To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

User's Manual

μSAP77016-B06

AMR Speech Codec Middleware

Target Devices

μPD77018A
μPD77019
μPD77110
μPD77111
μPD77112
μPD77113A
μPD77114

Document No. U15165EJ3V0UM00 (3rd edition)
Date Published June 2002 N CP(N)

[MEMO]

Windows and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Regional Information

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

J02.4

Major Revisions in This Edition

Page	Contents
Throughout	Deletion of μPD77018 and 77113 from the target device
p.12	Change of **1.3.1 Features**
p.12	Change of **1.3.2 (2) Required memory size**
p.13	Change of **Table 1-3 Operation Quantity of AMR Speech CODEC**
p.14	Change of **1.3.4 Directory configuration**
p.15	Change of **1.3.5 Combination of libraries**
p.20	**2.2.3 amr_EncodeFrame function** Change of hardware resource value
p.21	**2.2.4 amr_InitDecoder function** Change of hardware resource value
p.22	**2.2.5 amr_ResetDecoder function** Change of hardware resource value
p.23	**2.2.6 amr_DecodeFrame function** Change of hardware resource value
p.23	Change of **Table 2-3 Receive Frame Type**
p.25	Change of **Table 2-4 Transmit Frame Type**
p.26	**2.2.10 amr_TX_to_RX function** Change of register name and hardware resource value
p.26	Change of **Table 2-5 Transmit/Receive Frame Type**
p.28	**2.2.13 amr_GetVersion function** Change of the description of function
p.44	Change of **APPENDIX A SAMPLE PROGRAM SOURCE**

The mark ★ shows major revised points.

PREFACE

Target Readers

This manual is intended for users who wish to design and develop application systems using the μPD77016 Family.

The μPD77016 Family includes the μPD7701X Family (μPD77015, 77016, 77017, 77018A, 77019) and the μPD77111 Family (μPD77110, 77111, 77112, 77113A, 77114, 77115). This manual, however, only covers the μPD77018A, 77019, 77110, 77111, 77112, 77113A, and 77114.

Purpose

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

Organization

This manual consists of the following sections.

CHAPTER 1 INTRODUCTION
CHAPTER 2 LIBRARY SPECIFICATIONS
CHAPTER 3 BIT STREAM FORMAT
CHAPTER 4 INSTALLATION
APPENDIX A SAMPLE PROGRAM SOURCE
APPENDIX B RELATED DOCUMENTS

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 μPD7701X Family hardware functions
→ Refer to **μPD7701X Family User's Manual Architecture**.

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

To learn about μPD77016 Family instruction 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:	$\overline{\times\times\times}$ (overscore over pin or signal name)
Note:	Footnote for item marked with **Note** in the text
Caution:	Information requiring particular attention
Remark:	Supplementary information
Number representation:	Binary ×××× or 0b××××
	Decimal ××××
	Hexadecimal 0x××××

Related Documents The related documents indicated in this publication may include preliminary versions. However, preliminary versions are not marked as such.

Documents Related to μPD77016 Family

Document Name / Part Number	Pamphlet	Data Sheet	User's Manual		Application Note
			Architecture	Instructions	Basic Software
μPD77016	–	U10891E	U10503E	U13116E	U11958E
μPD77015		U10902E			
μPD77017					
μPD77018A		U11849E			
μPD77019					
μPD77019-013		U13053E			
μPD77110	U12395E	U12801E	U14623E		
μPD77111					
μPD77112					
μPD77113A		U14373E			
μPD77114					
μPD77115		U14867E			

Documents Related to Development Tools

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

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

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 INTRODUCTION

1.1 Middleware

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

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

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

1.2 AMR Speech CODEC

The AMR speech CODEC is a 4.75 kbps to 12.2 kbps speech compression/decompression codec standardized by 3GPP (3rd Generation Partnership Project). It includes a multi-rate speech coder, a silence compression function, and an error concealment function. The multi-rate speech coder enables selection of the bit rate from a total of 8 (refer to Table 1-1), except MRDTX[Note 1], and allows bit rate switching on a frame by frame (20 ms) basis.

Table 1-1. AMR Speech CODEC Bit Rates

Codec Mode	Bit Rate
MR122	12.20 kbps (GSM EFR[Note 2])
MR102	10.20 kbps
MR795	7.95 kbps
MR74	7.40 kbps (IS-641[Note 3])
MR67	6.70 kbps (PDC-EFR[Note 4])
MR59	5.90 kbps
MR515	5.15 kbps
MR475	4.75 kbps
MRDTX[Note 1]	1.80 kbps

Notes 1. The encoder automatically executes compression in the MRDTX mode if the silence compression function is set to ON.

2. GSM EFR: ETSI GSM 06.90 Enhanced Full Rate Speech Codec

3. IS-641: TIA/EIA IS-641 TDMA Enhanced Full Rate Speech Codec

4. PDC-EFR: ARIB 6.7 kbps Enhanced Full Rate Speech Codec

Remark For details of each standard, refer to the reports in **APPENDIX B RELATED DOCUMENTS**.

1.3 Outline of System

1.3.1 Features

★ • Supports AMR speech CODEC Version 7.6.0 of 3GPP (excluding channel codec).
• Supports eight bit rates for compression/decompression (refer to **Table 1-1 AMR Speech CODEC Bit Rates**).
• Silence compression function (VAD1 and VAD2 options supported)
• Coding/decoding of 160 samples/frame at sampling frequency of 8 kHz
• All speech I/O data is 16-bit linear PCM data[Note].

Note In the μSAP77016-B06 library, 13-bit data from the MSB is used.

1.3.2 Operating environment

★ **(1) Target DSP**

• μPD77018A, 77019, 77110, 77111, 77112, 77113A, 77114

★ **(2) Required memory size**

Table 1-2. Required Memory Size

Memory Space	Type	Size [Words]		
		Codec	Encoder	Decoder
Instruction memory	–	14.1 K	10.2 K	4.4 K
X memory	RAM	2.9 K	2.5 K	2.4 K
	ROM	11.2 K	11.1 K	10.7 K
Y memory	RAM	2.4 K	1.9 K	1.5 K
	ROM	3.6 K	3.2 K	2.8 K

Remarks 1. Locate the X memory and Y memory areas used for the library of the μSAP77016-B06 in the internal ROM/RAM space.

2. The required memory size shown in the table above does not include the buffers for speech data and bit stream data I/O. For details, refer to **2.3 Memory Structure**.

(3) Software tools (Windows™ version)

DSP tools: Work bench WB77016
High-speed simulator HSM77016
Debugger ID77016

1.3.3 Performance

Table 1-3 shows the MIPS values necessary for executing processing of one frame in real time (20 ms).

[Measurement conditions]

- High-speed simulator: HSM77016
- Evaluated speech: Speech file of 3GPP Test Sequences (TS 26.074) is used.
- Evaluation result: Operation quantity is measured when the speech file is compressed and decompressed, to calculate the worst case value.
- The operation quantity for compression is that of the amr_EncodeFrame function and the operation quantity for decompression is that of the amr_DecodeFrame function. The operation quantity of other functions and interrupt handlers is not included.

★ **Table 1-3. Operation Quantity of AMR Speech CODEC**

Codec Mode	VAD OFF [MIPS]		VAD1 ON [MIPS]		VAD2 ON [MIPS]	
	Compression	Decompression	Compression	Decompression	Compression	Decompression
MR475	15.116	2.658	15.658	2.658	16.749	2.683
MR515	12.000	2.675	12.542	2.675	13.635	2.675
MR59	14.901	2.674	15.425	2.674	16.552	2.661
MR67	19.331	2.740	19.817	2.740	20.970	2.745
MR74	17.371	2.408	17.864	2.408	18.992	2.407
MR795	18.318	2.783	18.830	2.783	19.934	2.835
MR102	18.548	2.495	19.054	2.495	20.161	2.495
MR122	19.997	2.489	20.510	2.489	21.626	2.489

