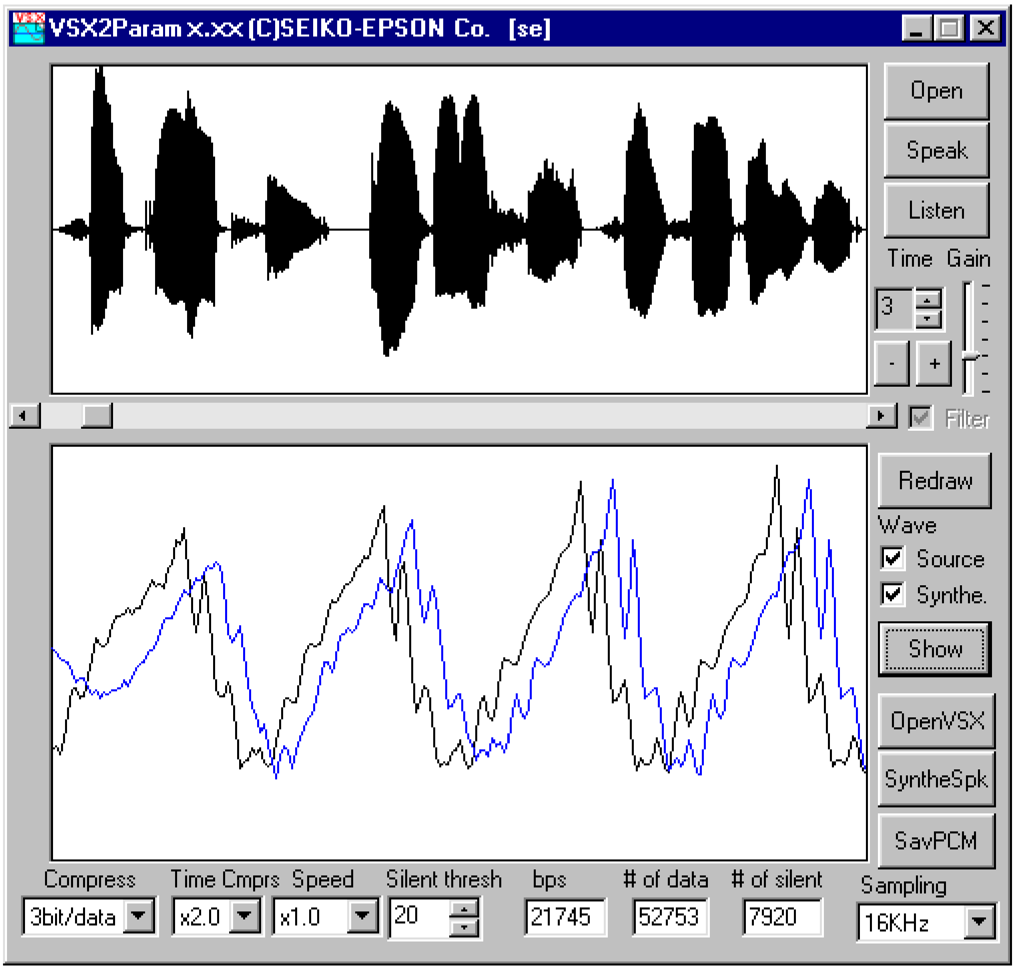
Double-click on the "vsx2param.exe" icon to start the tool.

To quit "vsx2param.exe", click on the [Close] button at the upper right corner of the [VSX2Param] window.

Windows and the function of each part

[VSX2Param] window

When "vsx2param.exe" starts up, the [VSX2Param] window appears. To perform any operation from this window, click on the appropriate control button using the mouse.



Compress: 3bit/data
Time Cmprs: x2.0
Speed: x1.0
Silent thresh: 20
bps: 21745
# of data: 52753
# of silent: 7920
Sampling: 16KHz
Time: 3

Controls for waveform display

Upper window

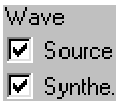
Full-waveform display area
Displays the full waveform of the source voice data loaded from a file or entered from a microphone.

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.

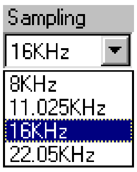
[Redraw] button
Redraws the waveforms in both waveform display areas.

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



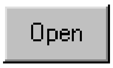
[Wave] check box
Used to choose the waveform to be displayed in the partial-waveform display area.
If you choose [Source], the source voice waveform is displayed in black.
If you choose [Synthe.], the compressed data waveform is displayed in blue.
You also can choose both [Synthe.] and [Source].
Note 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



[Sampling] combo box
Selects the voice sampling rate. This parameter determines sound quality.
When loading a PCM data, select the sampling rate of the file

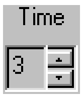
8kHz	(for PCM playback)
11.025kHz	(for VSX2 compression)
16kHz	(for VSX2 compression, default)
22.05kHz	(for VSX2 compression)



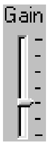
[Open] button
Loads a 16-bit PCM file.
The waveform of the loaded voice data is displayed in the waveform display area.

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

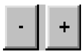
[Listen] button
Enters voice data from a microphone.
The waveform of the entered voice data is displayed in the waveform display area.



[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 VSX2-compressed voice file. Always be sure to choose a VSX2 voice file that has been generated by "vsx2cmprs.exe".
The waveform of the loaded voice data is displayed in the waveform display area.
When a VSX2 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.].

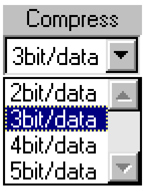
Controls for compression

[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.

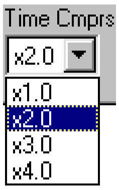
[SavPCM] button

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



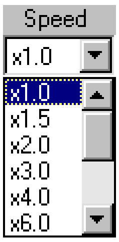
[Compress] combo box

Used to choose an ADPCM-compatible compression ratio.
Here, one of the following compression ratios can be selected:
2bit/data (Compress voice to 16 kbps equivalent)
3bit/data (Compress voice to 24 kbps equivalent; default)
4bit/data (Compress voice to 32 kbps equivalent)
5bit/data (Compress voice to 40 kbps equivalent)



[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)

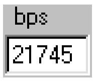


[Speed] combo box

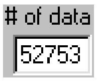
Chooses the speed of speech.
Here, one of the following speeds can be selected:
×1.0 (same as the source voice; default)*
×1.5 (speed converted to 1.5 times that of source voice)*
×2.0 (speed converted to 2 times that of source voice)*
×3.0 (speed converted to 3 times that of source voice)
×4.0 (speed 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 *.)

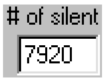
[Silent thresh] edit box
To further compress silent parts of voice data, set the threshold level at which it is judged that there is no sound. The greater the value of this threshold, the higher the compression ratio, but the lower the sound quality. Normally, set this threshold in the range of 0 to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

Basic operation procedure

1. Choose a sampling rate from the [Sampling] combo box.
2. 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.
3. Choose compression parameters and playback speed from the corresponding boxes: [Compress], [Time Cmprs], [Silent thresh], and [Speed].
4. Use the [SyntheSpk] button to reproduce the compressed voice data.
5. The compressed voice data can be saved using the [SavPCM] button.

