MF1267-01

EPSON

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

VRE33 MIDDLEWARE MANUAL

ENERGY SAVING EPSON

SEIKO EPSON CORPORATION

PREFACE

This manual describes the configuration and functions of Speech Recognition Middleware VRE33 for the E0C33 Family, and explains methods for using this middleware. It is targeted to developers of applications for the E0C33 Family of microcomputers.

CONTENTS

1 Outline of VRE33 Middleware 1
1.1 Components of the VRE33 Package 1
1.2 Basic Configuration of a Speech Recognition System 2

2 Installation 4
2.1 Operating Environment 4
2.2 Method of Installation 4

3 Software Development 6
3.1 Creating Speech Recognition Dictionary Data by Using the VRE33 Tool 7
3.1.1 Outline of Recognition Dictionary Data 8
3.1.2 Gathering Speech Data to Create a Recognition Dictionary 9
3.1.3 Separating between Words and Processing PCM Data 10
3.1.4 Creating Dictionary Data 11
3.1.5 Evaluating Recognition Performance on a PC 12
3.1.6 Converting Recognition Dictionary Data into Assembly Source Files 13
3.2 Creating an Application and Linking to the VRE33 Library 14

4 VRE33 Tool Reference 15
4.1 Outline of VRE33 Tools 15
4.2 Description of Tools 17
4.2.1 getpcm.exe 17
4.2.2 addslnt.exe 18
4.2.3 pcm_norm.exe 19
4.2.4 mpmSlct.exe 20
4.2.5 mpmDict.exe 21
4.2.6 mpmRecog.exe 23
4.2.7 bin2s.exe 24
4.2.8 Executing from a Batch File 25

5 VRE33 Library Reference 29
5.1 Outline of VRE33 Library 29
5.2 Hardware Resources and Initialization 31
5.3 Top-level Functions 32
5.3.1 Compile Options 32
5.3.2 Defining the Dictionary 32
5.3.3 Structure of Quantified Characteristics Data 33
5.3.4 vreRecognise( ) 34
5.3.5 vreRecogInit( ) 35
5.3.6 vreMakeDictionary( ) 36
5.3.7 mpmRecognition( ) 37
5.4 VRE33 Library Functions 38
5.4.1 Speech-recognition Processing Functions 39
5.4.2 Input (Listen) Functions 42
5.5 Techniques for Speeding Up Processing 46
5.6 Library Performance 46

5.7 Combined Use with VOX33 Library 47
5.8 Program Examples 48
5.9 Precautions 53

Appendix Verifying Operation with DMT33 Boards 54

A.1 System Configuration Using DMT33004 54
A.1.1 Hardware Configuration 54
A.1.2 Software 56
A.2 Sample Program Execution Procedure 57
A.3 Making a Program 61
A.4 Entering Speech for Dictionary Data Creation/Evaluation on DMT Boards 62
A.4.1 Entering Speech to Create Recognition Dictionary Data Using pcmrec.bat 62
A.4.2 Evaluating Speech Recognition Using recogXXX.bat 63
A.5 When Using the DMT33005 Board 64

1 Outline of VRE33 Middleware

VRE33 is speech-recognition middleware for the E0C33 Family of microcomputers. It is designed to perform speech-recognition processing on the E0C33 Family chip. Speech recognition functions are supplied as library functions, which can be used by linking with the target program.
The product also contains PC software tools for creating speech recognition dictionaries and for evaluating speech recognition.
VRE33 middleware is ideally suited to the development of applications such as voice memos, databanks with voice functions, PDAs, electronic stationery, and toys.

The main features of the VRE33 middleware are given below:

- Designed for use with the E0C33 Family; incorporates an A/D converter and 16-bit programmable timer.
- Can recognize 20 to 100 words in real-time using Seiko Epson's exclusive isolated word-recognition technology.
- Supports both specific and nonspecific user-speech recognition.
  Nonspecific user-speech recognition:
  Creation of a recognition dictionary to recognize the speech of a number of different individuals
  Specific user-speech recognition:
  Speech recognition for a specific individual, allowing recognition data to be registered directly on the actual unit
- Supports the Cepstrum and VQCode recognition data formats.
  Cepstrum: Speech characteristics are turned into data. Higher recognition accuracy correlates with greater data volume. Can be used for both specific and nonspecific user-speech recognition.
  VQCode: Derived from speech characteristics (Cepstrum) by quantization. The volume of data is about 1/10 that of Cepstrum, but recognition accuracy is lower. Suitable only for specific user-speech recognition.

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 Components of the VRE33 Package

The contents of the VRE33 package are listed below. When unpacking, check to see that all of the following items are included.

(1) Tool disk (CD-ROM)	1 pc.
(2) E0C33 Family VRE33 Middleware Manual (this manual)	1 pc. each in English and Japanese
(3) Warranty card	1 pc. each in English and Japanese

1.2 Basic Configuration of a Speech Recognition System

Hardware configuration

Figure 1.2.1 shows the basic hardware configuration of a speech recognition system, including a speech input/output unit. Such a system is built around the E0C33 chip and incorporates various peripherals, such as external memory, amplifier, microphone, or speaker (unnecessary for speech recognition only).
Note that the VRE33 library uses one channel of the A/D converter and one channel of the 16-bit programmable timer incorporated into the E0C33 chip. A certain volume of internal RAM is also used for high-speed operation.

Note: The VRE33 middleware does not include a speech output unit. Separate speech processing routines such as VOX33 middleware are required for speech output.

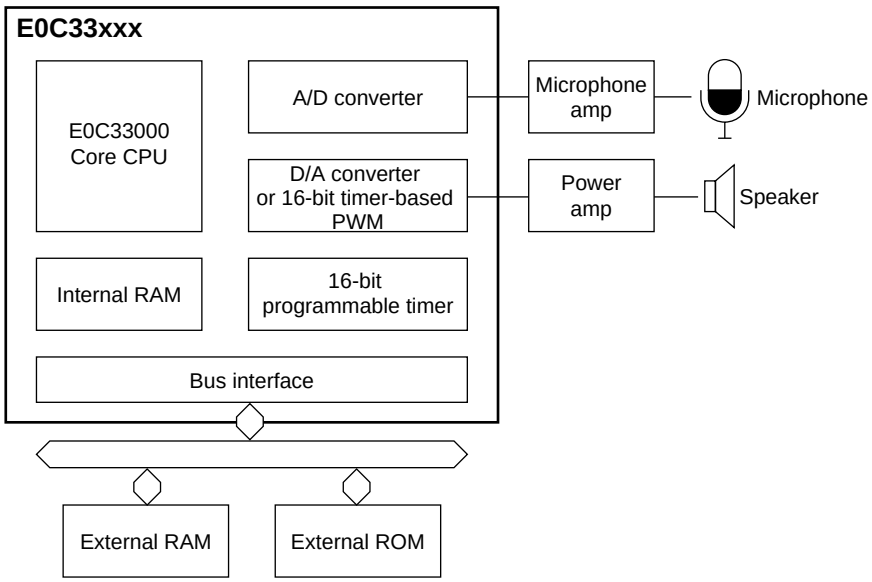


Figure 1.2.1 Hardware Configuration for Speech Input/Output Unit

Software configuration

The VRE33 library is middleware, positioned between the E0C33 hardware and user applications, performing hardware control associated with speech input and recognition. By including or linking the top-level functions supplied as C source files into user applications, you can easily perform speech-recognition processing, without having to call VRE33 library functions directly from the applications.

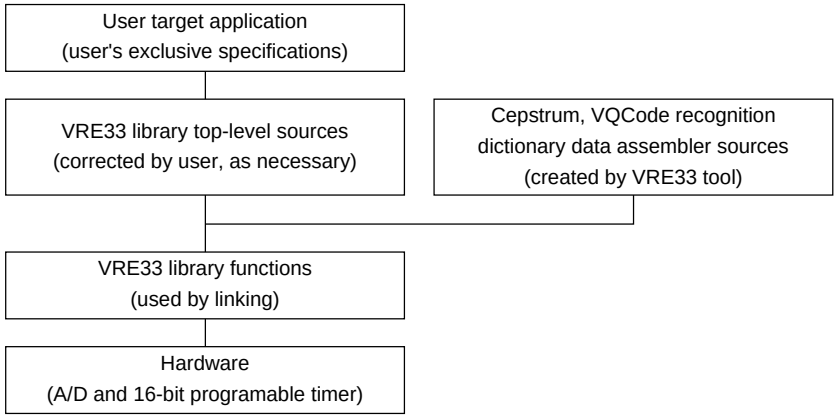


Figure 1.2.2 Software Configuration for Speech Input/Output Unit

For detailed information on VRE33 library functions and top-level functions, see Section 5, "VRE33 Library Reference".

VRE33 tools are PC software used to create and evaluate speech recognition dictionary data. When complete, this data is downloaded to an E0C33 Family chip. All tools are 32-bit applications executable from a DOS prompt, and will run under Windows 95, Windows NT 4.0, or later Windows versions.
For detailed information on VRE33 tools, see Section 4, "VRE33 Tool Reference".

2 Installation

This section describes the operating environment for the VRE33 tools and explains how to install the VRE33 middleware.

2.1 Operating Environment

The following are the minimum requirements for creating and evaluating speech recognition directory data with the VRE33:

Personal computer

An IBM PC/AT or fully-compatible machine. We recommend a Pentium 90 MHz or higher CPU, and 32 MB or more of RAM. A CD-ROM is required to install tools from the CD-ROM.

Display

An SVGA (800 × 600) monitor or better. In the Windows Control Panel, choose "Small font" among the display options.

System software

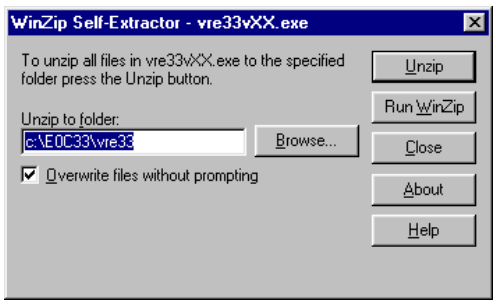
VRE33 tools run under Microsoft® Windows® 95 or Windows NT® 4.0, or later versions of Windows (English and Japanese versions).

Other

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

2.2 Method of Installation

The VRE33 library and VRE33 tools are supplied on CD-ROM. Open the self-extracting file on the CD-ROM named "vre33vXX.exe" to install the VRE33 library and VRE33 tools in your computer. (The XX in this file name denotes a version number. For Version 1.0, for example, the file is named "vre33v10.exe".)
Double-click on "vre33vXX.exe" to start installation. The dialog box shown below appears.



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

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

(root)\

readme.txt		Supplementary explanation, etc. (in English)
readmeja.txt		Supplementary explanation, etc. (in Japanese)
vretools	..... VRE33 tool directory	
readme.txt		VRE33 tool supplementary explanation, etc. (in English)
readmeja.txt		VRE33 tool supplementary explanation, etc. (in Japanese)
bin	..... VRE33 tools	
mpmRecog.exe		Recognition behavior evaluation tool
mpmDict.exe		Dictionary creation tool
mpmSlct.exe		File list creation tool
bin2s.exe		Binary to assembly source conversion tool
getpcm.exe		PCM file conversion tool
pcm_norm.exe		PCM normalization tool
addslnt.exe		Silent data insertion tool
src	..... Source directory	
Public tool source file		
utility	..... Utility directory	
DMT board-based PCM sampling file		
rec104\		Used for DMT33004 + DMT33AMP + DMT33MON
rec208\		Used for DMT33005 + DMT33AMP + DMT33MON
sample	..... Sample directory	
Sample speech, batch files, sample dictionary creation data		
vrelib	..... VRE33 library-related	
readme.txt		VRE33 library supplementary explanation, etc. (in English)
readmeja.txt		VRE33 library supplementary explanation, etc. (in Japanese)
lib	..... VRE33 library directory	
vre.lib		VRE33 library
sl104.lib		Input/output library for E0C33A104
sl208.lib		Input/output library for E0C33208
memMesa.o, memAsm.o, mpmFt.o, mpmVq.o		
		Objects retrieved from vre.lib for high-speed operation
include	..... Header file directory for VRE33 library functions	
vre.h		VRE33 header file
src	..... Library source directory	
vretop.c		VRE33 top-level functions
vrecache.c		VRE33 internal RAM cache functions
hardsrc	..... Hardware dependent source directory	
Listen.s		Listen.o source (E0C33A104)
LisAD.s		LisAD.o source (E0C33A104)
Speak.s		Speak.o source (E0C33A104)
SpkDA.s		SpkDA.o source (E0C33A104)
Lis208.s		Lis208.o source (E0C33208)
Lis208AD.s		Lis208AD.o source (E0C33208)
Spk208.s		Spk208.o source (E0C33208)
Spk208PW.s		Spk208PW.o source (E0C33208)
slintr.def		
(Refer to these sources when changing timer A/D or D/A channel ports.)		
smpl104	..... DMT33004 sample program directory	
smpl208	..... DMT33005 sample program directory	
(For detailed information on the configuration of sample programs and how to use them, see "readme.txt" or "readmeja.txt" in "vrelib".)		

Although you can select a different directory structure and file organization, the discussions in the following pages will assume that the files have been copied from the CD-ROM according to the directory structure given above.

3 Software Development

This section describes how to develop speech-recognition processing software on the E0C33 Family chip. Shown below is the basic flow of development:

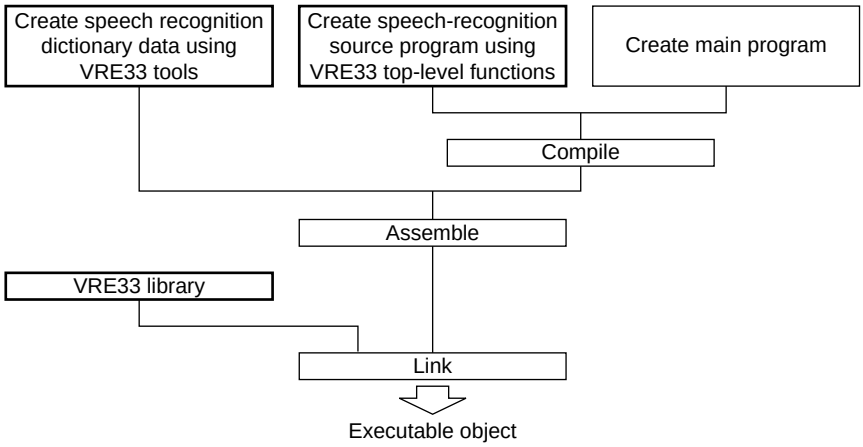


Figure 3.1 Procedure for Developing E0C33 Speech-recognition Processing Software

1) Using VRE33 tools, create a speech recognition dictionary from PCM data into which speech to be recognized has been sampled, and convert the created dictionary into an assembly source file.
2) Create a user application. For speech-recognition processing, use the top-level functions provided in the VRE33 library. You can include the source file for the speech recognition dictionary data created in Step1 in the user application source.
3) Compile and assemble the source program.
4) Link the objects generated in Step 3 with the VRE33 library. This generates an executable object file.

3.1 Creating Speech Recognition Dictionary Data by Using the VRE33 Tool

Figure 3.1.1 shows procedures of creating speech recognition dictionary data and VRE33 tool configuration.

