MF1347-01

EPSON

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

SOUND33 MIDDLEWARE MANUAL

ENERGY SAVING EPSON

SEIKO EPSON CORPORATION

PREFACE

Written for developers of application systems using the E0C33 Family of microcomputers, this manual describes the configuration, functions, and operation of the SOUND33 as sound output middleware for the E0C33 Family.

CONTENTS

1 Outline of the SOUND33 Middleware

SOUND33 is the music reproduction middleware for the E0C33 Family of microcomputers. It produces musical reproduction output data from ROM using PWM (Pulse Width Modulation) implemented by 16-bit timers. The output routine is supplied in the form of library functions, linked with the target program for use in applications. This product also includes tools used to create sound ROM data and to evaluate sound on a personal computer. The main features of this product are listed below:

- Seiko Epson's exclusive WAVE sound source
  Able to reproduce the sounds of various musical instruments, using minimal data size.
- Supports 20 types of musical instruments as a standard feature, including eight types of percussion instruments. (Also supports custom user-made sound sources.)
- Supports musical reproduction from 8 kHz monaural to 32 kHz stereo.
- Able to output a maximum of 47 discrete sounds simultaneously (8 kHz data when operating at 40 MHz).
- Supports 7-octave musical scales.
  A total of 84 musical scales (including semitones) from standard C1 (33 Hz) to B7 (3,951 Hz)
- Accommodates a wide range of musical notes lengths, from whole notes to thirty-second notes.
  Allows detailed settings for tempo, sound volume, and gate time.
- Sound ROM data may be created on a PC from standard MIDI files.
- Supports the E0C332xx, which is capable of 40 MHz operation.

CAUTION

- Be sure to fully evaluate the operation of your application system before shipping. Seiko Epson assumes no responsibility for problems arising from use of this middleware in your commercial products.
- Rights to sell this middleware are owned solely by Seiko Epson. Resale rights are not transferred 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 SOUND33 Package

The following lists the contents of the SOUND33 package. After unpacking, check to see that all items are included with your package.

(1) Tool disk (CD-ROM)	1 disk
(2) E0C33 Family SOUND33 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 the Sound Output System

The SOUND33 library is a middleware positioned between the E0C33 hardware and the user program, providing hardware control for sound output.

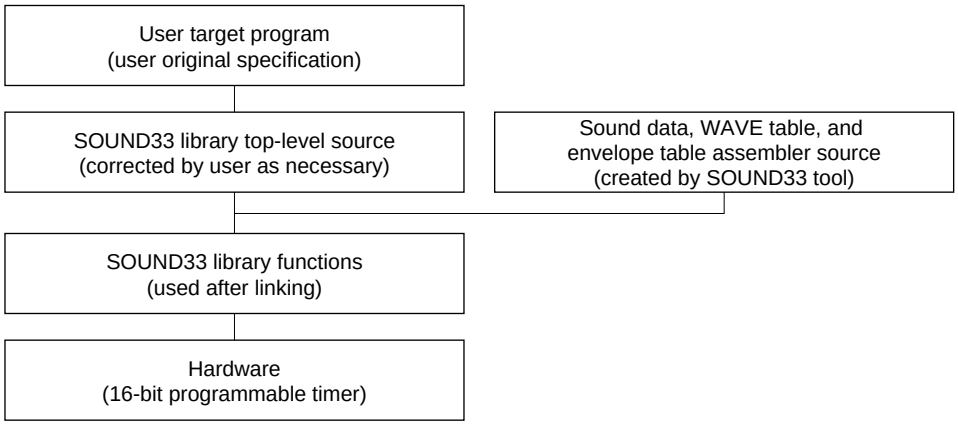


Figure 1.2.1 Software Configuration of the Sound Output System

For more information on the SOUND33 library, see Section 5, "SOUND33 Library Reference".

The SOUND33 library uses two to five channels of 16-bit programmable timers on the E0C33 chip to output sound signals. This output drives a speaker or piezoelectric buzzer, as shown below.

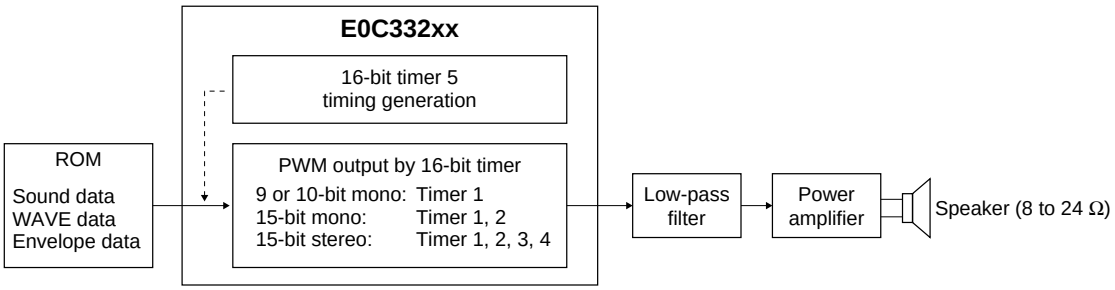


Figure 1.2.2 Hardware Configuration of the Sound Output System

For more information on designing external analog circuits such as the low-pass filter and power amplifier, or on 15-bit output using two channels of timers, refer to the "E0C33 Family Application Notes".

1.3 SOUND33 Tools

SOUND33 tools are PC software for creating and evaluating the sound ROM data to be written to the E0C33 Family chip. All of these tools run under Windows 95/98, Windows NT 4.0, or later versions.

For more information on creating sound data, see Section 3, "Software Development Procedure". For more information on SOUND33 tools, see Section 4, "SOUND33 Tool Reference".

2 Installation

This section describes the operating environment for SOUND33 tools and explains how to install the SOUND33 middleware.

2.1 Operating Environment

Sound ROM data creation and evaluation by SOUND33 requires the following operating environment:

Personal computer

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

Display

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

System software

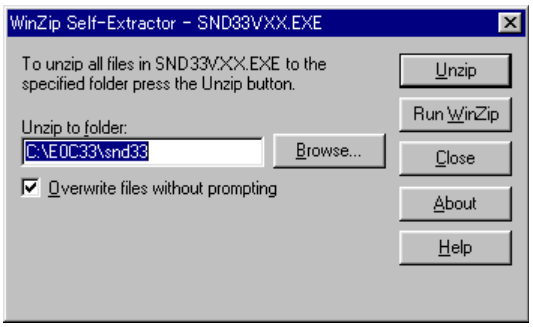
The SOUND33 tools run under Microsoft® Windows®95/98, Windows NT®4.0, or later versions (in Japanese or English).

Other requirements

E0C33 Family C Compiler Package is required for software development.

2.2 Method of Installation

The SOUND33 library and SOUND33 tools are supplied on CD-ROM. Open the self-extracting file on the CD-ROM named "SND33VXX.EXE" to install the SOUND33 library and SOUND33 tools in your computer. (The XX in this file name denotes a version number. For Version 1.0, for example, the file is named "SND33V10.EXE".) Double-click on "SND33VXX.EXE" to start installation. The dialog box shown below appears.



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

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

(root)\

readme.txt	Supplementary explanation, etc. (in English)
readmeja.txt	Supplementary explanation, etc. (in Japanese)

sndtool ..... SOUND33 tool directory

readme.txt	SOUND33 tool supplementary explanation, etc. (in English)
readmeja.txt	SOUND33 tool supplementary explanation, etc. (in Japanese)

bin ..... SOUND33 tools

sb33.exe	Work bench for sound evaluation
txt2snd.exe	Sound text → SND file conversion tool
snd2bin.exe	SND file → binary file conversion tool
snd2pcm.exe	SND file → PCM file conversion tool
bin2s.exe	Binary→ assembly source conversion tool
bdmp.exe	Binary file dump program
pcm2stb.exe	WAVE table musical scale data creation tool
stb12.exe	WAVE table creation tool
stbadd12.exe	WAVE table waveform addition tool
etb.exe	Envelope table creation tool
dct_cnv.exe	Sampling conversion tool
pcm_norm.exe	PCM file normalize tool
ccap.exe	Tool message filing tool
midi2snd.exe	MIDI file → SND file/sound binary file conversion tool

sample ..... Sample directory
Sample sound data, batch files, etc.

smplstb ..... Sample WAVE data directory
Sample sound source, batch files, etc.

mid ..... Sample MIDI data directory
Sample MIDI files, etc.

stb ..... WAVE table data
WAVE table data used in snd2pcm.exe and demonstration
Note) Do not change the location or name of this directory.

etb ..... Envelope table data
Envelope table data used in snd2pcm.exe
Note) Do not change the location or name of this directory.

midi ..... MIDI file musical reproduction program directory

utility ..... Utility directory
The source code is open to users. Although the source code can be used without restriction, they fall outside the scope of the product warranty.

dmp\	Dump tool
VBCtrl\	Patch program required to make the workbench compatible with older versions of Windows 95 preceding OSR2
all_inst\	Batch file, etc. for creating assembler files for all types of musical instrument

sndlib SOUND33 library-related

readme.txt	SOUND33 library supplementary explanation, etc. (in English)
readmeja.txt	SOUND33 library supplementary explanation, etc. (in Japanese)

lib ..... SOUND33 library directory

snd.lib	Sound engine library
sndcpy.o, sndcpy2.o, snd2.o	Objects for sound engines that require high-speed execution. Copy to internal RAM for use.
spk208.lib, spkintr1.o, spkintr2.o, spkintr3.o, spkintr4.o	Speak library shared with VOX33

include ..... SOUND33 library function include file directory

snd.h	Library include file
sndcomm.h	slutil.c include file
speak.h	sl208.lib include file

src ..... Source directory

sndtop.c	Top-level library functions
slutil.c	PWM output final data creation routine
sndbuf.c	SPEAK/LISTEN buffer setup file
slutil2.c	PWM output final data creation tool (stereo)

hardsrc ..... Hardware-dependent source directory

Spk208.s	Spk208.o source (for E0C33208)
Spk208PW.s	Spk208PW.o source (for E0C33208)
slcomm.def	
slintr.def	
SpkIntr1.s	15-bit monaural interrupt functions
SpkIntr2.s	15-bit stereo interrupt functions
SpkIntr3.s	10-bit monaural interrupt functions
SpkIntr4.s	9-bit monaural interrupt functions

demoX ..... Sample program directory
(For details on the configuration of sample programs, refer to "readme.txt "or "readmeja.txt" in "sndlib".)

3 Software Development Procedure

This section describes the procedure for developing software to output sound on the E0C33 chip. The basic development flow is shown below.

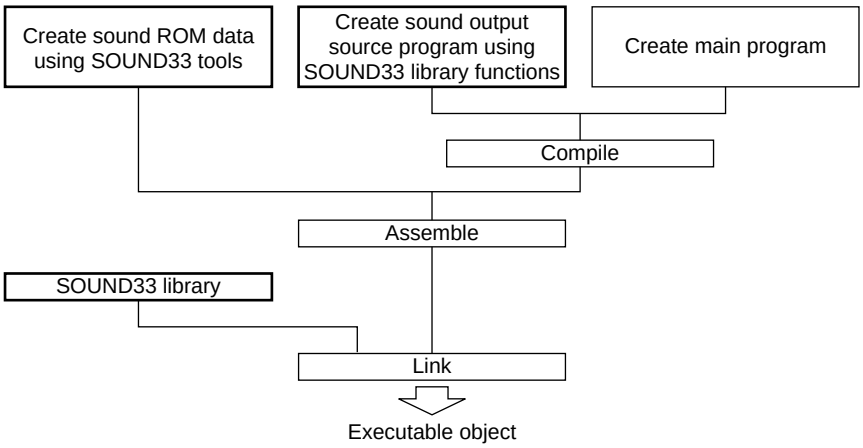


Figure 3.1 Procedure for Developing E0C33 Sound Output Software

1) Create a text file containing a description of musical score data, then create an assembly source file for sound ROM data using SOUND33 tools. In addition to text entry, you can create sound data from standard MIDI files by converting them with the SOUND33 tools. Additionally, create assembly source files for the WAVE data and envelope data for the sound sources used.
2) Create the user program. Sound output is produced by calling SOUND33 library functions. The source file for sound ROM data created in 1) above may be included in the user program source.
3) Compile and assemble the source programs.
4) Link the objects generated in 3) above along with the SOUND33 library. This results in an executable object.

3.1 Creating Sound ROM Data

Figure 3.1.1 gives a procedure for creating sound ROM data and the configuration of SOUND33 tools.

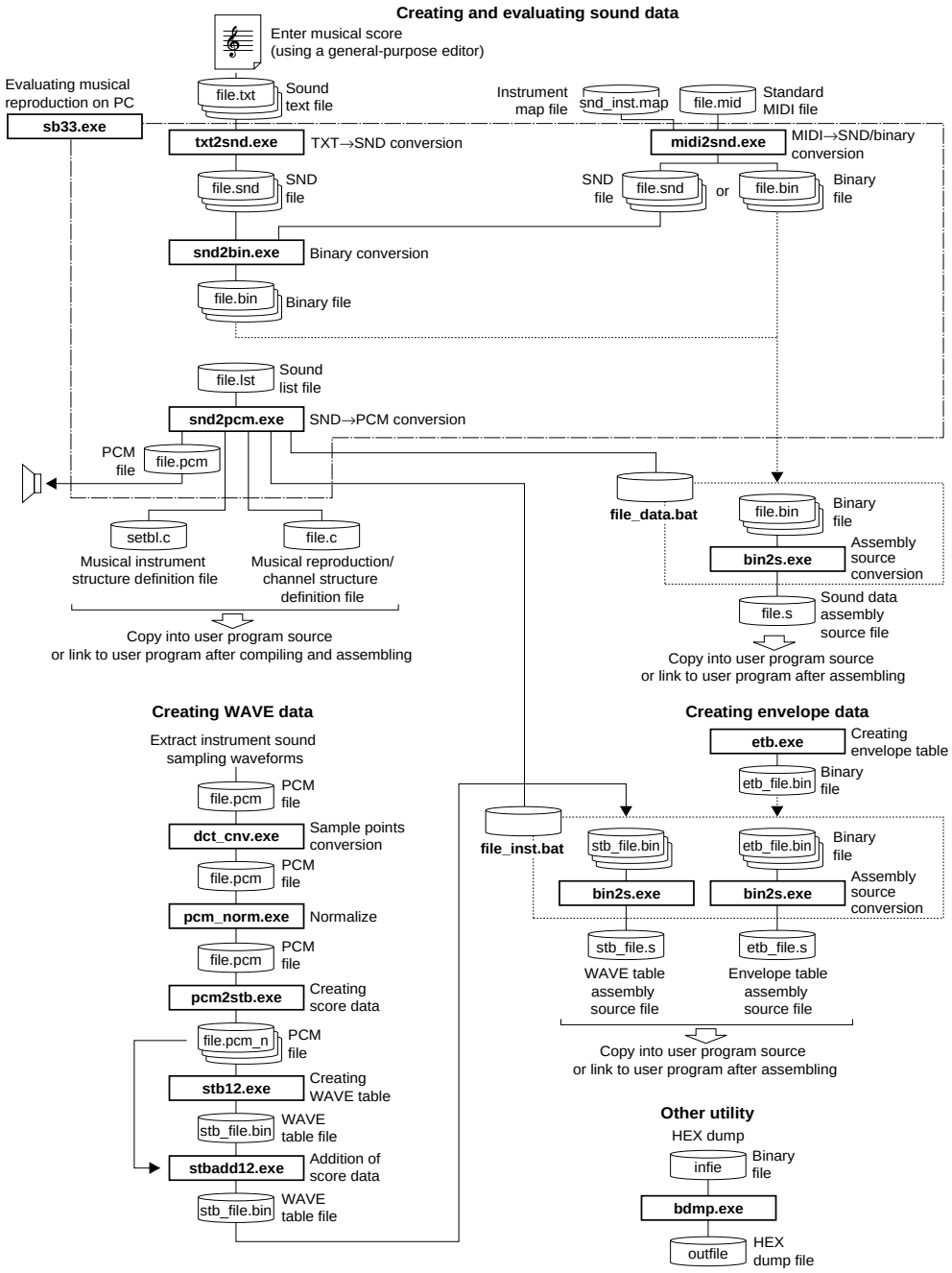


Figure 3.1.1 Flowchart for Creating Sound ROM Data

This section provides only an overview of how to use SOUND33 tools. For more information, refer to Section 4, "SOUND33 Tool Reference".

In the following explanation, we use sample files in the "sndtool\sample\" directory. The explanation assumes that "sndtool\sample\" is the current directory and that PATH is set in the "sndtool\bin\" directory.

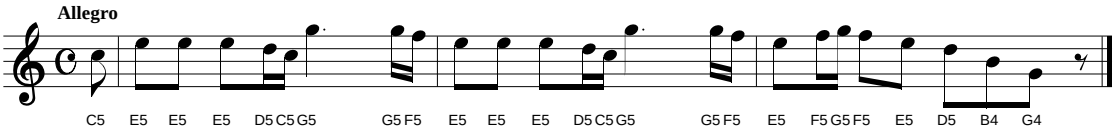
Example:
```
DOS>CD e0c33\snd33\sndtool\sample
DOS>PATH c:\e0c33\snd33\sndtool\bin
```

Note: PCM files handled by SOUND33 tools are 8–32 kHz, 16-bit row data in little-endian format.

3.1.1 Creating Sound Text Files

Start by entering data for musical scores using an editor, then save as a text file (.txt). Create a separate text file as an independent channel for each part of a simple tone using one sound source. Note that the number of channels on which sound can be simultaneously produced on the actual IC is limited by the resources available. (Refer to Section 5.6, "Memory Size and Number of Simultaneously Reproduced Sound Channels".
The musical score and data shown below are for "spring.txt" in the "sndtool\sample\" directory ("Spring" of Vivaldi's The Four Seasons).

"spring.txt"



Enter data as shown below:

```
8    do5                        8    C5
8    mi5                        8    E5
8    mi5                        8    E5
8    mi5                        8    E5
16   re5            or          16   D5
16   do5                        16   C5
4    so5  c                     4    G5  c
8    so5                        8    G5
   :                             :
8    so4  ppp                   8    G4  ppp
8    -1                         8    -1
```

Each line of data represents a single musical note or rest in the format shown below:

Specification of musical notes: <Length of sound> <Pitch> <Control>
Specification of rests: <Length of rest> -1

<Length of sound> and <Pitch> can be written in succession. Spaces and tabs between the two are ignored.

<Length of sound> and <Length of rest>

<Length of sound> and <Length of rest> can be specified in a range from whole notes (or whole rests) to thirty-second notes (thirty-second rests), using the numeric values 1, 2, 4, 8, 16, and 32.

Table 3.1.1 Specification of Notes and Rests

Specified Value	1	2	4	8	16	32
Note	𝅝	𝅗𝅥	♩	♪	𝅘𝅥𝅯	𝅘𝅥𝅰
Rest	𝄻	𝄼	𝄽	𝄾	𝄿	𝅀

<Pitch>

Use sound names and octave numbers (1–7) to specify <Pitch>. Write the sound names and octave numbers in succession without separating them with a space, etc.

Sound name: do, re, mi, fa, so, la, si (lowercase only) or **C, D, E, F, G, A, B (uppercase only)**
do#, re#, fa#, so#, la# or **C#, D#, F#, G#, A# (semitone higher)**

Acceptable specification range: do1/C1 (33 Hz) to **si7/B7 (3,951 Hz)**

Only simple tones can be specified. Write chords as separate files.

<Control>

The "c" and "ppp" in the example input data above add modifications to the musical reproduction. You can use the symbols listed below.

Example:
```
4  C3  c
8  C3
8  C3  ppp
8  C3  pp
8  C3  p
2  C3  b+2
2  C3  b-2
8  C3  f
8  C3  ff
8  C3  fff
```

Use lowercase letters to enter symbols. If multiple control symbols are occur in the same line, the first symbol written is processed while all others are ignored.

c

The tone for which this symbol is written and the tone that follows are played in succession, assuming the keys are held down. This helps specify a dot or tie.
Always enter a space between <Pitch> and c.
Example: Dotted quarter note
```
4    so5  c
8    so5
```

p, pp, ppp

Indicates various degrees of playing a sound softly. The "p" specifies playing somewhat softly, while the "ppp" indicates playing most softly. In the absence of any specification, a value of 128 in the sound volume specification range of 0 (silent) to 255 (loudest) is assumed. Each control is assigned the values p = 104, pp = 80, and ppp = 56 when a sound file is created. Loudness can be fine-adjusted later by correcting the sound file with an editor.

f, ff, fff

Indicates various degrees of playing a sound loudly. The "f" specifies playing rather loudly, while the "fff" indicates playing most loudly. In the absence of any specification, a value of 128 in the sound volume specification range of 0 (silent) to 255 (loudest) is assumed. Each control is assigned the values f = 152, ff = 176, and fff = 200 when a sound file is created. Loudness can be fine-adjusted later by correcting the sound file with an editor.

b+X, b-X

Specifies the direction and amount of pitch bending in the sustained portion of the note. b+ shifts the pitch higher, while b- shifts the pitch lower. The X specifies the amount of this shift in semitone units, using a numeric value. (X = 12 represents one octave.) This facility is effective only for the release time after the envelope key (note) is off. Thus, it has no effect unless you specify gate-off parameters when converting files with txt2snd.exe.

