NEC

User's Manual

μSAP77016-B19

MP3 Audio Encoder Middleware

Target Device

μPD77110
μPD77113A
μPD77114
μPD77115
μPD77210
μPD77213

Document No. U16553EJ1V0UM00 (1st edition)
Date Published March 2003 NS CP(N)

Printed in Japan

[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 March, 2003. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC Electronics 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 the prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.
- NEC Electronics 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 Electronics 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 Electronics 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 a customer's equipment shall be done under the full responsibility of the customer. NEC Electronics assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC Electronics endeavors to enhance the quality, reliability and safety of NEC Electronics 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 Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.
- NEC Electronics products are classified into the following three quality grades: "Standard", "Special" and "Specific".

  The "Specific" quality grade applies only to NEC Electronics products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of an NEC Electronics product depend on its quality grade, as indicated below. Customers must check the quality grade of each NEC Electronics 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 Electronics products is "Standard" unless otherwise expressly specified in NEC Electronics data sheets or data books, etc. If customers wish to use NEC Electronics products in applications not intended by NEC Electronics, they must contact an NEC Electronics sales representative in advance to determine NEC Electronics' willingness to support a given application.

(Note)

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

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

M8E 02. 11-1

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC Electronics product in your application, please contact the NEC Electronics 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 America, Inc. (U.S.)
Santa Clara, California
Tel: 408-588-6000
800-366-9782
Fax: 408-588-6130
800-729-9288

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

- **Tyskland Filial**
  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: 6253-8311
Fax: 6250-3583

J02.11

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, 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, 77115, 77210, and 77213 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.

- OVERVIEW
- LIBRARY SPECIFICATIONS
- INSTALLATION
- SYSTEM EXAMPLE
- 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 μPD77111 Family User's Manual Architecture.

To learn about μPD77210 Family hardware functions
→ Refer to μPD77210 Family User's Manual Architecture.

To learn about μPD77016 Family instructions
→ Refer to μPD77016 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

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			
μPD77210		U15203E	U15807E		
μPD77213					

Documents Related to Development Tools

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

Documents Related to Middleware

Document Name	Document No.
μSAP77106-B07 User's Manual (MP3 Decoder)	U15134E
μSAP77106-B19 User's Manual (MP3 Encoder)	This manual

Documents Related to Standard

Document Name
ISO/IEC 11172-3:1993 Information technology - Coding of moving pictures and associated audio for digital storage Media at up to about 1.5 Mbit/s - Part3
JIS X 4323-1995 Information technology of Japan Industrial Standard - Coding of moving pictures and associated audio for digital storage Media at up to about 1.5 Mbit/s - Part3: Audio (MR-4284 1995.10.27)
ISO/IEC 13818-3 Information technology - Generic coding of moving pictures and associated audio information - Part 3: Audio
JIS X 4327 Generic coding of moving pictures and associated audio information - Part 3: Audio

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 Electronics 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-B19 is middleware that provides MP3-technology encoding functions.

1.2 MP3 Audio Encoder

MP3 is a method used for coding or decoding high-quality audio signals for storage media. It includes MPEG-1 Audio Layer-3 and MPEG-2 Audio Layer-3 LSF (Low Sampling Frequency). The μSAP77016-B19 conforms to this coding method.

The compressed data format of MPEG-1 Audio Layer-3 conforms to “ISO/IEC 11172-3”, and that of MPEG-2 Audio Layer-3 conforms to “ISO/IEC 13818-3”. MP3 handles 16-bit linear PCM audio data sampled at a frequency from 16 kHz to 48 kHz (refer to Table 1-1).

Table 1-1. Sampling Frequencies

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

1.2.1 Encoder outline

Figure 1-1 shows an example of the configuration of an encoder using the μSAP77016-B19. Figure 1-2 shows an example of the configuration of a system that includes an encoder using the μSAP77016-B19.

Figure 1-1. Encoder Configuration Example

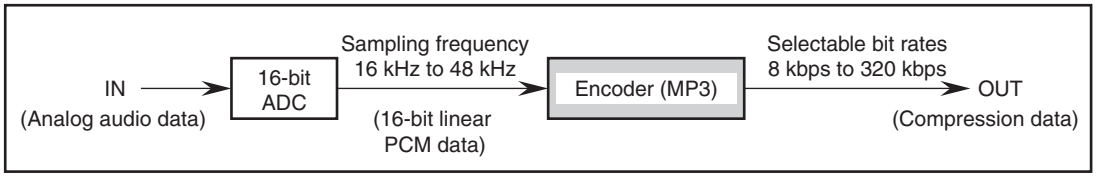
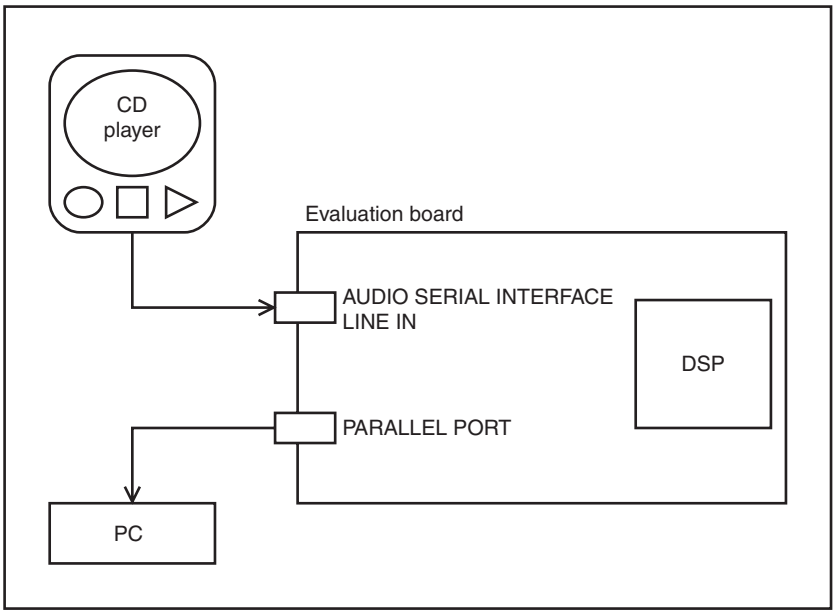


Figure 1-2. System Configuration Example



(1) Input data

The input data is 16-bit linear PCM data sampled at 16 kHz to 48 kHz (refer to Table 1-1).

Table 1-2 shows the number of words of input data per frame.

Table 1-2. Number of Words of Input Data

Compression Ratio	Number of Channels	Number of Words of Input Audio Data
MPEG-1	1	1152
Audio Layer-3	2	2304
MPEG-2	1	576
Audio Layer-3 LSF	2	1152

Remark 1 word = 16 bits

(2) MP3 audio encoder

The MP3 audio encoder reads 16-bit linear PCM data and outputs data while controlling the code quantity at a set bit rate. The bit rate per frame is variable.

Table 1-3 shows the bit rates that can be set.

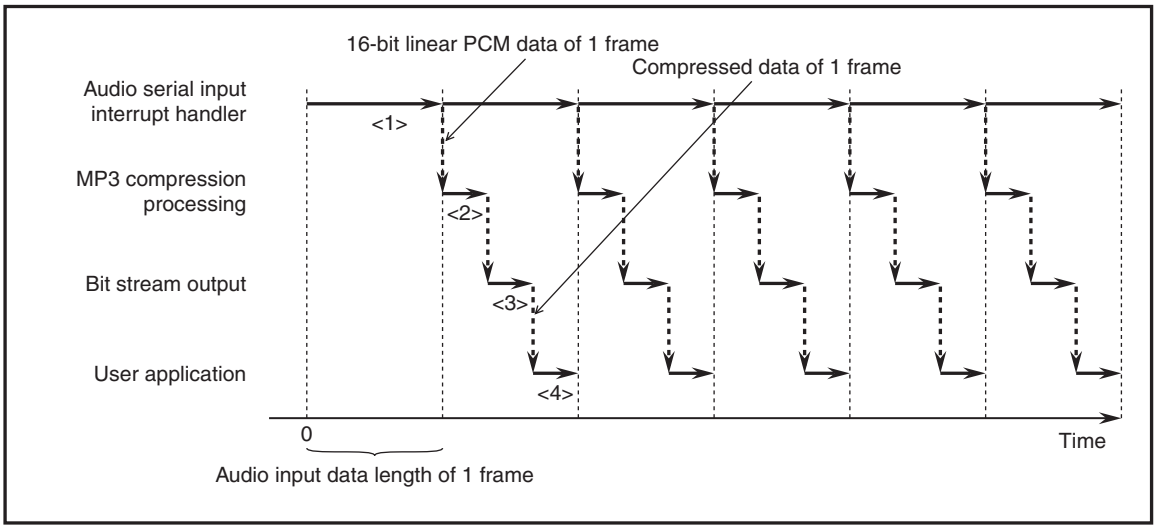
Table 1-3. Selectable Bit Rates

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

(3) Timing diagram

Figure 1-3 shows the timing diagram of the MP3 audio encoder.