Precautions

- This evaluation tool uses 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 this 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 this 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 tool may not work with a sound card that only supports 8-bit data.
- The VSX2 format used to compress 11/16/22-kHz sampling 16-bit PCM files is not compatible with the VSX format used to compress 8-kHz sampling 16-bit PCM files.

5 Library Reference

This section describes the precautions to be observed when using TS33 library functions and explains each function in detail.

5.1 Outline of TS33 Library

Functional outline

The TS33 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:

- TS data playback function
- VSX2 data compression/recording and expansion/playback functions

∗ VSX2 conversion incorporates Seiko Epson's original sound-processing technology, which supports voice data sampled at 11.025 kHz, 16 kHz, and 22.050 kHz. The VOX33 voice compression/expansion middleware has a VSX conversion function for voice data sampled at 8 kHz. This format is incompatible with VSX2.

This software package also contains the C source of the top-level 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 an application program is shown in Figure 5.1.1.

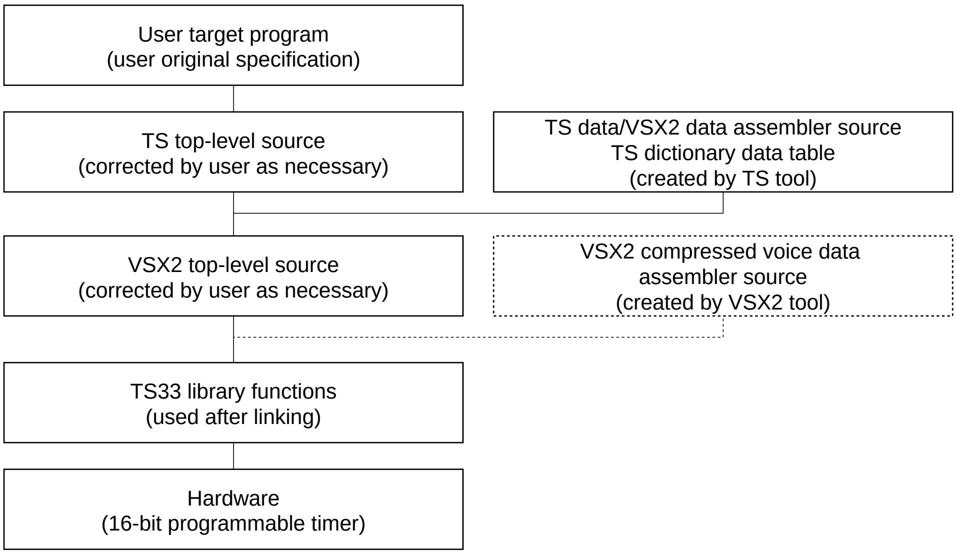


Figure 5.1.1 Program Structure

TS33 library structure

The TS33 library and all related files are provided in the "tslib" folder (directory). The contents of the "tslib" folder are listed below.

```
tslib\      ..... TS33 library-related directory
            readme.txt          TS33 library supplementary explanation, etc. (in English)
            readmeja.txt        TS33 library supplementary explanation, etc. (in Japanese)
            lib208\    ..... TS33 library for E0C33208 directory
                       vox208.lib   TS33 library for E0C33208
                       sl208.lib    Voice input / output library for E0C33208
                       vsx2.o       VSX2 object
                       fadpcm16.o, fadpcm24.o, fadpcm32.o, fadpcm40.o, vsxgcp.o
                                    Objects retrieved from vox208.lib to accelerate operation
            include\   ..... TS33 library function header file directory
                       ts.h         TS header file
                       vsx2.h       VSX2 header file
                       voxcomn.h    Library common 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
                       ts2top.c     TS top-level function
                       vsx2top.c    VSX2 top-level function
                       voxcomn.c    Library common function
                       slutil.c     SPEAK and LISTEN utility functions
            smpl208\   ..... DMT33005 sample program directory
                             Refer to "readme.txt" or "readmeja.txt" in the "tslib\" directory for the
                             composition and the method for using sample programs.
            hardsrc\   ..... Hardware dependent source directory
```

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

5.2 Hardware Requirements

Hardware resources used by the library

The TS33 library 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)

The following resources are required to enter voice data to the chip and perform compression in VSX2 format, but are not required for text-voice conversion output alone.

- 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 TS33 library assumes that the high-speed (OSC3) clock frequency used for the E0C33208 is 20 MHz (typ.), and that PLL is in x2 mode (CPU operating clock frequency is 40 MHz).

Memory

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

- Make sure all of the BSS sections used by the TS33 library are mapped into the internal RAM.
- Be sure to use the internal RAM for the stack.
- When mapping TS33 library program code into an external memory area, make sure this area is accessed in 1 or no wait cycle, 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 TS33 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
ts2top.c	int tsInit( )	Initializes TS processing
(TS processing)	unsigned char *tsSpeak( )	Starts TS data voice output
vsx2top.c	int vsx2Speak( )	Expands, speed-converts, and reproduces VSX2 data
(VSX2 processing)	int vsx2Listen( )	Compresses and records VSX2 data
	void vsx2TopDecode( )	Call-back function for reproducing
	void vsx2TopEncode( )	Call-back function for recording
	void vsx2TopEncodeEnd( )	Call-back function for completion of recording
ppctop.c	unsigned char *ppcSpeak( )	Reproduces PCM data
(PCM processing)	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	void voxCodecpy( )	Copies code section
(Common functions)	void adpcmCodecpy( )	Copies VSX2 code section
slutil.c	void setSpeakVolume( )	Sets output volume
(Input/output data	void slPcm2Spk( )	Converts output data
conversion)	void slLis2Pcm( )	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. You must copy all of "ts2top.c", since it describes functions other than the above that do not need to be directly called from the user program.

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.

Notes:
- The TS33 library functions use the CPU’s R8 register. Therefore, when linking the TS33 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.
- Even when performing TS voice conversion output alone, you must use the "vsx2top.c" and "slutil.c" playback (Speak) functions and the "voxcomn.c" functions. Copy or link these functions.

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. The cut-off frequency of the filter is set to 120 Hz for the 16 kHz sampling rate. 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".

ADPCM

To use the VSX2 function, specify this option when compiling "voxcomn.c".

SAMPLING22K

Specify this option when using raw data in 22-kHz 16-bit little endian format as PCM data.

SAMPLING16K

Specify this option when using raw data in 16-kHz 16-bit little endian format as PCM data.

SAMPLING11K

Specify this option when using raw data in 11-kHz 16-bit little endian format as PCM data.

SAMPLING8K (for TS using optional VSX)

Specify this option when using raw data in 8-kHz 16-bit little endian format as PCM 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.

• HIGH_PASS_FILTER	Defined
• IRAM_CACHE	Defined
• ADPCM	Defined
• SAMPLINGxx	SAMPLING16 defined
• 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.

