NEC

User's Manual

μSAP77016-B17

AAC Audio Encoder Middleware

Target Device

μPD77110
μPD77113A
μPD77114
μPD77115
μPD77210
μPD77213

Document No. U16314EJ1V0UM00 (1st edition)
Date Published January 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 January, 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, 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, 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 EXAMPLES
- 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 hardware functions
→ 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:	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-B08 User's Manual (AAC Decoder)	U15152E
μSAP77106-B17 User's Manual (AAC Encoder)	This manual

Documents Related to Standard

Document Name	Date Published
ISO/IEC 13818-7 MPEG-2 Advanced Audio Coding, AAC	April, 1997

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-B17 is middleware that provides AAC-technology encoding functions.

1.2 AAC Audio Encoder

AAC stands for Advanced Audio Coding.

MPEG-2 AAC is an audio coding method that achieves a high quality and a high compression rate by removing compatibility with MPEG-1 audio. The μSAP77016-B17 conforms to this coding method.

The compressed data format conforms to “ISO/IEC 13818-7 MPEG-2 Advanced Audio Coding, AAC”. The audio data that is handled is 16-bit linear PCM data sampled at a frequency of 8 kHz to 96 kHz (refer to Table 1-1).

Table 1-1. Sampling Frequencies

Frequency [Hz]
8000
11025
12000
16000
22050
24000
32000
44100
48000
64000
88200
96000

1.2.1 Encoder outline

Figure 1-1 shows an example of the configuration of an encoder using the μSAP77016-B17. Figure 1-2 shows an example of the configuration of a system that includes an encoder using the μSAP77016-B17.

Figure 1-1. Encoder Configuration Example

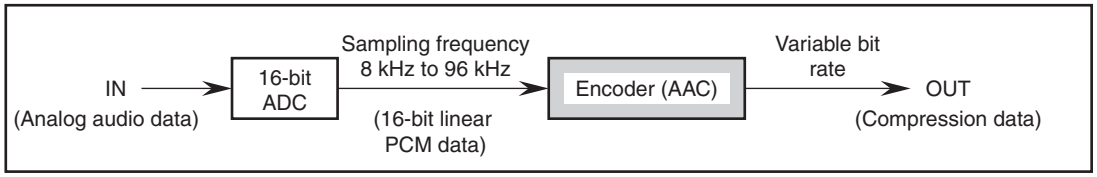
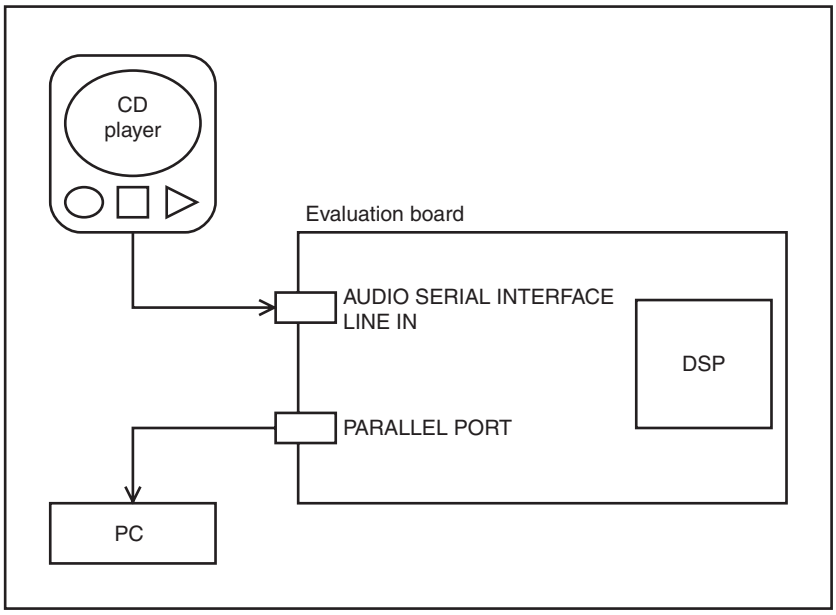


Figure 1-2. System Configuration Example



(1) Input data

The input data is 16-bit linear PCM data sampled at 8 kHz to 96 kHz (refer to Table 1-1).

(2) AAC audio encoder

The AAC 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.

The maximum value of the bit rate that can be set differs depending on the sampling frequency. The bit rate takes any value up to the maximum value. Table 1-2 shows the maximum value of the bit rate at a given sampling frequency.