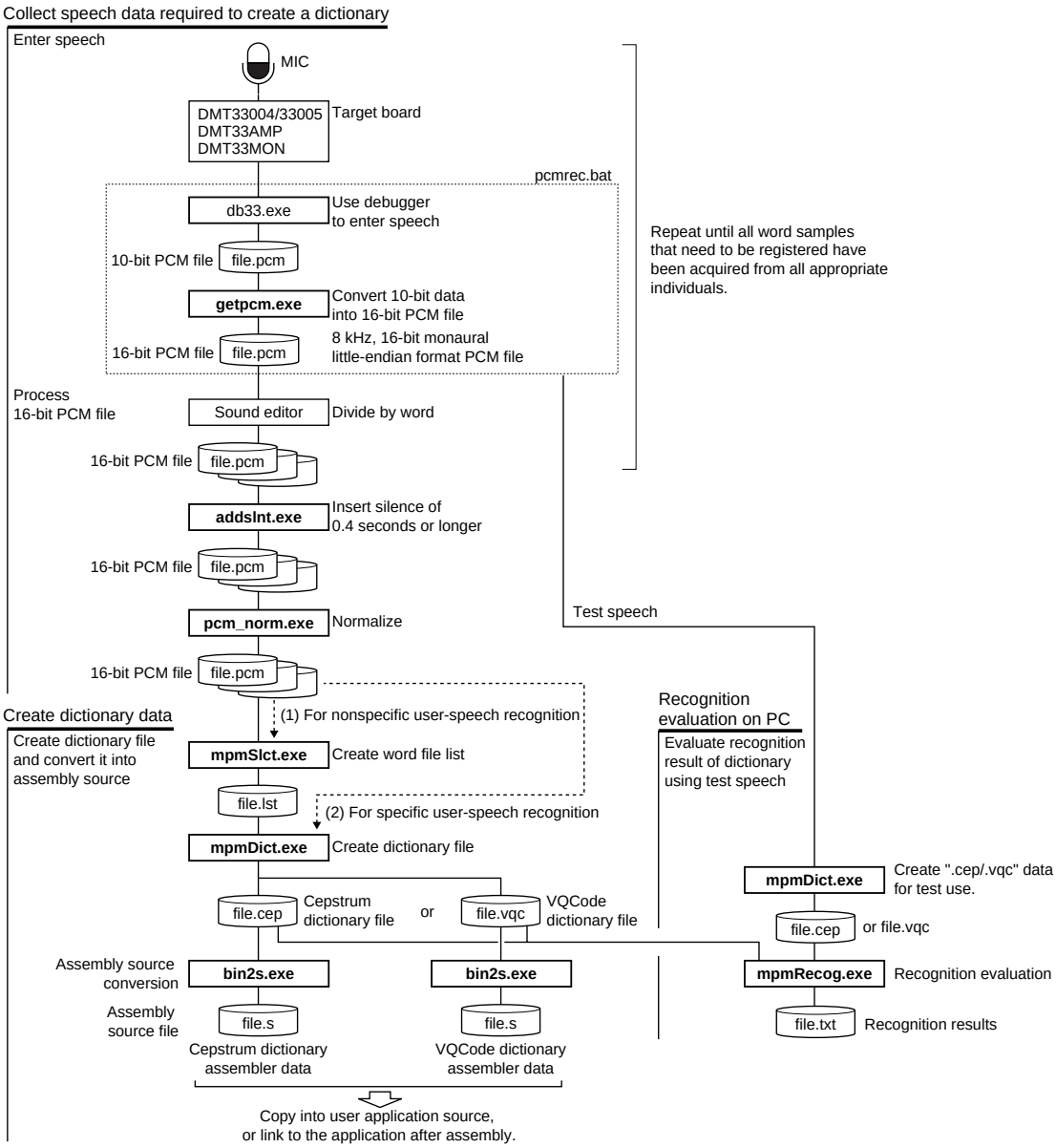


Figure 3.1.1 Flowchart for Creating Speech Recognition Dictionary Data

The following provides a mere outline of usage of the VRE33 tools. For additional information, see Section 4, "VRE33 Tool Reference".

The following explanation assumes that PATH is set to the "vretool\bin\" directory.

Example: `DOS>PATH c:\e0c33\vre33\vretool\bin`

3.1.1 Outline of Recognition Dictionary Data

Nonspecific user-speech recognition and specific user-speech recognition

Consider the recognition of the single phrase, "Good morning." To create a recognition system for a single individual, the dictionary needs recorded voice data for that individual only. This is referred to as **specific user-speech recognition**. In this case, the dictionary is generally created by recording the user's speech on the actual product. Since only one data entry is required for this particular phrase, a dictionary for specific user-speech recognition purpose is created by preparing one data entry for each word or phase to be recognized. The VRE33 library provides the functions for recording speech and creating a dictionary on the actual machine.
The following is a dictionary structure.
Example:

```
extern const short goodMorning[];  Recognition data for "good morning"
          :                        Other recognition data
extern const short goodBye[];
/* dictionary table */
const short *Greet[10] = {         When there are a total of 10 data entries
        goodMorning,               (words or phases)
            :
        goodBy
};
```

General products that come with a built-in dictionary need to be able to recognize the speech of many (nonspecific) individuals. This is referred to as **nonspecific user-speech recognition**. In this case, speech characteristics and speech mannerisms will vary greatly, with sex and age being two primary factors. For this reason, multiple instances of "good morning" recognition data are created, one for each group with the same features (e.g. one for adult women, one for boys). In a dictionary for nonspecific user-speech recognition purpose, one word or phrase consists of multiple data entries. You can improve the recognition rate for any group by including samples from as many people as possible.
The following is a dictionary structure.
Example:

```
extern const short goodMorning_women[]; Recognition data 1 for "good morning"
extern const short goodMorning_men[];   Recognition data 2 for "good morning"
          :                             Other recognition data
extern const short goodBye_girls[];
extern const short goodBye_boys[];
/* dictionary table */
const short *Greet[40] = {              When there are a total of 40 data entries
        goodMorning_women,              (words or phases)
        goodMorning_men,
            :
        goodBy_girls
        goodBy_boys
};
```

Dictionary data formats (Cepstrum and VQCode)

The VRE33 tools and library support the dictionary data structure of the following two data formats:

Cepstrum

Speech characteristics encoded in 20 bytes per frame (processing unit of speech recognition, 16 ms in the case of 8K sampling). Provides high accuracy, with correspondingly high data volume. Suitable for both specific and nonspecific user-speech recognition.

VQCode

Derived from speech characteristics (Cepstrum) by quantization into 2-byte code. Although data volumes are about 1/10 that of Cepstrum, accuracy is lower. Suitable only for specific user-speech recognition.

3.1.2 Gathering Speech Data to Create a Recognition Dictionary

The VRE33 tools create dictionary data from PCM files in the following format:

8 kHz-sampling, 16-bit, little-endian format monaural speech file

For each word and phase to be recognized, record the speech of as many people as possible, preferably 30 people or more. Prepare a PCM file in the above format. Record speech data for the target age and sex groups of the product. If you cannot identify a target age or sex, we recommend dividing the target into the following groups and preparing data for 30 or more people in each group.

1. Adult men and others
   Create two recognition data types for words and phrases. The recognition rate will be moderate.
2. Adult men, adult women, boys, and girls
   Create four recognition data types for words and phrases. This gives improved recognition rates.

Given below are some precautions regarding the sample gathering environment:

- Use a microphone and microphone amp as similar as possible to the ones used in the actual product. (Recommended sample gathering tools: DMT33004/33005 + DMT33AMP + DMT33MON)
- Adjust microphone input levels properly, avoiding excessively high or low settings.
- Take measures to avoid picking up electrical noise generated by the microphone, or noise generated by vibrations and breathing.
- Pay attention to the sample gathering environment. Choose an echo-free and quiet environment, or an environment similar to the one in which product will be used.

To increase recognition rates, carefully choose the registered words or phrases. Choose words or phrases that are as simple as possible, then avoid registering similar words or phrases. The following words or phrases tend to generate high rates of incorrect recognition:

- Words differing in only their beginning syllable(s):
  Example: Johnny, Tony, pony, etc.
- Words beginning with the same syllable(s):
  Example: Thirteen, thirtieth, thirty,
- Words commonly pronounced differently by different people
  Example: Hello, OK, etc.
- Excessively long words, or a phrase consisting of many words
- Words in which the syllables are difficult to demarcate
  Example: Johnny, journey, etc.

Make sure that the speech is recorded at normal speed. The permissible range of deviation for speech pace during recognition is ±20% to 30% of the dictionary data.

When using the DMT33004/33005 + DMT33AMP + DMT33MON to record speech, you can use the batch file "pcmrec.bat" provided in the "vretools\utility\rec104\" directory (for DMT33004), or "vretools\utility\rec208\" directory (for DMT33005). This batch file starts debugger db33 to control the DMT boards, allowing you to make recordings of up to 60 seconds. Record multiple words during this time, then separate them by word in a later process. For additional information on recording, see Appendix.

Because the PCM file created here consists of 10-bit data, it is converted into a 16-bit PCM file in "pcmrec.bat" with the VRE33 tool "getpcm.exe".

3.1.3 Separating between Words and Processing PCM Data

Demarcate individual words from the sampled speech data (PCM file). Use a general sound editor.

The dictionary data creation tool "mpmDict.exe" determines that a word is finished when a preset period of silence (by default, 20 packets = 20 × 16 ms = approx. 0.4 seconds) elapses after termination of the speech waveform. To delimit individual words, add a period of silence after the word (normally 0.4 seconds or longer, or equal to or longer than the time set by "mpmDict.exe"). This period of silence can also be added with the VRE33 tool "addslnt.exe".
Example: >`addslnt hello.pcm hello0.pcm 2560`

In this example, "hello0.pcm" is created by adding a period of silence of 2,560 samples (20 packets × 128 samples) after "hello.pcm".

Because "mpmDict.exe" also has an automatic distinguishing function, PCM files of a single recorded word do not require delimiting between words using a sound editor. However, automatic delimiting may not always function properly, due to noise and other factors. (In such cases, "mpmDict.exe" generates the message "Warning: Cannot process file ...") In this case, while creating data, cut the periods of silence before and after the word (including the 0.4-second period of silence) and turn off the "mpmDict.exe" automatic delimiting function.

After creating a PCM file for each word, adjust the level of each word. Use the VRE33 tool "pcm_norm.exe" (it is also included among the VOX33 voice compression/expansion tools). The "pcm_norm.exe" tool is used to adjust the entire amplitude of sound in the PCM file so that its maximum amplitude is corrected to a specified value. The value is specified as a percent of the maximum value of 16-bit PCM data. By default, this value is set to 90% for use in voice-compression processing. However, for speech recognition dictionary data, specify 100%.
Example: >`pcm_norm -r 100 hello0.pcm hello1.pcm`

In this example, maximum amplitude for "hello0.pcm" is adjusted to 100% by the "-r 100" option, and the adjusted data is written to "hello1.pcm". This adjustment can be processed simultaneously when delimiting words with a sound editor.

Perform the maximum amplitude processing for all recorded words. Create dictionary data for each word for each user-voice group. Use "mpmSlct.exe" to create folders and assign file names to allow easy selection of files by word or by group, using the wildcard (*).
Example:

Group of adult men	Words "good morning"	Man 1	\men\oha1.pcm
		:	
		Man 30	\men\oha30.pcm
		(All files can be specified by \men\oha*.pcm)	
	Word "goodnight"	Man 1	\men\oya1.pcm
		:	
		Man 30	\men\oya30.pcm
		(All files can be specified by \men\oya*.pcm)	
Group of adult women	Words "good morning"	Woman 1	\women\oha1.pcm
		:	
		Woman 30	\women\oha30.pcm
		(All files can be specified by \women\oha*.pcm)	
	Word "goodnight"	Woman 1	\women\oya1.pcm
		:	
		Woman 30	\women\oya30.pcm
		(All files can be specified by \women\oya*.pcm)	

3.1.4 Creating Dictionary Data

Use the PCM files prepared for all words in the preceding sections to create dictionary data.

First, using "mpmSlct.exe", create a list of PCM files that comprise one dictionary data entry. A single dictionary data entry generally refers to data for one word in one user-voice group. Refer to the preceding example. The dictionary data entry produced from the 30 "good morning" data entries for the group of adult men (\men\oha*.pcm) constitutes one dictionary data entry. To write these 30 data to a list file, run "mpmSlct.exe", as shown below:

Example: `>mpmSlct` (path)`\men\oha*.pcm > ohaM.lst`

The list file created by this example is a text file containing the following:

Example: After the files "oha1.pcm" to "oha30.pcm" have been prepared in the "\men\" directory

```
>type ohaM.lst
(path)\men\oha1.pcm
(path)\men\oha2.pcm
        :
(path)\men\oha30.pcm
```

Create this list file for all words for each user-voice group.

Example: "Good morning" for the group of adult men: ohaM.lst
"Good morning" for the group of adult women: ohaW.lst
"Goodnight" for the group of adult men: oyaM.lst
"Goodnight" for the group of adult women: oyaW.lst

Note: The executable "mpmSlct.exe" cannot select files on a network drive that is logically regarded as a disk partition. To create a list of data files on a network, assign a directory to the network drive, or copy the files to a local drive.

Run "mpmDict.exe" to create recognition dictionary data.

Example:
```
>mpmDict -f ohaM.lst ohaM.cep          (1)
>mpmDict -vqc -f ohaM.lst ohaM.vqc     (2)
```

In example (1), the recognition dictionary data entry "ohaM.cep" for Cepstrum (the default data format) is created from the files written in "ohaM.lst". In example (2), the recognition dictionary data entry "ohaM.vqc" for VQCode is created because the -vqc option is specified. Cepstrum is recommended for better recognition rates, but to create dictionary data that is as compact as possible, create data for VQCode along with one for Cepstrum, then check to see if it is usable by evaluating its recognition performance on a PC. Perform this processing for all list files.

The "mpmDict.exe" has other options available (e.g. for automatic word delimiting). Depending on the input PCM files, you may be unable to delimit words (by default, the automatic delimiting function is enabled), in which case you may need to specify other options or process the source PCM data. For additional information, see Section 4.2.5, "mpmDict.exe".

3.1.5 Evaluating Recognition Performance on a PC

After creation of recognition dictionary data is completed, check to see that it is actually capable of recognizing speech by running "mpmRecog.exe" on a PC. You can perform this confirmation by preparing data files for evaluation purpose on a PC, or by entering speech with the DMT33004/33005 + DMT33AMP + DMT33MON boards. The explanation given here applies to the first method (i.e., preparing data files on a PC). For information on the second method, see the Appendix.

First, prepare speech data for evaluation (16-bit PCM files). Prepare another recorded data set separate from the one used to create recognition dictionary data. Try preparing as much data as possible, including expressions not present in the dictionary. This helps to increase the accuracy of the evaluation. Prepare speech files for each word before beginning, if possible. There is no need to add period of silence or adjust sound levels.

Next, create a list of the recognition dictionary data created in the preceding sections as a text file. You can use "mpmSlct.exe" to do this.

Example:
```
>mpmSlct (path1)\*.cep > test.dic
>mpmSlct (path1)\*.vqc >> test.dic
>mpmSlct (path2)\*.cep >> test.dic
                :
```

Convert the prepared speech data into ".cep" or ".vqc" files with the executable "mpmDict.exe", and run "mpmRecog.exe" to perform speech recognition.

Example: Evaluating with Cepstrum data created for testing
```
>mpmDict.exe word1.pcm temp.cep          (1)
>mpmRecog text.dic temp.cep result.txt   (2)
```

In (1), Cepstrum data is created from one PCM file for testing. In (2), recognition of "temp.cep" is checked using the created recognition dictionary data. Evaluations can also be made by entering data not in the Cepstrum or VQCode format used for the dictionary. Recognition results are output to "result.txt", a file in text format.

Example: Example of recognition results output
```
001 0x0003c606 ..\dict\thanks.cep
003 0x0012c4d6 ..\dict\hello.cep
002 0x0008c2fe ..\dict\morning.cep
```

The contents of each line are as follows:
<rank> <distance> <recognition dictionary data file name>

The <rank> is assigned numbers from 001 in order of <distance>, beginning with the smallest.
The <distance> represents the difference in features between the test data and recognition dictionary data. The smaller the value, the closer the similarity. The above example indicates that the input data was closest to "thanks.cep".

The range for <distance> within which words are assumed to be identical must be determined after examining the actual results of evaluation. The library functions include a reject function by which words with distances greater than a specified value are regarded as not matching.
Use this recognition result to determine whether VQCode is useable in place of Cepstrum. On an actual machine, recognition can only be performed with one of Cepstrum or VQCode. Note that only Cepstrum can be used for specific user-speech recognition.

If the recognition rate is poor, take appropriate corrective measures. For example, you can increase the number of individuals from whom recognition dictionary data is gathered.

3.1.6 Converting Recognition Dictionary Data into Assembly Source Files

Once you finish the recognition evaluation, convert the completed recognition dictionary data into an assembly source file for the E0C33 assembler for inclusion in or linking to your application. To perform this conversion, use "bin2s.exe".

Example: `>bin2s thanks.cep > dict.s` (using the DOS redirect function)
`>bin2s morning.cep >> dict.s`

In this example, "thanks.cep" is converted into the assembly source "dict.s", then the data derived by conversion from "morning.cep" is appended to "dict.s". Each converted data entry has the input file names "thanks" and "morning" added as global symbols. (You can change symbol names using the "-l *symbol*" option of "bin2s.exe".)

