NEC

User's Manual

μSAP77016-B07

MP3 Audio Decoder Middleware

Target Device

μPD77110
μPD77113A
μPD77114
μPD77115

Document No. U15134EJ4V0UM00 (4th edition)
Date Published September 2002 NS CP(N)

[MEMO]

Windows is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries.

- **The information in this document is current as of September, 2002. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC's data sheets or data books, etc., for the most up-to-date specifications of NEC semiconductor products. Not all products and/or types are available in every country. Please check with an NEC sales representative for availability and additional information.**
- No part of this document may be copied or reproduced in any form or by any means without prior written consent of NEC. NEC assumes no responsibility for any errors that may appear in this document.
- NEC does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC semiconductor products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC or others.
- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of customer's equipment shall be done under the full responsibility of customer. NEC assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC endeavours to enhance the quality, reliability and safety of NEC semiconductor products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC semiconductor products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment, and anti-failure features.
- NEC semiconductor products are classified into the following three quality grades:
  "Standard", "Special" and "Specific". The "Specific" quality grade applies only to semiconductor products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of a semiconductor product depend on its quality grade, as indicated below. Customers must check the quality grade of each semiconductor product before using it in a particular application.

  "Standard": Computers, office equipment, communications equipment, test and measurement equipment, audio and visual equipment, home electronic appliances, machine tools, personal electronic equipment and industrial robots

  "Special": Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-crime systems, safety equipment and medical equipment (not specifically designed for life support)

  "Specific": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.

  The quality grade of NEC semiconductor products is "Standard" unless otherwise expressly specified in NEC's data sheets or data books, etc. If customers wish to use NEC semiconductor products in applications not intended by NEC, they must contact an NEC sales representative in advance to determine NEC's willingness to support a given application.

  (Note)

  (1) "NEC" as used in this statement means NEC Corporation and also includes its majority-owned subsidiaries.

  (2) "NEC semiconductor products" means any semiconductor product developed or manufactured by or for NEC (as defined above).

M8E 00.4

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC product in your application, please contact the NEC office in your country to obtain a list of authorized representatives and distributors. They will verify:

- Device availability
- Ordering information
- Product release schedule
- Availability of related technical literature
- Development environment specifications (for example, specifications for third-party tools and components, host computers, power plugs, AC supply voltages, and so forth)
- Network requirements

In addition, trademarks, registered trademarks, export restrictions, and other legal issues may also vary from country to country.

NEC Electronics Inc. (U.S.)
Santa Clara, California
Tel: 408-588-6000
800-366-9782
Fax: 408-588-6130
800-729-9288

NEC do Brasil S.A.
Electron Devices Division
Guarulhos-SP, Brasil
Tel: 11-6462-6810
Fax: 11-6462-6829

NEC Electronics (Europe) GmbH
Duesseldorf, Germany
Tel: 0211-65 03 01
Fax: 0211-65 03 327

- **Sucursal en España**
  Madrid, Spain
  Tel: 091-504 27 87
  Fax: 091-504 28 60

- **Succursale Française**
  Vélizy-Villacoublay, France
  Tel: 01-30-67 58 00
  Fax: 01-30-67 58 99

- **Filiale Italiana**
  Milano, Italy
  Tel: 02-66 75 41
  Fax: 02-66 75 42 99

- **Branch The Netherlands**
  Eindhoven, The Netherlands
  Tel: 040-244 58 45
  Fax: 040-244 45 80

- **Branch Sweden**
  Taeby, Sweden
  Tel: 08-63 80 820
  Fax: 08-63 80 388

- **United Kingdom Branch**
  Milton Keynes, UK
  Tel: 01908-691-133
  Fax: 01908-670-290

NEC Electronics Hong Kong Ltd.
Hong Kong
Tel: 2886-9318
Fax: 2886-9022/9044

NEC Electronics Hong Kong Ltd.
Seoul Branch
Seoul, Korea
Tel: 02-528-0303
Fax: 02-528-4411

NEC Electronics Shanghai, Ltd.
Shanghai, P.R. China
Tel: 021-6841-1138
Fax: 021-6841-1137

NEC Electronics Taiwan Ltd.
Taipei, Taiwan
Tel: 02-2719-2377
Fax: 02-2719-5951

NEC Electronics Singapore Pte. Ltd.
Novena Square, Singapore
Tel: 253-8311
Fax: 250-3583

J02.4

Major Revisions in This Edition

Page	Description
p.27	Modification of description in **2.2.5 mp3_GetErrorStatus function**

The mark ★ shows major revised points.

PREFACE

Target Readers

This manual is for users who design and develop μPD77016 Family application systems.

μPD77016 Family is the generic name for the μPD7701x family (μPD77015, 77016, 77017, 77018, 77018A, 77019), the μPD77111 Family (μPD77110, 77111, 77112, 77113A, 77114, 77115) and the μPD77210 Family (μPD77210, 77213). However, this manual is for μPD77110, 77113A, 77114, and 77115 devices.

Purpose

The purpose of this manual is to help users understand the supporting middleware when designing and developing μPD77016 Family application systems.

Organization

This manual consists of the following contents.

CHAPTER 1 OVERVIEW
CHAPTER 2 LIBRARY SPECIFICATIONS
CHAPTER 3 INSTALLATION
CHAPTER 4 SYSTEM EXAMPLES
APPENDIX SAMPLE PROGRAM SOURCE

How to Read This Manual

It is assumed that the reader of this manual has general knowledge in the fields of electrical engineering, logic circuits, microcontrollers, and the C language.

To learn about μPD77111 Family hardware functions
→ Refer to ***μ*PD77111 Family User's Manual Architecture**.

To learn about μPD77016 Family hardware functions
→ Refer to ***μ*PD77016 Family User's Manual Instruction**.

Conventions

Data significance:	Higher digits on the left and lower digits on the right
Active low representation:	$\overline{XXX}$ (overscore over pin or signal name)
Note:	Footnote for item marked with **Note** in the text
Caution:	Information requiring particular attention
Remark:	Supplementary information
Numerical representation:	Binary ... XXXX or 0bXXXX
	Decimal ... XXXX
	Hexadecimal ... 0xXXXX

Related Documents The related documents listed below may include preliminary versions. However, preliminary versions are not marked as such.

Documents Related to Devices

Document Name / Part Number	Pamphlet	Data Sheet	User's Manual		Application Note
			Architecture	Instructions	Basic Software
μPD77110	U12395E	U12801E	U14623E	U13116E	U11958E
μPD77111					
μPD77112					
μPD77113A		U14373E			
μPD77114					
μPD77115		U14867E			

Documents Related to Development Tools

Document Name		Document No.
RX77016 User's Manual	Function	U14397E
	Configuration Tool	U14404E
RX77016 Application Note	HOST API	U14371E

Caution The related documents listed above are subject to change without notice. Be sure to use the latest version of each document for designing.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 OVERVIEW

1.1 Middleware

Middleware is the name given to a group of software that has been tuned so that it draws out the maximum performance of the processor and enables processing that is conventionally performed by hardware to be performed by software.

The concept of middleware was introduced with the development of a new high-speed processor, the DSP, in order to facilitate operation of the environments integrated in the system.

By providing appropriate speech codec and image data compression/decompression-type middleware, NEC is offering users the kind of technology essential in the realization of a multimedia system for the μPD77016 Family, and is continuing its promotion of system development.

μSAP77016-B07 is middleware that provides MP3 decoding functions.

1.2 MP3 Audio Decoder

In this manual, audio signal encoding and decoding methods MPEG-1 Audio Layer-3 and MPEG-2 Audio Layer-3 LSF (Low Sampling Frequency) are called MP3. The relevant standards for these are ISO/IEC 11172-3 and ISO/IEC 13818-3.

μSAP77016-B07 complies with this decoding method. The compressed data is data that encodes a digital signal converted to 16-bit linear PCM data after sampling an analog signal at frequencies shown in Table 1-1.

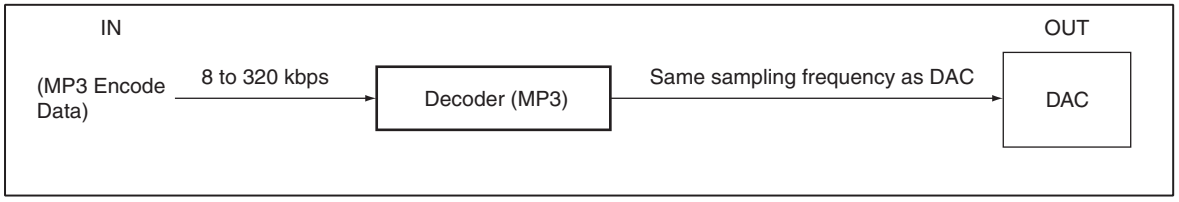
Table 1-1. Sampling Frequencies

MPEG-1 Audio Layer-3 [Hz]	MPEG-2 Audio Layer-3 LSF [Hz]
44100	22050
48000	24000
32000	16000

1.2.1 Decoder outline

Figure 1-1 is a sample configuration of decoding using μSAP77016-B07.

Figure 1-1. Sample Configuration of Decoding



(1) MP3 Encode Data

MP3 Encode Data is data sampled at frequencies shown in Table 1-1 (16-bit linear PCM data) and encoded at a fixed bit rate from Table 1-2 or a variable bit rate.

Table 1-2. Bit Rates

MPEG-1 Audio Layer-3 [kbps]	MPEG-2 Audio Layer-3 LSF [kbps]
32	8
40	16
48	24
56	32
64	40
80	48
96	56
112	64
128	80
160	96
192	112
224	128
256	144
320	160

(2) Decoder

The decoder reads input data, performs decoding, and outputs 16-bit linear PCM data.

μSAP77016-B07 decodes and outputs a maximum of two channels.

Finally, it performs CRC processing as error compensation.

(3) DAC

The DAC converts 16-bit linear PCM data to an analog signal.

The DAC must operate at the sampling frequency attached to the encoded data.

If the sampling frequency of the encoded data differs from the DAC, separate frequency conversion software (such as a rate converter) is needed.

1.3 Product Overview

1.3.1 Features

(1) Employs ISO/IEC standardized MP3 decoder algorithm
(2) Bit rate can be fixed bit rate of Table 1-2 or variable bit rate
(3) Input data is 16-bit linear PCM data sampled at frequencies shown in Table 1-1 and encoded at a fixed bit rate from Table 1-2 or a variable bit rate
(4) Output data is 16-bit linear PCM data of the same frequency as the input data sampling frequency
(5) 1152 samples/frame/channel (MPEG1), 576 samples/frame/channel (MPEG2), decoding
(6) Provides CRC as error compensation
(7) Does not handle free format
(8) Supports 2-channel decoding

1.3.2 Functions

(1) Decompression processing
Decompression processing converts compressed data to one frame of 16-bit linear PCM data.

(2) Error compensation
CRC is performed as error compensation processing.

1.3.3 Operating environment

(1) Operable DSPs:
μPD77110, 77113A, 77114, 77115

(2) Required memory size:
μSAP77016-B07 requires memory sizes shown in the following table.

Table 1-3. Required Memory Sizes

Memory	Type		Size [Kwords]
Instruction memory	–		6.4
X memory	RAM	Scratch area	2.3
		Static area	2.2
	ROM		1.7
Y memory	RAM	Scratch area	0.6
		Static area	3.7
	ROM		5.4

Caution One word of instruction memory is 32 bits.
One word of X memory or Y memory is 16 bits.

A scratch area is a memory area that can be discarded when μSAP77016-B07 is not operating. The user can use scratch areas when μSAP77016-B07 is not operating.
However, caution is required when using these areas. Since μSAP77016-B07 will use these areas again when it operates, if a user has set information in scratch areas, the set information cannot be guaranteed.
Refer to **Table 1-4** if using scratch areas.