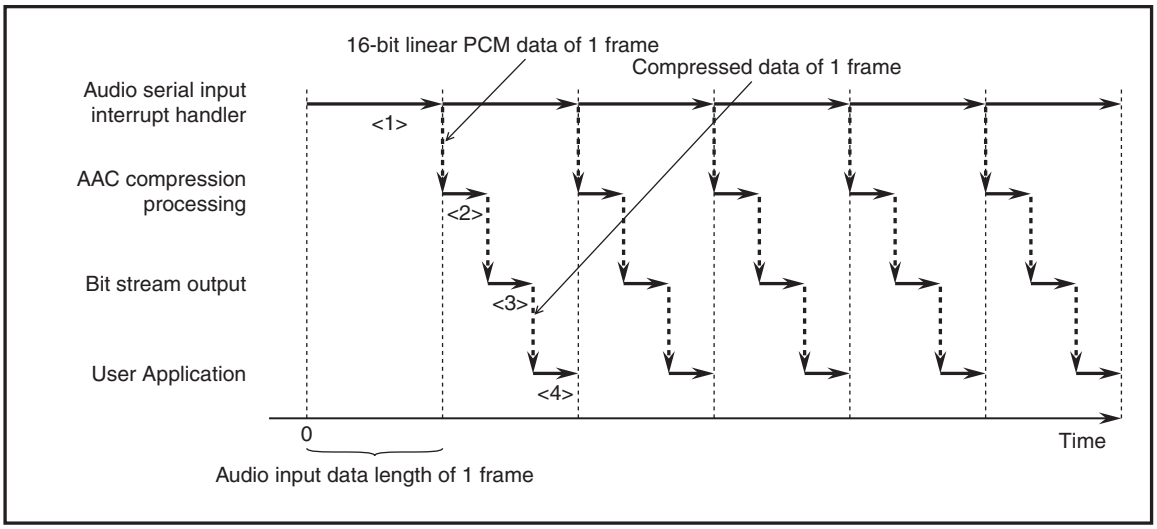
Table 1-2. Maximum Bit Rates

Sampling frequency [Hz]	Maximum bit rate [bps/ch]
8000	48000
11025	66150
12000	72000
16000	96000
22050	132300
24000	144000
32000	192000
44100	264600
48000	288000
64000	384000
88200	529200
96000	576000

(3) Timing diagram

Figure 1-3 shows the timing diagram of the AAC 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

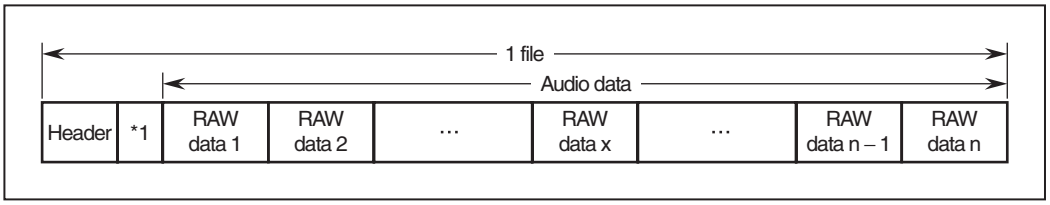
For details of the compressed data format, refer to standards (**ISO/IEC 13818-7 MPEG-2 Advanced Audio Coding, AAC**).

The μSAP77016-B17 specifications conform to standards.

1.3.1 ADIF format outline

Figure 1-4 shows the structure of ADIF format.

Figure 1-4. ADIF Format

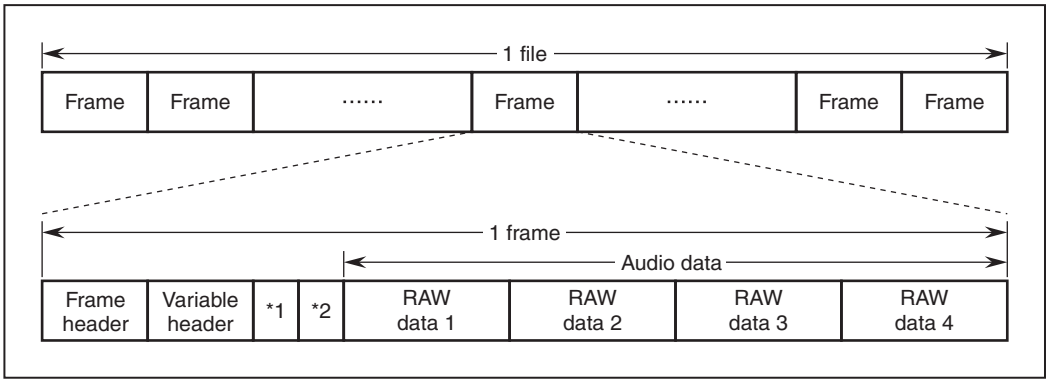


Remark Header: Contains information for synchronizing, such as sampling frequency, bit rate, and mode.
Audio data: This is information related to the audio sample. It consists of multiple RAW data bit streams.
RAW data: This is a bit stream of the smallest unit that is decoded.
*1: Bit alignment

1.3.2 ADTS format outline

Figure 1-5 shows the structure of ADTS format.

Figure 1-5. ADTS Format



Remark Frame header: Contains information for synchronizing, such as sampling frequency, bit rate, and mode.
Variable header: This is information needed in decoding, such as the number of RAW data bit streams included in the audio data.
Audio data: This is information related to the audio sample. It consists of multiple RAW data bit streams.
RAW data: This is a bit stream of the smallest unit that is decoded. There are up to four in one frame.
*1: Error check
*2: Bit alignment

1.4 Product Overview

1.4.1 Features

- Supports MPEG-2 AAC (Advanced Audio Coding) LC (Low Complexity) profile.
- Supports only two front channels (mono/stereo).
- 16-bit linear PCM data input.
- Code quantity control at set bit rate (refer to **Table 1-2 Maximum Bit Rates**) (bit rate per frame variable).
- Sampling frequency: 8 kHz to 96 kHz (refer to **Table 1-1 Sampling Frequencies**).
- Codes 1024 samples/frame in mono mode (1 channel).
- Codes 2048 samples/frame in stereo mode (2 channels).
- ADTS, ADIF, and RAW formats used as compressed data formats.
- Short block processing, TNS processing, and intensity stereo processing not supported.

