MF1257-01

EPSON

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

MELODY33 MIDDLEWARE MANUAL

ENERGY SAVING EPSON

SEIKO EPSON CORPORATION

NOTICE

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

Windows95 and Windows NT are registered trademarks of Microsoft Corporation, U.S.A.
PC/AT and IBM are registered trademarks of International Business Machines Corporation, U.S.A.
All other product names mentioned herein are trademarks and/or registered trademarks of their respective owners.

PREFACE

Written for developers of application systems using the E0C33 Family of microcomputers, this manual describes the configuration, functions, and operation of the MELODY33 as melody output middleware for the E0C33 Family.

CONTENTS

1 Outline of the MELODY33 Middleware

MELODY33 is the melody output middleware for the E0C33 Family of microcomputers that is capable of outputting a melody from ROM using pulse-width-modulation with a 16-bit timer.

The output routine is supplied as a library function, which is linked for use with the target program. In addition, the MELODY33 package includes melody ROM data creation tools and a tool for evaluating the melody created on a PC.

Its main features are listed below:

- Capable of simultaneously outputting up to four channels (maximum three channels for the E0C33A104)
- Outputs melodies in five octaves.
  Standard: 61 scales (including semitones) from C3 (131 Hz) to C8 (4,186 Hz)
- Supports whole notes to thirty-second notes.
- Supports ♩ = 30 ~ 300 tempos.
- Supports direct drive (inverted signal output) of a piezoelectric buzzer (E0C332xx only).
- Can reproduce and evaluate created melody data using a Windows GUI tool on a PC.

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 MELODY33 Package

The following lists the contents of the MELODY33 package. After unpacking, check to see that all items are included with your package.

(1) Tool disk (CD-ROM)	1 disk
(2) E0C33 Family MELODY33 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 Melody Output System

The MELODY33 library is a middleware positioned between the E0C33 hardware and the user program, providing hardware control for melody output.

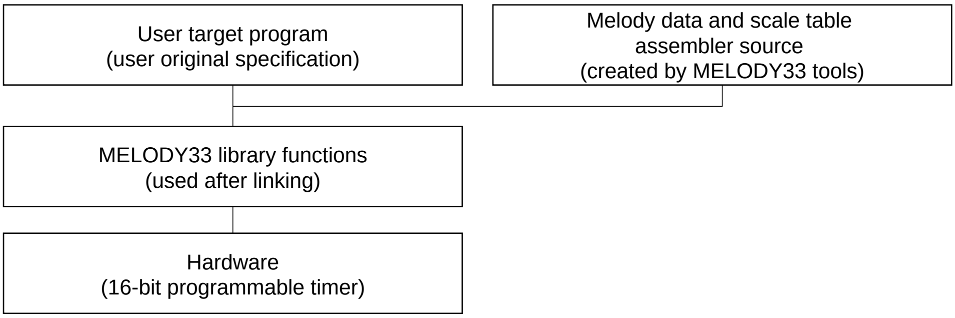
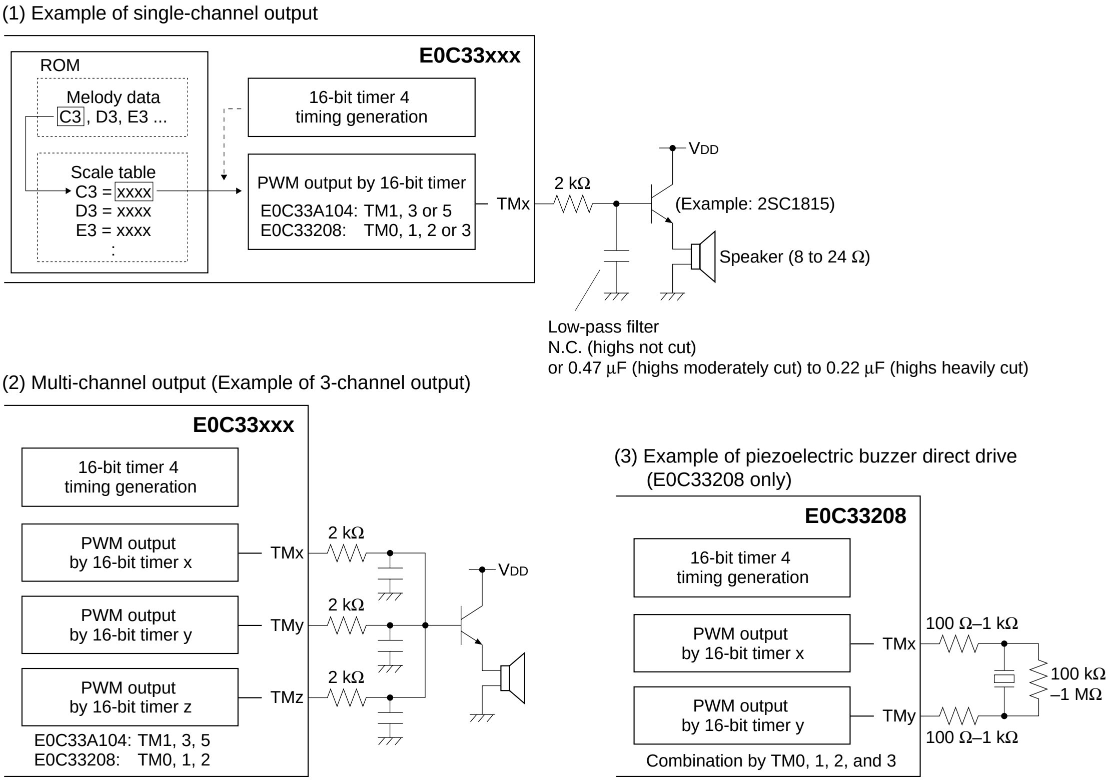

User target program
(user original specification)
Melody data and scale table
assembler source
(created by MELODY33 tools)
MELODY33 library functions
(used after linking)
Hardware
(16-bit programmable timer)

Figure 1.2.1 Software Configuration of the Melody Output System

For more information on the MELODY33 library, see Section 5, "MELODY33 Library Reference".

The MELODY33 library uses one to four channels of 16-bit programmable timers on the E0C33 chip to output melody signals. This output drives a speaker or piezoelectric buzzer, as shown below.


(1) Example of single-channel output

(2) Multi-channel output (Example of 3-channel output)

(3) Example of piezoelectric buzzer direct drive
(E0C33208 only)

Figure 1.2.2 Hardware Configuration of the Melody Output System

1.3 MELODY33 Tools

MELODY33 tools are PC software for creating and evaluating the melody ROM data to be written to the E0C33 Family chip. All of these tools run under Windows 95, Windows NT 4.0, or later versions.

For more information on creating melody data, see Section 3, "Software Development Procedure". For more information on MELODY33 tools, see Section 4, "MELODY33 Tool Reference".

2 Installation

This section describes the operating environment for MELODY33 tools and explains how to install the MELODY33 middleware.

2.1 Operating Environment

Melody ROM data creation and evaluation by MELODY33 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 MELODY33 tools run under Microsoft® Windows®95, Windows NT®4.0, or higher versions (in Japanese or English).

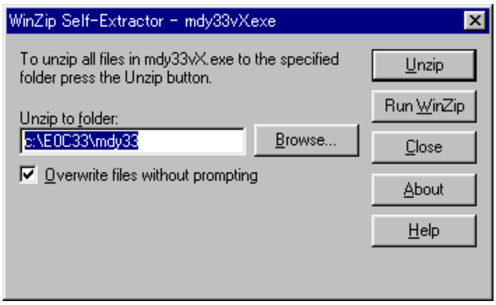
Other requirements

E0C33 Family C Compiler Package is required for software development.

2.2 Method of Installation

The MELODY33 library and MELODY33 tools are supplied on CD-ROM. Open the self-extracting file on the CD-ROM named "mdy33vXX.exe" to install the MELODY33 library and MELODY33 tools in your computer. (The XX in this file name denotes a version number. For Version 1.0, for example, the file is named "mdy33v10.exe".)

Double-click on "mdy33vXX.exe" to start installation. The dialog box shown below appears.



WinZip Self-Extractor - mdy33vXexe

To unzip all files in mdy33vX.exe to the specified folder press the Unzip button.

Unzip to folder: c:\E0C33\mdy33

☑ Overwrite files without prompting

Enter the path and folder name under which you want to install the files in the text box and click on the [Unzip] button. The specified folder is created and all files are copied into it.

If the specified folder already exists in the specified path and [Overwrite Files Without Prompting] is checked (turned on), the files in the folder are overwritten without asking for your confirmation.

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

```
(root)\
        readme.txt                  Supplementary explanation, etc. (in English)
        readmeja.txt                Supplementary explanation, etc. (in Japanese)
        mdytool\  ..... MELODY33 tool directory
                  readme.txt        MELODY33 tool supplementary explanation, etc. (in English)
                  readmeja.txt      MELODY33 tool supplementary explanation, etc. (in Japanese)
                  bin\      .....MELODY33 tools
                            mb33.exe          Melody data creation/evaluation tool
                            txt2mdy.exe       Melody text → MDY file conversion tool
                            mdy2bin.exe       MDY file → binary file conversion tool
                            mdy2pcm.exe       MDY file → PCM file conversion tool
                            mtb.exe           Scale table creation tool
                            bin2s.exe         Binary→ assembly source conversion tool
                            pcm2wav.exe       PCM → WAV conversion tool
                            pcm_add.exe       PCM data synthesizing tool
                            pcm_mul.exe       PCM data amplitude adjusting tool
                            bdmp.exe          Binary file dump program
                            ccap.exe          Tool message filing tool
                            mci32.ocx, msvcrt40.dll, olepro32.dll, spin32.ocx, vb40032.dll
                                              Files for mb33
                  sample\   .....Sample directory
                            Sample melody data, batch files, etc.
        mdylib\   ..... MELODY33 library-related
                  readme.txt        MELODY33 library supplementary explanation, etc. (in English)
                  readmeja.txt      MELODY33 library supplementary explanation, etc. (in Japanese)
                  lib\      ..... MELODY33 library directory
                            mdy104.lib   MELODY33 library for E0C33A104
                            mdy208.lib   MELODY33 library for E0C33208
                  include\  ..... MELODY33 library function header file directory
                            mdy.h        Library include file
                  libsrc\   ..... Hardware-dependent source directory
                            mdy104.c     Low-level functions for E0C33A104 control
                            mdy208.c     Low-level functions for E0C33208 control
                  demoX\    ..... Sample program directory
                           (For details on the configuration of sample programs, refer to "readme.txt "or
                           "readmeja.txt" in "mdylib".)
```

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

3 Software Development Procedure

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

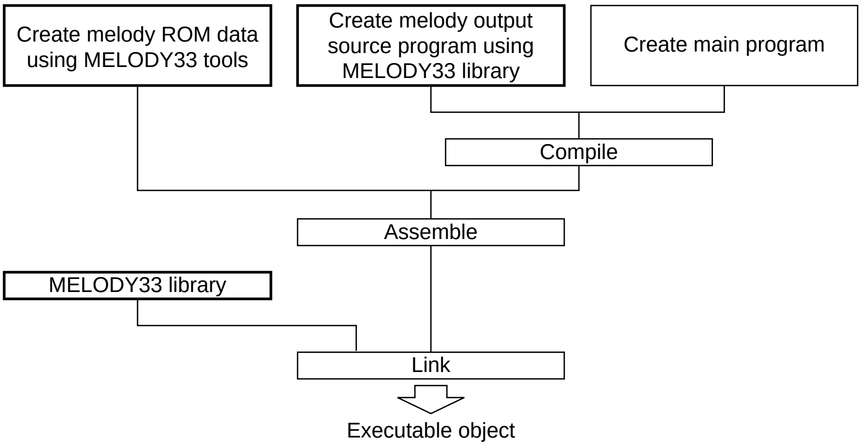

Create melody ROM data using MELODY33 tools
Create melody output source program using MELODY33 library
Create main program
Compile
Assemble
MELODY33 library
Link
Executable object

Figure 3.1 Procedure for Developing E0C33 Melody Output Software

1) Create a text file containing a description of melody data, then create the assembly source file of melody ROM data using MELODY33 tools. Also create an assembly source file for a scale table matched to the chip's operating frequency.

2) Create a user program. Melody output is obtained by calling MELODY33 library functions from the program. The source of melody ROM data created in Step 1 may be included in the user program source.

3) Compile and assemble the source program.

4) Link the object generated in Step 3 and MELODY33 library functions to generate an executable object file.

3.1 Creating Melody ROM Data

Figure 3.1.1 shows the procedure for creating melody ROM data and the configuration of MELODY33 tools.

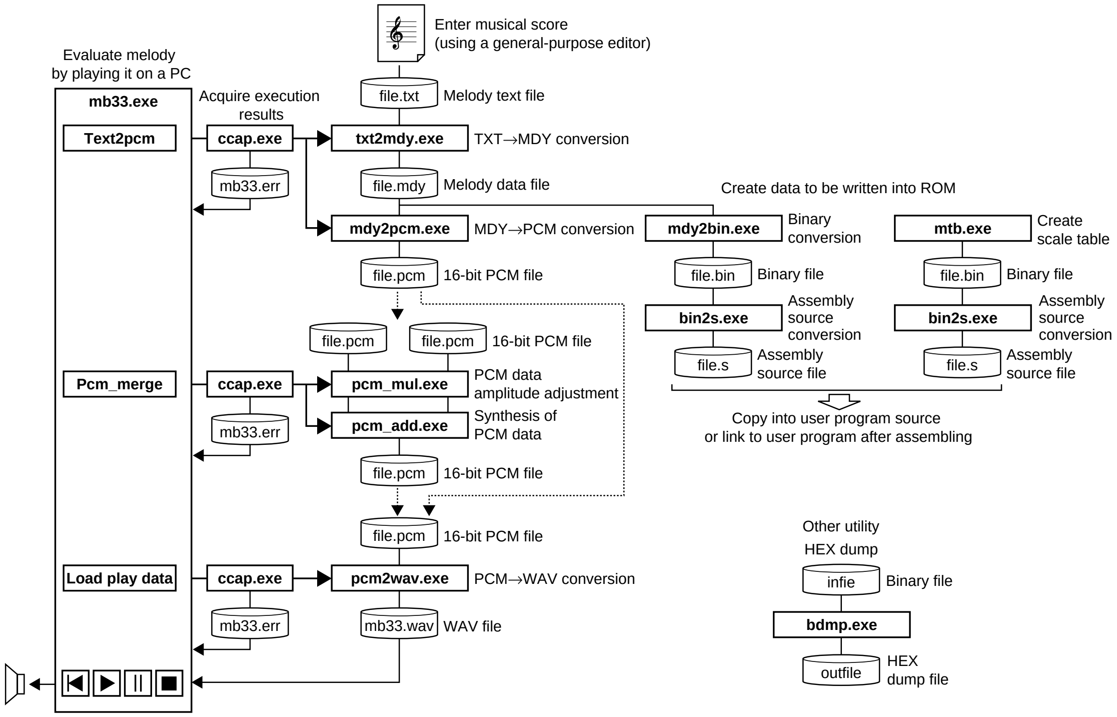


Figure 3.1.1 Flow Chart for Creating Melody ROM Data

Only an outline of the MELODY33 tools is given here. For more information, see Section 4, "MELODY33 Tool Reference".