Example: Contents of "dict.s"

```
        .global thanks
        .align  2
  thanks:
        .byte   0x33 0x20 0x24 0x00 0xa9 0x09 0xe5 0x03
        .byte   0x9c 0xfe 0x95 0xfe 0xf7 0xfe 0x3d 0x01
        .byte   0xe6 0x01 0x0b 0xff 0xa7 0x00 0x14 0x00
                                    :
        .byte   0xf8 0xff 0x3a 0x00
  ; total 724 bytes data

        .global morning
        .align  2
  morning:
        .byte   0x33 0x20 0x20 0x00 0x14 0x0a 0x1f 0x07
        .byte   0x67 0x02 0x15 0x00 0x57 0xfe 0x3d 0xff
        .byte   0xa6 0xff 0x20 0x00 0xf8 0xff 0x47 0x00
                                    :
        .byte   0x22 0x00 0x9f 0xff
  ; total 644 bytes data
```

3.2 Creating an Application and Linking to the VRE33 Library

You can implement speech recognition on the E0C33 chip by calls to VRE33 library functions. Note that in addition to low-level library objects, this product contains source code for top-level functions created in C. This allows you to create speech processing routines simply by including the functions in your program. For detailed information on top-level and VRE33 library functions, see Section 5, "VRE33 Library Reference". Sample programs are provided in the "vrelib\samlxxx\" directory for reference.

You must also include the source of the created recognition dictionary data in your program, or link it along with the VRE33 library after assembly.

When creating and linking a program, watch the following:

(1) VRE33 library functions use the CPU's internal R8 register. When linking VRE33 library including the top-level functions (vretop.c), you cannot use the -gp option of instruction extender ext33 (optimization by global pointer/R8).

(2) Make sure that all of the BSS sections used by VRE33 library functions are mapped to internal RAM, and use the internal RAM for the stack.

(3) When mapping VRE33 library program code to external memory areas, reduce the number of wait states incurred when accessing that area to one or none, as often as possible. Make sure to use 16-bit wide memory.

Information on running sample programs using the DMT33004 and DMT33AMP boards are provided for reference in the Appendix.

4 VRE33 Tool Reference

This section describes the functions of each VRE33 tool and how to use them.

4.1 Outline of VRE33 Tools

VRE33 tools are PC software used to create and evaluate speech recognition dictionary data, which when completed is downloaded to a E0C33 Family chip. All tools are 32-bit applications executable from a DOS prompt, and will run under Windows 95, Windows NT 4.0, or later Windows versions. (For detailed information on operating environments, see Section 2.1, "Operating Environment. ")
All VRE33 tools and related files are located in the "vretool" folder (directory). The configuration of VRE33 tools and the procedure for creating recognition dictionary data are shown in Figure 4.1.1.

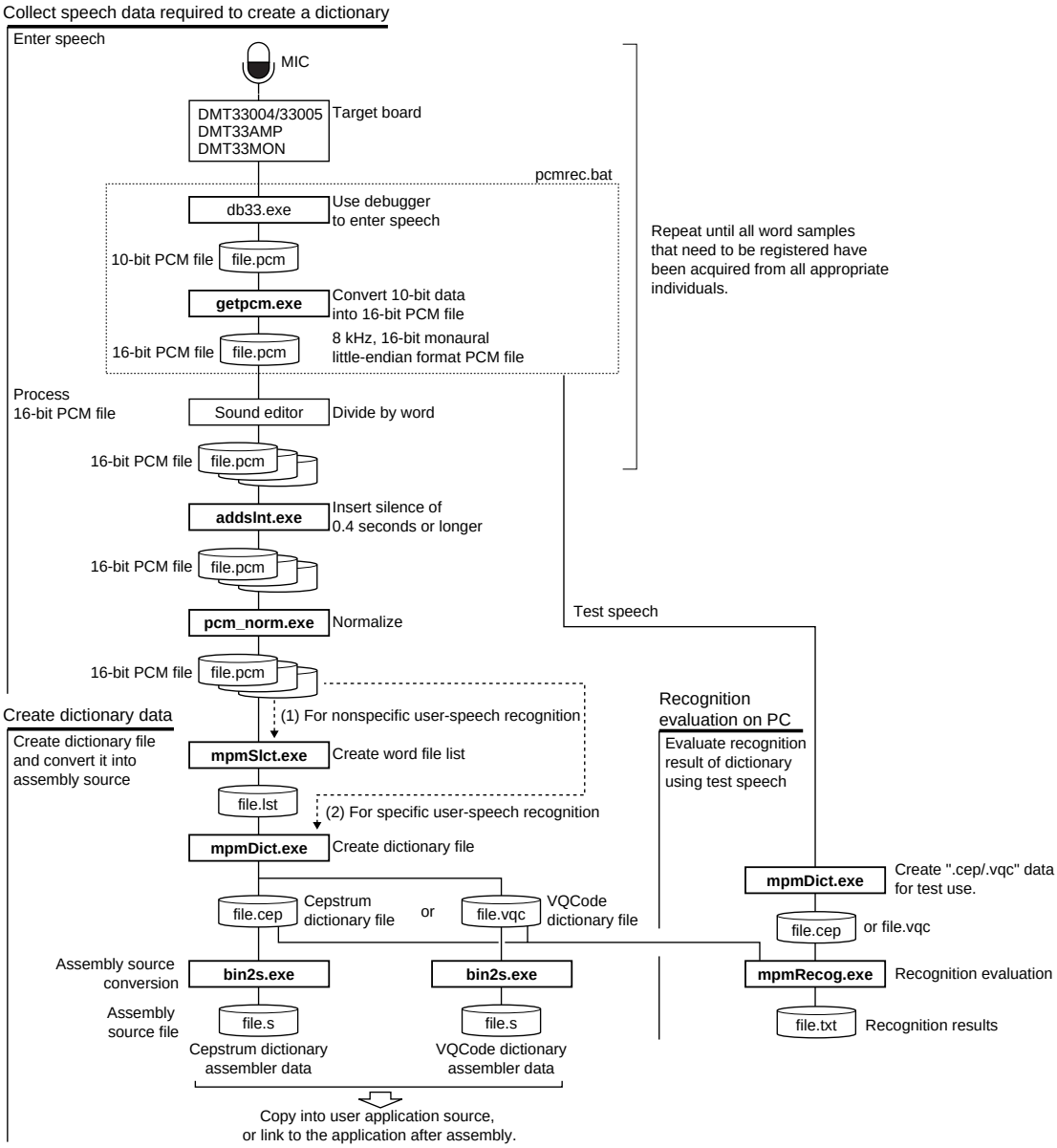


Figure 4.1.1 Flowchart for Creating Speech Recognition Directory Data

Note: Unless otherwise specified, VRE33 tools handle PCM files in 8-kHz, signed 16-bit, little-endian row data format.

The recognition dictionary data creation tools consist of a series of programs for processing speech files (pcm), converting them into dictionary data, and evaluations. They are also used to generate assembly source files for the E0C33. All tools are 32-bit applications executable from a DOS prompt, and can also be used from a batch file. The tools are listed in Table 4.1.1.

Table 4.1.1 List of Speech Recognition Directory Data Creation Tools

Tool	Description
mpmDict.exe	Creates recognition dictionary data from 16-bit PCM files.
mpmRecog.exe	Tool for evaluating recognition performance using created recognition dictionary data.
mpmSlct.exe	Creates a list of files to be entered into "mpmDict" and "mpmRecog".
bin2s.exe	Converts recognition dictionary data files into assembly source files.
getpcm.exe *	Converts 10-bit PCM data sampled on DMT boards (E0C33 chip) into 16-bit PCM files.
pcm_norm.exe	Normalizes 16-bit PCM data by adjusting it to a specified maximum amplitude.
addslnt.exe	Appends silent data of a specified length to the 16-bit PCM file.

* The source code for this tool is located in the "vretools\src\" directory. You can use it when developing applications with VRE33.

4.2 Description of Tools

This section describes the function of each VRE33 tool and explains how each is used.
Start each tool from the DOS prompt. Usage is displayed when you start a tool without specifying command line parameters. In the command line explanation, [ ] denotes optional parameters (may be omitted). Parameters in *italic* indicate an appropriate value or file name.

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

4.2.1 getpcm.exe

Function: Converts 10-bit PCM data sampled on the E0C33 chip (DMT33xxx board) to 16-bit PCM files by shifting the data 6 bits. You can change the number of shifted bits by including an optional parameter.

Usage: `DOS>getpcm [-h or -?] [-s X] input.pcm output.pcm↵`

Parameters:

input.pcm	Input file name (VOX33 10-bit PCM file)
output.pcm	Output file name (16-bit PCM file)
`-h` or **`-?`**	Display Usage (optional).
`-s` ***X***	Shift the input data left by *X* bits (optional). The effective value of *X* is 0 to 6. If this option is omitted, data is shifted left 6 bits.

Example:
```
DOS>getpcm word1_10.pcm word1_16.pcm
```

Reference: Located in the "vretools\utility\rec104\" directory (for DMT33004) or "vretools\utility\rec208\" directory (for DMT33005) is the batch file "pcmrec.bat", used to create 16-bit PCM files with speech data entered with the DMT33004/33005 + DMT33AMP + DMT33MON.

4.2.2 addslnt.exe

Function: Adds data for silence of a specified length to the input PCM file after its last line.

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

Parameters:

infile.pcm	Input file name (16-bit PCM file)
outfile.pcm	Output file name (16-bit PCM file)
NumOfSilent	Number of silent data items (samples) (optional) If this option is unspecified, the default value of 128 samples (equivalent to one packet) is assumed.

Example:
```
DOS>addslnt hello.pcm hello0.pcm 2560
```
Creates "hello0.pcm" by adding data for silence of 2560 samples (equivalent to 20 packets) after "hello.pcm".

Reference: The recognition dictionary data creation tool "mpmDict.exe" determines that a word is finished when it encounters data for silence of a given length. By default, this length is 20 packets equivalent, but can be changed by specifying an optional parameter. For PCM files that have almost silence after words, add data for silence of 2,560 samples (20 packets) in length.

4.2.3 pcm_norm.exe

Function: Changes the amplitude of speech data in 16-bit PCM files to a specified magnitude. Values that can be represented by signed 16 bits range from -32768 (SHORT_MIN) to +32767 (SHORT_MAX). In this program, as you convert the amplitude of the input speech, specify the maximum amplitude of the input speech data by a percent relative to SHORT_MAX.

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

Parameters:

input.pcm	Input file name (16-bit PCM file)
output.pcm	Output file name (16-bit PCM file)
`-r` ***XXX***	Coefficient of normalization (optional) Specify the amplitude of 16-bit PCM speech data by a percent relative to the maximum width of the amplitude. For *XXX*, enter a positive value ranging from 0.0 to 100.0. When creating recognition dictionary data, specify 100. If this option is omitted, the maximum amplitude of output sound is set to 90%. Always insert a space between -r and *XXX*.
`-c`	Read the "amp.rto" file (optional) The "amp.rto" file in the current directory is read in as the program adjusts the amplitude. This option is used during VOX compression, and is not used when creating recognition dictionary data.

Example:
```
DOS>pcm_norm -r 100 input.pcm output.pcm
```
Change to 100%

4.2.4 mpmSlct.exe

Function: By specifying a relative path and a file name including the wildcard, this program outputs a list of relevant files located in the specified directory and the subdirectories below it to the standard output device. Use this program to create a list of PCM files necessary to create dictionary data or a list of dictionary data required for recognition evaluation.

Usage: `DOS>mpmSlct [-a] path&filename [> outfile]↵`

Parameters:

path&filename	Relative path and file name (wildcard "*" can be used)
outfile	Output file name (text file) To output a list to a file, use the DOS redirect function.
`-a`	Absolute path specification (optional) The absolute path is added to the searched file names when they are listed. If this option is omitted, a relative path is added.

Example:

```
DOS>mpmSlct a*.pcm
```

Displays a list of PCM files beginning with "a" present in the current directory and that directory's subdirectories.

```
a1.pcm
a2.pcm
ab.pcm
   :
```

```
DOS>mpmSlct ..\*.cep > cep.lst
```

Displays a list of files with the extension ".cep" present in the root directory and the subdirectories of the root directory, and writes the list to the file "cep.lst".

```
DOS>Type cep.lst
..\dict\word1.cep
..\dict\word2.cep
..\dict\word3.cep
         :
```

Note: The executable "mpmSlct.exe" cannot select files on a network drive that is logically regarded as a disk partition. To create a list of data files on a network, assign a directory to the network drive, or copy the files to a local drive.

4.2.5 mpmDict.exe

Function: Accepts 16-bit PCM data entered for each word to create recognition dictionary data.

Usage: `DOS>mpmDict [options] [infile.pcm/cep] outfile.cep/vqc↵`

Parameters:

infile.pcm/cep	Input file name Specify when converting a single file. When creating Cepstrum dictionary data, specify a 16-bit PCM file (.pcm). For VQCode (when -vqc is specified), both Cepstrum dictionary data files (.cep) and 16-bit PCM files (.pcm) can be specified.
outfile.cep/vqc	Output file name (recognition dictionary data file) ".cep" = Cepstrum data file; ".vqc" = VQCode data file. PCM files are output when the -o option is specified.
[***options***]	The following options can be specified (may be omitted). When specifying multiple options, separate each option with one or more spaces.
`-f` ***filename***	Enters the text file that contains a list of PCM files during creation of dictionary data. When one phrase or expression has been gathered from multiple people, use this option to enter a list of those files. The file list is in the format shown below, with both relative and absolute paths accepted. (path\)`file1.pcm` (path\)`file2.pcm` `:` (path\)`fileN.pcm` This file can be created using "mpmSlct.exe".
`-vqc`	Creates recognition dictionary data for VQCode. Can only be used for specific user-speech recognition.
`-cep`	Creates recognition dictionary data for Cepstrum. Can be omitted, since Cepstrum is selected by default.
`-a`	Turns off the automatic word delimiting function. If this option is omitted, the function is turned on, the default value.
`-o`	Outputs automatically delimited data as PCM files. This option is effective in cases where you enter one PCM file (-f not specified). When this option is specified, the specified output file name is applied to the PCM file to be output, and a PCM file is created in place of a dictionary data file.
`-m` ***val***	Automatic delimiting option 1 (default value = 15) Specify the number of packets (in units of 16 ms) for a waveform section from the detection point to the point at which you want it to be recognized as speech.
`-s` ***val***	Automatic delimiting option 2 (default value = 20) Specify the amplitude value for the waveform start point.
`-d` ***val***	Automatic delimiting option 3 (default value = 20) Specify the amplitude value for the waveform end point.
`-e` ***val***	Automatic delimiting option 4 (default value = 20) Specify the number of samples (in units of 16 ms) for a period of silence from the end of a waveform to the point at which speech is finished.

Example: `DOS>mpmDict -f thanks.lst thanks.cep`
The recognition dictionary data "thanks.cep" is created from all the data written in the PCM data list "thanks.lst".

Reference: About automatic delimiting parameters

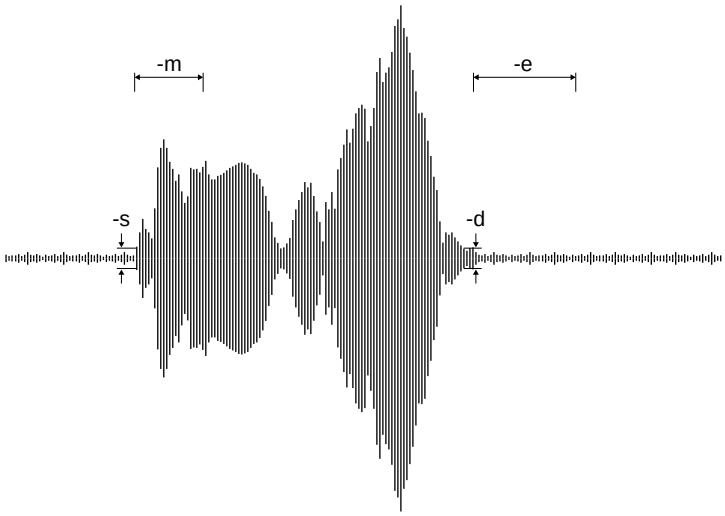


Figure 4.2.5.1 Automatic Delimiting Parameters

As shown above, the options -s and -m specify the amplitude threshold and waveform length required to determine that a waveform is speech. The larger the values, the greater the possibility that the beginning of an instance of speech will be truncated, or that short words will not be separated from one another. If values are small, noise may be processed as data.
Options -d and -e specify the threshold at which to denote the end of an instance of speech and the length of a period of silence. For words such as "three," the last half of which is not accented, you can specify 10 to 15 for the -d option to ensure that the words are recognized as distinct. However, if the values for these options are too small, words with low emphasis in the middle may be split, with the middle incorrectly assumed to be the end of a word. In noise-prone environments, the program may also fail to delimit words. For most situations, the optimum values are the default values. Use the default values unless they lead to difficulties with your application.