1.4.2 Function

The μSAP77016-B17 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:

μSAP77016-B17 requires memory sizes shown in the following table.

Table 1-3. Required Memory Sizes

Memory	Type		Size [Kwords]
Instruction memory	–		4.8
X memory	RAM	Scratch area	4.7
		Static area	2.0
	ROM		3.4
Y memory	RAM	Scratch area	4.0
		Static area	0.1
		Library area	0.1
	ROM		2.8

Caution Place the X memory and Y memory areas used for the library in the internal ROM/RAM space.
The required memory size shown above does not include the audio data and bit stream data buffers. Refer to 2.5.3 I/O buffers.

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

(4) Software tools (Windows® version)

Table 1-4. 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 1-5 shows the MIPS values (measured values) necessary for real-time execution of processing of one frame.

- Measurement conditions

  Simulator: HSM77016 (μPD77016 high-speed simulator)

  Sampling frequency: 32 kHz, 44.1 kHz, 48 kHz

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

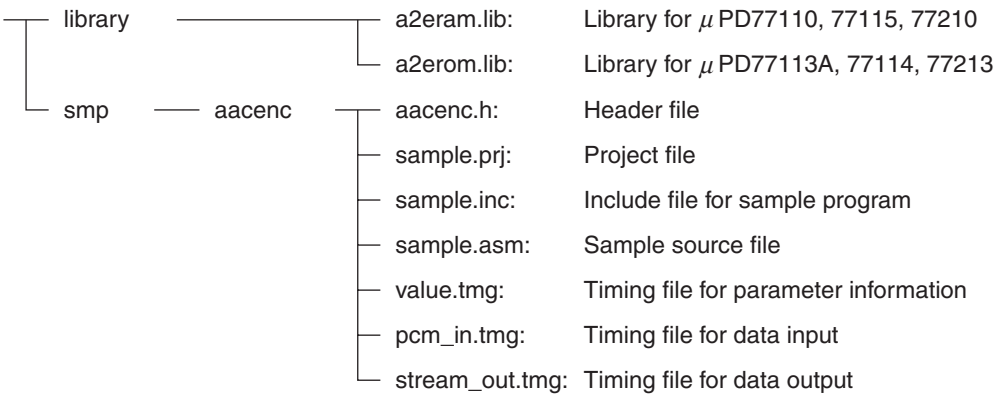
Table 1-5. MIPS Values of 1-Frame Compression Processing (Measured Values)

Sampling frequency [kHz]		32			44.1			48		
Setting bit rate [kbps]		64	96	128	64	96	128	64	96	128
Average value [MIPS]	Stereo	24.9	27.9	28.4	30.7	33.5	37.5	33.5	36.7	41.2
	Mono	13.4	13.7	14.8	17.9	18.2	18.8	19.1	19.7	20.7
Maximum value [MIPS]	Stereo	42.2	49.8	51.2	50.3	58.5	67.7	54.6	63.8	73.8
	Mono	24.9	26.9	28.5	33.1	34.6	36.4	35.3	36.5	39.5

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-B17 is shown below.



A summary of each directory is shown below.

- library
  This directory contains library files.

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

CHAPTER 2 LIBRARY SPECIFICATIONS

2.1 Library Overview

μSAP77016-B17 provides the following three functions.