⋆ 1.3.4 Directory configuration

The directory configuration of the μSAP77016-B06 is shown below.

```
─┬─ <include> ──────────────┬─ amr_lib.h:       Header file for Codec library
 │                          ├─ amr_libe.h:      Header file for Encodecr library
 │                          └─ amr_libd.h:      Header file for Decoder library
 │
 ├─ <library> ──────────────┬─ amrcodec.lib:    Codec library
 │                          ├─ amrcodera.lib:   Constant data library for Codec
 │                          │                   library RAM segment
 │                          ├─ amrcodero.lib:   Constant data library for Codec
 │                          │                   library ROM segment
 │                          ├─ amrencod.lib:    Encoder library
 │                          ├─ amrencra.lib:    Constant data library for Encodder
 │                          │                   library RAM segment
 │                          ├─ amrencro.lib:    Constant data library for Encodder
 │                          │                   library ROM segment
 │                          ├─ amrdecod.lib:    Decoder library
 │                          ├─ amrdecra.lib:    Constant data library for Decoder
 │                          │                   library RAM segment
 │                          └─ amrdecro.lib:    Constant data library for Decoder
 │                                              library ROM segment
 │
 └─ <smp> ── <amr> ─┬─ <include> ─┬─ upd7701x.h:    Definition file for μPD77016 Family
                    │             └─ upd7701x.mac:  Macro definition file for μPD77016 Family
                    │
                    ├─ <single> ──┬─ sample.prj:    Project file for single channel
                    │             └─ sample.asm:    Sample source file for single channel
                    │
                    └─ <multi> ───┬─ sample_m.prj:  Project file for multiple channels
                                  ├─ sample_m.asm:  Sample source file for multiple channels
                                  ├─ amrmulti.h:    Preprocessing function definition file for multiple channels
                                  └─ amrmulti.asm:  Preprocessing function for multiple channels
```

★ ### 1.3.5 Combination of libraries

Table 1-4 shows the combination of the header file and the library to be used.

Table 1-5 shows the combination of the library and the DSP when using the Codec library.

Table 1-4. Combination of Header File and Library

Library	Header File	Code	Constant for RAM	Constant for ROM
Codec library	amr_lib.h	amrcodec.lib	amrcodra.lib	amrcodro.lib
Encoder library	amr_libe.h	amrencod.lib	amrencra.lib	amrencro.lib
Decoder library	amr_libd.h	amrdecod.lib	amrdecra.lib	amrdecro.lib

Table 1-5. Combination of Target Device and Library (Codec Library)

Device Name	Code (amrcodec.lib)	RAM Constant (amrcodra.lib)	ROM Constant (amrcodro.lib)
μPD77015	×	×	×
μPD77016	×	×	×
μPD77017	×	×	×
μPD77018A	√	×	√
μPD77019	√	×	√
μPD77110	√	√	×
μPD77111	√	×	√
μPD77112	√	×	√
μPD77113A	√	√ Note	√ Note
μPD77114	√	√ Note	√ Note
μPD77115	×	×	×

Note Link either of amrcodra.lib or amrcodro.lib.

CHAPTER 2 LIBRARY SPECIFICATIONS

This chapter explains the function specifications and calling conventions of the μSAP77016-B06.

2.1 Application Processing Flow

Figures 2-1 and 2-2 show examples of application processing using the μSAP77016-B06.

Figure 2-1. Example of Application Processing Flow (Encoder)

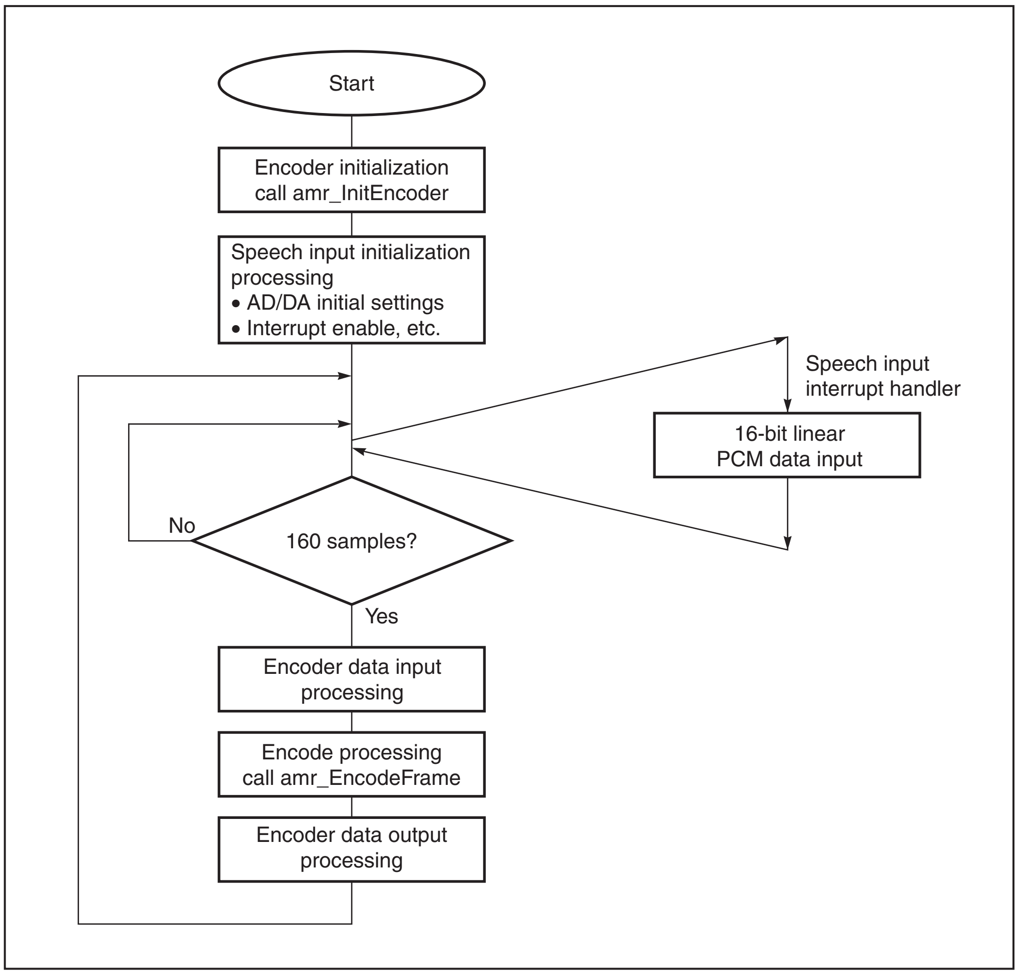
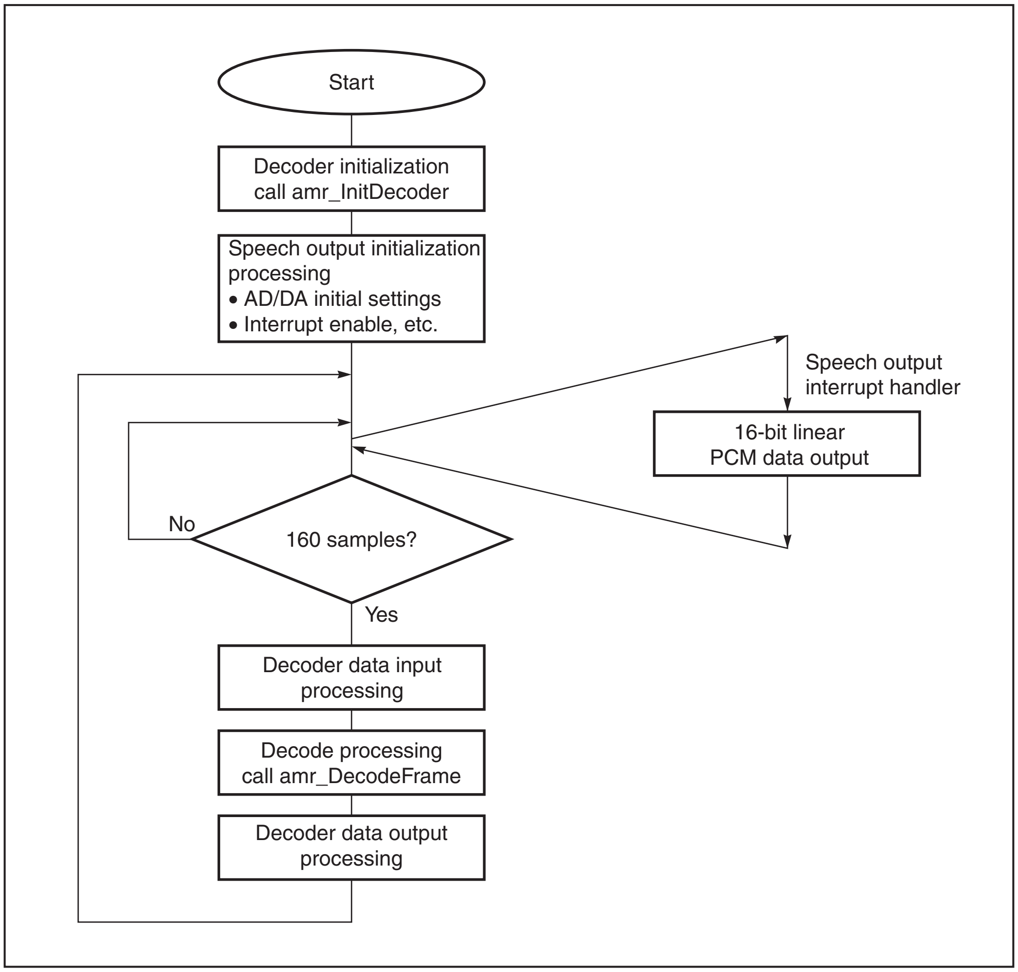


Figure 2-2. Example of Application Processing Flow (Decoder)



2.2 Function Specifications

2.2.1 amr_InitEncoder function

The amr_InitEncoder function initializes the constants, coefficient table, and buffers used for the encoder. Call this function only once when using the encoder.