Note: Depending on the data, words cannot be delimited automatically, in which case the following message is displayed.
`Warning: Cannot process file` *filename* `- Skip this data.`

In this case, reprocess the data as described below and turn off the automatic delimiting function (by specifying -a) before running "mpmDict.exe".
• Increase the amplitude using a sound editor.
• Cut silent and noisy sections from the speech data using a sound editor.

4.2.6 mpmRecog.exe

Function: Using the recognition dictionary data created, and by entering Cepstrum or VQCode test data, this program evaluates how well speech is recognized. The evaluation results are output to a specified file in text format.

Usage: DOS>**mpmRecog** ***dictfile infile.cep/vqc outfile.txt***↵

Parameters:

dictfile Dictionary file for recognition (text file)
This file contains a list of created recognition dictionary data files. The file list is in the format shown below, with both relative and absolute paths accepted. It also allows including of Cepstrum and VQCode data.

```
(path\)word1.cep
(path\)word2.vqc
(path\)word3.cep
        :
```

This file can be created using "mpmSlct.exe".

infile.cep/vqc Input file name (Cepstrum or VQCode data file)
This is the data file used to test recognition performance. This file is derived from speech data for testing (16-bit PCM file) by processing it with "mpmDict.exe", just as when you create recognition dictionary data. Only one word may be tested at a time.

outfile.txt Output file name (text file)
Evaluation results (see the usage example) are output to this file.

Example:

```
DOS>mpmRecog cep.dic test.cep result.txt
```

The test data "test.cep" is compared against all recognition dictionary data written in "cep.dic", and the results are output to "result.txt". The evaluation results stored in the output file are in the following format:
Example:

```
001 0x0003c606 ..\dict\thanks.cep
003 0x0012c4d6 ..\dict\hello.cep
002 0x0008c2fe ..\dict\morning.cep
```

The contents of each line are as follows:
<rank> <distance> <recognition dictionary data file name>

The <rank> is assigned numbers from 001 in order of <distance>, beginning with the smallest. The <distance> represents the difference in features between the test data and recognition dictionary data. The smaller the value, the closer the similarity. The above example indicates that the input data was closest to "thanks.cep".

The range of <distance> in which words are assumed to be the same must be determined after examining the actual evaluation results.

4.2.7 bin2s.exe

Function: Converts binary files (cep file, vqc file) into a text file in E0C33 assembly source format. Since results by default are output to the standard output device (stdout), use the DOS redirect function to save the results to a file.

Usage: DOS>**bin2s** [**-l** ***symbol***] ***infile.cep/vqc*** **>** ***outfile.s***↵

Parameters:

infile.cep/vqc	Input file name (binary file)
outfile.s	Output file name (assembly source file)
-l ***symbol***	Definition of assembler symbol name (optional) If this option is omitted, the input file name is used as a symbol name.

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

```
DOS>bin2s word1.cep > word1.s
DOS>type word1.s
      .global word1
      .align 2
word1:
      .byte   0x33 0x20 0x24 0x00 0xa9 0x09 0xe5 0x03
      .byte   0x9c 0xfe 0x95 0xfe 0xf7 0xfe 0x3d 0x01
      .byte   0xe6 0x01 0x0b 0xff 0xa7 0x00 0x14 0x00
                        :
      .byte   0xf8 0xff 0x3a 0x00
; total 724 bytes data
DOS>
```

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

```
DOS>bin2s -l thanks word1.cep > word1.s
DOS>type word1.s
      .global thanks
      .align 2
thanks:
      .byte   0x33 0x20 0x24 0x00 0xa9 0x09 0xe5 0x03
      .byte   0x9c 0xfe 0x95 0xfe 0xf7 0xfe 0x3d 0x01
      .byte   0xe6 0x01 0x0b 0xff 0xa7 0x00 0x14 0x00
                        :
      .byte   0xf8 0xff 0x3a 0x00
; total 724 bytes data
DOS>
```

Note: The specification of symbol names is subject to the following limitations.
- Symbol name length: Up to 32 characters
- Valid characters: a to z, A to Z, 0 to 9, _

4.2.8 Executing from a Batch File

Recognition dictionary data creation and evaluation tools are all 32-bit applications executable from a DOS prompt. You can run a processing series by creating a batch file and running it from the DOS prompt. The following is an example of processing by batch file, located in the "vretools\sample\" directory.

Example of a batch file for creating recognition dictionary data

\sample\dict1\run.bat

Lists PCM speech data having a specified name and creates Cepstrum and VQCode recognition dictionary data.

Process:
1) Create a PCM list file using "mpmSlct.exe"
2) Create Cepstrum recognition dictionary data using "mpmDict.exe"
3) Create VQCode recognition dictionary data using "mpmDict.exe"
4) Convert recognition dictionary data into an assembly source file using "bin2s.exe"

Output files:

file_name.lst	PCM list file
file_name.cep	Cepstrum recognition dictionary data
*file_name*V.vqc	VQCode recognition dictionary data
dict1.s	Assembly source file

File contents:

```
@echo off

set vrepath=..\..\bin\

rem *** parameter check ***
if "%1"==""     goto ERROR
if "%2"==""     goto ERROR

rem *** Make PCM file list ***
%vrepath%mpmSlct %1 > %2.lst

rem *** Make Cepstrum data ***
%vrepath%mpmDict -f %2.lst %2.cep
%vrepath%bin2s %2.cep >> dict1.s

set vrepath=

goto END
:ERROR
echo Usage:

:END
```

Example: `>run ..\data\men\ari*.pcm arigato`

(Executed in the "vretools\sample\dict1\" directory)
Using PCM files in the "data\man\" directory beginning "ari", this batch file generates Cepstrum the recognition dictionary data file "arigato.cep" and VQCode recognition dictionary data file "arigato.vqc". This data is converted into assembly list format, which is then added to "dictl.s" after the last line in "dictl.s". (If "dictl.s" does not exist, "dictl.s" is created.)
The global labels "arigato" (Cepstrum data) and "arigatoV" (VQCode data) are defined in "dictl.s".

```
    .global  arigato
    .align   2
arigato:
    .byte    0x33 0x20 0x24 0x00 0xa9 0x09 0xe5 0x03
    .byte    0x9c 0xfe 0x95 0xfe 0xf7 0xfe 0x3d 0x01
    .byte    0xe6 0x01 0x0b 0xff 0xa7 0x00 0x14 0x00
                                :
    .byte    0xf8 0xff 0x3a 0x00
; total 724 bytes data

    .global  arigatoV
    .align   2
arigatoV:
    .byte    0x33 0x22 0x24 0x00 0x00 0x00 0x56 0x01
    .byte    0xfe 0x02 0xfe 0x02 0xfe 0x02 0xfe 0x02
    .byte    0x1e 0x02 0xe3 0x02 0xf3 0x00 0xf3 0x00
                                :
    .byte    0x24 0x01 0xb8 0x01 0x76 0x02
; total 78 bytes data
```

The file "run.bat" must be executed for each word in each user-voice group. An example for processing this collectively is provided as "mkdict.bat".
<mkdict.bat>

```
del dict1.s
call run ..\data\men\ari*.pcm arigato
call run ..\data\men\kon*.pcm konnitiwa
call run ..\data\men\oha*.pcm ohayo
call run ..\data\men\oya*.pcm oyasumi
call run ..\data\men\tad*.pcm tadaima
```

Reference: "4.2.4 mpmSlct.exe", "4.2.5 mpmDict.exe", "4.2.7 bin2s.exe"

Example of a batch file for evaluating recognition

(1) Creating a directory

\sample\recog\mkmuldic.bat

Creates a list (text file) of Cepstrum recognition dictionary data located in a specified directory. This file is used as a dictionary file when executing "mpmRecog.exe".

Process: Create a Cepstrum data list using "mpmSlct.exe".

Output files: *file_name*CE.dic Cepstrum dictionary file (text format data list file)

File contents:

```
@echo off
rem
rem Make dictionary bat file
rem

if "%1"==""  goto ERR
if "%2"==""  goto EXE

echo Generating Cepstrum dictionary for PC.
..\..\bin\mpmSlct %1\*.cep > %2CE.dic

goto END

:EXE
echo Generating Cepstrum dictionary for PC.
..\..\bin\mpmSlct *.cep > %2CE.dic

goto END
:ERR
echo Usage:
echo     mkmuldic.bat input_file_directory output_dictionary_name
echo Example:
echo     mkmuldic.bat .\sample sample
:END
```

Example:

```
>mkmuldic ..\dict3 greet
```

(Executed in the "vretools\sample\recog\" directory)
Create a list "greetCE.dic" of Cepstrum data files present in the "sample\dict3\" directory (if the first parameter is omitted, this is the current directory).
<greetCD.dic>

```
..\dict3\ariM.cep
..\dict3\ariW.cep
        :
..\dict3\tadM.cep
..\dict3\tadW.cep
```

Reference: "4.2.4 mpmSlct.exe"

(2) Evaluating recognition

\sample\recog\recog.bat

After specifying PCM speech data for testing, this batch file evaluates how speech is recognized, using the created dictionary data.

Process:
1) Convert test speech data into Cepstrum or VQCode data, using "mpmDict.exe".
2) Execute recognition evaluation using "mpmRecog.exe".
3) Display evaluation results on-screen.

Input files:

file_name.pcm	16-bit PCM speech file for test parsing
file_name.dic	Cepstrum or VQCode dictionary file (created by mkmuldic.bat)

Output files:

result.txt	Evaluation result file

File contents:

```
@echo off
rem
rem execute recognition bat file
rem
@echo off
if "%1"==""  goto ERR
if "%2"==""  goto ERR
if "%3"==""  goto ERR
rem Generating feature parameter
..\..\bin\mpmDict %1 %2 temp.cep
if errorlevel 1 goto END1
rem Recognition
..\..\bin\mpmRecog %3 temp.cep  result.txt
if errorlevel 1 goto END
@del temp.cep > NUL
type result.txt
rem
goto END2
:ERR
echo Usage:
echo     recog.bat -cep(-vqc) input_file input_dictionary
echo Example:
echo     recog.bat -cep sample.pcm sample.dic
goto END1
:END
@del temp.cep > NUL
goto END2
:END1
echo     Warning: mpmdict cannot process file.
:END2
```

Example:

```
>recog -cep sample.pcm greetCE.dic  (Cepstrum dictionary evaluation)
>recog -vqc sample.pcm greetVQ.dic  (VQCode dictionary evaluation)
```

(Executed in the "vretools\sample\recog\" directory)
The test speech file "sample.pcm" is converted into recognition data in the specified format (Cepstrum or VQCode). Recognition performance is tested using the created dictionary. Results are output to "result.txt" in the format shown below, and can also be displayed on-screen using the TYPE command.

```
 003 0x0011d727 ..\dict3\ariM.cep
 005 0x00149970 ..\dict3\ariW.cep
                :
 006 0x00150b52 ..\dict3\tadM.cep
 009 0x0019869f ..\dict3\tadW.cep
```

Reference: "4.2.5 mpmDict.exe", "4.2.6 mpmRecog.exe"

5 VRE33 Library Reference

This section gives a detailed explanation of VRE33 library functions, as well as precautions regarding their use.

5.1 Outline of VRE33 Library

Functional outline

The VRE33 library is a set of speech-recognition processing functions in srf33 library format, used after linking functions to the target application. The following functions can be executed by calling the required function from the target application:

- Nonspecific user-speech recognition using quantified characteristics data (Cepstrum recognition dictionary) created on a PC
- Specific user-speech recognition, for which quantified characteristics data for a single individual (Cepstrum or VQCode recognition dictionary) can be registered on a PC

* VRE33 speech recognition is based on the recognition of isolated words.

This package also contains C source files for top-level functions based on VRE33 library functions and assembly source files for initialization purposes, all or part of which may be copied for use into the target application. These function sets permit easy implementation of speech-recognition processing functions in your system.

Program structure

Figure 5.1.1 illustrates the structure of the application programs.

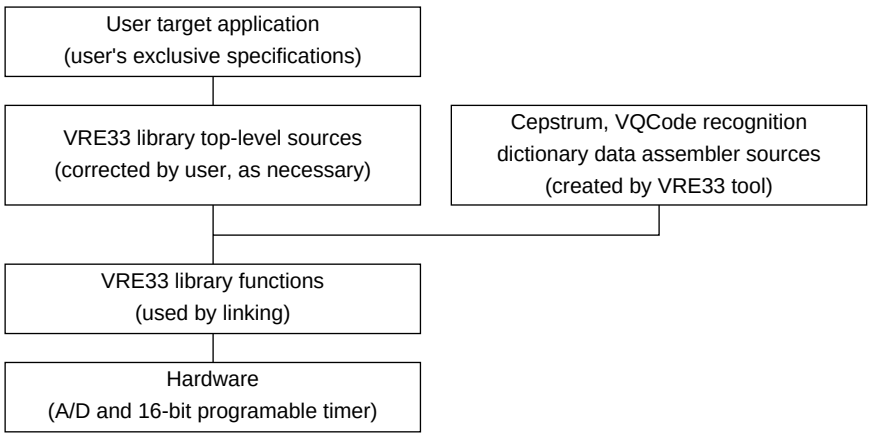


Figure 5.1.1 Program Structure

VRE33 Library Configuration

All of the VRE33 library and related files are located in the "vrelib" folder (directory). The contents of the "vrelib" folder are given below.

vrelib ..... VRE33 library-related
- readme.txt — VRE33 library supplementary explanation, etc. (in English)
- readmeja.txt — VRE33 library supplementary explanation, etc. (in Japanese)

lib ..... VRE33 library directory
- vre.lib — VRE33 library
- sl104.lib — Input/output library for E0C33A104
- sl208.lib — Input/output library for E0C33208
- memMesa.o, memAsm.o, mpmFt.o, mpmVq.o — Objects retrieved from vre.lib for high-speed operation

include ..... Header file directory for VRE33 library functions
- vre.h — VRE33 header file

src ..... Library source directory
- vretop.c — VRE33 top-level functions
- vrecache.c — VRE33 internal RAM cache functions

hardsrc ..... Hardware dependent source directory
- Listen.s — Listen.o source (E0C33A104)
- LisAD.s — LisAD.o source (E0C33A104)
- Speak.s — Speak.o source (E0C33A104)
- SpkDA.s — SpkDA.o source (E0C33A104)
- Lis208.s — Lis208.o source (E0C33208)
- Lis208AD.s — Lis208AD.o source (E0C33208)
- Spk208.s — Spk208.o source (E0C33208)
- Spk208PW.s — Spk208PW.o source (E0C33208)
- slintr.def

(Refer to these sources when changing timer A/D or D/A channel ports.)

smpl104 ..... DMT33004 sample program directory

smpl208 ..... DMT33005 sample program directory

(For detailed information on the configuration of sample programs and how to use them, see "readme.txt" or "readmeja.txt" in "vrelib".)

∗ The configuration for the top-level and library functions is described later.

5.2 Hardware Resources and Initialization

Note: If you want to use other channels or ports in addition to the playback/recording hardware resources shown below, change the sources in the "hardsrc\" directory. However, program operation is not guaranteed after such changes are made. For detailed information on each hardware resource, see the Technical Manual included with the microcomputer used in your system.

Hardware resources used by input/output libraries (sl104.lib, sl208.lib)

Input/output libraries for the E0C33A104 (sl104.lib) and E0C33208 (sl208.lib) use the chip's internal hardware resources, shown below. For this reason, these hardware resources cannot be used in the user target application.