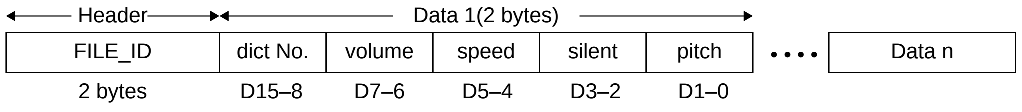
Slparam SlParam;

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

5.3.3 Data Structure

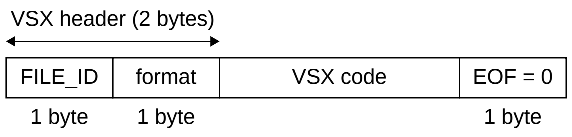
Given below is the structure of TS data and voice data.

TS data



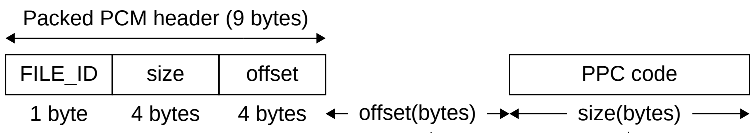
FILE_ID:	File ID (0x3333)
dict No.:	Dictionary No. (0x00 to 0xfe: 0xff indicates silent data, and D7–0 contains the number of silent packets.)
volume:	Volume (0 = x1, 1 = x0.7, 2 = x 0.5, 3 = x1.4)
speed:	Talking speed (0 = x1, 1 = x1/1.5, 2 = x1/2, 3 = x1.5)
silent:	Length of silent period (0 = 0 msec, 1 = 16 msec, 2 = 32 msec, 3 = 48 msec)
pitch:	Pitch (0 = -10%, 1 = -5%, 2 = Standard pitch, 3 = +10%)

VSX2 data



FILE_ID:	File ID (0x53 = 'S')
format:	Compression ratio

Packed PCM data



FILE_ID:	File ID (0x50 ='P')
size:	Size of PPC code in bytes
offset:	Offset from header to PPC code

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

5.3.4 Error Codes Returned by Top-Level Functions

Error codes returned by TS functions

You can determine whether processing by the tslnit() function in "ts2top.c" terminated normally from the following message codes returned by the functions.

Error code	TS_SUCCESS	(1)	Terminated normally
	TS_ERROR	(0)	Parameter is abnormal
	TS_VSX_ERROR	(-1)	Reading VSX failed
	TS_MAX_ERROR	(-2)	Dictionary No. is too large

Error codes returned by voice-processing functions

You can determine whether VSX2 voice processing (including tsSpeak()) terminated normally from the error codes stored in the following error variables defined in "voxcomn.h".

Error variable	vsx2TopError	For VSX2 processing function
	ppcTopError	For packed PCM processing function

Error code	VOX_SUCCESS	(1)	Terminated normally
	VOX_BUF_FULL	(-1)	Specified data storage buffer filled when recording

5.3.5 TS Data Processing Functions (ts2top.c)

tsInit( )

Function: Initializes TS processing.

Format:

```
int tsInit(short ftbl[], int ptbl[], int Cmpress, int Volume);
```

Arguments:

`short`	`ftbl[]`	16-bit timer setting table
`int`	`ptbl[]`	VSX2 data table
`int`	`Cmpress`	VSX2 compression ratio
`int`	`Volume`	Volume

Return values: Terminated normally........TS_SUCCESS (1)
Terminated abnormally...TS_ERROR (0)

Description: Initializes the VSX2 compression ratio, the setting for the 16-bit timer for voice output, and other settings for TS processing.

Specify the tsFtbl[ ] in the header file generated by "tstbl.exe" for *ftbl[ ]*, and the tsPtbl[ ] in the header file for *ptbl[ ]*.

Specify the argument *Compress* with the constant VSX_CMP defined in the header file generated by "tstbl.exe". For this argument, you can also use the following values defined in "vsx2.h". In this case, set the compression ratio specified when VSX2 data was generated by "tstbl.exe".

```
/* compression rate */
#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
```

The argument *Volume* 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 a tolerable level, it is rounded off. Since the value can be specified in increments of 1, the setting can be finally adjusted.

`volume = 0x100`	1.0-fold sound volume
`volume = 0x80`	0.5-fold sound volume
`volume = 0x200`	2.0-fold sound volume

Example:

```
int iErr;
iErr = tsInit((short *)tsFtbl, (int *)tsPtbl, VSX_CMP, 0x100);
```

Specifies the table in the header file generated by "tstbl.exe" and sets the volume as 1.0 fold.

tsSpeak( )

Function:	Starts TS data voice output.
Format:	`int tsSpeak(unsigned char *tspts, int Speed);`

Arguments:

`unsigned char`	`*tspts`	Pointer to the beginning of the TS data table
`int`	`Speed`	Talking speed

Return values: Terminated normally...........Pointer to output data (SpkParams)
Terminated abnormally......0

Description: Specifies the TS data and the talking speed and starts voice output.

For *tspts*, specify the global symbol in the assembly source file generated from the original text by "jtxt2pts.exe", "pts2ts.exe", "ts2bin.exe", and "bin2s.exe".

The talking speed depends on the value specified for *Speed.* You can use the values defined in "vsx.h". Under most circumstances, select from the following categories.

`#define TS_SPEED_10`	`(0)`	Performs playback at standard speed.
`#define TS_SPEED_15`	`(1)`	Performs playback at 1.5 times the standard speed.
`#define TS_SPEED_20`	`(2)`	Performs playback at twice the standard speed.
`#define TS_SPEED_30`	`(3)`	Performs playback at three times the standard speed.
`#define TS_SPEED_40`	`(4)`	Performs playback at four times the standard speed.

Example:

```
unsigned char *SpkParams;
SpkParams = tsSpeak(sample1c, TS_SPEED_10);
```

Plays back TS data with sample1c global symbol at standard speed.

5.3.6 VSX2 Data Processing Functions (vsx2top.c)

Note: Because VSX2 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. When using the VSX2 compression/recording function, vsxgcp.o must be mapped into the internal memory (it is not necessary when using the VSX2 expansion function only). For details, refer to Section 5.5, "Techniques for Speeding Up Operation".

vsx2Speak( )

Function: Expands, speed-converts, and reproduces VSX2 data.

