NEC

User's Manual

μSAP703000-B04, μSAP70732-B04

ADPCM middleware

Target device

μSAP703000-B04: V850 family™
μSAP70732-B04 : V810 family™

Document No. U11381EJ2V0UM00 (2nd edition)
Date Published July 1998 N CP(K)
Printed in Japan

[MEMO]

M7A 96.10

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 Electronics (Germany) GmbH
Duesseldorf, Germany
Tel: 0211-65 03 02
Fax: 0211-65 03 490

NEC Electronics (UK) Ltd.
Milton Keynes, UK
Tel: 01908-691-133
Fax: 01908-670-290

NEC Electronics Italiana s.r.1.
Milano, Italy
Tel: 02-66 75 41
Fax: 02-66 75 42 99

NEC Electronics (Germany) GmbH
Benelux Office
Eindhoven, The Netherlands
Tel: 040-2445845
Fax: 040-2444580

NEC Electronics (France) S.A.
Velizy-Villacoublay, France
Tel: 01-30-67 58 00
Fax: 01-30-67 58 99

NEC Electronics (France) S.A.
Spain Office
Madrid, Spain
Tel: 01-504-2787
Fax: 01-504-2860

NEC Electronics (Germany) GmbH
Scandinavia Office
Taeby, Sweden
Tel: 08-63 80 820
Fax: 08-63 80 388

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 Singapore Pte. Ltd.
United Square, Singapore 1130
Tel: 65-253-8311
Fax: 65-250-3583

NEC Electronics Taiwan Ltd.
Taipei, Taiwan
Tel: 02-719-2377
Fax: 02-719-5951

NEC do Brasil S.A.
Cumbica-Guarulhos-SP, Brasil
Tel: 011-6465-6810
Fax: 011-6465-6829

J98. 2

MAJOR REVISIONS IN THIS EDITION

Page	Contents
p.20	Change of **1.3.2 (3) Supported tools**
p.21	Change of **1.3.4 Directory configuration**
p.43	Change of **3.1.1 V810 family**
p.43	Change of **3.1.2 V850 family**
p.44	Change of **3.1.3 Directory and file**
p.45	Change of **3.2.1 UNIX version**
p.46	Change of **3.2.2 MS-DOS/PC DOS version**
p.47	Change of **3.3 Creating Sample Program**
p.49	Change of **APPENDIX SOURCE PROGRAM OF sample.c**

The mark ★ shows major revised points.

[MEMO]

PREFACE

Readers

This manual is intended for user engineers who wish to design and develop application systems using the V810 family/V850 family .

Purpose

The purpose of this manual is to help user engineers understand the middleware that supports the design and development V810 family/V850 family application systems.

Organization

This manual contains the following items:

- General
- Library specification
- Installation
- Appendix

How to read this manual

It is assumed that the readers of this manual have a general knowledge of electrical engineering, logic circuits, microcomputers, and the C language.

To understand the hardware functions of the V810 family/V850 family
→ Refer to the user's manuals (hardware) of each product.

To understand the instruction functions of the V810 family/V850 family
→ Refer to the user's manuals (architecture) of each product.

Legend

Data significance : Most-significant digit on left, least-significant digit on right
Active low : $\overline{xxx}$ (bar over pin and signal names)
Memory map address: Top-low, bottom-high
Note : Description of **Note** in the text
Caution : Important information
Remark : Supplement
Numeric notation : Binary ... ×××× or ××××B
Decimal ... ××××
Hexadecimal ... ××××H or 0× ××××
Prefix indicating power of 2 (address space, memory capacity)
K (kilo) = 2^{10} = 1024
M (mega) = 2^{20} = 1024^2
G (giga) = 2^{30} = 1024^3

Related documents

Some of the related documents listed below are preliminary versions but not so specified here.