The following explanation uses sample files in the "mdytool\sample\" directory. Also, the explanation below assumes that "mdytool\sample\" is the current directory, and that PATH is set in the "mdytool\bin\" directory.

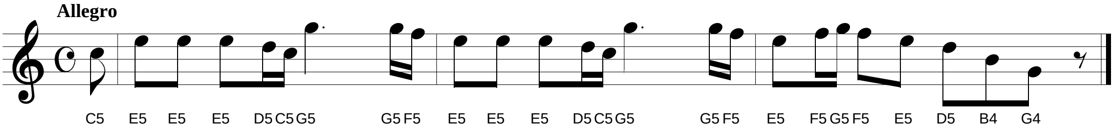
Example:
```
DOS>CD e0c33\mdy33\mdytool\sample
DOS>PATH c:\e0c33\mdy33\mdytool\bin
```

Note: PCM files handled by MELODY33 tools are in 44.1 kHz, 16-bit monaural, little-endian format.

3.1.1 Creating a Melody Text File

Using an editor, enter data for the music and save as a standard text file (.txt). To output different melodies from multiple channels, create a separate text file for each channel.
The music and data shown below are written in "spring.txt" (Vivaldi's *The Four Seasons*, "Spring") in the "mdytool\sample\" directory.

"spring.txt"



Enter the 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   -1                        8   -1
```

Each line represents one note or rest in the format shown below.

Specification of a note: <length of sound> <pitch of sound>
Specification of a rest: <length of break> -1

The <length of sound> and <pitch of sound> may be written sequentially. Intervening spaces or tabs are ignored. The <length of sound> (<length of break>) may be specified from whole notes (whole rest) to thirty-second notes (thirty-second rest) using the number values 1, 2, 4, 8, 16, and 32.

Table 3.1.1.1 Specification of Notes and Rests

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

Specify the <pitch of sound> by a pitch name and an octave number (3 to 8). The pitch name and octave number must always be written back to back, without intervening spaces or other characters.

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

Valid specification range: do3/C3 (131 Hz) to do8/C8 (4,086 Hz)

∗ When played, the specified musical intervals can be shifted in the range of -2 to +2 octaves.

To specify a dot or tie

The lengths of notes on a musical score do not indicate actual playing time. Notes normally include a short break time and are played discretely. Specify this break time when converting the created text file into a melody data file. (The break time specified here can be 1/4, 1/2, or 1/1 of a thirty-second note.)

Dots are represented by writing a half-length note after any note. (Example: Dotted quarter note = quarter note + eighth note) In this case, the break time described above must be eliminated from between these notes, which is accomplished by writing 'c' after <pitch of sound>. Always insert a space or tab between <pitch of sound> and the 'c'.

Example: Dotted quarter note

```
    4      so5   c
    8      so5
```

These two notes are played without separating their sounds. You may specify dotted notes ranging from dotted whole notes to dotted sixteenth notes. This can be used to specify a tie.

Note: In the following cases, an error occurs if you convert the created text file into a melody data file with "txt2mdy.exe".

- When any value other than 1, 2, 4, 8, 16, or 32 is specified for <length of sound (break)> Spaces and tabs preceding <length of sound (break)> are ignored.
- When any symbol or numeric value out of the valid range is specified for <pitch of sound>
- When any space or tab is inserted between the pitch name and octave number specified for <pitch of sound>
- When any character other than space, tab, or new line is written immediately after <pitch of sound>
  When writing 'c', always be sure to insert a space or tab between <pitch of sound> and 'c'. Following a space or tab after <pitch of sound>, all characters except 'c' are ignored.

3.1.2 Evaluating Melody Data Using mb33

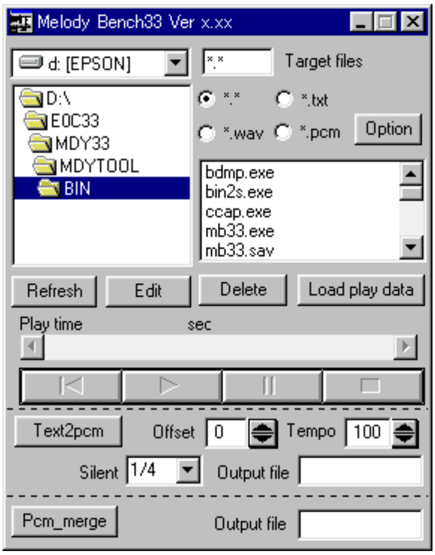
Melody bench mb33 allows you to convert a melody text file and play it on a PC. The following describes the basic procedure for using "mb33.exe".

(1) Starting mb33.exe

Mb33.exe

Double-click the "mb33.exe" icon to start mb33. To quit, click the [Close] button on the title bar.

When mb33 starts, the [Melody Bench33] window appears.



Melody Bench33 Ver x.xx

d: [EPSON]

. Target files

D:\
E0C33
MDY33
MDYTOOL
BIN

● *.* ○ *.txt
○ *.wav ○ *.pcm

bdmp.exe
bin2s.exe
ccap.exe
mb33.exe
mb33.sav

Play time sec

Offset 0 Tempo 100

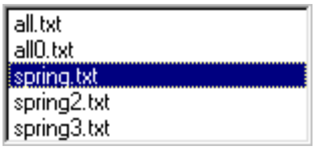
Silent 1/4 Output file

Output file

[Melody Bench33] window

(2) Converting a melody text file

To play on a PC, first create PCM data from the melody text file.



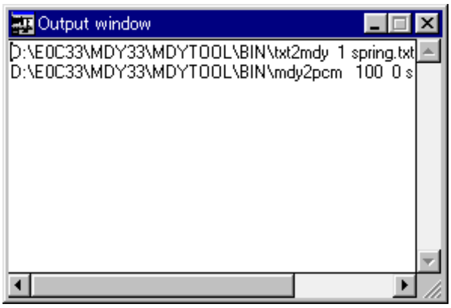
1. Choose "mdytool\sample\" from the directory list box and then "spring.txt" from the file list box.

∗ The selected text file can be opened and edited in "Notepad" using the [Edit] button. You may also change to any other text editor by making a selection with the [Option] button.

[Text2pcm] button

2. Click the [Text2pcm] button.

When you click the [Text2pcm] button, mb33 starts two MELODY33 tools to convert the selected melody text file into a PCM file.



```
D:\E0C33\MDY33\MDYTOOL\BIN\txt2mdy 1 spring.txt
D:\E0C33\MDY33\MDYTOOL\BIN\mdy2pcm  100  0 s
```

[Output] window

Tool startup commands and tool messages are displayed in the [Output] window, which opens as necessary (for default settings).

txt2mdy.exe

Converts a melody text file into a melody data file.

mdy2pcm.exe

Converts a melody data file into a PCM file.

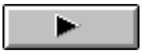
This generates the file "spring.pcm". To create a PCM file under another name, enter the desired name in the [Output file] text box below [Tempo], then click the [Text2pcm] button.

(3) Playing music

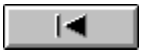
[Load play data] button

1. Choose the "spring.pcm" created in (2) from the file list box and click the [Load play data] button.

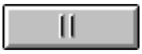
   mb33 converts "spring.pcm" into a WAV file "mb33.wav" using "pcm2wav.exe" before loading.



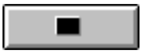
[Play] button



[Beginning] button



[Pause] button



[Stop] button

2. Click the [Play] button to play the melody.

   As with general players, use the buttons shown below to control melody playback.

   [Beginning] button: Returns to the beginning of a melody.

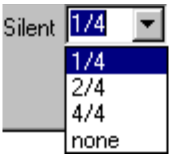
   [Pause] button: Temporarily stops playback. Use the [Play] button to restart from the position at which playback stopped.

   [Stop] button: Stops playback and returns to the beginning of a melody.

Note: PCM and WAV files are reproduced at 44.1 kHz. For playback on a PC, the resolution is insufficient with 44.1 kHz, making the tones do7/C7 (2,093 Hz) to do8/C8 (4,186 Hz) appear to be out of tune. For playback on the actual IC, however, no tonal deviation occurs.

(4) Available options for melody data conversion

Although we converted data with the default settings in (2), you can select the following options before conversion. When options are changed, repeat steps (2) and (3).



Silent 1/4

[Silent]

Selects the length of a break inserted to play music by separating each note.

1/4	1/4 the length of a thirty-second note
2/4	1/2 the length of a thirty-second note
4/4	Equal to the length of a thirty-second note
none	None

When you choose *1/4* or *2/4*, the time during which each note is generated is reduced by that fraction, without changing the length of the overall music.

When you choose *4/4*, a thirty-second rest is inserted between each note, and the length of music is extended.

When you choose *none*, no break is inserted, and the music is played one note after another from beginning to end.

Offset 0

[Offset]

Shifts musical intervals in semitone increments. This can be specified over a range of -24 to +24 (±2 octaves). Generating sounds outside the range do3/C3 through do8/C8 on the chip requires a scale table created for that purpose.

Tempo 100

[Tempo]

Specifies the tempo at which to play music, using a value from 30 to 300. This value represents the number of quarter notes to be played per minute.

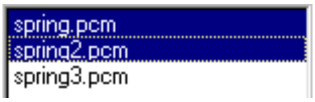
The following shows typical words representing tempos and the approximate values set for Tempo.

Name of tempo	Tempo range	Approximate value
Lento, Largo, Grave	30–50	40
Adagio	40–60	50
Larghetto	50–60	55
Adagietto	60–70	65
Andante	65–75	70
Andantino	70–80	80
Moderato	80–90	90
Allegretto	90–100	100
Allegro, Conmoto	100–130	120
Vivo	110–140	130
Vivace	140–170	150
Presto	170–200	180
Prestissimo	200–	200

(5) Synthesizing melodies

The MELODY33 library supports up to four-channel melody output (for the E0C33208). When a piece of music is created in separate multiple parts, as for the samples "spring.txt", "spring2.txt", and "spring3.txt", these data must be synthesized into one PCM file before being played for evaluation by mb33. The MELODY33 tool "pcm_add.exe" synthesizes two PCM files. This tool can also be executed from mb33. To synthesize "spring.pcm" and "spring2.pcm" for example, follow the procedure given below.

1. Convert "spring2.txt" into a PCM file in the same way as for "spring.txt". Make sure all options are set in the same way.



2. Choose "spring.pcm" and "spring2.pcm" from the file list box. To select two files, select a file, then hold down the [Ctrl] key while selecting another.

[Pcm_merge] button

3. Click the [Pcm_merge] button. Two PCM data are synthesized to generate the file "spring_spring2.pcm." This file can be reproduced following the same procedure described in (3), "Playing music". Note that simply combining two files using "pcm_add.exe" produces excessively large amplitude. Before synthesizing files, mb33 calls the amplitude adjusting tool "pcm_mul.exe" to adjust the amplitude of each PCM data to 50%.

To synthesize four channels, follow the procedure described below.
Example: Synthesizing "ch1.pcm", "ch2.pcm", "ch3.pcm", and "ch4.pcm"
1. Synthesize "ch1.pcm" and "ch2.pcm" (to create "ch1_ch2.pcm").
2. Synthesize "ch3.pcm" and "ch4.pcm" (to create "ch3_ch4.pcm").
3. Synthesize the two files ("ch1_ch2.pcm" and "ch3_ch4.pcm") created in 1 and 2.

A file "ch1_ch2_ch3_ch4.pcm" is generated in which the four files have been synthesized.

Synthesizing three channels requires extra care.
Example: Synthesizing "ch1.pcm", "ch2.pcm", and "ch3.pcm"
If the file derived by synthesizing "ch1.pcm" and "ch2.pcm" is synthesized directly with "ch3.pcm", the sound volume in "ch3.pcm" increases two-fold relative to the other. Follow the procedure given below as you synthesize three files.
1. Create a PCM file having size 0 (e.g., "zero.pcm").
   You can use a blank text file ("zero.txt") for this purpose by changing its extension to ".pcm".
2. Synthesize "ch1.pcm" and "ch2.pcm" (to create "ch1_ch2.pcm").
3. Synthesize "ch3.pcm" and "zero.pcm" (to create "ch3_zero.pcm").
   This will reduce the sound volume of "ch3.pcm" to half.
4. Synthesize the two files ("ch1_ch2.pcm" and "ch3_zero.pcm") created in 2 and 3.

A file "ch1_ch2_ch3_zero.pcm" is generated, in which all files have been synthesized.

3.1.3 Converting Melody Data into an Assembly Source File

To include or link the created melody data in or to the user program, generate an assembly source file for the E0C33 assembler. The following operations must all be performed from the DOS prompt.

1. Convert the melody text file into a melody data file, using "txt2mdy.exe".
   Example: `DOS>txt2mdy 1 spring.txt spring.mdy`

   In this example, "spring.txt" is converted into "spring.mdy". The option "1" inserts a rest of 1/4 the length of a thirty-second note into each note (The number "2" inserts a 2/4 long rest, "3" inserts a 4/4 long rest, and "0" inserts no rest). If [Text2pcm] has been executed in mb33, you do not execute this tool, because the melody data file (.mdy) has already been created.
   In any case, the converted melody data file (.mdy) is a text file and can be displayed on screen using a text editor. For file contents, see Section 4.2.1, "txt2mdy.exe".

2. Convert the melody data file into a binary file using "mdy2bin.exe".
   Example: `DOS>mdy2bin spring.mdy spring.bin`

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

3. Convert the melody binary file into an assembly source file using "bin2x.exe".
   Example: `DOS>bin2s spring.bin > spring.s` (using the DOS redirect function)

   In this example, "spring.bin" is converted into "spring.s". The file "spring.s" is generated using the input file name "spring" as a global symbol. (The symbol name can be changed using the "-l *symbol*" option of "bin2s.exe".)

   Contents of "spring.s"