Figure 1-3. Timing Diagram



<1> 16-bit linear PCM data of one frame is input.

<2> The 16-bit linear PCM data of one frame is buffered and compressed.

<3> The compressed data is buffered and output.

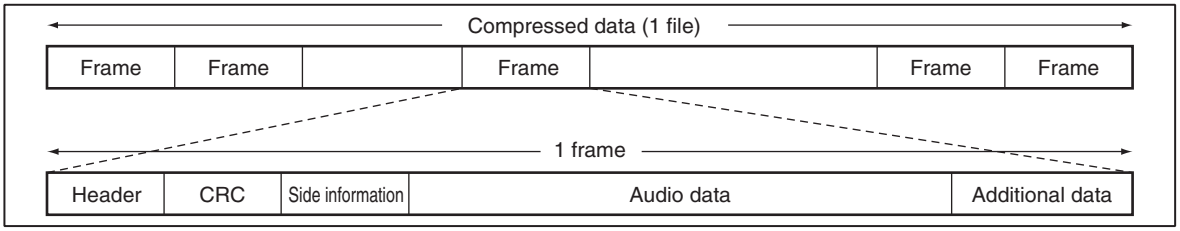
<4> The user performs appropriate processing.

1.3 Compressed Data Format

The compressed data format conforms to ISO/IEC 11172-3 and/or JIS X4323. However, the μSAP77016-B19 does not support the free format when the bitrate_index is 0000. The compressed data consists of a number of bit streams, each of which is called a frame.

Figure 1-4 shows the structure of the compressed data format.

Figure 1-4. Compressed Data Format



1.3.1 Header

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

Table 1-4. 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 (Fixed to 01 in the μSAP77016-B19.)
Protection bit	1	'0': Has CRC '1': No CRC (Fixed to 1 in the μSAP77016-B19.)
Bit rate	4	Refer to **Table 1-5 Relationship of Bit Values to Bit Rates**.
Sampling frequency	2	Refer to **Table 1-6 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 (Fixed to 0 in the μSAP77016-B19.)
Mode	2	'00': stereo '01': joint_stereo '10': dual_channel '11': single_channel (In the μSAP77016-B19, fixed to 01 in the case of stereo, and 11 in the case of mono.)
Mode extension[Note 2]	2	'00': is_off, ms_off '01': is_on, ms_off '10': is_off, ms_on '11': is_on, ms_on (Fixed to 10 in the μSAP77016-B19.)
Copyright	1	'0': no copyright '1': copyright protected (Fixed to 0 in the μSAP77016-B19.)
Original/copy distinction	1	'0': copy '1': original (Fixed to 1 in the μSAP77016-B19.)
Emphasis	2	'00': no emphasis '01': 50/15 μs '10': reserved '11': CCITT J.17 (Fixed to 00 in the μSAP77016-B19.)

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 1-5. 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-B19

Table 1-6. 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	

1.3.2 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.

In the case of the μSAP77016-B19, the data is output without CRC.

1.3.3 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 1-7.

Table 1-7. Size of Side Information

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

1.3.4 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.

1.3.5 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.

This data is not appended in the μSAP77016-B19.

1.4 Product Overview

1.4.1 Features

- Supports mono (single channel) and stereo (joint stereo).
- 16-bit linear PCM data input.
- Code quantity control at set bit rate (refer to **Table 1-5 Relationship of Bit Values to Bit Rates**). The bit rate per frame is variable. A variable bit rate (VBR) is not supported.
- Sampling frequency: 32 kHz/44.1 kHz/48 kHz for MPEG-1 Audio Layer-3; 16 kHz/22.05 kHz/24 kHz for MPEG-2 Audio Layer-3 LSF (refer to **Table 1-6 Relationship of Bit Values to Sampling Frequencies**).
- In mono mode (1 channel), codes 1152 samples/frame for MPEG-1 Audio Layer-3, and 576 samples/frame for MPEG-2 Audio Layer-3 LSF.
- In stereo mode (2 channels), codes 2304 samples/frame for MPEG-1 Audio Layer-3, and 1152 samples/frame for MPEG-2 Audio Layer-3 LSF.
- MS stereo supported; intensity stereo not supported.

1.4.2 Function

The μSAP77016-B19 converts 16-bit linear PCM data of one frame into compressed data.

1.4.3 Operating environment

(1) Operable DSPs:

μPD77110, 77113A, 77114, 77115, 77210, 77213

(2) Required memory size:

The μSAP77016-B19 requires memory sizes shown in the following table.

Table 1-8. Required Memory Sizes

Memory	Type		Size [Kwords]
Instruction memory	–		6.6
X memory	RAM	Scratch area	4.9
		Static area	2.3
	ROM		3.1
Y memory	RAM	Scratch area	2.5
		Static area	3.2
		Library area	0.2
	ROM		0.8

Caution Place the X memory and Y memory areas used for the library in the internal ROM/RAM space.

The μSAP77016-B19 uses a certain area as both a scratch area and bit stream buffer area. The required memory size shown above does not include the input buffer (PCM buffer) for audio data. Refer to 2.5.3 Input buffer.

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

(3) Required A/D converter specifications

2 channels, 16-bit resolution, sampling frequency shown in **Table 1-1 Sampling Frequencies**.

(4) Software tools (Windows® version)

Table 1-9. Software Tools

Relevant DSP	Software Tools
μPD77110 Family	WB77016 (Workbench (Assembler/Linker)) HSM77016 (High-speed simulator) ID77016 (Debugger)
μPD77210 Family	Atair Developer Studio (Workbench (Assembler/Linker)) μPD7721x High-speed simulator μPD7721x Debugger

Remark These DSP software tools are produced by Atair Software GmbH.

1.4.4 Performance

Table 1-10 shows the MIPS values (measured values) necessary for real-time execution of processing of one frame.

- Measurement conditions

  Simulator: HSM77016 (μPD77016 high-speed simulator)

  Evaluation result: The processing speed is measured when a stereo/mono audio file is compressed, and the average value and maximum value are calculated.

  The processing speed of only the MP3ENC_Encode function is included in compression. The processing speed of the other functions and interrupt handlers is not included.

Table 1-10. MIPS Values of 1-Frame Compression Processing (Measured Values)

(a) MPEG-1 Audio Layer-3

Sampling frequency [kHz]		32			44.1			48		
Setting bit rate [kbps]		64	96	128	64	96	128	64	96	128
Average value	Stereo	44.5	45.0	47.5	49.3	50.0	53.8	51.4	51.6	56.0
[MIPS]	Mono	22.7	23.0	23.0	25.1	25.9	25.9	26.2	26.9	27.0
Maximum value	Stereo	68.0	69.2	70.7	77.3	75.5	79.1	76.2	73.7	84.2
[MIPS]	Mono	37.6	38.0	38.3	42.4	42.6	40.9	41.3	42.7	44.0

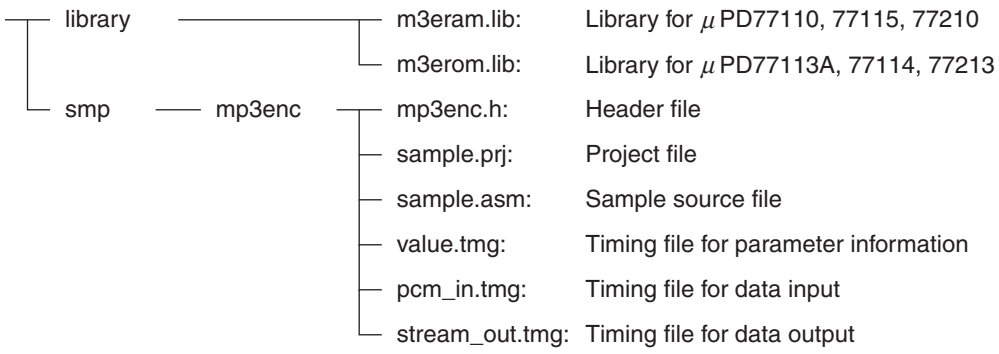
(b) MPEG-2 Audio Layer-3 LSF

Sampling frequency [kHz]		16			22.05			24		
Setting bit rate [kbps]		64	96	128	64	96	128	64	96	128
Average value	Stereo	22.6	23.0	23.2	30.8	31.1	31.5	31.7	34.1	34.8
[MIPS]	Mono	11.2	11.5	11.9	15.3	15.5	15.8	17.1	16.9	17.4
Maximum value	Stereo	36.2	37.4	36.8	51.7	48.8	51.9	54.7	53.4	56.5
[MIPS]	Mono	21.2	20.9	20.7	27.6	28.2	29.0	30.3	33.7	33.6

Remark These MIPS values were measured when evaluation was made by NEC Electronics. The maximum values do not guarantee the worst values.