- A/D converter (channel 0) and all related control registers
- K60 port and all related control registers
- 16-bit programmable timer (timer 0) and all related control registers
- A/D conversion-completed interrupt
  Make sure the LisIntr0( ) function is entered at the A/D conversion-completed interrupt vector address. The interrupt level is set to 4 by the LisOpen( ) function.
  Example: `.word  LisIntr0  ; Vector No.64 (ADC)`

Operating clock

This library assumes a value of 20 MHz (typ.) for high-speed (OSC3) clock frequency. For the E0C33208, operation at 40 MHz (PLL in x2 mode) can be supported as specified by a compile option (CLOCK40).

Memory

The following describes precautions concerning memory:

- Make sure all BSS sections used by the VRE33 library are mapped to internal RAM.
- Use internal RAM for the stack.
- Make sure the external memory area containing VRE33 library program code is accessed within one or no wait states, and use 16-bit wide memory.

The VRE33 library uses the following memory sizes:

- Internal RAM: Approx. 5K bytes (including BSS section, internal RAM cache, and stack)
- Internal ROM: Approx. 27K bytes (including top-level functions and output library)
- External RAM: Not needed for recognition alone.

The recognition dictionary data requires the following per-second memory sizes:

- Cepstrum: $64 \times 10 \times 2$ bytes + 4-byte header = 1,284 bytes/second
- VQCode: 64×2 bytes + 6-byte header = 134 bytes/second

5.3 Top-level Functions

The top-level functions "vretop.c" are C source files used to help implement each function, and are implemented using VRE33 library functions. Table 5.3.1 lists the top-level functions.

Table 5.3.1 List of Top-level Functions

Function name	Description
int vreRecognise( )	Recognizes speech in real-time
void vreRecogInit( )	Initializes processing for rejection (determination of mismatch)
int vreMakeDictionary( )	Acquires characteristics data from input speech (to create dictionary data)
int mpmRecognition( )	Compares characteristics data to dictionary data

You can link the source file directly to the user application source, or paste in only those functions actually used. In this case, include the header file from the "include\" directory into the user application.

Note: VRE33 library functions use the CPU's internal R8 register. When linking VRE33 library including the top-level functions (vretop.c), you cannot use the -gp option of instruction extender ext33 (optimization by global pointer/R8).

5.3.1 Compile Options

When compiling the source file for top-level functions, you can specify the compile option described below. Specify the name as necessary when compiling (by using the -D option of gcc33).

CLOCK40

Define this name when you want to run the E0C33208 at 40 MHz. In addition to specifying this option, you also need to set the E0C33208 chip's PLL mode to x2 mode. Unless this option is set, the source file is compiled for operation at 20 MHz. The sample program for the DMT33005 has been compiled with this option set.

When compiling a program for the E0C33A104, leave this option unspecified.

5.3.2 Defining the Dictionary

The following shows how to define recognition dictionary data created on a PC for use in top-level functions. You must first include or link recognition dictionary data to the program after creating the data itself with "mpmDict.exe" and converting it into assembly source format with "bin2s.exe". If you created the dictionary data "hello" and "goodbye", for example, define them in the C source as shown below:

```
#include "vre.h"

extern const short hello[];
extern const short goodbye[];

#define GREET_NUM   (2)

/* dictionary table (Cepstrum) */
const short *Greet[GREET_NUM] = {
    hello,
    goodbye
};
```

If user-voice groups have been divided into men and women, for example, the recognition dictionary data for the same word will differ for men and women. In this case, the dictionary tables become as shown below:

```
extern const short hello_men[];
extern const short hello_women[];
extern const short goodbye_men[];
extern const short goodbye_women[];

#define GREET_NUM   (4)

/* dictionary table */
const short *Greet[GREET_NUM] = {
    hello_men;
    hello_women;
    goodbye_men;
    goodbye_women;
};
```

5.3.3 Structure of Quantified Characteristics Data

The VRE33 library supports two types of data structures for quantified speech characteristics – Cepstrum and VQCode. For both, all constituent elements are 2 bytes and are handled as short-type arrays.

Cepstrum data

Speech characteristics are quantified by encoding by 20 bytes per frame. Although this provides high accuracy, it also results in high data volume. This type of data can be used for both specific and nonspecific user-speech recognition.

* Frame: Processing unit of speech recognition, which is 16 ms for 8 kHz sampling

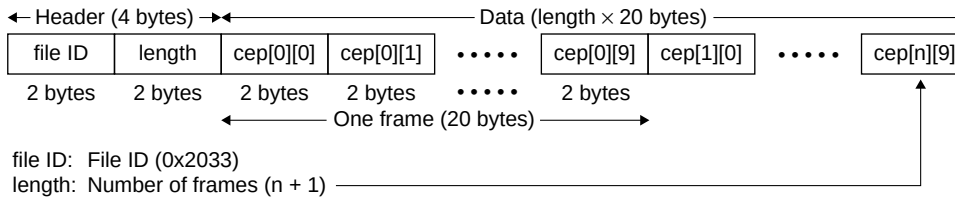


Figure 5.3.3.1 Structure of Cepstrum Data

VQCode data

The quantified speech characteristics (Cepstrum frames) are quantized into 2-byte codes using a predefined codebook (1,024 codes). One code consists of 10 bits (expressed in 2 bytes), so that data volumes are 1/10 that of Cepstrum, but with lower accuracy. This data type can only be used for specific user-speech recognition.

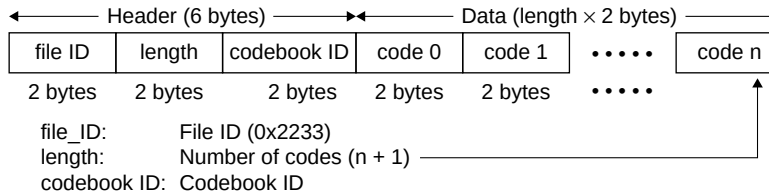


Figure 5.3.3.2 Structure of VQCode Data

5.3.4 vreRecognise( )

Function: Recognizes speech in real-time.

Format: `int vreRecognise(short *Dict[], char Mode, int DictNum, int Distance[]);`

Parameters:

`short *Dict[]`		Pointer array to quantified characteristics (dictionary)
`char Mode`		Characteristics extraction format in input data (Cepstrum, VQCode)
`int DictNum`		Number of quantified characteristics specified in dictionary
`int Distance[]`		Distance storage array

Return value:

Recognition data number in dictionary		(0 to *DictNum*) upon successful recognition
`MPM_GET_SPEECH_TIMEOUT_ERROR`	(-1)	Characteristics could not be extracted for quantification within before time-out occurred, or matching data could not be found in the dictionary.
`MPM_GET_SPEECH_BUF_FULL_ERROR`	(-3)	The characteristics extraction buffer was exceeded.
`MPM_GET_SPEECH_NOT_IN_TIME_ERROR`	(-4)	Input outstrips speed of speech recognition.
`MPM_RECOG_THRESHOLD_ERROR`	(-5)	Rejected.
`MPM_GET_SPEECH_CONTINUE`	(-6)	Speech interval continues without a break.

Description: Data entered from the 10-bit A/D converter is delimited into speech intervals in real-time to extract speech characteristics for quantification. The extracted characteristics data and the recognition data in *Dict[]* are compared, and a value giving the recognition data closest in distance to the characteristics data is returned. Recognition data numbers in *Dict[]* begin with 0.

The *Dict* array stores the pointer to each Cepstrum or VQCode format recognition dictionary data.

To specify *Mode*, use the following constants:

`MPM_GET_FEATURE`	(1)	When converting input speech into Cepstrum data
`MPM_GET_CODE`	(0)	When converting input speech into VQCode data

For *DictNum*, specify the number of recognition data entries in the dictionary.

The *Distance* array stores distance information between input-converted characteristics data and each recognition data entry in the dictionary. There should be as many array elements as *DictNum*. If you specify 0 for *Distance*, nothing is stored in the array. If an error is found in dictionary data, one of the following error codes is placed in the corresponding element of the *Distance* array:

`MPM_NOT_MATCHING_ERROR`	(-3)	The data volumes to be compared exceed memory capacity, or number of frames between the input data and dictionary data differ significantly.
`MPM_FILE_FORMAT_ERROR`	(-4)	The data format (Cepstrum or VQCode) differs.

Note:

- Make sure the *Dict* array stores only pointers to the data type (Cepstrum or VQCode) specified by parameter *Mode*. Both types of data cannot coexist.
- The time-out for recognition processing is set to 10 seconds by the parameter to mmpGetSpeechFeature(). Change if necessary.
- Internal variables for this function must always be placed in the stack (internal RAM).
- The size of the internal variable array *feature* used to store the quantified characteristics of input speech by default is set to MPM_FEATURE_LEN (approx. 1 second for Cepstrum). When using only VQCode data, you can change this to MPM_VQCODE_LEN (size about 1/10).
  `short feature[MPM_FEATURE_LEN];` → `short feature[MPM_VQCODE_LEN];`
- This function calls another top-level function, mpmRecognition(). When using this function after copying it into your program, always be sure to copy the mpmRecognition() function as well.

5.3.5 vreRecogInit( )

Function: Initializes processing for rejection (determination of mismatch)

Format: `void vreRecogInit(int threshold);`

Parameters:

`int threshold`	Upper distance threshold for determination of successful recognition (0x0 to 0x7fffffff)

Return value: None

Description: This function sets the threshold level of the reject function.
Normally, the recognition engine returns a number for the dictionary data that is closest in distance (most similar) to the quantified characteristics data to be recognized. A dictionary data number is always returned as a recognition result, even for an entirely different word, mere noise, or tics in speech (e.g. "er – " or "umm"). The reject function prevents such incorrect recognition.
Use *threshold* to set the upper-limit value of the distance for determination of successful recognition. If the distance between the quantified characteristics data to be recognized and the dictionary data closest to it is smaller than the threshold set here, the recognition function assumes that the characteristics data coincides with the dictionary data and returns the dictionary data number. If the distance is greater than the threshold, the recognition function assumes that the characteristics data does not coincide and returns the error code MPM_RECOG_THRESHOLD_ERROR(-5).
The appropriate threshold varies with the environment used, the performance of the analog system (e.g. microphone and amp), and the contents of the recognition word sets. Entering a word similar to but not actually found in the recognition dictionary to compare the distance at which erroneously recognized and the distance at which it is correctly recognized. This helps to determine the appropriate threshold. If *threshold* is set to 0x7fffffff (defined as MPM_MAX_RECOG_ THRESHOLD in "vre.h"), the reject function is disabled.

5.3.6 vreMakeDictionary( )

Function: Acquires characteristics data from input speech (to create dictionary data).

Format: `int vreMakeDictionary(short *Dict, int Mode, int MaxLen);`

Parameters:

`short *Dict`		Buffer in which quantified characteristics data is saved
`char Mode`		Characteristics extraction format in input data (Cepstrum, VQCode)
`int MaxLen`		Size (short type size) of buffer (Dict)

Return value:

`MPM_SUCCESS`	(1)	Succeeded

Description: The data entered from the 10-bit A/D converter is separated between speech intervals in real-time to acquire the quantified characteristics data with which to create a dictionary. This data is used to create dictionary data for specific user-speech recognition on the actual product. After calling mpmGetSpeechInit(), this function repeatedly calls mpmGetSpeechFeature() until it receives quantified characteristics data. For additional information, see mpmGetSpeechFeature().

Use *Dict* and *MaxLen* to specify the start address and size of the buffer used to save quantified characteristics data.

To specify *Mode,* use the following constants:

`MPM_GET_FEATURE`	(1)	When converting input speech into Cepstrum data
`MPM_GET_CODE`	(0)	When converting input speech into VQCode data

Note:

- The processing time-out is set to 3 seconds by the parameter for mpmGetSpeechFeature(). Change if necessary.
- Internal variables for this function must always be placed in the stack (internal RAM).
- By default, the size of the internal variable array *feature* used to store the quantified characteristics of input speech is set to MPM_FEATURE_LEN (approx. 1 second for Cepstrum). When using only VQCode data, you can change this to MPM_VQCODE_LEN (size about 1/10).
  `short feature[MPM_FEATURE_LEN];` → `short feature[MPM_VQCODE_LEN];`

5.3.7 mpmRecognition( )

Function: Compares characteristics data and dictionary data.

Format:
```
int mpmRecognition(short *InData, short *Dict[], int DictNum,
int Distance[]);
```

Parameters:

`short *InData`		Quantified characteristics data to be recognized (compared)
`short *Dict[]`		Pointer array to dictionary data
`int DictNum`		Number of dictionary data entries
`int Distance[]`		Distance storage array

Return value: Recognition data number in dictionary (0 to *DictNum*) upon successful recognition

`MPM_ERROR`	(-1)	All dictionary data in Dict[] resulted in errors
`MPM_RECOG_THRESHOLD_ERROR`	(-5)	Rejected

Description: *InData* and recognition data in *Dict[]* are compared, and a value for the dictionary data closest to *InData* is returned. Recognition data values in *Dict[]* begin with 0.

The *Dict* array stores the pointer to the recognition dictionary data in the same format (Cepstrum or VQCode) as in InData.

For *DictNum*, specify the number of recognition data entries in the dictionary.

The *Distance* array stores the distance between *InData* and each recognition data entry in the dictionary. There must be as many array elements as *DictNum*. If you specify 0 for *Distance*, nothing is stored in the array. If an error is found in the dictionary data, one of the following error codes is placed in the corresponding element of the *Distance* array:

`MPM_NOT_MATCHING_ERROR`	(-3)	The data volumes to be compared exceed memory capacity, or number of frames between the input data and dictionary data differ significantly.
`MPM_FILE_FORMAT_ERROR`	(-4)	The data format (Cepstrum or VQCode) differs.

Note:
- Make sure the *Dict* array stores only pointers to the same data type (Cepstrum or VQCode) as in *InData*. Both types of data cannot coexist.
- Internal variables for this function must always be placed in the stack (internal RAM).

5.4 VRE33 Library Functions

The VRE33 library "vre.lib" contains object files that include functions required for speech-recognition processing, while the libraries "sl104.lib" and "sl208.lib" contain object files that include functions required for speech input. Speech recognition is implemented by linking these libraries to the user application. The VRE33 library functions are listed in Table 5.4.1.

Table 5.4.1 List of VRE33 Library Functions

vre.lib

Function name	Description
mpmGetSpeechInit( )	Initializes speech delimiting processing
mpmGetSpeechFeature( )	Inputs speech and delimits words
mpm_calc_distance( )	Calculates distance between two quantified characteristics
mpmIsMpmFeature( )	Checks the data format (Cepstrum)
mpmIsVqCode( )	Checks the data format (VQCode)
mpmGetBufferLen( )	Gets buffer size

\src\vrecache.c

Function name	Description
mpm_feature_cache( )	Transfers object into internal RAM cache (used for mpmGetSpeechFeature)
mpm_recog_cache( )	Transfers object into internal RAM cache (used for mpm_calc_distance)

<Reference> **sl104.lib, sl208.lib** (used for speech input from A/D converter)

Function name	Description
LIS_SAMPLING( )	Gets 16-bit timer reload value (macro)
LisInit( )	Initializes internal library variables
LisOpen( )	Opens input channel
LisClose( )	Closes input channel
LisStart( )	Starts voice input
LisHalt( )	Halts voice input
LisAppend( )	Appends the voice to input data queue
LisRoom( )	Gets number of entries in queue
LisQueue( )	Gets number of entries waiting for input
LisIsRunning( )	Checks input status
LisOnDone( )	Enters recording call-back function
LisOnEmpty( )	Enters recording-complete call-back function
LisOnNotInTime( )	Enters non-real-time operating call-back function
LisIntr0( )	Processes voice input by interrupt

Note: The VRE33 library functions use the CPU's internal R8 register. Therefore, when linking VRE33 library including the top-level functions, you cannot use the -gp option of instruction extender ext33 (optimization by global pointer/R8). Make sure that the BSS sections used by VRE33 library functions are mapped to internal RAM.

The specifications for each function are explained below. For usage examples, see the source file for top-level functions.

5.4.1 Speech-recognition Processing Functions

mpmGetSpeechInit( )

Function: Initializes speech delimiting processing.

Format:
```
int mpmGetSpeechInit(short StartThreshold, short DurThreshold,
short EndInterval, short SegMinLen, short Sampling);
```

Parameters:

`short StartThreshold`		Threshold for beginning of speech interval
`short DurThreshold`		Threshold for end of speech interval
`short EndInterval`		Length to determine completion
`short SegMinLen`		Length of minimum speech interval
`short Sampling`		Input speech sampling frequency

Return value:

`MPM_SUCCESS`	(1)	Succeeded

Description: This function initializes speech delimiting processing by setting the parameters necessary to delimit speech intervals.