Documents related to V810 family

Product Name		Data Sheet	User's Manual	
Nickname	Device Name		Hardware	Architecture
V810™	μPD70732	U10691E	U10661E	U10082E
V821™	μPD70741	U11678E	U10077E	

Documents related to V850 family

Product Name		Data Sheet	User's Manual	
Nickname	Device Name		Hardware	Architecture
V852™	μPD703002	U11826E	U10038E	U10243E
	μPD70P3002	U11827E		
V853™	μPD703003, 703003A, 703005A	U12261E	U10913E	
	μPD70F3003	U12036E		
	μPD70F3003A, 70F3025A	U13189E		
V854™	μPD703008	Planned	U11969E	
	μPD703008Y	Planned		
	μPD70F3008	U12756E		
	μPD70F3008Y	U12755E		
V850/SA1™	μPD703015	Planned	U12768E	
	μPD703015Y	Planned		
	μPD70F3017	Planned		
	μPD70F3017Y	Planned		

Document related to V810 family development tools (user's manual)

Document Name		Document Number
CA732 (C compiler)	Operation (UNIX™ based)	U11013E
	Operation (Windows™ based)	U11068E
	Assembly language	U11016E
	C language	U11010E
RX732 (Real-time OS)	Fundamental	U10346E
	Nucleus installation	U10347E
	Technical	U10490E

Documents related to V850 family development tools (user's manual)

Document Name		Document Number
IE-703002-MC (In-circuit emulator for V851, V852, V853, V854, V850/SA1)		U11595E
IE-703003-MC-EM1 (In-circuit emulator option board for V853)		U11596E
IE-703008-MC-EM1 (In-circuit emulator option board for V854)		U12420E
IE-703017-MC-EM1 (In-circuit emulator option board for V850/SA1)		U12898E
CA850 (C compiler)	Operation (UNIX based)	U12839E
	Operation (Windows based)	U12827E
	C language	U12840E
	Assembly language	U10543E
	Project manager (Windows based)	U11991E
ID850 (C source debugger)	Operation (Windows based)	U11196E
	Installation (UNIX based)	U12210E
RX850 (real-time OS)	Fundamental	U12861E
	Technical	U13002E
	Nucleus installation	U11038E
	Installation (UNIX based)	U12863E
	Installation (Windows based)	U12862E
	Debugger (Windows based)	U11158E
AZ850 (system performance analyzer) – operation		U11181E

For inquiries about the tools produced by Green Hills Software™, Inc. (GHS), please contact:

Green Hills Software, Inc.
510 Castillo Street, Santa Barbara,
California 93101
Tel: 805-965-6044
Web site: http://www.ghs.com

[MEMO]

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 GENERAL

This chapter explains the middleware and ADPCM decoder.

1.1 Middleware

Middleware is a software group tuned to draw out the full performance of a processor.

Because many high-performance RISC processors are available today, the processing conventionally implemented by dedicated hardware can now be realized by a high-performance RISC processor and software. The software used for this purpose is called middleware.

NEC supplies human-machine interface and signal processing technologies in the form of middleware. It provides excellent system solutions to satisfy the various needs of users.

This ADPCM middleware is a library that compresses and expands speech code (telephone quality).

Remark RISC: Reduced Instruction Set Computer

1.2 ADPCM

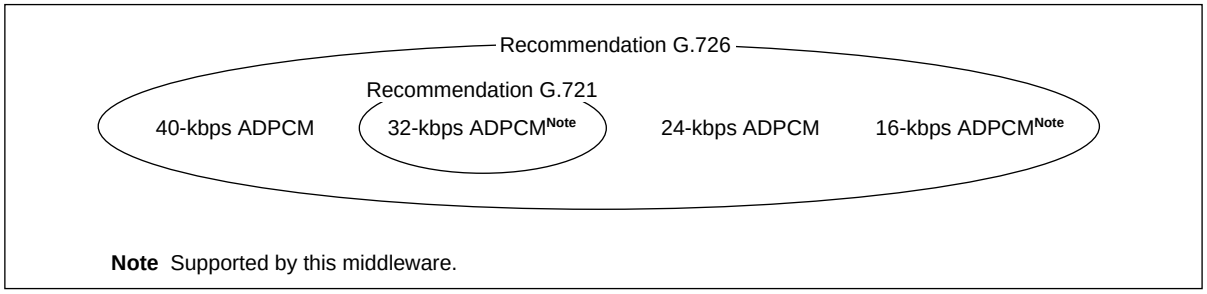
The ADPCM of this middleware codes speech in the analog telephone band (0.3 to 3.4 kHz) stipulated by ITU-T Recommendation G.726.