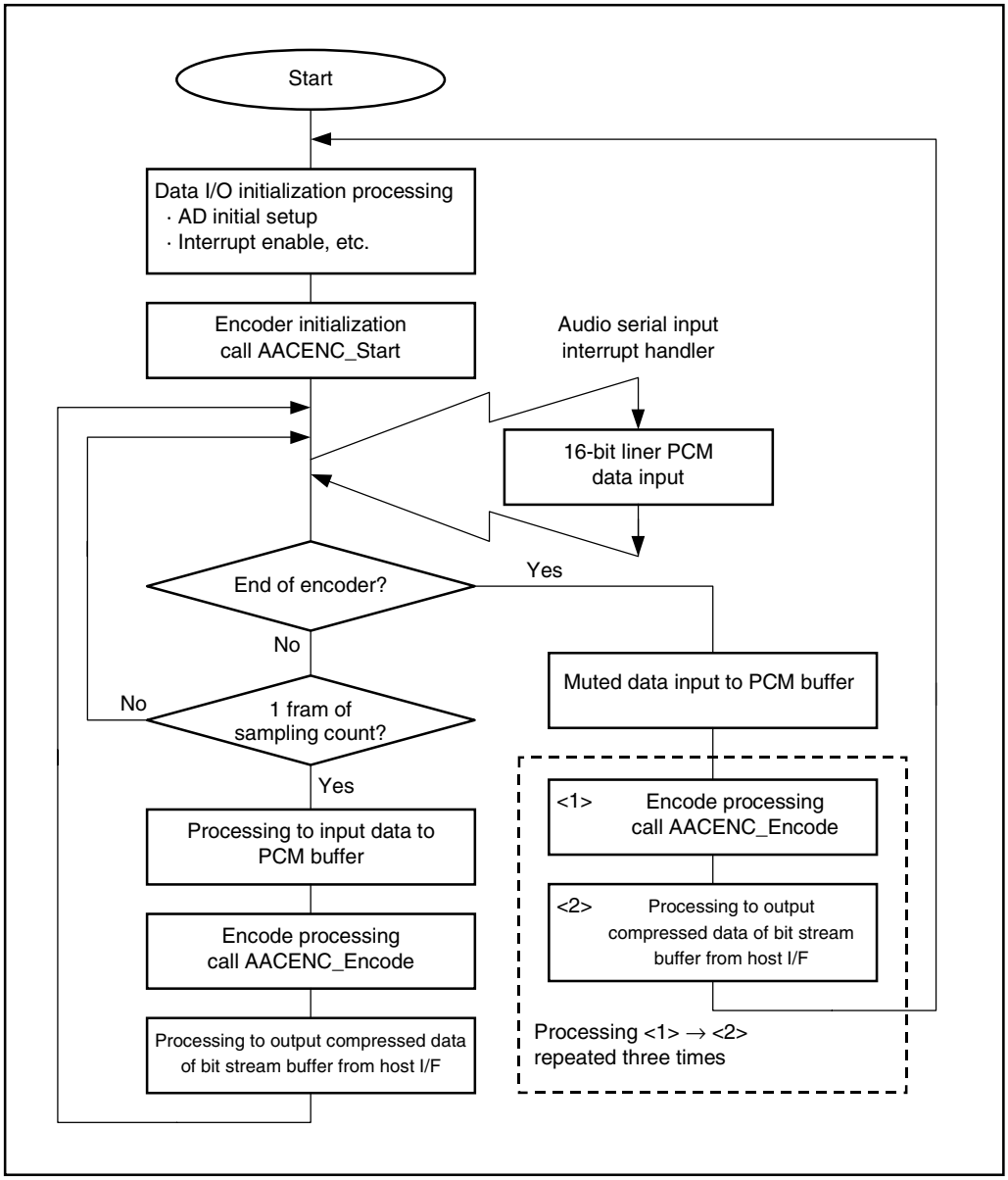
Table 2-1. List of Library Functions

Function Name	Function
AACENC_Start	Initialization
AACENC_Encode	Compression process
AACENC_GetVersion	Obtain version information

2.2 Application Processing Flow

Figure 2-1 shows an example of the processing of an application that uses μSAP77016-B17.

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 AACENC_Start function

The AACENC_Start function initializes each parameter used by the encoder. It outputs the first header in the ADIF format to the bit stream buffer only in the ADIF format. Call this function only once before using the AACENC_Encode function.

[Classification] AAC encoder initialization processing

[Function name] AACENC_Start

[Summary of function] Initializes the parameters used by the μSAP77016-B17 (bit rate, number of audio channels, sampling frequency, compression format, compression method, and buffer pointer used by μSAP77016-B17).

If the compression format is ADIF only, this function outputs the first header in the ADIF format to the bit stream buffer.

[Format] call AACENC_Start

[Arguments]

- R0L First address of structure consisting of parameters necessary for compressing in X memory[Note 1]
- R1L First address of output buffer (bit stream buffer) for encoder in X memory[Note 2].
  The ADIF header is output to this bit stream buffer only when encoding is performed in the ADIF format.
- R2L First address of static area in X memory[Note 2]
- R3L First address of scratch area in X memory[Note 2]
- R4L First address of static area in Y memory[Note 2]
- R5L First address of scratch area in Y memory[Note 2]

[Return value] R0L When 0 or greater: Size of compressed bit stream (number of bytes)

When negative: Error

An error is returned if any of the following conditions is satisfied.

- If a value other than 0 or 1 is set to the speaker_config member of the structure consisting of the parameters necessary for compression[Note 1].
- If a sampling frequency other than those supported by the μSAP77016-B17 is set to the higher word and lower word of the sampling_freq member of the structure consisting of the parameters necessary for compression[Note 1].

[Registers used] r0, r1, r2, r3, r5, dp0, dp4

[Hardware resources]

Maximum stack level	2
Maximum loop stack level	1
Maximum number of repeats	7
Maximum number of cycles	13937

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

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

Caution Secure a memory area before calling this function.

2.3.2 AACENC_Encode function

The AACENC_Encode function compresses audio data of 16 bits × 2048 samples (stereo) or 16 bits × 1024 samples (mono) at a specified bit rate.

[Classification]	AAC encode processing	
[Function name]	AACENC_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 AACENC_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]
	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 If the AACENC_Encode function is called after the AACENC_Start function has returned an error, the AACENC_Encode function also returns an error.
[Registers used]	r0, r1, r2, r3, r4, r5, dp0, dp1, dp2, dp3, dp4, dp5, dp6, dp7, dn0, dn1, dn2, dn3, dn4, dn5, dn6, dn7, dmx, dmy	
[Hardware resources]	Maximum stack level	6
	Maximum loop stack level	3
	Maximum number of repeats	51
	Maximum MIPS value	73.8 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 buffer, refer to **2.5 Memory Structure**.