Table 1-4. Scratch Areas

Memory	Public Symbol Name	Size [words]
X memory	lib_Scratch_x	2304
Y memory	lib_Scratch_y	576

Scratch areas are reserved by the user and set using the mp3_InitDec function. Specification of align/at is not necessary when declaring lib_Scratch_x or lib_Scratch_y.
A static area is a memory area that cannot be discarded even when μSAP77016-B07 is not operating. A user cannot use a static area.
Besides these, input data buffers and output buffers are needed.
In the sample program described later, 1442 words are used as an input data buffer and 2304 words as two output buffers, for a total of three buffers.
Although it is possible to increase the size of the input data buffer, do not decrease it or it may affect operations.
The output buffer size cannot be changed. However, the number of buffers can be changed.

(3) Required D/A specs

D/A 2 ch, 16-bit resolution, sampling frequency shown in Table 1-1

(4) Software tools (Windows™ version)

Table 1-5. Software Tools

Relevant DSP	Software Tools
μPD77016 Family	DSP tools WB77016 (Workbench) HSM77016 (High-speed simulator)

1.3.4 Performance

Table 1-6 shows the MIPS values that are necessary in order to execute decompression processing for one frame in real time.

Table 1-6. MIPS Needed in One Frame Decompression Processing

Conditions		DSP: μPD77110 (Operating frequency 75 MHz, 75 MIPS)		
Decompression	Sampling frequency	32 kHz [MIPS]	44.1 kHz [MIPS]	48 kHz [MIPS]
	Logical[Note 1] maximum MIPS value	Approx. 26	Approx. 35	Approx. 37
	Measured[Note 2] maximum MIPS value	22.6	31.1	33.9
	Measured[Note 2] average MIPS value	19.2	26.4	28.7
CRC	Logical[Note 1] value	0.4	0.5	0.5

Notes 1. Logical means that the value is calculated by taking the maximum number of cycles for the number of loops, number of repeats, and algorithm processing route in a program.

2. Measured means that the value was measured by actually executing μSAP77016-B07 on a real machine using a bit rate of 320 kbps and stereo (2-channel) encoded data.

1.3.5 Directory configuration

The directory configuration of μSAP77016-B07 is shown below.

```
┬ library ┬ m3dram.lib : Library for μPD77110, 77115
│         └ m3drom.lib : Library for μPD77113A, 77114
│
└ smp     ── mp3dec ┬  m3d_dec.h     : Library header file (For decoder)
                    ├  m3d_err.h     : Header file for error information
                    ├  sample.prj    : Project file
                    ├  sample.h      : Header file for sample program
                    ├  sample.asm    : Sample source file
                    ├  sam_data.asm  : Data area file
                    ├  sam_int.asm   : Source file for interrupt servicing
                    ├  sample.tmg    : Timing file
                    ├  serclk.tmg    : Timing file for clocks input serially
                    └  serout16.tmg  : Timing file for saving serial output in file
```

A summary of each directory is shown below.

(1) library

This directory contains library files.

(2) smp --- mp3dec

This directory contains sample program source files and header files.

It also provides timing files described later.

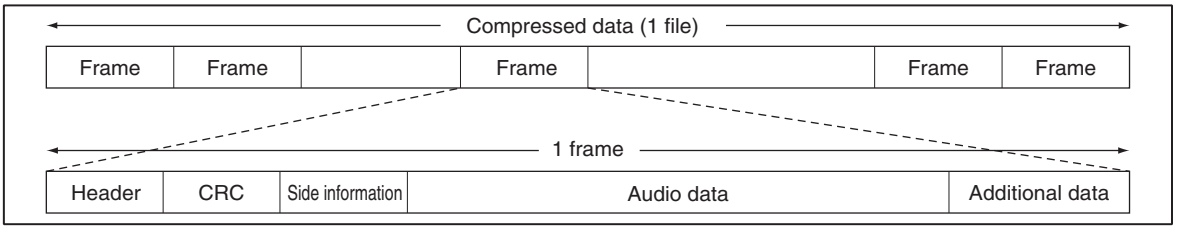
1.4 Compressed Data Format

For details of the compressed data format, refer to standards (**ISO/IEC 11172-3, JIS X 4323**).

μSAP77016-B07 specifications conform to standards. However, μSAP77016-B07 is not in compliance with free format on a bitrate_index of '0000'.

Figure 1-2 shows the format structure of compressed data.

Figure 1-2. Compressed Data Format



1.4.1 Header

Headers contain information such as the sampling frequency, bit rate, and mode for synchronizing.

Table 1-7. Details of Header Section

Information	Number of Bits Used	Value	
Frame sync word	12	'1111 1111 1111': Fixed value	
ID	1	'1': MPEG-1 '0': MPEG-2	
Layer	2	'11': Layer1/'10': Layer2/'01': Layer3	
Protection bit	1	'0': Has CRC	'1': No CRC
Bit rate	4	Refer to **Table 1-8 Relationship of Bit Values to Bit Rates.**	
Sampling frequency	2	Refer to **Table 1-9 Relationship of Bit Values to Sampling Frequencies.**	
Padding bit	1	'1': 1 byte is added to frame[Note 1] '0': 1 byte not added to frame	
Private bit	1	Not used	
Mode	2	'00': stereo '10': dual_channel	'01': joint_stereo '11': single_channel
Mode extension	2	'00': is_off, ms_off '10': is_off, ms_on	'01': is_on, ms_off '11': is_on, ms_on[Note 2]
Copyright	1	'0': no copyright	'1': copyright protected
Original/copy distinction	1	'0': copy	'1': original
Emphasis	2	'00': no emphasis '10': reserved	'01': 50/15 μs '11': CCITT J.17

Notes 1. For a sampling frequency of 44.1 kHz, 1 byte is added to adjust frame ends.

2. "is" designates intensity stereo and "ms" MS stereo.

Table 1-8. Relationship of Bit Values to Bit Rates

Value	MPEG-1 Audio Layer-3 [kbps]	MPEG-2 Audio Layer-3 LSF [kbps]
'0000'	Free format[Note]	
'0001'	32	8
'0010'	40	16
'0011'	48	24
'0100'	56	32
'0101'	64	40
'0110'	80	48
'0111'	96	56
'1000'	112	64
'1001'	128	80
'1010'	160	96
'1011'	192	112
'1100'	224	128
'1101'	256	144
'1110'	320	160
'1111'	Setting prohibited	

Note Not supported by μSAP77016-B07

Table 1-9. Relationship of Bit Values to Sampling Frequencies

Value	MPEG-1 Audio Layer-3 [Hz]	MPEG-2 Audio Layer-3 LSF [Hz]
'00'	44100	22050
'01'	48000	24000
'10'	32000	16000
'11'	Setting prohibited	

1.4.2 CRC

There are two bytes of information that follow the header only when the header protection bit indicates that CRC is in effect. If it is not in effect, the two bytes of information do not exist.

1.4.3 Side information

Side information includes information such as the starting position of audio data in a frame and the decoding method as information needed in decoding audio data.

The size of side information is shown in Table 1-10.

Table 1-10. Size of Side Information

	MPEG-1 Audio Layer-3 [byte]	MPEG-2 Audio Layer-3 LSF [byte]
Monaural	17	9
Stereo	32	17

1.4.4 Audio data

This is data related to an audio sample. The starting position of the audio data is set in the side information. Audio data can start in the same frame as the side information that shows the position of the audio data or it can start in a preceding frame.

Moreover, the starting position of audio data is not limited to one frame before. It also can start two frames before.

Refer to **ISO/IEC 11172-3** for details about audio data.

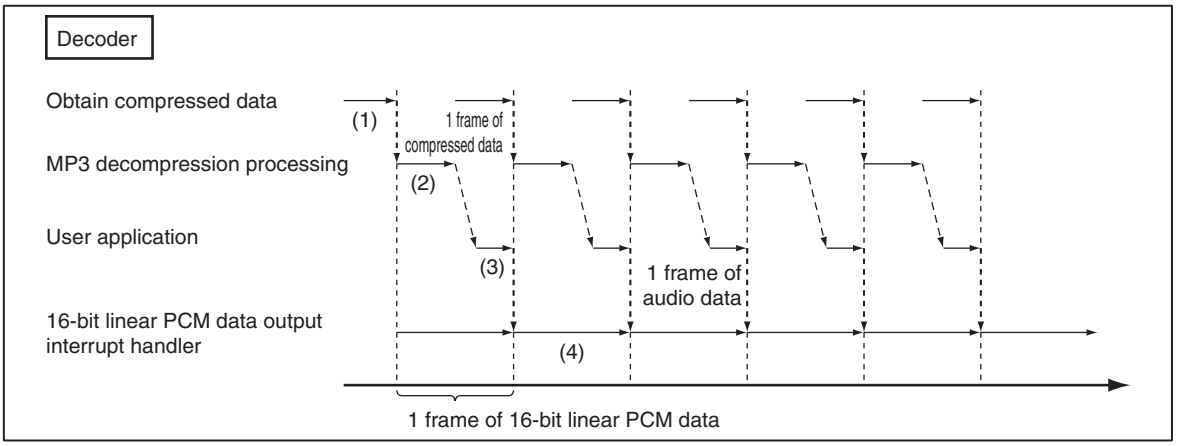
1.4.5 Additional data

This is a segment in which data that the user can define is loaded. This data sometimes does not exist in a frame.

μSAP77016-B07 does not perform any processing on this data. It does not even read it.

1.5 Timing Diagram

Figure 1-3. Decoder Timing Diagram



(1) Read 1 frame of compressed data and pass it to decompression processing.

(2) Convert 1 frame of compressed data to 1 frame of decompressed data.

(3) Buffer decompressed data. Besides this, perform application processing such as rate conversion.

(4) Perform D/A conversion of 1 frame of 16-bit linear PCM data.

CHAPTER 2 LIBRARY SPECIFICATIONS

2.1 Library Overview

μSAP77016-B07 provides the following five functions.

Table 2-1. List of Library Functions

Function Name	Function
mp3_InitDec	Initialize decompression processing
mp3_Dec	Decompression processing
mp3_GetVersion	Obtain version information
mp3_GetStatus	Obtain status information
mp3_GetErrorStatus	Obtain error information

2.2 Function Specifications

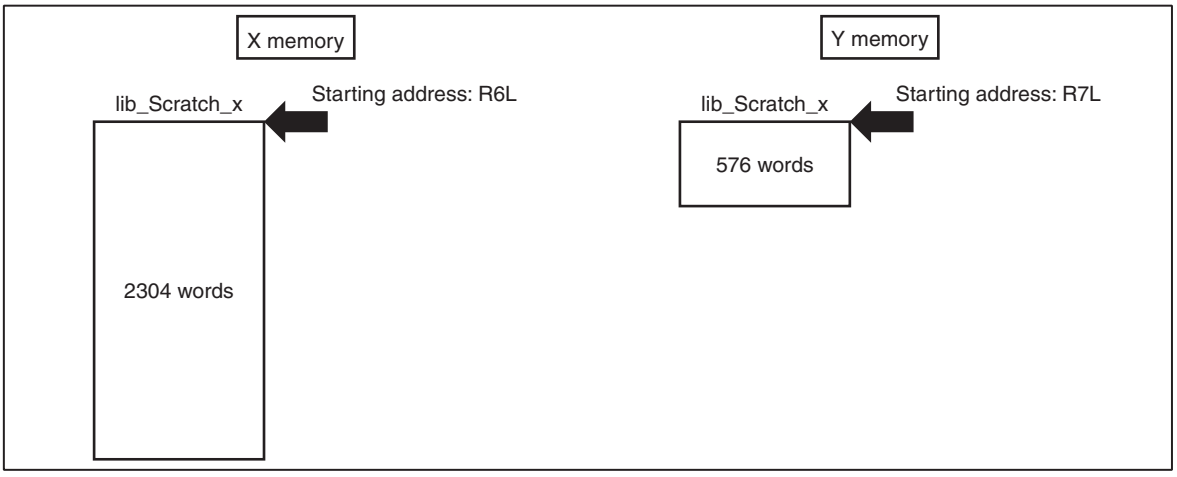
Specifications when calling each library function are shown below.