```
         .global    spring
         .align     2
spring:
         .byte      0x02 0x33 0x99 0x3e 0x9d 0x3e 0x9d 0x3e
         .byte      0x9d 0x3e 0x5b 0x3e 0x59 0x3e 0xe0 0xa0
         .byte      0x3e 0x60 0x3e 0x5e 0x3e 0x9d 0x3e 0x9d
         .byte      0x3e 0x9d 0x3e 0x5b 0x3e 0x59 0x3e 0xe0
         .byte      0xa0 0x3e 0x60 0x3e 0x5e 0x3e 0x9d 0x3e
         .byte      0x5e 0x3e 0x60 0x3e 0x9e 0x3e 0x9d 0x3e
         .byte      0x9b 0x3e 0x98 0x3e 0x94 0x3e 0xff 0x3e
         .byte      0x00
; total 57 bytes data
```

Since the above tools are programs executable from the DOS prompt, they may be executed by creating a batch file. For batch files corresponding to the samples, see Section 4.2.10.

3.1.4 Creating Scale Table Data

To generate the correct musical scale on the E0C33 chip, create a scale table. This table consists of 16-bit timer setup values corresponding to the entire frequency spectrum, from do3/C3 to do8/C8, and is written to ROM along with the melody data. To create a scale table, follow the procedure given below.

1. Create a scale table data file (binary file) using "mtb.exe".
   Example: `DOS>mtb 2.5 scale.bin`

   For the first parameter — which in this example is set to "2.5" — specify the input clock frequency (MHz) of the 16-bit timer by which melody is output.
   For the E0C33A104, specify 1/8 of the CPU operating frequency.
   Example: CPU operating frequency = 20 MHz → Specified value = 20/8 = 2.5
   CPU operating frequency = 33 MHz → Specified value = 33/8 = 4.125

   For the E0C33208, specify 1/16 of the CPU operating frequency.
   Example: CPU operating frequency = 20 MHz → Specified value = 20/16 = 1.25
   CPU operating frequency = 33 MHz → Specified value = 33/16 = 2.0625
   CPU operating frequency = 40 MHz → Specified value = 40/16 = 2.5

This basic setting allows a melody to be output in a range from do3/C3 (131 Hz) to do8/C8 (4,186 Hz).
For special effects, you can specify values differing from the basic setting to reproduce scales out of this range.
For example, specify 5 for 2.5 in the above example to create a scale table an octave lower, from do2/C2 (65.5 Hz) to do7/C7 (2,093 Hz). Conversely, specify 1.25 to create a scale table one octave higher, from do4/C4 (262 Hz) to do9/C9 (8,372 Hz).
You can prepare multiple scale table data on the E0C33 chip and select one of those tables to be used by library functions.

Note: Specify frequency values in the range 0.5 to 8.0. Values lower than 0.5 produce tonal deviations in high frequencies, while values greater than 8 impair sound playback in low frequencies.

2. Convert the scale table data into an assembly source file using "bin2s.exe".
Example: `DOS>bin2s scale.bin > scale.s` (using the DOS redirect function)

In this example, "scale.bin" is converted into "scale.s". The file "scale.s" is generated using the input file name "scale" as a global symbol. (The symbol name can be changed using the "-l *symbol*" option of "bin2s.exe".)

Contents of "scale.s"

```
         .global    scale
         .align     2
scale:
        .byte       0x01 0x33 0xa7 0x4a 0x76 0x46 0x82 0x42
        .byte       0xc6 0x3e 0x40 0x3b 0xed 0x37 0xc9 0x34
        .byte       0xd3 0x31 0x07 0x2f 0x63 0x2c 0xe6 0x29
        .byte       0x8c 0x27 0x53 0x25 0x3b 0x23 0x41 0x21
        .byte       0x63 0x1f 0xa0 0x1d 0xf6 0x1b 0x65 0x1a
        .byte       0xe9 0x18 0x84 0x17 0x32 0x16 0xf3 0x14
        .byte       0xc6 0x13 0xaa 0x12 0x9e 0x11 0xa0 0x10
        .byte       0xb2 0x0f 0xd0 0x0e 0xfb 0x0d 0x32 0x0d
        .byte       0x75 0x0c 0xc2 0x0b 0x19 0x0b 0x79 0x0a
        .byte       0xe3 0x09 0x55 0x09 0xcf 0x08 0x50 0x08
        .byte       0xd9 0x07 0x68 0x07 0xfe 0x06 0x99 0x06
        .byte       0x3a 0x06 0xe1 0x05 0x8c 0x05 0x3d 0x05
        .byte       0xf1 0x04 0xaa 0x04 0x67 0x04 0x28 0x04
        .byte       0xec 0x03 0xb4 0x03 0x7f 0x03 0x4d 0x03
        .byte       0x1d 0x03 0xf0 0x02 0xc6 0x02 0x9e 0x02
        .byte       0x79 0x02 0x55 0x02
; total 124 bytes data
```

Include the created scale table data into the user program along with the melody data, or link it after assembly.

Table 3.1.4.1 shows the reference frequency of each musical interval and the frequencies generated on the chip.
The fPWM denotes the 16-bit timer's input clock frequency (parameter value of "mtb.exe").
The error is relative (1/1000%), compared to the reference frequency.

Table 3.1.4.1 Frequency List

Interval	Reference	f_{PWM} = 1.25 MHz		f_{PWM} = 2.5 MHz		f_{PWM} = 5 MHz	
	Frequency (Hz)	Frequency (Hz)	Error	Frequency (Hz)	Error	Frequency (Hz)	Error
C3	130.8128	130.8216	6.73	130.8147	1.45	130.8147	1.45
C#3	138.5913	138.5963	3.61	138.5963	3.61	138.5963	3.61
D3	146.8325	146.8343	1.23	146.8343	1.23	146.8343	1.23
D#3	155.5635	155.5694	3.79	155.5694	3.79	155.5694	3.79
E3	164.8138	164.8207	4.19	164.8207	4.19	164.8152	0.85
F3	174.6143	174.6298	8.88	174.6176	1.89	174.6176	1.89
F#3	184.9973	184.9933	-2.16	185.0070	5.24	185.0002	1.57
G3	195.9978	196.0169	9.75	196.0016	1.94	196.0016	1.94
G#3	207.6525	207.6412	-5.44	207.6584	2.84	207.6584	2.84
A3	220.0000	219.9930	-3.18	220.0123	5.59	220.0026	1.18
A#3	233.0820	233.0785	-1.5	233.0785	-1.5	233.0894	3.17
B3	246.9418	246.9380	-1.54	246.9380	-1.54	246.9502	3.4
C4	261.6256	261.6157	-3.78	261.6431	6.69	261.6294	1.45
C#4	277.1826	277.1619	-7.47	277.1926	3.61	277.1926	3.61
D4	293.6650	293.7030	12.94	293.6685	1.19	293.6685	1.19
D#4	311.1270	311.1000	-8.68	311.1388	3.79	311.1388	3.79
E4	329.6276	329.6414	4.19	329.6414	4.19	329.6414	4.19
F4	349.2286	349.2596	8.88	349.2596	8.88	349.2352	1.89
F#4	369.9946	370.0414	12.65	369.9867	-2.14	370.0141	5.27
G4	391.9956	391.9724	-5.92	392.0339	9.77	392.0031	1.91
G#4	415.3050	415.2824	-5.44	415.2824	-5.44	415.3169	2.87
A4	440.0000	439.9859	-3.2	439.9859	-3.2	440.0246	5.59
A#4	466.1640	466.2439	17.14	466.1570	-1.5	466.1570	-1.5
B4	493.8836	493.8759	-1.56	493.8759	-1.56	493.8759	-1.56
C5	523.2512	523.2315	-3.76	523.2315	-3.76	523.2862	6.69
C#5	554.3652	554.3237	-7.49	554.3237	-7.49	554.3852	3.61
D5	587.3300	587.4060	12.94	587.4060	12.94	587.3370	1.19
D#5	622.2540	622.2001	-8.66	622.2001	-8.66	622.2775	3.78
E5	659.2552	659.2827	4.17	659.2827	4.17	659.2827	4.17
F5	698.4572	698.3240	-19.07	698.5191	8.86	698.5191	8.86
F#5	739.9892	740.0829	12.66	740.0829	12.66	739.9734	-2.14
G5	783.9912	784.1907	25.45	783.9448	-5.92	784.0677	9.76
G#5	830.6100	830.5648	-5.44	830.5648	-5.44	830.5648	-5.44
A5	880.0000	880.2817	32.01	879.9718	-3.2	879.9718	-3.2
A#5	932.3280	932.1402	-20.14	932.4879	17.15	932.3140	-1.5
B5	987.7672	988.1423	37.97	987.7519	-1.55	987.7519	-1.55
C6	1046.5024	1046.9012	38.11	1046.463	-3.76	1046.4630	-3.76
C#6	1108.7304	1109.1393	36.88	1108.6475	-7.48	1108.6475	-7.48
D6	1174.6600	1174.8100	12.94	1174.8120	12.94	1174.8120	12.94
D#6	1244.5080	1245.0199	41.13	1244.4002	-8.66	1244.4002	-8.66
E6	1318.5104	1318.5654	4.17	1318.5654	4.17	1318.5654	4.17
F6	1396.9144	1396.6480	-19.07	1396.6480	-19.07	1397.0383	8.87
F#6	1479.9784	1479.2899	-46.52	1480.1658	12.66	1480.1658	12.66
G6	1567.9824	1568.3814	25.45	1568.3814	25.45	1567.8896	-5.92
G#6	1661.2200	1662.2340	61.04	1661.1296	-5.44	1661.1296	-5.44
A6	1760.0000	1760.5634	32.01	1760.5634	32.01	1759.9437	-3.2
A#6	1864.6560	1865.6716	54.47	1864.2804	-20.14	1864.9758	17.15
B6	1975.5344	1974.7235	-41.05	1976.2846	37.97	1975.5038	-1.55
C7	2093.0048	2093.8023	38.1	2093.8023	38.1	2092.9259	-3.77
C#7	2217.4608	2216.3121	-51.8	2218.2786	36.88	2217.2949	-7.48
D7	2349.3200	2349.6241	12.94	2349.6241	12.94	2349.6241	12.94
D#7	2489.0160	2490.0398	41.13	2490.0398	41.13	2488.8004	-8.66
E7	2637.0208	2637.1308	4.17	2637.1308	4.17	2637.1308	4.17
F7	2793.8288	2796.4206	92.77	2793.2961	-19.07	2793.2961	-19.07
F#7	2959.9568	2962.0853	71.91	2958.5799	-46.52	2960.3316	12.66
G7	3135.9648	3132.8321	-99.9	3136.7629	25.45	3136.7629	25.45
G#7	3322.4400	3324.4681	61.04	3324.4681	61.04	3322.2591	-5.44
A7	3520.0000	3521.1268	32.01	3521.1268	32.01	3521.1268	32.01
A#7	3729.3120	3731.3433	54.47	3731.3433	54.47	3728.5608	-20.14
B7	3951.0688	3955.6962	117.12	3949.4471	-41.04	3952.5692	37.97
C8	4186.0096	4180.6020	-129.1	4187.6047	38.11	4187.6047	38.11

3.2 Creating a User Program and Linking a MELODY33 Library

On the E0C33 chip, a melody can be played by calls to MELODY33 library functions. For more information on MELODY33 library functions and program examples, see Section 5, "MELODY33 Library Reference".

Include the created melody ROM data and scale table data sources into the user program, or link them along with the MELODY33 library after assembly.
The MELODY33 library to be linked is "mdy104.lib" for the E0C33A104, and "mdy208.lib" for the E0C33208.

Refer to the Appendix for a description of the procedure for executing the sample program using the DMT33004 and DMT33AMP boards.

4 MELODY33 Tool Reference

This section describes the functions of each MELODY33 tool and explains how to use them.

4.1 Outline of MELODY33 Tools

MELODY33 tools are PC software applications for creating and evaluating the melody ROM data to be written to E0C33 Family chips. All the tools run under Windows 95, Windows NT 4.0, or later Windows versions. For more information on the operating environment, see Section 2.1, "Operating Environment".
All MELODY33 tools and related files are found in the "mdytool" folder (directory).
The configuration of MELODY33 tools and the procedure for creating melody ROM data are shown in Figure 4.1.1.

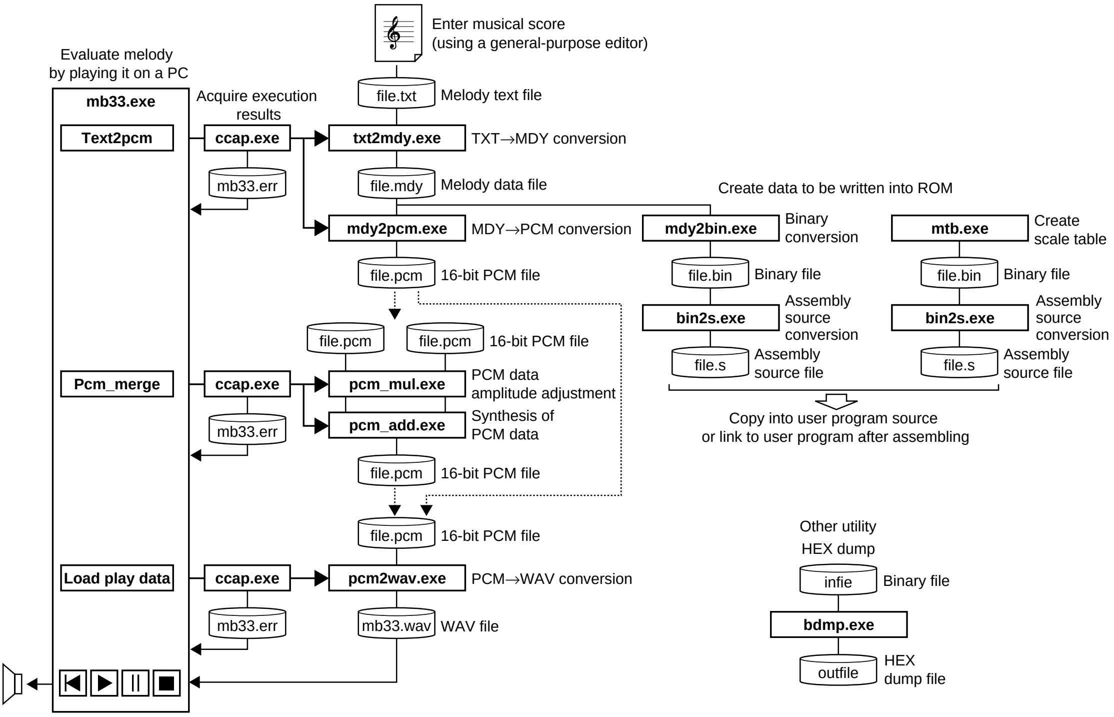


Figure 4.1.1 Flow Chart for Creating Melody ROM Data

Note: The PCM file formats handled by MELODY33 tools include 44.1 kHz, 16-bit monaural, and little-endian format.

List of MELODY33 tools

MELODY33 tools consist of a series of programs that generate PCM files or E0C33 assembly source files from melody text files. All of these programs are 32-bit applications that can be executed from the DOS prompt. They can also be used from a batch file or make file. The melody bench mb33 is a 32-bit Windows GUI application, capable of converting sound text files to PCM files and playing sound on a PC in a single window.

Table 4.1.1 lists the MELODY33 tools.

Table 4.1.1 List of Melody ROM Data Creation Tools

Tool	Function
mb33.exe	A Windows GUI application for creating melody data and playing it for evaluation.
txt2mdy.exe	Converts a melody text file into a melody data file.
mdy2bin.exe	Converts a melody data file (text format) into a binary file.
mdy2pcm.exe	Converts a melody data file (text format) into a PCM file.
mtb.exe	Creates scale table data.
bin2s.exe	Converts a binary data file into an assembly source file.
bdmp.exe	Dumps a binary data file in hexadecimal form.
pcm2wav.exe	Converts a PCM file into a WAV file.
pcm_add.exe	Synthesizes two PCM data.
pcm_mul.exe	Adjusts the amplitude of PCM data.
ccap.exe	Prepares in a file messages output by other tools during execution. For more information on this tool, refer to the "E0C33 Family C Compiler Package Manual".

* Unless otherwise specified, the PCM data format handled by MELODY33 tools is as follows:
44.1 kHz-sampling, 16-bit little-endian format row data

4.2 Explanation of Each MELODY33 Tool

This section describes the function of each MELODY33 tool and explains how to use them. For the "mb33.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 txt2mdy.exe

Function: Converts a melody text file into a melody data file. At this step in the process, the data necessary to play a melody by separating each note is added according to specified parameter values. The parameters specify the length of a rest to be inserted between notes for note from note separate playback. This parameter has one of the following four settings:

0: None
1: 1/4 the length of a thirty-second note
2: 1/2 the length of a thirty-second note
3: Equal to the length of a thirty-second note

Selecting 1 or 2 reduces the duration of each note by the corresponding proportion, but the length of the entire piece remains the same.

Selecting 3 inserts a thirty-second rest between each note, thereby lengthening the overall duration of the music.

When you select 0, no break is inserted, and the melody is played from beginning to end without modifications, one note after another.

Format: DOS>**txt2mdy** ***silent infile.txt outfile.mdy***↵

Parameters: ***silent*** Length of break between notes
0: None
1: 1/4 the length of a thirty-second note
2: 1/2 the length of a thirty-second note
3: Equal to the length of a thirty-second note

infile.txt Input file name (melody text file)

outfile.mdy Output file name (melody data file)

Example:

```
DOS>txt2mdy 1 spring.txt spring.mdy
```

Reference: For more information on melody text file formats and creating them, see Section 3.1.1, "Creating Melody Text Files".

The melody data file is created as a text format file. Sample file content is shown below.

Example: spring.mdy

```
(1)  (2)      (3)               (4)
4    25      //24              8    do5
1    62
4    29      //48              8    mi5
1    62
4    29      //72              8    mi5
1    62
4    29      //96              8    mi5
1    62
2    27      //108             16   re5
1    62
2    25      //120             16   do5
```

```
1    62
8    32      //168          4    so5  c
4    32      //192          8    so5
1    62
2    32      //204          16   so5
1    62
                 :
4    24      //552          8    si4
1    62
4    20      //576          8    so4
1    62
8    63      //624          4   -1
1    62
```

(1) Indicates the duration of the tone by one of the values given below.
1: Thirty-second note 2: Sixteenth note 4: Eighth note 8: Quarter note
Half notes and whole notes are represented by multiple quarter notes.

However, a parameter (2) of 62 indicates a break (specified by *silent*) used to separate each note, and the meaning of the value specified here changes as shown below.
1: Rest equal to 1/4 of a thirty-second note 2: Rest equal to 1/2 of a thirty-second note

(2) Indicates a musical interval.
The lowest frequency in the scale table is 1, and the highest is 61. Normally, 1 = do3/C3 (131 Hz) while 61 = do8/C8 (4,186 Hz).
For a value of 62, this parameter indicates a break to separate each note, so that the duration of the tone immediately preceding this line is reduced by an amount equal to the length of the break. The break is not inserted after notes for which the letter 'c' is specified in the melody text file.
If the value is 63, the parameter represents a rest. The duration of the tone written in (1) is the same as for notes.

(3) Indicates the period of time from the point at which the system starts playing to the completion of the note on the line. This is equal to the numeric value in (1) multiplied by 6. If the specified parameter (2) is 62, the duration of the note immediately preceding includes the break time.

(4) These are the contents of the melody text file.

4.2.2 mdy2bin.exe

Function: Converts a melody data file (text format) into a binary file. This is required when converting melody data into an assembly source file.

Format: `DOS>`**`mdy2bin`** ***infile.mdy outfile.bin***↵

Parameters: ***infile.mdy*** Input file name (melody data file)

outfile.bin Output file name (melody binary file)

Example:

```
DOS>mdy2bin spring.mdy spring.bin
```

Reference: The "mdy2bin.exe" converts the values found in the first row (tone duration) and second row (musical interval) on each line of the melody data file into binary quantities. Each line is converted into one-byte data, with the two high-order bits representing tone duration and the six lower-order bits representing the musical interval. The binary file is ended by adding 0 as a terminating symbol.

4.2.3 mtb.exe

Function: Creates a scale table corresponding to the operating frequency of the 16-bit timer on the E0C33 chip.

Format: `DOS>`**`bin2s`** ***16tmClk outfile.bin***↵

Parameters:

16tmClk 16-bit timer input clock frequency used for PWM output (0.5 to 8.0 MHz)

outfile.bin Output file name (scale table file)

Example:

1) For the E0C33A104, set 1/8 of the CPU operating frequency for *16tmClk.* If the CPU operates at 20 MHz, the parameter set is 2.5; for 33 MHz, the parameter is 4.125.

```
DOS>mtb 2.5 mTable25.bin
```

2) For the E0C33208, set 1/16 of the CPU operating frequency for *16tmClk.* If the CPU operates at 40 MHz, the parameter set is 2.5; for 20 MHz, the parameter is 1.25.

```
DOS>mtb 1.25 mTable12.bin
```

For standard specifications 1 and 2, a scale table is created in do3/C3 (131 Hz) to do8/C8 (4,186 Hz). By changing the specified value, you can create a scale table in which the scale range is shifted one octave. This is useful in creating a scale table to play scales below do3/C3 or above do8/C8.

3) For the E0C33A104 operating at 20 MHz (normally *16tmClk* = 2.5)

```
DOS>mtb 5 mTable5.bin
```

A scale table is created in scales one octave lower from do2/C2 (65.5 Hz) to do7/C7 (2,093 Hz).

```
DOS>mtb 1.25 mTable12.bin
```

A scale table is created in scales one octave higher from do4/C4 (262 Hz) to do9/C9 (2,093 Hz).

Note: Make sure that *16tmClk* is within the range 0.5 to 8.0. In frequencies outside this range, tones may be reproduced out of tune or may not be reproduced at all.

4.2.4 bin2s.exe

Function: Converts a binary file (files created by "mdy2bin.exe" or "mtb.exe") into a text file in E0C33 assembly source format. Since results are output to the standard output device (stdout), use the DOS redirect function when saving them to a file.

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

Parameters:

`-l` ***symbol*** Defines an assembler symbol name (option).
When this option is omitted, the input file name is used as a symbol name.

infile.bin Input file name (binary file)

outfile.s Output file name (assembly source file)

Example:

1) When the -l option is omitted, the input file name becomes a symbol name for the assembler.

```
DOS>bin2s spring.bin > spring.s
DOS>type spring.s
        .global spring
        .align 2
spring:
        .byte    0x02 0x33 0x99 0x3e 0x9d 0x3e 0x9d 0x3e
        .byte    0x9d 0x3e 0x5b 0x3e 0x59 0x3e 0xe0 0xa0
                                         :
DOS>
```

2) To use a symbol that differs from the input file name, specify the -l option.

```
DOS>bin2s -l mdy01 spring.bin > mdy01.s
DOS>type mdy01.s
        .global mdy01
        .align 2
mdy01:
        .byte    0x02 0x33 0x99 0x3e 0x9d 0x3e 0x9d 0x3e
        .byte    0x9d 0x3e 0x5b 0x3e 0x59 0x3e 0xe0 0xa0
                                         :
DOS>
```

Note: Symbol names are subject to the following limitations.

• Symbol length:	Up to 32 characters
• Usable characters:	a to z, A to Z, 0 to 9, _

4.2.5 bdmp.exe

Function: Dumps an input binary file in a specified format. Since results are output to the standard output device (stdout), use the DOS redirect function when saving them to a file.

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

Parameters:

option Specifies the output format (may not be omitted).
Use the following switches to specify 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 02 33 99 3E 9D 3E 9D 3E 9D 3E 5B 3E 59 3E E0 A0
00000010 3E 60 3E 5E 3E 9D 3E 9D 3E 9D 3E 5B 3E 59 3E E0
00000020 A0 3E 60 3E 5E 3E 9D 3E 5E 3E 60 3E 9E 3E 9D 3E
00000030 9B 3E 98 3E 94 3E FF 3E 00
```

```
DOS>bdmp -l spring.bin
00000000 3302 3E99 3E9D 3E9D 3E9D 3E5B 3E59 A0E0
00000010 603E 5E3E 9D3E 9D3E 9D3E 5B3E 593E E03E
00000020 3EA0 3E60 3E5E 3E9D 3E5E 3E60 3E9E 3E9D
00000030 3E9B 3E98 3E94 3EFF
```

```
DOS>bdmp -m spring.bin
00000000 0233 993E 9D3E 9D3E 9D3E 5B3E 593E E0A0
00000010 3E60 3E5E 3E9D 3E9D 3E9D 3E5B 3E59 3EE0
00000020 A03E 603E 5E3E 9D3E 5E3E 603E 9E3E 9D3E
00000030 9B3E 983E 943E FF3E
```

4.2.6 mdy2pcm.exe

Function: Converts a melody data file into a PCM file. Offset values to shift the tempo or interval may be specified.
Use this tool to create data for evaluation for PC playback. It is not intended for use in creating melody data that is to be actually written to the E0C33 chip.

Format: `DOS>`**`mdy2pcm`** ***tempo offset infile.mdy outfile.pcm***↵

Parameters:

tempo	Tempo (30 to 300)
offset	Amount of interval shifted (-24 to 24, in semitone increments)
infile.mdy	Input file name (melody data file)
outfile.pcm	Output file name (PCM file)

Example:

```
DOS>mdy2pcm 100 12 test.mdy test.pcm
```

A PCM file is created in which the melody "test.mdy" is played at the tempo = 100, with the interval raised one octave.

Reference: The file created here is a PCM file in 44.1 kHz, 16-bit little-endian format. This file can be reproduced after loading it with mb33.

4.2.7 pcm2wav.exe

Function: Converts a PCM file into a WAV file in the same frequency.
Use this tool to create data for evaluation for PC playback. It is not intended for use in creating melody data that is to be actually written to the E0C33 chip.

Format: `DOS>`**`pcm2wav`** [***SamplingRate***] ***infile.pcm outfile.wav***↵

Parameters:

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

Example:

```
DOS>pcm2wav 44.1 sample1.pcm sample1.wav
```

A 44.1 kHz-sampling PCM file is converted into a WAV file.

4.2.8 pcm_add.exe

Function: Synthesizes two PCM files by combining data. Following the combination, if the amplitude exceeds the range of 16-bit PCM data, portions above 0x7fff(+) or below 0x8000(-) are clipped. Since simply synthesizing two files with this tool will produce an overly large amplitude, before synthesizing files with this tool, adjust the amplitude of PCM files to be synthesized to lower levels (e.g. 50%), using "pcm_mul.exe".
Use this tool to create data for evaluation for PC playback. It is not intended for use in creating melody data that is to be actually written to the E0C33 chip.

Format: `DOS>`**`pcm_add`** ***infile1.pcm infile2.pcm outfile.pcm***↵

Parameters:

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

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

Example:

```
DOS>pcm_add spring.pcm spring2.pcm spring_spring2.pcm
```

Files "spring.pcm" and "spring2.pcm" are synthesized to create spring_spring2.pcm.

4.2.9 pcm_mul.exe

Function: Adjusts the amplitude of PCM data to a specified multiple. If the amplitude as a result of multiplication exceeds the range of 16-bit PCM data, portions above 0x7fff(+) or below 0x8000(-) are clipped.
Use this tool to create data for evaluation for PC playback. It is not intended for use in creating melody data that is to be actually written to the E0C33 chip.

Format: `DOS>`**`pcm_mul`** ***value infile.pcm outfile.pcm***↵

Parameters:

value Specified multiple (decimal, 0.0 or greater)

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

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

Example:

```
DOS>pcm_mul 1.3 sample.pcm sample13.pcm
```

The sample.pcm amplitude is multiplied by a factor of 1.3 to create sample13.pcm.

4.2.10 Executing from a Batch File

Since all the melody ROM data creation tools described above are 32-bit applications that can be run from a DOS prompt, you can create a batch file to execute a series of processes.

The following illustrates an example of processing executed using the batch files "mdata1.bat" and "mt.bat," found in the "mdytool\sample\" directory.

Each file was created assuming that "mdytool\sample\" is the current directory, and that the melody ROM data creation tools in "mdytool\bin\" are executed. If necessary, amend these files before use.

mdata1.bat

Converts sample melody text files "spring.txt" and "all.txt" to create the assembly source file "mdata1.s" containing both data sets.

Process:
1) Converts a melody text file into a melody data file.
2) Converts the melody data file into a binary file.
3) Converts the binary file into an assembly source and output the result to a file.

File contents:

```
..\bin\txt2mdy 1 spring.txt spring.mdy
..\bin\txt2mdy 1 all.txt all.mdy
..\bin\mdy2bin spring.mdy spring.bin
..\bin\mdy2bin all.mdy all.bin
..\bin\bin2s -l spring spring.bin > mdata1.s
..\bin\bin2s -l all all.bin >> mdata1.s
```

Example:

```
>mdata1
```

The global labels specified by the -l option of "bin2s.exe" are defined in each data.

```
    .global  spring
    .align   2
spring:
    .byte    0x02 0x33 0x99 0x3e 0x9d 0x3e 0x9d 0x3e
    .byte    0x9d 0x3e 0x5b 0x3e 0x59 0x3e 0xe0 0xa0
                               :
    .byte    0x00
; total 57 bytes data

    .global  all
    .align   2
all:
    .byte    0x02 0x33 0x81 0x3e 0x83 0x3e 0x85 0x3e
    .byte    0x86 0x3e 0x88 0x3e 0x8a 0x3e 0x8c 0x3e
                               :
    .byte    0x00
; total 97 bytes data
```

Reference: "4.2.1 txt2mdy.exe", "4.2.2 mdy2bin.exe", "4.2.4 bin2s.exe"

mt.bat

Creates a scale table for cases in which the CPU runs at an operating frequency of 20 MHz. In this example batch file, two types of scale tables are created, "mTable25" for the E0C33A104 and "mTable12" for the E0C33208.