Recommendation G.726 was issued in 1990 by adding 40-kbps ADPCM, 24-kbps ADPCM, and 16-kbps ADPCM to Recommendation G.721 (32-kbps ADPCM) which was adopted as an international standard of telephone quality coding in 1988.

This middleware supports the 32-kbps ADPCM and 16-kbps ADPCM of Recommendation G.726.

Remark ADPCM: Adaptive Differential Pulse Code Modulation
ITU-T: International Telecommunication Union-Telecommunication standardization sector

Figure 1-1. Recommendation G.726

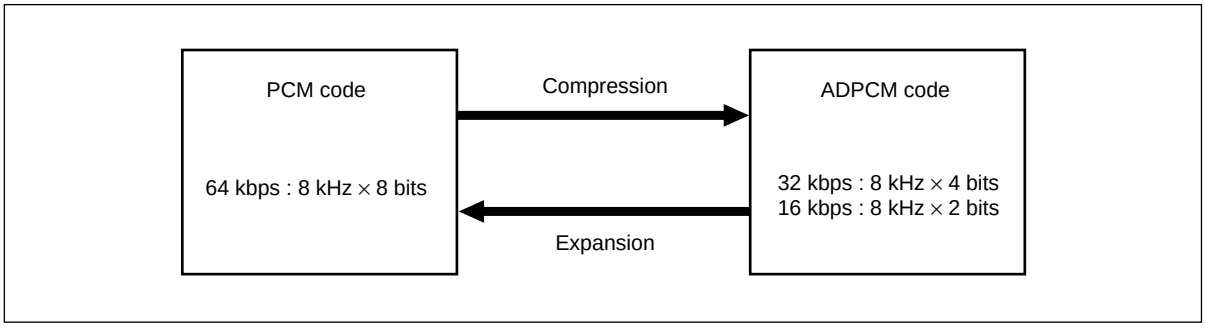


The interface of ADPCM is based on the 64-kbps PCM code stipulated by ITU-T Recommendation G.711. This is because the 64-kbps PCM of Recommendation G.721 has been widely used as an international standard telephone quality coding method, and because ADPCM was developed as a standard to improve the compression rate.

The 64-kbps PCM of ITU-T Recommendation G.711 limits the bandwidth to 0.3 to 3.4 kHz by using a bandpass filter and codes analog speech signals sampled at 8 kHz into 8-bit digital signals by means of non-linear quantization. The 64-kbps PCM has two modes: μ-law mode (used in Japan and North America) and A-law mode (used in Europe).

ADPCM is a standard for further compressing the speech codes compressed by means of 64-kbps PCM.