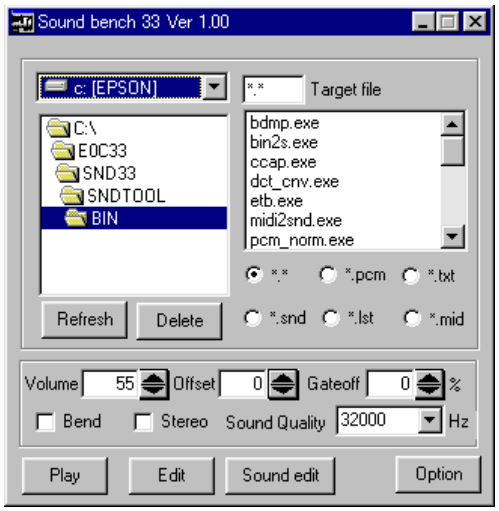
3.1.2 Evaluating Sound Data with sb33

Use of the sound bench "sb33" allows you to convert sound text or standard MIDI files into PCM files for playback on a PC. The basics of using "sb33.exe" are described below:

(1) Starting sb33.exe

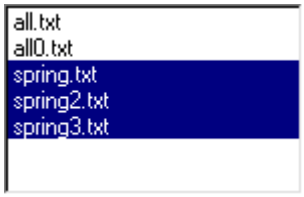

sb33.exe

Double-click the "sb33.exe" icon. To exit, click the [Close] button on the title bar.
The [Sound bench 33] window appears when sb33 starts.



[Sound bench 33] window

(2) Converting and playing sound text files



[Play] button

1. Choose "sndtool\sample\" from the directory list box, then "spring.txt" from the file list box. While holding down the [Ctrl] key, also select "spring2.txt" and "spring3.txt" too. This allows you to evaluate the musical reproduction of multiple channels at the same time.

* The selected text file may be displayed and/or corrected using the [Edit] button after opening it with Notepad. However, if you select multiple files, only the file at the top of the list is opened. You can switch to a different editor using the [Option] button.

2. Set [Gateoff] to 50%. When playing music on a keyboard, for example, release your finger from the key for the second half (50%) of each note.

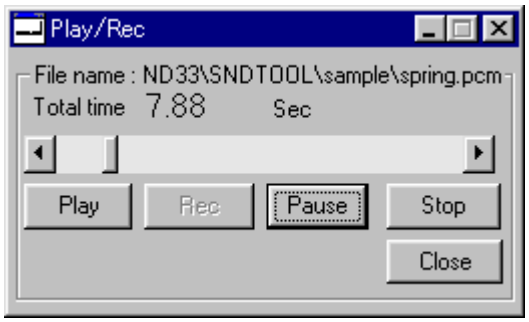
* Besides [Gateoff], there are other selectable parameters: [Volume] to control the sound volume, [Offset] to shift the musical interval in semitone units, [Sound Quality] to determine sound quality (sampling frequency), and [Stereo] to specify sound output in stereo. For more information, refer to Section 4, "SOUND33 Tool Reference".

3. Click the [Play] button.

The sb33 executes the following tools as it creates and begins playing a PCM file:

1) txt2snd.exe	Converts sound text files into SND files. Specifying [Gateoff] is optional for this tool.
2) snd2bin.exe	Converts SND files into binary files.
3) sb33 internal processing	Creates a sound list file.
4) snd2pcm.exe	Creates PCM files for evaluation on a PC and the files to be installed on the actual machine. Specifying [Volume], [Offset], and [Sound Quality] are optional for this tool.

When sb33 begins playing, the [Play/Rec] window is displayed to allow you to control the manner in which the music is played.



[Play] button:	Use this button to restart after stopping playback with the [Stop] or [Pause] button.
[Rec] button:	Disabled for sb33.
[Pause] button:	Temporarily stops playback. Use the [Play] or [Pause] button to start from the point at which you hit [Pause].
[Stop] button:	Stops playback and returns to the beginning of the music.

(3) Fine control for each note

The txt2snd.exe executed first when you press the [Play] button converts a sound text file into an SND file.
Example SND file:

```
24      48      128     // 0            8    do5
24      52      128     // 24           8    mi5
24      52      128     // 48           8    mi5
24      52      128     // 72           8    mi5
:       :        :        :           :     :
24      47      56      // 528          8    si4 ppp
24      43      56      // 552          8    so4 ppp
48      0       0       // 576          4    -1
```

Data for sound length, interval, and volume are added for each musical note in the sound text file. Correct this data using an editor to change the file for more expressive playback. To correct, choose the file name (.snd) from the file list box and click the [Edit] button. Notepad (or alternate editor) starts, automatically opening an SND file. Make the necessary corrections and save the data.
For more information on the contents of the SND file, refer to Section 3.1.3, "SND Files".

Note: To play the SND file with the [Play] button of sb33 after correcting it, always be sure to select the corrected SND file (.snd) before clicking the [Play] button. Selecting the sound text file (.txt) and clicking the [Play] button executes txt2snd.exe, overwriting the SND file.

(4) Specifying a musical instrument and tempo

At step (2), we did not specify tempo or a musical instrument. Thus, the music is played at the default tempo (100) on the default musical instrument (piano). Specifying these parameters requires a sound list file separate from the sound text file, whose contents are much like the one shown below:

Example: spring.lst, a sound list file automatically created by sb33 (monaural)

```
;Tempo
100
;STB wave table directory
..\stb\
;ETB envelove table directory
..\etb\
;Sound file Instrument  Volume  Echo    Etbadj  Offset   Position
spring.bin piano 55 6  3  0  -1
spring2.bin piano 55 6  3  0  -1
spring3.bin piano 55 6  3  0  -1
```

Choose the sound text file and click the [Play] button. sb33 creates a sound list file with default settings and outputs a PCM file for evaluation. You can use this file as a template from which to create a formal file. To correct it, choose the file name (.lst) from the file list box and click the [Edit] button. Notepad (or alternate editor) starts, automatically opening a sound list file. Make the necessary corrections and save the data.

For more information on the contents of the sound list file, refer to Section 3.1.4, "Specifying Tone Qualities Using a Sound List File".

Note: To play the sound list file with the [Play] button of sb33 after making corrections, always select the corrected sound list file (.lst) before clicking the [Play] button. Selecting the sound text file (.txt) or SND file (.snd) and clicking the [Play] button overwrites the corrected sound list file at the default settings.

(5) Converting and playing a standard MIDI file

The SOUND33 tools include a utility (midi2snd.exe) for converting standard MIDI files in SMF0 or SMF1 format to SND and sound list files. This utility is also supported by sb33, allowing you to convert a MIDI file into SND and sound list files and play the converted file by clicking on the [Play] button. However, due to limitations involving such conversions, data in the converted file may not be played exactly as in the original. Be especially careful when using MIDI data created with a MIDI sequencer. For more information on MIDI file conversion results and the various precautions involved, refer to Section 3.1.6, "Converting MIDI Files".

3.1.3 SND Files

A sample sound text file and a sample SND file converted by txt2snd.exe are shown below:
Example: Converted with [Gateoff] = 50%

Sound text file

```
8   C3  ppp
8   C3  pp
8   C3  p
2   C3  b+2
2   C3  b-2
8   C3  f
8   C3  ff
8   C3  fff
2   -1
```

SND file

```
12   24   56    // 1692      8   C3  ppp
12   -1   80    // 1704
12   24   80    // 1716      8   C3  pp
12   -1   80    // 1728
12   24   104   // 1740      8   C3  p
12   -1   80    // 1752
48   24   128   // 1764      2   C3  b+2
9    -2   25    // 1812
39   -2   26
48   24   128   // 1860      2   C3  b-2
9    -2   23    // 1908
39   -2   22
12   24   152   // 1956      8   C3  f
12   -1   80    // 1968
12   24   176   // 1980      8   C3  ff
12   -1   80    // 1992
12   24   200   // 2004      8   C3  fff
12   -1   80    // 2016
48   0    0     // 2028      2   -1
48   -1   80    // 2076
```

The contents of each line of the SND file are described below:

<Length> <Interval> <Volume> //<Position from the beginning> <Content of sound text file>

<Length>	Represents the length of sound relative to a quarter note = 48. Whole note = 192, Half note = 96, Quarter note = 48, Eighth note = 24, Sixteenth note = 12, Thirty-second note = 6
<Interval>	Represents pitch relative to C1 (do1) = 1 and B6 (si6) = 83. Rests are 0.
<Volume>	loudness in the range 0 (silent) to 255 (maximum). As standard (no loudness symbol specified), sound volume is converted to 128; rests are converted to 0. If any loudness symbol is specified, sound volume is converted as follows: p=104, pp=80, ppp=56, f=152, ff=176, fff=200
//<Position from the beginning>	Indicates the position of each musical note or rest by length (quarter note = 48) from the beginning of the file. This numeric value helps ascertain whether musical reproduction between multiple channels (files) is out of synch.

<Length> varies with the value specified for [Gateoff]. In the above example, since [Gateoff] = 50%, the eighth note (length = 24) is halved to 12, with the remaining length of 12 assumed to be the sustained note (release time). For some types of musical instruments, a sustained tone is output even after key-off.

```
12  24   56   // 1692        8   C3  ppp
12  -1   80   // 1704        ← Release time of the length specified by [Gateoff]
```

Any line of the sound text file in which the control "c" is written is handled along with the next line as a single note in length. Thus, even when [Gateoff] is set, no release times are inserted. Lines accompanied by "c" do not appear in <Content of sound text file> of the SND file.
In lines in which release times are added, <Interval> = -1 and <Volume> = 80.

In the lines of the sound text file in which "b+X" and "b-X" are specified, if [Gateoff] is specified, the following two lines of key-off part are written:

```
48  24   128  // 1764          2   C3  b+2
9   -2   25   // 1812          ← Pitch increased by 1 (X/2; X = 2)
39  -2   26                    ← Pitch increased by 2 (X)
```

The <Interval> part is changed to -2, and a description is entered for the <Volume> part in which the pitch changes in two steps.

Specifying "b+1" and "b-1" adds one line of key-off part.

```
48  24   128  // 1764          2   C3  b+1
48  -2   25   // 1812
```

3.1.4 Specifying Tone Qualities Using a Sound List File

The sound list file is used to write tempos at which to play music, the directory that contains sound source data, and information on each channel. This file is used for input to snd2pcm.exe, from which the tool produces batch files necessary to create PCM files for evaluation on a PC as well as data to be installed on the actual system.
When you run sb33, it creates a sound list file with basic settings from the specified sound text file or MIDI file. Use it after making the desired corrections.

Note: When you choose sound text files or a MIDI file and click the [Play] button in sb33, it automatically creates sound list files that overwrite any existing files of the same name. The sound list file created in this way by sb33 is named after the selected sound text file (or the file at the top of the list when multiple files are selected) or MIDI file by adding the extension ".lst". If you created another sound list file separately for evaluation by sb33, always select the sound list file before clicking the [Play] button. In this case, SND files must have been created in advance for each channel.

Shown below are the contents of a sound list file:
Example: spring.lst, an example sound list file automatically created by sb33 (stereo)

```
;Tempo
100
;STB wave table directory
..\stb\
;ETB envelove table directory
..\etb\
;Sound file Instrument  Volume  Echo    Etbadj  Offset   Position
spring.bin piano 70 6  3  0  50
spring2.bin piano 70 6  3  0  50
spring3.bin piano 70 6  3  0  50
```

(1) Specifying a tempo

```
;Tempo      ← Lines beginning with a semicolon (";") are comment lines.
100          ← Tempo
```

Specify a tempo at which to play music using integer values from 30 to 300. This value represents the number of quarter notes to be played per minute. The following lists typical speed designations and the approximate set values recommended for tempo.

Speed designation	Range of tempo	Approximate set value
Lento, Largo, Grave	30 to 50	40
Adagio	40 to 60	50
Larghetto	50 to 60	55
Adagietto	60 to 70	65
Andante	65 to 75	70
Andantino	70 to 80	80
Moderato	80 to 90	90
Allegretto	90 to 100	100
Allegro, Conmoto	100 to 130	120
Vivo	110 to 140	130
Vivace	140 to 170	150
Presto	170 to 200	180
Prestissimo	200 or higher	200

(2) Specifying directories of tone quality data

```
;STB wave table directory
..\stb\      ← Directory for WAVE data
;ETB envelove table directory
..\etb\      ← Directory for envelope data
```

Specify the directories that contain the WAVE data and envelope data for the musical instrument used, using either a relative or absolute path. The tone quality data that comes standard with the package are copied to "e0c33\snd33\sndtool\stb\" and "e0c33\snd33\sndtool\etb\" on your hard disk.

(3) Channel information

```
;Sound file Instrument  Volume  Echo  Etbadj  Offset  Position
spring.bin    piano       70     6     3       0        50
spring2.bin   piano       70     6     3       0        50
spring3.bin   piano       70     6     3       0        50
```

Specify the following information for each channel.

Sound file This is the sound binary file created from SND files by conversion with snd2bin.exe.

Instrument Specify the name of the musical instrument. For the types of musical instruments supported by the standard SOUND33 library, refer to Section 3.1.5, "List of Standard Supported Musical Instruments".

Volume This value specifies the sound volume that applies only for the specific channel. Set a value in the range 0 (silent) to 255 (maximum).
For sound list files automatically created by sb33, the values set from the [Volume] box of the [Sound bench 33] window are entered directly here.

Echo Specify the amount of delay when you want to generate a reverberating tone. Sound generation is delayed by 1/6 of the thirty-second note per increment of integer value 1 to produce an echo. The value 6 in the above example specifies an amount of delay equal to the length of the thirty-second note. Note that when producing reverberating tones, the tool plays the same content as the actual sound after halving the volume.

Etbadj This is the coefficient of correction for the envelope (the curve representing changes in sound volume from when sound starts sounding till when it goes out). Specify using integer values from 0 to 4.
When you specify 0, both bass and treble are played with the same envelope. If you specify any value equal to or greater than 1, sound is attenuated increasingly gradually for low-pitched tones or increasingly rapidly for high-pitched tones, relative to C3/do3 (262 Hz). Attenuation is greatest for the value 4, changing two-fold every octave. In the default sb33 settings, "piano" is selected as the musical instrument, and Etbadj is set to 3. For information on setting appropriate values, refer to Section 3.1.5, "List of Standard Supported Musical Instruments", which provides the Etbadj values used to convert General MIDI tone qualities to those of the SOUND33 library.

Offset When shifting a musical interval over an entire channel in question, specify the amount of shift in semitone units using integer values. The specification range is ±5 octaves (-60 to 60).
Specifying 0 plays tone data with the original data unchanged. If this specification falls outside the range C1/do1 to B6/si6, tone data are played as C1/do1 or B6/si6.

Position Specify the position of the musical instrument for stereo playback, using integer values from 0 to 100. The values 0, 50, and 100 specify the leftmost, middle, and rightmost positions, respectively. Specify -1 for monaural playback.

To play only a specific channel for sound evaluation by sb33, you can comment out lines you do not want to play by placing a semicolon (;) at the beginning of the lines.
In the SOUND33 library, the base sound is set low. To emphasize it, raise the interval by one octave (+12), or increase sound volume to around the level at which the waveform will not be clipped.

3.1.5 List of Standard Supported Musical Instruments

The SOUND33 library comes with tone quality data for a total of 20 types of musical instruments, including 8 types of percussion instruments.

General instruments: **piano, harpc (harpsichord), celesta, organ, guitar, bguitar (bass), eguitar (electric guitar), violin, trumpet, clarinet, piccolo, flute**

Percussion instruments: **bdrum (bass drum), stick (side stick), snare, tom, ccymbal (crash cymbal1), highhat (hi-hat), bongo, triangle**

The list below shows the relationship between the instrument names in General MIDI and those in SOUND33. The values Echo and Etbadj are used when creating sound list files from standard MIDI files by converting them with midi2snd.exe. Instruments marked "unknown" in SOUND33 are not supported.

Table 3.1.2 List of General Instruments

General MIDI instrument name	Name for SOUND33	Echo	Etbadj	General MIDI instrument name	Name for SOUND33	Echo	Etbadj
1 Acoustic Grand Piano	piano	12	2	41 Violin	violin	6	0
2 Bright Acoustic Piano	piano	6	2	42 Viola	violin	6	0
3 Electric Grand Piano	piano	12	2	43 Cello	violin	6	0
4 Honkey-tonk Piano	piano	6	2	44 Contrabass	violin	6	0
5 Electric Piano1	piano	0	2	45 Tremolo Strings	violin	6	0
6 Electric Piano2	piano	0	2	46 Pizzicato Strings	violin	6	0
7 Harpsichord	harpc	6	0	47 Orchestral Harp	unknown	0	0
8 Clavi	unknown	6	0	48 Timpani	unknown	0	0
9 Celesta	celesta	0	0	49 String Emsemble1	violin	6	0
10 Glockenspiel	unknown	0	0	50 String Emsemble2	violin	6	0
11 Music Box	celesta	0	0	51 Synth String1	violin	6	0
12 Vibraphone	celesta	0	0	52 Synth String2	violin	6	0
13 Marinmba	celesta	0	0	53 Choir Aahs	unknown	0	0
14 Xylophone	celesta	0	0	54 Voice Oohs	unknown	0	0
15 Tubular Bells	celesta	0	0	55 Synth Vox	unknown	0	0
16 Dulcimer	unknown	0	0	56 Orchestra Hit	unknown	0	0
17 Drawbar Organ	organ	6	0	57 Trumpet	trumpet	6	0
18 Percussive Organ	organ	6	0	58 Trombone	trumpet	6	0
19 Rock Organ	organ	6	0	59 Tuba	trumpet	6	0
20 Church Organ	organ	12	0	60 Muted Trumpet	trumpet	6	0
21 Reed Organ	organ	6	0	61 French Horn	trumpet	6	0
22 Accordion	organ	6	0	62 Brass Section	trumpet	6	0
23 Harmonica	unknown	0	0	63 Synth Brass1	trumpet	6	0
24 Tango Accordion	organ	6	0	64 Synth Brass2	trumpet	6	0
25 Acoustic Guitar(nylon)	guiter	0	1	65 Soprano Sax	trumpet	6	0
26 Acoustic Guitar(steel)	guiter	0	1	66 Alto Sax	trumpet	6	0
27 Electric Guitar(jazz)	eguiter	0	1	67 Tenor Sax	trumpet	6	0
28 Electric Guitar(clean)	eguiter	0	1	68 Baritone Sax	trumpet	6	0
29 Electric Guitar(muted)	eguiter	0	1	69 Oboe	unknown	6	0
30 Overdriven Guitar	eguiter	0	1	70 English Horn	unknown	6	0
31 Distortion Guitar	eguiter	0	1	71 Bossoon	unknown	6	0
32 Guitar Harmonics	eguiter	0	1	72 Clarinet	clarinet	6	0
33 Acoustic Bass	bguiter	0	1	73 Piccolo	piccolo	0	0
34 Electric Bass(finger)	bguiter	0	1	74 Flute	flute	6	0
35 Electric Bass(pick)	bguiter	0	1	75 Recorder	flute	6	0
36 Fretless Bass	bguiter	0	1	76 Pan Flute	flute	6	0
37 Slap Bass1	bguiter	0	1	77 Blown Bottle	flute	6	0
38 Slap Bass2	bguiter	0	1	78 Shakuhachi	piccolo	12	0
39 Synth Bass1	bguiter	0	1	79 Whistle	unknown	6	0
40 Synth Bass2	bguiter	0	1	80 Ocarina	unknown	6	0