Process: Creates a scale table using "mtb.exe"

File contents:

```
..\bin\mtb 2.5 mtb25.bin
..\bin\bin2s -l mTable25 mtb25.bin > mtable.s
..\bin\mtb 1.25 mtb125.bin
..\bin\bin2s -l mTable12 mtb125.bin >> mtable.s
```

Example:

```
>mt
```

"mTable25" is a scale table designed for use with the E0C33A104 operating at 20 MHz, in which 2.5 (= 20/8) is specified for the "mtb.exe" parameter. "mTable12" is a scale table designed for use with the E0C33208 operating at 20 MHz, in which 1.25 (= 20/16) is specified for the "mtb.exe" parameter. Normally, you would not create scale tables for two or more types of microcomputers in a single file, but you may wish to prepare multiple scale tables to play a melody by shifting intervals. This example batch file may prove useful in such cases. For example, when creating scale tables for the E0C33208 (20 MHz), you can use "mTable12" as a standard scale table and "mTable25" as an octave-lower scale table.

```
    .global mTable25
    .align  2
mTable25:
    .byte   0x01 0x33 0xa7 0x4a 0x76 0x46 0x82 0x42
    .byte   0xc6 0x3e 0x40 0x3b 0xed 0x37 0xc9 0x34
                            :
    .byte   0x79 0x02 0x55 0x02
; total 124 bytes data

    .global mTable12
    .align  2
mTable12:
    .byte   0x01 0x33 0x53 0x25 0x3b 0x23 0x41 0x21
    .byte   0x63 0x1f 0xa0 0x1d 0xf6 0x1b 0x65 0x1a
                            :
    .byte   0x3c 0x01 0x2b 0x01
; total 124 bytes data
```

Reference: "4.2.3 mtb.exe", "4.2.4 bin2s.exe"

4.3 Melody Bench mb33

Melody Bench mb33 is a Windows GUI tool that converts melody text files into the necessary format for evaluative playback on a PC.

4.3.1 Starting and Quitting

Double-click the "mb33.exe" icon to start.

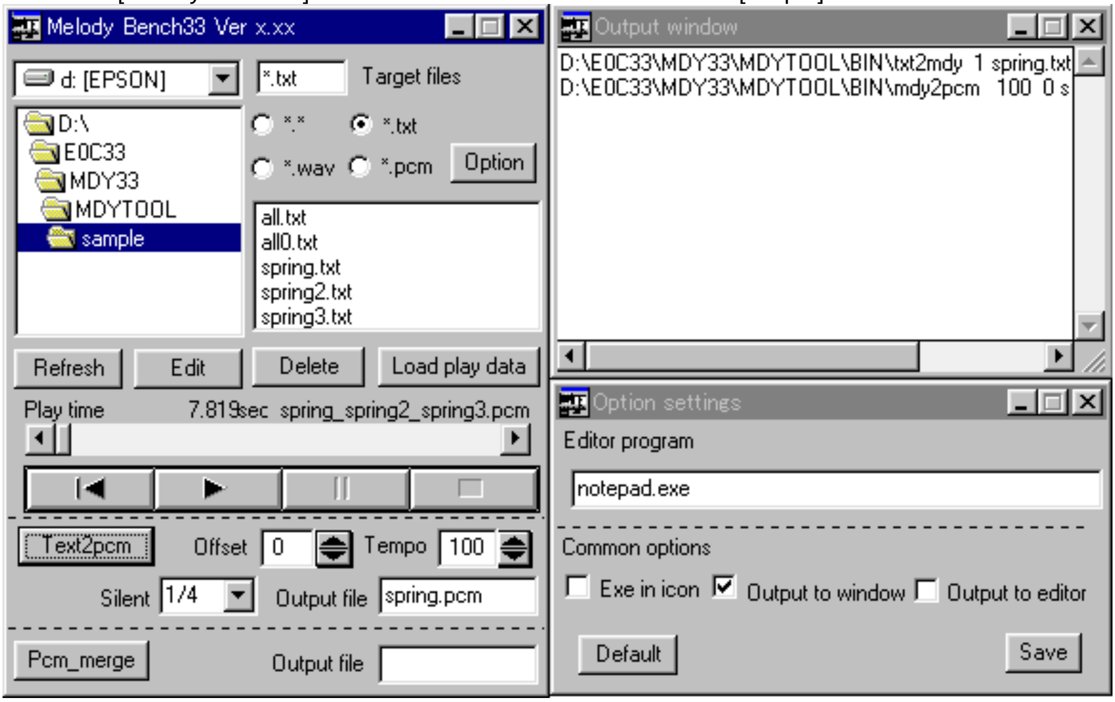
To quit mb33, click the [Close] button at the upper right corner of the [Melody Bench33] window.

4.3.2 Window Structure

"mb33.exe" consists of the three windows shown below.

[Melody Bench33] window

[Output] window



Drive: d: [EPSON]

*.txt Target files

D:\ E0C33 MDY33 MDYTOOL sample

○ *.* ● *.txt ○ *.wav ○ *.pcm

all.txt
all0.txt
spring.txt
spring2.txt
spring3.txt

Play time 7.819sec spring_spring2_spring3.pcm

Offset 0 Tempo 100

Silent 1/4 Output file spring.pcm

Output file

```
D:\E0C33\MDY33\MDYTOOL\BIN\txt2mdy  1 spring.txt
D:\E0C33\MDY33\MDYTOOL\BIN\mdy2pcm  100 0 s
```

Editor program notepad.exe

□ Exe in icon ☑ Output to window □ Output to editor

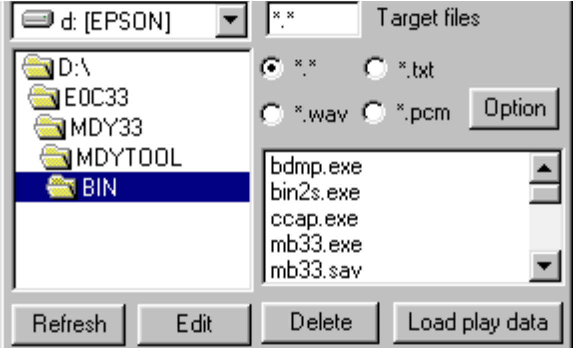
[Option settings] window

[Melody Bench33] window
This window appears when "mb33.exe" starts. All operations required to play a melody are performed in this window.

[Output] window
Displays execution commands or results (output messages) of the tools called to convert data.
This window opens automatically when you run a tool.
However, this window must be set in advance from the [Option] window to be the output destination for execution results.

[Option settings] window
Used to select the editor to use or an execution option. Open this window by clicking the [Melody Bench33] window [Option] button.

4.3.3 Selecting a File



To choose a file to convert or play, use the directory list and file list box of the [Melody Bench33] window.
Use the radio buttons to select the file type to display in the file list box.

[Refresh] button

The contents displayed in the file list box are not automatically updated when files are added or deleted by a tool other than mb33. Click the [Refresh] button to update the list.

[Edit] button

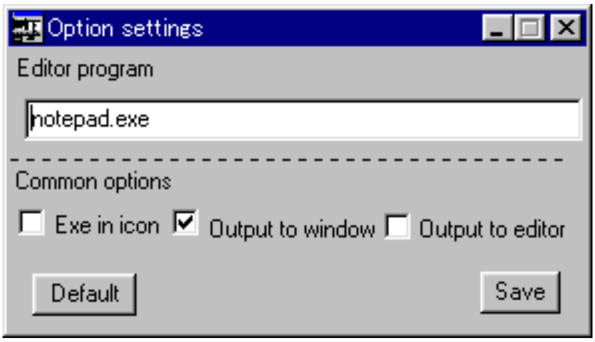
Choose a text file from the file list box and click the [Edit] button. The editor starts up, and the selected file opens. Although Windows Notepad is set as the default editor, this may be changed to any other editor from the [Option settings] window.

[Delete] button

Deletes a file currently selected in the file list box.

4.3.4 Selecting an Option

Click the [Option] button to bring up the [Option settings] window.



Option settings

Editor program

notepad.exe

Common options

□ Exe in icon ☑ Output to window □ Output to editor

[Editor program] text box

Use this text box to specify the editor you want to start by clicking on the [Edit] button. Include the absolute path when you enter the editor name.

[Exe in icon] check box

Executes the tool in an icon state started from mb33.

[Output to window] check box

Displays the startup command or output message of a tool in the [Output] window. The melody bench mb33 writes the startup command and output messages of each tool to a file named "mb33.err", using "ccap.exe" (see the E0C33 Family C Compiler Package Manual). The contents of an "mb33.err" are displayed in the [Output] window.

[Output to editor] check box

Displays the startup command or output message of a tool on a specified editor after starting it. As in the case of [Output to window], the editor opens the "mb33.err" file.

[Default] button

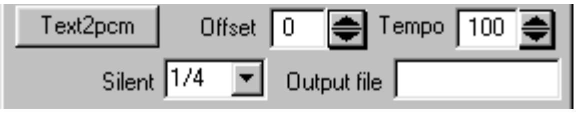
Restores the setup contents of options to their default state.
Editor program: Notepad (notepad.exe)
Common options: [Output to window] is selected

[Save] button

Saves the setup contents of options to a file (mb33.sav), including editor specifications. When mb33 starts the next time, the set options are reloaded.

4.3.5 Converting Melody Text Files

The melody bench mb33 allows you to convert a melody text file into a PCM file in one step. The procedure is given below.



Offset 0

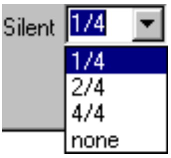
Tempo 100

Silent 1/4

Conversion operation part

1) From the file list box, choose a melody text file to convert. If you choose multiple files, only the file listed at the top of those selected is effective.

2) Choose the options [Silent], [Offset], and [Tempo] and change the output file name in [Output file] as necessary. The default output file name is the same as the selected melody text file, with its extension changed to ".pcm".



[Silent]
Selects the length of a break inserted to play music by separating each note.

1/4	1/4 the length of a thirty-second note
2/4	1/2 the length of a thirty-second note
4/4	Equal to the length of a thirty-second note
none	None

When you choose *1/4* or *2/4*, the time during which each note is generated is reduced by that fraction, without changing the length of the overall music.
When you choose *4/4*, a thirty-second rest is inserted between each note, and the length of music is extended.
When you choose *none*, no break is inserted, and the music is played one note after another from beginning to end.

Offset 0

[Offset]
Shifts musical intervals in semitone increments. This can be specified over a range of -24 to +24 (±2 octaves). Generating sounds outside the range do3/C3 through do8/C8 on the chip requires a scale table created for that purpose.

Tempo 100

[Tempo]
Specifies the tempo at which to play music, using a value from 30 to 300. This value represents the number of quarter notes to be played per minute.
The following shows typical words representing tempos and the approximate values set for Tempo.

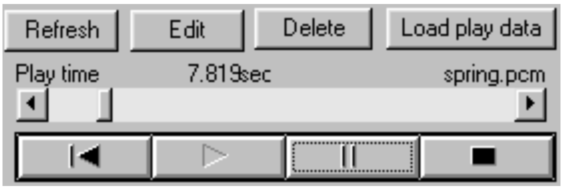
Name of tempo	Tempo range	Approximate value
Lento, Largo, Grave	30–50	40
Adagio	40–60	50
Larghetto	50–60	55
Adagietto	60–70	65
Andante	65–75	70
Andantino	70–80	80
Moderato	80–90	90
Allegretto	90–100	100
Allegro, Conmoto	100–130	120
Vivo	110–140	130
Vivace	140–170	150
Presto	170–200	180
Prestissimo	200–	200

[Silent] determines the "txt2mdy.exe" parameter, and [Offset] and [Tempo] determine the "mdy2pcm.exe" parameters.

3) Click the [Text2pcm] button.
The melody bench mb33 executes "txt2mdy.exe" to convert the selected melody text file into a melody data file (.mdy). Next, it executes "mdy2pcm.exe" to create a PCM file from the melody data file.

4.3.6 Playing a Melody

The melody bench mb33 can reproduce PCM or WAV files. The procedure is given below.



Play time 7.819sec spring.pcm

Playback operation part

1) From the file list box, choose a PCM or WAV file to reproduce.

2) Click the [Load play data] button.
When a PCM file is selected, it is first converted into a WAV file "mb33.wav" by "pcm2wav.exe" before being loaded. The name of the selected file and the play time are displayed in [Play time].

3) Click the [Play] button.

[Play] button

The following describes the functions of the buttons used to control melody playback.

[Beginning] button: Returns to the beginning of a melody.

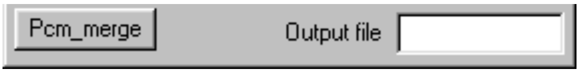
[Pause] button: Temporarily stops playback. Use the [Play] button to restart from the position at which playback stopped.

[Stop] button: Stops playback and returns to the beginning of the melody.

Note: Due to inadequate resolution on PC playback, the tones do7/C7 (2,093 Hz) to do8/C8 (4,186 Hz) may be out of tune.

4.3.7 Synthesizing Melodies

The MELODY33 library supports melody output in up to four channels (up to three channels for the E0C33A104). To play multiple channels for evaluation on a PC, the data must be synthesized into one PCM file beforehand.
The MELODY33 tools include "pcm_add.exe", which synthesizes two PCM files. This tool can also be executed from mb33.
The procedure is given below.



Melody synthesizing operation part

1) Use the [Text2pcm] button to convert all the melody text files to be synthesized into PCM files. Make sure all options are set identically.

2) Choose two PCM files to synthesize from the file list box. To choose two files, choose one file, then hold down the [Ctrl] key while you select another.

3) Change the output file name in the [Output file] text box as necessary.

4) Click the [Pcm_merge] button. A PCM file is generated in which the two data sets have been synthesized.

5) Enter the synthesized PCM file using the [Load play data] button, then click the [Play] button to play the file.

Note that simply adding two files together by using "pcm_add.exe" results in excessive amplitude. Before synthesizing files, mb33 calls the amplitude adjusting tool "pcm_mul.exe" to adjust the amplitude of each PCM data to 50%.