Format:
```
unsigned char *vsx2Speak(unsigned char *Src, int Speed);
```

Arguments:

`unsigned char`	`*Src`	Playback VSX2 data pointer
`int`	`Speed`	Talking speed

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

Description: After loading VSX2-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 vsx2Listen( )

```
unsigned char       *SpkParams, *LisParams, CmpData[16000*3];
LisParams = vsx2Listen(16000*3,16000*3,CmpData,
VSX_TIME_CMP_10|VSX_COMPRESS_24K,50);
            :
SpkParams = vsx2Speak(CmpData, VSX_SPEED_FAST20);
```

The VSX2 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 TS33 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 = vsx2Speak(sev24t1, VSX_SPEED_NORMAL);
```

vsx2Listen( )

Function: Compresses and records voice data in VSX2 format.

Format:
```
unsigned char *vsx2Listen(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 abnormally..........0

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[16000*3];
LisParams = vsx2Listen(16000*3,16000*3,CmpData,
VSX_TIME_CMP_10|VSX_COMPRESS_24K,50);
```

Voice data is input for about 3 seconds while sampling at 16 kHz. The input voice data is compressed to 24 kbps equivalent and then stored in the buffer CmpData.

vsx2TopDecode( )

Function:	Call-back function for reproducing
Format:	`void vsx2TopDecode(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 "vsx2top.c")

vsx2TopEncode( )

Function:	Call-back function for recording
Format:	`void vsx2TopEncode(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 "vsx2top.c")

vsx2TopEncodeEnd( )

Function:	Call-back function for completion of recording
Format:	`void vsx2TopEncodeEnd(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 "vsx2top.c")

5.3.7 PCM Data Processing Functions (ppctop.c)

ppcSpeak( )

Function: 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 (Fixed at 0)
`int`	`real_time`	Real-time parameter (Fixed at 0)

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

Description: After loading PCM-format data (10-bit amplitude, 8K sampling) from *Data*, this function reproduces it.

Unlike ppsSpeak() in the VOX library, this library does not support VSC conversion (pitch/talking speed conversion). Always set the second (*pitch_speed*) and third (*real_time*) arguments to 0.

Example: Reproducing the data input by pcmListen( )
```
unsigned char *SpkParams, *LisParams, pcmData[16000*3];
unsigned char pcmData[16000*3+PACKPCM_HEADER_SIZE];
LisParams = ppcListen(16000*3, 16000*3, pcmData+PACKPCM_HEADER_SIZE,
pcmData);
            :
SpkParams = ppcSpeak(pcmData, VSC_NORMAL);
```

The PCM data in ppcData is reproduced without changing the tone pitch and talking speed.

Reproducing the PCM data created by the user
Create the voice assembly source data using the TS33 (VSX33) tool "bin2s.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.

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 abnormally..........0

Description: This function inputs 10-bit amplitude PCM data.

Example:
```
unsigned char *LisParams, pcmData[16000*3+PACKPCM_HEADER_SIZE];
LisParams = ppcListen(16000*3, 16000*3, pcmData+PACKPCM_HEADER_SIZE,
pcmData);
```

Voice data is input for about 3 seconds.

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".)

5.3.8 Common Functions (voxcomn.c)

Note: When handling data in VSX2 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 VSX2 code section.

Format: `void adpcmCodecpy(unsigned char format);`

Argument: `unsigned char format` Compression ratio

Return value: None

Description: This function transfers VSX2 code from external ROM to internal RAM (&_START_ADPCODE). For the argument (*format*), use one of the values defined in "vsx2.h":

`#define VSX_COMPRESS_16K (1)`	Copy object to be compressed at 16 kbps
`#define VSX_COMPRESS_24K (2)`	Copy object to be compressed at 24 kbps
`#define VSX_COMPRESS_32K (4)`	Copy object to be compressed at 32 kbps
`#define VSX_COMPRESS_40K (8)`	Copy object to be compressed at 40 kbps

Example:

```
adpcmCodecpy(VSX_COMPRESS_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}
```

5.3.9 Input/Output Data Convert Functions (slutil.c)

The following shows the format of each type of data handled by input/output data functions.

VSX2 data:	Signed 12-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:
vsx2Speak( ), 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:

`short`	`*Src`	Pointer to source data array
`short`	`*Dst`	Pointer to write array
`int`	`Length`	Number of data to be converted (short)
`Slparam`	`*slParam`	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 VSX2 (signed 12-bit) data and output the result to a device used for 8-bit signed data, define slParam as follows:

`slParam->offset = 0x40;`	Adds offset of 0x40
`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

slLis2Pcm( )

Function: Converts input data

Format: `void slLis2Pcm(short *Src, short *Dst, int Length, Slparam *slParam);`

Arguments:

`short`	`*Src`	Pointer to source data array
`short`	`*Dst`	Pointer to write arra
`int`	`Length`	Number of data to be converted (short)
`Slparam`	`*slParam`	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 VSX2 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 TS33 Library Functions

The TS33 (VSX2) libraries "vsx2.lib" and "vox208.lib" contain the functions required for expansion, compression in VSX2 format. "sl208.lib" contains the functions required for voice input/output. By linking them 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 mapped into the internal memory. For details, refer to Section 5.5, "Techniques for Speeding Up Operation".
Table 5.4.1 below lists the library functions.

Table 5.4.1 Library Functions

vsx2.o

Type	Function name	Description
VSX2 processing	vsx2ReadHeader( )	Read VSX2 header (parameter)
	vsx2DecodeInit( )	Initialize for VSX2 expansion
	vsx2GetDecodePacketSize( )	Get VSX2 expansion packet size
	vsx2Decode( )	Expand 1 VSX2 packet
	vsx2IsEOF( )	Check for VSX2 end data
	vsx2WriteHeader( )	Write VSX2 header (parameter)
	vsx2EncodeInit( )	Initialize for VSX2 compression
	vsx2SetEncodeData( )	Enter VSX2 compression data
	vsx2GetEncodePacket( )	Get VSX2-compressed packet
	vsx2EncodeFlush( )	Finish VSX2 compression
	vsx2WriteEOF( )	Write VSX2 end data

vox208.lib

Type	Function name	Description
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

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 TS33 (VSX2) library functions use the CPU's R8 register. Therefore, when linking the 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 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 VSX2 Processing Functions

vsx2ReadHeader( )

Function:	Reads VSX2 header (parameter)	
Format:	`int vsx2ReadHeader(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 VSX2 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.	

vsx2DecodeInit( )

Function:	Initializes VSX2 expansion	
Format:	`int vsx2DecodeInit(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 VSX2 expansion processing with vsxParams and sets the playback speed using the second argument (*Speed*).

Reference: Defined values of talking speed in vsx2.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)
```

vsx2GetDecodePacketSize( )

Function:	Gets VSX2-expansion data packet size	
Format:	`int vsx2GetDecodePacketSize(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.	

vsx2Decode( )

Function:	Expands one packet of VSX2 data
Format:	`int vsx2Decode(unsigned char *Src, int *Cont, short *Dst, int DstSize);`

Arguments:

`unsigned char`	`*Src`	Byte array for data input
`int`	`*Cont`	Number of repetitions
`short`	`*Dst`	short-type data array at output destination
`int`	`DstSize`	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 vsx2Decode( ) returns 0.
- For repeated calls, set 0 in *Cont* when first calling up the function for high-order data.

vsx2IsEOF( )

Function: Checks for VSX2 end data

Format: `int vsx2IsEOF(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 VSX2.

vsx2WriteHeader( )

Function: Writes VSX2 header (parameter)

Format:
```
int vsx2WriteHeader(vsxParams *Params, int MaxBytes,
unsigned char *Dst);
```

Arguments:

`vsxParams`	`*Params`	Pointer to vsxParams
`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*

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.

vsx2EncodeInit( )

Function: Initializes VSX2 compression

Format: `int vsx2EncodeInit(vsxParams *Params);`

Argument:

`vsxParams`	`*Params`	Pointer to vsxParams

Return values: Terminated normally...............Other than 0
Terminated abnormally..........0

Description: This function initializes settings for VSX2 compression processing using vsxParams.

vsx2SetEncodeData( )

Function: Enters data for VSX2 compression

Format: `int vsx2SetEncodeData(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 VSX2-compressed.