General MIDI instrument name	Name for SOUND33	Echo	Etbadj	General MIDI instrument name	Name for SOUND33	Echo	Etbadj
81 Lead1(square)	unknown	0	0	105 Sitar	unknown	0	0
82 Lead2(sawtooth)	unknown	0	0	106 Banjo	unknown	0	0
83 Lead3(calliope)	unknown	0	0	107 Shamisen	unknown	0	0
84 Lead4(chiff)	unknown	0	0	108 Koto	unknown	0	0
85 Lead5(charang)	unknown	0	0	109 Kalimba	unknown	0	0
86 Lead6(voice)	unknown	0	0	110 Bag Pipe	unknown	0	0
87 Lead7(fifths)	unknown	0	0	111 Fiddle	unknown	0	0
88 Lead8(bass + lead)	unknown	0	0	112 Shanai	unknown	0	0
89 Pad1(new age)	unknown	0	0	113 Tinkle Bell	unknown	0	0
90 Pad2(warm)	unknown	0	0	114 Agogo	unknown	0	0
91 Pad3(polysynth)	unknown	0	0	115 Steel Drums	unknown	0	0
92 Pad4(choir)	unknown	0	0	116 Woodblock	unknown	0	0
93 Pad5(bowed)	unknown	0	0	117 Taiko	unknown	0	0
94 Pad6(metallic)	unknown	0	0	118 Melodic Tom	unknown	0	0
95 Pad7(halo)	unknown	0	0	119 Synth Drum	unknown	0	0
96 Pad8(sweep)	unknown	0	0	120 Reverse Cymbal	unknown	0	0
97 Fx1(rain)	unknown	0	0	121 Guitar Fret Noise	unknown	0	0
98 Fx2(soundtrack)	unknown	0	0	122 Breath Noise	unknown	0	0
99 Fx3(crystal)	unknown	0	0	123 Seashore	unknown	0	0
100 Fx4(atmosphere)	unknown	0	0	124 Bird Tweet	unknown	0	0
101 Fx5(brightness)	unknown	0	0	125 Telephone Ring	unknown	0	0
102 Fx6(goblins)	unknown	0	0	126 Helicopter	unknown	0	0
103 Fx7(echoes)	unknown	0	0	127 Applause	unknown	0	0
104 Fx8(sci-fi)	unknown	0	0	128 Gunshot	unknown	0	0

Table 3.1.3 List of Percussion Instruments

General MIDI instrument name	Name for SOUND33	General MIDI instrument name	Name for SOUND33
35 Acoustic Bass Drum64 start at 35 (B1)	bdrum	59 Ride Cymbal2	ccymbal
36 Bass Drum1	bdrum	60 Hi Bongo	bongo
37 2Side Stick	stick	61 4Low Bongo	bongo
38 Acoustic Snare	snare	62 Mute Hi Conga	bongo
39 2Hand Clap	stick	63 4Open Hi Conga	bongo
40 Electric Snare	snare	64 Low Conga	bongo
41 Low Floor Tom	tom	65 High Timbale	unknown
42 2Closed Hi-Hat	highhat	66 4Low Timbale	unknown
43 High Floor Tom	tom	67 High Agogo	unknown
44 2Pedal Hi-Hat	highhat	68 4Low Agogo	unknown
45 Low Tom	tom	69 Cabasa	unknown
46 2Open Hi-Hat	highhat	70 4Maracas	unknown
47 Low-Mid Tom	tom	71 Short Whistle	unknown
48 Hi-Mid Tom	tom	72 Long Whistle	unknown
49 3Crash Cymbal1	ccymbal	73 5Short Guiro	unknown
50 High Tom	tom	74 Long Guiro	unknown
51 3Ride Cymbal1	ccymbal	75 5Clavas	unknown
52 Chinese Cymbal	ccymbal	76 Hi Wood Block	unknown
53 Ride Bell	triangle	77 Low Wood Block	unknown
54 Tambourine	stick	78 5Mute Cuica	unknown
55 Splash Cymbal	ccymbal	79 Open Cuica	unknown
56 3Cowbell	unknown	80 5Mute Triangle	triangle
57 Crash Cymbal2	ccymbal	81 Open Triangle	triangle
58 3Vibraslap	unknown	–	–

Note: The tone quality data for percussion instruments are created at relatively short sound lengths, in consideration of the data size on the actual system. When played as long notes, the sound may break in the middle.

3.1.6 Converting MIDI Files

The SOUND33 tools include a utility (midi2snd.exe) for converting standard MIDI files in SMF0 or SMF1 format into SND and sound list files. This facility is also supported by sb33. Selecting a MIDI file and clicking the [Play] button converts it into SND and sound list files, which is then played back.
This means that a general MIDI sequencer can be used to create the sound data for playback. However, because conversion is subject to the limitations described below, the data in the converted file may not be played exactly as in original.

Limitations

1. Only MIDI files in SMF0 or SMF1 format can be converted.

2. Channel information for musical instruments not supported by SOUND33 (those marked "unknown" in Tables 3.1.2 and 3.1.3) are written out as comments in the sound list file. To play the data, you must rewrite it for another instrument. Data on a single track in the MIDI file may be converted into multiple channels. If the conversion results in more channels than can be played simultaneously, make sure that the channels with the least effect on musical reproduction are commented out in the sound list file.
   Example: Sound list file converted from a MIDI file

```
;Tempo
98
;STB wave table directory
..\stb\
;ETB envelove table directory
..\etb\
;Sound file    Instrument  Volume  Echo  Etbadj  Offset  Position
tr1ch2_1.bin   piano       c0      0     2       0       50
tr1ch2_2.bin   piano       c0      0     2       0       50
tr1ch2_3.bin   piano       c0      0     2       0       50
tr1ch2_4.bin   piano       c0      0     2       0       50
;tr2ch3_1.bin  unknown[Pad8(sweep)]     c4      0       0       0     50
;tr2ch3_2.bin  unknown[Pad8(sweep)]     c4      0       0       0     50
;tr2ch3_3.bin  unknown[Pad8(sweep)]     c4      0       0       0     50
```

3. The SOUND33 library does not support changes in tempo in the middle of musical reproduction. If multiple tempo change events are found in the MIDI data, a warning is generated during conversion, and the first settings are used.

4. The SOUND33 library does not support changes in sound volume in the middle of musical reproduction. If sound volume in the MIDI data to be converted into a single channel changes in the middle, a warning is generated during conversion, and the last settings are used.

5. If any MIDI file whose timebase is not 48 (quarter note = 48) is converted, a warning is generated. A timebase above 48 exceeds the resolution of SOUND33, resulting in errors. Although midi2snd.exe can make adjustments to eliminate cumulative errors, sound may drift slightly, compared to musical reproduction by a MIDI sequencer.

When using a general MIDI sequencer, make sure the tempo and sound volume in the data you create are constant, and that the timebase is 48.

3.1.7 Creating Tone Quality Data

SOUND33 supports 20 types of musical instruments (see Section 3.1.5). You can also add custom-created tone quality data.

Composition of tone quality data

Tone quality data from the SOUND33 library consists of waveform data sampled from instrumental sounds and envelope data representing curves of sound volume changes from the point at which the sound is sounded to the point at which it fades completely. The waveform data for standard instruments supported are provided in the "snd33\sndtool\stb\" directory, while envelope data is located in the "snd33\sndtool\etb\" directory.
For example, data for pianos consists of the following:

snd33\sndtool\stb\08\stb_piano.bin ← 8 kHz sampled piano waveform data
snd33\sndtool\stb\16\stb_piano.bin ← 16 kHz sampled piano waveform data
snd33\sndtool\stb\22\stb_piano.bin ← 22 kHz sampled piano waveform data
snd33\sndtool\stb\32\stb_piano.bin ← 32 kHz sampled piano waveform data
snd33\sndtool\etb\etb_piano.bin ← Piano envelope data

The file names are "stb_<instrument name>.bin" and "etb_<instrument name>.bin".

Creating WAVE data

Because the high-tone range has fewer harmonic components than the low-tone range, we recommend preparing separate sound sources for the treble and bass parts for each musical instrument at almost the same length.
Given below is the procedure for creating WAVE data. In the steps in which no specific tool names are given, you can use any commercially-available sound editor.

1. Sample the instrument sound.
   To smooth the curves of the extracted waveform data, sample the PCM data of the instrument sound before extraction so that a single waveform of a low tone consists of 1,000 to 2,000 points. (Example: Load 44 kHz data as 8 kHz data with a sound editor and upsample at 48 kHz.)
2. Extract one waveform of data. (Example: \smplstb\sample1.pcm)
   A section of waveform from a rise from 0 toward the positive direction to the next rise above 0 is required.
3. Using dct_cnv.exe, adjust the extracted waveform until it consists of 1,000 to 2,000 points.
   (Example: \smplstb\sample2.pcm)
   When creating two sound sources, one for high-pitched tones and one for low-pitched tones, make sure the length of the respective sound sources are nearly identical.
4. Using pcm_norm.exe, normalize the waveform data so that its maximum amplitude is 90%.
   (Example: \smplstb\sample3.pcm)
5. Adjust the start and end positions of data as shown below: (Example: \smplstb\sample4.pcm)
   - The data must comprise a waveform in which it always starts from 0, rising toward the positive direction.
   - The data immediately preceding the last point must be nearly 0 (i.e., the end of one cycle).
     The last data consists of one cycle + one point of data.
6. When preparing another sound source on the high-tone side, repeat steps 1 to 5.
   (Example: \smplstb\sample5.pcm)
   To prepare separate sound sources, we recommend creating the low-tone sound sources for C1 to C4 or C5 and high-tone sound sources for C5 or C6 to C7.
7. Execute \smplstb\stb.bat to generate sound sources.
   Because stb.bat has been made for the sample data, change the parameters as required for your data processing needs before executing it. This batch file executes the following tools as it creates 32 kHz, 22 kHz, 16 kHz, and 8 kHz sampled sound source data (stb_<instrument name>.bin).
   1) pcm2stb.exe Creates one-octave waveform data referenced to C2.
   2) stb12.exe Creates WAVE data.
   3) stbadd12.exe Adds waveform data to WAVE data (treble part).

   For more information on each tool, refer to Section 4, "SOUND33 Tool Reference".

The contents of stb.bat are shown below. When using stb.bat for data processing, change the parameters displayed in bold. Depending on your current directory, you may also need to change PATH to the tool and stb directory.

stb.bat

```
set binpath=..\bin\           ← Relative path to the tool
set stb22path=..\stb\22\      ← Relative path to the created data
set stb32path=..\stb\32\
set stb16path=..\stb\16\
set stb8path=..\stb\08\
set lowdata= sample4.pcm      ← Low tonal data (sample4 sampling = 2,026 points)
set highdata= sample5.pcm     ← High tonal data (sample4 sampling = 2,003 points)
set inst=flute                ← Instrument name
                                                    Used for 32 kHz sampling
%binpath%pcm2stb 2003 984 %highdata%← High tonal side consists of two waveforms of data
                                                    C1–C5 data, 984 = 32000/65*2
%binpath%pcm2stb 2026 492 %lowdata% ← Low tonal side consists of one waveform of data
                                                    C6–C7 data, 492 = 32000/65
%binpath%stb12 %lowdata% org.bin
%binpath%stbadd12 org.bin 5 %highdata% %stb32path%stb_%inst%.bin

%binpath%pcm2stb 2003 678 %highdata%          Used for 22 kHz sampling
%binpath%pcm2stb 2026 339 %lowdata%
%binpath%stb12 %lowdata% org.bin
%binpath%stbadd12 org.bin 5 %highdata% %stb22path%stb_%inst%.bin

%binpath%pcm2stb 2003 492 %highdata%          Used for 16 kHz sampling
%binpath%pcm2stb 2026 246 %lowdata%
%binpath%stb12 %lowdata% org.bin
%binpath%stbadd12 org.bin 5 %highdata% %stb16path%stb_%inst%.bin

%binpath%pcm2stb 2003 246 %highdata%          Used for 8 kHz sampling
%binpath%pcm2stb 2026 123 %lowdata%
%binpath%stb12 %lowdata% org.bin
%binpath%stbadd12 org.bin 5 %highdata% %stb8path%stb_%inst%.bin

del org.bin
```

Creating envelope data

When creating new WAVE data, you must create corresponding envelope data.

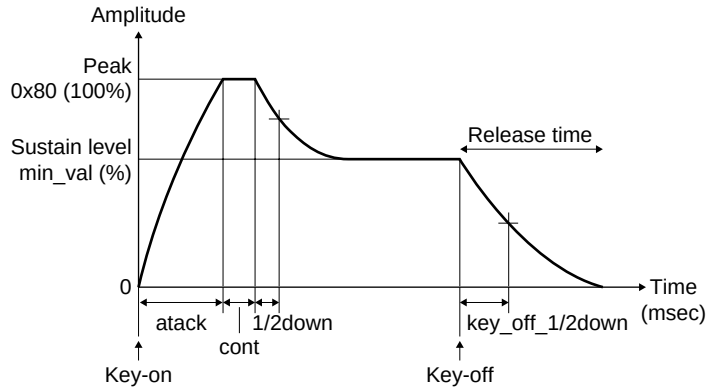


Figure 3.1.2 Envelope

From the point at which a sound begins sounding until it fades, an envelope represents a curve of changing sound volume. This curve is a primary factor in identifying a musical instrument by ear.
Use etb.exe to create envelope data. The command line format of stb.exe is shown below:
>etb attack cont 1/2down min_val am_freq am_ratio peak key_off_1/2down outfile.bin

Shown below are the contents of each parameter:

attack Attack time (msec)
Specify the duration in which sound rises from key-on to peak (0x80).

cont Peak continuance time (msec)
Specify a duration for which time the peak continues.

1/2down Decay time adjustment (msec)
Specify a duration from the peak to the point at which the signal level attenuates to twice the value of the sustain level (min_val).

min_val Sustain level (0–100%)
Specify a signal level that is retained until the key is turned off after attenuating from the peak as a percent value relative to the peak (= 100%).

am_freq Vibrato rate (AM modulation period, msec)
When creating vibrato, specify the velocity at which the pitch is to waver as an AM modulation period.

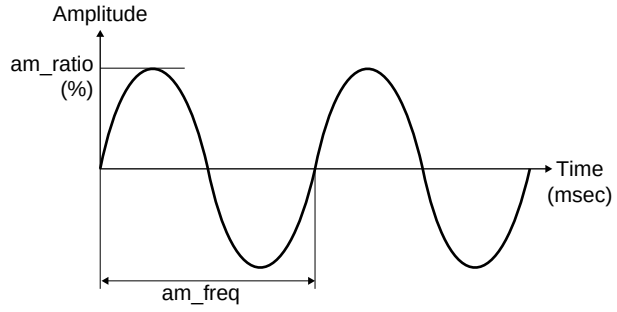


Figure 3.1.3 Specifying Vibrato

am_ratio Vibrato depth (AM modulation amplitude ratio, 0–50%)
When creating vibrato, specify the wavering width of pitch as a percent value relative to the envelope-processed final waveform.

peak Peak adjustment (1.0–1.9)
To express a quickly-rising sharp tone like that of a trumpet, specify a multiple for the peak value.

key_off_1/2down Release time adjustment (msec)
Specify a duration during which the signal level is attenuated to 1/2 of the sustain level after key-off.

outfile.bin Output file name (envelope table binary file)
For the output file name, use "etb_<instrument name>.bin" to match it to the WAVE data instrument name and output the file to the envelope data directory "\sndtool\etb\".

The envelope data is created as a table in which key-on and key-off periods, respectively, are divided into 256 steps, with the amplitude correction value for each step stored in the table as unsigned char types. The table data is multiplied relative to x1.0-fold for the peak value 0x80 to obtain the playback volume (specified in SND file). If the vibrato parameter is set to a value other than 0, AM modulation arithmetic is also performed when calculating the envelope.

Registering to the instrument map file

A list of musical instruments shown in Tables 3.1.2 and 3.1.3 is provided as a text file "snd_inst.map" in the "\sndtool\bin\" directory. Since this file is used in snd2pcm.exe, etc., register new tone quality data (if any) in this list.

snd_inst.map

```
[Instrument Section 1]      (General instruments)
piano       12      2 // 1 Acoustic Grand Piano
piano       6       2 // 2 Bright Acoustic Piano
piano       12      2 // 3 Electric Grand Piano
piano       6       2 // 4 Honkey-tonk Piano
piano       0       2 // 5 Electric Piano1
piano       0       2 // 6 Electric Piano2
harpc       6       0 // 7 Harpsichord
unknown     6       0 // 8 Clavi
celesta     0       0 // 9 Celesta
unknown     0       0 // 10 Glockenspiel
   :        :       :       :
unknown     0       0 // 127 Applause
unknown     0       0 // 128 Gunshot
[Instrument Section 2]      (Percussion instruments)
bdrum                 // 35 Acoustic Bass Drum64 start at 35 (B1)
bdrum                 // 36 Bass Drum1
stick                 // 37 2Side Stick
snare                 // 38 Acoustic Snare
   :                        :
unknown               // 79 Open Cuica
triangle              // 80 5Mute Triangle
triangle              // 81 Open Triangle
```

Shown at the beginning of each line are the instrument names used in the SOUND33 library. The names following // are General MIDI instrument names. For general instruments, Echo and Etbadj values are entered between the two, which are used when creating sound list files from MIDI files.
When creating new tone quality data, rewrite the SOUND33 instrument name and the Echo and Etbadj values corresponding to the General MIDI instrument closest to the tone quality created.
Example: When creating data equivalent to a grand piano in General MIDI using an instrument name "user"

```
[Instrument Section 1]
user        12      2 // 1 Acoustic Grand Piano ← Register etb_user.bin and stb_user.bin
piano       6       2 // 2 Bright Acoustic Piano
```

3.1.8 Converting Sound Data to Assembly Source Files

To include the created sound data and tone quality data in, or to link such data to your program, generate assembly source files for use with the E0C33 Assembler. We will explain this process, starting with the conversion of sound text files, discussing one tool at a time, instead of using sb33. You can skip processes executed by sb33 when using data created by sb33 (i.e., steps 1 to 7 below can be performed by sb33).
Perform all of the following actions from the DOS prompt.

1. Create sound text files for all channels.

2. Using "txt2snd.exe", convert sound text files into SND files. Perform this process for all the sound text files you have created.
   Example: `DOS>txt2snd 50 spring.txt spring.snd`

   In this example, "spring.txt" is converted into "spring.snd", assuming a gate-off period = 50%.

3. Correct the SND files as necessary.

4. Using "snd2bin.exe", convert SND files into sound binary files. Repeat for all created SND files.
   Example: `DOS>snd2bin spring.snd spring.bin`

   In this example, "spring.snd" is converted into "spring.bin".

5. Create a sound list file.

6. To use user-defined tone quality, create the tone quality data here. (Refer to Section 3.1.7, "Creating Tone Quality Data".)

7. Execute "snd2pcm.exe".
   Example: `DOS>snd2pcm spring.lst 70 32 0 spring`

   In this example, "snd2pcm.exe" is executed after specifying "spring.lst" for the sound list file, 0x70 for the sound volume, 32 kHz for the sampling frequency, and 0 for the tone offset.

8. Execute the batch file "xxxx_data.bat" generated by "snd2pcm.exe" to convert sound binary files into an assembly source file.
   Example: sndlib\demo2\spring_data.bat

```
set binpath=..\..\sndtool\bin\
set sndpath=..\..\sndtool\sample\

%binpath%bin2s -l spring %sndpath%spring.bin > snd_spring.s
%binpath%bin2s -l spring2 %sndpath%spring2.bin >> snd_spring.s
%binpath%bin2s -l spring3 %sndpath%spring3.bin >> snd_spring.s
```

   In this example, "spring.bin", "spring2.bin", and "spring3.bin" are converted into "snd_spring.s". The data for the respective files (channels) are generated using "spring", "spring2", and "spring3" as the global symbols, due to the "bin2s.exe" -l option.
   Example: sndlib\demo2\snd_spring.s

```
            .global spring
            .align  2
spring:
            .byte   0x12 0x33 0x18 0x30 0x80 0x18 0x34 0x80
            .byte   0x18 0x34 0x80 0x18 0x34 0x80 0x0c 0x32
                                :
            .byte   0x00 0x00 0x00
; total 83 bytes data
                                :
```

9. Execute the batch file "xxxx_inst.bat" generated by "snd2pcm.exe" to convert tone quality data into assembly source files.
   Example: sndlib\demo2\spring_inst.bat

```
set binpath=..\..\sndtool\bin\
set stbpath=..\..\sndtool\stb\32\
set etbpath=..\..\sndtool\etb\

%binpath%bin2s -l stb_piano %stbpath%stb_piano.bin > stb_spring.s
%binpath%bin2s -l etb_piano %etbpath%etb_piano.bin > etb_spring.s
```

   The WAVE data for piano is converted into "stb_spring.s", and the envelope data is converted into "etb_spring.s". The respective data is generated using "stb_piano" and "etb_piano" as the global symbols due to the "bin2s.exe" -l option.
   Example: sndlib\demo2\stb_spring.s

```
            .global stb_piano
            .align  2
stb_piano:
            .byte   0x11 0x33 0x00 0x53 0x50 0x01 0x00 0x00
            .byte   0xec 0x01 0xff 0x00 0x3c 0x03 0x00 0x00
                                :
            .byte   0x3c 0xf3 0x4e 0xff
; total 13844 bytes data
```

   Although only the data written in the sound list file for piano is converted in this example, you can add other tone quality data by modifying the batch file. To do so, you also need to correct the "setbl.c" generated in conjunction with the above batch file using "snd2pcm.exe".
   Example: sndlib\demo2\setbl.c

```
#include "snd.h"

extern unsigned char stb_piano[];
extern unsigned char etb_piano[];
const struct SETBL setbl_piano = {
    &stb_piano[0],
    &etb_piano[0],
    3
};
```

   1. Pointer to the WAVE table
   2. Pointer to the envelope table
   3. Structure of the tone quality data
      Pointer to the WAVE table
      Pointer to the envelope table
      Value of Etbadj

   Write lines 1–3 as many times as the number of tone qualities added. A batch file (all_inst.bat) for converting all standard supported instruments into assembly source files is provided, along with setbl.c, in "sndtool\utility\inst_all\".