Figure 1-2. Concept of ADPCM

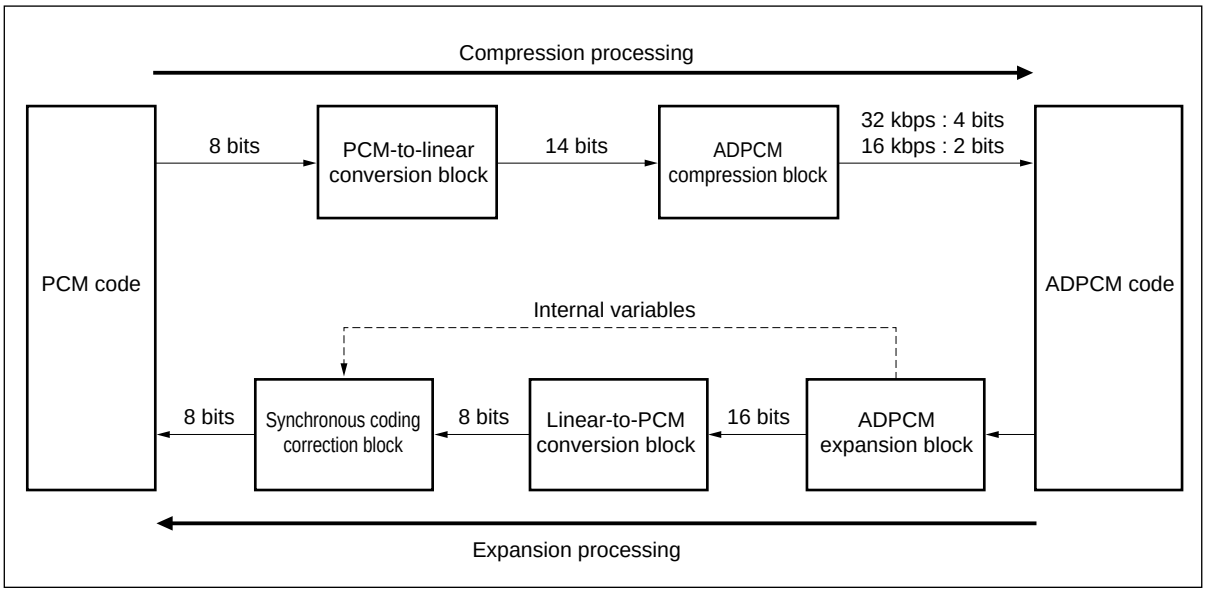


The compression/expansion processing of ADPCM is described next.

The compression processing of ADPCM is performed by a PCM-to-linear conversion block and an ADPCM compression block, as shown in Figure 1-3.

The expansion processing of ADPCM is performed by an ADPCM expansion block, linear-to-PCM conversion block, and synchronous coding correction block, as shown in Figure 1-3. The expansion processing prevents degradation of the sound quality by repeatedly compressing and expanding speech data, by using the synchronous coding correction block, when incorporated to an existing digital (64-kbps PCM) communication network.

Figure 1-3. Organization of ADPCM Processing



1.2.1 PCM-to-linear conversion block

This block converts a 64-kbps PCM code (8 bits) into a linear code (14 bits: expressed as 2's complement).

The PCM code in the μ-law mode is converted by a PCM decoder conforming to Recommendation G.711.

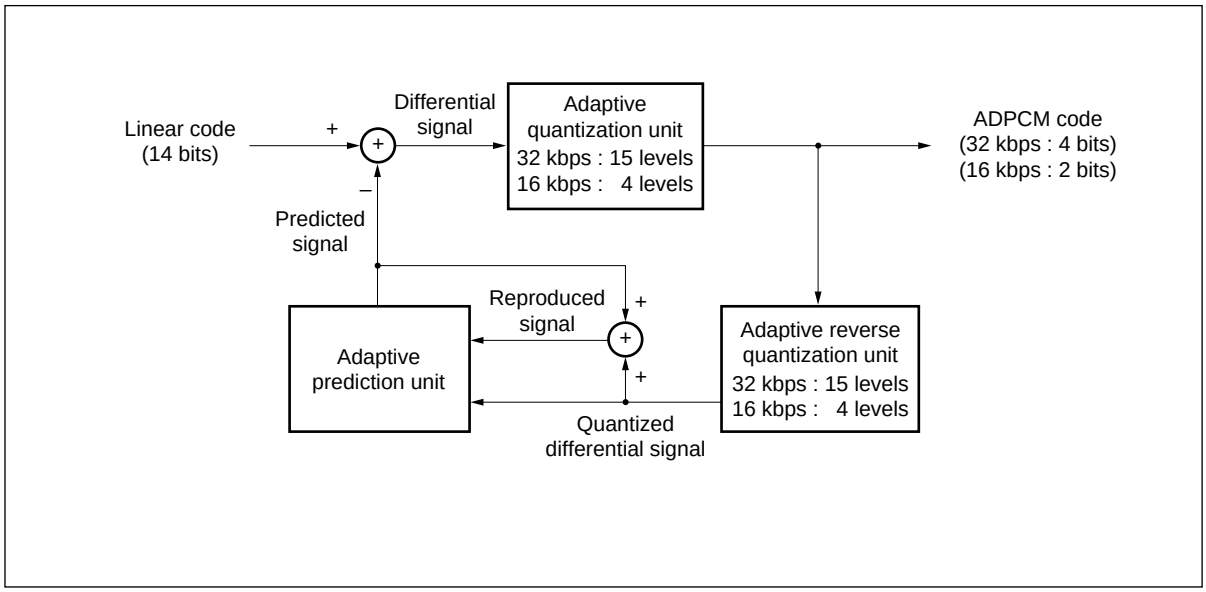
The PCM code in the A-law mode is converted by a PCM decoder conforming to Recommendation G.711. The resulting value is doubled and then adjusted to 14 bits.