To synthesize four channels, follow the procedure given below.
Example: Synthesizing "ch1.pcm", "ch2.pcm", "ch3.pcm", and "ch4.pcm"

1. Synthesize "ch1.pcm" and "ch2.pcm" (to create "ch1_ch2.pcm").
2. Synthesize "ch3.pcm" and "ch4.pcm" (to create "ch3_ch4.pcm").
3. Synthesize the two files ("ch1_ch2.pcm" and "ch3_ch4.pcm") created in 1 and 2.

A file "ch1_ch2_ch3_ch4.pcm" is generated in which the four files have been synthesized.

Synthesizing three channels requires caution.
Example: Synthesizing "ch1.pcm", "ch2.pcm", and "ch3.pcm"
If the file derived by synthesizing "ch1.pcm" and "ch2.pcm" is synthesized directly with "ch3.pcm", the sound volume in "ch3.pcm" increases two-fold relative to the other. Follow the procedure given below when you synthesize three files.

1. Create a PCM file with size 0 (e.g. "zero.pcm").
   Use a blank text file ("zero.txt"), changing its extension to ".pcm".
2. Synthesize "ch1.pcm" and "ch2.pcm" (to create "ch1_ch2.pcm").
3. Synthesize "ch3.pcm" and "zero.pcm" (to create "ch3_zero.pcm").
   This will reduce the sound volume of "ch3.pcm" to half.
4. Synthesize the two files ("ch1_ch2.pcm" and "ch3_zero.pcm") created in 2 and 3.

A file "ch1_ch2_ch3_zero.pcm" is generated in which all files have been synthesized.

4.3.8 Work after Finishing Evaluation

A melody data file (.mdy) is created by "txt2mdy.exe" when you execute it by clicking the [Text2pcm] button. Since melody data file to binary conversion (mdy2bin.exe) and binary file to assembly source conversion (bin2s.exe) are not executed in mb33, you need to run each tool to create the assembly source file after exiting mb33.

5 MELODY33 Library Reference

This section gives some precautions for using MELODY33 library functions and explains each function in detail.

5.1 Outline of the MELODY33 Library

Functional outline

The MELODY33 library consists of a set of melody output functions in srf33 library format, and is linked to the target program when used. The library controls a 16-bit timer to output PWM data. The data can be output simultaneously in up to four channels (up to three channels for the E0C33A104).

Figure 5.1.1 shows the structure of the applications used to output a melody.

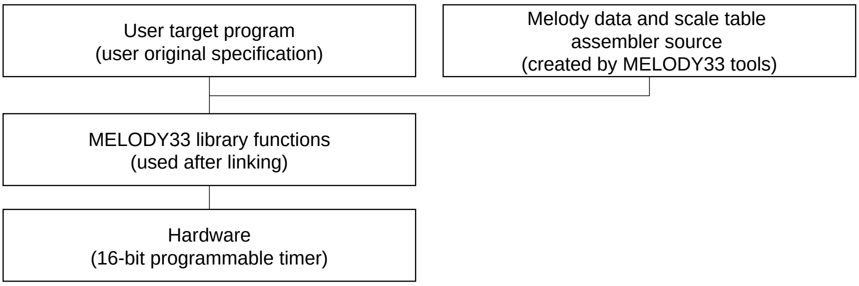

User target program
(user original specification)
Melody data and scale table
assembler source
(created by MELODY33 tools)
MELODY33 library functions
(used after linking)
Hardware
(16-bit programmable timer)

Figure 5.1.1 Program Structure

Configuration of the MELODY33 library

All MELODY33 library and related files are located in the "mdylib" folder (directory). The folder contents are given below:

mdylib\ ..... MELODY33 library-related

readme.txt	MELODY33 library supplementary explanation, etc. (in English)
readmeja.txt	MELODY33 library supplementary explanation, etc. (in Japanese)

lib\ ..... MELODY33 library directory

mdy104.lib	MELODY33 library for E0C33A104
mdy208.lib	MELODY33 library for E0C33208

include\ ..... MELODY33 library function header file directory

mdy.h	Library include file

libsrc\ ..... Hardware-dependent source directory

mdy104.c	Low-level functions for E0C33A104 control
mdy208.c	Low-level functions for E0C33208 control

demoX\ ..... Sample program directory

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

Table 5.1.1 lists MELODY33 library functions.

Table 5.1.1 MELODY33 Library Functions (mdy104.lib, mdy208.lib)

Function name	Description
mdyOpen( )	Starts melody output function.
mdyClose( )	Terminates melody output function.
mdyOnDone( )	Registers call-back function.
mdySet( )	Sets melody.
mdyStart( )	Starts playback.
mdyPause( )	Temporarily stops playback.
mdyReset( )	Resets the timer.
mdyStatus( )	Checks play status.
mdyIntOff( )	Disables 16-bit timer 4 interrupt.
mdyIntOn( )	Enables 16-bit timer 4 interrupt.
mdyInt( )	Processes 16-bit timer 4 interrupt.

Hardware resources

The MELODY33 library uses the internal hardware resources of the E0C33 chip listed below.

16-bit timers

The 16-bit timer 4 is always used to generate tempos. In addition, several other 16-bit timers among those shown below are used to output a melody.

For the E0C33A104:	16-bit timers 1, 3, and 5
For the E0C33208:	16-bit timers 0, 1, and 2

Memory

The following memory sizes are used:

ROM:	Approx. 1.7K bytes
RAM:	Approx. 60 bytes
Stack:	Approx. 140 bytes

Interrupt

An interrupt by 16-bit timer 4 is used.
For the E0C33A104, set the address of the mdyInt( ) function as the timer 41 underflow interrupt vector; for the E0C33208, set it as the timer 4 compare B interrupt vector.

CPU occupancy rate

From when mdyOpen( ) function is called until melody output is terminated by the mdyClose( ) function, a 16-bit timer 4 interrupt is generated at intervals equal to the length of a 32 thirty-second note divided by 8, according to the tempo set by mdyOpen( ) function (approximately every 19 ms when tempo = 100).
The time required for one instance of interrupt handling is shown below.
(When ROM and RAM = 1 wait state, with the internal RAM used for stack)

- At least 25 µs is required, even when no melody is output.
- During melody output (from mdySet( ) to end of playback), 25 µs is added for output in each channel.

Therefore, use of all four channels requires a time of up to 125 µs. The CPU occupancy rate during a 19 ms period is approximately 0.6%. This is the maximum value. The average value in actual use is smaller.

5.2 Description of Individual Functions

The following describes the MELODY33 library functions. For usage examples, see Section 5.3, "Program Example".

5.2.1 Constant Definition

The 16-bit timer channel numbers and the status/error codes returned by functions are defined in "include\mdy.h". Include this file in your source program.

Constants used for timer settings

MDY_CHANNELS	Defines the maximum number of channels that can be handled by MELODY33. (3)*
MDY_104_TM1	Represents 16-bit timer 1 of the E0C33A104. (0)
MDY_104_TM3	Represents 16-bit timer 3 of the E0C33A104. (1)
MDY_104_TM5	Represents 16-bit timer 5 of the E0C33A104. (2)
MDY_208_TM0	Represents 16-bit timer 0 of the E0C33208. (0)
MDY_208_TM1	Represents 16-bit timer 1 of the E0C33208. (1)
MDY_208_TM2	Represents 16-bit timer 2 of the E0C33208. (2)
MDY_208_TM3	Represents 16-bit timer 3 of the E0C33208. (3)

* To use four-channel output on the E0C33208, specify the -DCH4 option when compiling. This specification changes MDY_CHANNELS to (4).
Example: Specification in "demo208.mak"

```
GCC33_FLAG = -B$(TOOL_DIR)\ -S $(DEBUG) -O -I $(INCPATH) -DCH4
```

Status codes returned by functions

MDY_RESET	No melody data exists, or the system is in reset state. (0)
MDY_PAUSE	System temporarily stopped playback. (1)
MDY_START	System is performing playback. (2)
MDY_FINISH	System finished playback. (3)

Error codes returned by functions

MDY_OK	Normal (0)
MDY_INVALID_CHANNEL	Channels are incorrectly set. (-1)
MDY_INVALID_STATUS	Melody is incorrectly played. (-2)
MDY_INVALID_OFFSET	Offset is invalid. (-3)
MDY_INVALID_TABLE	Scale table is invalid. (-4)
MDY_INVALID_DATA	Melody data is invalid. (-5)

5.2.2 MDY_SAMPLING

Function: Acquires the set value of 16-bit timer 4 (macro).

Format: `MDY_SAMPLING(psc_f, tempo)`

Parameters: `psc_f` Prescaler input clock frequency
`tempo` Tempo at which to play a melody (30 to 300)

Return value: Preset value for 16-bit timer 4

Description: The 16-bit timer 4 generates an interrupt 32 times during a period equal to the length of a quarter note. The interrupt occurs at intervals equal to the duration of a thirty-second note divided by 4. This interrupt is used to change a note or control a break. The macro calculates the preset value to be set in 16-bit timer 4 based on the prescaler input clock and the tempo at which a melody is played. Use the following equation to find the value.

Set value of 16-bit timer 4 = ((psc_f / tempo) / 1,024) * 60) / 32 -1)

The prescaler's dividing ratio is set to 1/1,024.

Example: `mdyOpen(MDY_SAMPLING(20000000,100));`

This example calculates the set value of 16-bit timer 4 for cases when the prescaler input clock = 20 MHz and tempo = 100. MDY_SAMPLING is used as a parameter for the mdyOpen( ) function that causes 16-bit timer 4 to start counting.

5.2.3 mdyOpen( )

Function: Starts melody output.

Format: `void mdyOpen(int freq);`

Parameter: `int freq` Set value of 16-bit timer 4 obtained by MDY_SAMPLING

Return value: None

Description: This function causes 16-bit timer 4 to start counting after initialization.
Once the timer begins counting, it generates an interrupt at intervals equal to 1/4 the length of a thirty-second note.

Example: `mdyOpen(MDY_SAMPLING(20000000,100));`

The 16-bit timer 4 starts counting on the assumption that the prescaler input clock frequency is 20 MHz and the tempo is 100.

5.2.4 mdyClose( )

Function: Terminates melody output.

Format: `void mdyclose( );`

Parameter: None

Return value: None

Description: This function stops 16-bit timer 4, then resets it.
Always call this function to completely stop melody playback. Until this function is called, the 16-bit timer 4 interrupt is generated periodically, even when there is no data to play.

Example: `mdyclose( );`

The system terminates the entire melody playback operation.

5.2.5 mdyOnDone( )

Function: Registers a call-back function.

Format: `void mdyOnDone(void *func);`

Parameter: `void *func` Pointer to a call-back function

Return value: None

Description: This function registers a function that is called when the system finishes playing one melody. Because the registered call-back function is called from the interrupt routine in an interrupt-disabled state, it must be made as small as possible. The format of call-back functions is shown below. The timer number (0 to 2, see Section 5.2.1) in which the system finished playing a melody is passed as an argument to the called function.

```
static void CallBack(int tm_no);
```

Example:

```
mdyOnDone(&OnDone);
```

OnDone() is registered as a function to be called when the system finishes playing a melody.

An example of a call-back function is shown below.

```
volatile int flag;
static void OnDone(int i)
{
     flag = 1;               //set stop flag
}
        :
iErr = mdyStart(MDY_104_TM3);// start melody
flag = 0;
for (;;)
{
     if (flag == 1) break;
}
```

In this example, OnDone( ) is called when the system finishes playing a melody and the flag is simply set to 1.
The main routine enters a loop to check the flag after the system starts playback and exits the loop when the flag is set.

5.2.6 mdySet( )

Function: Sets a melody.

Format:

```
int mdySet(int channel, unsigned short *table, unsigned char *data,
int offset, int reverse);
```

Parameters:

`int`	`channel`	Timer to output a melody
`unsigned short`	`*table`	Pointer to scale table
`unsigned char`	`*data`	Pointer to melody data
`int`	`offset`	Interval shift (-24 to +24)
`int`	`reverse`	Inverted output flag (0: Normal output; 1: Inverted output) * For the E0C33208 only

Return value: Error code (see Section 5.2.1)

Description: This function sets the melody data to play and the 16-bit timer used to output PWM for that melody.
The specified 16-bit timer starts operations in a silent state.
The timers available for use for PWM output are timers 1, 3, and 5 for the E0C33A104, and timers 0, 1, 2, and 3 for the E0C33208.
Use the constants shown in Section 5.2.1 to specify the first parameter (channel).

To specify a scale table in the second parameter (table), use those created by "mtb.exe" and "bin2s.exe" (label defined by "bin2s.exe"). For more information on the scale table, see Section 4.2.3, "mtb.exe".

To specify melody data in the third parameter (data), use melody data created by "txt2mdy.exe", "mdy2bin.exe", and "bin2s.exe" (label defined by "bin2s.exe").

In the fourth parameter (offset), specify an offset value by which the musical interval is shifted in units of semitones when playing a melody. The musical interval can be shifted by up to ±2 octaves, using values from -24 to +24. If this produces results extending beyond the range of the scale table, the system assumes the minimum or maximum frequency of the scale table as it plays a melody.

The fifth parameter (reverse) is a flag used to directly drive a piezoelectric buzzer on the E0C33208. In this case, a timer is used for normal output (reverse = 0), while the other timer is used for inverted melody output (reverse = 1). When not directly driving a piezoelectric buzzer, always set 0 for this parameter. Output inversion is not supported for the E0C33A104, for which this parameter must be set to 0.

Example:

```
int iErr;
iErr = mdySet(MDY_208_TM0, &mTable25[0], &all[0], 0,1);
iErr = mdySet(MDY_208_TM1, &mTable25[0], &all[0], 0,0);
```

In this example, parameters are set to directly drive a piezoelectric buzzer using the E0C33208 16-bit timers 0 and 1. Both timers use the same melody data and same scale table, with timer 0 set for inverted output, and timer 1 set for normal output.

5.2.7 mdyStart( )

Function: Starts melody playback.

Format: `int mdyStart(int channel);`

Parameter: `int channel` Timer from which to output a melody

Return value: Error code (see Section 5.2.1)

Description: This function starts melody playback using the specified 16-bit timer.
Before a melody can be played, mdyOpen( ) and mdySet( ) must be executed.
The system is actually prompted to start playback by an interrupt generated by 16-bit timer 4 (generated at intervals equal to 1/4 the duration of a thirty-second note). For cases in which the system starts playback using multiple timers, if each timer starts playback on different interrupts, musical playback may fall out of synch. To avoid this problem, disable the timer interrupt before calling mdyStart( ). (See the following example.)