Note: The second argument (*Length*) and the return value are the number of voice samples, not the number of bytes.

vsx2GetEncodePacket( )

Function: Gets VSX2-compressed packet

Format: `int vsx2GetEncodePacket(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 VSX2-compressed data to *Dst.*
Before using this function, enter the data to be compressed using vsx2SetEncodeData( ).
Then call vsx2GetEncodePacket( ) repeatedly until the return value is 0. After this function returns 0, the next data to be compressed can be entered using vsx2SetEncodeData( ).

vsx2EncodeFlush( )

Function: Carry out processes to complete VSX2 compression

Format: `int vsx2EncodeFlush();`

Argument: None

Return value: Terminated normally...............1

Description: After entering all data to be compressed using vsx2SetEncodeData( ), call vsx2EncodeFlush( ) to close the internal buffers. Then call vsx2GetEncodePacket( ) repeatedly until the return value is 0. When vsx2GetEncodePacket( ) returns 0, compression is terminated.

vsx2WriteEOF( )

Function: Writes VSX2 end data

Format: `int vsx2WriteEOF(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 VSX2 data to *Dst.*

5.4.2 PCM Processing Functions

packpcmReadHeader( )

Function:	Reads the packed PCM header (parameter)
Format:	`int packpcmReadHeader(unsigned char *Src, packpcmParams *Params);`
Arguments:	`unsigned char *Src` Byte array for data input `packpcmParams *Params` 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:	`packpcmParams *Params` Pointer to packpcmParams `int PacketSize` 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:	`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 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.

5.4.3 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 × *SPK_SOFTENING* × CENTER_DATA [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 abnormally..........0

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

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 abnormally..........0
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 abnormally..........0
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 abnormally..........0

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( ).

5.4.4 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

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 abnormally..........0

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 abnormally..........0

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 abnormally..........0

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.

5.4.5 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.
- Seiko Epson recommends always specifying a cut-off frequency of about 120 Hz or 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
(You can specify these values directly only with 8-kHz sampling.)

The value for specifying the cut-off frequency depends on the sampling frequency, as shown in the following Table.

Table 5.4.2 Cut-off Specification Value and Frequency

Sampling frequency	Cut-off frequency (Hz)
8 kHz	*CutOff* × 1
11.025 kHz	*CutOff* × 1.4
16 kHz	*CutOff* × 2
22.05 kHz	*CutOff* × 2.8

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 "vox208.lib" after mapping them into the internal memory, it is possible to increase the library processing speed to some extent. 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 fadpcm16.o

   * The lib208 directory contains the sample objects listed below. Therefore, these objects can be used directly.
   fadpcm16.o, fadpcm24.o, fadpcm32.o, fadpcm40.o, vsxgcp.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 ADPCODE
   -section VSXGCP
   -ucode ADPCODE {..\..\lib208\fadpcm16.o ..\..\lib208\fadpcm24.o
   ..\..\lib208\fadpcm32.o ..\..\lib208\fadpcm40.o }
   -ucode VSXGCP {..\..\lib208\vsxgcp.o}
   ```

   If you do not use VSX2 compression and recording functions (using only VSX2 expansion), you do not need to map vsxgcp.o into internal memory.

   Looking at the linked map file, you will find that the execution addresses of the specified modules have been mapped into the internal memory.
   Example: Map file
   ```
   Address     Vaddress    Size        File
   006098f8    000010fc    00000620    ..\..\lib208\fadpcm16.o      0   REL
   00609f18    000010fc    00000654    ..\..\lib208\fadpcm24.o      0   REL
   0060a56c    000010fc    000006b8    ..\..\lib208\fadpcm32.o      0   REL
   0060ac24    000010fc    0000078c    ..\..\lib208\fadpcm40.o      0   REL
   0060b3b0    00001888    00000148    ..\..\lib208\vsxgcp.o        0   REL
   ```

3. Since the routine to transfer the object code to the internal memory is prepared in the top-level functions, you do not need to describe a transfer routine in the user program. However, since it is a compile option, define "IRAM_CACHE" when compiling "vsx2top.c" and "ADPCM" when compiling "voxcomn.c" with the -D option.
   Example: Transfer routine in vsx2Speak() and vsx2Listen().
   ```
   #ifdef IRAM_CACHE
     adpcmCodecpy(params.format);
     voxCodecpy(&__START_VSXGCP, &__START_vsxgcp_code, &__SIZEOF_vsxgcp_code);
   #endif
   ```

5.6 Library Performance and Memory Size

5.6.1 CPU Occupancy of Library

The CPU occupancy rates shown below have been calculated assuming the following conditions as the standard environment:

Operating frequency:	Internal 40 MHz, external 20 MHz
CODE section:	External ROM (1 wait cycle)
BSS section and stack:	Internal RAM (no wait cycle)
Internal RAM cache:	Used to copy fadpcm16.o, fadpcm24.o, fadpcm32.o, fadpcm40.o and vsxgco.o to internal RAM.
Input/output sampling rate:	16 kHz
VSX2 parameters:	Compression ratio = 24 kbps & ×2 equivalent compression (VSX_TIME_CMP_20 | VSX_COMPRESS_24K), Silent threshold = 20

Table 5.6.1 CPU Occupancy Rates During Voice Expansion/Output (Values Expressed in %)

Function	CPU occupation rates
VSX2 expansion	21%
SPEAK (voice output routine)	10%
VSX2 compression	50%
LISTEN (voice input routine)	10%

CPU occupancy rate during expansion with varying parameters

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.

CPU occupancy rate during compression with varying parameters

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.

Determination of whether real-time execution is possible

Check whether the splis queue is empty when the SpkOnDone() or LisOnDone() call-back function is called in the actual system. If this queue is empty, processing is executed in real time.

5.6.2 Memory Sizes Used

Table 5.6.2 lists the memory sizes used by the VSX2 playback and recording functions. The values in this table were measured using the sample programs in the "smpl208\vsx2\" directory. The internal RAM is used for the BSS section, stack, and program cache.

Table 5.6.2 Memory Sizes Used

Memory	Used size
CODE	17K bytes
BSS	3.6K bytes
Stack	0.3K bytes
Program cache	2.5K bytes
Internal RAM	6.4K bytes

5.7 Program Examples

The following explains how to create TS and VSX2 processing routines, using the sample programs located in the "tslib\smpl208\" directory as examples.

Setting interrupt vectors

Set the address of the SpkIntr0() function as the interrupt vector for 16-bit timer 5, and the address of the LisIntr0() function as the interrupt vector for the A/D converter. If the actual system requires only TS data voice output, without voice input functions, you do not need to set the A/D converter interrupt vector.
The sample program sets a trap table with "common\table.s". If you use this file, set the start-address of the processing routine that corresponds to the trap vector address required for the application.
Example: common\table.s

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Interruput Vectores
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    .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 compareB
                          :
    .word   LisIntr0        ;  64 ADC
                          :
exception:

    jp 0
```

Boot routine

Initializes processing at startup. The sample boot routine is prepared as "common\boot.s", which sets up the stack, enables interrupts, and sets bus conditions. Allocate the stack to internal RAM.
Example: common\boot.s

```
#define STACK_INIT      0x00002000
#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
```

Bus condition settings and other information are written in "common\demoasm.s". See this file for content details.