1.4.5 Directory configuration

The directory configuration of the μPSAP77016-B19 is shown below.



A summary of each directory is shown below.

- library
  This directory contains library files.

- smp/mp3enc
  This directory contains sample program source files, header files and timing files.

CHAPTER 2 LIBRARY SPECIFICATIONS

2.1 Library Overview

μSAP77016-B19 provides the following four functions.

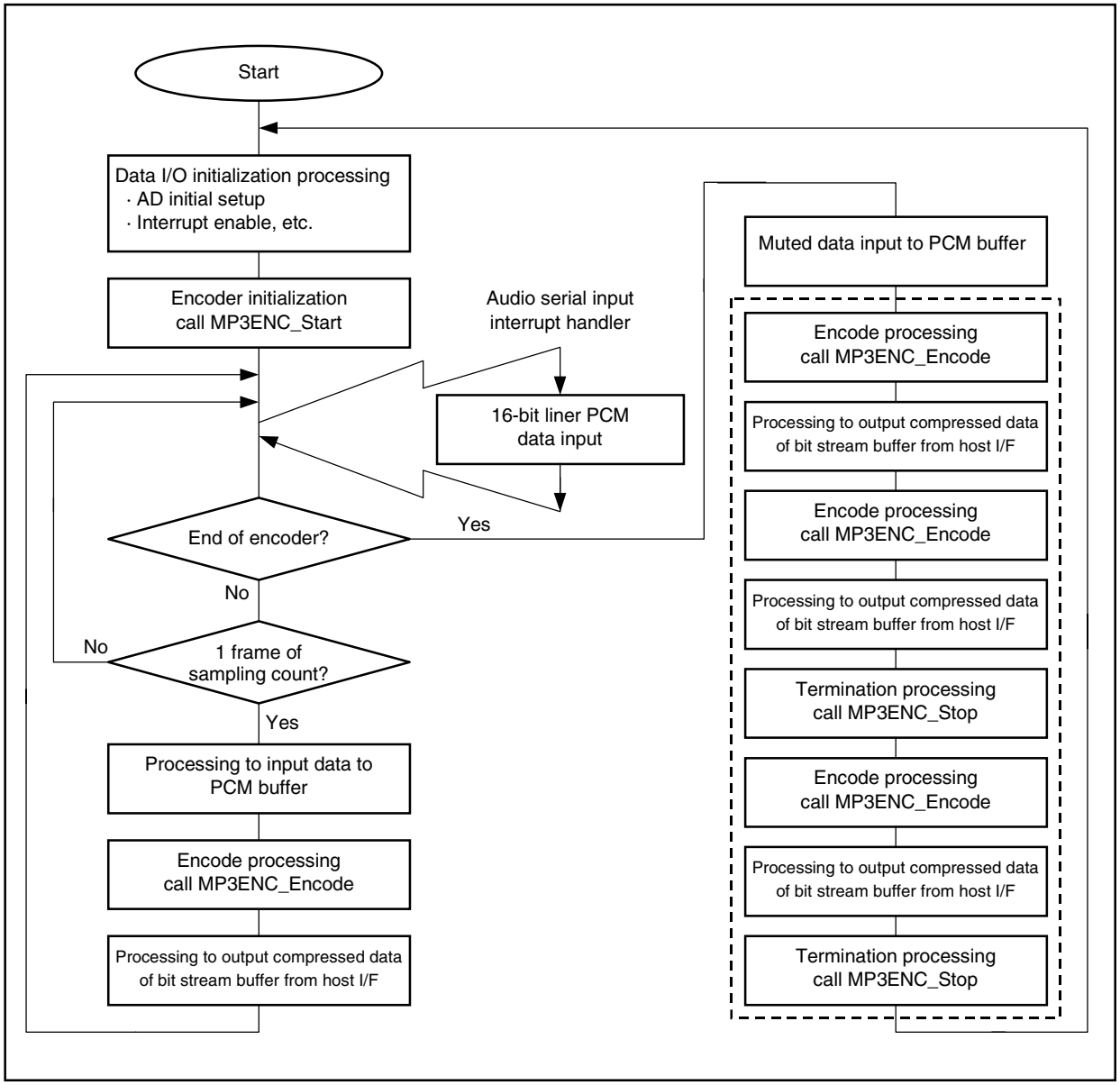
Table 2-1. List of Library Functions

Function Name	Function
MP3ENC_Start	Initialization
MP3ENC_Encode	Compression process
MP3ENC_Stop	Termination process
MP3ENC_GetVersion	Obtain version information

2.2 Application Processing Flow

Figure 2-1 shows an example of the processing of an application that uses μSAP77016-B19.

Figure 2-1. Application Processing Flow



Remark The 16-bit linear PCM data input processing of the interrupt handler is dependent upon the hardware of the target system. Therefore, the user should design the input processing in accordance with their target system.

2.3 Function Specifications

2.3.1 MP3ENC_Start function

The MP3ENC_Start function initializes each parameter used by the encoder. Call this function only once before using the MP3ENC_Encode function.

[Classification]	MP3 encoder initialization processing	
[Function name]	MP3ENC_Start	
[Summary of function]	Initializes the parameters used by the μSAP77016-B19	
[Format]	call MP3ENC_Start	
[Arguments]	R0L	First address of static area in X memory[Note]
	R1L	First address of scratch area in X memory[Note]
	R2L	First address of static area in Y memory[Note]
	R3L	First address of scratch area in Y memory[Note]
[Return value]	None	
[Registers used]	r0, r1, r2, r3, r4, r7,	
	dp0, dp4, dp5	
[Hardware resources]	Maximum stack level	2
	Maximum loop stack level	1
	Maximum number of repeats	3214
	Maximum number of cycles	11642

Note For the memory, refer to **2.5 Memory Structure**.

Caution Secure the scratch memory area and static memory area before calling this function.

2.3.2 MP3ENC_Encode function

The MP3ENC_Encode function compresses audio data equivalent to the number of words of data shown in **Table 1-2 Number of Words of Input Data** at a specified bit rate.

[Classification]	MP3 encode processing	
[Function name]	MP3ENC_Encode	
[Summary of function]	Compresses the 16-bit linear PCM data in the PCM buffer and then outputs the compressed data to the bit stream buffer.	
[Format]	call MP3ENC_Encode	
[Arguments]	R0L	First address of structure consisting of parameters necessary for compressing in X memory[Note 1]
	R1L	First address of input buffer (PCM buffer) for data input for encoder in X memory[Note 2]
	R2L	First address of output buffer (bit stream buffer) for data output from encoder in X memory[Note 2, 3]
	R3L	First address of static area in X memory[Note 2]
	R4L	First address of static area in Y memory[Note 2]
[Return value]	R0L	When 0 or greater: Size of compressed bit stream (number of bytes) When negative: Error
[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 resources]	Maximum stack level	7
	Maximum loop stack level	2
	Maximum number of repeats	31
	Maximum MIPS value	84.2 MIPS (48 kHz sampling, 128 kbps, stereo)

Notes 1. For the parameters necessary for compression, refer to **2.4 Parameters Necessary for Compression**.

2. For the memory area and I/O buffers, refer to **2.5 Memory Structure**.

3. The first address of the output buffer (bit stream buffer) is the same as that of the scratch area in X memory.

Caution On completion of encoding, the user should perform the following procedure. This can prevent the previous audio data from affecting compression of the next audio data.