2.2.1 mp3_InitDec function

[Classification]	MP3 decoder initialization processing	
[Function name]	mp3_InitDec	
[Summary of function]	Initializes RAM areas and sets parameters used by μSAP77016-B07	
[Format]	call mp3_InitDec	
[Arguments]	R0	Length of input MP3 data (bytes)
	R1 to R5	Reserved
	R6L	Starting address of lib_Scratch_x
	R7L	Starting address of lib_Scratch_y
[Return value]	None	
[Function]	Set the sub-band filter, IMDCT, reverse quantization, stereo, CRC, and main processing parameters and initialize RAM areas used by the MP3 decoder.	
[Registers used]	R0, R6, R7, DP0, DP4	
[Hardware resourcement]		
	Maximum stack level	2
	Maximum loop stack level	0
	Maximum number of repeats	576
	Maximum number of cycles	4636

Caution The mp3_InitDec function initializes only RAM areas that the MP3 decoder uses. Initialization of user-defined RAM areas (such as I/O buffers) should be performed in a user program. In addition, the user should reserve lib_Scratch_x and lib_Scratrch_y areas in advance.

Figure 2-1. Scratch Area Memory Image



2.2.2 mp3_Dec function

[Classification] MP3 decoder processing

[Function name] mp3_Dec

[Summary of function] Expands specified compressed data to one frame of audio data.

[Format] call mp3_Dec

[Arguments]

R0	0: Decode 1: Skip 1 frame of decoding
R7	Set the read order of identical addresses at the input buffer (user-defined) read pointer (**Figure 2-2**) 0: Use from higher byte of address indicated by DP2 1: Use from lower byte of address indicated by DP2
DP2	Set the input buffer (user-defined) read pointer (**Figure 2-2**)
DP3	Set the input buffer (user-defined) write pointer (**Figure 2-2**)
DP4	Set to output data buffer (user-defined) pointer
DMX	Set to size of input buffer (user-defined) – 1
DMY	Set to size of output buffer (user-defined) – 1

[Return value]

DP2	Returns input buffer (user-defined) read pointer (**Figure 2-2**) Used in next decoding
R7	Returns higher byte if input buffer (user-defined) read pointer (**Figure 2-2**) is 0 and lower byte if it is 1 Used in next decoding

[Function] mp3_Dec takes the address shown in DP2 and R7 as the start of input data and performs decompression on compressed data until just before the address specified in DP3.

Normally, the arguments DP2 and R7 use the return values of DP2 and R7 from the previous decoding.

The sample program in the Appendix has a two-frame output buffer. This is so that while mp3_Dec outputs one frame, another frame of data can be output to the D/A. The compressed data is a stream of bit units.

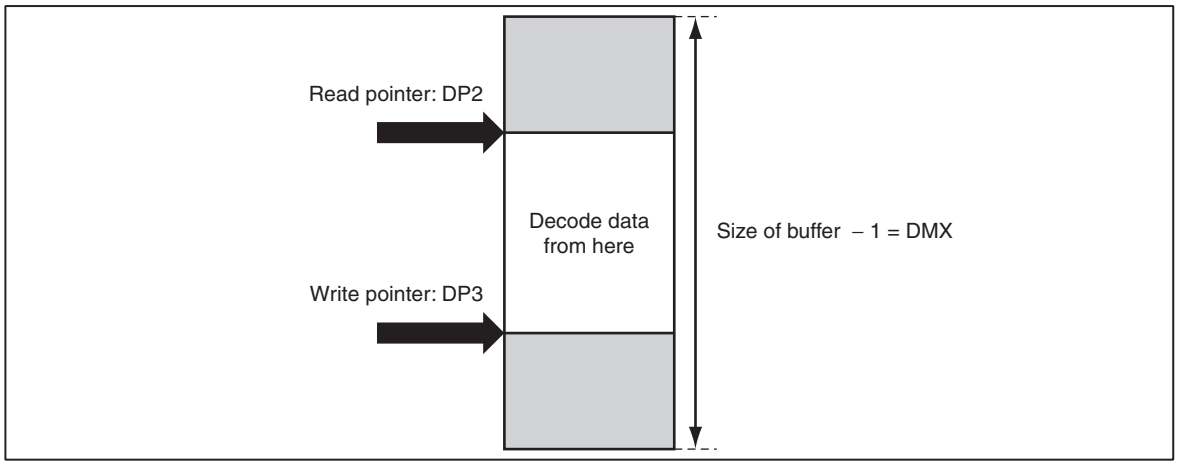
[Registers used] R0, R1, R2, R3, R4, R5, R6, R7
DP0, DP1, DP2, DP3, DP4, DP5, DP6, DP7
DN0, DN1, DN2, DN3, DN4, DN5, DN6, DN7, DMX, DMY

[Hardware resourcement]

Maximum stack level	5
Maximum loop stack level	3
Maximum number of repeats	256
Maximum MIPS value	37 MIPS (If there is no error encoding) 37.5 MIPS (If there is error encoding)

(1) User-defined input buffer

Figure 2-2. User-Defined Input Buffer



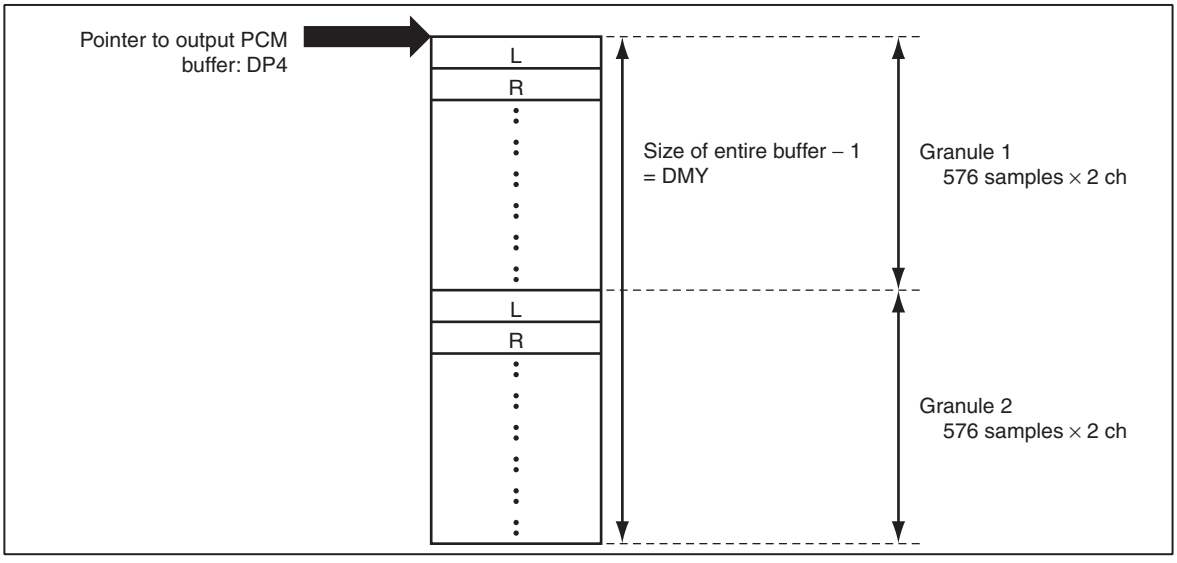
(2) User-defined output buffer

Granule: Internal processing unit of MP3 decoder. Normally, two granules are decoded within one frame. However, one granule is decoded within one frame if LSF.