TS output routine

The following example uses the "demo.c" program found in "smpl208\demo2\", which outputs the following sentence, "This is the E0C33 family text to speech demonstration," at normal and 1.5 times the normal talking speeds.

To check voice output, download this program to DMT33005 + DMT33MON + DMT33AMP. For details on checking voice output, refer to Appendix.

Example: demo2\demo.c

```
//*****************************************************************
//  demo.c : text to speech demonstration main function
//******************************************************************

#include "voxcomn.h"                                                            (*1)
#include "vsx2.h"
#include "ts.h"
#include "vsxdata.h"   // vsx data table by using ts tools.

// ts data

extern unsigned char sampleE[];                                                 (*2)

// this is speak batch function. Please copy this function to your code.

tsSpeakBatch(unsigned char *Data, int speed)                                    (*3)
    {
     unsigned char* SpkParams;

     SpkParams = tsSpeak(Data, speed);
     if(SpkParams==0)
              return;
     do
     { }
     while(SpkIsRunning(SpkParams));
    }

// roop
void exit(void){
     for (;;)
         {
         }
     }

// this is main function for TS33 demonstration
void main(void)
    {
     unsigned char* SpkParams;
     int iErr;

     // call initialize function

     iErr = tsInit((short *)tsFtbl, (int *)tsPtbl, VSX_COMPRESS_24K, 0x100); (*4)

     // use only a letter
     if (iErr == TS_SUCCESS)
         {
              tsSpeakBatch(sampleE, TS_SPEED_10);          // x1 speed       (*5)
              Wait(10000);                                 // wait 1sec
              tsSpeakBatch(sampleE, TS_SPEED_15);          // x1.5 speed
         }
     exit();
    }
```

*1 Include "ts.h" and the other header files required in the user program.
The "voxcomn.h" and "vsx2.h" files are required, even if no VSX2 compression or expansion is performed directly in the user program. Also include the dictionary header file ("vsxdata.h" in this example) generated by "tstbl.exe".

*2 Use the global symbol name generated by "bin2s.exe" to define the TS data generated by the TS tool for external reference. Describe all data to be used.

*3 The tsSpeakBatch() function is defined. This function calls the TS top-level function tsSpeak() using its arguments (TS data and talking speed), monitors output status with the SpklsRunning() function during voice output, and returns after completion of output.
This function can be copied to the user program and used without modification for general use.

*4 In TS processing, the tsInit() function initializes settings. A compression ratio equivalent to 24 kbps and the standard talking speed (0x100) are specified.

*5 The tsSpeakBatch() function in paragraph *3 performs TS data voice output. It outputs the above sentence at normal and 1.5 times the normal talking speeds.
Wait() is a wait routine described in "smpl208\common\demoasm.s". For the argument, specify a wait time in 0.1-msec increments.

VSX2 voice input/output routine

The "vsx2demo.c" sample program, located in the "sampl208\vsx2\" directory, is described below. The program performs the following functions.

1) The sample voice data (16 kHz sampling PCM data), "tstool\smplvsx2\se.pcm", is reproduced once.
2) The sample voice data "se.pcm" is expansion-reproduced using VSX2-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. For details on ways to do this, refer to Appendix.

Example: vsx2\vsx2demo.c

```
/*
 *  vsx2demo.c : VSX2 demonstration main function
 *
 *  1. Play Original data (PCM data).
 *  2. Play Low compress VSX2 data to high compress VSX2 data.
 *  3. Play VSX2 data with speed change.
 *  4. Push REC switch (SW4) then record voice data about 3 second.
 *     (SW1 and SW2 changes compression ratio)
 *  5. After recording, Push PLAY switch (SW3) then play recorded
 *     data with change speed.
 *
 */

#include "voxcomn.h"                                              (*1)
#include "vsx2.h"

extern unsigned char sep[];                                       (*2)
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 vsx2_ratio[] = {                                       (*3)
    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  (16000*3)                                     (*4)
static unsigned char  CmpData[DATA_SIZE];

vsx2SpeakBatch(unsigned char *Data, int speed)                    (*5)
{
    unsigned char* SpkParams;
    SpkParams = vsx2Speak(Data, speed);
    if(SpkParams==0) return;
    do { } while(SpkIsRunning(SpkParams));
}


Listen(int mode)                                                            (*6)
{
    unsigned char* LisParams;

    LedON();
    Wait(200);
    LisParams = vsx2Listen(DATA_SIZE,DATA_SIZE,CmpData,vsx2_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);                                                  (*7)

    SpkParams = ppcSpeak(sep, 0, 0);                                        (*8)
    do { } while(SpkIsRunning(SpkParams));

    vsx2SpeakBatch(sev40t2, VSX_SPEED_NORMAL);                              (*9)
    vsx2SpeakBatch(sev32t2, VSX_SPEED_NORMAL);
    vsx2SpeakBatch(sev24t2, VSX_SPEED_NORMAL);
    vsx2SpeakBatch(sev16t2, VSX_SPEED_NORMAL);
    Wait(10000);
    vsx2SpeakBatch(sev24t1, VSX_SPEED_NORMAL);
    vsx2SpeakBatch(sev24t2, VSX_SPEED_NORMAL);
    vsx2SpeakBatch(sev24t3, VSX_SPEED_NORMAL);
    vsx2SpeakBatch(sev24t4, VSX_SPEED_NORMAL);
    Wait(10000);
    vsx2SpeakBatch(sev24t2, VSX_SPEED_FAST20);

    /* record and play */
    for(;;) {                                                               (*10)
        /* mode is 4bit data [SW2 SW1 SW4 SW3] */
        mode = GetEvent();
        switch(mode & 3) {
        case 1:
            vsx2SpeakBatch(CmpData, VSX_SPEED_NORMAL);
            vsx2SpeakBatch(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);                                                   (*6')
            break;
        default:
            break;
        }
    }
}
```

*1 Include "voxcomn.h" and "vsx2.h" in the user program. (In this example, "ts.h" is not included, since the TS function is not used.)

*2 Use the global symbol name generated by "bin2s.exe" to define the compressed voice data generated from "se.pcm" for external reference. Describe all data to be used.
The voice data used here is generated by the make file "tstool\smplvsx2\vsx2data.mak" (refer to Section 4.3.12).