Caution On completion of encoding, the user should input muted data of 1 frame (2048 words (stereo)/1024 words (mono)) to the PCM buffer and call the AACENC_Encode function three times. This can prevent information of the previous audio data from affecting compression of the next audio data.

2.3.3 AACENC_GetVersion function

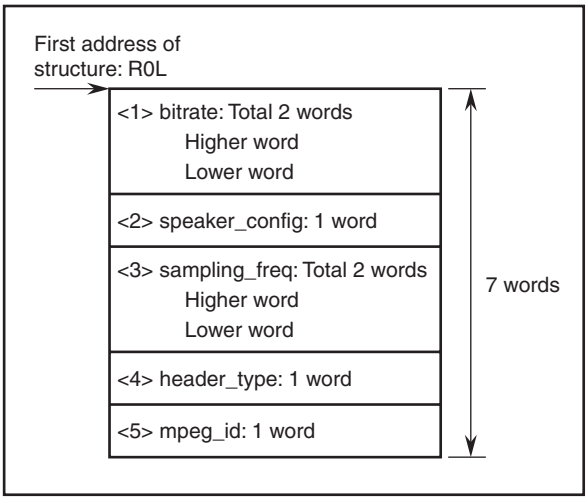
The AACENC_GetVersion function returns the version number of the μSAP77016-B17.

[Classification]	Version information acquisition	
[Function name]	AACENC_GetVersion	
[Function]	Returns the version number of the μSAP77016-B17 library in a 32-bit value.	
	Version when R0 = 0x00'0x0001'0x0100: V1.01	
[Format]	AACENC_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 on each parameter to this structure before calling the AACENC_Start and AACENC_Encoder functions.

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



<1> bitrate (2 words): This parameter sets the bit rate (bps).

Example: To set the bit rate to 264600 bps (0t264600 = 0x40998), set the higher word to 0x0004 and the lower word to 0x0998.

<2> speaker_config (1 word): This parameter sets the number of audio channels. If a value other than 0 or 1 is set to this parameter, the AACENC_Start function returns an error.

0: Mono (1 ch)

1: Stereo (2 chs)

Example To select stereo (2 chs), set speaker_config to 0x0001.

<3> sampling_freq (2 words): This parameter specifies the sampling frequency (Hz). The frequencies that can be specified are listed in Table 2-2 below.

If a frequency not listed in this table is specified, the AACENC_Start function returns an error.

Table 2-2. Sampling Frequencies

Frequency [Hz]
8000
11025
12000
16000
22050
24000
32000
44100
48000
64000
88200
96000

Example To set the bit rate to 96000 Hz (0t96000 = 0x17700), set the higher word of sampling_freq to 0x0001, and the lower word to 0x7700.

<4> header_type (1 word): This parameter specifies the header type. The following header types can be specified.

0: RAW format (without header)

1: ADIF format

2: ADTS format

Example To select the ADTS format, set header_type to 0x0002.

<5> mpeg_id (1 word): This parameter specifies the compression method. The μSAP77016-B17 does not support MPEG-4/AAC. Therefore, be sure to set mpeg_id to 0x0001 .

0: Setting prohibited (MPEG-4/AAC)

1: MPEG-2/AAC

2.5 Memory Structure

With the μSAP77016-B17, the user must define the memory area and I/O 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	4756	X	Scratch area
scratch_y_area	4096	Y	Scratch area
static_x_area	2048	X	Static area
static_y_area	5	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 audio data and bit stream data buffers. Refer to 2.5.3 I/O buffers.

2.5.1 Scratch area

The scratch area is a memory area that can be discarded when it is not used by the μSAP77016-B17.

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

When the μSAP77016-B17 uses this area again, however, the information set by the user to this area is not guaranteed.

Example
```
LIB_SCRATCH_X  xramseg
scratch_x_area:    ds    4756

LIB_SCRATCH_Y  yramseg
scratch_y_area:    ds    4096
```

2.5.2 Static area

The static area is a memory area that cannot be discarded even when the μSAP77016-B17 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-B17 cannot be guaranteed.

Example
```
LIB_STATIC_X   xramseg
static_x_area:     ds    2048

LIB_STATIC_Y   yramseg
static_y_area:     ds    5
```

2.5.3 I/O buffers

The I/O buffer is an area to which audio data (16-bit linear PCM data) is input and from which bit stream data is output. The user must secure an area for the I/O buffers in X memory.

The user can freely use the I/O buffer area after encoding processing of one frame.

If the audio data or bit stream data area is manipulated during encoding processing, the correct operation is not guaranteed.