[Classification] Encoder initialization function

[Function name] amr_InitEncoder

[Summary of function] Initializes the RAM area necessary for processing the AMR speech CODEC encoder, sets the VAD mode, calls the amr_ResetEncoder function, and initializes parameters.

[Format] call amr_InitEncoder

[Argument]

Type	Argument	Description
Register	R0L	Turns ON/OFF DTX (silence compression function) (see Table 2-1)

[Return value] None

[Registers used] R0, R1, DP0, DP1, DP4, DP5

[Hardware resources]
Maximum stack level: 4
Maximum loop stack level: 1
Maximum number of repetitions: 320
Maximum number of cycles: 1256

Remark When calling this function, be sure to first secure static memory and scratch memory areas. For details, refer to **2.3 Memory Structure**.

Table 2-1. DTX Mode

DTX Mode	Value	Description
DTX_OFF	0	Silent compression function OFF
DTX_ON_VAD1	1	Silent compression function ON (VAD1 option)
DTX_ON_VAD2	2	Silent compression function ON (VAD2 option)

2.2.2 amr_ResetEncoder function

The amr_ResetEncoder function resets the constants, coefficient table, and buffers used for the encoder to the initial status.

[Classification]	Encoder reset function	
[Function name]	amr_ResetEncoder	
[Summary of function]	Resets the parameters necessary for processing the AMR speech CODEC encoder to the initial status.	
[Format]	call amr_ResetEncoder	
[Argument]	None	
[Return value]	None	
[Registers used]	R0, R1, DP0, DP1, DP4, DP5	
[Hardware resources]	Maximum stack level:	3
	Maximum loop stack level:	1
	Maximum number of repetitions:	320
	Maximum number of cycles:	1248

Remark When calling this function, be sure to first secure static memory and scratch memory areas. For details, refer to **2.3 Memory Structure**.

2.2.3 amr_EncodeFrame function

The amr_EncodeFrame function compresses speech data of 16 bits × 160 samples at a specified bit rate. When the silence compression function is set to ON during initialization, the silence compression function is automatically enabled in accordance with speech input.

[Classification] Encode processing function

[Function name] amr_EncodeFrame

[Summary of function] Compresses speech of one frame (160 samples) at a specified bit rate.

[Format] call amr_EncodeFrame

[Argument]

Type	Argument	Description
Register	R0L	Encode mode (see Table 2-2, excluding MRDTX)
Register	DP4	First address of speech input buffer
Register	DP0	First address of bit stream output buffer

[Return value]

Type	Return Value	Description
Register	R0L	Encode mode used for compression (see Table 2-2)

[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: 5
Maximum loop stack level: 3
Maximum number of repetitions: 239
★ Maximum number of MIPS: 21.7

Remark When calling this function, be sure to first secure static memory and scratch memory areas. For details, refer to **2.3 Memory Structure**.

Table 2-2. Operation Mode of Encoder/Decoder

Codec Mode	Value	Description
MR475	0	Compression/decompression mode at 4.75 Kbps
MR515	1	Compression/decompression mode at 5.15 Kbps
MR59	2	Compression/decompression mode at 5.90 Kbps
MR67	3	Compression/decompression mode at 6.70 Kbps
MR74	4	Compression/decompression mode at 7.40 Kbps
MR795	5	Compression/decompression mode at 7.95 Kbps
MR102	6	Compression/decompression mode at 10.20 Kbps
MR122	7	Compression/decompression mode at 12.20 Kbps
MRDTX	8	Compression/decompression mode when silence compression function is enabled

2.2.4 amr_InitDecoder function

The amr_InitDecoder function initializes the constants, coefficient table, and buffers used for the decoder. Call this function only once when using the decoder.

[Classification]	Decoder initialization function
[Function name]	amr_InitDecoder
[Summary of function]	Initializes the RAM area necessary for processing the AMR speech CODEC decoder, calls the amr_ResetDecoder function, and initializes parameters.
[Format]	call amr_InitDecoder
[Argument]	None
[Return value]	None
[Registers used]	R0, R1, R2, DP0, DP1, DP2, DP4, DP5, DP6, DN5
[Hardware resources]	Maximum stack level: 3
	Maximum loop stack level: 1
	Maximum number of repetitions: 154
★	Maximum number of cycles: 2545

Remark When calling this function, be sure to first secure static memory and scratch memory areas. For details, refer to **2.3 Memory Structure**.

2.2.5 amr_ResetDecoder function

The amr_ResetDecoder function resets the constants, coefficient table, and buffers used for the decoder to the initial status.

[Classification]	Decoder reset function
[Function name]	amr_ResetDecoder
[Summary of function]	Resets the parameters necessary for processing the AMR speech CODEC decoder to the initial status.
[Format]	call amr_ResetDecoder
[Argument]	None
[Return value]	None
[Registers used]	R0, R1, R2, DP0, DP1, DP2, DP4, DP5, DP6, DN5
[Hardware resources]	Maximum stack level: 2
	Maximum loop stack level: 1
	Maximum number of repetitions: 154
★	Maximum number of cycles: 1009

Remark When calling this function, be sure to first secure static memory and scratch memory areas. For details, refer to **2.3 Memory Structure**.

2.2.6 amr_DecodeFrame function

The amr_DecodeFrame function decompresses compressed bit stream data to speech data of 16 bits × 160 samples.

[Classification] Decode processing function

[Function name] amr_DecodeFrame

[Summary of function] Decompresses speech of one frame (160 samples) from a bit stream.

[Format] call amr_DecodeFrame

★ **[Argument]**

Type	Argument	Description
Register	R0L	Decode mode[Note] (see **Table 2-2 Operation Mode of Encoder/ Decoder**. Except for MRDTX.)
Register	R1L	Receive frame type (see Table 2-3)
Register	DP4	First address of speech output buffer
Register	DP0	First address of bit stream input buffer

Note When the receive frame type is RX_NO_DATA, use the decode mode of the preceding frame as the argument.

[Return value] None

[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

[Hardware resources]
Maximum stack level: 5
Maximum loop stack level: 2
Maximum number of repetitions: 170
★ Maximum number of MIPS: 2.9

Remark When calling this function, be sure to first secure static memory and scratch memory areas. For details, refer to **2.3 Memory Structure**.

★ **Table 2-3. Receive Frame Type**

RX_TYPE	Value	Description
RX_SPEECH_GOOD	0	Speech frame (without error)
RX_SPEECH _DEGRADED	1	Speech frame (with error of at least 1 bit)
RX_ONSET	2	ONSET frame
RX_SPEECH_BAD	3	Speech frame (with error)
RX_SID_FIRST	4	Start of SID frame
RX_SID_UPDATE	5	Updates SID frame (without error)
RX_SID_BAD	6	Updates SID frame (with error)
RX_NO_DATA	7	Frame without data

2.2.7 amr_sid_sync_init function

The amr_sid_sync_init function initializes the constants, coefficient table, and buffers used for the SID (Silence Descriptor) synchronization function. Call this function only once when using the encoder.

[Classification]	SID synchronization initialization function	
[Function name]	amr_sid_sync_init	
[Summary of function]	Initializes the RAM area necessary for the SID synchronization function, calls the amr_sid_sync_reset function, and initializes parameters.	
[Format]	call amr_sid_sync_init	
[Argument]	None	
[Return value]	None	
[Registers used]	R0	
[Hardware resources]	Maximum stack level:	1
	Maximum loop stack level:	0
	Maximum number of repetitions:	0
	Maximum number of cycles:	15

Remark When calling this function, be sure to first secure static memory and scratch memory areas. For details, refer to **2.3 Memory Structure**.