1.2.2 ADPCM compression block

This block compresses a linear code (14 bits) to an ADPCM code (32 kbps: 4 bits, 16 kbps: 2 bits) in the following procedure (refer to **Figure 1-4**):

(1) A predicted signal is subtracted from the input signal (linear code) to calculate a differential signal.
(2) The differential signal is coded by using an adaptive quantization unit.
For 32-kbps ADPCM, the differential signal is coded to 4 bits by using a 15 level adaptive quantization unit.
For 16-kbps ADPCM, the differential signal is coded to 2 bits by using a 4 level adaptive quantization unit.
This coded signal is an ADPCM code and is the output of the compression processing.
(3) An adaptive reverse quantization unit is used to generate a quantized differential signal from the ADPCM code.
(4) The quantized differential signal is added to the predicted signal to reproduce the input signal.
(5) An adaptive prediction unit is used to calculated the predicted value of the next input signal from the reproduced signal and quantized differential signal.

Figure 1-4. Organization of ADPCM Compression Block

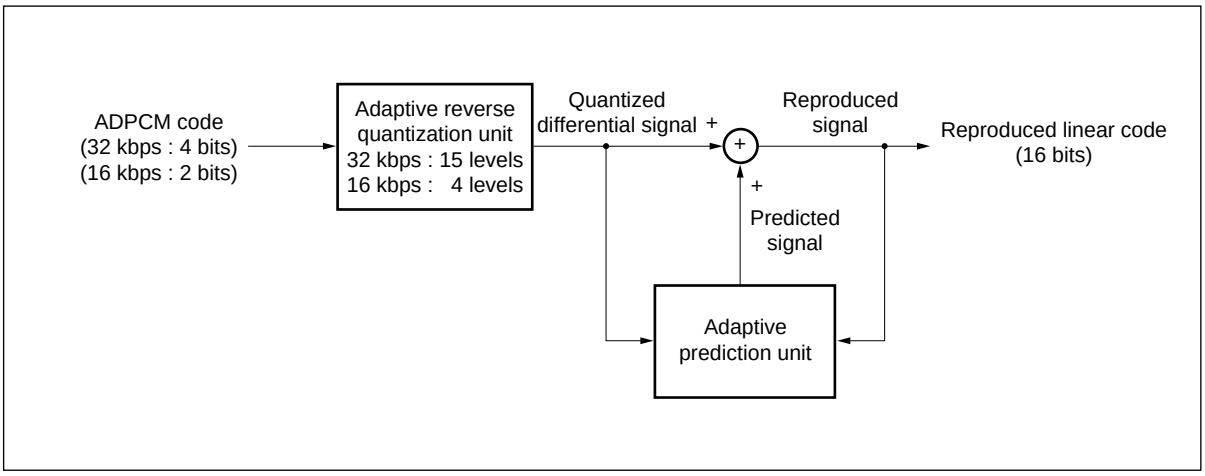


1.2.3 ADPCM expansion block

This block expands an ADPCM code (32 kbps: 4 bits, 16 kbps: 2 bits) to a reproduced linear code (16 bits) in the following procedure (refer to **Figure 1-5**):

(1) A quantized differential signal is generated from the ADPCM code by using an adaptive reverse quantization unit.
(2) The input signal is reproduced by adding the quantized differential signal to a predicted signal. This signal is the output signal of the ADPCM expansion processing.
(3) An adaptive prediction unit is used to calculate the predicted value of the next input signal from the reproduced signal and quantized differential signal.

Figure 1-5. Organization of ADPCM Expansion Block



1.2.4 Linear-to-PCM conversion block

This block converts a reproduced linear code (16 bits) to a 64-kbps PCM code (8 bits).

The PCM code in the μ-law mode is converted by a PCM coder conforming to Recommendation G.711.

The PCM code in the A-law mode is converted by a PCM coder conforming to Recommendation G.711 after the reproduced linear code has been adjusted (1/2)

1.2.5 Synchronous coding correction block

The synchronous coding block corrects for cumulative distortion that may take place when compression from PCM to ADPCM and expansion from ADPCM to PCM (synchronous tandem coding) are repeatedly performed.