(1) Input muted data of 1 frame to the PCM buffer frame (refer to **Table 1-2 Number of Words of Input Data**).
(2) Call the MP3ENC_Encode function twice.
(3) Call the MP3ENC_Stop function once.
(4) Call the MP3ENC_Encode function once.
(5) Call the MP3ENC_Stop function once.

2.3.3 MP3ENC_Stop function

The MP3ENC_Stop function terminates encoder processing. For the procedure after terminating encoding, refer to **Figure 2-1 Application Processing Flow**.

[Classification]	MP3 encoder termination processing	
[Function name]	MP3ENC_Stop	
[Function]	Terminates the encoder processing of the μSAP77016-B19.	
[Format]	call MP3ENC_Stop	
[Arguments]	R0L: 1 (during the encode termination processing: first call) 0 (during the encode termination processing: second call) During the encode terminating processing, when calling the MP3ENC_Stop function for the first time, set 1; for the second time, set 0.	
[Return value]	None	
[Registers used]	r0, r7, dp4, dp5	
[Hardware resources]	Maximum stack level	2
	Maximum loop stack level	1
	Maximum number of repeats	0
	Maximum number of cycles	79

2.3.4 MP3ENC_GetVersion function

The MP3ENC_GetVersion function returns the version number of the μSAP77016-B19.

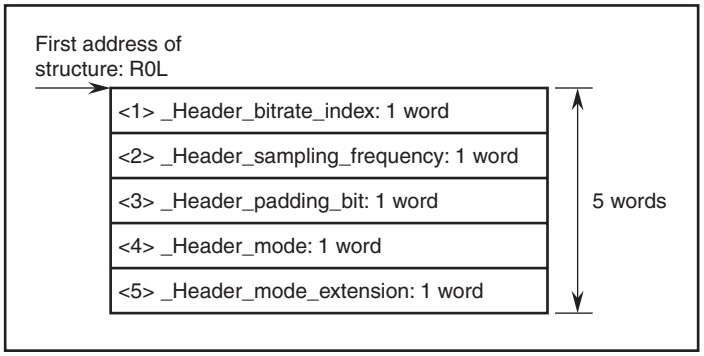
[Classification]	Version information acquisition	
[Function name]	MP3ENC_GetVersion	
[Function]	Returns the version number of the μSAP77016-B19 library in a 32-bit value.	
	Version when R0 = 0x00'0x0001'0x0100: V1.01	
[Format]	call MP3ENC_GetVersion	
[Arguments]	None	
[Return value]	R0H Major version number	
	R0L Minor version number	
[Registers used]	r0	
[Hardware resources]	Maximum stack level	0
	Maximum loop stack level	0
	Maximum number of repeats	0
	Maximum number of cycles	6

2.4 Parameters Necessary for Compression

Secure a structure (refer to Figure 2-2) consisting of the parameters necessary for compression on X memory.

Set the information for each parameter to this structure just once between the execution of the MP3ENC_Start and MP3ENC_Encoder functions.

Figure 2-2. Structure Consisting of Parameters Necessary for Compression



<1> _Header_bitrate_index (1 word): This parameter sets the bit rate. For the setting values, refer to **Table 1-5 Relationship of Bit Values to Bit Rates**.
For example, to set a bit rate of 16 kbps using MPEG-2 Audio Layer-3 LSF, set 0x2.

<2> _Header_sampling_frequency (1 word): This parameter specifies the sampling frequency. The setting values for this word are listed in Table 2-2.
For example, to set a frequency of 16000 Hz using MPEG-2 Audio Layer-3 LSF, set 0x6.

Table 2-2. Values to Specify Sampling Frequency

Compression Method	Value	Frequency [Hz]
MPEG-1 Audio Layer-3	0x0	44100
	0x1	48000
	0x2	32000
	0x3	Setting prohibited
MPEG-2 Audio Layer-3 LSF	0x4	22050
	0x5	24000
	0x6	16000
	0x7	Setting prohibited

<3> _Header_padding_bit (1 word): This parameter does not need to be specified by users. Set any values here.

<4> _Header_mode (1 word): Set 0x3 when the number of channels is one and 0x1 when two.

<5> _Header_mode_extension (1 word): This parameter does not need to be specified by users. Set any values here.

2.5 Memory Structure

With the μSAP77016-B19, the user must define the memory area and input buffer area necessary for processing. The scratch memory area and static memory area must be separately defined. For the size of each memory area, refer to Table 2-3.

Table 2-3. Symbol Name/Memory Sizes

Symbol Name	Size [Words]	X/Y Plane	Description
scratch_x_area	5000	X	Scratch area
scratch_y_area	2533	Y	Scratch area
static_x_area	2304	X	Static area
static_y_area	3214	Y	Static area

Caution Place the X memory and Y memory areas used for the library in the internal ROM/RAM space. The size of the scratch memory area and static memory area shown above does not include the input buffer (PCM buffer). Refer to 2.5.3 Input buffer.

2.5.1 Scratch area

The scratch area is a memory area that can be discarded when it is not used by the μSAP77016-B19.

The user can use the scratch area freely after encoding processing of one frame.

When the μSAP77016-B19 uses this area again, however, the information set by the user to this area is not guaranteed.

Example
```
SCRATCH_X       xramseg
scratch_x_area: ds   5000

SCRATCH_Y       yramseg
scratch_y_area: ds   2533
```

Caution The μSAP77016-B19 also uses the scratch area in X memory as the bit stream buffer. Therefore, copy the encode data (bit stream) that is output to the scratch area (bit stream buffer) in X memory to another area after completing encoding of one frame, and then perform the encoding for the next frame.

2.5.2 Static area

The static area is a memory area that cannot be discarded even when the μSAP77016-B19 does not operate. The user must not use the static area.

If the user manipulates this area after initialization, the correct operation of the μSAP77016-B19 cannot be guaranteed.

Example
```
STATIC_X        xramseg
static_x_area:  ds   2304

STATIC_Y        yramseg
static_y_area:  ds   3214
```

2.5.3 Input buffer

The input buffer is an area to which audio data (16-bit linear PCM data) is input. The user must secure an area for the input buffer in X memory. The required input buffer size is shown in Table 2-4.

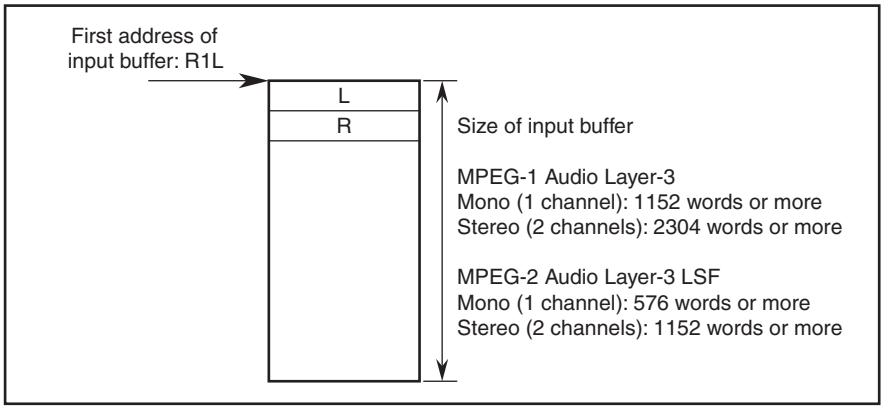
The user can freely use the input buffer area after encoding of one frame. If the input buffer area is manipulated during encoding, correct operation is not guaranteed.

Table 2-4. Input Buffer Size

Compression Ratio	Number of Channels	Number of Words of Input Audio Data	Required Input Buffer Size [word]
MPEG-1	1	1152	1152 or more
Audio Layer-3	2	2304	2304 or more
MPEG-2	1	576	576 or more
Audio Layer-3 LSF	2	1152	1152 or more

Remark 1 word = 16 bits

Figure 2-3. User-Defined Input Buffer (PCM Buffer)



2.5.4 Output buffer

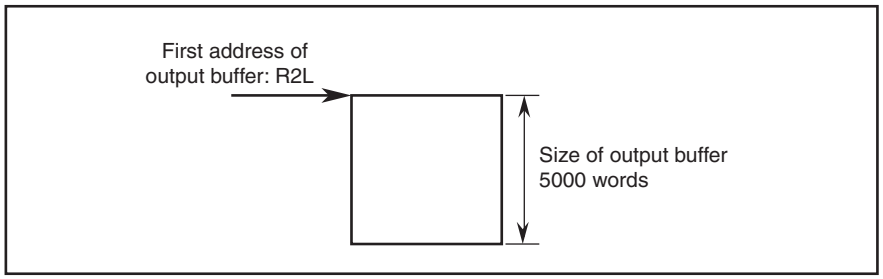
The output buffer (bit stream buffer) is an area from which bit stream data is output. The μSAP77016-B19 specifies a certain area as both a scratch area and bit stream buffer area. Therefore, the user does not need to secure the output buffer in an area other than the scratch area in X memory.