2.2.8 amr_sid_sync_reset function

The amr_sid_sync_reset function resets the constants, coefficient table, and buffers used for the SID (Silence Descriptor) synchronization function to the initial status.

[Classification]	SID synchronization initialization function	
[Function name]	amr_sid_sync_reset	
[Summary of function]	Resets the parameters necessary for the SID synchronization function to the initial status.	
[Format]	call amr_sid_sync_reset	
[Argument]	None	
[Return value]	None	
[Registers used]	R0	
[Hardware resources]	Maximum stack level:	0
	Maximum loop stack level:	0
	Maximum number of repetitions:	0
	Maximum number of cycles:	8

Remark When calling this function, be sure to first secure static memory and scratch memory areas. For details, refer to **2.3 Memory Structure**.

2.2.9 amr_sid_sync function

The amr_sid_sync function determines the transmit frame type in accordance with the encode mode used for compression. Call this function once per encode processing of one frame.

[Classification] SID synchronization function

[Function name] amr_sid_sync

[Summary of function] Determines the frame type from the encode mode used for compression.

[Format] call amr_sid_sync

[Argument]

Type	Argument	Description
Register	R0L	Encode mode used for compression (see **Table 2-2 Operation Mode of Encoder/Decoder**.)

[Return value]

Type	Return Value	Description
Register	R0L	Transmit frame type (see Table 2-4)

[Registers used] R0, R1

[Hardware resources]
Maximum stack level: 0
Maximum loop stack level: 0
Maximum number of repetitions: 0
Maximum number of cycles: 30

Remark When calling this function, be sure to first secure static memory and scratch memory areas. For details, refer to **2.3 Memory Structure**.

★ **Table 2-4. Transmit Frame Type**

TX_TYPE	Value	Description
TX_SPEECH_GOOD	0	Speech frame
TX_SID_FIRST	1	Start of SID frame
TX_SID_UPDATE	2	SID frame update
TX_NO_DATA	3	Frame without data

2.2.10 amr_TX_to_RX function

The amr_TX_to_RX function converts a transmit frame type (TX_TYPE) to a receive frame type (RX_TYPE).

[Classification] Frame type conversion function

[Function name] amr_TX_to_RX

[Summary of function] Converts a transmit frame type to a receive frame type.

[Format] call amr_TX_to_RX

[Argument]

Type	Argument	Description
Register	R1L	Transmit frame type (TX_TYPE)

[Return value]

Type	Return Value	Description
Register	R1L	Receive frame type (RX_TYPE)

★ **[Registers used]** R1, R2, R3

[Hardware resources] Maximum stack level: 0

Maximum loop stack level: 0

Maximum number of repetitions: 0

★ Maximum number of cycles: 30

Remark Table 2-5 shows the receive frame type corresponding to each transmit frame type. The values in parentheses are the values of each transmit/receive frame type. If any other transmit frame type is specified, 0xFFFF is returned.

★ **Table 2-5. Transmit/Receive Frame Type**

Transmit Frame Type	Receive Frame Type
TX_SPEECH_GOOD (= 0)	RX_SPEECH_GOOD (= 0)
TX_SID_FIRST (= 1)	RX_SID_FIRST (= 4)
TX_SID_UPDATE (= 2)	RX_SID_UPDATE (= 5)
TX_NO_DATA (= 3)	RX_SID_NO_DATA (= 7)
TX_SPEECH_DEGRADE (= 4)	RX_SPEECH_DEGRADE (= 1)
TX_SPEECH_BAD (= 5)	RX_SPEECH_BAD (= 3)
TX_SID_BAD (= 6)	RX_SID_BAD (= 6)
TX_ONSET (= 7)	RX_ONSET (= 2)

2.2.11 amr_ehf_test function

The amr_ehf_test function checks whether the input speech signal is the homing frame of the encoder.

[Classification] Encoder homing frame test function

[Function name] amr_ehf_test

[Summary of function] Identifies the homing frame of the speech data.

[Format] call amr_ehf_test

[Argument]

Type	Argument	Description
Register	DP4	First address of speech buffer

[Return value]

Type	Return Value	Description
Register	R0L	Test result 0: No homing frame 1: Homing frame

[Registers used] R0, R1, DP4

[Hardware resources]
Maximum stack level: 0
Maximum loop stack level: 1
Maximum number of repetitions: 0
Maximum number of cycles: 967

2.2.12 amr_dhf_test function

The amr_dhf_test function checks whether the input bit stream is the homing frame of the decoder.

[Classification] Decoder homing frame test function

[Function name] amr_dhf_test

[Summary of function] Identifies the homing frame of the bit stream data.

[Format] call amr_dhf_test

[Argument]

Type	Argument	Description
Register	DP0	First address of bit stream buffer
Register	R6L	Decode mode (see **Table 2-2 Operation Mode of Encoder/ Decoder**, excluding MRDTX)

[Return value]

Type	Return Value	Description
Register	R0L	Test result 0: No homing frame 1: Homing frame

[Registers used] R0, R1, R2, R3, R4, R5, R6, R7, DP0, DP1, DP2, DP4, DP5, DP6, DP7

[Hardware resources]
Maximum stack level: 2
Maximum loop stack level: 1
Maximum number of repetitions: 0
Maximum number of cycles: 1858

2.2.13 amr_GetVersion function

The amr_GetVersion function returns the version information of the AMR speech CODEC library.

[Classification] Version information acquisition function

[Function name] amr_GetVersion

[Summary of function] Returns the version information.

[Format] call amr_GetVersion

[Argument] None

[Return value]

Type	Value	Description
Register	R0H	Major version number of library
Register	R0L	Minor version number of library
Register	R1H	Major version number of AMR speech CODEC
Register	R1L	Minor version number of AMR speech CODEC

[Function] Returns the version numbers of the library and AMR speech CODEC.

★ **Example:** When R0 = 0x00’0x0002’0x0000, library version: Ver 2.00.00

When R1 = 0x00’0x0007’0x0600, version of AMR speech CODEC: Ver 7.6.0

[Registers used] R0, R1

[Hardware resources]

Maximum stack level: 0

Maximum loop stack level: 0

Maximum number of repetitions: 0

Maximum number of cycles: 10

2.3 Memory Structure

This section explains the structure of the memory required for the μSAP77016-B06 library.

In the μSAP77016-B06, scratch memory and static memory areas must be defined separately. Be sure to use the PUBLIC quasi directive to define a symbol name. For the size of each memory, refer to Table 2-6.

(1) Scratch memory area

This area does not have to be saved. The user can use this area freely after encoding/decoding one frame.

Example:

```
public     lib_Scratch_x
public     lib_Scratch_y

SCRATCH_X           XRAMSEG
lib_Scratch_x:      DS       AMR_MAX_SCRATCH_X_SIZE

SCRATCH_Y           YRAMSEG
lib_Scratch_y:      DS       AMR_MAX_SCRATCH_Y_SIZE
```

(2) Static memory area

This area always includes saved data. If the user manipulates this area after initialization processing, normal operation of the μSAP77016-B06 library is not guaranteed.

Example:

```
public     amr_Static_enc_x
public     amr_Static_enc_y
public     amr_Static_dec_x
public     amr_Static_dec_y

AMR_STATIC_X        XRAMSEG
amr_Static_enc_x:   DS       AMR_MAX_STATIC_ENC_X_SIZE
amr_Static_dec_x:   DS       AMR_MAX_STATIC_DEC_X_SIZE

AMR_STATIC_Y        YRAMSEG
amr_Static_enc_y:   DS       AMR_MAX_STATIC_ENC_Y_SIZE
amr_Static_dec_y:   DS       AMR_MAX_STATIC_DEC_Y_SIZE
```

Table 2-6. Symbol Name and Size of Memory

Symbol Name	Size [Words]	X/Y Plane	Description
lib_Scratch_x	2025	X	Encoder/decoder common scratch area
lib_Scratch_y	909	Y	Encoder/decoder common scratch area
amr_Static_enc_x	509	X	Static area for encoder
amr_Static_enc_y	955	Y	Static area for encoder
amr_Static_dec_x	402	X	Static area for decoder
amr_Static_dec_y	525	Y	Static area for decoder

Remark The memory area can be allocated by a macro defined by amr_lib.h (Refer to **2.4 Macro**).