*3 A table is defined for the compression parameters used as the arguments of vsx2Listen() for voice recording and compression, in order to select compression parameters from the external switch setting (variable). This table is not necessarily used for voice compression.

*4 A buffer is defined for voice compression. Its size is given by the used sampling rate multiplied by recording time (seconds). In this example, a buffer is prepared for a sampling rate of 16 kHz and recording time of three seconds.

*5 The vsx2SpeakBatch() function is defined. This function calls the VSX2 top-level function vsx2Speak() using its arguments (voice data and talking speed), monitors output status with the SpklsRunning() function during voice output, and returns after completion of output.
This function can be copied to the user program and used without modification for general use.

*6 A recording routine called from *6' of main(). This routine inputs voice with the VSX2 top-level function vsx2Listen() and compresses it in VSX2 format.

*7 Sets the play-back sound level (as normal (original) in this case).

*8 Plays back the original voice PCM data.

*9 Expands voice data compressed in VSX2 format with the vsx2SpeakBatch() defined at *5 and outputs the aforementioned nine categories of voice data.
Wait() is a wait routine described in "smpl208\common\demoasm.s". For the argument, specify a wait time in 0.1 msec increments.

*10 Records and plays back voice according to the settings of the DMT33005 switches.

5.8 Precautions

(1) The TS33 library functions use the CPU's R8 register. Therefore, when linking the TS33 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) Even when performing TS voice conversion output alone, you must use the "vsx2top.c" and "slutil.c" playback (Speak) functions and the "voxcomn.c" functions. Copy or link these functions.

(3) The requirements for real-time voice processing are as follows:
- Make sure all of the BSS sections used by the TS33 library are mapped into the internal RAM.
- Be sure to use the internal RAM for the stack.
- When mapping TS33 library program code into an external memory area, make sure this area is accessed in 1 or no wait cycle, if possible. Also, be sure to use a memory area 16 bits wide for this external area.

(4) When handling data in VSX2 format, always be sure to map vsxgcp.o and fadpcm16.o, fadpcm24.o, fadpcm32.o, or fadpcm40.o into the internal memory according to the compression ratio used.

(5) 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).

(6) When VSX2 data is reproduced, the number of data samples output differs by several packets from the source voice data.

(7) 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 processing by executing a sample program using E0C33 Family demonstration tools, the DMT33005, DMT33MON, and DMT33AMP.

A.1 System Configuration Using DMT33005

A.1.1 Hardware Configuration

Configure the system shown in Figure A.1.1 using the DMT33005, 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.

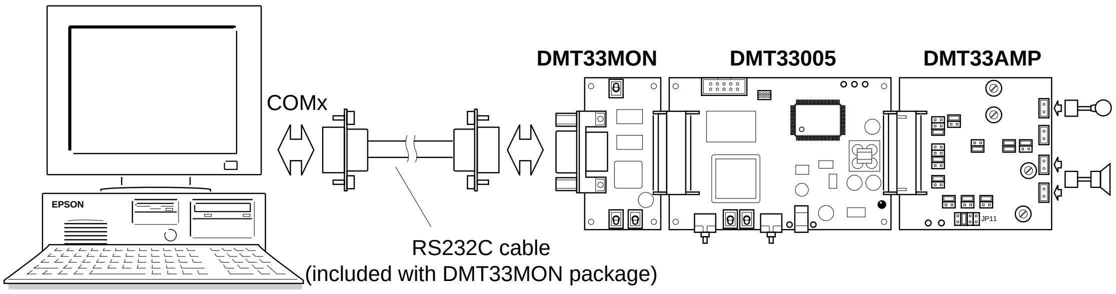


Figure A.1.1 System Configured with DMT33005, DMT33MON, and DMT33AMP

DMT33005 board

The DMT33005 is a demonstration tool for the E0C33208, 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.

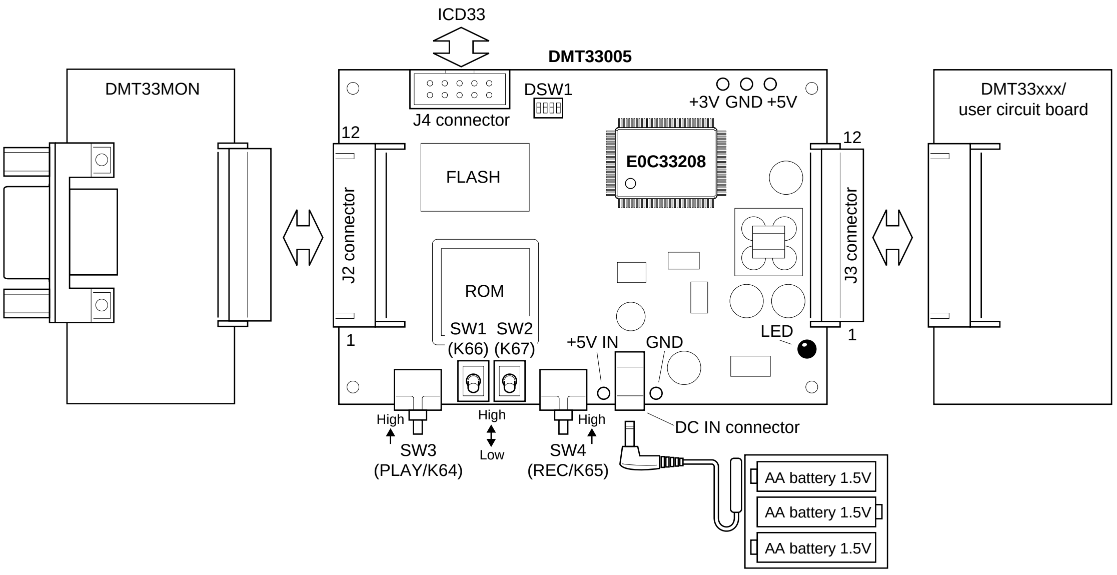


Figure A.1.2 DMT33005 Board

DMT33MON board

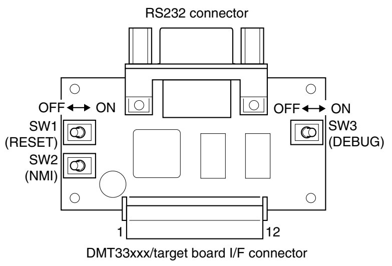


Figure A.1.3 DMT33MON Board

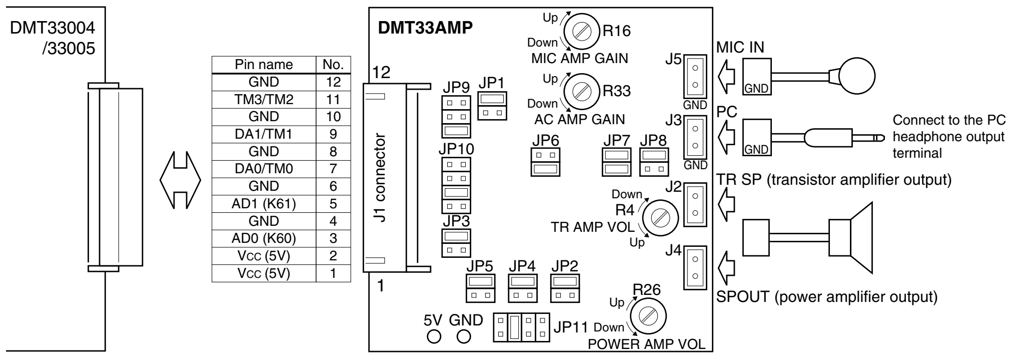
The DMT33MON interfaces with demonstration tools such as the DMT33005 and the user target board for a debugging monitor. By connecting the DMT33005 board to your personal computer via the DMT33MON board, you can debug programs on-board using the debugger (db33.exe) installed in your computer.