For all of these parameters except *Sampling*, the following constants are defined as default values in "vre.h":

`MPM_GETSP_START_THRESHOLD`	(20)	Used to specify *StartThreshold*
`MPM_GETSP_DURATION_THRESHOLD`	(20)	Used to specify *DurThreshold*
`MPM_GETSP_END_INTERVAL`	(20)	Used to specify *EndInterval*
`MPM_GETSP_MIN_LEN`	(15)	Used to specify *SegMinLen*

For the relationship between speech waveforms and parameters, see Figure 4.2.5.1, "Automatic Delimiting Parameters". If speech cannot be separated into words using the default values on the actual machine, change them according to the waveforms for standard data.

The parameter *Sampling* is found from the LIS_SAMPLING macro in "vre.h". For cases when the CPU clock = 40 MHz and the sampling rate = 8 kbps, make the following specifications:
`LIS_SAMPLING(40000000, 8000)`

mpmGetSpeechFeature( )

Function: Inputs speech and delimits words.

Format:
```
short mpmGetSpeechFeature(int Mode, short *Buffer, int BufferLen,
int SamplePoints);
```

Parameters:

`int Mode`		Characteristics extraction format in input data (Cepstrum, VQCode)
`short *Buffer`		Buffer in which to save characteristic-extracted data
`int BufferLen`		Buffer size (short type size)
`int SamplePoints`		Time-out duration (number of samples)

Return value:

`MPM_SUCCESS`	(1)	Succeeded
`MPM_GET_SPEECH_CONTINUE`	(0)	Speech interval continues without a break
`MPM_GET_SPEECH_TIMEOUT_ERROR`	(-1)	Characteristics could not be extracted within the time specified by *SamplePoints*
`MPM_GET_SPEECH_BUF_FULL_ERROR`	(-3)	The *BufferLen* size was exceeded
`MPM_GET_SPEECH_NOT_IN_TIME_ERROR`	(-4)	Input speed exceeds pace of speech recognition

Description: This function parses data entered from the 10-bit A/D converter into speech intervals in real-time to extract speech characteristics for quantification. It returns to the main routine after storing the extracted characteristics data in *Buffer*. If no speech interval appears within the time specified by *SamplePoints*, an error is assumed, in which case MPM_GET_SPEECH_TIMEOUT_ERROR(-1) is returned.

To specify *Mode*, use the following constants:

`MPM_GET_FEATURE`	(1)	When converting input speech into Cepstrum data
`MPM_GET_CODE`	(0)	When converting input speech into VQCode data

To specify the *Buffer* size, the following constants are defined in "vre.h". Cepstrum requires ten times the buffer size of VQCode.

`MPM_MAX_WORD_LEN`	`(64)`	Number of frames in characteristics extraction
`MPM_FEATURE_HEADER_SIZE`	`(2)`	Cepstrum data header size (short type)
`VQ_CODE_HEADER_SIZE`	`(3)`	VQCode data header size (short type)
`MPM_FEATURE_LEN`	`(MPM_MAX_WORD_LEN*10+MPM_FEATURE_HEADER_SIZE)`	Buffer size for Cepstrum (short type)
`MPM_VQCODE_LEN`	`(MPM_MAX_WORD_LEN+MPM_VQ_CODE_HEADER_SIZE)`	Buffer size for VQCode (short type)

MPM_MAX_WORD_LEN defines the maximum number of frames in characteristics extraction. One frame is 16 ms for 8K sampling; the value 64 allows characteristics to be extracted for up to 1 second per word (0.016 seconds × 64 = approx. 1 second).

Specify *SamplePoints* by a number of samples. In the case of 8K sampling, for example, specify 10 seconds as 10*8000.

mpm_calc_distance( )

Function: Calculates the distance between two quantified characteristics.

Format:
```
int mpm_calc_distance(short *InData, short *RefData);
```

Parameters:

`short *InData`	Pointer to input characteristics data
`short *RefData`	Pointer to referenced characteristics data (dictionary data)

Return value: Distance between two quantified characteristics upon successful recognition

`MPM_NOT_MATCHING_ERROR`	(-3)	The data volumes to be compared exceed memory capacity, or number of frames between the input data and dictionary data differ significantly.
`MPM_FILE_FORMAT_ERROR`	(-4)	The data format (Cepstrum or VQCode) differs.

Description: This function calculates the distance between the characteristics data extracted from the input speech specified by *InData* and the recognition dictionary data specified by *RefData*, and returns the result to the main routine.

mpmIsMpmFeature( )

Function: Checks the data format (Cepstrum).

Format:
```
int mpmIsMpmFeature(short *data);
```

Parameters:

`short *data`	Pointer to characteristics data

Return value: 1 for Cepstrum data
0 for any data that is not in Cepstrum format

Description: This function checks to see if *data* is in Cepstrum format.

mpmIsVqCode( )

Function: Checks the data format (VQCode).

Format:
```
int mpmIsVQCode(short *data);
```

Parameters:

`short *data`	Pointer to characteristics data

Return value: 1 for VQCode data
0 for any data that is not in VQCode format

Description: This function checks to see if *data* is in VQCode format.

mpmGetBufferLen( )

Function: Gets buffer size.

Format: `int mpmGetBufferLen(short *data);`

Parameters: `short *data` Pointer to characteristics data

Return value: Data length

Description: This function returns the length (short type) of *data*, not including the header to the main routine.

mpm_feature_cache( )

Function: Transfers object into the internal RAM cache (used for mpmGetSpeechFeature).

Format: `void mpm_feature_cache();`

Parameters: None

Return value: None

Description: This function transfers the objects (mpmAsm.o and mpmMesa.o) that need to be run at high speed in order to execute mpmGetSpeechFeature() into the cache area specified by a linker command file. When using an internal RAM cache, always call this function before calling mpmGetSpeechFeature(). Since this function is written in "vrelib\src\vrecache.c", copy all of the source contents into, or link to the user application after compiling. For additional information on defining the linker command file, see Section 5.5, "Techniques for Speeding Up".

mpm_recog_cache( )

Function: Transfers object into the internal RAM cache (used for mpm_calc_distance).

Format: `void mpm_recog_cache(short *InData);`

Parameters: `short *InData` Pointer to characteristics data

Return value: None

Description: This function transfers objects that need to run at high speed in order to execute mpm_calc_distance() into the cache area specified by a linker command file. The objects to be transferred are "mpmAsm.o" and "mpmFt.o" when *InData* is Cepstrum or "mpmAsm.o" and "mpmVq.o" when *InData* is VQCode. When using an internal RAM cache, always call this function before calling mpm_calc_distance(). Since this function is written in "vrelib\src\vrecache.c", copy all of the source contents into, or link to the user application after compiling. For additional information on defining the linker command file, see Section 5.5, "Techniques for Speeding Up".

5.4.2 Input (Listen) Functions

This section explains the voice input functions in "sl104.lib" and "sl208.lib".

LIS_SAMPLING( )

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

Format: `LIS_SAMPLING(CpuClock, SamplingRate)`

Parameters:

`CpuClock`	CPU clock frequency
`SamplingRate`	Sampling rate

Return value: 16-bit timer reload value

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

LisInit( )

Function: Initializes internal library variables

Format: `void LisInit(void);`

Parameters: None

Return value: None

Description: This function clears the internal variables used by the library to 0.

LisOpen( )

Function: Opens input channel

Format: `unsigned char *LisOpen(int Channel, int ReloadValue);`

Arguments:

`int Channel`	Channel number
`int ReloadValue`	16-bit timer set value

Return value:

Terminated normally	LisParams pointer corresponding to the opened channel
Terminated abnormally	0

Description: This function opens the specified input channel with a specified sampling rate. The LisParams value returned by this function is used as an argument for other input (Lis) functions.
For ReloadValue, specify the value acquired by the LIS_SAMPLING macro.
In the following cases, the function fails to open and returns 0.
- When the specified channel is already open (For input, only channel 1 can be opened.)
- When an unavailable channel is specified
- When the reload value exceeds 16 bits

To change the channels used, modify the source in the "hardsrc\" directory.

LisClose( )

Function: Closes input channel

Format: `int LisClose(unsigned char *LisParams);`

Parameters:

`unsigned char`	`*LisParams`	LisParams pointer (return value of LisOpen)

Return value:

Terminated normally	Other than 0
Terminated abnormally	0

Description: This function closes the specified input channel. If the specified channel is not open, it returns 0.

LisStart( )

Function: Starts voice input

Format:
```
int LisStart(unsigned char *LisParams);
```

Parameters:

`unsigned char`	`*LisParams`	LisParams pointer (return value of LisOpen)

Return value:

Terminated normally	Other than 0
Terminated abnormally	0

Description: This function starts the operation to input voice data in a specified channel. If the specified channel is not open, it returns 0.

LisHalt( )

Function: Stops voice input

Format:
```
int LisHalt(unsigned char *LisParams);
```

Parameters:

`unsigned char`	`*LisParams`	LisParams pointer (return value of LisOpen)

Return value:

Terminated normally	Other than 0
Terminated abnormally	0

Description: This function stops the operation to input voice data in a specified channel. If input in the specified channel has not been started by LisStart( ), it returns 0.

LisAppend( )

Function: Appends data to input data queue

Format:
```
int LisAppend(unsigned char *LisParams, void *Buffer, int Length);
```

Arguments:

`unsigned char`	`*LisParams`	LisParams pointer (return value of LisOpen)
`void`	`*Buffer`	Pointer to the data to be entered
`int`	`Length`	Data size

Return value:

Terminated normally	Other than 0
Terminated abnormally	0

Description: This function appends the input buffer to the input queue of a channel specified by LisParams. If input in the specified channel has not been started by LisStart( ) or there is no free entry in the queue, 0 is returned.

LisRoom( )

Function: Gets the number of remaining entries in the queue

Format:
```
int LisRoom(unsigned char *LisParams);
```

Parameters:

`unsigned char`	`*LisParams`	LisParams pointer (return value of LisOpen)

Return value: Number of available entries

Description: This function returns the number of available remaining entries in the input queue.
When this function is called immediately after opening an input channel, it shows the maximum number of available entries.
The value returned during voice input operation is as follows:
(Maximum number of entries) - (Number of queued entries) - (Number of entries that are not called back)

LisQueue( )

Function: Gets the number of entries waiting for input

Format:
```
int LisQueue(unsigned char *LisParams);
```

Parameters:

`unsigned char`	`*LisParams`	LisParams pointer (return value of LisOpen)

Return value: Number of entries waiting for input

Description: This function returns the number of entries waiting for input in the input queue.
The value returned during voice input operation is as follows:
(Number of queued entries) - (Number of entries that are not called back) - (Number of entries that are called back)

LisIsRunning( )

Function: Checks input status

Format:
```
int LisIsRunning(unsigned char *LisParams);
```

Parameters:

`unsigned char`	`*LisParams`	LisParams pointer (return value of LisOpen)

Return value:

When input operation is under way	Other than 0
When input operation has halted	0

Description: This function returns a value indicating whether input operation in the specified input channel is under way.

LisOnDone( )

Function: Enters the recording call-back function

Format:
```
int LisOnDone(unsigned char *LisParams, void *Callback);
```

Arguments:

`unsigned char`	`*LisParams`	LisParams pointer (return value of LisOpen)
`void`	`*Callback`	Pointer to the call-back function to be entered

Return value: Pointer to the original call-back function

Description: This function enters the function in a specified input channel that is called back when recording voice data. The call-back function has the following format:
```
void Callback(unsigned char *LisParams, void *Buffer, int Length)
```

LisOnEmpty( )

Function: Enters the recording-complete call-back function

Format:
```
int LisOnEmpty(unsigned char *LisParams, void *Callback);
```

Arguments:

`unsigned char`	`*LisParams`	LisParams pointer (return value of LisOpen)
`void`	`*Callback`	Pointer to the call-back function to be entered

Return value: Pointer to the original call-back function

Description: This function enters the function in a specified input channel that is called back upon completion of recording. The call-back function has the following format:
```
void Callback(unsigned char *LisParams)
```

LisOnNotInTime( )

Function: Enters the non-real-time operating call-back function

Format:
```
int LisOnNotInTime(unsigned char *LisParams, void *Callback);
```

Arguments:

`unsigned char`	`*LisParams`	LisParams pointer (return value of LisOpen)
`void`	`*Callback`	Pointer to the call-back function to be entered

Return value: Pointer to the original call-back function

Description: This function enters the function in a specified input channel that is called back if voice data cannot be recorded in real time. The call-back function has the following format:
```
void Callback(unsigned char *LisParams, void *Buffer, int Length)
```

LisIntr0( )

Function: Processes voice input by interrupt

Format:
```
void LisIntr0(void);
```

Parameters: None

Return value: None

Description: This function processes voice input by an interrupt. Use this function only as an interrupt vector.

5.5 Techniques for Speeding Up Processing

The library processing speed can be increased from a few percent to around 30% by mapping several objects from "vre.lib" to internal memory before executing them. Use the U-section function of the linker to map objects to internal memory. Write the following in the linker command file:

```
-objsym                                          ; Creates object symbol
-section VRECACHEASM                             ; Creates section symbol
-section VRECACHE
-ucode VRECACHEASM {..\..\lib\mpmAsm.o}          ; Maps into U-section
-ucode VRECACHE {..\..\lib\mpmMesa.o ..\..\lib\mpmFt.o ..\..\lib\mpmVq.o}
```

* Objects that can be mapped to internal memory are stored in the "lib\" directory:
mpmAsm.o, mpmMesa.o, mpmFt.o, mpmVq.o

When using the top-level functions for speech recognition, you do not need to change the sources, since the above object transfer routines are written in top-level functions. The transfer routine itself is written in "vrecache.c", so you can link the file after compiling.
When creating a program that calls library functions directly, call mpm_feature_cache() before calling mpmGetSpeechFeature(), and mpm_recog_cache() before calling mpm_calc_distance() to transfer the object codes required for each into internal memory.
Example: To call mpmGetSpeechFeature()

```
mpm_feature_cache();
error_code = mpmGetSpeechFeature(Mode, feature, MPM_FEATURE_LEN, 10*8000);
```

Note: When using the VRE33 library in combination with libraries such as the VOX33, the internal RAM may fall short of capacity. In such cases, speech recognition can be performed without using an internal RAM cache. However, if speech needs to be recognized in real-time, you may need to reduce the number of words registered in the dictionary. To disable the internal RAM cache, specify the compile option "-DNO_IRAM_CACHE" when compiling "vrecache.c". The U-section definition in the linker command file is also not required.

5.6 Library Performance

The explanation here assumes that speech recognition occurs under the following conditions:

Operating frequency:	20 MHz
CODE section:	External ROM (with 1 wait state)
BSS section and stack:	Internal RAM (no wait states)
Internal RAM cache:	Enabled
Dictionary word set:	5 words ("Arigato" (Thank you), "Oyasumi" (Goodnight), "Konnichiwa" (Good afternoon), "Ohayo" (Good morning), "Tadaima" (Hello, I'm here).)

Under these conditions, if one word is about 0.7 seconds long, comparison between the data input by mpmRecognition() and each word in the recognition dictionary data requires about 18 ms. For five words, this amounts to about 90 ms (= 18 ms × 5). Generally, the duration of time in which you feel speech is recognized in real-time is about 0.5 seconds. Under the above conditions, up to 30 words can be recognized simultaneously (18 ms × 30 words = 540 ms).

Without an internal RAM cache, processing speed is about 30% slower. Each increment increase in the number of wait states in external memory access reduces processing speed approximately 15%.

The maximum length of a recognizable word by default is 1 second. This value can be changed using MPM_MAX_WORD_LEN in "vre.h". MPM_MAX_WORD_LEN times 16 ms is the maximum length of a recognizable word. Note that this change affects the buffer size to be placed in the stack by vreRecognise() and mpmRecognition(), and the stack size used by the library consequently changes.

5.7 Combined Use with VOX33 Library

The VRE33 library can be used in combination with speech output generated by the VOX33 voice compression/expansion library. For usage examples, see the sample in the "smplXXX\sample3\" directory.