After encoding of one frame, the user can use the output buffer area freely when the output encode data (bit stream)[Note] is copied to another area. If the output buffer area is manipulated during encoding, correct operation is not guaranteed.

Note The maximum size of the bit stream that is output during encoding of one frame is 1872 words.

Caution The μSAP77016-B19 also uses the scratch area in X memory as the bit stream buffer. Therefore, copy the encode data (bit stream) that is output to the scratch area (bit stream buffer) in X memory to another area after completing encoding of one frame, and then perform the encoding for the next frame.

Figure 2-4. Output Buffer (Bit Stream Buffer)



Remark The first address of the output buffer (bit stream buffer) is the same as that of the scratch area in X memory.

CHAPTER 3 INSTALLATION

3.1 Installation Procedure

The μSAP77016-B19 (MP3 audio encoder middleware) is supplied on a CD-ROM. The procedure for installing the μSAP77016-B19 in the host machine is outlined below.

<1> Set the CD-ROM in the CD-ROM drive and copy the files to the directory where WB77016 and HSM77016 (DSP tools) are used (e.g. C:\DSPTools).
The following is an example of when files are copied from the Q drive to the C drive.

```
Q:\>xcopy /s *.* C:\DSPTools<CR>
```

<2> Confirm that the files have been copied. Refer to **1.4.5 Directory configuration** for details on the directories.

```
C:\>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 **APPENDIX SAMPLE PROGRAM SOURCE** regarding timing files.

The following is an explanation of how to build a sample program of the μSAP77016-B19.

<1> Start up the WB77016 (workbench) Ver.2.4 or later.

<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) Ver.2.32 or later.

<5> Open the sample.lnk file.

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

<6> Open timing files (value.tmg, pcm_in.tmg, stream_out.tmg).

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

3.3 Symbol Naming Conventions

Table 3-1 shows the naming conventions of the symbols used for the μSAP77016-B19. Do not use symbol names in duplicate when other applications are used in combination.

Table 3-1. Symbol Naming Conventions

Classification	Convention
Function name, code segment name (IMSEG), constant segment name (ROMSEG/RAMSEG), constant name, variable area name	MP3ENC_XXXX

Remark XXXX is an alphanumeric character string.

CHAPTER 4 SYSTEM EXAMPLE

4.1 Simulation Environment Using Timing File

The simulation environment of the compression processing of the μSAP77016-B19 is shown below. Audio data (16-bit linear PCM data) is input and compressed in 1-frame units, and the compressed data is output.

[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: value.tmg, pcm_in.tmg, stream_out.tmg

4.2 Operation

<1> Start up the high-speed simulator.

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

Example Specify sample.lnk by clicking 'file → open'.

<3> Open the timing files (value.tmg, pcm_in.tmg, stream_out.tmg).

Example Specify value.tmg by clicking 'file → open'.

<4> Execute using Run.

APPENDIX SAMPLE PROGRAM SOURCE

A.1 Header File (mp3enc.h)

```
/*---------------------------------------------------------------------*/
/*      File Information                                               */
/*---------------------------------------------------------------------*/
/*      Name      : mp3enc.h                                           */
/*      Type      : header file                                        */
/*      Version   : 1.00                                               */
/*      Date      : 2002 Nov 26                                        */
/*      CPU       : uPD7701x Family                                    */
/*      Assembler : WB77016 Ver 2.4                                    */
/*      About     : Header                                             */
/*                                                                     */
/*---------------------------------------------------------------------*/
/*      Copyright (C) NEC Electronics Corporation 2002                 */
/*      NEC ELECTRONICS CONFIDENTIAL AND PROPRIETARY                   */
/*      All rights reserved by NEC Electronics Corporation.            */
/*      Use of copyright notice does not evidence publication          */
/*---------------------------------------------------------------------*/

#ifndef   __mp3enc_h
#define   __mp3enc_h

#define   MP3ENC_SCRATCH_AREA_X_SIZE    5000
#define   MP3ENC_SCRATCH_AREA_Y_SIZE    2533
#define   MP3ENC_STATIC_AREA_X_SIZE     2304
#define   MP3ENC_STATIC_AREA_Y_SIZE     3214

#define   HeaderSize    5
;typedef struct {
;         short   bitrate_index
;         short   sampling_frequency
;         short   padding_bit
;         short   mode
;         short   mode_extension
;}

extrn     MP3ENC_Start
extrn     MP3ENC_Encode
extrn     MP3ENC_Stop
extrn     MP3ENC_GetVersion

#endif
```

A.2 Sample Source File (sample.asm)

(1/10)

```
/*---------------------------------------------------------------------*/
/*      File Information                                               */
/*---------------------------------------------------------------------*/
/*      Name      : sample.asm                                         */
/*      Type      : Assembler program module                           */
/*      Version   : 1.00                                               */
/*      Date      : 2002 Nov 26                                        */
/*      CPU       : uPD7701x Family                                    */
/*      Assembler : WB77016 Ver 2.4                                    */
/*      About     : sample                                             */
/*                                                                     */
/*---------------------------------------------------------------------*/
/*      Copyright (C) NEC Electronics Corporation 2002                 */
/*      NEC ELECTRONICS CONFIDENTIAL AND PROPRIETARY                   */
/*      All rights reserved by NEC Electronics Corporation.            */
/*      Use of copyright notice does not evidence publication          */
/*---------------------------------------------------------------------*/

#include "mp3enc.h"

/*---------------------------------------------------------------------*/
#defineBitstreamBufferSize         MP3ENC_SCRATCH_AREA_X_SIZE
#definePcmBufferSize               1152*2

#defineheader_bitrate_index        _Header_bitrate_index:x
#defineheader_sampling_frequency   _Header_sampling_frequency:x
#defineheader_padding_bit          _Header_padding_bit:x
#defineheader_mode                 _Header_mode:x
#defineheader_mode_extension       _Header_mode_extension:x

#defineOutputBufferFlag            _OutputBufferFlag:x
#definevalued                      _valued:x
#defineBitstreamLength_H           _BitstreamLength:x
#defineBitstreamLength_L           _BitstreamLength+1:x
#defineframe                       _frame:x

#defineChannel                     _Channel:x
#definebitrate_H                   _bitrate:x
#definebitrate_L                   _bitrate+1:x
#definestream_length               _stream_length:x
#defineFW_out_flag                 _FW_out_flag:x
#defineFW_run_flag                 _FW_run_flag:x
/*---------------------------------------------------------------------*/

MAIN_CTRL_WORKxramseg at 0x0ff0
       _Channel:            ds     1
       _bitrate:            ds     2
       _OutputBufferFlag:   ds     1
       _valued:             ds     1
```

(2/10)

```
       Header:
       _Header_bitrate_index:              ds     1
       _Header_sampling_frequency:         ds     1
       _Header_padding_bit:                ds     1
       _Header_mode:                       ds     1      ; use in timing file
       _Header_mode_extension:             ds     1

       _BitstreamLength:                   ds     2
       _frame:                             ds     1

       _stream_length:                     ds     1      ; use in timing file
       _FW_out_flag:                       ds     1      ; use in timing file
       _FW_run_flag:                       ds     1      ; use in timing file

MAIN_X_WORK1  xramseg at 0x2000
       PCM:                                ds     PcmBufferSize

MAIN_Y_WORK   yramseg at 0x2000
       OutBitstreamBuffer:                 ds     0x1000

STATIC_X      xramseg
       static_x_area:                      ds     MP3ENC_STATIC_AREA_X_SIZE

SCRATCH_X     xramseg
       __BitstreamBuffer:                  ds     MP3ENC_SCRATCH_AREA_X_SIZE

STATIC_Y      yramseg
       static_y_area:                      ds     MP3ENC_STATIC_AREA_Y_SIZE