Since all these tools can be executed from the DOS prompt, they can also be run from a batch file. Refer to the batch files in the "sndtool\sample\" directory.

3.2 Creating User Programs and Linking with SOUND33 Library

You can play sound on the E0C33 chip by calling SOUND33 library functions. For more information on the SOUND33 library and example programs, refer to Section 5, "SOUND33 Library Reference".

You can include the sound data you created and the assembly source files of WAVE table data and envelope table data, as well as the C source files created by "snd2pcm.exe", in the user program or link them along with the SOUND33 library after compiling/assembling.

The procedure for executing sample programs using the DMT board is provided in the Appendix as reference.

4 SOUND33 Tool Reference

This section describes the functions of each SOUND33 tool and explains how to use them.

4.1 Outline of SOUND33 Tools

SOUND33 tools are PC software applications for creating and evaluating the sound ROM data to be written to E0C33 Family chips. All the tools run under Windows 95/98, Windows NT 4.0, or later Windows versions. For more information on the operating environment, see Section 2.1, "Operating Environment".
All SOUND33 tools and related files are found in the "sndtool" folder (directory).
The configuration of SOUND33 tools and the procedure for creating sound ROM data are shown in Figure 4.1.1.

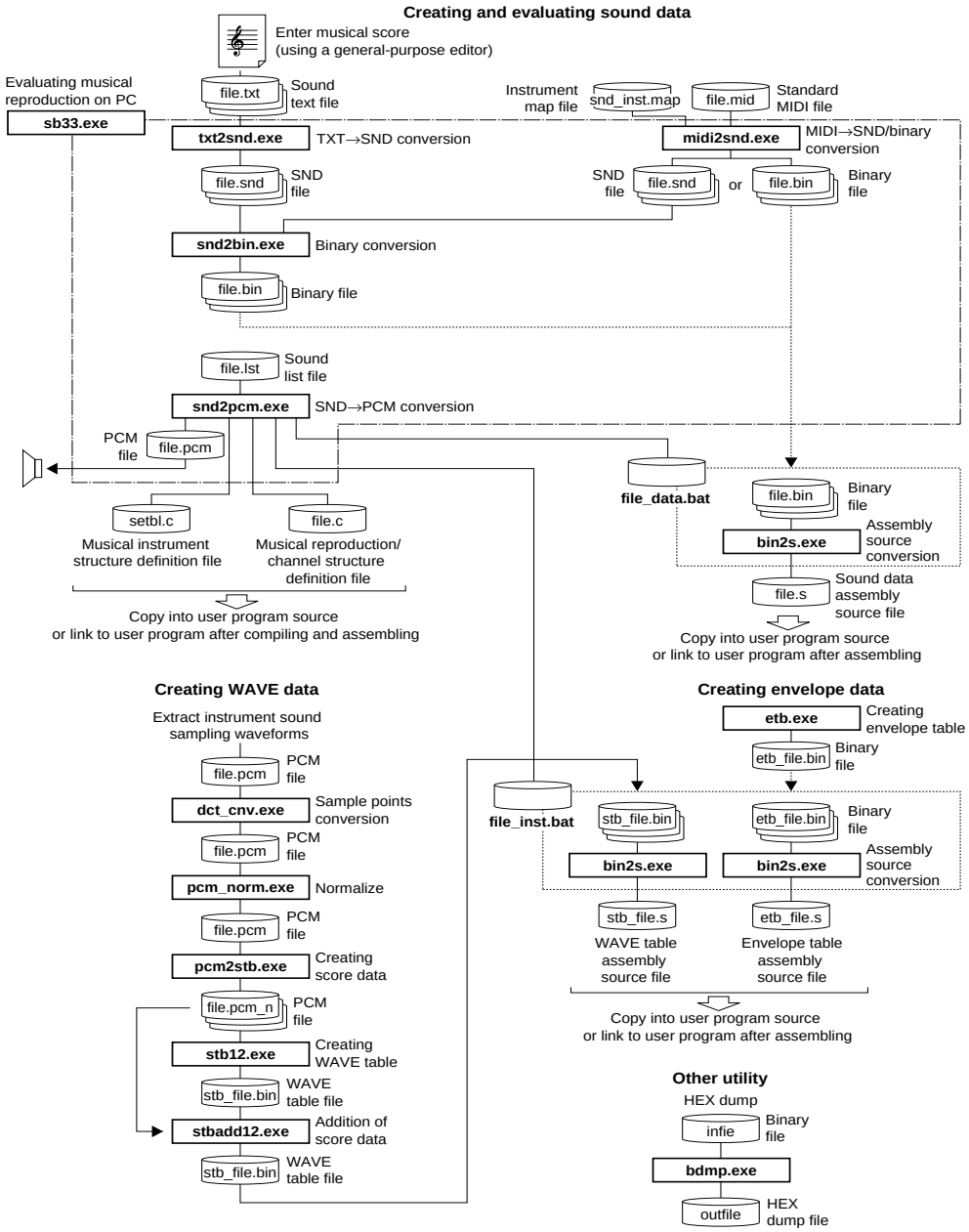


Figure 4.1.1 Flowchart for Creating Sound ROM Data

Note: The PCM files handled by SOUND33 tools are the 8–32 kHz, 16-bit row data in little-endian format.

List of SOUND33 tools

The SOUND33 tools are a series of applications for generating PCM files or assembly source files for the E0C33 from sound text or standard MIDI files. Each tool is a 32-bit application executable from the DOS prompt. Each tool can also be run from batch or make files.
The sound bench "sb33" is a 32-bit Windows GUI application that allows you to convert sound text/MIDI files into PCM files for playback, all from a single PC window. The SOUND33 tools are listed in Table 4.1.1 below.

Table 4.1.1 List of Sound ROM Data Creation Tools

Tool	Function
sb33.exe	A Windows GUI application for creating sound data, as well as for playing and evaluating sound data on a PC.
txt2snd.exe	Converts sound text files into SND files.
snd2bin.exe	Converts SND files (text format) into binary files.
midi2snd.exe	Converts standard MIDI files into SND and sound list files.
snd2pcm.exe	Creates PCM files for evaluation on a PC or batch files for downloading to ROM from sound list files.
bin2s.exe	Converts binary data files into assembly source files.
bdmp.exe	Produces hexadecimal dump output from binary data files.
pcm2stb.exe	Creates one-octave sound source data from PCM files.
stb12.exe	Creates WAVE table data from sound source data.
stbadd12.exe	Adds sound source data to a WAVE table.
etb.exe	Creates envelope data.
dct_cnv.exe	A down-sampler for converting WAV or PCM files to any sampling rate. While you can substitute any commercially-available sound editor, you must take care to avoid sound quality degradation.
pcm_norm.exe	Normalizes PCM data to a specified amplitude.
ccap.exe	Prepares the messages output by a tool during its execution as files. For more information on this tool, refer to the "E0C33 Family C Compiler Package Manual".

* The PCM files handled by SOUND33 tools are 8 kHz, 16 kHz, 22 kHz, or 32 kHz sampled, 16-bit row data in little-endian format, unless otherwise specified.

4.2 Description of Each SOUND33 Tool

This section describes the function of each SOUND33 tool and explains how to use them. For the "sb33.exe", however, see Section 4.3.
Start each tool from the DOS prompt. When a tool is started without specifying command line parameters, Usage is displayed. In the explanation of command lines, [ ] denotes options that may be omitted. The parameters written in *italic* mean specifying an appropriate value or file name.

Note: The file names that can be specified in each tool are subject to the following limitations.
- File name: up to 32 characters
- Legal characters: a to z, A to Z, 0 to 9, _, .

4.2.1 txt2snd.exe

Function: Converts sound text files into SND files.

Format: `DOS>txt2snd` ***gateoff infile.txt outfile.snd***↵

Parameters:

	gateoff	Gate-off time (0–99) Specify a duration during which the key is turned off as a percentage (%) relative to the length of each note. For example, when you specify 40, sb33 assumes the key is depressed for the first 60% of the note and released for the remaining 40% when creating sound data to play. When you specify 0, the key is assumed to be held down, and neither sustained tones nor sustained tone pitch bends are added.
	infile.txt	Input file name (sound text file)
	outfile.snd	Output file name (SND file)

Example:
```
DOS>txt2snd 50 spring.txt spring.snd
```

Reference: For a description of the contents of the sound text files or for information on creating sound text files, refer to Section 3.1.1, "Creating Sound Text Files". For a description of the contents of SND files, refer to Section 3.1.3, "SND Files".

4.2.2 snd2bin.exe

Function: Converts SND files (text format) into binary files.

Format: `DOS>snd2bin` ***infile.snd outfile.bin***↵

Parameters:

	infile.snd	Input file name (SND file)
	outfile.bin	Output file name (sound binary file)

Example:
```
DOS>snd2bin spring.snd spring.bin
```

Reference: This tool converts each line of an SND file into 3-byte binary data. Note that an ID (0x3312) is located at the beginning of the sound binary file.

4.2.3 midi2snd.exe

Function: Converts standard MIDI files in SMF0 or SMF1 format into SND files.

Format: `DOS>midi2snd` ***stereo bin_flag bend_num per_ch inst_map infile.mid stb_dir etb_dir outfile***↵

Parameters:

stereo	-s = stereo, -m = monaural Specify whether you want to create the sound list file for stereo or monaural reproduction.
bin_flag	-b = binary file output, -s = SND file output Specify whether you want the file to be output in binary or SND file formats. If you choose SND file, a batch file for creating a binary file is also output.
bend_num	Pitch bending (0–12) Set a pitch bending value for pitch bending change events. For the value 0, pitch bending has no effect. Specify a pitch bending value in the range 1 to 12 for up to one octave in semitone increments.
per_ch	Percussion channel number Specify a MIDI channel number in which a percussion instrument is used. In General MIDI, this would normally be channel 10.
inst_map	Instrument map file name Specify the map file name used to convert instrument names. The map file "\sndtool\bin\snd_inst.map" contains a description of standard instruments supported by SOUND33.
infile.mid	Input file name (MIDI file in SMF0 or SMF1 format)
stb_dir	Relative path to the WAVE data directory to be written in the sound list file
etb_dir	Relative path to the envelope data directory to be written in the sound list file
outfile	Output file name (sound list file) Specify the name of the sound list file (.lst) and that of the batch file (.bat) created when you specify -s for *bin_flag*. The names of SND files (.snd) or sound binary files (.bin) output by conversion as musical data are assigned a track number and MIDI channel number. (Example: tr1ch2_1.snd)

Example:
```
DOS>midi2snd -s -b 0 10 inst.map test.mid ..\stb\ ..\etb\ test
```

Note:

- Only MIDI files in SMF0 or SMF1 format can be converted.
- Channel information on musical instruments not supported by SOUND33 are written out to comments in the sound list file. Rewrite the data for another instrument for playback.
- If multiple tempo change events are found in the MIDI data, a warning is generated during conversion, and the first setting is used.
  ```
  Warning: Different tempo setting.
  ```
- If the sound volume of MIDI data to be converted into a single channel changes in the middle of the file, a warning is generated during conversion, and the last setting is used.
  ```
  Warning: Different volume setting.
  ```
- If any MIDI file whose timebase is not 48 (length of the quarter note = 48) is converted, a warning is generated. A timebase of over 48 will exceed the resolution of SOUND33, resulting in errors. Although midi2snd.exe makes adjustments to eliminate cumulative errors, sound may drift slightly when compared to musical reproduction by a MIDI sequencer.
  ```
  Warning: Different time base setting XX.
  ```

4.2.4 snd2pcm.exe

Function: Creates the following files based on information in the sound list file:

<file>.pcm	PCM files used for evaluation on the PC
<file>_inst.bat	Batch files for creating assembly source files of instrument (tone quality) data
<file>_data.bat	Batch files for creating assembly source files of musical data
setbl.c	Structure of instrument (tone quality) data
<file>.c	Structure of musical data

Format: `DOS>snd2pcm` ***infile.lst volume sampling offset outfile***↵

Parameters:

infile.lst Input file name (sound list file)

volume Sound volume adjustment (0–4000, hex)
Adjust the sound volume of all channels relative to x1.0 for 0x100. The sound volume changes in proportion to numeric values, e.g., x0.5 for 0x80 and x1.5 for 0x180.

sampling Sampling frequency (8, 16, 22, or 32)
Specify a sampling frequency for the PCM file to be output.
8 = 8 kHz, 16 = 16 kHz, 22 = 22kHz, 32 = 32 kHz

offset Amount of tone shift (-60 to 60, in semitone increments)

outfile Output file name
Do not specify an extension.

Example:
```
DOS>snd2pcm spling.lst 100 32 0 spling
```

Note:

- To run this tool, you must have the "\stb\xx\" (xx = 08, 16, 22, or 32) and "\etb\" directories at the correct path (the path recorded in the sound list file).
- A warning results if the amplitude of data in the output file leads to an overflow. Since setting the amplitude to eliminate overflow altogether reduces sound levels significantly, set the sound volume slightly higher (more or less overflowing) than indicated in the warning message.
  ```
  Warning: Over flow ! Please change volume about 0x4e below.
  ```
- An error is assumed if the upper limit of the SOUND33 output channels is exceeded. Modify the sound list file to avoid exceeding the upper limit of these channels.
  ```
  Error: Too many internal channel 90. Max channel is 80.
  ```

  The number of channels over which sound can be simultaneously produced on the actual system is limited by the resources available, as shown below:

8 kHz monaural	Tempo 100: 68ch	*1
8 kHz stereo	Tempo 100: 58ch	*1
16 kHz monaural	Tempo 100: 60ch	*1
16 kHz stereo	Tempo 100: 45ch	*1
22 kHz monaural	Tempo 100: 58ch	*1
22 kHz stereo	Tempo 100: 28ch	*1
32 kHz monaural	Tempo 100: 55ch	*1
32 kHz stereo	Tempo 100: 40ch	*2

*1: The internal RAM (8K) only is used in the BSS area.
*2: The Speak buffer is located in external RAM.

  The faster the tempo, the smaller the amount of RAM consumed. Conversely, the amount of RAM consumed increases as the tempo is reduced. When it is to be executed in internal RAM, make sure the music data falls within these limits. Since the number of sounds that can be generated simultaneously is also limited, we recommend evaluating the musical data by playing it on an actual system, such as a DMT.

Reference: For the example batch files and C files output, refer to Section 3.1.8, "Converting Sound Data to Assembly Source Files".

4.2.5 bin2s.exe

Function: Converts binary files into text files in E0C33 assembly source format. Since the results are presented to standard output (stdout), use the DOS redirect function to save them to a file. This tool is actually executed from a batch file for creating assembly source files output by snd2pcm.exe.

Format: DOS>**bin2s** [**-l** ***symbol***] ***infile.bin*** > ***outfile.s***↵

Parameters:

-l ***symbol***	Assembler symbol name definition (optional) Omitting this option sets the input file name as the symbol name.
infile.bin	Input file name (binary file)
outfile.s	Output file name (assembly source file)

Example: 1) Omitting the -l option sets the input file name as the assembler symbol name.

```
DOS>bin2s spring.bin > spring.s
DOS>type spring.s
     .global spring
     .align 2
spring:
     .byte   0x12 0x33 0x18 0x24 0x80 0x18 0x28 0x80
     .byte   0x18 0x28 0x80 0x18 0x28 0x80 0x0c 0x26
     .byte   0x80 0x0c 0x24 0x80 0x48 0x2b 0x80 0x0c
                          :
     .byte   0x00 0x00 0x00
; total 83 bytes data
DOS>
```

2) To use another symbol name that is not the file name, specify using the -l option.

```
DOS>bin2s -l snd01 spring.bin > snd01.s
DOS>type snd01.s
     .global snd01
     .align 2
snd01:
     .byte   0x12 0x33 0x18 0x24 0x80 0x18 0x28 0x80
     .byte   0x18 0x28 0x80 0x18 0x28 0x80 0x0c 0x26
     .byte   0x80 0x0c 0x24 0x80 0x48 0x2b 0x80 0x0c
                          :
     .byte   0x00 0x00 0x00
; total 83 bytes data
DOS>
```

Note: Specification of symbol names is subject to the following limitations:
- Symbol length: 32 characters or less
- Valid characters: a to z, A to Z, 0 to 9, and _

4.2.6 bdmp.exe

Function: Produces dump output in a specified format from the input binary file. Since the results are presented to standard output (stdout), use the DOS redirect function to save to a file.

Format: DOS>**bdmp** ***option infile*** > ***outfile***↵

Parameters:

option Specification of output format (cannot be omitted)
Use the following switches to specify the output format:
-b Output in byte format
-l Output in little-endian short format
-m Output in big-endian short format

infile Input file name (binary file)

outfile Output file name (text file)

Example:

```
DOS>bdmp -b spring.bin
00000000 12 33 0C 30 80 0C FF 50 0C 34 80 0C FF 50 0C 34
00000010 80 0C FF 50 0C 34 80 0C FF 50 06 32 80 06 FF 50
00000020 06 30 80 06 FF 50 24 37 80 24 FF 50 06 37 80 06
00000030 FF 50 06 35 80 06 FF 50 0C 34 80 0C FF 50 0C 34
00000040 80 0C FF 50 0C 34 80 0C FF 50 06 32 80 06 FF 50
00000050 06 30 80 06 FF 50 24 37 80 24 FF 50 06 37 80 06
00000060 FF 50 06 35 80 06 FF 50 0C 34 80 0C FF 50 06 35
00000070 80 06 FF 50 06 37 80 06 FF 50 0C 35 38 0C FF 50
00000080 0C 34 38 0C FF 50 0C 32 38 0C FF 50 0C 2F 38 0C
00000090 FF 50 0C 2B 38 0C FF 50 18 00 00 18 FF 50 00 00
000000A0 00

DOS>bdmp -l spring.bin
00000000 3312 300C 0C80 50FF 340C 0C80 50FF 340C
00000010 0C80 50FF 340C 0C80 50FF 3206 0680 50FF
00000020 3006 0680 50FF 3724 2480 50FF 3706 0680
00000030 50FF 3506 0680 50FF 340C 0C80 50FF 340C
00000040 0C80 50FF 340C 0C80 50FF 3206 0680 50FF
00000050 3006 0680 50FF 3724 2480 50FF 3706 0680
00000060 50FF 3506 0680 50FF 340C 0C80 50FF 3506
00000070 0680 50FF 3706 0680 50FF 350C 0C38 50FF
00000080 340C 0C38 50FF 320C 0C38 50FF 2F0C 0C38
00000090 50FF 2B0C 0C38 50FF 0018 1800 50FF 0000
000000A0

DOS>bdmp -m spring.bin
00000000 1233 0C30 800C FF50 0C34 800C FF50 0C34
00000010 800C FF50 0C34 800C FF50 0632 8006 FF50
00000020 0630 8006 FF50 2437 8024 FF50 0637 8006
00000030 FF50 0635 8006 FF50 0C34 800C FF50 0C34
00000040 800C FF50 0C34 800C FF50 0632 8006 FF50
00000050 0630 8006 FF50 2437 8024 FF50 0637 8006
00000060 FF50 0635 8006 FF50 0C34 800C FF50 0635
00000070 8006 FF50 0637 8006 FF50 0C35 380C FF50
00000080 0C34 380C FF50 0C32 380C FF50 0C2F 380C
00000090 FF50 0C2B 380C FF50 1800 0018 FF50 0000
000000A0
```

4.2.7 pcm2stb.exe

Function: Creates 12 waveforms of data to be registered to the WAVE table from one waveform of PCM data that contains sampled instrument sound. The sound source data created here is basically one-octave PCM data relative to C2 (65 Hz), which is generated by linearly interpolating intermediate values through floating-point calculation. The output files assume the names <input file name>.pcm_01 through <input file name>.pcm_12.