Example Necessary I/O buffers

```
MAIN_X_WORK     xramseg
    sPCM:               ds      2048
    __BitstreamBuffer:  ds      768
```

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

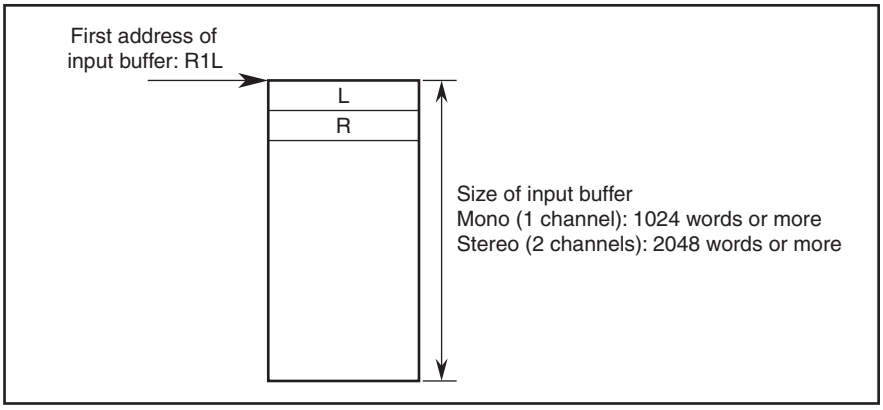
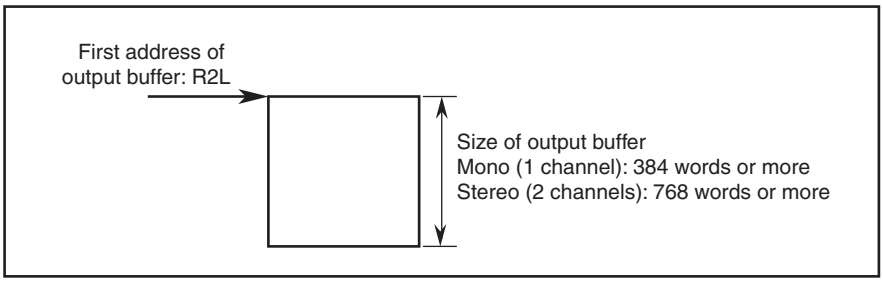


Figure 2-4. User-Defined Output Buffer (Bit Stream Buffer)



Caution For mono (when the number of channels is 1), set the size of the PCM buffer to 1024 words or more. Also secure 384 words or more for the bit stream buffer.
For stereo (when the number of channels is 2), set the size of the PCM buffer to 2048 words or more. Also secure 768 words or more for the bit stream buffer.
If a buffer of the necessary size is not secured, the output result cannot be guaranteed.

CHAPTER 3 INSTALLATION

3.1 Installation Procedure

The μSAP77016-B17 (AAC encoder middleware) is supplied on a CD-ROM. The procedure for installing the μSAP77016-B17 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 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 the AAC decoder middleware sample program.

(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, pcmin.tmg, streamout.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-B17. 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	AACENC_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-B17 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 (aacenc.h)

```
#ifndef   __aacenc_h
#define   __aacenc_h

#define   AACENC_STATIC_AREA_X_SIZE    2048
#define   AACENC_STATIC_AREA_Y_SIZE    5
#define   AACENC_SCRATCH_AREA_X_SIZE   4756
#define   AACENC_SCRATCH_AREA_Y_SIZE   4096

#define   HeaderSize      7

extrn     AACENC_Start
extrn     AACENC_Encode
extrn     AACENC_GetVersion

#endif
```

A.2 Include File for Sample Program (sample.inc)

(1/2)

```
VECTOR   imseg at 0x200

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

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

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            ;
```

(2/2)

```
%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           ;
        dn1 = r0l           ;
        dn2 = r0l           ;
        dn3 = r0l           ;
        dn4 = r0l           ;
        dn5 = r0l           ;
        dn6 = r0l           ;
        dn7 = r0l           ;
        dmx = r0l           ;
        dmy = r0l           ;
)
```

A.3 Sample Source File (sample.asm)

(1/5)

```
#include "aacenc.h"

LIB_STATIC_X xramseg
       static_x_area:           ds     AACENC_STATIC_AREA_X_SIZE

LIB_STATIC_Y yramseg
       static_y_area:           ds     AACENC_STATIC_AREA_Y_SIZE

LIB_SCRATCH_X      xramseg
       scratch_x_area:          ds     AACENC_SCRATCH_AREA_X_SIZE

LIB_SCRATCH_Y      yramseg
       scratch_y_area:          ds     AACENC_SCRATCH_AREA_Y_SIZE

MAIN_X_WORK  xramseg at 0x6000
       sPCM:                    ds     1024*2
       __BitstreamBuffer:       ds     2048/2

       Header:
       _Header_bitrate_H:       ds     1
       _Header_bitrate_L:       ds     1
       _Header_speaker_config:  ds     1
       _Header_sampling_freq_H: ds     1
       _Header_sampling_freq_L: ds     1
       _Header_header_type:     ds     1
       _Header_mpeg_id:         ds     1
       _OutputBufferFlag:       ds     1
       _valued:                 ds     1
       _frame_number:           ds     1

MAIN_CTRL_WORK     xramseg at 0x5ffc
       _s:                      ds     1      ; Reference value of timing files
       _stream_length:          ds     1      ; Reference value of timing files
       _FW_out_flag:            ds     1      ; Reference value of timing files
       _FW_run_flag:            ds     1      ; Reference value of timing files

#defineHeader_bitrate_H         _Header_bitrate_H:x
#defineHeader_bitrate_L         _Header_bitrate_L:x
#defineHeader_speaker_config    _Header_speaker_config:x
#defineHeader_sampling_freq_H   _Header_sampling_freq_H:x
#defineHeader_sampling_freq_L   _Header_sampling_freq_L:x
#defineHeader_header_type       _Header_header_type:x
#defineHeader_mpeg_id           _Header_mpeg_id:x
#defineOutputBufferFlag         _OutputBufferFlag:x
#definevalued                   _valued:x
#defineframe_number             _frame_number:x

#defines                        _s:x
#definestream_length            _stream_length:x
#defineFW_out_flag              _FW_out_flag:x
#defineFW_run_flag              _FW_run_flag:x
```