Sample: Smallest unit of 16-bit linear PCM data output

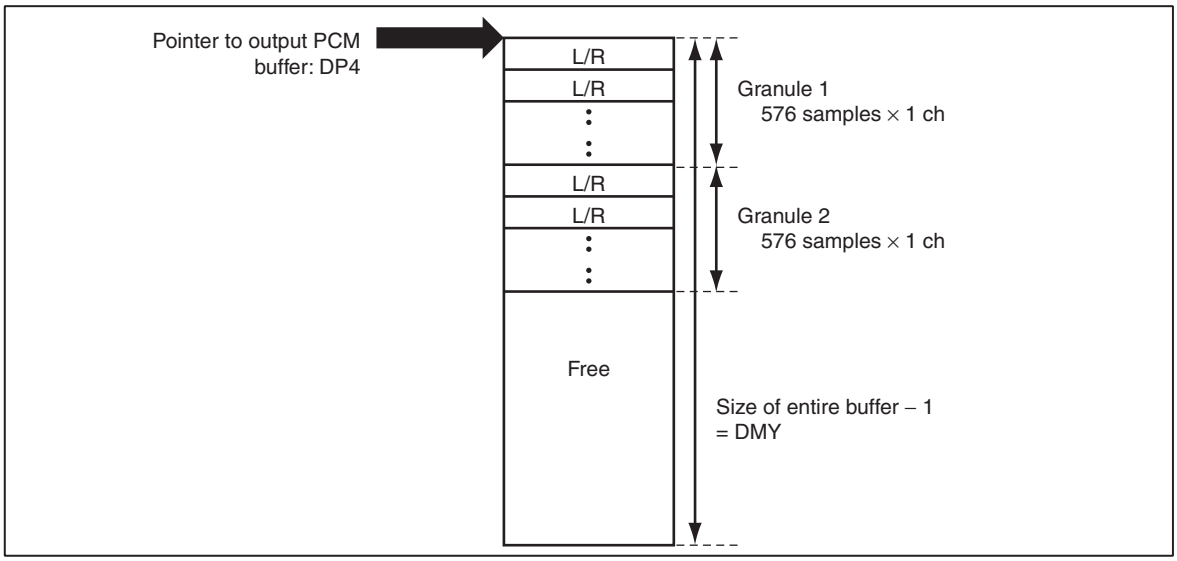
(i) For stereo channel

Figure 2-3. User-Defined Output Buffer for Stereo Channel



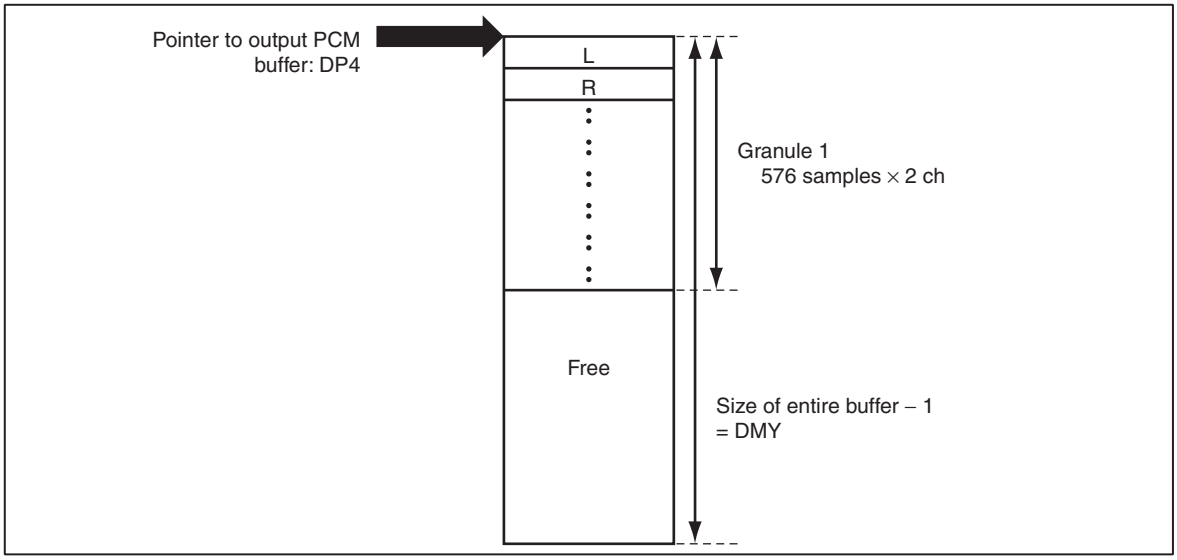
(ii) For single channel

Figure 2-4. User-Defined Output Buffer for Single Channel



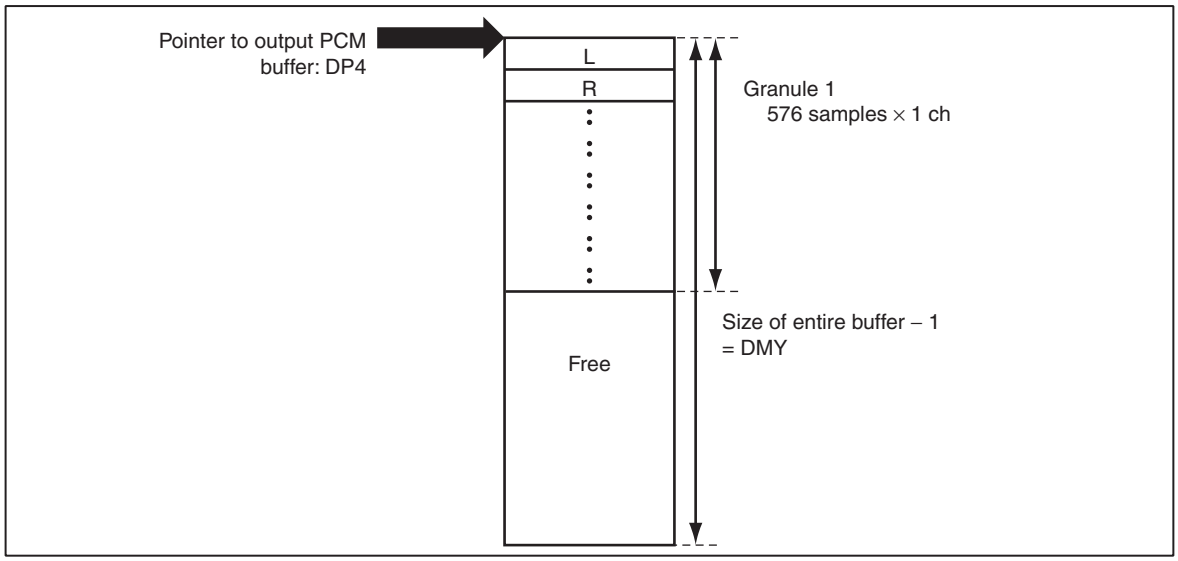
(iii) For LSF and stereo channel

Figure 2-5. User-Defined Output Buffer for LSF and Stereo Channel



(iv) For LSF and single channel

Figure 2-6. User-Defined Output Buffer for LSF and Single Channel



2.2.3 mp3_GetVersion function

[Classification]	Version information acquisition	
[Function name]	mp3_GetVersion	
[Summary of function]	Returns the version of the library.	
[Format]	call mp3_GetVersion	
[Arguments]	None	
[Return value]	R0H	Major version number
	R0L	Minor version number
[Function]	Return the version number of the μSAP77016-B07 library in a 32-bit value.	
	Version when R0 = 0x00'0x0001'0x0100: V1.01	
[Registers used]	R0	
[Hardware resourcement]		
	Maximum stack level	0
	Maximum loop stack level	0
	Maximum number of repeats	0
	Maximum number of cycles	6

2.2.4 mp3_GetStatus function

[Classification] Status information acquisition

[Function name] mp3_GetStatus

[Summary of function] Obtains the status of the MP3 decoder. Information from the decoding performed just before this function was called is returned as the status.

[Format] call mp3_GetStatus

[Arguments] None

[Return value]

R0	0: MPEG-2 Audio Layer-3 LSF Other: MPEG-1 Audio Layer-3
R1	0: Monaural Other: Stereo
R2	Index value of sampling frequency (**Table 2-2**)
R3	0: Normal decoding Other: Skip 1 frame of decoding
R4	Index value of bit rate (**Table 2-3**)

[Function] Obtain status information and set registers.

[Registers used] R0, R1, R2, R3, R4

[Hardware resourcement]

Maximum stack level	0
Maximum loop stack level	0
Maximum number of repeats	0
Maximum number of cycles	15

Table 2-2. Frequencies of mp3_GetStatus() Return Value R2

Value of R2	MPEG-1 Audio Layer-3 [Hz]	MPEG-2 Audio Layer-3 LSF [Hz]
0x00	44100	22050
0x01	48000	24000
0x02	32000	16000

Table 2-3. Bit Rates of mp3_GetStatus() Return Value R4

Value of R4	MPEG-1 Audio Layer-3 [kbps]	MPEG-2 Audio Layer-3 LSF [kbps]
0x00	Free format[Note]	
0x01	32	8
0x02	40	16
0x03	48	24
0x04	56	32
0x05	64	40
0x06	80	48
0x07	96	56
0x08	112	64
0x09	128	80
0x0a	160	96
0x0b	192	112
0x0c	224	128
0x0d	256	144
0x0e	320	160

Note Not supported by μSAP77016-B07

2.2.5 mp3_GetErrorStatus function

[Classification] Error information acquisition

[Function name] mp3_GetErrorStatus

[Summary of function] Obtain MP3 decoder error information.

[Format] call mp3_GetErrorStatus

[Arguments] None

[Return value] R0L Error status

R1L Skip size (number of bytes)

[Function] Obtain the error information of the MP3 decoder using R0L and R1L. Use this function after calling the mp3_Dec function.

- The following values are returned for R0L as error information.

Name	Value	Contents	
MP3_DEC_NO_ERROR	0x0000	Normal	Resumable
MP3_DEC_CRC_ERROR	0x0001	CRC error	Resumable
MP3_DEC_HEADER_FOUND_ERROR	0x0002	Header detection error	Resumable
★ MP3_DEC_BITSTREAM_ERROR	0x0003	Bit stream abnormality • The size of bit stream in the input buffer is zero. • No data found at the point which “main_data_begin” in the bit stream indicates. The bit stream is abnormal.	Resumable

(i) CRC error

This is a CRC error. At this time, DP2 and R7 (read pointer) have been moved to the position next to the end of the frame that is to be decoded.

(ii) Header detection error

Either the file is not an mp3 file or the file may have been corrupted. The user should suspend decoding or continue by filling the input buffer with the bit stream up to where the mp3_Dec function detected a header.

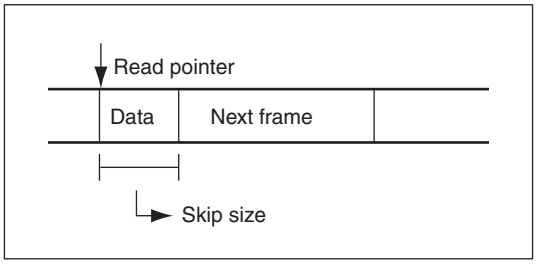
★ (iii) Bit stream abnormality

Either the bit stream needed in decoding is inadequate or the bit stream is abnormal. The user must fill the bit stream. Set the read pointer and write pointer to appropriate values and execute the mp3_Dec function again.

- Skip size of return value

This is the size skipped from the read pointer until the next frame header when the frame header is detected. If the read pointer does not indicate the frame header, the value skipped is returned as the byte size.

Figure 2-7. Skip Size of Read Pointer



[Registers used] R0, R1

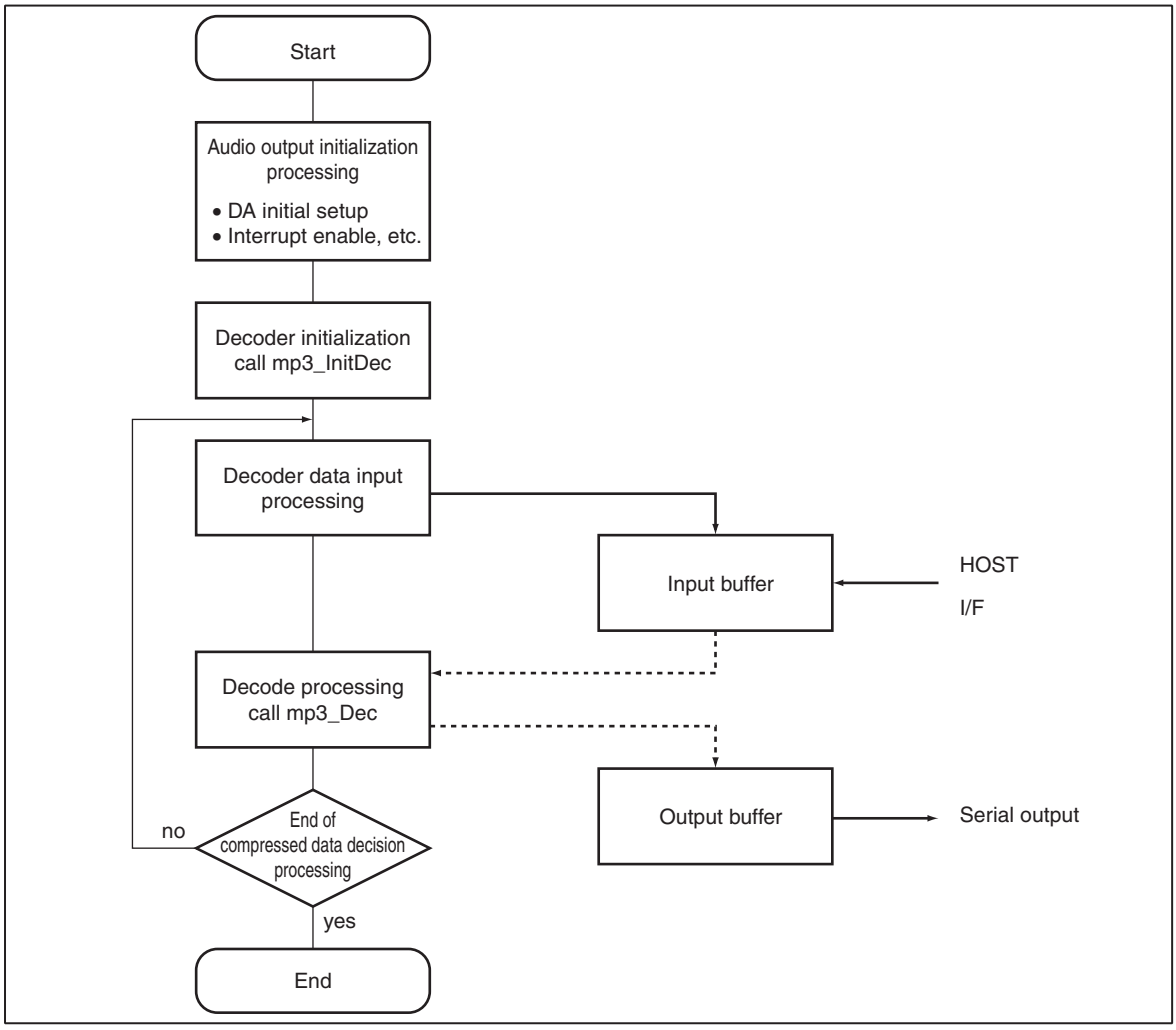
[Hardware resourcement]

Maximum stack level	0
Maximum loop stack level	0
Maximum number of repeats	0
Maximum number of cycles	5

2.3 Application Processing Flow

Figure 2-8 shows an example of the processing of an application that uses the MP3 decoder.

Figure 2-8. Application Processing Flow (Decoder)



The audio data I/O processing section of the interrupt handler is processing that depends on the hardware of the target system. Consequently, the user should design it to suit the target system.