SCRATCH_Y     yramseg
       scratch_y_area:                     ds     MP3ENC_SCRATCH_AREA_Y_SIZE


;-------------------------------------------------
VECTOR  imseg at 0x200

%define (JmpVect(addr))
(
       jmp addr;
       nop;
       nop;
       nop;
)

%define (NopVect)
(
       nop;
       reti;
       nop;
       nop;
)
```

(3/10)

```
ivReset:
        %JmpVect(StartUp)           ;
        %NopVect                    ;
        %NopVect                    ;
        %NopVect                    ;
ivINT1:
        %NopVect                    ;
ivINT2:
        %NopVect                    ;
ivINT3:
        %NopVect                    ;
ivINT4:
        %NopVect                    ;
ivINT5:
        %NopVect                    ;
ivINT6:
        %NopVect                    ;
ivINT7:
        %NopVect                    ;
ivINT8:
        %NopVect                    ;
ivINT9:
        %NopVect                    ;
ivINT10:
        %NopVect                    ;
ivINT11:
        %NopVect                    ;
ivINT12:
        %NopVect                    ;


MAIN    imseg at 0x240

%define(Initialize)
(
        clr (r0)                    ;
        r1 = r0                     ;
        r2 = r0                     ;
        r3 = r0                     ;
        r4 = r0                     ;
        r5 = r0                     ;
        r6 = r0                     ;
        r7 = r0                     ;
        dp0 = r0l                   ;
        dp1 = r0l                   ;
        dp2 = r0l                   ;
        dp3 = r0l                   ;
        dp4 = r0l                   ;
        dp5 = r0l                   ;
        dp6 = r0l                   ;
        dp7 = r0l                   ;
        dn0 = r0l                   ;
```

(4/10)

```
       dn1 = r0l                    ;
       dn2 = r0l                    ;
       dn3 = r0l                    ;
       dn4 = r0l                    ;
       dn5 = r0l                    ;
       dn6 = r0l                    ;
       dn7 = r0l                    ;
       dmx = r0l                    ;
       dmy = r0l                    ;
       sp = r0l                     ;
       lsp = r0l                    ;
       r0l = 0xffff                 ;
       sr = r0l                     ;
       eir = r0l                    ;
)

StartUp:
       %Initialize                  ;

       clr (r0)                     ;
       dp0 = __BitstreamBuffer      ;
       rep BitstreamBufferSize      ;
              *dp0++ = r0h          ;
       dp0 = PCM                    ;
       rep PcmBufferSize            ;
              *dp0++ = r0h          ;

       *OutputBufferFlag = r0h      ;
       *valued = r0h                ;
       *stream_length = r0h         ;

       r0l = static_x_area          ;MP3ENC_Start (static_x_area, (scratch_y_area=)__BitstreamBuffer,
       r1l = __BitstreamBuffer      ;       static_y_area, scratch_y_area)
       r2l = static_y_area          ;
       r3l = scratch_y_area         ;
       call MP3ENC_Start            ;

       clr (r0)                     ;run_flag = 0
       *FW_run_flag = r0h           ;
       r0 = *FW_run_flag            ;while (run_flag == 0) {}
       if (r0 == 0) jmp $-1         ;

       ;; header analysis ;; (MSB) ;; 22+2+4+16 = 44byte -> 22word
       dp0 = PCM                    ;
       rep 22/2                     ;fread (buff, 1, 22, fi)
              r0 = *dp0++           ;
              r0 = *dp0++           ;fread (&Channel, 1, 2, fi)
       *Channel = r0h               ;
              r0l = *dp0++          ;fread (&freq, 1, 4, fi)
              r0eh = *dp0++         ;
                                    ;fread (buff, 1, 16, fi)
```

(5/10)

```
       clr (r1)                    ;index = 0
       r2 = r0 - 44100             ;switch (freq) {
       if (r2 == 0) jmp _break     ;case 44100:  index = 0;    break
       r1 = r1 + 1                 ;
       r2 = r0 - 48000             ;case 48000:  index = 1;    break
       if (r2 == 0) jmp _break     ;
       r1 = r1 + 1                 ;
       r2 = r0 - 32000             ;case 32000:  index = 2;    break

       if (r2 == 0) jmp _break     ;
       r1 = r1 + 2                 ;
       r2 = r0 - 22050             ;case 22050:  index = 4;    break
       if (r2 == 0) jmp _break     ;
       r1 = r1 + 1                 ;
       r2 = r0 - 24000             ;case 24000:  index = 5;    break
       if (r2 == 0) jmp _break     ;
       r1 = r1 + 1                 ;
       r2 = r0 - 16000             ;case 16000:  index = 6;    break
       if (r2 == 0) jmp _break     ;
       jmp $                       ;defualt:     exit
_break:                            ;}
       *header_sampling_frequency = r1l;header.sampling_frequency = index

       clr (r0)                    ;
       r1 = r0 + 1                 ;k = 1
       r0l = *Channel              ;if (Channel != 2)
       r2 = r0 - 2                 ;      k += 2
       if (r2 != 0) r1 = r1 + 1    ;
       if (r2 != 0) r1 = r1 + 1    ;
       *header_mode = r1l          ;header.mode = k

       clr (r1)                    ;
       r0 = r1 + 64000             ;64kbps
       nop                         ;
       *bitrate_H = r0h            ;
       *bitrate_L = r0l            ;

       r1l = *header_sampling_frequency;
       r2 = r1 - 4                 ;
       r0 = *bitrate_H             ;
       r0l = *bitrate_L            ;
       clr (r1)                    ;
       if (r2 >= 0) jmp _MPEG2     ;if (header.sampling_frequency < 4) {
_MPEG1:                            ;
       r2 = r0 & 7                 ;      switch (bitrate) {
       if (r2 != 0) jmp _break2    ;
       r0 = r0 sra 3               ;
       r2 = r0 - (32000/8)         ;      case 32000:
       r1 = r1 + 1                 ;             index = 1
       if (r2 == 0) jmp _break2    ;             break
       r2 = r0 - (40000/8)         ;      case 40000:
       r1 = r1 + 1                 ;             index = 2
```

(6/10)

```
        if (r2 == 0) jmp _break2    ;               break
        r2 = r0 - (48000/8)         ;       case 48000:
        r1 = r1 + 1                 ;               index = 3
        if (r2 == 0) jmp _break2    ;               break
        r2 = r0 - (56000/8)         ;       case 56000:
        r1 = r1 + 1                 ;               index = 4
        if (r2 == 0) jmp _break2    ;               break
        r2 = r0 - (64000/8)         ;       case 64000:
        r1 = r1 + 1                 ;               index = 5
        if (r2 == 0) jmp _break2    ;               break
        r2 = r0 - (80000/8)         ;       case 80000:
        r1 = r1 + 1                 ;               index = 6
        if (r2 == 0) jmp _break2    ;               break
        r2 = r0 - (96000/8)         ;       case 96000:
        r1 = r1 + 1                 ;               index = 7
        if (r2 == 0) jmp _break2    ;               break
        r2 = r0 - (112000/8)        ;       case 112000:
        r1 = r1 + 1                 ;               index = 8
        if (r2 == 0) jmp _break2    ;               break
        r2 = r0 - (128000/8)        ;       case 128000:
        r1 = r1 + 1                 ;               index = 9
        if (r2 == 0) jmp _break2    ;               break
        r2 = r0 - (160000/8)        ;       case 160000:
        r1 = r1 + 1                 ;               index = 10
        if (r2 == 0) jmp _break2    ;               break
        r2 = r0 - (192000/8)        ;       case 192000:
        r1 = r1 + 1                 ;               index = 11
        if (r2 == 0) jmp _break2    ;               break
        r2 = r0 - (224000/8)        ;       case 224000:
        r1 = r1 + 1                 ;               index = 12
        if (r2 == 0) jmp _break2    ;               break
        r2 = r0 - (256000/8)        ;       case 256000:
        r1 = r1 + 1                 ;               index = 13
        if (r2 == 0) jmp _break2    ;               break
        r2 = r0 - (320000/8)        ;       case 320000:
        r1 = r1 + 1                 ;               index = 14
        if (r2 == 0) jmp _break2    ;
        clr (r1)                    ;       default:
        jmp _break2                 ;               index = 0
                                    ;       }
_MPEG2:                             ;} else {
        r2 = r0 & 3                 ;       switch (bitrate) {
        if (r2 != 0) jmp _break2    ;
        r0 = r0 sra 2               ;
        r2 = r0 - (8000/4)          ;       case 8000:
        r1 = r1 + 1                 ;               index = 1
        if (r2 == 0) jmp _break2    ;               break
        r2 = r0 - (16000/4)         ;       case 16000:
        r1 = r1 + 1                 ;               index = 2
        if (r2 == 0) jmp _break2    ;               break
        r2 = r0 - (24000/4)         ;       case 24000:
        r1 = r1 + 1                 ;               index = 3
```

(7/10)