(3) I/O buffer

This is an area that is used to input/output speech data and bit stream data. The user can use the I/O buffer used by the encoder/decoder after encoding/decoding one frame. If the buffer is manipulated during encoding/decoding processing, normal operation of the μSAP77016-B06 is not guaranteed.

- **I/O buffer necessary for encoder**

Output buffer of bit stream data: 16 words in X memory space
Input buffer of speech data: 160 words in Y memory space

Example:

```
I_O_BUFFER_X        XRAMSEG
bitstream_out:      DS       16
I_O_BUFFER_Y        YRAMSEG
speech_in:          DS       160
```

- **I/O buffer necessary for decoder**

Input buffer of bit stream data: 16 words in X memory space
Output buffer of speech data: 160 words in Y memory space

Example:

```
I_O_BUFFER_X        XRAMSEG
bitstream_in:       DS       16
I_O_BUFFER_Y        YRAMSEG
speech_out:         DS       160
```

Remark Generally, the I/O buffer of speech data is configured as a double buffer when compression/decompression is executed in real time. In this case, a 320-word data memory is required in the Y memory space as an I/O buffer used by the encoder/decoder, and another 320-word data memory is required in either the X or Y memory space as an I/O buffer used for serial I/O.

[Notes on simulator]

The library of the μSAP77016-B06 reads a memory space other than an allocated area for faster pointer processing. Therefore, the simulator may issue a warning, depending on the status of memory allocation.

Table 2-7. Memory Access Range

Area Name	Description
amr_Static_enc_x	Memory space other than this area is not accessed.
amr_Static_enc_y	Memory space other than this area is not accessed.
amr_Static_dec_x	Memory space other than this area is not accessed.
amr_Static_dec_y	Memory space other than this area is not accessed.
lib_Scratch_x	Memory in a range from this area to +5 words is accessed.
lib_Scratch_y	Memory in a range from the beginning of this area to –1 word is accessed.

2.4 Macros

2.4.1 AMR_AllocateScratchMemory macro

[Classification] Scratch memory allocation

[Macro name] AMR_AllocateScratchMemory

[Summary of function] Allocates the scratch memory necessary for AMR speech CODEC processing and declares a symbol name.

[Format] %AMR_AllocateScratchMemory

[Argument] None

[Return value] None

[Definition symbol] ib_Scratch_x, lib_Scratch_y

2.4.2 AMR_AllocateStaticMemoryForEncoder macro

[Classification] Static memory allocation for encoder

[Macro name] AMR_AllocateStaticMemoryForEncoder

[Summary of function] Allocates the static memory necessary for encoding processing of AMR speech CODEC and declares a symbol name.

[Format] %AMR_AllocateStaticMemoryForEncoder

[Argument] None

[Return value] None

[Definition symbol] amr_Static_enc_x, amr_Static_enc_y

2.4.3 AMR_AllocateStaticMemoryForDecoder macro

[Classification] Static memory allocation for decoder

[Macro name] AMR_AllocateStaticMemoryForDecoder

[Summary of function] Allocates the static memory necessary for decoding processing of AMR speech CODEC and declares a symbol name.

[Format] %AMR_AllocateStaticMemoryForDecoder

[Argument] None

[Return value] None

[Definition symbol] amr_Static_dec_x, amr_Static_dec_y

Remark The macro used for the library of the μSAP77016-B06 is defined by amr_lib.h. To use the macro, include amr_lib.h.

CHAPTER 3 BIT STREAM FORMAT

Tables 3-1 through 3-9 show the bit allocation of the bit stream in each mode of the μSAP77016-B06.

Table 3-1. Bit Allocation in MR122 Mode

Bit Position	Description
b1 to b7	index of 1st LSF submatrix
b8 to b15	index of 2nd LSF submatrix
b16 to b23	index of 3rd LSF submatrix
b24	sign of 3rd LSF submatrix
b25 to b32	index of 4th LSF submatrix
b33 to b38	index of 5th LSF submatrix
First subframe	
b39 to b47	adaptive codebook index
b48 to b51	adaptive codebook gain
b52	sign information for 1st and 6th pulses
b53 to b55	position of 1st pulse
b56	sign information for 2nd and 7th pulses
b57 to b59	position of 2nd pulse
b60	sign information for 3rd and 8th pulses
b61 to b63	position of 3rd pulse
b64	sign information for 4th and 9th pulses
b65 to b67	position of 4th pulse
b68	sign information for 5th and 10th pulses
b69 to b71	position of 5th pulse
b72 to b74	position of 6th pulse
b75 to b77	position of 7th pulse
b78 to b80	position of 8th pulse
b81 to b83	position of 9th pulse
b84 to b86	position of 10th pulse
b87 to b91	fixed codebook gain
Second subframe	
b92 to b97	adaptive codebook index (relative)
b98 to b141	same description as b48 to b91
Third subframe	
b142 to b194	same description as b39 to b91
Fourth subframe	
b195 to b244	same description as b92 to b141

Table 3-2. Bit Allocation in MR102 Mode

Bit Position	Description
b1 to b8	index of 1st LSF subvector
b9 to b17	index of 2nd LSF subvector
b18 to b26	index of 3rd LSF subvector
First subframe	
b27 to b34	adaptive codebook index
b35	sign information for 1st and 5th pulses
b36	sign information for 2st and 6th pulses
b37	sign information for 3rd and 7th pulses
b38	sign information for 4th and 8th pulses
b39 to b48	position of 1st, 2nd, and 5th pulse
b49 to b58	position of 3rd, 6th, and 7th pulse
b59 to b65	position of 4th and 8th pulse
b66 to b72	codebook gains
Second subframe	
b73 to b77	adaptive codebook index (relative)
b78 to b115	same description as b35 to b72
Third subframe	
b116 to b161	same description as b27 to b72
Fourth subframe	
b162 to b204	same description as b73 to b115

Table 3-3. Bit Allocation in MR795 Mode

Bit Position	Description
b1 to b9	index of 1st LSF subvector
b10 to b18	index of 2nd LSF subvector
b19 to b27	index of 3rd LSF subvector
First subframe	
b28 to b35	adaptive codebook index
b36 to b39	position of 4th pulse
b40 to b42	position of 3rd pulse
b43 to b45	position of 2nd pulse
b46 to b48	position of 1st pulse
b49	sign information for 4th pulse
b50	sign information for 3rd pulse
b51	sign information for 2nd pulse
b52	sign information for 1st pulse
b53 to b56	adaptive codebook gain
b57 to b61	fixed codebook gain
Second subframe	
b62 to b67	adaptive codebook index (relative)
b68 to b93	same description as b36 to b61
Third subframe	
b94 to b127	same description as b28 to b61
Fourth subframe	
b128 to b159	same description as b62 to b93

Table 3-4. Bit Allocation in MR74 Mode

Bit Position	Description
b1 to b8	index of 1st LSF subvector
b9 to b17	index of 2nd LSF subvector
b18 to b26	index of 3rd LSF subvector
First subframe	
b27 to b34	adaptive codebook index
b35 to b38	position of 4th pulse
b39 to b41	position of 3rd pulse
b42 to b44	position of 2nd pulse
b45 to b47	position of 1st pulse
b48	sign information for 4th pulse
b49	sign information for 3rd pulse
b50	sign information for 2nd pulse
b51	sign information for 1st pulse
b52 to b58	codebook gains
Second subframe	
b59 to b63	adaptive codebook index (relative)
b64 to b87	same description as b35 to b58
Third subframe	
b88 to b119	same description as b27 to b58
Fourth subframe	
b120 to b148	same description as b59 to b87

Table 3-5. Bit Allocation in MR67 Mode

Bit Position	Description
b1 to b8	index of 1st LSF subvector
b9 to b17	index of 2nd LSF subvector
b18 to b26	index of 3rd LSF subvector
First subframe	
b27 to b34	adaptive codebook index
b35 to b38	position of 3rd pulse
b39 to b42	position of 2nd pulse
b43 to b45	position of 1st pulse
b46	sign information for 3rd pulse
b47	sign information for 2nd pulse
b48	sign information for 1st pulse
b49 to b55	codebook gains
Second subframe	
b56 to b59	adaptive codebook index (relative)
b60 to b80	same description as b35 to b55
Third subframe	
b81 to b109	same description as b27 to b55
Fourth subframe	
b110 to b134	same description as b56 to b80