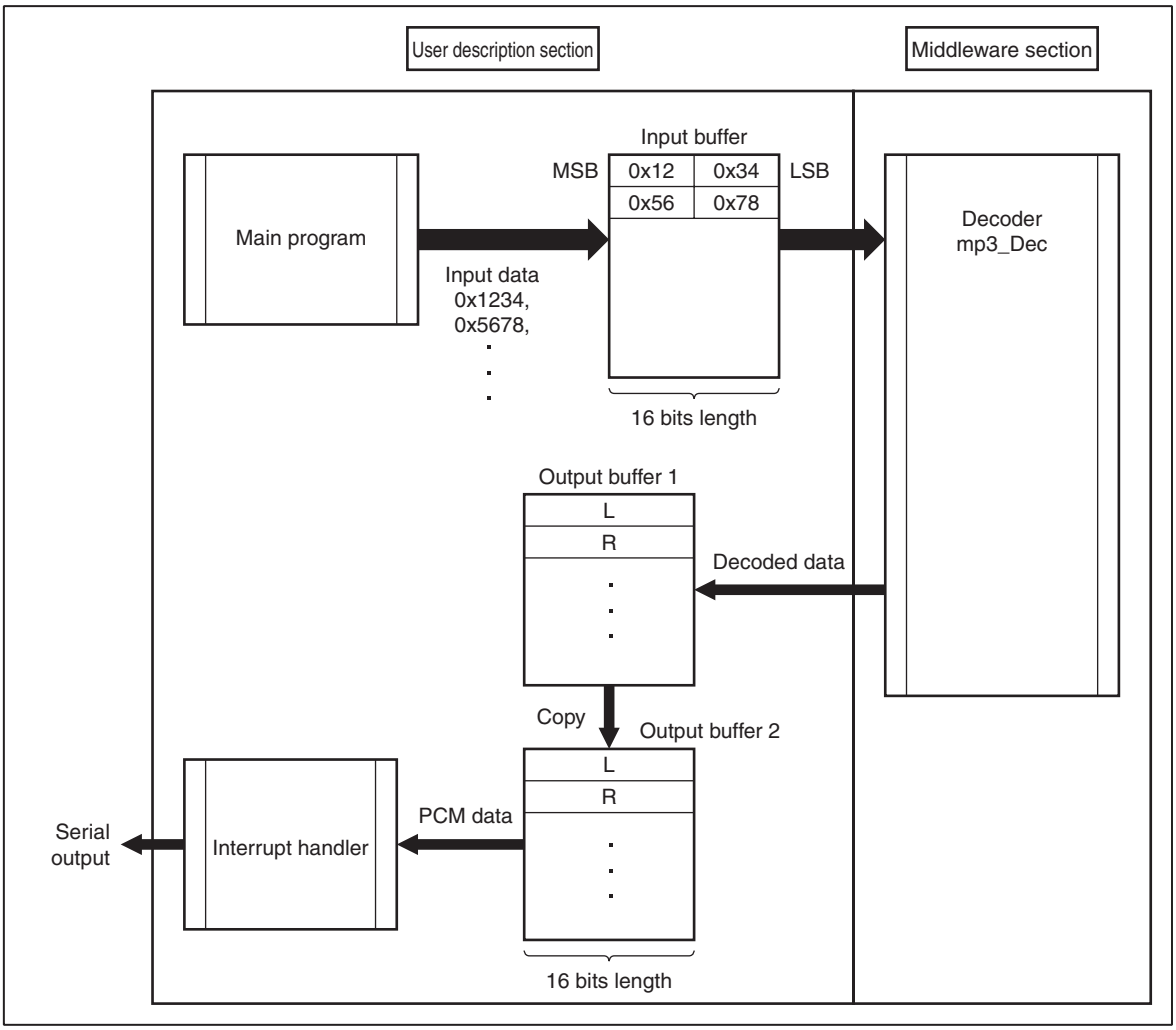
2.3.1 Data flow

Figure 2-9 shows an example of the data flow when decoding.

Data in the input buffer must be set in order from MSB to LSB.

For example, place the data 0x1234, 0x5678, ... in the order 0x12, 0x34, 0x56, 0x78, ...

Figure 2-9. Sample Data Flow



CHAPTER 3 INSTALLATION

3.1 Installation Procedure

The μSAP77016-B07 (MP3 decoder middleware) is supplied on a 3.5-inch floppy disk (1.44 MB). The procedure for installing the μSAP77016-B07 in the host machine is outlined below.

(1) Set the floppy disk in the floppy disk drive and copy the files to the directory where software tools are used (e.g. C:\DSPTools).
The following is an example of when files are copied from the A drive to the C drive.

```
A:\>xcopy /s *.* c:\DSPTools<CR>
```

(2) Confirm that the files have been copied. Refer to **1.3.5 Directory configuration** for details on the directories.

```
A:\>dir c:\DSPTools<CR>
```

3.2 Sample Program Creation Procedure

A sample program is stored in the smp directory.

The sample program operates on HSM77016 (high-speed simulator) Ver. 2.32 or later. Using the timing files described later makes it possible to simulate data I/O. Refer to **CHAPTER 4 SYSTEM EXAMPLE** regarding timing files.

The following is an explanation of how to build the MP3 decoder middleware sample program.

(1) Start up the WB77016 (workbench).

(2) Open the sample.prj project file.

Example Specify sample.prj with the Open Project command on the Project menu.

(3) Execute Build and confirm that sample.lnk has been created.

Example The sample.lnk file can be created by selecting the Build All command from the Make menu.

(4) Start up the HSM77016 (high-speed simulator).

(5) Open the sample.lnk file.

Example Specify sample.lnk with the Open command on the File menu.

(6) Open timing files (sample.tmg, serclk.tmg, serot16.tmg).
serclk.tmg and serot16.tmg are files provided by HSM77016 (high-speed simulator) in Example.

Example Specify sample.tmg with the Open command on the File menu.

3.3 Symbol Naming Regulations

The symbols used in this library are named according to the following regulations.

Classification	Regulation
Function name, variable name	mp3_xxxx
Macro, constant name	mp3_XXXX
Section name	_ _MP3_XXXX (Two leading underscores)

3.4 Sample Program Processing Flow

Figures 3-1 and 3-2 show the processing of a sample program that uses the MP3 decoder.

Figure 3-1. Sample Program Processing Flow (1)

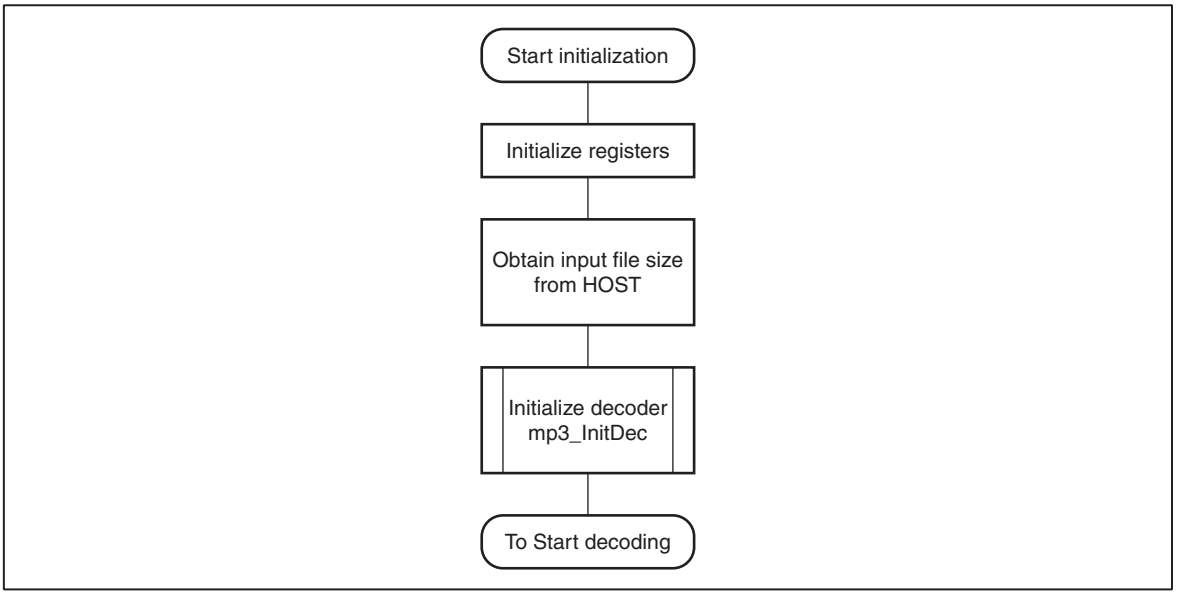
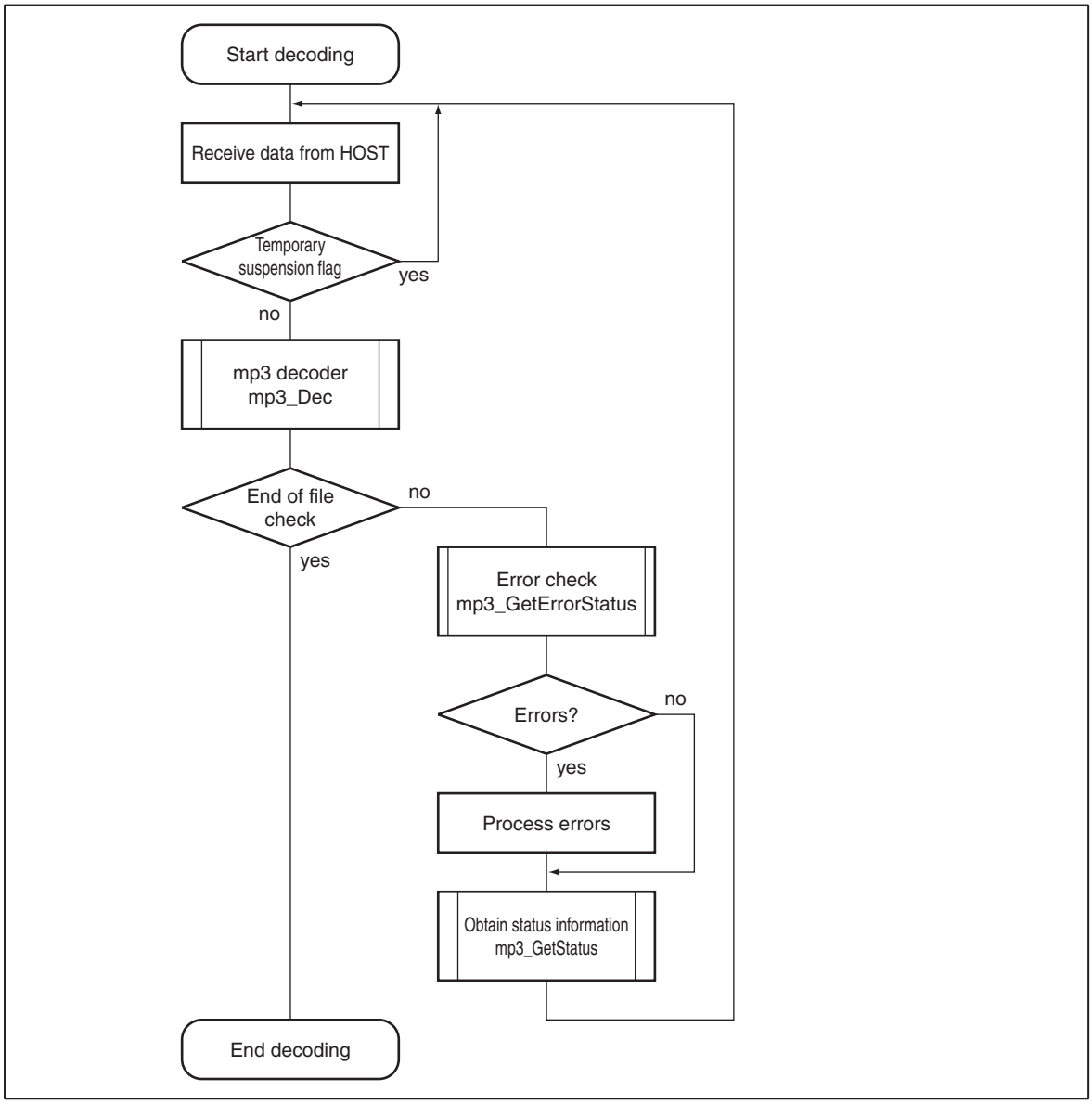


Figure 3-2. Sample Program Processing Flow (2)



CHAPTER 4 SYSTEM EXAMPLE

4.1 Simulation Environment Using Timing Files

An example in which an audio decoding decompression-processing simulator and timing files are used is shown below. Encoded data is input, and then output frame by frame after each frame has undergone compression/decompression processing.