```
       if (r2 == 0) jmp _break2     ;              break
       r2 = r0 - (32000/4)          ;       case 32000:
       r1 = r1 + 1                  ;              index = 4
       if (r2 == 0) jmp _break2     ;              break
       r2 = r0 - (40000/4)          ;       case 40000:
       r1 = r1 + 1                  ;              index = 5
       if (r2 == 0) jmp _break2     ;              break
       r2 = r0 - (48000/4)          ;       case 48000:
       r1 = r1 + 1                  ;              index = 6
       if (r2 == 0) jmp _break2     ;              break
       r2 = r0 - (56000/4)          ;       case 56000:
       r1 = r1 + 1                  ;              index = 7
       if (r2 == 0) jmp _break2     ;              break
       r2 = r0 - (64000/4)          ;       case 64000:
       r1 = r1 + 1                  ;              index = 8
       if (r2 == 0) jmp _break2     ;              break
       r2 = r0 - (80000/4)          ;       case 80000:
       r1 = r1 + 1                  ;              index = 9
       if (r2 == 0) jmp _break2     ;              break
       r2 = r0 - (96000/4)          ;       case 96000:
       r1 = r1 + 1                  ;              index = 10
       if (r2 == 0) jmp _break2     ;              break
       r2 = r0 - (112000/4)         ;       case 112000:
       r1 = r1 + 1                  ;              index = 11
       if (r2 == 0) jmp _break2     ;              break
       r2 = r0 - (128000/4)         ;       case 128000:
       r1 = r1 + 1                  ;              index = 12
       if (r2 == 0) jmp _break2     ;              break
       r2 = r0 - (144000/4)         ;       case 144000:
       r1 = r1 + 1                  ;              index = 13
       if (r2 == 0) jmp _break2     ;              break
       r2 = r0 - (160000/4)         ;       case 160000:
       r1 = r1 + 1                  ;              index = 14
       if (r2 == 0) jmp _break2     ;
       clr (r1)                     ;       default:
                                    ;              index = 0
                                    ;       }
_break2:                            ;}
       *header_bitrate_index = r1l  ;

       clr (r0)                     ;
       *frame = r0h                 ;frame = 0
       *BitstreamLength_H = r0h     ;BitstreamLength = 0
       *BitstreamLength_L = r0l     ;

_while1:                            ;while (1) {
       clr (r0)                     ;       run_flag = 0
       *FW_run_flag = r0h           ;
       r0 = *FW_run_flag            ;       while (run_flag == 0) {}
       if (r0 == 0) jmp $-1         ;
       if (r0 < 0) jmp _break1      ;       if (run_flag < 0) break
```

(8/10)

```
       r0l = Header                ;       ret = MP3ENC_Encode (&Header, PCM, BitstreamBuffer,
       r1l = PCM                   ;              static_x_area, static_y_area)
       r2l = __BitstreamBuffer     ;
       r3l = static_x_area         ;
       r4l = static_y_area         ;
       call MP3ENC_Encode          ;
       if (r0 < 0) jmp _exit       ;       if (ret < 0) exit(ret)
       call CopyOutputBuffer       ;       CopyOutputBuffer (ret, BitstreamBuffer)

       r1l = *frame                ;       frame++
       r1 = r1 + 1                 ;
       *frame = r1l                ;

       jmp _while1                 ;}

_break1:
;; MP3ENC_Stop process ;;
       clr (r0)                    ;
       dp0 = PCM                   ;
       rep 1152*2                  ;
              *dp0++ = r0h         ;

       r1 = r0 + PCM               ;ret = MP3ENC_Encode (&Header, PCM, BitstreamBuffer,
       r2 = r0 + __BitstreamBuffer ;       static_x_area, static_y_area)
       r0l = Header                ;
       r3l = static_x_area         ;
       r4l = static_y_area         ;
       call MP3ENC_Encode          ;
       if (r0 < 0) jmp _exit       ;if (ret < 0) exit(ret)
       call CopyOutputBuffer       ;CopyOutputBuffer (ret, BitstreamBuffer)
       r0l = Header                ;ret = MP3ENC_Encode (&Header, PCM, BitstreamBuffer,
       r1l = PCM                   ;       static_x_area, static_y_area)
       r2l = __BitstreamBuffer     ;
       r3l = static_x_area         ;
       r4l = static_y_area         ;
       call MP3ENC_Encode          ;
       if (r0 < 0) jmp _exit       ;if (ret < 0) exit(ret)
       call CopyOutputBuffer       ;CopyOutputBuffer (ret, BitstreamBuffer)

       r0l = 1                     ;FinalFrame = 1
       call MP3ENC_Stop            ;

       r0l = Header                ;ret = MP3ENC_Encode (&Header, PCM, BitstreamBuffer,
       r1l = PCM                   ;       static_x_area, static_y_area)
       r2l = __BitstreamBuffer     ;
       r3l = static_x_area         ;
       r4l = static_y_area         ;
       call MP3ENC_Encode          ;
       if (r0 < 0) jmp _exit       ;if (ret < 0) exit(ret)
       call CopyOutputBuffer       ;CopyOutputBuffer (ret, BitstreamBuffer)
```

(9/10)

```
       clr (r0)                          ;FinalFrame = 0
       call MP3ENC_Stop                  ;

       r0 = *OutputBufferFlag            ;
       if (r0 == 0) jmp _exit            ;
              r0 = *valued               ;
              *OutBitstreamBuffer:y = r0h ;
              r0l = 1                    ;
              *stream_length = r0l       ;

              clr (r1)                   ;
              *FW_out_flag = r1h         ;
              r1 = *FW_out_flag          ;
              if (r1 == 0) jmp $-1       ;
_exit:
       clr (r0)                          ;
       r0l = 0x2222                      ;length = 0x2222
       *stream_length = r0l              ;
       *FW_out_flag = r0h                ;out_flag = 0
       *FW_run_flag = r0h                ;run_flag = 0
       jmp StartUp                       ;

;; Copy from BitstreamBuffer to output buffer ;;
CopyOutputBuffer:
       if (r0 == 0) ret                  ;

       r1 = *BitstreamLength_H           ;
       r1l = *BitstreamLength_L          ;
       r1 += r0                          ;
       *BitstreamLength_H = r1h          ;
       *BitstreamLength_L = r1l          ;

       dp4 = OutBitstreamBuffer          ;wpt = OutBitstreamBuffer
       dp0 = __BitstreamBuffer           ;rpt = BitstreamBuffer
       r2 = *OutputBufferFlag            ;if (OutputBufferFlag == 0) {
       if (r2 != 0) jmp _L1              ;

       r1 = r0 sra 1                     ;      count = len >> 1
       *stream_length = r1l              ;
       r2 = r0 & 1                       ;
       loop r1l {                        ;      for ( ; count > 0; count--) {
                     r1 = *dp0++         ;             tmp = *rpt++
                     *dp4++ = r1h        ;             *wpt++ = tmp
       }                                 ;      }
       *OutputBufferFlag = r2l           ;
       if (r2 == 0) jmp _L2              ;      if (len & 1) {
              r1 = *dp0                  ;             valued = *rpt
              *valued = r1h              ;      }
```

(10/10)

```
       jmp _L2                          ;}
_L1:                                    ;else {
       r1 = r0 sra 1                    ;       count = len >> 1
       r2 = r0 & 1                      ;       if (len & 1)
       if (r2 != 0) r1 = r1 + 1         ;               count++
       *stream_length = r1l             ;
       r0 = *valued                     ;       tmp = valued << 8
       r0 = r0 sra 8                    ;
       loop r1l {                       ;       for ( ; count > 0; count--) {
                     r0l = *dp0++       ;               tmp |= *rpt++
              r0 = r0 sll 8             ;               tmp <<= 8
                     *dp4++ = r0h       ;               *wpt++ = (tmp >> 16)
              r0 = r0 sll 8             ;               tmp <<= 8
       }                                ;       }
       r0 = r0 sll 8                    ;
       *valued = r0h                    ;       valued = tmp >> 16
       clr (r1)                         ;       *wpt = valued
       if (r2 == 0) r1 = r1 + 1         ;
       *OutputBufferFlag = r1l          ;