For economical use of internal RAM, the buffer *SplisBuf* is used for both input and output. For this reason, specify the -d option of the linker when linking the library. Because the library object "mpmGetSp.o" includes *SplisBuf*, separate it from the library when you link. It can be restored into the object file with the librarian lib33 -x option.
Example: `lib33 -x vox.lib mpmGetSp.o`

The internal RAM cache of VOX33 is not used.
For the purpose of real-time processing, make sure the number of wait states when accessing external memory containing the library code is one or less, and use 16-bit wide memory.

Combined use with VOX33 may exceed internal RAM capacity. In this case, stop using the VRE33 internal RAM cache. Disabling the internal RAM cache increases the processing time of mpmRecognition(), forcing you to reduce the number of words to be recognized simultaneously. It also prevents you from acquiring characteristics data in VQCode format using mpmGetSpeechFeature(). To disable the internal VRE33 RAM cache, specify the compile option "-DNO_IRAM_CACHE" when compiling "vrecache.c", and remove any specification of internal RAM cache from the linker command file.

You can also map the BSS sections of some objects in either library to external RAM. When the BSS sections of either library are placed in external RAM for real-time processing, make sure the number of wait states for external RAM access is one or less, and use 16-bit wide memory.

5.8 Program Examples

The following explains how to create speech recognition routines, using the sample programs located in the "smpl104\" directory (for E0C33A104) and "smpl208\" directory (for E0C33208) as examples.

Setting interrupt vectors

For the A/D converter interrupt vector, set the address of the LisIntr0() function. In the sample program, the trap table is set in "common\table.s". To use this file, set the start address of the handler routine that corresponds to the trap vector address required for your application.

Example: common\table.s

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

    .word   Boot            ;   0 Reset
    .word   exception       ;   1 reserved
    .word   exception       ;   2 reserved
    .word   exception       ;   3 reserved
    .word   exception       ;   4 Zero Div.
    .word   exception       ;   5 reserved
    .word   exception       ;   6 Address Error
    .word   NMI             ;   7 NMI
                        :
    .word   LisIntr0        ;  64 ADC
                        :
exception:

    jp 0
```

Boot routine

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

Example: common\boot.s

```
#define STACK_INIT      0x00001800
#define PSR_INIT        0x00000110     ; InitIntr. Level 1, Intr. enable

.global Boot
Boot:
        xld.w   %r4,STACK_INIT
        ld.w    %sp,%r4                ; set STACK
        xld.w   %r4,PSR_INIT
        ld.w    %psr,%r4               ; set PSR
;
        xcall   InitBusCtrl
        xcall   InitCPUClock

        xcall   main
```

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

Speech recognition routine

The following program example is "sample4.c", located in the "sample4" directory. This program uses dictionary data created separately for men and women to perform nonspecific user-speech recognition. The dictionary data is Cepstrum data "dict3.s", created by the "vretools\sample\dict3\" sample files.

Example: sample4\sample4.c

```
/*
 *  sample4.c : VRE33 demonstration with stdio main function
 *
 *  Separate dictionary data by speaker group (men and women).
 *  Voice feature parameters are different in speaker group.
 *
 */

#include <stdio.h>
#include "vre.h"                                                    (*1)

extern const short ariM[];                                          (*2)
extern const short ariW[];
extern const short konM[];
extern const short konW[];
extern const short ohaM[];
extern const short ohaW[];
extern const short oyaM[];
extern const short oyaW[];
extern const short tadM[];
extern const short tadW[];

#define GREET_NUM   (10)

/* dictionary table (Cepstrum) */
const short *Greet[GREET_NUM] = {                                   (*3)
    ariM,
    ariW,
    konM,
    konW,
    ohaM,
    ohaW,
    oyaM,
    oyaW,
    tadM,
    tadW
};

/* Name table for print_result */
const char *Name[GREET_NUM] = {
    "arigato men",
    "arigato women",
    "konnichiwa men",
    "konnichiwa women",
    "ohayo men",
    "ohayo women",
    "oyasumi men",
    "oyasumi women",
    "tadaima men",
    "tadaima women"
};

int   Distance[GREET_NUM];   /* Distance save buffer */
```

```
/* print recognition result and distance */
int print_result(int result_num, int Distance[], int DictNum)
{
    int i;

    _init_sys();    /* call _init_sys() in sys.c */
    _init_lib();    /* call ANSI lib initialize */

    if(result_num < 0)
        printf("result: %d\n", result_num);
    else
        printf("result: %s\n", Name[result_num]);

    for(i=0; i<DictNum; i++) {
        printf("%08x %s\n", Distance[i], Name[i]);
    }

    printf("\n");
}

main()
{
    int result_num;

    vreRecogInit(MPM_MAX_RECOG_THRESHOLD);                                    (*4)

    /* Start recognition */
    for(;;) {
        result_num = vreRecognise((short**)Greet, MPM_GET_FEATURE,            (*5)
                                  GREET_NUM, Distance);
        print_result(result_num, Distance, GREET_NUM);
    }
}
```

*1 Include "vrelib\include\vre.h" in source files that call the VRE33 library function.

*2 This defines the recognition dictionary data created by VRE33 tools as externally referenced. Write all data to be used here.

*3 This defines a dictionary array whose elements are comprised by the start address of each dictionary data in *2.

*4 This sets the threshold level for rejection in the vreRecogInit() function. The MPM_MAX_RECOG _THRESHOLD used here is the constant defined in "vre.h". Because the value is 0x7fffffff, the reject function is disabled.

*5 Using the vreRecognise() function, this performs speech recognition using the input data and Cepstrum data. Upon successful recognition, the array element number (0 or greater) of the dictionary data entry closest to the input data is returned in result_num.

The following program example is "sample2.c" in the "sample2" directory. This program creates VQCode recognition dictionary data from input speech on the product actually used to perform specific user-speech recognition.

Example: sample2\sample2.c

```
/*
 *  sample2.c : VRE33 register and recognition demonstration with LED
 *              Speaker dependent recognition, use VQCode as feature.
 *
 *  - Make dictionary data. Push REC switch (SW4) and register your voice
 *     to dictionary.
 *  - Repeat registration until push Play switch (SW3).
 *  - After that speak something. Result blink LED.
 *
 */

#include "vre.h"

#define WORD_NUM    (10)
#define BUF_LEN     (MPM_VQCODE_LEN * WORD_NUM)

short *myDict[WORD_NUM];    /* Dictonary table */                          (*1)
short DictBuf[BUF_LEN];     /* Voice Future save buffer */

main()
{
    int result_num, DictNum;

    /* Make dictionary */
    DictNum = 0;
    for(;;) {
        /* push PLAY stop make dictionary */
        if((GetEvent() & 0x3) == 1)
            break;

        /* push REC add to dictonary new data */
        if((GetEvent() & 0x3) == 2) {
            myDict[DictNum] = &DictBuf[MPM_VQCODE_LEN*DictNum];
            LedON();
            Wait(200);
            vreMakeDictionary(myDict[DictNum],MPM_GET_CODE,MPM_VQCODE_LEN); (*2)
            LedOFF();
            DictNum++;
            if(DictNum >= WORD_NUM)
                break;
        }
    }

    /* Start recognition */

    vreRecogInit(MPM_MAX_RECOG_THRESHOLD);                                 (*3)

    for(;;) {
        result_num = vreRecognise(myDict, MPM_GET_CODE, DictNum, 0);       (*4)

        if(result_num < 0) continue;

        /* blink LED */
        result_num += 1;
        while(result_num--) {
            LedON();
            Wait(2000);
            LedOFF();
            Wait(2000);
        }
    }
}
```

*1 This specifies a dictionary table. The number of words that can registered is set to 10.

*2 This extracts speech characteristics for quantification from input data using the vreMakeDictionary() function and registers the result as VQCode data in the dictionary. This operation is repeated until you press the PLAY switch on the DMT33xxx, or until a total of 10 words has been registered.

*3 This sets the threshold level for rejection in the vreRecogInit() function. The MPM_MAX_RECOG _THRESHOLD used here is the constant defined in "vre.h". Because the value is 0x7fffffff, the reject function is disabled.

*4 Using the vreRecognise() function, this recognizes speech using the input data and the VQCode data registered in the dictionary. Upon successful recognition, the array element number (0 or greater) of the dictionary data closest to the input data is returned in result_num.

5.9 Precautions

(1) VRE33 library functions use the CPU's internal R8 register. When linking the VRE33 library including top-level functions, you cannot use the -gp option for instruction extender ext33 (optimization by global pointer/R8).

(2) The following lists requirements for implementing speech-recognition processing in real-time:

- Make sure all BSS sections used by the VRE33 library are mapped to internal RAM.
- Use the internal RAM for the stack.
- When mapping VRE33 library program codes to an external memory area, reduce the number of wait states incurred when accessing that area to one or less as much as possible, and use 16-bit wide memory.
- If possible, use the internal RAM cache to execute parts of objects in internal RAM. (See Section 5.5.)

(3) The speech input unit is strongly affected by DC levels, power-supply noise, and the S/N ratio. Make sure you incorporate sufficient protective measures in the input circuit.

- Shield the lines from local 50–60 Hz noise.
- Protect the microphone to avoid picking up wind noise.
- Adjust input levels so that they are neither excessively small or large.

Appendix Verifying Operation with DMT33 Boards

This section describes how to verify the operation of speech recognition by executing a sample program using E0C33 Family demonstration tools, the DMT33004, DMT33MON, and DMT33AMP.

A.1 System Configuration Using DMT33004

A.11 Hardware Configuration

Configure the system shown in Figure A.1.1 using the DMT33004, DMT33MON, and DMT33AMP. This system allows you to recognize speech data that has been input from a microphone using the DMT33AMP.

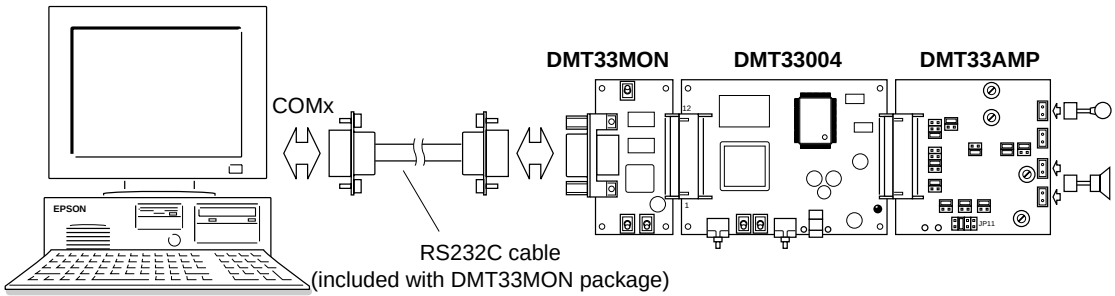


Figure A.1.1 System Configured with DMT33004, DMT33MON, and DMT33AMP

DMT33004 board

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

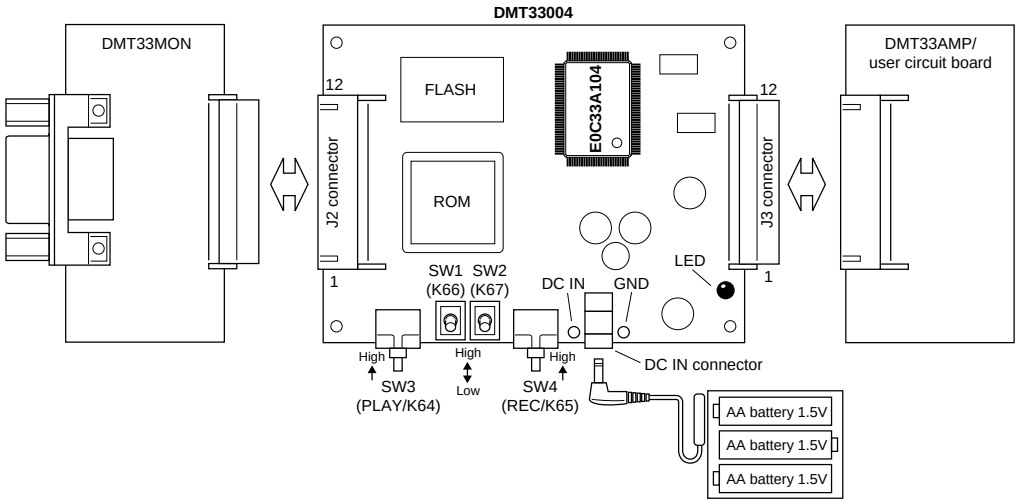


Figure A.1.2 DMT33004 Board

DMT33MON board

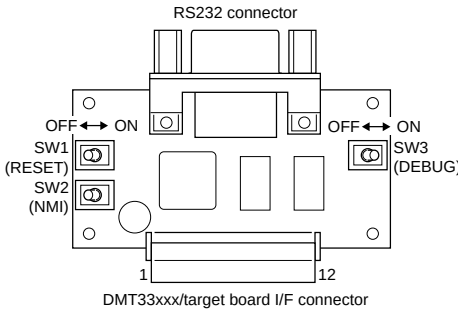


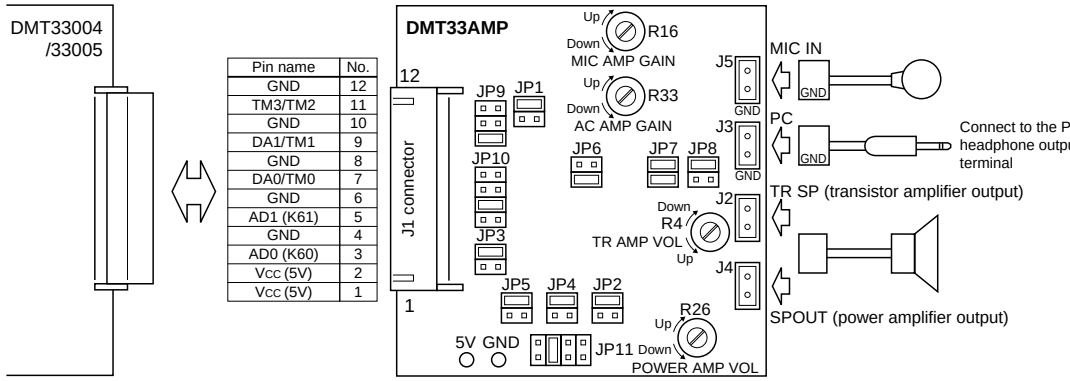
Figure A.1.3 DMT33MON Board

The DMT33MON interfaces with demonstration tools such as the DMT33004 and the user target board for a debugging monitor. By connecting the DMT33004 board to your personal computer via the DMT33MON board, you can debug programs on-board using the debugger (db33.exe) installed in your computer.

Note: For the DMT33004 board, always be sure to use the DMT33MON, which is designed to operate with a 5-V power supply. The DMT33MONLV board, which is designed to operate at 3.3 V, cannot be used.

DMT33AMP board

The DMT33AMP is an optional board that adds voice input/output functions to the DMT33004, etc. It allows voice input from a microphone and voice output from the amplifier mounted on the board. It also allows you to test the configuration (and effect) of speaker/microphone low-pass and high-pass filters, which determine sound quality.



Jumper switch

JP1 DMT / MIC
Selects the voice source to be output.
DMT: DMT33004/33005 output (default)
MIC: Microphone input of this board

JP2 DA / TM
Selects an input for the transistor amplifier circuit.
DA: DMT33004 D/A output (default)
TM: DMT33005 PWM output

JP3 DA / TM
Selects an input for the CR 2nd order filter circuit.
DA: DMT33004 D/A output (default)
TM: DMT33005 PWM output

JP4 SP / MIC
Selects a filter in the OP AMP 4th order filter circuit (for speaker and MIC).
SP: For speaker (default)
MIC: For microphone

JP5 SP / MIC
Selects an filter in the OP AMP 4th order filter circuit (for speaker and MIC).
SP: For speaker (default)
MIC: For microphone

JP6 AD1 / AD0
Selects the A/D channel on the DMT33004/33005 used to convert the MIC input.
AD0: Channel 0 (default)
AD1: Channel 1

JP7 3300pF / 1500pF
Selects a cutoff frequency in the CR 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

In systems where the DMT33AMP is used along with the DMT33004 board, the jumper switches do not need to be changed; default settings suffice.

System connections

Note: Before connecting or disconnecting the system, always turn off power to all connected boards and equipment. For other precautions regarding the handling of each board, see the "E0C33 Family Demonstration Board Manual".

1. Attach DMT33MON and DMT33AMP to the DMT33004.
2. Adjust the DMT33AMP's microphone gain to minimum.
3. Connect the microphone and speaker included with the DMT33AMP package to the DMT33AMP. A speaker is not needed for sample programs that do not involve speech output.
4. Connect the DMT33MON and PC with the RS-232C cable included with the DMT33MON package.
5. Set the DEBUG switch (SW3) on the DMT33MON to the ON position.
6. Place batteries in the battery holder attached to the DMT33004 and connect the battery holder to the DMT33004.
7. Switch on power to the PC.

A.12 Software

The PC hosting the VRE33 must have the "E0C33 Family C Compiler Package" development tools installed on it. To download your program into the DMT33004 using the debug monitor, you must have debugger (db33) ver. 1.72 or later.

A.2 Sample Program Execution Procedure

Preparatory operations before startup

Since each sample program directory "vrelib\smpl104\sampleX\" contains absolute object files in executable format, there is no need to compile or link the sample program specifically. Also provided is a batch file from which you can launch the debugger. The following explains how to start each sample program after downloading them to the DMT33004.

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

(2) Before a program can be downloaded to the DMT33004, the debug monitor must already be active. Reconfirm that the DMT33MON DEBUG switch (SW3) is set to the ON position, then reset the system using the RESET switch (SW1).

(3) Make "vrelib\smpl104\sampleX\" the current directory, and run "sample.bat" from the DOS prompt.
Example: `C:\E0C33\VRE33\VRELIB\SMPL104\SAMPLE1\>sample`

The batch file for each sample assumes that the debugger (db33) is installed in the "c:\cc33\" directory when it starts the debugger in debug monitor mode (-mon).
Example: \sample1\sample.bat