[Software environment]

- High-speed simulator: HSM77016 Ver. 2.32 or later
- Sample program: sample.lnk (created in 3.2 Sample Program Creation Procedure)
- Timing file: sample.tmg

4.2 Operation

<1> Start up the HSM77016 (High-speed simulator)

<2> Open sample.lnk created in 3.2 Sample Program Creation Procedure.

Example Specify sample.lnk with Open command on the File menu.

<3> Open the timing files (sample.tmg, serclk.tmg, serot16.tmg).
serclk.tmg and serot16.tmg are files provided by HSM77016 (high-speed simulator) in Example.

Example Specify sample.tmg with Open command on the File menu.

<4> Make the wait settings.

Example Set waits to the DWTR/IWTR registers in the setting windows opened by selecting Periphery Register on the Window menu.

<5> Execute with Run.

(a) Timing file sample.tmg
HSM77016 (high-speed simulator) provides a function for simulating external I/O using a timing file.

(b) Data file input (16-bit data)
Data is input from a file via the host interface. An example of the description format is shown below.

- **Preparation**

```
local data                                  ; Compressed data storage variable
local size                                  ; MP3 file size (bytes)

set DEBUG_ID = 1                            ; select target
open input "mp3data.dat"                    ; File name of MP3 file converted to text data
set size = 888058                           ; MP3 file size (size of binary file in bytes)
```

- **Input processing (16-bit data)**

(1/2)

```
            set pin hcs = 1                 ; terminate any write access, which might
            set pin hwr = 1                 ; be active
            set pin hrd = 1                 ;
; send MP3 file size
             wait cond pin hwe == 0         ; wait till write is allowed
             wait cond pin hcs == 1         ; and no read is in progress
             set pin hcs = 0                ; perform the access...
             set port ha = 0                ; select higher byte of HDT
             set pin hwr = 0                ; start input
             set port hd = (size>>16)&0xFF  ; input low byte to host port
             wait 100ns                     ; access duration
             set pin hwr = 1                ; end output
             set pin hcs = 1                ; terminate first access...
             wait 5ns                       ; delay
             set port ha = 1                ; select higher byte of HDT
             wait 5ns                       ; delay
             set pin hcs = 0                ; performe second access...
             set pin hwr = 0                ; start output
             set port hd = (size>>24)&0xFF  ; input high byte to host port
             wait 100ns                     ; access duration
             set pin hwr = 1                ; end input
             set pin hcs = 1                ; end access

             wait 1                         ;

             wait cond pin hwe == 0         ; wait till write is allowed
             wait cond pin hcs == 1         ; and no read is in progress
             set pin hcs = 0                ; perform the access...
             set port ha = 0                ; select higher byte of HDT
             set pin hwr = 0                ; start input
             set port hd = (size>>0)&0xFF   ; input low byte to host port
             wait 100ns                     ; access duration
             set pin hcs = 1                ; terminate first access...
             set pin hwr = 1                ; end output
             wait 5ns                       ; delay
             set port ha = 1                ; select higher byte of HDT
             wait 5ns                       ; delay
             set pin hcs = 0                ; performe second access...
             set pin hwr = 0                ; start output
             set port hd = (size>>8)&0xFF   ; input high byte to host port
             wait 100ns                     ; access duration
             set pin hwr = 1                ; end input
             set pin hcs = 1                ; end access

            do
             exit size<=0 ;
             wait cond pin hwe == 0         ; wait till write is allowed
             wait cond pin hcs == 1         ; and no read is in progress
             set pin hcs = 0                ; perform the access...
             set port ha = 0                ; select higher byte of HDT
             set pin hwr = 0                ; start input
             input data                     ; input host data to temp variable
```

(2/2)

```
            set port hd = data&0xFF             ; input low byte to host port
            wait 100ns                          ; access duration
            set pin hcs = 1                     ; terminate first access...
            set pin hwr = 1                     ; end output
            wait 5ns                            ; delay
            set port ha = 1                     ; select higher byte of HDT
            wait 5ns                            ; delay
            set pin hwr = 0                     ; start output
            set pin hcs = 0                     ; performe second access...
            set port hd = (data>>8)&0xFF        ; input high byte to host port
            wait 100ns                          ; access duration
            set pin hwr = 1                     ; end input
            set pin hcs = 1                     ; end access
            set size = size-2                   ;
          enddo

          close input

          wait cond (reg ip & 0xffff)==(mp3_mon_input_bs_finish & 0xffff)
```

mp3_mon_input_bs_finish is a label name that is defined at the beginning of the sample program source.

- **Termination**

```
Break                                           ; Terminate
end
```

APPENDIX SAMPLE PROGRAM SOURCE

- **sample.h**

```
#define MP3_SAMPLE_INPUT_BUFF_SIZE (1440+2)

#define HDT  0x3806
#define HST  0x3807
#define SDT1 0x3800
#define SST1 0x3801
#define SDT2 0x3802
#define SST2 0x3803

;Scratch area
extrn lib_Scratch_x
;XRAM, size= 2304 word
extrn lib_Scratch_y
;XRAM, size= 576 word

extrn mp3_sample_input_read_ptr             ;
extrn mp3_sample_input_read_byte_flag       ;
extrn mp3_sample_input_write_ptr            ;

extrn mp3_sample_dat_length                 ;data length
extrn mp3_sample_remain_length              ;
extrn mp3_sample_decode_length              ;decoded data length
extrn mp3_sample_error_frame_count          ;error counter

extrn mp3_sample_command_stop_flag          ;
extrn mp3_sample_command_skip_flag          ;
extrn mp3_sample_lsf_frame_odd              ;

extrn mp3_sample_input_buff                 ;
extrn mp3_sample_pcm_buff1                  ;
extrn mp3_sample_pcm_buff2                  ;

extrn mp3_sample_int_so1_output_ptr         ;
extrn mp3_sample_int_so1_output_dmx         ;
extrn mp3_sample_int_so1_output_dn          ;

extrn mp3_sample_int_saver0e                ;
extrn mp3_sample_int_saver0h                ;
extrn mp3_sample_int_saver0l                ;

extrn mp3_sample_int_savedp4                ;
extrn mp3_sample_int_savedn4                ;
extrn mp3_sample_int_savedmy                ;

extrn mp3_sample_int1_saver0e               ;
extrn mp3_sample_int1_saver0h               ;
extrn mp3_sample_int1_saver0l               ;
```

- **sample.asm**

(1/5)

```
/*****************************************************/
/*                                                   */
/*       MP3 DECODER MIDDLE WARE                     */
/*                                                   */
/*         SAMPLE.ASM                                */
/*                                                   */
/*                                                   */
/*****************************************************/

#include "m3d_dec.h"
#include "sample.h"
#include "m3d_err.h"

;for evaluation
public mp3_mon_input_bs_finish                        ;debug symbol
public mp3_mon_decode_result                          ;debug symbol

extrn mp3_sample_copy_mono                            ;
extrn mp3_sample_copy_stereo                          ;

/*****************************************************
mp3_Dec define constant
MP3_DEC_OUTPUT_PCM_BUFFER_SIZE

user define constant
MP3_SAMPLE_INPUT_BUFF_SIZE

user define variable
mp3_sample_input_write_ptr user input buffer write pointer
mp3_sample_input_read_ptr user input buffer read pointer   (mp3_Dec use)
mp3_sample_input_read_byte_flag user input buffer read byte flag (mp3_Dec use)

mp3_sample_dat_length  file length
mp3_sample_remain_length remain file length
mp3_sample_decode_length decoded data length

mp3_sample_error_frame_count error counter

user define buffer
mp3_sample_input_buff ds MP3_SAMPLE_INPUT_BUFF_SIZE input buffer
mp3_sample_pcm_buff1 ds MP3_DEC_OUTPUT_PCM_BUFFER_SIZE output buffer page1
mp3_sample_pcm_buff2 ds MP3_DEC_OUTPUT_PCM_BUFFER_SIZE output buffer page2

*****************************************************/
__sample_start_main imseg at 0x200                    ;
        jmp mp3_sample_init_start                     ; jmp sample main module

__sample_main  imseg                                  ;

mp3_sample_init_start:

/******************  initial routine   ***************/
 r0l = 0x0401                                         ;
 *HST:x = r0l                                         ;

/******************  init register    ****************/
;       Init Reg
        clr(r0)                                       ;
        clr(r1)                                       ;
        clr(r2)                                       ;
        clr(r3)                                       ;
        clr(r4)                                       ;
        clr(r5)                                       ;
        clr(r6)                                       ;
        clr(r7)                                       ;
        dn0 = 0x01                                    ;
        dn1 = 0x01                                    ;
        dn2 = 0x01                                    ;
        dn3 = 0x01                                    ;
        dn4 = 0x01                                    ;
        dn5 = 0x01                                    ;
        dn6 = 0x01                                    ;
        dn7 = 0x01                                    ;
        dp0 = 0x0                                     ;
        dp1 = 0x0                                     ;
        dp2 = 0x0                                     ;
        dp3 = 0x0                                     ;
        dp4 = 0x0                                     ;
        dp5 = 0x0                                     ;
        dp6 = 0x0                                     ;
        dp7 = 0x0                                     ;
        dmx = 0x01                                    ;
        dmy = 0x01                                    ;
```

(2/5)

```
/************* INIT interrupt handler variable ************/
        r0l = mp3_sample_pcm_buff2                          ;
        *mp3_sample_int_so1_output_ptr:y = r0l              ;

/***************   INIT output buffer    **************/
        clr(r0)                                             ;
        dp4 = mp3_sample_pcm_buff2                          ;
        rep MP3_DEC_OUTPUT_PCM_BUFFER_SIZE * 4              ;
        *dp4++ = r0l                                        ;
        dp4 = mp3_sample_pcm_buff1                          ;
        rep MP3_DEC_OUTPUT_PCM_BUFFER_SIZE * 4              ;
        *dp4++ = r0l                                        ;

/****************** INIT SST register  ****************/
        R0L = 0x200                                         ;
        *SST1:X = R0L                                       ;
        *SST2:X = R0L                                       ;
        NOP                                                 ;

/****************** init interrupt mask ****************/
        clr(R0)                                             ;
        R0L = SR                                            ;
        r0 = r0 | 0x03ff                                    ;
;       HO/HI/SO1/INT1/INT2 enable
        R0 = R0 & 0x7fdc                                    ;
        SR = R0L                                            ;
        NOP                                                 ;

        *SDT1:X = R0H                                       ;
        *SDT2:X = R0H                                       ;

/************* receive file length from HOST ************/
        r0L= *HST:x                                         ;
        r0 = r0 & 1                                         ;
        if(r0!=0) jmp $-2                                   ;
        clr(R1)                                             ;
        r2=*HDT:x                                           ;

        r0L= *HST:x                                         ;
        r0 = r0 & 1                                         ;
        if(r0!=0) jmp $-2                                   ;
        r2l=*HDT:x                                          ;

/***************   INIT user variable     **************/
        *mp3_sample_dat_length:x = r2h                      ; data length[byte]
        *mp3_sample_dat_length+1:x = r2l                    ;
        *mp3_sample_remain_length:x = r2h                   ;
        *mp3_sample_remain_length+1:x = r2l                 ;

        clr(r0)                                             ;
        *mp3_sample_decode_length:x  = r0h                  ;decoded length[byte]
        *mp3_sample_decode_length+1:x = r0l                 ;

        *mp3_sample_error_frame_count:x = r0l               ; error counter

        clr(r0)                                             ;
        *mp3_sample_lsf_frame_odd:y = r0l                   ;
        r0l = mp3_sample_input_buff                         ;
        *mp3_sample_input_write_ptr:y = r0l                 ; user input buffer write ptr
        *mp3_sample_input_read_ptr:y = r0l                  ; user input buffer read ptr
        *mp3_sample_input_read_byte_flag:y = r0l            ;
        *mp3_sample_command_stop_flag:y = r0l               ;
        *mp3_sample_command_skip_flag:y = r0l               ;

/***************    init input buffer   **************/
        dp0 = mp3_sample_input_buff                         ;
        clr(r0)                                             ;
        rep MP3_SAMPLE_INPUT_BUFF_SIZE                      ;
        *dp0++ = r0l                                        ;

/***************    init output PCM buffer   **************/
        dp4 = mp3_sample_pcm_buff2                          ;
        rep MP3_DEC_OUTPUT_PCM_BUFFER_SIZE * 4              ;
        *dp4++ = r0l                                        ;
         dp4 = mp3_sample_pcm_buff1                         ;
        rep MP3_DEC_OUTPUT_PCM_BUFFER_SIZE * 4              ;
        *dp4++ = r0l                                        ;

/******************  Init Decoder   ****************/
        clr(r0)                                             ;
        r0 = *mp3_sample_dat_length:x                       ;
        r0l = *mp3_sample_dat_length+1:x                    ;

        r6l = lib_Scratch_x                                 ;
        r7l = lib_Scratch_y                                 ;

        call mp3_InitDec                                    ;
```