Format: `DOS>pcm2stb` ***in_num out_num infile.pcm***↵

Parameters:

in_num	Number of sample points in the input PCM data
out_num	Number of sample points for one waveform relative to C2 (65 Hz) *out_num* = Sampling frequency (Hz) / 65 (Hz) 8 kHz = 123, 16 kHz = 246, 22 kHz = 339, 32 kHz = 492
infile.pcm	Input file name (PCM file)

Example:
```
DOS>pcm2stb 2026 492 sample.pcm
```

Note:

- Since the high tonal range has fewer harmonic components than the low tonal range, we recommend preparing separate sound sources for each musical instrument for treble and bass parts at almost equal lengths. In this case, make sure the lengths of the high and low tones are approximately the same. For *out_num* for the treble part, specify twice the value of the bass part.
  Example:
  ```
  pcm2stb 2003 984 highdata.pcm   ← High tonal side consists of two waveforms
                                    of data (984 = 32000/65*2)
  pcm2stb 2026 492 lowdata.pcm    ← Low tonal side consists of one waveform
                                    of data (492 = 32000/65)
  ```
- Adjust the original waveform data as shown below before entering into this tool.
  - Using dct_cnv.exe, adjust the extracted waveform until it consists of 1,000 to 2,000 points.
  - Using pcm_norm.exe, normalize the waveform data so that its maximum amplitude is 90%.
  - Extract waveform data so way that it always starts from 0, rising in the positive direction.
  - Extract waveform data so that the data immediately preceding the last point is approximately 0.

4.2.8 stb12.exe

Function: Creates WAVE table binary data with a specified instrument name from one-octave sound source data or single PCM file generated by pcm2stb.exe. Use stbadd12.exe to add sound source data for the treble part (for example) to a previously-created WAVE table.

Format: `DOS>stb12 [-s] infile.pcm outfile.bin↵`

Parameters:

`-s`	Registration of single PCM file (optional) Specify this parameter when registering single PCM data to the WAVE table instead of 12 pieces of scale data.
infile.pcm	Input file name (PCM file) This is the PCM file of 12 pieces of scale data (xxx.pcm_01 to xxx.pcm_12) generated by pcm2stb or the PCM file of less than 3 seconds (for 22 kHz) (when the -s option is specified).
outfile.bin	Output file name (WAVE table binary file)

Example:
```
DOS>stb12 sample.pcm stb_user.bin
```

Note:
- The 12 pieces of one-octave data are registered as octave 1 (C1 and over) data. Unless any data is added later with stbadd12.exe, the entire scale is generated on the basis of this data.
- When loading a single PCM file using the -s option, the data length is limited as follows:
  8 kHz: Within 8 seconds
  16 kHz: Within 4 seconds
  22 kHz: Within 3 seconds
  32 kHz: Within 2 seconds
- To create a complete WAVE table binary file with no data added, specify "stb_<instrument name>.bin" for the output file name and generate it in the WAVE data directory that matches the sampling frequency, "\sndtool\stb\xx\" (xx = 08, 16, 22, or 32). Register the instrument name along with the Echo and Etbadj values in "\sndtool\bin\snd_inst.map".

4.2.9 stbadd12.exe

Function: Additionally registers new 12 pieces of scale data generated by pcm2stb.exe to the WAVE table. You can specify an octave number for the scale data to be registered, so that the registered data covers the musical scale above it.

Format: `DOS>stbadd12 infile.bin octave_num infile.pcm outfile.bin↵`

Parameters:

infile.bin	Input file name (WAVE table binary file)
octave_num	Octave number for the data to be additionally registered (2–6)
infile.pcm	Input file name (PCM file) 12 pieces of scale data generated by pcm2stb.exe (xxx.pcm_01 to xxx.pcm_12)
outfile.bin	Output file name (WAVE table binary file)

Example:
```
DOS>stbadd12 temp.bin 4 high.pcm stb_user.bin
```
Pieces of data high.pcm_01 through high.pcm_12 are registered to the WAVE table temp.bin as scale data starting from C4 to create stb_user.bin (instrument name = user).

Note:
- This tool requires that the WAVE table binary file *infile.bin* be available.
- When executing stbadd12.exe several times on the same WAVE table, always be sure to register data in order of musical scales, beginning with the lowest scale.
- To create a complete WAVE table binary file with no additional data, specify "stb_<instrument name>.bin" for the output file name and generate it in the WAVE data directory corresponding to the sampling frequency, "\sndtool\stb\xx\" (xx = 08, 16, 22, or 32). Register the instrument name along with the Echo and Etbadj values in "\sndtool\bin\snd_inst.map".

4.2.10 etb.exe

Function: Creates envelope data.

Format: `DOS>etb` ***attack cont 1/2down min_val am_freq am_ratio peak key_off_1/2down outfile.bin***↵

Parameters:

attack Attack time (msec)
Specify the duration in which sound rises from key-on to peak (0x80).

cont Peak continuance time (msec)
Specify a duration for which time the peak continues.

1/2down Decay time adjustment (msec)
Specify a duration from the peak to the point at which the signal level attenuates to twice the value of the sustain level (*min_val*).

min_val Sustain level (0–100%)
Specify a signal level that is retained until the key is turned off after attenuating from the peak as a percent value relative to the peak (= 100%).

am_freq Vibrato rate (AM amplitude period, msec)
When creating vibrato, specify the velocity at which the pitch is to waver as an AM modulation period.

am_ratio Vibrato depth (AM modulation amplitude ratio, 0–50%)
When creating vibrato, specify the wavering width of pitch as a percent value relative to the envelope-processed final waveform.

peak Peak adjustment (1.0–1.9)
To express a quickly-rising sharp tone like that of a trumpet, specify a multiple for the peak value.

key_off_1/2down Release time adjustment (msec)
Specify a duration during which the signal level is attenuated to 1/2 of the sustain level after key-off.

outfile.bin Output file name (envelope table binary file)

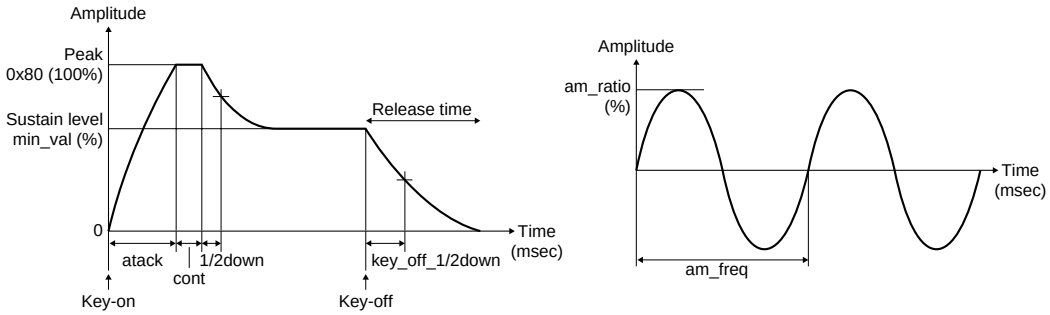


Figure 4.2.1 Envelope and Vibrato

Example:
```
DOS>etb 20 0 100 70 100 10 1 100 etb_user.bin
```

Note: For the output file name, use "etb_<instrument name>.bin" to match it to the WAVE data instrument name, and output the file to the envelope data directory "\sndtool\etb\".

4.2.11 dct_cnv.exe

Function: Converts the input sound file to any sampling rate specified by a parameter.

Format: `DOS>dct_cnv` ***DctFrom DctTo infile.(wav|pcm) outfile.pcm***↵

Parameters:

DctFrom	Number of input data entries to convert
DctTo	Number of corresponding output data entries
infile.wav	Input file name (WAV file)
infile.pcm	Input file name (PCM file)
outfile.pcm	Output file name (PCM file)

Example: For both *DctFrom* and *DctTo,* we recommend specifying integer multiples of the original sampling rate. For example, when down-sampling a 48-kHz WAV file to 16 kHz, specify the parameters 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 higher the values specified for *DctTo* and *DctFrom,* the better the sound quality, but with penalties in processing speed. Processing speed is faster for smaller values, but with lower sound quality. To avoid degrading sound quality, we recommend converting files with five times or more of the original sampling rate.

4.2.12 pcm_norm.exe

Function: Converts the amplitude of PCM file sound data to a specified magnitude. The numeric values specifiable by signed 16 bits range from -32768 (SHORT_MIN) to +32767 (SHORT_MAX). In this program, specify the maximum amplitude of input sound data as a percentage of SHORT_MAX when converting the input sound amplitude.

Format: `DOS>pcm_norm [-r XXX] [-c] input.pcm output.pcm↵`

Parameters:

input.pcm	Input file name (PCM file)
output.pcm	Output file name (PCM file)
`-r` ***XXX***	Coefficient of normalization (optional) Specify the amplitude of 16-bit PCM sound data as a percentage of the maximum amplitude. For *XXX*, specify a positive value in the range 0.0 to 100.0. Omitting this option sets the maximum amplitude of output sound to 90%. Always enter a space between -r and *XXX*.
`-c`	Loading of "amp.rto" file (optional) This option loads the "amp.rto" file from the current directory to adjust the sound amplitude. This option is provided for VOX compression and is not used in SOUND33.

Example:

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

4.3 Sound Bench sb33

Sound Bench sb33 is a Windows GUI tool used to convert sound text files or standard MIDI files into the required file format for playback and evaluation on a PC.

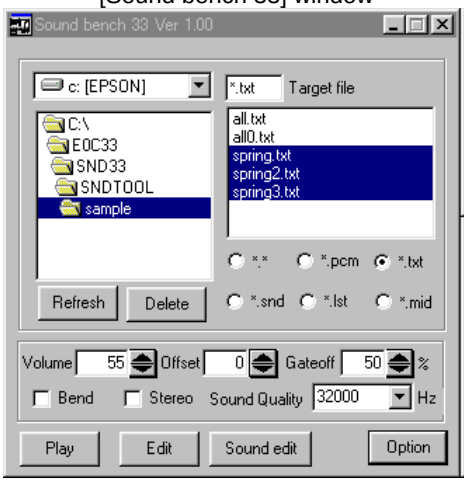
4.3.1 Starting and Exiting

Double-click the "sb33.exe" icon to launch the application.
To exit sb33, click the [Close] button located at the upper right corner of the [Sound bench 33] window.

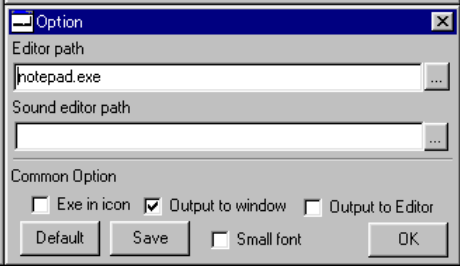
4.3.2 Window Configuration

"sb33.exe" contains the following four windows:

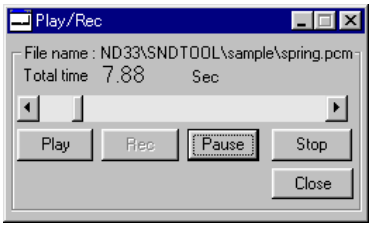
[Sound bench 33] window



[Output] window



[Option] window



[Play/Rec] window

[Sound bench 33] window

This window appears when "sb33.exe" starts. All data conversion operations are performed from this window.

[Output] window

This window is used to display execution commands or execution results (output messages) for the tool invoked to convert data. This window opens automatically when you execute a tool. However, you must select this window from the [Option] window as the destination to which execution results are to be output.

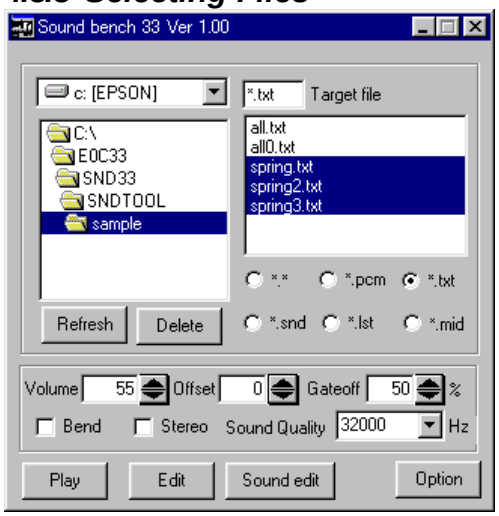
[Option] window

Use this window to select an editor or execution options. The window is opened by clicking the [Option] button on the [Sound bench 33] window.

[Play/Rec] window

From this window, you can play or stop playing the converted sound for reproduction control. This window is displayed after PCM file conversion is complete and reproduction is started, when you press the [Play] button in the [Sound bench 33] window.

4.3.3 Selecting Files



Use the directory list and file list boxes on the [Sound bench 33] window to select files to convert or play. Select a file format to be displayed in the file list box using the radio buttons located below it.

Note: The sound text files and MIDI files to be processed by sb33 must be located in the subdirectory "\sndtool\xxxx\" of the sndtool directory.

[Refresh] button

The contents displayed in the file list box are not automatically updated by the addition or deletion of files, unless addition and deletion are performed with sb33. Click the [Refresh] button to update the list.

[Delete] button

Deletes the files selected in the file list box.

[Edit] button

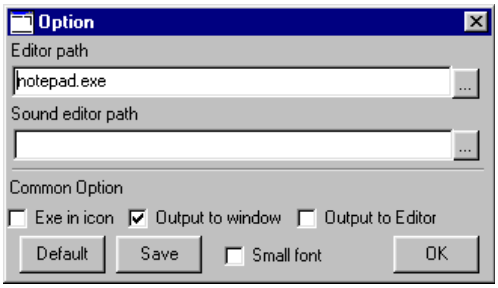
Click the [Edit] button after selecting a text format file from the file list box. This selected file is opened in an editor. This feature allows you to correct sound text, SND, or sound list files directly from sb33. The default editor is the Windows Notepad. You can select another editor in the [Option] window.
Clicking the [Edit] button after selecting multiple files opens only the file at the top of the list.

[Sound edit] button

Click the [Sound edit] button after selecting a PCM file from the file list box to open the selected file in the Sound editor. This feature works only after you set a Sound editor in the [Option] window.
Clicking the [Sound edit] button after selecting multiple files opens only the file at the top of the list.

4.3.4 Selecting Options

Clicking on the [Option] button opens the [Option] window.



[Editor path] text box

In this text box, specify an editor to open by pressing the [Edit] button. Include the absolute path.

[Sound editor path] text box

In this text box, specify the sound editor to open by pressing the [Sound edit] button. Include the absolute path.

[Exe in icon] text box

Clicking this check box starts the tool in its icon state invoked from sb33.

[Output to window] check box

Clicking this check box displays the startup commands or output messages for tools in the [Output] window. In sb33, the startup commands and output messages for each tool are written out to a file entitled "sb33.err" using "ccap.exe" (see the "E0C33 Family C Compiler Package Manual"). The contents of "sb33.err" are displayed in the [Output] window.

[Output to Editor] check box

Clicking this check box displays the startup commands or output messages for tools after a specified editor is launched. As with [Output to window], the editor opens the file "sb33.err".

[Small font] check box

Clicking this check box changes the [Output] window display font to the small font.

[Default] button

Returns all options to their default settings.

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

[Save] button

Saves a set of optional parameter settings, including specification of an editor, to a file (sb33.sav). The saved contents are loaded and set up the next time sb33 starts.

[OK] button

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

4.3.5 Converting Files and Starting Play

Note: The sound text and MIDI files processed by sb33 must be located in the subdirectory "\sndtool\xxxx\" of the sndtool directory.

sb33 begins the necessary conversion and begins playback only when you click the [Play] button after selecting files. The particular tools are activated for file conversion by sb33 depends on the specific file selected. Playback of the generated PCM file is an sb33 function.

Converting and playing sound text files

To select and play sound text files, select files for all the channels you have created. To select multiple files, hold down the [Ctrl] key while selecting.

When you click the [Play] button, sb33 executes the following tools:

1) txt2snd.exe	Converts sound text files into SND files. Executed as many times as the number of files selected.
2) snd2bin.exe	Converts SND files into binary files. Executed as many times as the number of files selected.
3) Internal processing of sb33	Creates sound list files (Note).
4) snd2pcm.exe	Creates PCM files for evaluation on a PC and files to be installed on the actual system.

At this point, sb33 plays the PCM file created in 4).

Note: Sound list files created by sb33

When sb33 creates a sound list file, it assigns it a name based on the name of the selected sound text file (the file at the top of the list if you selected multiple files), adding the extension ".lst". Any existing file with the same name is overwritten.

The sound list is created with the following default settings:

Tempo 100
WAVE data directory ..\stb\
Envelope data directory ..\etb\
Volume As set from [Sound bench 33] window
Echo 6
Etbadj 3
Offset As set from [Sound bench 33] window
Position 50 when [Stereo] is selected from [Sound bench 33] window
-1 when [Stereo] is not selected from [Sound bench 33] window

Example:

```
;Tempo
100
;STB wave table directory
..\stb\
;ETB envelove table directory
..\etb\
;Sound file Instrument  Volume  Echo    Etbadj  Offset   Position
spring.bin piano 70 6  3  0  50
spring2.bin piano 70 6  3  0  50
spring3.bin piano 70 6  3  0  50
```

Converting and playing standard MIDI files

Even when you select multiple standard MIDI files, only the selection at the top of the list is effective.
When you click the [Play] button, sb33 executes the following tools:

1) midi2snd.exe Converts a MIDI file into sound binary and sound list files.
2) snd2pcm.exe Creates a PCM file for evaluation on a PC and files to be installed on the actual system.

At this point, sb33 plays the PCM file created in 2).

Converting and playing SND files

When correcting parameters in a SND file, select the SND file from the file list box and click the [Play] button. Selecting the original sound text file here may result in the corrected SND file being overwritten by txt2snd.exe as it is executed by sb33.
To select and play SND files, select files for all the channels you have created. To select multiple files, hold down the [Ctrl] key while selecting.
When you click the [Play] button, sb33 executes the following tools:

1) snd2bin.exe Converts SND files into binary files.
   Executed as many times as the number of files selected.
2) Internal processing of sb33 Creates a sound list file.
3) snd2pcm.exe Creates a PCM file for evaluation on a PC and files to be installed on the actual system.

At this point, sb33 plays the PCM file created in 3).

Playing from sound list files

After correcting parameters in a sound list file, select the sound list file from the file list box and click the [Play] button. Note that selecting a file in any other format (except PCM files) may result in the corrected sound list file being overwritten. Note also that all of the sound binary files listed in the sound list file must be available before you can play the file.
When you click the [Play] button, sb33 executes the following tool:

1) snd2pcm.exe Creates a PCM file for evaluation on a PC and files to be installed on the actual system.

At this point, sb33 plays the PCM file created in 1).

Playing PCM files

To play a previously created PCM file, select the PCM file from the file list box and click the [Play] button. Reproduction begins immediately, executed by sb33.

Conversion options

The [Sound bench 33] window allows selection of various file conversion options. Specify or select the necessary options before clicking the [Play] button.

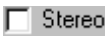
[Volume]
This is the startup option for snd2pcm.exe. Specify a value between 0 and 0x4000. The sound volume set in a SND file is adjusted by multiplying it by (for example): 1.0 for 0x100, 0.5 for 0x80, or 1.5 for 0x180. The default value is 0x70 (multiplied by 0.44) if stereo is selected, or 0x55 (multiplied by 0.33) when monaural is selected.

Note: A warning results if the amplitude of data in the PCM file leads to an overflow. Since setting the amplitude to eliminate overflow altogether reduces sound levels significantly, set the sound volume slightly higher (more or less overflowing) than indicated in the warning message.

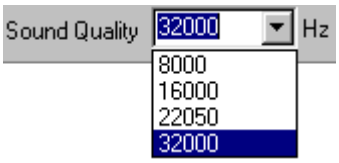
[Offset]
This is the startup option for snd2pcm.exe. Musical intervals are shifted in semitone units. Specify a value in the range -60 to +60 (±5 octaves). Note that the playable range of steps from C1/do1 to B6/si6 remains unchanged, even after the musical interval is shifted. Notes exceeding this range as a result of a shift are adjusted to C1/do1 or B6/si6.

[Gateoff]
This is the startup option for txt2snd.exe. It specifies a time during which the key is turned off as a percent value relative to the length of each note. Specify a value in the range 0 to 99%. For example, if you specify 40%, sb33 assumes that the key is depressed for the first 60% of the note and released for the remaining 40% when creating the SND file.

[Bend]
This is the startup option for midi2snd.exe, specifying enabling or disabling of pitch bending. Clicking the check box enables pitch bending, so that the amount of pitch bending is set to 1 (equivalent to a semitone). To specify any degree of pitch bending from 2 to 12, run midi2snd.exe from the DOS prompt.