```
@echo off
start c:\cc33\db33 -p 33104_v.par -b 115200 -c sample.cmd -mon
```

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

(4) When the debugger starts, the sample program (sample.srf) is loaded into the DMT33004's RAM area (beginning with address 0x600000) by the commands in "sample.cmd".

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

The following explains how to run the sample programs and their function.

sample1

Sample1 lets you try out specific and nonspecific user-speech recognition. This sample program supports both Cepstrum and VQCode data formats, allowing you to compare their recognition accuracy. As shown below, you can select the desired data format using SW1 and SW2 of the DMT33004 board:
SW1 = low: Converts input speech into Cepstrum data.
SW1 = high: Converts input speech into VQCode data.
SW2 = low: Uses Cepstrum dictionary data to perform speech recognition.
SW2 = high: Uses VQCode dictionary data to perform speech recognition.

See Figure A.1.2 for the correct direction to push the switch to set it high or low. The setup status of the two switches are read in when the program starts after a reset. If you changed switch settings after starting the program, interrupt forcibly program execution with a forced break and execute the rstc command ([Reset Cold] button) before executing the g command [Go] button) once again.

Demonstration of nonspecific user-speech recognition

In this demonstration, dictionary data created on a PC is used to perform speech recognition. The dictionary data contains five words recorded for men – "Arigato" (Thank you), "Konnichiwa" (Good afternoon), "Ohayo" (Good morning), "Oyasumi" (Goodnight), and "Tadaima" (Hello, I'm here). This dictionary data (dict2.s) was created from a sample file in the "vretools\sample\dict2\" directory.
To run the demonstration, do the following:
(1) Execute "vrelib\smpl104\sample1\sample.bat".

(2) Bring up the [Simulated I/O] window.

(3) Execute the g command.

(4) Press the PLAY switch on the DMT33004 board.

(5) Speak one of the above words into the microphone connected to the DMT33AMP. The recognition results are displayed on the [Simulated I/O] window like the one shown below.

Example:
```
result: 5
1: 0011efd7 arigato
2: 0013ef7b konnichiwa
3: 0011304b ohayo
4: 001547a4 oyasumi
5: 00067f17 tadaima
```

The value "result:" indicates the dictionary data number closest to the speech spoken into the microphone. If your speech could not be recognized, it indicates the error code returned by vreRecognise(), which is a negative number. The hexadecimal numbers immediately after "1:" through "5:" indicate the distance between the input speech quantified characteristics and each dictionary data entry. In this example, the fifth dictionary data entry "Tadaima" (Hello, I'm here), which had the lowest distance, is the recognition result. If speech is not properly recognized, adjust the pace of your speech, or raise the DMT33AMP's microphone gain slightly before trying again.

(6) You can repeat step (5). To finish, click the [Key Break] button.

Demonstration of specific user-speech recognition

In this demonstration, dictionary data created by entering speech into the DMT33004 is used to perform speech recognition. To run the demonstration, do the following:

(1) Execute "vrelib\smpl104\sample1\sample.bat".

(2) Bring up the [Simulated I/O] window.

(3) Execute the g command.

(4) Press the REC switch on the DMT33004 board.

(5) After confirming that the DMT33004's LED indicator is lit, speak a word you want to register toward the microphone connected to the DMT33AMP. The LED will go out to indicate that the word is registered. If the LED does not go out, re-register the word. If you experience difficulty, raise the DMT33AMP's microphone gain slightly before trying again.

(6) You can register up to 10 words by repeating steps (4) and (5).

(7) When you have finished registering words into the dictionary, press the PLAY switch on the DMT33004 board.

(8) Speak one of the registered words into the microphone connected to the DMT33AMP. The recognition results are displayed on the [Simulated I/O] window like the one shown below.

Example:
```
result: 4
1: ffffffd
2: 000d8361
3: ffffffd
4: 00097701
```

The value "result:" indicates the dictionary data number closest to the speech spoken into the microphone. If your speech could not be recognized, it indicates the error code returned by vreRecognise(), which is a negative number. The hexadecimal numbers immediately after "1:" through "5:" indicate the distance between the input speech quantified characteristics and each dictionary data entry. In this example, the fourth dictionary data entry, which had the lowest distance, is the recognition result. If speech is not properly recognized, adjust the pace of your speech, or raise the DMT33AMP's microphone gain slightly before trying again.

(9) You can repeat step (8). To finish, click the [Key Break] button.

sample2

Sample2 lets you try specific user-speech recognition in VQCode data format. In this demonstration, VQCode dictionary data created by entering speech into the machine in use is used to perform speech recognition.
To run the demonstration, do the following:

(1) Execute "vrelib\smpl104\sample2\sample.bat".

(2) Execute the g command.

(3) Press the REC switch on the DMT33004 board.

(4) After confirming that the DMT33004's LED indicator is lit, speak a word you want to register toward the microphone connected to the DMT33AMP. The LED will go out to indicate that the word is registered. If the LED does not go out, re-register the word. If you experience difficulty, raise the DMT33AMP's microphone gain slightly before trying again.

(5) You can register up to 10 words by repeating steps (3) and (4).

(6) When you have finished registering words into the dictionary, press the PLAY switch on the DMT33004 board.

(7) Speak a registered word toward the microphone connected to the DMT33AMP. When recognition is successful, the LED on the DMT33004 board will flash a number of times corresponding to the number of the matching dictionary data entry. If recognition fails, the LED remains off. In this case, re-enter the word. If recognition does not occur, adjust the pace of your speech or raise the DMT33AMP's microphone gain slightly before trying again.

(8) You can repeat step (7). To finish, click the [Key Break] button.

sample3

Sample3 lets you try out nonspecific user-speech recognition and a function for outputting audible results using the VOX33 library. As with the "dict2.s" file used in sample1, the dictionary data contains five words registered for men – "Arigato" (Thank you), "Konnichiwa" (Good afternoon), "Ohayo" (Good morning), "Oyasumi" (Goodnight), and "Tadaima" (Hello, I'm here).
To run the demonstration, do the following:

(1) Execute "vrelib\smpl104\sample3\sample.bat".

(2) Execute the g command.

(3) Speak a word into the microphone connected to the DMT33AMP. When recognition is successful, the matching dictionary data word is reproduced from the speaker connected to the DMT33AMP. The reproduction data has been prepared as voice data compressed in VSX format, which is expanded by the VOX33 library when reproduced from the speaker. When recognition fails, no sound is output from the speaker. In this case, re-enter the word. If speech is not properly recognized, adjust the pace of your speech or raise the DMT33AMP's microphone gain slightly before trying again.

(4) You can repeat step (3). To finish, click the [Key Break] button.

sample4

Sample4 lets you try out nonspecific user-speech recognition using dictionary data prepared separately for men and women. The dictionary data contains five words registered separately for men and women – "Arigato" (Thank you), "Konnichiwa" (Good afternoon), "Ohayo" (Good morning), "Oyasumi" (Goodnight), and "Tadaima" (Hello, I'm here). This dictionary data (dict3.s) was created from sample files in the "vretools\sample\dict3\" directory.

To run the demonstration, do the following:

(1) Execute "vrelib\smpl104\sample4\sample.bat".

(2) Bring up the [Simulated I/O] window.

(3) Execute the g command.

(4) Speak a word into the microphone connected to the DMT33AMP. The recognition results are displayed in a [Simulated I/O] window like the one shown below.

Example:
```
result: konnichiwa men
00141476 arigato men
0016a134 arigato women
000bad70 konnichiwa men
000d4839 konnichiwa women
         :
```

The value "result:" indicates the dictionary data number closest to the speech spoken into the microphone. If recognition fails, it indicates the error code returned by vreRecognise(), which is 0 or a negative number. The hexadecimal numbers on each line show the distance between the input speech's quantified characteristics data and each dictionary data entry. In this example, "Konnichiwa" (Good afternoon) for men, with the lowest distance, is the recognition result. If recognition does not occur, adjust the pace of your speech or raise the DMT33AMP's microphone gain slightly before trying again.

(5) You can repeat step (4). To finish, click the [Key Break] button.

A.3 Making a Program

The sample programs can be changed before testing, as necessary. For reference, the *make* procedure is explained below.

Each sample directory (sample1 through sample4) contains a *make* file "sample.mak", prepared individually for each sample. When you've corrected the source, use "sample.mak" to create an object file in executable format "sample.srf". The *make* for each sample program requires the source files in the "common\" directory, as well as files in the "sample\" directory.

Procedure for executing *make*

1. Change "vrelib\sample\sampleX\" to the current directory.
2. Enter the following command at the DOS prompt (for sample1).
   ```
   C:\E0C33\VRE33\VRELIB\SAMPLE\SAMPLE1>make -f sample.mak
   ```

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

Notes:
- VRE33 library functions use the CPU's internal R8 register. When linking the VRE33 library, avoid using the -gp option for instruction extender ext33 (optimization by global pointer/R8).
- The optional VOX33 library is required for *make* in sample3.

A.4 Entering Speech for Dictionary Data Creation/Evaluation on DMT Boards

A.4.1 Entering Speech to Create Recognition Dictionary Data Using pcmrec.bat

You can also use the DMT board system as a speech input tool when you create recognition dictionary data on a PC. Here, we explain the method for entering speech using "pcmrec.bat" in the "vretools\utility\rec104\" directory (for DMT33004) or "vretools\utility\rec208\" directory (for DMT33005). The following explanation assumes use of a DMT33004 system.

(1) Referring to "System connections" in Section A.1.1, connect the DMT33004, DMT33AMP, DMT33MON, and PC. Set SW1 and SW2 on the DMT33004 to the Low position (see Figure A.1.2), and turn on the power to each equipment.

(2) Before the speech input program can be downloaded to the DMT33004, the debug monitor must already be active. Reconfirm that the DMT33MON DEBUG switch (SW3) is set to the ON position, then reset the system with the RESET switch (SW1).

(3) Make "vretools\utility\rec104\" the current directory, then run "pcmrec.bat" from the DOS prompt.
Example: `C:\E0C33\VRE33\VRETOOLS\UTILITY\REC104\>pcmrec test.pcm rec10.cmd`

The first parameter is the file name to be output. The second parameter is the debugger command file name that determines recording time. Six command file types are then provided as below, with the number in each file name indicating the recording time in seconds (10 to 60 seconds).
`rec10.cmd, rec20.cmd, rec30.cmd, rec40.cmd, rec50.cmd, rec60.cmd`

(4) The debugger db33 starts, and after initial settings by the command file, enters standby status.

(5) Do the following to record speech and upload the data:
Each switch on the DMT33004 has the following functions:

SW3 PLAY:	Reproduces speech
SW4 REC:	Records speech
SW1:	Clears recorded content
SW2:	Uploads recorded data
NMI (DMT33MON):	Stops play or recording

- Press the REC switch to enter recording mode. The LED on the DMT33004 lights. Speak a word you want to register in the dictionary into the microphone connected to the DMT33AMP. Press the NMI switch to stop recording. The LED goes out. Press the REC switch again to start from the point at which you stopped recording. Recording stops and the LED goes out when the time specified by the command file expires.
- When you press the PLAY switch while idle, the recorded data is reproduced from the beginning. Use the NMI switch to stop playback.
- To clear recorded contents, temporarily set SW1 High while idle, then set it Low again. Press the REC switch to rerecord from the beginning.
- To upload the recorded data into the PC, temporarily set SW2 High while idle, then set it Low again.

(6) After uploading, db33 is automatically terminated, and "getpcm.exe" is subsequently executed. This creates the 8K-sampling, 16-bit little-endian PCM data file "test.pcm".

For information on performing the following operations, see the explanation in sections beginning with Section 3.1.3.

A.42 Evaluating Speech Recognition Using recogXXX.bat

To evaluate the recognition dictionary data created on a PC, you can use the DMT system as an instance of a speech input tool. Here, we explain a method for entering speech and evaluating speech recognition using "vretools\sample\recog\recog104.bat" (for DMT33004) or "vretools\sample\recog\recog208.bat" (for DMT33005). The following explanation assumes use of a DMT33004 system.

(1) Referring to "System connections" in Section A.1.1, connect the DMT33004, DMT33AMP, DMT33MON, and PC. Switch on power to each piece of equipment.

(2) Before the speech input program can be downloaded to the DMT33004, the debug monitor must already be active. Reconfirm that the DMT33MON DEBUG switch (SW3) is set to the ON position, then reset the system using the RESET switch (SW1).

(3) Make "vretools\sample\recog\" the current directory, then run "recog104.bat" from the DOS prompt.
Example: `C:\E0C33\VRE33\VRETOOLS\SAMPLE\RECOG\>recog104 -cep greetCep.dic`

The first parameter is the data format of quantified characteristics (specify -cep for Cepstrum or -vqc for VQCode), and the second parameter is the file name for the dictionary data (list file of recognition dictionary data) to be evaluated. The "greetCep.dic" prepared as sample contains five words – "Arigato" (Thank you), "Konnichiwa" (Good afternoon), "Ohayo" (Good morning), "Oyasumi" (Goodnight), and "Tadaima" (Hello, I'm here), with data for men and women listed for each entry. This dictionary data is located in the "vretools\sample\dict3" directory.

(4) The debugger db33 starts and enters speech input mode, after initial settings by the command file.

(5) The LED on the DMT33004 lights to indicate speech input mode. Speak the word you want recognized toward the microphone connected to the DMT33AMP. Speech input mode continues for 3 seconds, after which time the LED goes out.

(6) After 3 seconds, db33 is automatically terminated. Subsequently, "mpmDict.exe" is executed to convert the input speech data into the specified recognition data format, which is then evaluated for recognition by "mpmRecog.exe". Recognition results are output to "result.txt" and also displayed on-screen.
Example:
```
003 0x0011d727 ..\dict3\ariM.cep
005 0x00149970 ..\dict3\ariW.cep
008 0x0017de02 ..\dict3\konM.cep
010 0x001df2f6 ..\dict3\konW.cep
001 0x000c487a ..\dict3\ohaM.cep
007 0x00175ac3 ..\dict3\ohaW.cep
004 0x0014160c ..\dict3\oyaM.cep
002 0x0010f595 ..\dict3\oyaW.cep
006 0x00150b52 ..\dict3\tadM.cep
009 0x0019869f ..\dict3\tadW.cep
```

A.5 When Using the DMT33005 Board

The DMT33005 uses an E0C33208 for its CPU. Sample programs for the DMT33005 are provided separately from those for the DMT33004 (E0C33A104) described above. To perform testing with the DMT33005, use the programs in the "vrelib\smpl208\" and the "vretools\utility\rec208\" directories. Operation verification can be performed in the same way as for the DMT33004. Use DSW1 on the DMT33005 as set by default (PLL x2/20 to 30 MHz input).

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

When creating applications for the E0C33208, you must link "vrelib\lib\sl208.lib". For additional 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

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 NOVEMBER 1999, Printed in Japan Ⓜ A