Table 3-6. Bit Allocation in MR59 Mode

Bit Position	Description
b1 to b8	index of 1st LSF subvector
b9 to b17	index of 2nd LSF subvector
b18 to b26	index of 3rd LSF subvector
First subframe	
b27 to b34	adaptive codebook index
b35 to b39	position of 2nd pulse
b40 to b43	position of 1st pulse
b44	sign information for 2nd pulse
b45	sign information for 1st pulse
b46 to b51	codebook gains
Second subframe	
b52 to b55	adaptive codebook index(relative)
b56 to b72	same description as b35 to b51
Third subframe	
b73 to b97	same description as b27 to b51
Fourth subframe	
b98 to b118	same description as b52 to b72

Table 3-7. Bit Allocation in MR515 Mode

Bit Position	Description
b1 to b8	index of 1st LSF subvector
b9 to b16	index of 2nd LSF subvector
b17 to b23	index of 3rd LSF subvector
First subframe	
b24 to b31	adaptive codebook index
b32	position subset
b33 to b35	position of 2nd pulse
b36 to b38	position of 1st pulse
b39	sign information for 2nd pulse
b40	sign information for 1st pulse
b41 to b46	codebook gains
Second subframe	
b47 to b50	adaptive codebook index (relative)
b51 to b65	same description as b32 to b46
Third subframe	
b66 to b84	same description as b47 to b65
Fourth subframe	
b85 to b103	same description as b47 to b65

Table 3-8. Bit Allocation in MR475 Mode

Bit Position	Description
b1 to b8	index of 1st LSF subvector
b9 to b16	index of 2nd LSF subvector
b17 to b23	index of 3rd LSF subvector
First subframe	
b24 to b31	adaptive codebook index
b32	position subset
b33 to b35	position of 2nd pulse
b36 to b38	position of 1st pulse
b39	sign information for 2nd pulse
b40	sign information for 1st pulse
b41 to b48	codebook gains
Second subframe	
b49 to b52	adaptive codebook index(relative)
b53 to b61	same description as b32 to b40
Third subframe	
b62 to b65	same description as b49 to b52
b66 to b82	same description as b32 to b48
Fourth subframe	
b83 to b95	same description as b49 to b61

Table 3-9. Bit Allocation in MRDTX Mode

Bit Position	Description
b1 to b3	index of reference vector
b4 to b11	index of 1st LSF subvector
b12 to b20	index of 2nd LSF subvector
b21 to b29	index of 3rd LSF subvector
b30 to b35	index of logarithmic frame energy

The relationship between the bit allocation (b1 through b242) in Tables 3-1 through 3-9 and word data is shown below.

Figure 3-1. Relationship Between Bit Allocation and Word

Offset	MSB															LSB
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+0	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10	b11	b12	b13	b14	b15	b16
+1	b17						•	•	•	•						b32
+2	b33						•	•	•	•						b48
+3	b49						•	•	•	•						b64
+4	b65						•	•	•	•						b80
+5	b81						•	•	•	•						b96
+6	b97						•	•	•	•						b112
+7	b113						•	•	•	•						b128
+8	b129						•	•	•	•						b144
+9	b145						•	•	•	•						b160
+10	b161						•	•	•	•						b176
+11	b177						•	•	•	•						b192
+12	b193						•	•	•	•						b208
+13	b209						•	•	•	•						b224
+14	b225						•	•	•	•						b240
+15	b241						•	•	•	•						b256

CHAPTER 4 INSTALLATION

4.1 Installation Procedure

The μSAP77016-B06 is supplied in a 3.5” floppy disk (1.44 MB). How to install the μSAP77016-B06 in a host machine is described below. Install the μSAP77016-B06 in a host machine in which an OS such as Windows 95, 98, 2000, or Windows NT™ 4.0 or later has been installed.

<1> Set the floppy disk in the floppy disk drive.

<2> Execute amr_mw.exe from the floppy disk (the files of the μSAP77016-B06 are compressed in a self-extracting format). The following is an example of when files are copied from the A drive to the C drive.

```
A:\>amr_mw.exe<CR>
```

<3> A dialog box to specify the directory for installing the μSAP77016-B06 is displayed.
Specify a directory. Any directory may be specified. In this example, it is assumed that C:\DSPTools is specified.

<4> Click the OK button and expansion of the files begins.

<5> Confirm that the files have been expanded. For each directory, refer to **1.3.4 Directory configuration**.

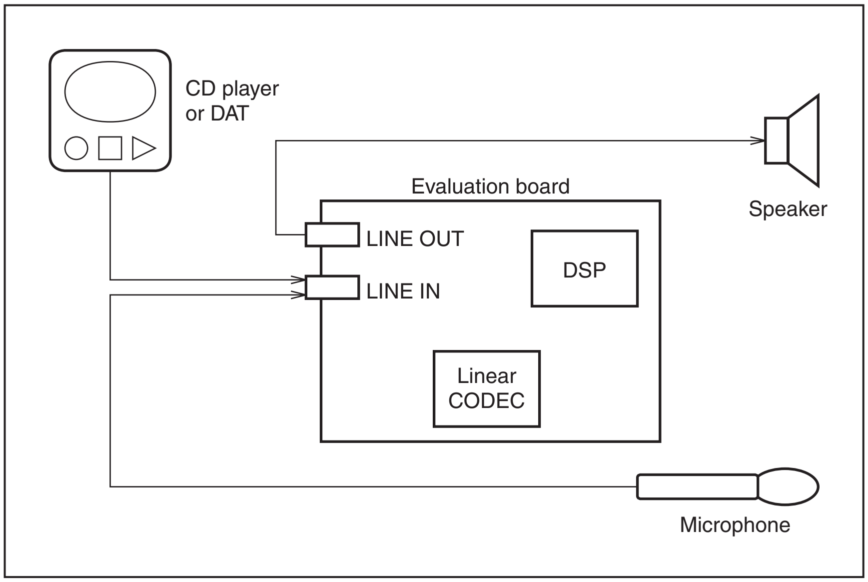
```
A:\>dir C:\DSPTools<CR>
```

4.2 Sample Creation Procedure

The sample programs for a single channel and multiple channels (2 channels) are stored in the smp directory (refer to **1.3.4 Directory configuration**). For the source program for a single channel, refer to **APPENDIX A SAMPLE PROGRAM SOURCE**.

An AMR speech CODEC system can be evaluated using a sample program by actually connecting a CD player, DAT, microphone, or speaker to the μSAP77016-B06. Note that additional code such as for a system-dependent block may be required.

Figure 4-1. Example of Sample Program Evaluation System



An example of how to build a sample program (for a single channel) of the μSAP77016-B06 is shown below.

(1) Start up the WB77016 (workbench).

(2) Open the sample.prj project file.

Example: Specify sample.prj with the Open Project command from 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) Download sample.lnk to the target system using the ID77016 (debugger) and execute it.

4.3 Change of Location

The section names shown in Table 4-1 are given to the library of the μSAP77016-B06. The location can be changed in accordance with the user target system.

Table 4-1. Section Name

Section Name	Type	Description
__AMR_CONST_X__	XROMSEG/XRAMSEG	Constant data storage area
__AMR_CONST_Y__	YROMSEG/YRAMSEG	Constant data storage area
__AMR_CODEC__	IROMSEG/IRAMSEG	AMR speech CODEC program

4.4 Symbol Naming Regulations

The symbols used for the library of the μSAP77016-B06 are named in accordance with the following regulation. Make sure not to duplicate these symbol names when using the μSAP77016-B06 in combination with another application.

Table 4-2. Symbol Naming Regulations

Classification	Naming Regulation
Function name	amr_XXXX
Constant symbol name	amr_cnst_XXXX
Static RAM area symbol name	amr_Static_[enc|dec]_[x|y]
Scratch RAM area symbol name	lib_Scratch_[x|y]
Code segment name (IMSEG)	__AMR_CODEC__
Constant segment name (ROMSEG/RAMSEG)	__AMR_CONST_[X|Y]__
Macro, constant name (such as #define and equ declarations)	AMR_XXXX
File name	amrXXXX.*

Remark "XXXX" indicates arbitrary alphanumeric characters.

APPENDIX A SAMPLE PROGRAM SOURCE

This appendix shows a sample source of the μSAP77016-B06 (AMR speech CODEC middleware). This sample source was created by referring to the sample source of 3GPP (3rd Generation Partnership Project). For details of each processing, refer to the sample source of 3GPP.