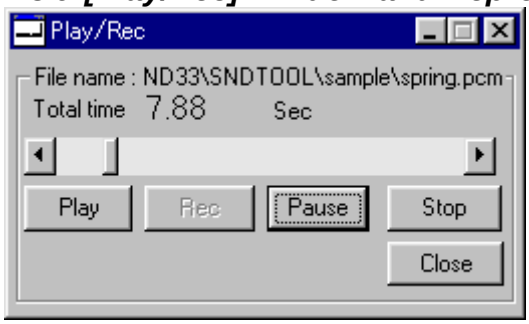


[Stereo]
Specifies whether the sound list file should be created for reproduction in stereo or monaural. Clicking the check box selects stereo, so that the sound list file is created by setting Position to 50 (center). To change the position, correct the sound list file directly. Note that selecting stereo sets [Volume] to 70. Leaving this option unchecked selects monaural reproduction, in which case the sound list file is created by setting Position to -1. [Volume] is set to 55.



[Sound Quality]
This is the startup option for snd2pcm.exe. It sets the sound quality (sampling frequency).

4.3.6 [Play/Rec] Window and Reproduction Control



The [Play/Rec] window is displayed when you play back a selected PCM file following file conversion by clicking the [Play] button on the [Sound bench 33] window. The window closes automatically when reproduction finishes.

Press the [Pause] button to interrupt playback. The [Play/Rec] window remains open. When you click the [Play] button, reproduction resumes from the point at which it was interrupted. You can shift the resumption point with the scroll bar.

Clicking the [Stop] button cancels playback and returns the resumption point to the beginning; the [Play/Rec] window also remains open. Click the [Play] button to resume playback from the beginning of the file.

The [Close] button remains enabled during playback and when playback is interrupted. It closes the [Play/Rec] window.

The [Rec] button has no effect in sb33.

4.3.7 Operation after Evaluation Finishes

Batch files for creating assembly source files for music and instruments are generated by "snd2pcm.exe", which you can execute by clicking the [Play] button. If there is no need to reexecute tools individually — to specify options separately, for example — you can execute the generated batch files immediately to create assembly source files.

5 SOUND33 Library Reference

This section gives some precautions for using SOUND33 library functions and explains each function in detail.

5.1 Outline of the SOUND33 Library

Functional outline

The SOUND33 library consists of a set of functions for sound output in srf33 library format, and is used by linking it to the target program. The programmed piece of music is played in real time by calling the necessary functions from the target program.

This package contains the top-level functions created as C source files, all or part of which may be copied for use within the target program.

These sets of functions allow easy implementation of sound output features in your system. Figure 5.1.1 shows the structure of an application program for sound output.

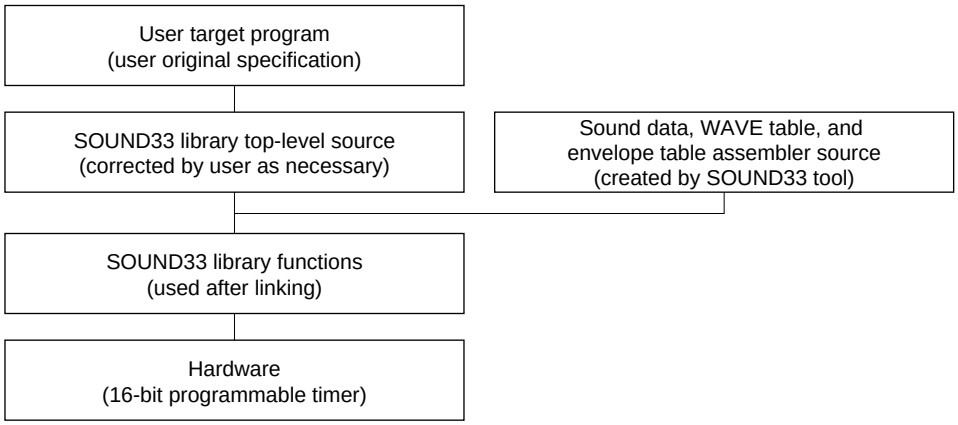


Figure 5.1.1 Program Structure

Configuration of the SOUND33 library

All SOUND33 library and related files are located in the "sndlib" folder (directory). The folder contents are given below:

sndlib SOUND33 library-related
- readme.txt — SOUND33 library supplementary explanation, etc. (in English)
- readmeja.txt — SOUND33 library supplementary explanation, etc. (in Japanese)

lib ..... SOUND33 library directory
- snd.lib — Sound engine library
- sndcpy.o, sndcpy2.o, snd2.o — Objects for sound engines that require high-speed execution. Copy to internal RAM for use.
- spk208.lib, spkintr1.o, spkintr2.o, spkintr3.o, spkintr4.o — Speak library shared with VOX33

include ..... SOUND33 library function include file directory
- snd.h — Library include file
- sndcomm.h — slutil.c include file
- speak.h — sl208.lib include file

src ..... Source directory
- sndtop.c — Top-level library functions
- slutil.c — PWM output final data creation routine
- sndbuf.c — SPEAK/LISTEN buffer setup file
- slutil2.c — PWM output final data creation tool (stereo)

hardsrc ..... Hardware-dependent source directory
- Spk208.s — Spk208.o source (for E0C33208)
- Spk208PW.s — Spk208PW.o source (for E0C33208)
- slcomm.def
- slintr.def
- SpkIntr1.s — 15-bit monaural interrupt functions
- SpkIntr2.s — 15-bit stereo interrupt functions
- SpkIntr3.s — 10-bit monaural interrupt functions
- SpkIntr4.s — 9-bit monaural interrupt functions

demoX ..... Sample program directory

(For details on the configuration of sample programs, refer to "readme.txt "or "readmeja.txt" in "sndlib".)

5.2 Hardware Requirements

Hardware resources used by the library

The SOUND33 library uses the following internal hardware resources of the chip. These hardware resources cannot be used in the user target program.

For 9-bit monaural output

- 16-bit timers 5 and 1 and all associated control registers
- 16-bit timer 1 output port
- 16-bit timer 5 compare B interrupt
  Make sure the SpkIntr4() function address is stored at the vector address for the 16-bit timer 5 compare B interrupt.
  Example: `.word  SpkIntr4  ; Vector No. 50 (16-bit timer #5 compare B)`

For 10-bit monaural output

- 16-bit timers 5 and 1 and all associated control registers
- 16-bit timer 1 output port
- 16-bit timer 5 compare B interrupt
  Make sure the SpkIntr3() function address is stored at the vector address for the 16-bit timer 5 compare B interrupt.
  Example: `.word  SpkIntr3  ; Vector No. 50 (16-bit timer #5 compare B)`

For 15-bit monaural output

- 16-bit timers 5, 1, and 2 and all associated control registers
- 16-bit timers 1 and 2 output ports
- 16-bit timer 5 compare B interrupt
  Make sure the SpkIntr1() function address is stored at the vector address for the 16-bit timer 5 compare B interrupt.
  Example: `.word  SpkIntr1  ; Vector No. 50 (16-bit timer #5 compare B)`

For 15-bit stereo output

- 16-bit timers 5, 1, 2, 3, and 4 and all associated control registers
- 16-bit timers 1, 2, 3, and 4 output ports
- 16-bit timer 5 compare B interrupt
  Make sure the SpkIntr2() function address is stored at the vector address for the 16-bit timer 5 compare B interrupt.
  Example: `.word  SpkIntr2  ; Vector No. 50 (16-bit timer #5 compare B)`

Note: To use other timer channels, you must correct the source files in the "\sndlib\hardsrc\" directory. However, libraries recompiled after correcting these source files fall outside the scope of the product warranty.

Operating clock

This library assumes that the E0C332xx high-speed (OSC3) clock frequency is 20 MHz (typ.) and that PLL is in x2 mode (with the CPU operating at 40 MHz).

Memory

The SOUND33 library uses internal RAM to run fast routines, as well as for the stack and for the BSS section used by the SOUND33 library. For more information on various memory requirements, refer to Section 5.6, "Memory Size and Number of Simultaneously Reproduced Sound Channels".

5.3 Top-level Functions

The top-level functions are provided with the C source file (sndtop.c) to facilitate implementation of sound output features, which are implemented using SOUND33 library functions. Table 5.3.1 lists the functions in sndtop.c.

Table 5.3.1 List of Top-level Functions

Function name	Description
unsigned char ***sndSpeak( )**	Processing to begin playing sound data
static unsigned char ***TopSpeakStart( )**	Starts sound output
int **sndTopDecode( )**	Callback function during playback
int **sndSpeakStart( )**	Begins playback
int **sndSpeakStop( )**	Stops playback
void **sndCodecpy( )**	Code section copy function for fast operation

To use "sndtop.c", correct for the maximum number of channels defined as necessary and copy the entire file to the user program source, or link the source file directly as is.
In addition to "sndtop.c", the "src\" directory contains "sndbuf.c", "slutil.c" (for monaural output), and "slutil2.c" (for stereo output). These source files contain functions other than those listed above, which do not need to be called directly from the user program. Always make sure to link them (for "slutil.c" and "slutil2.c," link one or the other).

To use the source file by linking it directly, make sure the header files in the "include\" directory are included in the user program.

5.3.1 Compile Options

Compiling the source file of top-level functions allows specification of the following compile options. Define the respective names during compilation as necessary (by using the -D option of gcc33).

CLOCK40

Specify this option when operating the E0C332xx at 40 MHz (PLL in x2 mode).

SPK_10

Specify this option when producing 10-bit monaural output. This option cannot be set simultaneously with STEREO.

SPK_9

Specify this option when producing 9-bit monaural output. For 20 MHz operation, specify this option instead of SPK_10. This option cannot be set simultaneously with STEREO.

STEREO

Specify this option when producing 15-bit stereo output. This option cannot be used simultaneously with SPK_10, SPK_9, or MONO.

MONO

Specify this option when producing monaural output. Always specify this option when not using STEREO.

SAMPLING8K

Specify this option when 8-kHz sampled data is used.

SAMPLING16K

Specify this option when 16-kHz sampled data is used.

SAMPLING22K

Specify this option when 22-kHz sampled data is used.

SAMPLING32K

Specify this option when 32-kHz sampled data is used.

PWM_ADJUST

This option corrects the PWM output to improve output characteristics. (Setting the option degrades performance, however.)

INIT

Define this option when playing music asynchronously between channels and prepare an interrupt service routine (see demo3\int.s).

DEBUG

This option inserts code required to check whether output is produced in real time (whether processing is executed in time). Because this check routine increases overhead, we recommend removing this option when compiling the final code for the product.

5.3.2 Changing the Maximum Number of Channels and Permitted Tempo

Change the maximum number of channels and the permitted tempo using "sndtop.c" and "snd.h", respectively.

Changing the maximum number of channels (sndlib\src\sndtop.c)

```
      :
#ifdef MSVC
#define MAX_CHANNELS   84  // max sound channels
#else
#define MAX_CHANNELS   30  // max sound channels    ← Maximum number of channels
#endif                                                 in the actual system
      :
```

By default, the maximum number of channels is 30. To use more channels, you must change the default setting. The number of channels in cases where tempo = 100 is shown below. To use a tempo below 100, you must increase the buffer size of the internal RAM, which results in the maximum number of channels being reduced accordingly.

8 kHz monaural	Tempo 100	68ch
8 kHz stereo	Tempo 100	58ch
16 kHz monaural	Tempo 100	60ch
16 kHz stereo	Tempo 100	45ch
22 kHz monaural	Tempo 100	58ch
22 kHz stereo	Tempo 100	28ch
32 kHz monaural	Tempo 100	55ch
32 kHz stereo	Tempo 100	40ch *

* When the Speak buffer is located in external RAM
  Otherwise, only the internal RAM (8K) is used for the BSS area.

Changing the permitted tempo (sndlib\include\snd.h)

By default, the buffer size is defined to allow you to set tempos of 100 or greater. To select a tempo below 100, modify the value defined for PACKET_SIZE in "snd.h".

```
/* sampling rate */

#ifdef SAMPLING8K            (Settings for sampling rate = 8 kHz)
    #define SND_SAMPLING    8000
    #define PACKET_SIZE     (100>>SPEAK_BUF_MUL)
                            // 1 packet is 100 data tempo is 100
#endif

#ifdef SAMPLING16K           (Settings for sampling rate = 16 kHz)
    #define SND_SAMPLING    16000
    #define PACKET_SIZE     (200>>SPEAK_BUF_MUL)
                            // 1 packet is 200 data tempo is 100
#endif

#ifdef SAMPLING22K           (Settings for sampling rate = 22 kHz)
    #define SND_SAMPLING    22050
    #define PACKET_SIZE     (276>>SPEAK_BUF_MUL)
                            // 1 packet is 276 data tempo is 100
#endif

#ifdef SAMPLING32K           (Settings for sampling rate = 32 kHz)
    #define SND_SAMPLING    32000
    #define PACKET_SIZE     (400>>SPEAK_BUF_MUL)
                            // 1 packet is 400 data tempo is 100
#endif
```

You can obtain the relationship of tempo to PACKET_SIZE by the following equation:

PACKET_SIZE = <sampling rate (Hz)> × 60 / <tempo> / 48

Example: When sampling rate = 8 kHz and tempo = 100
PACKET_SIZE = 8000 × 60 / 100 / 48 = 100

The maximum number of channels depends on the value for PACKET_SIZE set here. Since the channel data, decode buffer, and Speak buffer in SOUND33 are located in internal RAM, the maximum number of channels decreases as the buffer size increases.
Buffer sizes are listed below:
• Decode buffer: PACKET_SIZE in bytes (PACKET_SIZE × 2 for stereo)
• Speak buffe: PACKET_SIZE in bytes (PACKET_SIZE × 2 for stereo)
• Channel data: Approx. 100 bytes per channel

The following macros used to calculate the necessary buffer size for a specified tempo are defined in "snd.h":

`TEMPO8_LEN(a)`	For 8 kHz sampling
`TEMPO16_LEN(a)`	For 16 kHz sampling
`TEMPO22_LEN(a)`	For 22 kHz sampling
`TEMPO32_LEN(a)`	For 32 kHz sampling

For "a", specify tempo.

5.3.3 Error Codes Returned by Functions

The error codes for SOUND33 library functions are defined in "snd.h" as constants, as shown below. For functions that return error codes, use these constants to check whether function execution has finished without problems.

SND_OK	0	Terminated normally
SND_FINISH	1	Finished playing
SND_ERROR	2	Error occurred in SOUND33 library
SND_STB_ERROR	3	WAVE table data is invalid
SND_ETB_ERROR	4	Envelope table data is invalid
SND_DATA_ERROR	5	Sound data is invalid
SND_ADJ_ERROR	6	Specified value of Etbadj is invalid (Effective values = 0 to 4)
SND_TEMPO_ERROR	7	Specified value of Tempo is invalid (Effective values = 30 to 300)
SND_VOL_ERROR	8	Specified value of Volume is invalid (Effective values = 0 to 0x4000)
SND_OFFSET_ERROR	9	Specified value of Offset is invalid (Effective values = -60 to 60)

5.3.4 SOUND33 Data Structure

The musical pieces and instruments for playback and the playback tempos are specified using the structures provided for each.

Play structure (struct SNDDATA1)

Example: Excerpt from sndlib\demo2\spring.c

```
const struct SNDDATA1 sd1_spring[] = {
    100,    // sTempo                                ← Tempo
    0x90,   // sVolume for all channels              ← Volume
    0,      // This flag is 0
    0,      // tone offset for all channels          ← Offset
    3       // total channel numbers in SNDDATA2     ← Number of channels
};
```

This structure is written in "xxxx.c", which is output by "snd2pcm.exe", with one created for each piece of music. The values of the structure members are applied to the entire piece of music.
Stored in the number of channels member is the value written in the sound list file loaded as input to "snd2pcm.exe". Modify the value stored here to add channels for asynchronous playback.

Channel structure (struct SNDDATA2)

Example: Excerpt from sndlib\demo2\spring.c

```
const struct SNDDATA2 sd2_spring[] = {
{
    0x11,           ← ID = 0x11
    0,              ← Offset for this channel
    0x90,           ← Volume for this channel
    &spring[0],     ← Pointer to sound data
    &setbl_piano,   ← Pointer to instrument data
    6,              ← Echo delay time (in units of 1/6 of thirty-second note)
    1,              ← Asynchronous play flag (1: Synchronous play; 0: Asynchronous play)
    -1              ← Stereo play position (0 ← 50 → 100, -1 for monaural play)
},
                    :
{
    0x11,           // ID = 0x11
    0,              // tone offset for this channel
    0x90,           // sVolume for this channel
    &spring3[0],    // sound data pointer
    &setbl_piano,   // instruments pointer
    6,              // echo delay for this channel
    1,              // play flag for this channel
    -1              // position for this channel
}
};
```

This structure is written in "xxxx.c", which is output by "snd2pcm.exe", with one created for each piece of music. Information on all channels written in the sound list file are set in this structure. The values of structure members are applied only to the channel in question.
To add channels for asynchronous play, insert information on the additional channels at the end of the structure as described above, setting the asynchronous play flag to 0.

Instrument structure (struct SETBL)

Example: Excerpt from sndlib\demo2\setbl.c

```
const struct SETBL setbl_piano = {
    &stb_piano[0],   ← Pointer to WAVE table data
    &etb_piano[0],   ← Pointer to envelope table data
    3                ← Etbadj (0–4)
};
```

This structure is written in "setbl.c", which is output by "snd2pcm.exe", with one created for each instrument. If you added instrument data to link, you must also write additional information using this structure.

5.3.5 sndSpeak( )

Function: Processing to begin playing sound data

Format:
```
unsigned char *sndSpeak(struct SNDDATA1 *sd1, struct SNDDATA2 *sd2);
```

Parameters:
```
struct SNDDATA1 *sd1  Play structure
struct SNDDATA2 *sd2  Channel structure
```

Return value: 0.....Terminated normally
Not 0...Error (see Section 5.3.3)

Description: Initializes sound processing and calls a playback routine.

5.3.6 TopSpeakStart( )

Function: Starts sound output

Format:
```
static unsigned char *TopSpeakStart();
```

Parameters: None

Return value: 0.....Terminated normally
Not 0...Error (see Section 5.3.3)

Description: Sets sndToDecode() in the callback function and calls the Speak function to output sound.

5.3.7 sndTopDecode( )

Function: Callback function during playback

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

Parameters:
```
unsigned char *SpkParams  Pointer to Speak parameter
short         *Buffer     Pointer to decode buffer
int           Length      Decode data length
```

Return value: 0.....Error
Not 0...Actual decoded data length

Description: When the Speak data queue has free space available, this function is called back from the Speak function. Based on information on each channel, it copies playback data to the buffer.

5.3.8 sndSpeakStart( )

Function: Begins playback

Format:
```
int sndSpeakStart(int iChannel, struct SNDDATA2 *sd2);
```

Parameters:
```
int    iChannel       Channel number in the structure SNDDATA2
struct SNDDATA2 *sd2  Channel structure
```

Return value: 0.....Terminated normally

Description: Sets the play flag to 1 to begin playback. If the beginning channel is an echoed channel, it also begins playing the next internal channel for delayed play.

5.3.9 sndSpeakStop( )

Function: Stops playback

Format: `int sndSpeakStop(int iChannel, struct SNDDATA2 *sd2);`

Parameters:

`int`	`iChannel`	Channel number in the structure SNDDATA2
`struct SNDDATA2`	`*sd2`	Channel structure

Return value: 0.....Terminated normally

Description: Sets the play flag to 0 to stop playback. If the stop channel is an echoed channel, it also stops playing the next internal channel for delayed play.

5.3.10 sndCodecpy( )

Function: Code section copy function for fast operation

Format: `void voxCodecpy(int *dst, int *src, int *size);`

Parameters:

`int`	`*dst`	Transfer destination address (internal RAM)
`int`	`*src`	Source address of transfer (external ROM)
`int`	`*size`	Transfer code size (bytes)

Return value: None

Description: Transfers code from the external ROM to the internal RAM. The following object codes must be transferred to the internal RAM before they can be executed:

1. sndcpy.o (for monaural) or sndcpy2.o (for stereo)
2. snd2.o (for stereo)
3. spkintr1.o (for 15-bit monaural), spkintr2.o (for 15-bit stereo), spkintr3.o (for 10-bit monaural), or spkintr4.o (for 9-bit monaural)
4. slutil.o (for monaural) or slutil2.o (for stereo)

Example:
```
sndCodecpy(&__START_CACHE1, &__START_sndcpy2_code,
&__SIZEOF_sndcpy2_code);
```
Copies the code for "sndcpy2.o" to the position of &_START_CACHE1 in the internal RAM. In this case, the following description is required in the linker command file.
```
-section CASHE1
-ucode CACHE1 {(pass)\sndcpy2.o}
```

5.4 SOUND33 Library Functions

The SOUND33 libraries "snd.lib", "sndcpy.o/sndcpy2.o", and "snd2.o" contain the functions needed to process sound data, while the libraries "spk208.lib", "slutil.c/slutil2.c", and "spkintrX.o (X = 1–4)" contain the functions required for PWM output. You can implement sound output features simply by linking these functions to the user program. To decode and reproduce musical pieces in real time, note that parts of the objects must be mapped to the internal memory. For more information, refer to Section 5.5, "Techniques for Speeding Up Processing".
Table 5.4.1 below lists the library functions.