Note: For the DMT33005 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 DMT33005, 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.

Note: DMT33AMP is a board for 8-kHz sampling. For 16 or 22-kHz sampling voice input/output, use the DMT33AMP2 board.



Pin name	No.
GND	12
TM3/TM2	11
GND	10
DA1/TM1	9
GND	8
DA0/TM0	7
GND	6
AD1 (K61)	5
GND	4
AD0 (K60)	3
Vcc (5V)	2
Vcc (5V)	1

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 1sr 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 DA1/TM1: 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 DMT33005 board, choose TM with the jumper switches JP2 and JP3, and DA1/TM1 with JP9. Other jumper switch settings 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 DMT/EPOD/MEM Board Manual".

1. Attach the DMT33MON and DMT33AMP to the DMT33005.
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 DMT33005) and connect it to the DMT33005.
6. Turn on the power to the personal computer.

A.12 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 DMT33005, a debugger (db33) of Ver. 1.72 or later is required.

A.2 Sample Program Execution Procedure

Preparatory operations before startup

Since sample program directories "tslib\smpl208\demo2\" and "tslib\smpl208\vsx2\" contain absolute object files in executable format, there is no need to compile or link the sample program specifically. Also provided is a batch file from which you can launch the debugger. The following explains how to start each sample program after downloading them to the DMT33005.

(1) Connect the boards and PC and turn on the power to each equipment according to "System connections" in Section A.1.1.

(2) Before a program can be downloaded to the DMT33005, the debug monitor must already be active. Reconfirm that the DMT33MON DEBUG switch (SW3) is set to the ON position, then reset the system using the RESET switch (SW1).

(3) When executing the TS sample program, make "tslib\smpl208\demo2\" the current directory, and run "208demo.bat" from the DOS prompt.

```
C:\E0C33\TS33\TSLIB\SMPL208\DEMO2\>208demo
```

When executing the VSX2 sample program, make "tslib\smpl208\vsx2\" the current directory, and run "vsx2demo.bat" from the DOS prompt.

```
C:\E0C33\TS33\TSLIB\SMPL208\VSX2\>vsx2demo
```

The batch file for each sample assumes that the debugger (db33) is installed in the "c:\cc33\" directory when it starts the debugger in debug monitor mode (-mon).
Example: \demo1\208demo.bat

```
@echo off
start db33 -p 33208_v.par -b 115200 -c 208demo.cmd -mon
```

The debugger can also be launched from the work bench (wb33). Choose the debug monitor mode from wb33 and the command file (208demo.cmd, vsx2demo.cmd) to be executed at startup time.

(4) When the debugger starts, the sample program is loaded into the DMT33005's RAM area (beginning with address 0x600000) by the commands in the command file.

(5) Use db33's g command ([Go] button) to run the sample program. To stop, use the debugger's forced break function ([Key Break] button).

The functions and usage of the demo2 and vsx2demo files are described below.

TS data processing sample program (demo2)

Run "208demo.bat" in the "tslib\smpl208\demo2\" directory and then execute the debugger g command. The sentence, "This is the E0C33 family text to speech demonstration," is output at the normal and 1.5 times the normal talking speeds.
The program enters a permanent loop following output of this voice data.
Use the [Key Break] button to halt the program. To restart, execute the rsth command before the g command.

VSX2 voice input/output processing sample program (vsx2demo)

Run "vsx2demo.bat" in the "tslib\smpl208\vsx2\" directory. Execute the degugger g command to start the following sequence.

1) The sample voice data (16 kHz sampling PCM data), "tstool\smplvsx2\se.pcm", is reproduced once.

2) The sample voice data "se.pcm" is expansion-reproduced using VSX2-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 DMT33005 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 DMT33005
   These switches set VSX2 compression parameters.
   Each time you press the DMT33MON NMI switch, the compression ratio changes between 24 kbps and 32 kbps.
   DMT33005 SW1 and SW2 determine the compression ratio in the timebase direction.

Table A.2.1 Settings of SW1 and SW2 (DMT33005)

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 DMT33005
   When you press the REC switch, the LED on the DMT33005 lights for about 3 seconds. The voice input from the microphone on the DMT33AMP is recorded in VSX2 format during this time.

   PLAY(SW3) on DMT33005
   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.

Use the [Key Break] button to halt the program. To restart, execute the rsth command before the g command.

A.3 Building Sample Programs

Each sample directory contains a *make* file "xxxx.mak". When you've corrected the source, use "xxxx.mak" to create an object file in executable format "xxxx.srf". The *make* for each sample program requires the source files in the "common\" directory, as well as files in each sample directory. For the required files, refer to each linker command file.

Procedure for executing *make*

1. Change "tslib\smpl208\demoX\" or "tslib\smpl208\vsx2\" to the current directory.
2. Enter the following command at the DOS prompt.
   For TS (example of demol):
   ```
   C:\E0C33\TS33\TSLIB\SMPL208\DEMO1>make -f 208demo.mak
   ```
   For VSX2:
   ```
   C:\E0C33\TS33\TSLIB\SMPL208\VSX2>make -f vsx2demo.mak
   ```

You can also run *make* from the work bench wb33. (See the "E0C33 Family C Compiler Package Manual".)

Note: The TS33 library functions use the CPU's R8 register. Therefore, when linking the TS33 library to the program, do not use the -gp option (optimization using global pointer/R8) of the instruction extender ext33.

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 Munich, GERMANY
Phone: +49-(0)89-14005-0 Fax: +49-(0)89-14005-110

- GERMANY -
SALES OFFICE
Altstadtstrasse 176
51379 Leverkusen, GERMANY
Phone: +49-(0)2171-5045-0 Fax: +49-(0)2171-5045-10

- UNITED KINGDOM -
UK BRANCH OFFICE
Unit 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: 64107319

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 -
EPSON TAIWAN TECHNOLOGY & TRADING LTD.
10F, No. 287, Nanking East Road, Sec. 3
Taipei, TAIWAN
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
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, 150-763, 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

ENERGY
SAVING
EPSON

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

■ EPSON Electronic Devices Website
http://www.epson.co.jp/device/

Issue MARCH 2000, Printed in Japan Ⓜ A