(2/5)

```
#include " sample.inc"

MAIN  imseg at 0x240

StartUp:
      %Initialize                              ;

      clr (r0)                                 ;
      dp0 = __BitstreamBuffer                  ;
      rep 2048/2                               ;
                  *dp0++ = r0h                 ;
      dp0 = sPCM                               ;
      rep 1024*2                               ;
                  *dp0++ = r0h                 ;

      dp4 = 0x4000                             ;
      rep 0x2000                               ;
                  *dp4++ = r0h                 ;

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

      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+6+2+8 = 44byte -> 22word
      dp0 = sPCM                               ;
      rep 22/2                                 ;fread (header, 1, 22, fi)
                  r0 = *dp0++                  ;
                  r0 = *dp0++                  ;fread (&s, 1, 2, fi)
      *s = r0h                                 ;
      r1 = r0 sra 16                           ;Header.speaker_config = (s == 1) ? 0 : 1
      r1 = r1 - 1                              ;
      clr (r0)                                 ;
      if (r1 != 0) r0 = r0 + 1                 ;
      *Header_speaker_config = r0l             ;
                  r0 = *dp0++                  ;fread (&i, 1, 4, fi)
                  r0l = *dp0++                 ;
      *Header_sampling_freq_H = r0l            ;Header.sampling_freq = i
      *Header_sampling_freq_L = r0h            ;
                                               ;fread (tmp, 1, 6+2+8, fi)

      clr (r0)                                 ;
      r0l = 64000                              ;64kbps
```

(3/5)

```
nop;
      *Header_bitrate_H = r0h          ;
      *Header_bitrate_L = r0l          ;
      r0l = 1                          ;MPEG-2 AAC
      *Header_mpeg_id = r0l            ;
      r0l = 2                          ;header_type = 2
      *Header_header_type = r0l        ;
      r0l = Header                     ;ret = AACENC_Start (&Header, BitstreamBuffer)
      r1l = __BitstreamBuffer          ;

      r2l = static_x_area              ;
      r3l = scratch_x_area             ;
      r4l = static_y_area              ;
      r5l = scratch_y_area             ;

      call AACENC_Start                ;
      if (r0 < 0) jmp _exit            ;
      call CopyOutputBuffer            ;CopyOutputBuffer (ret, BitstreamBuffer)

_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

      r0l = Header                     ;  ret = AACENC_Encode (&Header, sPCM, BitstreamBuffer)
      r1l = sPCM                       ;
      r2l = __BitstreamBuffer          ;

      r3l = static_x_area              ;
      r4l = static_y_area              ;

      call AACENC_Encode               ;
      if (r0 < 0) jmp _exit            ;  if (ret < 0) exit(ret)
      call CopyOutputBuffer            ;  CopyOutputBuffer (ret, BitstreamBuffer)

      r1l = *frame_number              ;  frame_numfer++
      r1 = r1 + 1                      ;
      *frame_number = r1l              ;

      jmp _while1                      ;}

_break1:
      dp0 = sPCM                       ;Clear (0) sPCM area
      clr (r0)                         ;for (i = 0; i < 2048; i++) {
      rep 2048                         ;  sPCM[i] = 0
                  *dp0++ = r0h         ;}
```

(4/5)

```
      r0l = Header                      ;ret = AACENC_Encode (&Header, sPCM, BitstreamBuffer)
      r1l = sPCM                        ;
      r2l = __BitstreamBuffer           ;

      r3l = static_x_area               ;
      r4l = static_y_area               ;

      call AACENC_Encode                ;
      if (r0 < 0) jmp _exit             ;if (ret < 0) exit(ret)
      call CopyOutputBuffer             ;CopyOutputBuffer (ret, BitstreamBuffer)

      r0l = Header                      ;ret = AACENC_Encode (&Header, sPCM, BitstreamBuffer)
      r1l = sPCM                        ;
      r2l = __BitstreamBuffer           ;

      r3l = static_x_area               ;
      r4l = static_y_area               ;

      call AACENC_Encode                ;
      if (r0 < 0) jmp _exit             ;if (ret < 0) exit(ret)
      call CopyOutputBuffer             ;CopyOutputBuffer (ret, BitstreamBuffer)

      r0l = Header                      ;ret = AACENC_Encode (&Header, sPCM, BitstreamBuffer)
      r1l = sPCM                        ;
      r2l = __BitstreamBuffer           ;

      r3l = static_x_area               ;
      r4l = static_y_area               ;

      call AACENC_Encode                ;
      if (r0 < 0) jmp _exit             ;if (ret < 0) exit(ret)
      call CopyOutputBuffer             ;CopyOutputBuffer (ret, BitstreamBuffer)

      r0 = *OutputBufferFlag            ;
      if (r0 == 0) jmp _exit            ;
            r0 = *valued                ;
            *0x4000: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                       ;
```