Example:

```
int iErr;
mdyIntOff();                  // Disable timer interrupt
iErr = mdyStart(MDY_208_TM0);
iErr = mdyStart(MDY_208_TM1);
mdyIntOn();                   // Enable timer interrupt
```

In this example, the system starts playback using 16-bit timers 0 and 1 of the E0C33208. To prevent falling out of sync, the timer interrupt is disabled by mdyIntOff( ) until after mdyStart( ) is called for the two timers.

5.2.8 mdyPause( )

Function: Temporarily stops playback.

Format: `int mdyPause(int channel);`

Parameter: `int channel` Timer by which to stop melody output

Return value: Error code (see Section 5.2.1)

Description: This function temporarily stops melody playback when played by the specified 16-bit timer.
To restart playback, call mdyStart( ).

Example:

```
int iErr;
iErr = mdyPause(MDY_208_TM0);
```

In this example, the system temporarily stops melody playback when played by 16-bit timer 0 of the E0C33208.

5.2.9 mdyReset( )

Function: Resets a timer.

Format: `int mdyReset(int channel);`

Parameters: `int channel` Timer to be reset

Return value: Error code (see Section 5.2.1)

Description: This function resets the specified 16-bit timer used for PWM output. You cannot restart play with mdyStart( ). Normally, this function is called before calling mdyClose( ) when you want to turn off all melody output.

Example:

```
int iErr;
iErr = mdyReset(MDY_208_TM0);
```

In this example, 16-bit timer 0 of the E0C33208 is halted.

5.2.10 mdyStatus( )

Function: Checks the status of melody playback.

Format: `int mdyStatus(int channel);`

Parameter: `int channel` Timer to be checked

Return value: Status code (see Section 5.2.1)

Description: This function returns the current status of the specified 16-bit timer being used for PWM output.

Example:

```
int iErr;
iErr = mdyStart(MDY_208_TM0);
for(;;){
            if(mdyStatus(MDY_208_TM0) == MDY_FINISH) break;
}
```

In this example, mdyStatus( ) is used to check whether the E0C33208 16-bit 0 timer has finished playing a melody.

5.2.11 mdyIntOff( )

Function: Disables 16-bit timer 4 interrupt.

Format: `void mdyIntOff( );`

Parameter: None

Return value: None

Description: This function disables the interrupt generated by 16-bit timer 4.

5.2.12 mdyIntOn( )

Function: Enables 16-bit timer 4 interrupt.

Format: `void mdyIntOn( );`

Parameter: None

Return value: None

Description: This function enables the interrupt generated by 16-bit timer 4.

5.2.13 mdyInt( )

Function: Processes the interrupt generated by 16-bit timer 4.

Format: `void mdyInt(void);`

Parameter: None

Return value: None

Description: This function sets PWM output data upon an interrupt generated by 16-bit timer 4 (generated at intervals equal to 1/4 the length of a thirty-second note). This function can only be used as an interrupt vector value.

Set the address of this function as a timer 41 underflow interrupt vector for the E0C33A104, or as a timer 4 compare B interrupt vector for the E0C33208.

5.3 Program Example

The following shows how to create a melody output routine, using the sample program in the demo3 directory as an example.

Interrupt vector and interrupt handling routine

Set the address of mdyInt( ) function as an interrupt vector for 16-bit timer 4.
This interrupt vector corresponds to the timer 41 underflow interrupt for the E0C33A104 and the timer 4 compare B interrupt for the E0C33208. In either case, the vector address is the trap table start address + 184 (decimal).

Example:
```
.word  mdyInt          ; Trap table base address + 184
```

The mdyInt() function is provided in the MELODY33 library. There is no need to create it.
In "vector.s" of demo3, only the NMI handler routine shown below is provided, so that all interrupts except for a reset and 16-bit timer 4 are made to jump to this routine. Before using "vector.s", modify it to suit your system.
In the example below, each time this routine is called (for each NMI input), operation alternates between melody playback and pausing.

Example: NMI routine in vector.s

```
         pushn    %r15
         .global ESC
ESC:
         xld.w    %r0, iPauseFlag
         ld.w     %r1, [%r0]
         cmp      %r1, 0                  ; check pause flag
         jreq     START                   ; if not 0, goto pause mode
PAUSE:
         ld.w     %r1, 0                  ; pause TM1(A104) or TM0(A208)
         ld.w     [%r0], %r1
         ld.w     %r12, 0
         xcall    mdyPause
         ld.w     %r1, 0                  ; pause TM3(A104) or TM1(A208)
         ld.w     [%r0], %r1
         ld.w     %r12, 1
         xcall    mdyPause
         ld.w     %r1, 0                  ; pause TM5(A104) or TM2(A208)
         ld.w     [%r0], %r1
         ld.w     %r12, 2
         xcall    mdyPause
         jp       BACK
START:
         ld.w     %r1, 1                  ; start TM1(104) or TM0(208)
         ld.w     [%r0], %r1
         ld.w     %r12, 0
         xcall    mdyStart
         ld.w     %r1, 1                  ; start TM3(104) or TM1(208)
         ld.w     [%r0], %r1
         ld.w     %r12, 1
         xcall    mdyStart
         ld.w     %r1, 1                  ; start TM5(104) or TM2(208)
         ld.w     [%r0], %r1
         ld.w     %r12, 2
         xcall    mdyStart
BACK:
         popn     %r15
         reti

.comm    iPauseFlag, 4
```

Melody output routine

The next program example is "208demo3.c" in the demo3 directory.

```
//*******************************************************************
// main
//   demo program
//*******************************************************************
#include "mdy.h"                                                             (*1)
extern unsigned short mTable25[];                                            (*2)
extern unsigned short mTable12[];
extern unsigned char all[];                                                  (*3)
extern unsigned char spring[];                                               (*3)
extern int iPauseFlag;
volatile int flag;
static void OnDone(int i)                                                    (*5')
     {
              flag = 1;                                   //set stop flag
     }
void main()
     {
     int iErr;
     iPauseFlag = 1;
     mdyOpen(MDY_SAMPLING(40000000, 100));        // 40MHz, tempo 100        (*4)
     mdyOnDone(&OnDone);                          // set OnDone function     (*5)
     iErr = mdySet(MDY_208_TM0, &mTable25[0], &all[0], 0,1);                 (*6)
                       // set table and data to TM0 with reverse output change mtable
     iErr = mdySet(MDY_208_TM1, &mTable25[0], &all[0], 0,0);                 (*6)
                       // set table and data to TM1
     mdyIntOff();                                 // interrupt disable       (*7)
     iErr = mdyStart(MDY_208_TM0);                // start melody            (*7)
     iErr = mdyStart(MDY_208_TM1);                // start melody            (*7)
     mdyIntOn();                                  // interrupt enable        (*7)
     flag = 0;
     for (;;)                                                                (*8)
         {
              if (flag == 1)
                       break;
         }
     iErr = mdySet(MDY_208_TM0, &mTable25[0], &spring[0], 0,1);
                       // set table and data to TM0 with reverse output
     iErr = mdySet(MDY_208_TM1, &mTable25[0], &spring[0], 0,0);
                       // set table and data to TM1
     mdyIntOff();                                 // interrupt disable
     iErr = mdyStart(MDY_208_TM0);                // start melody
     iErr = mdyStart(MDY_208_TM1);                // start melody
     mdyIntOn();                                  // interrupt enable
     flag = 0;
     for (;;)                                                                (*8')
         {
              if (mdyStatus(MDY_208_TM1) == MDY_FINISH)
                       break;
         }
     iErr = mdyReset(MDY_208_TM0);                // stop PWM                (*9)
     iErr = mdyReset(MDY_208_TM1);                // stop PWM                (*9)
     mdyClose();                                  // stop interrupt          (*10)
     }
```

This program illustrates control the piezoelectric buzzer direct drive using outputs from 16-bit timers 0 and 1 of the E0C33208.

*1 Make sure the source file from which MELODY33 library functions are called includes "mdylib\include\mdy.h".

*2 Define the scale tables to be referenced externally that have been created by MELODY33 tools "mtb.exe" and "bin2s.exe".
"mTable25" is a scale table for C3 through C8, created for use on the E0C33208 operating at 40 MHz by specifying 2.5 for the "mtb.exe" parameter (2.5 = 40/16).
"mTable12" is a scale table created for use on the E0C33208 operating at 20 MHz (parameter = 20/16 = 1.25). This scale table can also be used to output the musical scale in C4 through C9 when operating at 40 MHz.
When creating scale tables for the E0C33A104, you must specify the operating frequency divided by 8 for the parameter.

*3 Define the melody data to play as being referenced externally.
The melody used here has been created from "all.txt" and "spring.txt" using "txt2mdy.exe", "mdy2bin.exe", and "bin2s.exe."

*4 Initialize 16-bit timer 4 using mdyOpen( ) function before starting melody output.
In this example, the operating frequency of the E0C33208 is assumed to be 40 MHz, and the tempo is set to 100.

*5 Set a call-back function using the mdyOnDone( ) function, the function called when the system finishes playback.
Here, OnDone( ) function (*5') is registered. This function is called when the system finishes playing one channel and the flag is simply set to 1 before returning.
Instead of using a call-back function, you can use mdyStatus( ) function to determine whether the system has finished playback. (See *8'.)

*6 Set a timer by which to output a melody using mdySet( ) function.
Since this example assumes piezoelectric buzzer direct drive, 16-bit timers 0 and 1 are both used to output a melody, with timer 0 set to output an inverted waveform.

*7 Start melody playback using mdyStart ( ) function.
Because two timers are started here, the timer interrupt is disabled by mdyIntOff( ) to prevent falling out of sync before calling mdyStart ( ). Then the interrupt is reenabled by mdyIntOn( ).

*8 Check the flag set by a call-back function to see if the system finished playback.
In *8', as an example of not using call-back functions, mdyStatus( ) function is used to check whether the system finished playback. To handle a large load, as when playing multiple channels, use the smallest call-back function possible.

*9 After the system finishes playback, reset the timer using the mdyReset( ) function used for melody output.

*10 At the end of operation, turn 16-bit timer 4 off using the mdyClose( ) function.
After the system finishes playing all the channels, always call mdyClose( ). Even when melody output is turned off by mdyReset( ), the timer interrupt is generated successively at intervals equal to 1/4 the length of a thirty-second note until mdyClose( ) is called.

5.4 Precautions

(1) Call mdyOpen( ) function immediately before the system starts playback. Call mdyClose( ) function immediately after the system finishes playback. From when mdyOpen( ) is called until mdyClose( ) is called, the 16-bit timer 4 interrupt is constantly generated periodically.

(2) To output a melody from multiple timers synchronously, always use the mdyIntOff( ) function to disable the 16-bit timer 4 interrupt before calling mdyStart( ) function for each timer. After mdyStart( ) function is called, reenable the 16-bit timer 4 interrupt using the mdyIntOn( ) function.

(3) To play four channels on the E0C33208, add the -DCH4 option to the compile options in the make file. Specifying this option changes the definition in "mdy.h" to four channels.
Example: Specification in "demo208.mak"

```
GCC33_FLAG = -B$(TOOL_DIR)\ -S $(DEBUG) -O -I $(INCPATH) -DCH4
```

Appendix Verifying Operation on DMT33 Boards

The following explains how to verify melody output operation, using the E0C33 Family demonstration tools DMT33004, DMT33MON, and DMT33AMP to execute a sample program.

A.1 System Configuration Using DMT33004

A.1.1 Hardware Configuration

Configure the system shown in Figure A.1.1 using DMT33004, DMT33MON, and DMT33AMP. This system allows a melody to be output in one channel. Although we use DMT33AMP in this configuration to verify operation without preparing a specific melody output circuit, you must create an output circuit in order to output a melody on multiple channels or to drive a piezoelectric buzzer directly. For an example of this output circuit, see Section 1.2, "Basic Configuration of the Melody Output System".

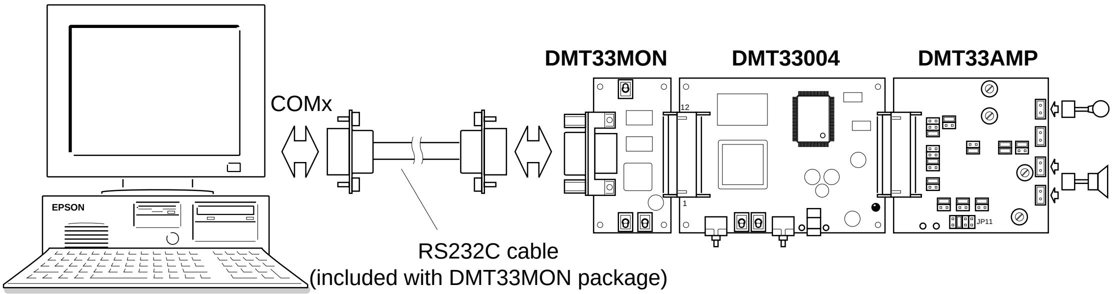


Figure A.1.1 System Configured with DMT33004, DMT33MON, and DMT33AMP

DMT33004 board

The DMT33004 is a demonstration tool for the E0C33A104, a 32-bit RISC-type microcomputer. Mounted on this board are 128-KB ROM, 1-MB RAM, 1-MB flash memory, an interface connector for the DMT33MON board, and an interface connector for a sound input/output circuit such as the DMT33AMP board. The ROM contains a debug monitor.