_L2:                                    ;}

       clr (r1)                         ;
       *FW_out_flag = r1l               ;
       r1 = *FW_out_flag                ;
       if (r1 == 0) jmp $-1             ;

       ret                              ;return
end
```

A.3 Timing File for Parameter Information (value.tmg)

(1/2)

```
;;-------------------------------------------------------------------------
;;  File Information
;;-------------------------------------------------------------------------
;;  Name      : value.tmg
;;  Type      : Timing File
;;  Version   : 1.00
;;  Date      : 2002 Nov 26
;;  CPU       : uPD7701x Family
;;  Assembler : WB77016 Ver2.4
;;  About     : for MP3 audio encoder Version 1.00
;;
;;-------------------------------------------------------------------------
;;  Copyright (C) NEC Electronics Corporation 2002
;;  NEC ELECTRONICS CONFIDENTIAL AND PROPRIETARY
;;  All rights reserved by NEC Corporation.
;;  Use of copyright notice does not evidence publication
;;-------------------------------------------------------------------------

global Channel, FW_run_flag, FW_out_flag, stream_length, pcm_start, stream_start
global InstAddrBitrate, InstAddrLSF, Sampling

; MPEG1 / 32kHz / stereo
global s32_01, s32_02, s32_03

; bitrate
global b320, b256, b224, b192      ; for MPEG1
global b144, b024, b016, b008      ; for MPEG2
global b160, b128, b112, b096, b080, b064, b056, b048, b040, b032      ; for MPEG1/2

local  MPEG1_MASK, MPEG2_MASK

;;; address setting ;;;
set    Channel          = 0x0ff0;;
set    Sampling         = 0x0ff6;;
set    stream_length    = 0x0ffd;;
set    FW_out_flag      = 0x0ffe;;
set    FW_run_flag      = 0x0fff;;
set    pcm_start        = 0x2000;;
set    stream_start     = 0x2000;;

set    InstAddrBitrate  = 0x0296;;
set    InstAddrLSF      = 0x02fd;;

;;; bitrate flag setting ;;;
set    b320 = 1 << 17
set    b256 = 1 << 16
set    b224 = 1 << 15
set    b192 = 1 << 14
set    b160 = 1 << 13
set    b144 = 1 << 12
set    b128 = 1 << 11
```

(2/2)

```
set    b112 = 1 << 10
set    b096 = 1 << 9
set    b080 = 1 << 8
set    b064 = 1 << 7
set    b056 = 1 << 6
set    b048 = 1 << 5
set    b040 = 1 << 4
set    b032 = 1 << 3
set    b024 = 1 << 2
set    b016 = 1 << 1
set    b008 = 1 << 0

;;; mask flag setting ;;;
set    MPEG1_MASK = b320|b256|b224|b192|b160|b128|b112|b096|b080|b064|b056|b048|b040|b032
set    MPEG2_MASK = b160|b144|b128|b112|b096|b080|b064|b056|b048|b040|b032|b024|b016|b008

sub FlagClear
       ;;; zero clear ;;;
       set    s32_01 = 0
       set    s32_02 = 0
       set    s32_03 = 0
endsub

sub FlagMask
       ;;; masking ;;;
       set    s32_01 = s32_01 & MPEG1_MASK
       set    s32_02 = s32_02 & MPEG1_MASK
       set    s32_03 = s32_03 & MPEG1_MASK
endsub


;;; main ;;;
FlagClear
set s32_02 = b040
FlagMask

end
```

A.4 Timing File for Data Input (pcm_in.tmg)

(1/3)

```
;;-------------------------------------------------------------------------
;;  File Information
;;-------------------------------------------------------------------------
;;  Name      : pcm_in.tmg
;;  Type      : Timing File
;;  Version   : 1.00
;;  Date      : 2002 Nov 26
;;  CPU       : uPD7701x Family
;;  Assembler : WB77016 Ver2.4
;;  About     : for MP3 audio encoder Version 1.00
;;
;;-------------------------------------------------------------------------
;;  Copyright (C) NEC Electronics Corporation 2002
;;  NEC ELECTRONICS CONFIDENTIAL AND PROPRIETARY
;;  All rights reserved by NEC Corporation.
;;  Use of copyright notice does not evidence publication
;;-------------------------------------------------------------------------
global Channel, FW_run_flag, FW_out_flag, stream_length, pcm_start
global InstAddrBitrate, InstAddrLSF, Sampling

; MPEG1 / 32kHz / stereo
global s32_01, s32_02, s32_03

; bitrate
global b320, b256, b224, b192      ; for MPEG1
global b144, b024, b016, b008      ; for MPEG2
global b160, b128, b112, b096, b080, b064, b056, b048, b040, b032      ; for MPEG1/2

global size, sz
local  data, pcm
local  bitrate

sub hanten    ; (data)
       set data = ((data >> 8) & 0xff) | ((data & 0xff) << 8)
endsub

sub WriteData
       hanten
       set *pcm:x = data
       set pcm = pcm + 1
endsub

sub Writting  ; (size, sz)
       local  tmp

       set pcm = pcm_start
       if (size >= sz)
              rept sz
                     input data
                     WriteData
              endrept
```

(2/3)

```
                set size = size - sz
        else
                set tmp = sz - size
                rept size
                        input data
                        WriteData
                endrept
                set data = 0
                rept tmp
                        WriteData
                endrept
                set size = 0
        endif
endsub

sub wave2stream
        input format hex
        input size

        set *FW_run_flag:x = 1
        wait cond (*FW_run_flag:x != 1)

        set sz = 22
        Writting

        set *FW_run_flag:x = 1
        wait cond ip == InstAddrBitrate
        set r0 = bitrate

        wait cond ip == InstAddrLSF

        set sz = 576
        if (*Sampling:x < 4)
                set sz = sz << 1
        endif
        if (*Channel:x != 1)
                set sz = sz << 1
        endif

        do
                set *FW_run_flag:x = 1
                wait cond (*FW_run_flag:x != 1)

                exit (size < sz)

                Writting
        enddo

        set *FW_run_flag:x = -1
        wait cond (*FW_run_flag:x != -1)
endsub
```

(3/3)

```
;;; Wait for initialize ;;;
wait 1

if (s32_02)
       if (s32_02 & b040)
              open input "02u32.dat"
              set bitrate = 40000
              wave2stream
              close input
       endif
endif

break
end
```

A.5 Timing File for Data Output (stream_out.tmg)

(1/2)

```
;;-------------------------------------------------------------------------
;;  File Information
;;-------------------------------------------------------------------------
;;  Name      : stream_out.tmg
;;  Type      : Timing File
;;  Version   : 1.00
;;  Date      : 2002 Nov 26
;;  CPU       : uPD7701x Family
;;  Assembler : WB77016 Ver2.4
;;  About     : for MP3 audio encoder Version 1.00
;;
;;-------------------------------------------------------------------------
;;  Copyright (C) NEC Electronics Corporation 2002
;;  NEC ELECTRONICS CONFIDENTIAL AND PROPRIETARY
;;  All rights reserved by NEC Corporation.
;;  Use of copyright notice does not evidence publication
;;-------------------------------------------------------------------------
global FW_out_flag, stream_length, stream_start

; MPEG1 / 32kHz / stereo
global s32_01, s32_02, s32_03

; bitrate
global b320, b256, b224, b192      ; for MPEG1
global b144, b024, b016, b008      ; for MPEG2
global b160, b128, b112, b096, b080, b064, b056, b048, b040, b032      ; for MPEG1/2

sub wave2stream2
       local  data, rpt, length

       output #10 format dec

       do
              set *FW_out_flag:x = 1
              wait cond *FW_out_flag:x != 1

              set length = *stream_length:x

              exit length >= 0x2000

              if (length)
                     set rpt = stream_start
                     rept length
                            set data = *rpt:y
                            output #10 data
                            set rpt = rpt + 1
                     endrept
              endif
       enddo
endsub
```

(2/2)

```
;;; main ;;;
wait 1
if (s32_02)
       if (s32_02 & b040)
              open output #10 "02u32u40.dat"
              wave2stream2
              close output #10
       endif
endif

end
```

[MEMO]