(5/5)

```
;; Copy from BitstreamBuffer to output buffer ;;
CopyOutputBuffer:
      if (r0 == 0) ret                    ;
      dp4 = 0x4000                        ;wpt = 0x4000
      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           ;  }

      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 (r0)                            ;
      *FW_out_flag = r0l                  ;
      r0 = *FW_out_flag                   ;
      if (r0 == 0) jmp $-1                ;

      ret                                 ;return
end
```

A.4 Timing File for Parameter Information (value.tmg)

```
global   s, FW_run_flag, FW_out_flag, stream_length, pcm_start, stream_start

global   out44_128, out44_96, out44_80, out44_64
global   out32_96, out32_64, out32_48, out32_40

global   flag01, flag02, flag03, flag04, flag05
global   flag08, flag09, flag12, flag13, flag19
global   flag21

set s                  = 0x5ffc
set stream_length  = 0x5ffd
set FW_out_flag    = 0x5ffe
set FW_run_flag    = 0x5fff
set pcm_start          = 0x6000
set stream_start   = 0x4000

set out44_128  = 1 << 7
set out44_96   = 1 << 6
set out44_80   = 1 << 5
set out44_64   = 1 << 4
set out32_96   = 1 << 3
set out32_64   = 1 << 2
set out32_48   = 1 << 1
set out32_40   = 1 << 0

set flag01 = 0
set flag02 = 0
set flag03 = 0
set flag04 = 0
set flag05 = 0
set flag08 = 0
set flag09 = 0
set flag12 = 0
set flag13 = 0
set flag19 = 0
set flag21 = 0

set flag01 = out44_128 | out44_96 | out44_80 | out44_64 | out32_96 | out32_64 | out32_48 | out32_40
set flag02 = out44_128 | out44_96 | out44_80 | out44_64 | out32_96 | out32_64 | out32_48 | out32_40
set flag03 = out44_128 | out44_96 | out44_80 | out44_64 | out32_96 | out32_64 | out32_48 | out32_40
set flag04 = out44_128 | out44_96 | out44_80 | out44_64 | out32_96 | out32_64 | out32_48 | out32_40
set flag05 = out44_128 | out44_96 | out44_80 | out44_64 | out32_96 | out32_64 | out32_48 | out32_40
set flag08 = out44_128 | out44_96 | out44_80 | out44_64 | out32_96 | out32_64 | out32_48 | out32_40
set flag09 = out44_128 | out44_96 | out44_80 | out44_64 | out32_96 | out32_64 | out32_48 | out32_40
set flag12 = out44_128 | out44_96 | out44_80 | out44_64 | out32_96 | out32_64 | out32_48 | out32_40
set flag13 = out44_128 | out44_96 | out44_80 | out44_64 | out32_96 | out32_64 | out32_48 | out32_40
set flag19 = out44_128 | out44_96 | out44_80 | out44_64 | out32_96 | out32_64 | out32_48 | out32_40
set flag21 = out44_128 | out44_96 | out44_80 | out44_64 | out32_96 | out32_64 | out32_48 | out32_40

end
```

A.5 Timing File for Data Input (pcm_in.tmg)

(1/2)

```
global   s, FW_run_flag, FW_out_flag, stream_length, pcm_start
global   out44_128, out44_96, out44_80, out44_64
global   out32_96, out32_64, out32_48, out32_40
global   flag01, flag02, flag03, flag04, flag05
global   flag08, flag09, flag12, flag13, flag19
global   flag21
local    size, data, pcm
local    sz
local    bitrate
local    addr

set addr = 0x27c

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
                  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
```

(2/2)

```
sub wave2stream
        input format hex
        input size

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

        set sz = 22
        Writting

        if (*s:x == 1)
                set sz = 1024
        else
                set sz = 2048
        endif

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

        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


;;; Wait for initialize ;;;
wait 1

if (flag02)
        if (flag02 & out32_40)
                ;;; 02.wav / 32kHz / 40kbps ;;;
                open input "02_32.dat"
                set bitrate = 40000
                wave2stream
                close input
        endif
endif

break
end
```

A.6 Timing File for Data Output (stream_out.tmg)

```
global   FW_out_flag, stream_length, stream_start
global   out44_128, out44_96, out44_80, out44_64
global   out32_96, out32_64, out32_48, out32_40
global   flag01, flag02, flag03, flag04, flag05
global   flag08, flag09, flag12, flag13, flag19
global   flag21

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


;;; main ;;;
wait 1
if (flag02)
         ;;; 02.wav ;;;
         if (flag02 & out32_40)
                  open output #10 "02_32_40.dat"
                  wave2stream2
                  close output #10
         endif
endif

end
```

[MEMO]