• sample.asm (1/7)

```
/*-------------------------------------------------------------------*/
/*      File Information                                             */
/*-------------------------------------------------------------------*/
/*      Name    : sample.asm                                         */
/*      Type    : Assembler program module                           */
/*      Version : 2.00                                               */
/*      Date    : 2002 APR 24                                        */
/*      CPU     : uPD7701x Family                                    */
/*      Assembler: WB77016                                           */
/*      About   : GSM AMR speech codec   Version 7.6.0               */
/*                                                                   */
/*-------------------------------------------------------------------*/
/*      Copyright (C) NEC Corporation 1999, 2000, 2001, 2002         */
/*      NEC CONFIDENTIAL AND PROPRIETARY                             */
/*      All rights reserved by NEC Corporation.                      */
/*      Use of copyright notice does not evidence publication        */
/*-------------------------------------------------------------------*/

/* ========================================================================
 *  Include files
 * ===================================================================== */
#include "upd7701x.mac"
#include "amr_lib.h"

/* ========================================================================
 *  Memory allocate for AMR CODEC
 * ===================================================================== */
    %AMR_AllocateScratchMemory
    %AMR_AllocateStaticMemoryForEncoder
    %AMR_AllocateStaticMemoryForDecoder

/* ========================================================================
 *  Local variables and tables
 * ===================================================================== */
#define DTX       DTX_OFF       /* set DTX mode.    DTX_OFF, DTX_ON_VAD1, DTX_ON_VAD2 */
#define MULTI     0             /* set MULTI mode.  0: normal mode. 1:multi mode      */

#define L_FRAME   160

WORK_X  XRAMSEG at 0x0000
    f_run:        ds  1           ; codec start flag, "1" is start codec.
    f_reset:      ds  1           ; test sequence reset flag, "-1" is reset.
    f_multi:      dw  MULTI       ; multi mode flag, not "0" is multi rate mode.
    frame_type:   ds  1           ; RX/TX frame type.
    used_mode:    ds  1           ; used AMR Encoder mode.
    bitstream:    ds  16          ; bitstream load/store buffer.
    e_mode:       dw  MR475       ; AMR Encoder mode.
```

• sample.asm (2/7)

```
    d_mode:       ds  1            ; AMR Decoder mode.
    prev_d_mode:  ds  1            ; AMR previous codec mode
    dtx_mode:     dw  DTX          ; DTX mode.
    f_e_reset:    ds  1            ; encoder reset flag.
    f_d_reset:    ds  1            ; decoder reset flag.
    f_d_reset_old:ds  1            ; decoder old reset flag.
    r7save:       ds  3            ; r7 register load/sotre buffer for interrupt handler.
    dp7save:      ds  1            ; dp7 register load/sotre buffer for interrupt handler.
    mode_cnt:     ds  1            ; coder mode count for multi rate mode.
    frame_cnt:    ds  1            ; frame count.

WORK_Y  YRAMSEG at 0x0000
    si1_ptr:      ds  1            ;
    si1_buff:     ds  L_FRAME      ;
    so1_buff:     ds  L_FRAME      ;
    speech_in:    ds  L_FRAME      ;
    speech_out:   ds  L_FRAME      ;

#define F_RUN               *f_run:x
#define F_RESET             *f_reset:x
#define F_MULTI             *f_multi:x
#define TX_TYPE             *frame_type:x
#define RX_TYPE             *frame_type:x
#define USED_MODE           *used_mode:x

#define E_MODE              *e_mode:x
#define D_MODE              *d_mode:x
#define DTX_MODE            *dtx_mode:x
#define PREV_D_MODE         *prev_d_mode:x
#define E_RESET_FLAG        *f_e_reset:x
#define D_RESET_FLAG        *f_d_reset:x
#define D_RESET_FLAG_OLD    *f_d_reset_old:x

#define MODE_COUNT          *mode_cnt:x
#define FRAME_COUNT         *frame_cnt:x

/* ========================================================================
 *  Vector registration
 * ========================================================================*/
%BeginVector(StartUp)              ;Regist start up routine
    %NotUseVector(VectorINT1)      ;
    %NotUseVector(VectorINT2)      ;
    %NotUseVector(VectorINT3)      ;
    %NotUseVector(VectorINT4)      ;
    %RegistVector(VectorSI1, SI1Handler)    ;Regist SI1 handler
    %NotUseVector(VectorSO1)       ;
    %NotUseVector(VectorSI2)       ;
    %NotUseVector(VectorSO2)       ;
    %NotUseVector(VectorHI)        ;
    %NotUseVector(VectorHO)        ;
%EndVector
```

• sample.asm (3/7)

```
/* ====================================================================
 *  Programe code section
 * ====================================================================*/
MAIN_CODE   IMSEG at 0x4000
StartUp:
    ;;============================================================;;
    ;;  Initialize uPD7701x                                       ;;
    ;;============================================================;;
    %InitializeSystemRegister       ;Initialize system register
    %ClearAllRegister              ;Clear all uPD7701x register

    ;;============================================================;;
    ;;  Initialize Register & Peripheral Units                    ;;
    ;;============================================================;;
    r0l = 0x0200                    ;
    *SST1:x = r0l                   ;

    ;;============================================================;;
    ;;  Initialize AMR CODEC module                               ;;
    ;;============================================================;;
    /* Initialize AMR encoder */
    clr(r0)                         ;
    r0l = DTX_MODE                  ;set DTX mode
    call amr_InitEncoder            ;
    call amr_sid_sync_init          ;
    /* Initialize AMR decoder */
    call amr_InitDecoder            ;

    ;;============================================================;;
    ;;  Initialize buffer                                         ;;
    ;;============================================================;;
    /* clear serial input/output buffer */
    r0l = si1_buff                  ;Initialize serial i/o buffer pointer.
    *si1_ptr:y = r0l                ;
    dp4 = si1_buff                  ;Initialize serial i/o buffer.
    rep L_FRAME*2                   ;
        *dp4++ = r0h                ;
    /* clear speech input/output buffer */
    dp4 = speech_in                 ;
    dp5 = speech_out                ;
    loop L_FRAME {                  ;
        *dp4++ = r0h                ;
        *dp5++ = r0h                ;
    }                               ;
    /* clear flags */
    clr(r0)                         ;
    F_RUN = r0l                     ;clear codec start flag.
    F_RESET = r0l                   ;clear codec rest  flag.

    TX_TYPE = r0l                   ;clear TX type.
    RX_TYPE = r0l                   ;clear RX type.
    USED_MODE = r0l                 ;clear used AMR encoder mode.

    D_MODE = r0l                    ;clear AMR decoder mode.
    PREV_D_MODE = r0l               ;clear prev AMR decoder mode.

    E_RESET_FLAG = r0l              ;clear encoder reset flag.
    D_RESET_FLAG = r0l              ;clear decoder reset flag.
```

• sample.asm (4/7)

```
    MODE_COUNT = r0l                ;clear mode count.
    FRAME_COUNT = r0l               ;clear frame count.

    r0l = 1                         ;
    D_RESET_FLAG_OLD = r0l          ;set decoder old reset flag.

    ;;==============================================================;;
    ;;  Set interrupt mask                                           ;;
    ;;==============================================================;;
    %DisableInterrupt               ;
    %SetMask(SR_ALL)                ;
    %UnSetMask(SR_SI1)              ;
    %EnableInterrupt                ;

    ;;==============================================================;;
    ;;  Main routine                                                 ;;
    ;;==============================================================;;
MainLoop:
    /* check reset codec flag */
    r0 = F_RESET                    ;
    if(r0 < 0) jmp StartUp          ;
    /* check start codec flag */
    r0 = F_RUN                      ;
    if(r0 == 0) jmp MainLoop        ;
    clr(r0)                         ;
    F_RUN = r0l                     ;clear codec start flag.
    /* increment frame count */
    r0l = FRAME_COUNT               ;
    r0 = r0 + 1                     ;
    FRAME_COUNT = r0l               ;
    /* Copy input/output PCM data */
    dp4 = speech_in                 ;
    dp5 = speech_out                ;
    dp6 = si1_buff                  ;
    loop L_FRAME {                  ;
        r0l = *dp6++                ;read input PCM data from si1_buff.
        r0 = r0 & 0xfff8            ;
        *dp4++ = r0l                ;store PCM data to speech_in.
    }                               ;
    loop L_FRAME {                  ;
        r0l = *dp5++                ;read output PCM data from speech_out.
        r0 = r0 & 0xfff8            ;
        *dp6++ = r0l                ;store PCM data to so1_buff.
    }                               ;
    /* Encode process */
    call Proc_AMR_Encoder           ;
    /* Decode process */
    call Proc_AMR_Decoder           ;
    jmp MainLoop                    ;

/* ========================================================================
[Function Name] Proc_AMR_Encoder
 ========================================================================*/
Proc_AMR_Encoder:
    /* check for homing frame */
    dp4 = speech_in                 ;set speech buffer address.
    call amr_ehf_test               ;test homing frame.
    E_RESET_FLAG = r0l              ;save reset flag.
```