Table 5.4.1 List of Library Functions

Classification	Function name	Description
Sound data processing	sndInit( )	Initializes channel parameters
	sndSetTempo( )	Sets tempo
	sndGetData( )	Data copy 1 to buffer
	sndCpyData( )	Data copy 2 to buffer (for monaural)
	sndCpyData2( )	Data copy 2 to buffer (for stereo)
	sndCpyDataN( )	Data copy 3 to buffer (for monaural)
	sndCpyData2N( )	Data copy 3 to buffer (for stereo)
	sndZeroFill( )	Initializes buffer (for monaural)
	sndZeroFill2( )	Initializes buffer (for stereo)
	sndChReset( )	Resets WAVE data output position
Output data conversion	setSpeakVolume( )	Sets volume
	slPcm2Spk( )	Converts output data (for monaural)
	slPcm2SpkLR( )	Converts output data (for stereo)
Output (Speak)	SpkSoftening( )	Soft start volume
	SpkSampleRate( )	Changes sampling rate
	SPK_SAMPLING( )	Gets 16-bit timer reload value (macro)
	SpkInit( )	Initializes internal library variables
	SpkOpen( )	Opens output channel
	SpkClose( )	Closes output channel
	SpkStart( )	Starts output
	SpkHalt( )	Stops output
	SpkAppend( )	Enqueues output data
	SpkRoom( )	Gets number of remaining queue entries
	SpkQueue( )	Gets number of entries waiting for output
	SpkIsRunning( )	Checks output status
	SpkOnDone( )	Enters callback function during playback
	SpkOnEmpty( )	Enters callback function when playback finishes
	SpkOnNotInTime( )	Enters callback function for playback not in time
	slGetVersion( )	Gets version information
Interrupt processing	SpkIntr1( )	Processes interrupt (for 15-bit monaural)
	SpkIntr2( )	Processes interrupt (for 15-bit stereo)
	SpkIntr3( )	Processes interrupt (for 10-bit monaural)
	SpkIntr4( )	Processes interrupt (for 9-bit monaural)

The following pages give the specification of each function. For usage examples, refer to the sources of top-level functions.

5.4.1 Sound Data Processing Functions

The functions described below are defined in "snd.lib", "sndcpy.o/sndcpy2.o", and "snd2.o".

sndInit( )

Function: Initializes channel parameters

Format:
```
int sndInit(char *vpara, char *stable, char *etable, char *sdata,
int volume, int offset, int tempo, int etbadj);
```

Parameters:

`char *vpara`	Pointer to sound parameter
`char *stable`	Pointer to WAVE table
`char *etable`	Pointer to envelope table
`char *sdata`	Pointer to sound data
`int volume`	Initial volume
`int offset`	Tone offset (-60 to +60)
`int tempo`	Tempo (30 to 300)
`int etbadj`	Correction value for high tonal envelope

Return value: 0.....Terminated normally
Not 0...Error (see Section 5.3.3)

Description: Initializes sound parameters on each channel.

sndSetTempo( )

Function: Sets tempo

Format:
```
int sndSetTempo(int iTempoLen, int i16);
```

Parameters:

`int iTempoLen`	Tempo (30 to 300)
`int i16`	Sampling frequency (0 = 8 kHz, 1 = 16 kHz, 2 = 22 kHz, 3 = 32 kHz)

Return value: 0.....Terminated normally
Not 0...Error (see Section 5.3.3)

Description: Calculates the unit length of PCM data corresponding to the specified tempo and sampling frequency.

sndGetData( )

Function: Data copy 1 to buffer

Format:
```
int sndGetData(char *vpara, long *sBuf, char cPosition);
```

Parameters:

`char *vpara`	Pointer to sound parameter
`long *sBuf`	Buffer pointer
`char cPosition`	Stereo play position (1 to 100, -1 for monaural)

Return value: 0.....Error
Not 0...Decoded length

Description: Called from the sndTopDecode() function, this function copies data to the buffer according to the values of sound parameters.

sndCpyData( ), sndCpyData2( )

Function: sndCpyData( ) Data copy 2 to buffer (for monaural)
sndCpyData2( ) Data copy 2 to buffer (for stereo)

Format:
```
int sndCpyData(struct CpyPara *para, long *sBuf, short sLen);
int sndCpyData2(struct CpyPara *para, long *sBuf, short sLen);
```

Parameters:

`struct CpyPara *para`	Copy parameter pointer in sound parameter
`long *sBuf`	Buffer pointer
`short sLen`	Decode length

Return value: 0.....Error
Not 0...Decoded length

Description: Called from the sndGetData() function, this function passes copy parameter values and buffer addresses to the function actually copying data to the buffer. When volume = 0, it returns without performing an operation.

sndCpyDataN( ), sndCpyData2N( )

Function: sndCpyDataN( ) Data copy 3 to buffer (for monaural)
sndCpyData2N( ) Data copy 3 to buffer (for stereo)

Format:
```
int sndCpyDataN(struct CpyPara *para, long *sBuf, short sLen);
int sndCpyData2N(struct CpyPara *para, long *sBuf, short sLen);
```

Parameters:

`struct CpyPara *para`	Copy parameter pointer in sound parameter
`long *sBuf`	Buffer pointer
`short sLen`	Decode length

Return value: 0.....Error
Not 0...Decoded length

Description: Called from the sndCpyData()/sndCpyData2() functions, this function copies data to the buffer according to the value of the copy parameters.

sndZeroFill( ), sndZeroFill2( )

Function: sndZeroFill( ) Initializes buffer (for monaural)
sndZeroFill2( ) Initializes buffer (for stereo)

Format:
```
void sndZeroFill(long *dst, int len);
void sndZeroFill2(long *dst, int len);
```

Parameters:

`long *dst`	Decode buffer pointer
`int len`	Decode buffer size

Return value: None

Description: Initializes the buffer before decoding sound data.

sndChReset( )

Function: Resets WAVE data output position

Format:
```
void sndChReset(char *vpara, char *sdata);
```

Parameters:

`char *vpara`	Pointer to copy data structure
`char *sdata`	Pointer to sound data structure

Return value: None

Description: Resets WAVE data output position during playback.

5.4.2 Output Data Conversion Functions

The functions described below are defined in "slutil.c" for monaural and "slutil2.c" for stereo. These functions create the data to be sent to the output device. At this time, they convert signed 32-bit PCM data into unsigned 16-bit PCM data by clipping excursions. Specifying PWM_ADJUST as a compile option includes a routine for correcting PWM output values in the program.

setSpeakVolume( )

Function: Sets output volume

Format: `void setSpeakVolume(unsigned short spkv);`

Parameter: `unsigned short spkv` Specified value for volume

Return value: None

Description: Sets sound volume. The parameter is a relative value referenced to 0x100 (x1). The value specified here is multiplied by the internal sound data value to determine the sound volume. Overflows are rounded off. Very fine settings are possible, since the value can be specified in increments of 1.

```
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 (for monaural)

Format: `void slPcm2Spk(long *Src, short *Dst, int Length, Slparam *slParam);`

Parameters:

`long`	`*Src`	Pointer to source data array
`short`	`*Dst`	Pointer to array to which to write
`int`	`Length`	Number of data entries to convert (short)
`Slparam`	`*slParam`	Conversion parameter

Return value: None

Description: For PCM data, this function applies offset, volume, and clipping processing according to the parameters defined by slParam and setSpeakVolume. Src and Dst must be located in separate arrays. Be sure to set the offset and other parameters necessary to clip signed 32-bit PCM data to unsigned 16-bit PCM data, as shown below:

```
slParam->offset = 0x8000;       ← Adds offset 0x8000
slParam->shift  = 0
slParam->limit  = 0xffff;       ← Clips excursions above the upper-limit 0xffff and below the
                                  lower-limit 0x0.
```

slPcm2SpkLR( )

Function: Converts output data (for stereo)

Format:
```
void slPcm2Spk(long *Src, short *Dst, int Length, Slparam *slParam);
```

Parameters:

`long`	`*Src`	Pointer to source data array
`short`	`*Dst`	Pointer to array to which to write
`int`	`Length`	Number of data entries to convert (short)
`Slparam`	`*slParam`	Conversion parameter

Return value: None

Description: For PCM data, this function applies offset, volume, and clipping processing according to the parameters defined by slParam and setSpeakVolume. Src and Dst must be located in separate arrays. This function is dedicated for stereo output. For Src, data must be input alternately for right and left. Be sure to set the offset and other parameters necessary to clip signed 32-bit PCM data to unsigned 16-bit PCM data, as shown below:

```
slParam->offset = 0x8000;      ← Adds offset 0x8000
slParam->shift  = 0
slParam->limit  = 0xffff;      ← Clips excursions above the upper-limit 0xffff and below the
                                 lower-limit 0x0.
```

5.4.3 Output (Speak) Functions

The functions described below are defined in "spk208.lib". These functions control the PWM output produced by 16-bit timers.

SpkSoftening( )

Function: Output soft start volume

Format: `void SpkSoftening(unsigned char SPK_SOFTENING);`

Parameters: `unsigned char SPK_SOFTENING` Output ON/OFF delay time

Return value: None

Description: This function is used to reduce the switching noise that occurs at the start and end of reproduction output. Always set this function before calling SpkStart(). The output ON delay time is determined by the following equation:

1/(Sampling rate [Hz] × 2) × SPK_SOFTENING × CENTER_DATA [msec]

The value for CENTER_DATA is as follows:
0x200 for the 10-bit case; 0x100 for 15-bit and 9-bit cases
Normally, set SPK_SOFTENING for approximately 50 msec of delay time. Check for switching noise on the actual system before determining the delay time.

SpkSampleRate( )

Function: Changes sampling rate

Format:
```
void SpkSampleRate(unsigned char *SpkParams, void *Buffer,
int ReloadValue);
```

Parameters:

`unsigned char *SpkParams`		Pointer to SpkParams (return value of SpkOpen)
`void`	`*Buffer`	Pointer to output data
`int`	`ReloadValue`	16-bit timer set value

Return value: None

Description: Changes the sampling rate on the channel specified by spkParams according to ReloadValue from the point at which sb33 outputs data beginning with Buffer. For ReloadValue, specify the value obtained by SPK_SAMPLING macro.
Use this function when you want to dynamically change the sampling rate after calling SpkStart(). The parameter Buffer must be the buffer specified by SpkAppend(). Use this function immediately before SpkAppend().
Normally, use SpkOpen() to specify the sampling rate.

SPK_SAMPLING( )

Function: Gets 16-bit timer reload value (macro)

Format: `SPK_SAMPLING(CpuClock, SamplingRate)`

Parameters:

`CpuClock`	CPU clock frequency
`SamplingRate`	Sampling rate

Return value: 16-bit timer reload value

Description: This macro is used to acquire the reload value for the 16-bit timer from the specified CPU clock frequency and sampling rate.

SpkInit( )

Function: Initializes internal library variables

Format: `void SpkInit(void);`

Parameters: None

Return value: None

Description: Clears the internal variables used by the library to 0.

SpkOpen( )

Function: Opens output channel

Format: `unsigned char *SpkOpen(int Mode, int ReloadValue);`

Parameters:

`int Mode`	Output mode
`int ReloadValue`	16-bit timer setting

Return value: 0.....Error
Not 0...Pointer to SpkParams corresponding to the output channel opened

Description: Opens an output channel according to output mode at a specified sampling rate. SpkParams returned by this function is used as a parameter for other output (Spk) functions. For Mode, specify one of the constants listed below. Note that the vectors to be entered for the timer 5 compare B interrupt and objects that include interrupt functions vary, depending on the output mode.

Constant	Output mode	Vector value/object
SPK_15_MONO	15-bit monaural (for 20-MHz or 40-MHz operation)	SpkIntr1( )/spkintr1.o
SPK_15_STEREO	15-bit stereo (for 20-MHz or 40-MHz operation)	SpkIntr2( )/spkintr2.o
SPK_10_MONO	10-bit monaural (for 40-MHz operation)*	SpkIntr3( )/spkintr3.o
SPK_9_MONO	9-bit monaural (for 20-MHz or 40-MHz operation)	SpkIntr4( )/spkintr4.o

* We do not recommend 10-bit monaural output for 20-MHz operation, due to the harsh sound quality caused by PWM noise.

For ReloadValue, specify the value obtained 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 unimplemented channel is specified
- When the reload value exceeds the 16-bit range

Always call SpkSoftening() and SpkInit() before using this function.

SpkClose( )

Function: Closes output channel

Format: `int SpkClose(unsigned char *SpkParams);`

Parameter: `unsigned char *SpkParams` Pointer to SpkParams (return value of SpkOpen)

Return value: 0.....Error
Not 0...Terminated normally

Description: Closes a specified output channel. If the specified channel is not open, the function returns 0.

SpkStart( )

Function: Starts sound output

Format:
```
int SpkStart(unsigned char *SpkParams);
```

Parameter:

`unsigned char *SpkParams`	Pointer to SpkParams (return value of SpkOpen)

Return value: 0.....Error
Not 0...Terminated normally

Description: Starts sound output operation on the specified channel. If the specified channel is not open, the function returns 0.

SpkHalt( )

Function: Stops sound output

Format:
```
int SpkHalt(unsigned char *SpkParams);
```

Parameter:

`unsigned char *SpkParams`	Pointer to SpkParams (return value of SpkOpen)

Return value: 0.....Error
Not 0...Terminated normally

Description: Stops sound output from the specified channel. If sound output on the specified channel has not been initiated by SpkStart(), the function returns 0. This function can also be called from SpkClose().

SpkAppend( )

Function: Queues output data

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

Parameters:

`unsigned char *SpkParams`		Pointer to SpkParams (return value of SpkOpen)
`void`	`*Buffer`	Pointer to the data to be queued
`int`	`Length`	Data size

Return value: 0.....Error
Not 0...Terminated normally

Description: Queues the data in the output queue for output to the channel specified by SpkParams. If sound output on the specified channel has not been initiated by SpkStart(), or if the queue has no blank entry, no data is queued and the value 0 is returned.

SpkRoom( )

Function: Gets number of remaining queue entries

Format:
```
int SpkRoom(unsigned char *SpkParams);
```

Parameter:

`unsigned char *SpkParams`	Pointer to SpkParams (return value of SpkOpen)

Return value: Number of usable entries

Description: Returns the number of usable entries remaining in the output queue. If this function is called immediately after opening an output channel, the maximum number of entries available for the channel is acquired. The value returned during sound output operation is given by:
(maximum number of entries) - (number of queued entries) - (number of entries not called back)

SpkQueue( )

Function: Gets number of entries waiting for output

Format: `int SpkQueue(unsigned char *SpkParams);`

Parameter: `unsigned char *SpkParams` Pointer to SpkParams (return value of SpkOpen)

Return value: Number of entries waiting for output

Description: Returns the number of entries in the output queue waiting for output. The value returned during sound output operation is given by:
(number of queued entries) - (number of entries not called back) - (number of entries already called back)

In SOUND33, this function is used by specifying the DEBUG option when compiling top-level sources. When the queue = 0, sb33 halts output.

SpkIsRunning( )

Function: Checks output status

Format: `int SpkIsRunning(unsigned char *SpkParams);`

Parameter: `unsigned char *SpkParams` Pointer to SpkParams (return value of SpkOpen)

Return value: 0.....Output operation idle
Not 0...Output operation active

Description: Returns a value indicating whether the specified channel currently is in output operation.

SpkOnDone( )

Function: Enters callback function during playback

Format: `int SpkOnDone(unsigned char *SpkParams, void *Callback);`

Parameters: `unsigned char *SpkParams` Pointer to SpkParams (return value of SpkOpen)
`void *Callback` Pointer to the callback function to be entered

Return value: Pointer to the callback function before being updated

Description: Enters the function for the specified channel which is to be called back during playback. The callback function has the following format:
`void Callback(unsigned char *SpkParams, void *Buffer, int Length)`

SpkOnEmpty( )

Function: Enters callback function when playback finishes

Format: `int SpkOnEmpty(unsigned char *SpkParams, void *Callback);`

Parameters: `unsigned char *SpkParams` Pointer to SpkParams (return value of SpkOpen)
`void *Callback` Pointer to the callback function to be entered

Return value: Pointer to the callback function before updating

Description: Enters the function for the specified channel to be called back when playback finishes. The callback function has the following format:
`void Callback(unsigned char *SpkParams)`

SpkOnNotInTime( )

Function: Enters callback function for playback not in time

Format:
```
int SpkOnNotInTime(unsigned char *SpkParams, void *Callback);
```

Parameters:

`unsigned char *SpkParams`	Pointer to SpkParams (return value of SpkOpen)
`void *Callback`	Pointer to the callback function to be entered

Return value: Pointer to the callback function before updating

Description: Enters the function for the specified channel to be called back when sb33 fails to play synchronously in real time. The callback function has the following format:

```
void Callback(unsigned char *SpkParams, void *Buffer, int Length)
```

In SOUND33, this function is used by specifying the DEBUG option when compiling top-level sources. If sb33 cannot process sound data in time, it halts output.

slGetVersion( )

Function: Gets version information

Format:
```
int slGetVersion(void);
```

Parameters: None

Return value: Version number

Description: Returns the version number of the output (Speak) library. For ver. 2.2, the value returned is 0x22.

5.4.4 Interrupt Processing Functions

Interrupt processing functions SpkIntrX() are used as vectors for 16-bit timer 5 compare B interrupts. Four of these interrupt processing functions (X = 1 to 4) are available for separate use for each output mode. Object files including these functions have also been created individually. Link the object "spkintrX.o" that suits the output mode.

SpkIntr1( ), SpkIntr2( ), SpkIntr3( ), SpkIntr4( )

Function: PWM output interrupt processing

Format:
```
void SpkIntr0(void);  For 15-bit monaural output (spkintr1.o)
void SpkIntr1(void);  For 15-bit stereo output (spkintr2.o)
void SpkIntr2(void);  For 10-bit monaural output (spkintr3.o)
void SpkIntr3(void);  For 9-bit monaural output (spkintr4.o)
```

Parameters: None

Return value: None

Description: These functions perform PWM output processing in an interrupt. These functions can only be used as the vector value for the 16-bit timer 5 compare B interrupt.

5.5 Techniques for Speeding Up Processing

You can accelerate library processing speed to some extent by mapping several library objects to the internal memory before executing. Use the linker's U section feature for object mapping to the internal memory. The required processing is described below:

1. The objects required for high-speed operation are:
   - sndcpy.o (for monaural) or sndcpy2.o (for stereo)
   - snd2.o (for stereo)
   - spkintr1.o (15-bit monaural), spkintr2.o (15-bit stereo), spkintr3.o (10-bit monaural), or spkintr4.o (9-bit monaural)
   - slutil.o (for monaural) or slutil2.o (for stereo)

2. Write the following in the linker command file.
   -objsym ; Creates object symbol
   -section <*name*> ; Creates section symbol
   -ucode <*name*> { <*object file*> [<*object file*>....] } ; Maps into U section
   Example:

```
-objsym
-section CACHE1 = 0x30
-section CACHE2 = 0x160
-section CACHE3 = 0x1e0
-section CACHE4 = 0x26C
-ucode CACHE1 {..\lib\sndcpy2.o}  ; set code sections to absolute address
-ucode CACHE2 {..\lib\spkintr2.o} ; set code sections to absolute address
-ucode CACHE3 {slutil2.o}         ; set code sections to absolute address
-ucode CACHE4 {..\lib\snd2.o}     ; set code sections to absolute address
```

   Be sure to map objects that suit the output mode.

   Examine the map file after linking. You will see that the execution addresses of specified modules are mapped in the internal memory.
   Example: Map file

```
Code Section mapping
Address    Vaddress     Size        File                          ID  Attr
                              :
00600364   000001e0     0000008c    slutil2.o                     0   REL
                              :
00624634   00000030     00000130    ..\lib\sndcpy2.o              0   REL
                              :
00624838   00000160     00000080    ..\lib\spkintr2.o             0   REL
006248b8   0000026c     00000454    ..\lib\snd2.o                 0   REL
                              :
```

3. To transfer object code to the internal memory, use the sndCodecpy() function provided in the top-level source (see Section 5.3.10).
   Example:

```
sndCodecpy(&__START_CACHE1,&__START_sndcpy2_code,&__SIZEOF_sndcpy2_code);
sndCodecpy(&__START_CACHE2,&__START_spkintr2_code,&__SIZEOF_spkintr2_code);
sndCodecpy(&__START_CACHE3,&__START_slutil2_code,&__SIZEOF_slutil2_code);
sndCodecpy(&__START_CACHE4,&__START_snd2_code,&__SIZEOF_snd2_code);
```

5.6 Memory Size and Number of Simultaneously Reproduced Sound Channels

5.6.1 Memory Size

An example of "sndlib\demo3\demo3.srf" is shown below:

demo3

Data of musical piece

Title of musical piece:	Come Together 33
Instruments used:	trumpet, bguiter, eguiter, organ, highhat, snare, bdrum, tom, ccymbal
Play time:	138.3 seconds
Tempo:	120
Sampling rate:	22 kHz
Output mode:	15-bit stereo

ROM Total approx. 150K bytes

SOUND33 library:	2,256 bytes
STB data:	123,068 bytes
ETB data:	4,644 bytes
Sound data:	18,308 bytes
Other:	5,024 bytes

RAM Total approx. 8.2K bytes

slutil2.o:	148 bytes
sndcpy2.o:	304 bytes
spkintr2.o:	128 bytes
snd2.o:	1,108 bytes
sndtop.o:	4,068 bytes (30 channels, tempo = 100, stereo) → Decode 1,108 + channel 88 × 30 = 2,640 + other 320
sndbuf.o:	2,208 bytes

The following is an approximate guide, provided for reference.

Necessary ROM capacity

SOUND33 library:	Approx. 2.1K bytes
Instrument data:	Total size of the binary data for used instruments in sndtool\stb\XX (XX = sampling frequency) and the binary data for used instruments in sndtool\etb.
Sound data:	Total size of the binary files (*.bin) output by snd2bin and midi2snd.

Necessary internal RAM capacity

For sndcpy.o program execution:	212 bytes
For sndcpy2.o program execution:	304 bytes
For sndtop.c channels:	Approx. 90 bytes per channel 90 bytes × number of channels is required. The number of channels by default is 30. Thus, approximately 2.7K bytes of internal RAM is used.

Speak and decode buffers: One 32-bit buffer for decode operation and two 16-bit buffers for Speak operation are required. The Speak and decode buffers are defined in "sndbuf.c" and "sndtop.c", respectively. Set the buffer's data entries according to the musical piece played at the slowest tempo. By default, the buffers are allocated memory assuming that tempo = 100.
The following shows the total size of the buffers for monaural output cases: (For stereo, the buffer size is twice as large.)

Sampling rate	Tempo	Buffer size (decode + Speak)
8 kHz	30	1,360+1,356= 2,712 bytes
	100	404+ 400= 804 bytes
	300	140+ 136= 276 bytes
16 kHz	30	2,676+2,672= 5,348 bytes
	100	804+ 800= 1,604 bytes
	300	276+ 272= 548 bytes
22 kHz	30	3,680+3,676= 7,356 bytes
	100	1,108+1,104= 2,212 bytes
	300	368+ 364= 732 bytes
32 kHz	30	5,332+5,328= 10,660 bytes
	100	1,604+1,600= 3,204 bytes
	300	532+ 528= 1,060 bytes

Speak interrupt routine: Approx. 160 bytes

Stack: Approx. 256 bytes

5.6.2 Number of Simultaneously Reproduced Sound Channels

Table 5.6.1 lists the number of channels that can be used simultaneously for 15-bit output. The measurement conditions are shown below.

Instrument used: piano
Play: Played on 7 octaves from C1 at tempo 100

CPU operating clock: 40 MHz
Internal RAM cache: slutil.o, slutil2.o, sndcpy.o, and sndcpy2.o are copied to the cache before execution.
BSS area: All mapped to the internal RAM (no wait states)

Table 5.6.1 Number of Simultaneously Reproduced Sound Channels

Output mode	Sampling rate			
	8 kHz	16 kHz	22 kHz	32 kHz
15-bit stereo	47 ch	30 ch	23 ch	18 ch (*1)
15-bit monaural	68 ch (*2)	46 ch	36 ch	25 ch

*1) sndbuf.o is assigned to external RAM and snd2.o is used after being transferred to internal RAM.
*2) Shows the maximum number of channels that can be used when internal RAM is 8K bytes.

To confirm that the sound on all channels is synchronous when played on the actual system, define the DEBUG option when compiling "sndtop.c" and include the check function before execution. When sound cannot be processed in time, sb33 halts playback.

5.7 Example Programs

Creating sound output routines is explained below, using the sample program in the "sndlib\demo3\" directory as an example.

Setting interrupt vectors

In the demo3 sample program, the trap table is set in "demo3\atable.s". To use this file, set the start addresses of processing routines corresponding to the trap vector addresses required for your application.

Example: demo3\atable.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   exception       ;   8 reserved
                  :
    .word   exception       ;  48 16-bit Timer #5-0 underflow
    .word   exception       ;  49 16-bit Timer #5-0 compare/match
    .word   SpkIntr2        ;  50 16-bit Timer #5-1 underflow            (*1)
    .word   exception       ;  51 16-bit Timer #5-1 compare/match
                  :
    .word   exception       ;  65 Clock
    .word   exception       ;  66 reserved
    .word   exception       ;  67 reserved
    .word   INTPLAY         ;  68 Ext.Int.#0                             (*2)
    .word   INTREC          ;  69 Ext.Int.#1
    .word   exception       ;  70 Ext.Int.#2
    .word   exception       ;  71 Ext.Int.#3

exception:

    jp 0
```

*1 Set the address of SpkIntrX() function as the 16-bit timer 5 compare match B interrupt vector here. Select the function to enter in the table, depending on the output mode used.

Output mode	Vector value/object
15-bit monaural output (for 20-MHz or 40-MHz operation)	SpkIntr1( )/spkintr1.o
15-bit stereo output (for 20-MHz or 40-MHz operation)	SpkIntr2( )/spkintr2.o
10-bit monaural output (for 40-MHz operation)	SpkIntr3( )/spkintr3.o
9-bit monaural output (for 20-MHz or 40-MHz operation)	SpkIntr4( )/spkintr4.o

*2 This is the interrupt vector for the external switch input to control asynchronous play. To play sound data asynchronously on the target system, create an interrupt handler routine and enter the vector for it in the table according to the port used. For more information on the interrupt handler routine used in this sample, refer to "demo3\int.s".

Boot routine

Enter initial settings at startup. A sample boot routine is prepared as "demo3\boot.s", which sets the stack, enables interrupts, and sets bus conditions. Make sure the stack is allocated to the internal RAM.
Example: demo3\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

 .global END
 END:
         nop
         jp      END
```

Bus condition settings and other such parameters are written in "demo3\demoasm.s". Examine the file for content details.

Sound output routine

The example program shown below is "snddemo3.c" in the "demo3\" directory. When you start the program, it outputs sound created from "sndtool\midi\c33.mid". Press the Play switch on the demonstration board DMT33007 during playback; it will produce the sound of cymbals from the right speaker. Press the Rec switch; it produces the sound of a snare drum from the left speaker. This program can be downloaded to the demonstration boards DMT33007 + DMT33MON + DMT33AMP3 to verify sound output. Refer to the Appendix for operations.
Example: demo3\snddemo3.c

```
/*
 *  snddemo3.c : sound demonstration No.3 main function
 *
 */

#include "snd.h"                                                          (*1)
#include "sndcomn.h"

extern struct SNDDATA1 sd1_c33;                                           (*2)
extern struct SNDDATA2 sd2_c33[];
extern int* iEvent;             // evant flag 0x1: Play 0x2: rec          (*3)

extern int __START_CACHE1;                                                (*4)
extern int __START_sndcpy2_code;
extern int __SIZEOF_sndcpy2_code;
extern int __START_CACHE2;
extern int __START_spkintr2_code;
extern int __SIZEOF_spkintr2_code;
extern int __START_CACHE3;
extern int __START_slutil2_code;
extern int __SIZEOF_slutil2_code;
extern int __START_CACHE4;
```

```
extern int __START_snd2_code;
extern int __SIZEOF_snd2_code;

sndSpeakBatch(struct SNDDATA1 *sd1, struct SNDDATA2 *sd2)                    (*5)
    {
      unsigned char * SpkParams;
      SpkParams = sndSpeak(sd1, sd2);
      if(SpkParams==0)
            return;
      do { } while(SpkIsRunning(SpkParams));
    }

void main()
    {
      sndIntInit();            /* Initialize K64 K65 interupt  */          (*6)
                                                                           (*7)
      sndCodecpy(&__START_CACHE1,&__START_sndcpy2_code,&__SIZEOF_sndcpy2_code);
      sndCodecpy(&__START_CACHE2,&__START_spkintr2_code,&__SIZEOF_spkintr2_code);
      sndCodecpy(&__START_CACHE3,&__START_slutil2_code,&__SIZEOF_slutil2_code);
      sndCodecpy(&__START_CACHE4,&__START_snd2_code,&__SIZEOF_snd2_code);
      setSpeakVolume(0x100);                                               (*8)
      sndSpeakBatch(&sd1_c33, &sd2_c33[0]);                                (*9)
      sndIntClose();           /* Close K64 K65 interupt  */               (*10)
    }

/*
 *  get sound event function
 *
 *  Please change this function for your sound file.
 *  If you use other interrupt please change "int.s" in the sample.
 */

void sndGetEvent(int iEventFlag){                                            (*11)
      /* mode is 4bit data [SW2 SW1 SW4 SW3] */
      switch(iEventFlag & 0xff) {
            case 1:                      // PLAY[K64]
              sndSpeakStart(18);
              break;
            case 2:                      // REC[K65]
              sndSpeakStart(19);
              break;
            case 3:                      // REC/PALY
              sndSpeakStart(18);
              sndSpeakStart(19);
              break;
            default:
              break;
      }
      iEvent=0;
}
```

*1 Includes "snd.h" and "sndcomn.h".

*2 Defines the play structure (SNDDATA1) and channel structure (SNDDATA2) defined in the C source files created by "snd2pcm.exe" as externally referenced.

*3 Defines the flag indicating the asynchronous play switch status as externally referenced.

*4 Defines the transfer of objects to be executed in the internal RAM. For 32-kHz sampling play, always transfer the objects written here to internal RAM.

*5 This program uses SpeakBatch() that produces sound output by calling with the play and channel structures for the musical piece as the parametersand returns after completion of output. The sndSpeakBatch() function uses the passed parameters and calls the top-level function sndSpeak(). During sound output, the program monitors the output status with the SpkIsRunning() function. Following completion of output, it returns to the main routine. This general purpose function can be used without modification. You can use it by copying it as is into the user program.

*6 This routine initializes the interrupt for asynchronous play. For more information, see "demo3\int.s".

*7 Moves objects needed for high-speed operation to the internal RAM.

*8 Sets the sound volume to 0x100 (1.0-fold).

*9 Produces sound output using sndSpeakBatch( ) in *5.

*10 Disables the interrupt for asynchronous play. For more information, see "demo3\int.s".

*11 sndGetEvent() is an event function for asynchronous play, which is called from "sndTop.c". Even when not performing asynchronous play, write it as a dummy function.
This example program outputs the sound of cymbals or a snare drum using the sndSpeakStart() function according to the status of the flag (changed with a switch) which is set by the asynchronous play switch input interrupt routine. The parameters to sndSpeakStart() are channel numbers in the channel structure — 18 for the cymbals channel or 19 for the snare drum channel. (See "demo3\c33.c".)

Appendix Verifying Operation on DMT33 Boards

The following explains how to verify sound output operation, using the E0C33 Family demonstration tools DMT33007, DMT33MON, and DMT33AMP3 to execute a sample program.

A.1 System Configuration Using DMT33007

A.11 Hardware Configuration

Configure the system shown in Figure A.1.1 using the DMT33007, DMT33MON, and DMT33AMP3. This system allows sound output in stereo. Since no sound output circuits are specifically prepared for operational verification, we use the DMT33AMP3, which is capable of stereo sound output. A user-created stereo output circuit can be used as well. For examples of sound output circuits, refer to the "E0C33 Family Application Notes".

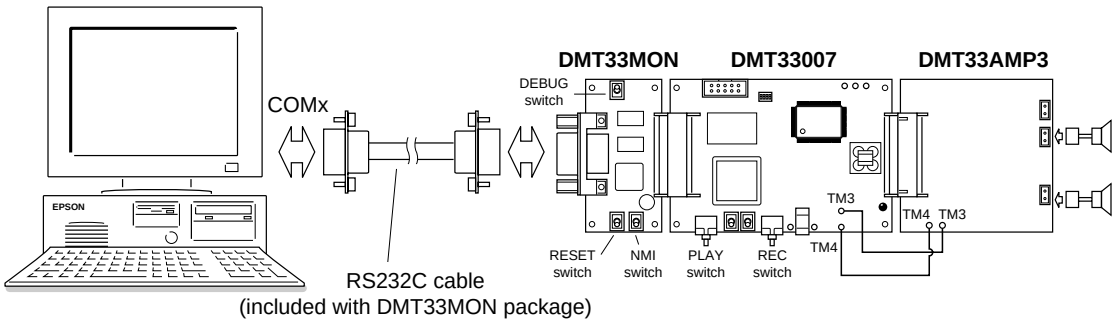


Figure A.1.1 System Configured with DMT33007, DMT33MON, and DMT33AMP3

For details of each DMT board, refer to the "E0C33 Family DMT/EPOD/MEM Board Manual".

System connections

Note: Before connecting or disconnecting to and from the system, always turn off power to all connected boards and equipment. For more information on precautions to observe when using each board, see the "E0C33 Family DMT/EPOD/MEM Board Manual".

1. Attach the DMT33MON and DMT33AMP3 to the DMT33007.
2. Connect the speaker to the DMT33AMP3.
3. Connect the DMT33MON and a PC (com1) with the RS232C cable (included with the DMT33MON package).
4. Set the [DEBUG] switch (SW3) on the DMT33MON to the ON position.
5. Place a battery in the battery holder (included with the DMT33007) and connect it to the DMT33007.
6. Turn on power to the PC.

A.12 Software

The PC hosting SOUND33 must have the E0C33 Family C Compiler Package development tools installed. Downloading a program into the DMT33007 with the debug monitor requires debugger (db33) Ver. 1.72 or later.

A.2 Program Execution Procedure

Each sample program directory (sndlib\demoX\) also contains absolute object files in executable format. There is no need to compile or link a sample program before use.
The following explains how to verify the operation of a program after downloading into the DMT33007.
In the following explanation, we use the sample program "snddemo3.srf" in the "sndlib\demo3\" directory.

(1) Connect the boards and a PC as described in Section A.1.1, then switch on the power to each piece of equipment.

(2) Before a program can be downloaded, the debug monitor must be operating on the DMT33007. After reconfirming that the [DEBUG] switch (SW3) of the DMT33007 is set to the ON position, reset the system using the [RESET] switch (SW1).

(3) Start workbench wb33 and make "sndlib\demo3\" the current directory. For debugger options, choose the following:
• Choose MON mode.
• Choose the port (com1) that connects to the DMT33MON and set the transfer rate to 115,200 bps.
• Check [db33*.cmd file] and choose the command file "snddemo3.cmd".
Choose "33208_v.par" as a parameter file and start the debugger.

The debugger can also be started from the DOS prompt without wb33, as follows:
(When "sndlib\demo3\" is the current directory)

```
>C:\cc33\db33 -mon -b 115200 -p 33208_v.par -c snddemo3.cmd
```

(4) When the debugger starts, the sample program (snddemo3.srf) is loaded into the RAM (0x600000 and above) of the DMT33007 by the commands written in "snddemo3.cmd".

(5) When you execute the g command, the system starts outputting a sound.

When the program begins running, it outputs sound created from "sndtool\midi\c33.mid" (for approximately 2 minutes, 18 seconds). For a demonstration of asynchronous play, press the Play switch on the DMT33007 during playback to produce the sound of cymbals from the right speaker. Press the Rec switch to produce the sound of snare drums from the left speaker. Pressing the Play and the Rec switches simultaneously produces the sounds of cymbals and snare drums from the respective right and left speakers at the same time.

When the program finishes sound (c33.mid) playback, it is halted by a hardware break. To resume playback, execute the rsth command and the g command once again.

A.3 Building a Program

The sample programs may be modified before testing, as necessary. The following describes the procedure to build a program and the files required.
In this explanation, as in Section A.2, we use a sample program (snddemo3) included in the "sndlib\demo3\" directory.

A.3.1 Explanation of Files

Source files

The main program for the DMT33007 (E0C33208) is "snddemo3.c". This sample program was created assuming it will be run on the E0C33208 operating at 40 MHz.
In addition, the following files are also used:

demo3\atable.s	Trap vector table
demo3\boot.s	Boot routine
demo3\demoasm.s	Device initialize routine
demo3\int.s	Asynchronous play interrupt handler routine
src\sndTop.c	SOUND33 top-level source
src\slutil2.c	Final PWM output data creating routine (stereo)
src\sndbuf.c	Speak buffer setup file
demo3\c33.c	Sound structure definition
demo3\stb_c33.s	WAVE table data for main sound
demo3\etb_c33.s	Envelope table data for main sound
demo3\stb_beat.s	WAVE table data for asynchronous play sound
demo3\etb_beat.s	Envelope table data for asynchronous play sound
demo3\setbl.c	Instrument structure definition

For more information on creating sound output routines, refer to Section 5.7, "Example Programs". For more information on creating sounds and tone quality data, refer to Section 3, "Software Development Procedure".

Linker command file

The contents of the linker command file (snddemo3.cm) used to link the sample program are shown below. Since the sample program is run in the external RAM of the DMT33007, the start address of the CODE section is set to 0x600000. Virtual sections are defined to enable the transfer of objects required for fast operation to internal RAM before execution. See below for additional information, including information on the files required for linking.

```
-objsym
-w
-d

;Map set
-code 0x00600000            ; set relative code section start address
-bss  0x006c0               ; set relative bss  section start address

-section CACHE1 = 0x30
-section CACHE2 = 0x160
-section CACHE3 = 0x1e0
-section CACHE4 = 0x26C
-ucode CACHE1 {..\lib\sndcpy2.o}  ; set code sections to absolute address
-ucode CACHE2 {..\lib\spkintr2.o} ; set code sections to absolute address
-ucode CACHE3 {slutil2.o}         ; set code sections to absolute address
-ucode CACHE4 {..\lib\snd2.o}     ; set code sections to absolute address
-bss  0x0680000 {sndbuf.o}        ; set bss  sections to absolute address

;Library path
-l C:\cc33\lib
-l ..\lib
```

```
;Executable file
-o snddemo3.srf

;Object files
atable.o
boot.o
int.o
demoasm.o
snddemo3.o
slutil2.o
sndtop.o
c33.o
snd_c33.o
setbl.o
stb_c33.o
etb_c33.o
sndbuf.o

;Sound library files
..\lib\sndcpy2.o
..\lib\sndcpy.o
..\lib\sl208.lib
..\lib\spkintr2.o
..\lib\snd2.o
..\lib\snd.lib
;spk.lib
;sl208.lib

;Library files
;io.lib
;lib.lib
math.lib
string.lib
ctype.lib
fp.lib
idiv.lib
```

A.3.2 make

To build the above sample program, use a make file "snddemo3.mak". If you corrected the source file, you need to create the object file in executable format "snddemo3.srf" by using "snddemo3.mak".

Execution procedure for make

1. Set "sndlib\demo3\" to the current directory.
2. Enter the command shown below from the DOS prompt:
   `C:\E0C33\SND33\SNDLIB\DEMO3>C:\CC33\make -f snddemo3.mak`

You also can execute make.exe from workbench wb33. (Refer to the "E0C33 Family C Compiler Package Manual".)

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 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 Asia
421-8, Hino, Hino-shi, Tokyo 191-8501, JAPAN
Phone: +81-(0)42-587-5814 Fax: +81-(0)42-587-5110

In pursuit of **"Saving" Technology**, Epson electronic devices.
Our lineup of semiconductors, liquid crystal displays and quartz devices
assists in creating the products of our customers' dreams.
Epson IS energy savings.

EPSON

SEIKO EPSON CORPORATION
ELECTRONIC DEVICES MARKETING DIVISION

■ EPSON Electronic Devices Website

http://www.epson.co.jp/device/

Issue JULY 2000, Printed in Japan Ⓜ A