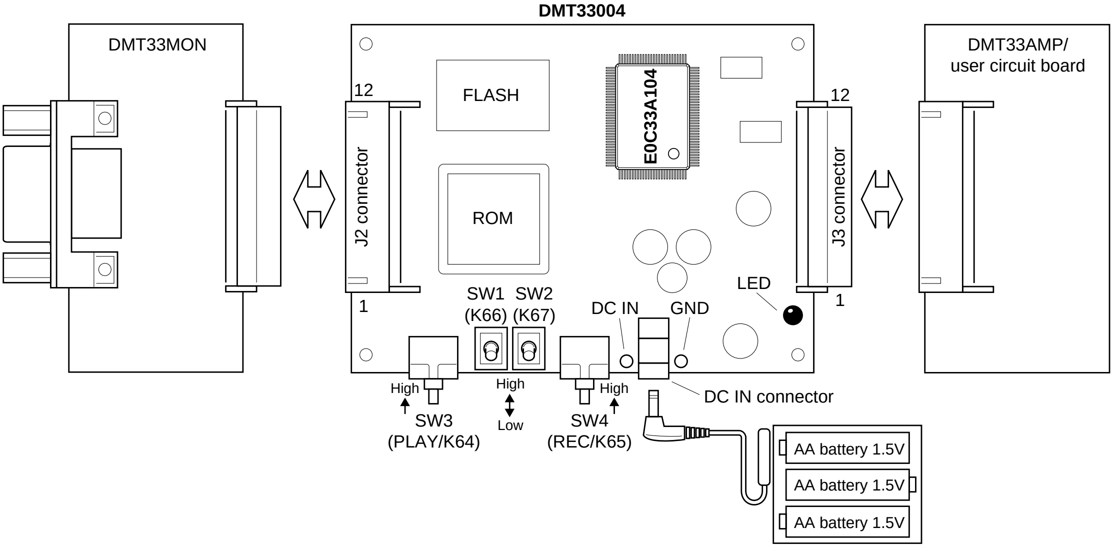


Figure A.1.2 DMT33004 Board

DMT33MON board

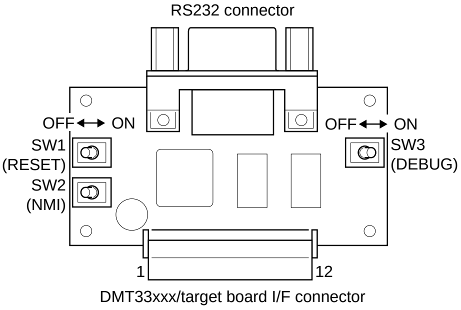


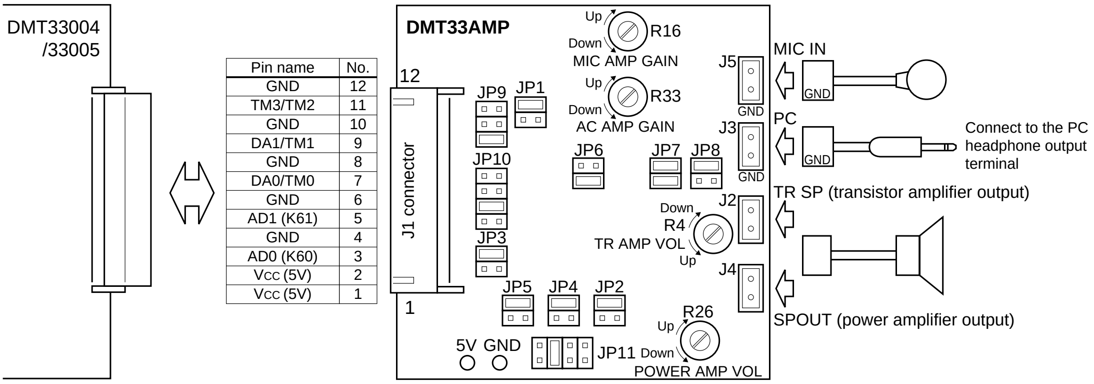
Figure A.1.3 DMT33MON Board

The DMT33MON is a tool to interface a debug monitor to demonstration tools, such as the DMT33004 or the user target board. On-board debugging using a debugger (db33.exe) running on a PC is performed by connecting the DMT33004 board to a PC via the DMT33MON board.

Note: For the DMT33004, always use a DMT33MON board manufactured to 5 V specification. DMT33MONLV boards manufactured to 3.3 V specifications cannot be used.

DMT33AMP board

The DMT33AMP is an optional board for adding sound input/output capability to the DMT33004 or other boards/microcomputers. Here, it is used to drive a speaker with the TM3 output of the DMT33004 (E0C33A104).



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

Jumper switch

JP1	DMT MIC	Selects the voice source to be output. DMT: DMT33004/33005 output (default) MIC: Microphone input of this board
JP2	DA TM	Selects an input for the transistor amplifier circuit. DA: DMT33004 D/A output (default) TM: DMT33005 PWM output
JP3	DA TM	Selects an input for the CR 2nd order filter circuit. DA: DMT33004 D/A output (default) TM: DMT33005 PWM output
JP4	SP MIC	Selects a filter in the OP AMP 4th order filter circuit (for speaker and MIC). SP: For speaker (default) MIC: For microphone
JP5	SP MIC	Selects an filter in the OP AMP 4th order filter circuit (for speaker and MIC). SP: For speaker (default) MIC: For microphone
JP6	AD1 AD0	Selects the A/D channel on the DMT33004/33005 used to convert the MIC input. AD0: Channel 0 (default) AD1: Channel 1
JP7	3300pF 1500pF	Selects a cutoff frequency in the CR 1sr order high-pass filter circuit. Short 3300pF only: 300 Hz Short 1500pF only: 500 Hz Short both: 250 Hz (default)
JP8	MIC MLP	Selects whether the OP AMP 4th order filter circuit for the MIC circuit is used or not. MIC: Not used (default) MLP: Used
JP9	TM3/TM2 DA1/TM1 DA0/TM0	Selects a DMT33004/33005 output signal. TM3/TM2: DMT33004 TM3 or DMT33005 TM2 DA1/TM1: DMT33004 DA1 or DMT33005 TM1 DA1/TM1: DMT33004 DA0 or DMT33005 TM0 (default)
JP10	TR 2LP 4LP MLP	Selects the circuit to be used for voice output. TR: Transistor amplifier circuit 2LP: CR 2nd order filter circuit 4LP: OP AMP 4th order filter circuit (for speaker) (default) MLP: OP AMP 4th order filter circuit (for speaker and MIC)
JP11	PC 2LP 4LP MLP	Selects a power amplifier input. PC: PC headphone output 2LP: CR 2nd order filter circuit 4LP: OP AMP 4th order filter circuit (for speaker) (default) MLP: OP AMP 4th order filter circuit (for speaker and MIC)

Note: The DMT33AMP is set for connecting the DMT33004 by default. When using with the DMT33005, select TM using JP2 and JP3, and DA1/TM1 using JP9.

Control

R26	Volume adjustment for the power amplifier
R4	Volume adjustment for the transistor amplifie
R33	Gain adjustment for the AC amplifier (x2 to x12)
R16	Gain adjustment for the microphone input (microphone amplifier) (x90 to x2000)

Figure A.1.4 DMT33AMP Board

To use the DMT33AMP board after connecting to the DMT33004 board, set the jumper switch JP9 to TM3/TM2. Leave all other jumper switches at their default settings.

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 Demonstration Board Manual".

1. Attach the DMT33MON and DMT33AMP to the DMT33004.
2. Connect the speaker (included with the DMT33AMP package) to the DMT33AMP.
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 DMT33004) and connect it to the DMT33004.
6. Turn on power to the PC.

A.12 Software

The PC hosting MELODY33 must have the E0C33 Family C Compiler Package development tools installed. Downloading a program into the DMT33004 with the debug monitor requires debugger (db33) Ver. 1.72 or later.

A.2 Program Execution Procedure

The sample program directory 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 DMT33004.
In the following explanation, we use the sample program "demo104.srf" for the E0C33A104 in the "mdylib\demo1\" 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 DMT33004. After reconfirming that the [DEBUG] switch (SW3) of the DMT33004 is set to the ON position, reset the system using the [RESET] switch (SW1).

(3) Start workbench wb33 and make "mdylib\demo1\" 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 "demo104.cmd".
Choose "33104_1.par" as a parameter file and start the debugger.

The debugger can also be started from the DOS prompt without wb33, as follows:
(When "mdylib\demo1\" is the current directory)

```
>C:\cc33\db33 -mon -b 115200 -p 33104_1.par -c demo104.cmd
```

(4) When the debugger starts, the sample program (demo104.srf) is loaded into the RAM (0x600000 and above) of the DMT33004 by the commands written in "demo104.cmd".

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

The sample program plays melodies *all* and *spring* one time each. The melody data played here are created from "all.txt" and "spring.txt" provided in the "mdytool\sample\" directory by "mdata1.bat".

To temporarily stop playing a melody midway through, press the NMI switch on the DMT33MON. Press the switch once more to restart.

When the program finishes playing two melodies, it is forced to stop by a hardware break. To play melodies again, execute the rsth command, then the g command.

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 (demo104) included in the "mdylib\demo1\" directory.

A.3.1 Explanation of Files

Source files

The main program for the DMT33004 (E0C33A104) is "104demo1.c". This sample program was created assuming it will be run on the E0C33A104 operating at 20 MHz.
In addition, several other files such as "boot.s", "vector.s", and "vector.h" are used. "mdata1.s" and "mtable.s" are also used as melody and scale table data.
For more information on creating a melody output routine, see Section 5.3, "Program Example". For more information on creating melody and scale table data, see Section 3.1, "Creating Melody ROM Data".

Linker command file

The contents of the linker command file (demo104.cm) used to link the sample program are shown below.
Since demo104 is created for use on the E0C33A104, the MELODY33 library "mdy104.lib" is linked. For use on the E0C33208, "mdy208.lib" must be linked.
Since the sample program is run in the external RAM of the DMT33004, the start address of the CODE section is set to 0x600000.

```
;Map set
-code 0x0600000            ; set relative code section start address
-bss  0x0000030            ; set relative bss  section start address

;Library path
-l C:\CC33\lib
..\lib\mdy104.lib

;Executable file
-o demo104.srf

;Object files start
vector.o
boot.o
mdata1.o
mtable.o
104demo1.o

;Object files end

;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 "demo104.mak". If you corrected the source file, you need to create the object file in executable format "demo104.srf" by using "demo104.mak".

Execution procedure for make

1. Set "mdylib\demo1\" to the current directory.
2. Enter the command shown below from the DOS prompt:

```
C:\E0C33\MDY33\MDYLIB\DEMO1>make -f demo104.mak
```

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

A.4 When Using the DMT33005 Board

The DMT33005 uses the E0C33208 as the CPU. Therefore, the sample program for the DMT33004 (E0C33A104) described above cannot be used with the DMT33005.
So use the sample program for the DMT33005 which is provided in the "mdylib\demo1\" directory. Program operation can be verified following the same procedure that is described for the DMT33004 above. When using the DMT33005, leave its DSW1 as set by default (PLL x2/20 to 33 MHz input).

When using the DMT33005 in place of the DMT33004 shown in the system configuration of Figure A.1.1, choose TM for the DMT33AMP jumper switches JP2 and JP3, and DA1/TM1 for JP9. Other jumper switches may be used as set, because there is no need to change settings between the DMT33004 and DMT33005.

Furthermore, when creating a program for the E0C33208, you need to link the MELODY33 library "mdy208.lib." For more information, see the sample linker command file.

EPSON International Sales Operations

AMERICA

EPSON ELECTRONICS AMERICA, INC.

- HEADQUARTERS -
1960 E. Grand Avenue
El Segundo, CA 90245, U.S.A.
Phone: +1-310-955-5300 Fax: +1-310-955-5400

- SALES OFFICES -

West
150 River Oaks Parkway
San Jose, CA 95134, U.S.A.
Phone: +1-408-922-0200 Fax: +1-408-922-0238

Central
101 Virginia Street, Suite 290
Crystal Lake, IL 60014, U.S.A.
Phone: +1-815-455-7630 Fax: +1-815-455-7633

Northeast
301 Edgewater Place, Suite 120
Wakefield, MA 01880, U.S.A.
Phone: +1-781-246-3600 Fax: +1-781-246-5443

Southeast
3010 Royal Blvd. South, Suite 170
Alpharetta, GA 30005, U.S.A.
Phone: +1-877-EEA-0020 Fax: +1-770-777-2637

EUROPE

EPSON EUROPE ELECTRONICS GmbH

- HEADQUARTERS -
Riesstrasse 15
80992 Muenchen, GERMANY
Phone: +49-(0)89-14005-0 Fax: +49-(0)89-14005-110

- GERMANY -
SALES OFFICE
Altstadtstrasse 176
51379 Leverkusen, GERMANY
Phone: +49-(0)217-15045-0 Fax: +49-(0)217-15045-10

- UNITED KINGDOM -
UK BRANCH OFFICE
2.4 Doncastle House, Doncastle Road
Bracknell, Berkshire RG12 8PE, ENGLAND
Phone: +44-(0)1344-381700 Fax: +44-(0)1344-381701

- FRANCE -
FRENCH BRANCH OFFICE
1 Avenue de l' Atlantique, LP 915 Les Conquerants
Z.A. de Courtaboeuf 2, F-91976 Les Ulis Cedex, FRANCE
Phone: +33-(0)1-64862350 Fax: +33-(0)1-64862355

ASIA

- CHINA -
EPSON (CHINA) CO., LTD.
28F, Beijing Silver Tower 2# North RD DongSanHuan
ChaoYang District, Beijing, CHINA
Phone: 64106655 Fax: 64107320

SHANGHAI BRANCH
4F, Bldg., 27, No. 69, Gui Jing Road
Caohejing, Shanghai, CHINA
Phone: 21-6485-5552 Fax: 21-6485-0775

- HONG KONG, CHINA -
EPSON HONG KONG LTD.
20/F., Harbour Centre, 25 Harbour Road
Wanchai, HONG KONG
Phone: +852-2585-4600 Fax: +852-2827-4346
Telex: 65542 EPSCO HX

- TAIWAN, R.O.C. -
EPSON TAIWAN TECHNOLOGY & TRADING LTD.
10F, No. 287, Nanking East Road, Sec. 3
Taipei, TAIWAN, R.O.C.
Phone: 02-2717-7360 Fax: 02-2712-9164
Telex: 24444 EPSONTB

HSINCHU OFFICE
13F-3, No. 295, Kuang-Fu Road, Sec. 2
HsinChu 300, TAIWAN, R.O.C.
Phone: 03-573-9900 Fax: 03-573-9169

- SINGAPORE -
EPSON SINGAPORE PTE., LTD.
No. 1 Temasek Avenue, #36-00
Millenia Tower, SINGAPORE 039192
Phone: +65-337-7911 Fax: +65-334-2716

- KOREA -
SEIKO EPSON CORPORATION KOREA OFFICE
50F, KLI 63 Bldg., 60 Yoido-Dong
Youngdeungpo-Ku, Seoul, 150-010, KOREA
Phone: 02-784-6027 Fax: 02-767-3677

- JAPAN -
SEIKO EPSON CORPORATION
ELECTRONIC DEVICES MARKETING DIVISION

Electronic Device Marketing Department
IC Marketing & Engineering Group
421-8, Hino, Hino-shi, Tokyo 191-8501, JAPAN
Phone: +81-(0)42-587-5816 Fax: +81-(0)42-587-5624

ED International Marketing Department I (Europe & U.S.A.)
421-8, Hino, Hino-shi, Tokyo 191-8501, JAPAN
Phone: +81-(0)42-587-5812 Fax: +81-(0)42-587-5564

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

ENERGY
SAVING
EPSON

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

EPSON

SEIKO EPSON CORPORATION

ELECTRONIC DEVICES MARKETING DIVISION

■ EPSON Electronic Devices Website

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

Issue SEPTEMBER 1999, Printed in Japan Ⓜ A