• sample.asm (5/7)

```
    /* multi rate mode */
    r0 = F_MULTI                    ;
    if(r0 != 0) call MultiRateMode  ;

    /* encode speech */
    clr(r0)                         ;
    r0l = E_MODE                    ;set AMR encoder mode.
    dp4 = speech_in                 ;set speech buffer address.
    dp0 = bitstream                 ;set bitstream buffer address.
    call amr_EncodeFrame            ;encode speech data.
    USED_MODE = r0l                 ;save used AMR encoder mode.

    /*include frame type and mode information in serial bitstream */
    clr(r0)                         ;
    r0l = USED_MODE                 ;set used AMR encoder mode.
    call amr_sid_sync               ;
    TX_TYPE = r0l                   ;save TX frame type.
    r0 = r0 - TX_NO_DATA            ;
    if( r0 != 0 ) jmp proc_enc_noupdate_mode;
        clr(r0)                     ;
        r0 = r0 - 1                 ;
        D_MODE = r0l                ;
        jmp proc_enc_next           ;
proc_enc_noupdate_mode:
        clr(r0)                     ;
        r0l = E_MODE                ;
        D_MODE = r0l                ;

proc_enc_next:
    /* perform homing if homing frame was detected at encoder input */
    r0 = E_RESET_FLAG               ;
    if(r0 == 0) ret                 ;
        call amr_ResetEncoder       ;
        call amr_sid_sync_reset     ;
        ret                         ;

/* ============================================================================
[Function Name] Proc_AMR_Decoder
 ============================================================================*/
Proc_AMR_Decoder:
    /* Convert TX frame type to RX frame type*/
    clr(r1)                         ;
    r1l = TX_TYPE                   ;set TX frame type.
    call amr_TX_to_RX               ;
    RX_TYPE = r1l                   ;save RX frame type.

    clr(r1)                         ;
    r1l = RX_TYPE                   ;
    r0 = r1 - RX_NO_DATA            ;
    if(r0 != 0) jmp proc_dec_set_prev_mode;
        r0l = PREV_D_MODE           ;
        D_MODE = r0l                ;
        jmp proc_dec_set_prev_mode_end;
proc_dec_set_prev_mode:
        r0l = D_MODE                ;
        PREV_D_MODE = r0l           ;
proc_dec_set_prev_mode_end:
```

• sample.asm (6/7)

```
    /* if homed: check if this frame is another homing frame */
    r0 = D_RESET_FLAG_OLD           ;
    if(r0 == 0) jmp proc_dec_next_1 ;
        /* only check until end of first subframe */
        clr(r6)                     ;
        R6l = D_MODE                ;set AMR decoder mode.
        dp0 = bitstream             ;set bitstream buffer address.
        call amr_dhf_test           ;test homing frame.
        D_RESET_FLAG = r0l          ;save reset flag.

proc_dec_next_1:
    /* produce encoder homing frame if homed & input=decoder homing frame */
    r0 = D_RESET_FLAG               ;
    if(r0 == 0) jmp proc_dec_start  ;
    r0 = D_RESET_FLAG_OLD           ;
    if(r0 == 0) jmp proc_dec_start  ;
        r0l = EHF_MASK              ;
        dp4 = speech_out            ;
        rep L_FRAME                 ;
            *dp4++ = r0l            ;
        jmp proc_dec_next_2         ;

proc_dec_start:
    /* decode frame */
    clr(r0)                         ;
    clr(r1)                         ;
    r0l = D_MODE                    ;set AMR decoder mode.
    r1l = RX_TYPE                   ;set RX_TYPE = RX_SPEECH.
    dp0 = bitstream                 ;set bitstream buffer address.
    dp4 = speech_out                ;set speech buffer address.
    call amr_DecodeFrame            ;decoder bitstream.

    /* if not homed: check whether current frame is a homing frame */
proc_dec_next_2:
    r0 = D_RESET_FLAG_OLD           ;
    if(r0 != 0) jmp proc_dec_next_3 ;
        /* check whole frame */
        clr(r6)                     ;
        R6l = D_MODE                ;set AMR decoder mode.
        dp0 = bitstream             ;set bitstream buffer address.
        call amr_dhf_test           ;test homing frame.
        D_RESET_FLAG = r0l          ;save reset flag.

    /* reset decoder if current frame is a homing frame */
proc_dec_next_3:
    r0 = D_RESET_FLAG               ;
    if(r0 != 0) call amr_ResetDecoder;
    r0 = D_RESET_FLAG               ;
    D_RESET_FLAG_OLD = r0h          ;
    ret                             ;
```

• sample.asm (7/7)

```
/* ==========================================================================
[Function Name] MultiRateMode
 ==========================================================================*/
MultiRateMode:
    clr(r0)                      ;
    r0l = MODE_COUNT             ;
    r1 = r0 & 0x7                ;
    E_MODE = r1l                 ;
    r1 = r1 + 1                  ;
    MODE_COUNT = r1l             ;
    ret                          ;

/* ==========================================================================
[Handler Name]  SI1Handler
[RAM]                    si1_buff, so1_buff, si1_ptr, f_run
[Use Register]  r7, dp7
[MIPS]          0.200[MIPS] (25*159+27*1[cycle])
[Use Stacks]    loop stack: 0, call stack: 0, repeat: 0
 ==========================================================================*/
SI1Handler:
    /* Save r7, dp7 register */
    *r7save+0:x = r7l            ;save r7l
    *r7save+1:x = r7h            ;save r7h
    *r7save+2:x = r7e            ;save r7e
    r7l = dp7                    ;save dp7
    *dp7save:x = r7l             ;
    /* input/output PCM data */
    r7l = *si1_ptr:y             ;
    dp7 = r7l                    ;
    r7h = *SDT1:x                ;
    *dp7##160 = r7h              ;
    r7h = *dp7##-159             ;
    *SDT1:x = r7h                ;
    /* check frame count */
    clr(r7)                      ;
    r7l = dp7                    ;
    *si1_ptr:y = r7l             ;
    r7 = r7 - ( si1_buff + L_FRAME );
    if(r7 != 0) jmp si1_end      ;
    r7l = 1                      ;
    F_RUN = r7l                  ;
    r7l = si1_buff               ;
    *si1_ptr:y = r7l             ;
si1_end:
    /* Restore r7, dp7 register */
    r7l = *dp7save:x             ;
    dp7 = r7l                    ;load dp7
    r7e = *r7save+2:x            ;load r7e
    r7h = *r7save+1:x            ;load r7h
    r7l = *r7save+0:x            ;load r7l
    reti                         ;

/* End of file */
END
```

APPENDIX B RELATED DOCUMENTS

The recommendations related to the AMR speech CODEC published by 3GPP are as follows.

- 3GPP TS 26.071 AMR speech Codec; General description
- 3GPP TS 26.073 AMR speech Codec; C-source code
- 3GPP TS 26.074 AMR speech Codec; Test sequences
- 3GPP TS 26.090 AMR speech Codec; Transcoding Functions
- 3GPP TS 26.091 AMR speech Codec; Error concealment of lost frames
- 3GPP TS 26.092 AMR speech Codec; comfort noise
- 3GPP TS 26.093 AMR speech Codec; Source Controlled Rate operation
- 3GPP TS 26.094 AMR speech Codec; Voice Activity Detector
- 3GPP TS 26.101 AMR speech Codec; Frame Structure
- 3GPP TS 26.102 AMR speech Codec; Interface to Iu and Uu

[MEMO]

NEC

Facsimile Message

From:

Name

Company

Tel. FAX

Address

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

Thank you for your kind support.

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

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

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

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

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

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

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

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

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

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

Document title: ______________________________

Document number: ______________________ Page number: ______________

If possible, please fax the referenced page or drawing.

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

CS 02.3