(3/5)

```
mp3_sample_main:                                        ; sample main module

/***************     DECODE START     *****************/
        clr(r0)                                         ;
        r0l = *mp3_sample_input_write_ptr:y             ; bit_stream write ptr
        dp2 = r0l                                       ;

        dmx = MP3_SAMPLE_INPUT_BUFF_SIZE-1              ;
        dn2 = 1                                         ;

/*************** calc input buffer blank size *****************/
        clr(r1)                                         ;
        r1l = *mp3_sample_input_read_ptr:y              ;
        r0  = r1 - r0                                   ;
        if(r0>0) jmp $ + 2                              ;
        r0  = r0 + MP3_SAMPLE_INPUT_BUFF_SIZE           ; r0[word]
        r0 = r0 - 1                                     ;
        r0  = r0 sll 0x01                               ; r0[byte]

/************** calc receive data size *****************/
        r2  = *mp3_sample_remain_length:x               ;
        r2l = *mp3_sample_remain_length+1:x             ;
        r1 = r0 - r2                                    ;
                                                        ;
        if( r1 > 0 ) r0 = r2                            ;

/*********** if get size == 0 jmp mp3_Dec **********/
        if (r0 <= 0 ) jmp skip_read_bs                  ;

/*************** Get data from HOST ***************/
        r0 = r0 + 1                                     ;
        r0 = r0 sra 1                                   ; r0[byte] -> r0[word]
        clr(r1)                                         ;
        loop r0l {                                      ;
                r1L= *HST:x                             ;
                r1 = r1 & 1                             ;
                if(r1!=0) jmp $-2                       ;
                r1l=*HDT:x                              ;
                *dp2%%=r1l                              ;
                nop                                     ;
                nop                                     ;
                nop                                     ;
        }

/***************** remain_length Update ******************/
        r0 = r0 sll 1                                   ; r0[word] -> r0[byte]
        r2 = r2 - r0                                    ;
        *mp3_sample_remain_length:x = r2h               ;remain file length Update
        *mp3_sample_remain_length+1:x = r2l             ;

/********* *mp3_sample_inpup_write_ptr:y Update *********/
                                                        ;bit_stream write ptr Update
        r0l = dp2                                       ;
        *mp3_sample_input_write_ptr:y=r0l               ;

/****************** check stop mode ******************/
        clr(r0)                                         ;
        r0l = *mp3_sample_command_stop_flag:y           ;
        if(r0 != 0 ) jmp mp3_sample_main                ;

skip_read_bs:

/****************** set register  *******************/
;       dp2 = read_ptr dmx = INPUT_BUFF_SIZE -1
;       dp3 = write_ptr
;       dmx =
;       r7l = byte_flag
;       dp4 = output_pcm
        r0l = dp2                                       ;
        dp3 = r0l                                       ;write_ptr

        r0l = *mp3_sample_input_read_ptr:y              ;
        dp2 = r0l                                       ;
        dmx = MP3_SAMPLE_INPUT_BUFF_SIZE -1             ;

        dp4 = mp3_sample_pcm_buff1                      ;
        dmy = (MP3_DEC_OUTPUT_PCM_BUFFER_SIZE*4) -1     ;
        clr(r7)                                         ;
        r7l = *mp3_sample_input_read_byte_flag:y        ;
        clr(r0)                                         ;
        r0l = *mp3_sample_command_skip_flag:y           ;

/****************** EXEC mp3 DECODE *******************/
        call mp3_Dec                                    ;  call main program
```

(4/5)

```
/**********   mp3_sample_decode_length Update   **********/
        clr(r0)                                             ;
        r0l = *mp3_sample_input_read_ptr:y                  ;
        clr(r1)                                             ;
        r1l = dp2                                           ;
        r0 = r1 - r0                                        ;
        *mp3_sample_input_read_ptr:y = r1l                  ;
        if(r0 > 0) jmp $ + 2                                ;
        r0 = r0 + MP3_SAMPLE_INPUT_BUFF_SIZE                ;
        r0 = r0 sll 0x01                                    ;word->byte
        r3 = *mp3_sample_decode_length:x                    ;
        r3l = *mp3_sample_decode_length+1:x                 ;
        r3 = r3 + r0                                        ;
        *mp3_sample_decode_length:x = r3h                   ;
        *mp3_sample_decode_length+1:x = r3l                 ;
        *mp3_sample_input_read_byte_flag:y = r7l            ;

/************* check remain file size  *************/
        r0 = *mp3_sample_dat_length:x                       ;
        r0l = *mp3_sample_dat_length+1:x                    ;
        r0 = r0 - r3                                        ;

/***************  jump END_OF_FILE check routine   ***************/
        if(r0 <= 0) jmp input_bs_end_chk                    ;

/*****************  error routine  *********************/
        call mp3_GetErrorStatus                             ;
;       r0: 0:ok     other:error
        if (r0 != 0) jmp mon_error                          ;jump error routine

        call mp3_GetStatus                                  ;
; r0  0:lsf other:normal
; r1  mono/stereo 0:mono other:stereo
; r2  sample freq (index)
; r3  skip        0:normal other:skip
; r4  bitrate

;copy decoded PCM data
        dmy = (MP3_DEC_OUTPUT_PCM_BUFFER_SIZE*4) -1 ;
        dn5 = 0x01                                          ;
        dn4 = 0x01                                          ;
        if(r0 != 0 ) jmp _chk_copy_normal                   ;
_chk_copy_lsf:
        clr(r0)                                             ;
        r0l = *mp3_sample_lsf_frame_odd:y                   ;
        dp4 = mp3_sample_pcm_buff1                          ;
        if(r0 != 0 ) jmp _frame_odd                         ;
_frame_even:
        dp5 = mp3_sample_pcm_buff2                          ;
        r0l = 0x01                                          ;
        *mp3_sample_lsf_frame_odd:y = r0l                   ;
_wait_lsf_even:
        clr(r0)                                             ;
        r0l = *mp3_sample_int_so1_output_ptr:y              ;
        r0  = r0 - (mp3_sample_pcm_buff2 + 576*2)           ;
        if(r0 < 0 ) jmp _wait_lsf_even                      ;
        jmp _copy_lsf                                       ;
_frame_odd:
        dp5 = mp3_sample_pcm_buff2 + 576*2                  ;
        r0l = 0x00                                          ;
        *mp3_sample_lsf_frame_odd:y = r0l                   ;
_wait_lsf_odd:
        clr(r0)                                             ;
        r0l = *mp3_sample_int_so1_output_ptr:y              ;
        r0  = r0 - (mp3_sample_pcm_buff2 + 576*2)           ;
        if(r0 >= 0 ) jmp _wait_lsf_odd                      ;

_copy_lsf:
        if(r1 == 0) call mp3_sample_copy_mono               ;
        if(r1 != 0) call mp3_sample_copy_stereo             ;
        jmp mp3_mon_decode_result                           ;

_chk_copy_normal:
        dp4 = mp3_sample_pcm_buff1                          ;
        dp5 = mp3_sample_pcm_buff2                          ;
_wait_gr1:
        clr(r0)                                             ;
        r0l = *mp3_sample_int_so1_output_ptr:y              ;
        r0  = r0 - (mp3_sample_pcm_buff2 + 576*2)           ;
        if(r0 < 0 ) jmp _wait_gr1                           ;
        if(r1 == 0) call mp3_sample_copy_mono               ;
        if(r1 != 0) call mp3_sample_copy_stereo             ;
_wait_gr2:
        clr(r0)                                             ;
        r0l = *mp3_sample_int_so1_output_ptr:y              ;
        r0  = r0 - (mp3_sample_pcm_buff2 + 576*2)           ;
        if(r0 >= 0 ) jmp _wait_gr2                          ;
        if(r1 == 0) call mp3_sample_copy_mono               ;
        if(r1 != 0) call mp3_sample_copy_stereo             ;
```

(5/5)

```
/****************  jump mp3_sample_main  ***************/
mp3_mon_decode_result:
        jmp mp3_sample_main                             ;

/***************** check end of file ******************/
input_bs_end_chk:
        r2 = *mp3_sample_dat_length:x                   ;
        r2l = *mp3_sample_dat_length+1:x                ;
        r1 = r2 - r3                                    ;
        if(r1 > 0) jmp skip_read_bs                     ;


/***************** decode finish  ******************/
mp3_mon_input_bs_finish:

/***************    clear output buffer   **************/
        clr(r0)                                         ;
        dp4 = mp3_sample_pcm_buff1                      ;
        rep MP3_DEC_OUTPUT_PCM_BUFFER_SIZE*4            ;
        *dp4++ = r0l                                    ;
        dp4 = mp3_sample_pcm_buff2                      ;
        rep MP3_DEC_OUTPUT_PCM_BUFFER_SIZE*4            ;
        *dp4++ = r0l                                    ;

input_bs_finish:
        clr(r0)                                         ;
        stk = r0l                                       ;
        r0l = stk                                       ;
        nop                                             ;
;       stop                                            ;
        nop                                             ;
        jmp 0x200                                       ;

/******************  error routine  *********************/

/******** *mp3_sample_error_frame_count:x += 1 *********/
mon_error:
        clr(r1)                                         ;
        r1l = *mp3_sample_error_frame_count:x           ;
        r1 = r1 + 1                                     ;
        *mp3_sample_error_frame_count:x = r1l           ;

; goto error type routine
        r1 = r0 ^ MP3_DEC_CRC_ERROR                     ;
        if(r1 ==0) jmp _error_return                    ;

        r1 = r0 ^ MP3_DEC_HEADDER_FOUND_ERROR           ;
        if(r1 ==0) jmp _header_error                    ;restartable error
; if(r1 ==0) jmp _error_finish                          ;fatal error

        r1 = r0 ^ MP3_DEC_BITSTREAM_ERROR               ;
        if(r1 ==0) jmp _error_return                    ;restartable error

        r1 = r0 ^ MP3_DEC_BIT_RATE_ERROR                ;
        if(r1 ==0) jmp _error_finish                    ;fatal error

        r1 = r0 ^ MP3_DEC_FREE_FORMAT_ERROR             ;
        if(r1 ==0) jmp _error_finish                    ;fatal error

        r1 = r0 ^ MP3_DEC_LAYER_ERROR                   ;
        if(r1 ==0) jmp _error_finish                    ;fatal error

        r1 = r0 ^ MP3_DEC_SAMPLE_RATE_ERROR             ;
        if(r1 ==0) jmp _error_finish                    ;fatal error

mp3_mon_error_finish:
_error_finish:

/****************  fatal error routine  *****************/
;       finish routine
        stop                                            ;
        halt                                            ;

/****************  restartable error ***************/
_header_error:
        r0  = *mp3_sample_remain_length:x               ;
        r0l = *mp3_sample_remain_length+1:x             ;
        r0  = r0 - (17*2)                               ;
        if(r0<=0) jmp input_bs_finish                   ;
_error_return:
;       if( r0 ==0 ) call 1frame_repeat                 ;1frame repeat
;       if( r0 !=0 ) call 1frame_skip                   ;1frame skip
;       call 1frame_mute                                ;
;       call mp3_sample_pcmbuf_wait_with_crc_error      ;

        jmp mp3_sample_main                             ;

END                                                     ;
```

- **sam_data.asm**

```
#include "m3d_dec.h"
#define MP3_SAMPLE_INPUT_BUFF_SIZE ( 1440 + 2 )


public mp3_sample_input_read_ptr                    ;user input buffer read pointer   (mp3_Dec use)
public mp3_sample_input_read_byte_flag              ;user input buffer read byte flag (mp3_Dec use)
public mp3_sample_input_write_ptr                   ;user input buffer write pointer

public mp3_sample_dat_length                        ;file length
public mp3_sample_remain_length                     ;remain data length
public mp3_sample_decode_length                     ;decoded data length
public mp3_sample_error_frame_count                 ;error counter

public mp3_sample_command_stop_flag                 ;user skop mode flag
public mp3_sample_command_skip_flag                 ;user skip mode flag

public mp3_sample_input_buff                        ;user input buffer
public mp3_sample_pcm_buff1                         ;user output buffer page1
public mp3_sample_pcm_buff2                         ;user output buffer page2

public mp3_sample_int_so1_output_ptr                ;user interrupt handler output pointer
public mp3_sample_int_so1_output_dmx                ;user interrupt handler output dmx
public mp3_sample_int_so1_output_dn                 ;user interrupt handler output dn
public mp3_sample_lsf_frame_odd                     ;user lsf mode page flag

public mp3_sample_int_saver0e                       ;
public mp3_sample_int_saver0h                       ;
public mp3_sample_int_saver0l                       ;

public mp3_sample_int_savedp4                       ;
public mp3_sample_int_savedn4                       ;
public mp3_sample_int_savedmy                       ;

public mp3_sample_int1_saver0e                      ;
public mp3_sample_int1_saver0h                      ;
public mp3_sample_int1_saver0l                      ;

__MP3_SAMPLE_INPUT_BUFF xramseg align
mp3_sample_input_buff: ds MP3_SAMPLE_INPUT_BUFF_SIZE ;

__MP3_SAMPLE_OUTPUT_PCM_BUFF1 yramseg align
mp3_sample_pcm_buff1:  ds MP3_DEC_OUTPUT_PCM_BUFFER_SIZE*4
__MP3_SAMPLE_OUTPUT_PCM_BUFF2 yramseg align
mp3_sample_pcm_buff2:  ds MP3_DEC_OUTPUT_PCM_BUFFER_SIZE*4
__MP3_SAMPLE_DATA_Y1 yramseg
mp3_sample_input_read_ptr: ds 1                     ;
mp3_sample_input_read_byte_flag:ds 1                ;
mp3_sample_input_write_ptr: ds 1                    ;


mp3_sample_command_stop_flag: ds 1                  ;
mp3_sample_command_skip_flag: ds 1                  ;

mp3_sample_int_so1_output_ptr: ds 1                 ;
mp3_sample_int_so1_output_dmx: ds 1                 ;
mp3_sample_int_so1_output_dn: ds 1                  ;
mp3_sample_lsf_frame_odd: ds 1                      ;

__MP3_SAMPLE_DATA_X1 xramseg

mp3_sample_dat_length:  ds 2                        ;file length
mp3_sample_remain_length: ds 2                      ;
mp3_sample_decode_length: ds 2                      ;decoded data length
mp3_sample_error_frame_count: ds 1                  ;error counter

mp3_sample_int_saver0e:  ds 1                       ;
mp3_sample_int_saver0h:  ds 1                       ;
mp3_sample_int_saver0l:  ds 1                       ;

mp3_sample_int_savedp4:  ds 1                       ;
mp3_sample_int_savedn4:  ds 1                       ;
mp3_sample_int_savedmy:  ds 1                       ;
mp3_sample_int1_saver0e: ds 1                       ;
mp3_sample_int1_saver0h: ds 1                       ;
mp3_sample_int1_saver0l: ds 1                       ;



end
```

- **sam_int.asm**

(1/3)

```
#include "m3d_dec.h"
#include "sample.h"
#include "m3d_err.h"


public mp3_sample_copy_mono                         ;
public mp3_sample_copy_stereo                       ;


_mp3_sample_intvector imseg at 0x0210
mp3_sample_int_INT1:
                call mp3_sample_command_stop        ;
                reti                                ;
                nop                                 ;
                nop                                 ;
mp3_sample_int_INT2:
                call mp3_sample_command_skip        ;
                reti                                ;
                nop                                 ;
                nop                                 ;
mp3_sample_int_INT3:
                nop                                 ;
                reti                                ;
                nop                                 ;
                nop                                 ;
mp3_sample_int_INT4:
                nop                                 ;
                reti                                ;
                nop                                 ;
                nop                                 ;

mp3_sample_int_si1:
;               call mp3_sample_int_si1_main        ;
                nop                                 ;
                reti                                ;
                nop                                 ;
                nop                                 ;
mp3_sample_int_so1:
                call mp3_sample_int_si1_main        ;
                reti                                ;
                nop                                 ;
                nop                                 ;
mp3_sample_int_si2:
                nop                                 ;
                nop                                 ;
                nop                                 ;
                nop                                 ;
mp3_sample_int_so2:
                nop                                 ;
                nop                                 ;
                nop                                 ;
                nop                                 ;
mp3_sample_int_HI:
                nop                                 ;
                reti                                ;
                nop                                 ;
                nop                                 ;
mp3_sample_int_HO:
                nop                                 ;
                reti                                ;
                nop                                 ;
                nop                                 ;



_mp3_sample_int imseg


mp3_sample_int_si1_main:
;       interrupt for PCM data output
;       save register
        *mp3_sample_int_saver0e:x = r0e             ;
        *mp3_sample_int_saver0h:x = r0h             ;
        *mp3_sample_int_saver0l:x = r0l             ;

        r0l = dp4                                   ;
        *mp3_sample_int_savedp4:x = r0l             ;

        r0l = dn4                                   ;
        *mp3_sample_int_savedn4:x = r0l             ;

        r0l = dmy                                   ;
        *mp3_sample_int_savedmy:x = r0l             ;
```

(2/3)

```
        r0l = *mp3_sample_int_so1_ptr:y             ;
        r0l = *mp3_sample_int_so1_output_ptr:y      ;
        dp4 = r0l                                   ;

;       r0l = *mp3_sample_int_so1_dmy:y             ;
;       dmy = r0l    ;
        dmy = (MP3_DEC_OUTPUT_PCM_BUFFER_SIZE*4) -1 ;
;       r0l = *mp3_sample_int_so1_dn4:y             ;
;       dn4 = r0l                                   ;
        dn4 = 0x01                                  ;

; output PCM DATA to SDT1 SDT2 port
        r0l = *dp4%%                                ;
        *SDT1:y = r0l                               ;

        r0l = dp4                                   ;
;       *mp3_sample_int_so1_ptr:y = r0l             ;
        *mp3_sample_int_so1_output_ptr:y= r0l       ;


;       load register
        r0l = *mp3_sample_int_savedmy:x             ;
        dmy = r0l                                   ;

        r0l = *mp3_sample_int_savedn4:x             ;
        dn4 = r0l                                   ;

        r0l = *mp3_sample_int_savedp4:x             ;
        dp4 = r0l                                   ;

        r0l = *mp3_sample_int_saver0l:x             ;
        r0h = *mp3_sample_int_saver0h:x             ;
        r0e = *mp3_sample_int_saver0e:x             ;

        ret                                         ;

/****************     user command      *****************/

/****************  stop command   *****************/
mp3_sample_command_stop:
        *mp3_sample_int1_saver0e:x = r0e            ;
        *mp3_sample_int1_saver0h:x = r0h            ;
        *mp3_sample_int1_saver0l:x = r0l            ;

        clr(r0)                                     ;
        r0l = *mp3_sample_command_stop_flag:y       ;
        if(r0 == 0) jmp _stop_off                   ;
_stop_on:
        r0l = 0x01                                  ;
        *mp3_sample_command_stop_flag:y = r0l       ;
; stop output interrupt
        r0l = SR                                    ;
        r0 = r0 | 0x00f0                            ; mask si1 so1 si2 so2
        sr = r0l                                    ;
        jmp _end_command                            ;
_stop_off:
        clr(r0)                                     ;
        *mp3_sample_command_stop_flag:y = r0l       ;
; run output interrupt
        r0l = SR                                    ;
        r0 = r0 & 0xff0f                            ; mask canncel si1 so1 si2 so2
        sr = r0l                                    ;
        jmp _end_command                            ;

/*****************  skip command  *****************/
mp3_sample_command_skip:
        *mp3_sample_int1_saver0e:x = r0e            ;
        *mp3_sample_int1_saver0h:x = r0h            ;
        *mp3_sample_int1_saver0l:x = r0l            ;

        clr(r0)                                     ;
        r0l = *mp3_sample_command_skip_flag:y       ;
        if(r0 != 0) jmp _skip_off                   ;
_skip_on:
        r0l = 0x01                                  ;
        *mp3_sample_command_skip_flag:y = r0l       ;
        jmp _end_command                            ;

_skip_off:
        clr(r0)                                     ;
        *mp3_sample_command_skip_flag:y = r0l       ;

_end_command:                                       ;
        r0l = *mp3_sample_int1_saver0l:x            ;
        r0h = *mp3_sample_int1_saver0h:x            ;
        r0e = *mp3_sample_int1_saver0e:x            ;

        ret                                         ;
```

(3/3)

```
/*****************  copy decoded PCM data  *****************/

mp3_sample_copy_mono:
;       copy data 1gra
;       mono to stereo   Monaural, so copy same data twice
;       dp4 = mp3_sample_pcm_buff1                    ;
;       dp5 = mp3_sample_pcm_buff2                    ;
;       dn5 = 0x01                                    ;
;       dmy = 576 * 2                                 ;
;
        loop 576 {                                    ;
                r0l = *dp4++                          ;
        *dp5%% = r0l                                  ;
        *dp5%% = r0l                                  ;
        }                                             ;
        ret                                           ;


mp3_sample_copy_stereo:
;       copy 1 gra pcm data
;
;       dp4 = mp3_sample_pcm_buff1  ;
;       dp5 = mp3_sample_pcm_buff2  ;
;       dn5 = 0x01                                    ;
;       dmy = 576 * 2                                 ;
;
        loop 576*2 {                                  ;
                r0l = *dp4++                          ;
                *dp5%% = r0l                          ;
        }                                             ;
        ret                                           ;

end
```

[MEMO]

NEC

Facsimile Message

From:

Name

Company

Tel. FAX

Address

Although NEC has taken all possible steps to ensure that the documentation supplied to our customers is complete, bug free and up-to-date, we readily accept that errors may occur. Despite all the care and precautions we've taken, you may encounter problems in the documentation. Please complete this form whenever you'd like to report errors or suggest improvements to us.

Thank you for your kind support.

North America
NEC Electronics Inc.
Corporate Communications Dept.
Fax: +1-800-729-9288
+1-408-588-6130

Europe
NEC Electronics (Europe) GmbH
Market Communication Dept.
Fax: +49-211-6503-274

South America
NEC do Brasil S.A.
Fax: +55-11-6462-6829

Hong Kong, Philippines, Oceania
NEC Electronics Hong Kong Ltd.
Fax: +852-2886-9022/9044

Korea
NEC Electronics Hong Kong Ltd.
Seoul Branch
Fax: +82-2-528-4411

P.R. China
NEC Electronics Shanghai, Ltd.
Fax: +86-21-6841-1137

Taiwan
NEC Electronics Taiwan Ltd.
Fax: +886-2-2719-5951

Asian Nations except Philippines
NEC Electronics Singapore Pte. Ltd.
Fax: +65-250-3583

Japan
NEC Semiconductor Technical Hotline
Fax: +81- 44-435-9608

I would like to report the following error/make the following suggestion:

Document title: _____

Document number: _____ Page number: _____

If possible, please fax the referenced page or drawing.

Document Rating	Excellent	Good	Acceptable	Poor
Clarity	❑	❑	❑	❑
Technical Accuracy	❑	❑	❑	❑
Organization	❑	❑	❑	❑