The value of the reproduced linear code, which is the input signal, is corrected by simulating the ADPCM code to be compressed next by using the PCM-to-linear conversion block described in 1.2.1 and in steps (1) and (2) described in 1.2.2 ADPCM compression block, so that the value of the reproduced linear code is the same as that of the current ADPCM code.

1.3 Product Outline

1.3.1 Features

Conforms to the international ADPCM standard (ITU-T Recommendation G.726).

(1) Bit rate

The following two bit rates are supported:

- 32-kbps ADPCM (compresses 1 sampling data to 4 bits)
- 16-kbps ADPCM (compresses 1 sampling data to 2 bits)

(2) 64 kbps PCM code interface

The following two modes conforming to the international 64-kbps PCM standard (ITU-T Recommendation G.711) are supported:

- μ-law mode
- A-law mode

If synchronous tandem coding is not performed for expansion, a mode in which the synchronous coding correction function is deleted from the ITU-T Recommendation G.726, with increased execution speed, is supported.

(3) Linear code interface

A liner code interface that deletes the functions (mutual conversion of 64 kbps PCM code and linear code and synchronous coding correction) that are related to the 64-kbps PCM code interface and that can be deleted from the international standard of ADPCM (ITU-T Recommendation G.726) is supported.

Remark The linear code interface does not support synchronous tandem coding.

1.3.2 Operating environment

(1) Target CPU

- V810 family
- V850 family

(2) Necessary memory

- ROM: total capacity of 9K bytes or less (The capacity varies depending on the functions to be linked.)
- RAM: 80 bytes max.

★ **(3) Supported tools**

(a) V810 family

- NEC C compiler package: CA732 (Windows 3.1 or above, SUN4™)
- GHS C compiler/assembler: C-V810 (Windows 3.1 or above, SUN4)

(b) V850 family

- NEC C compiler package: CA850 (Windows 3.1 or above, SUN4)
- GHS C compiler/assembler: C-V850 (Windows 3.1 or above, SUN4)

1.3.3 Performance

The processing time per sampling data is shown below.

(1) V810

[Condition] 25 MHz, 32-bit bus, cache OFF, 0 wait ROM/RAM

Table 1-1. V810 Performance

Compression/Expansion		Processing Time
Compression	Linear code interface	62 μs max.
	PCM code interface	63 μs max.
Expansion	Linear code interface (synchronous tandem coding not supported)	59 μs max.
	PCM code interface (synchronous tandem coding not supported)	61 μs max.
	PCM code interface	66 μs max.

(2) V853

[Condition] 33 MHz, internal ROM/RAM, 0 wait, external RAM (speech data)

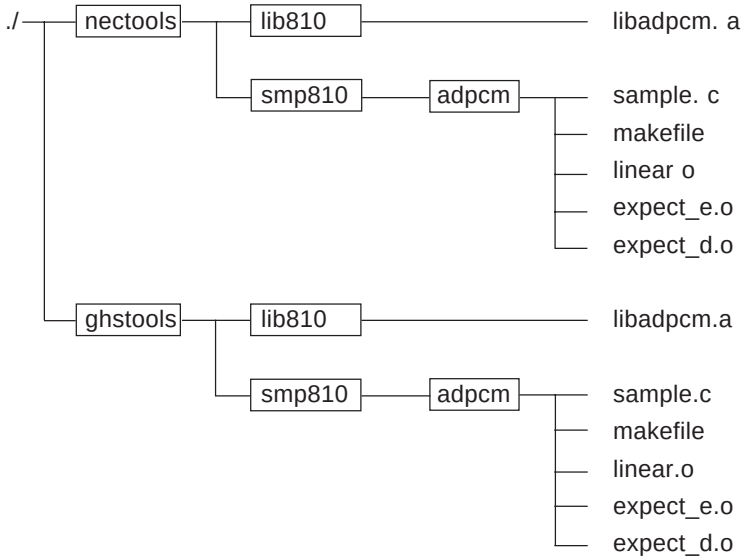
Table 1-2. V853 Performance

Compression/Expansion		Processing Time
Compression	Linear code interface	29 μs max.
	PCM code interface	30 μs max.
Expansion	Linear code interface (synchronous tandem coding not supported)	28 μs max.
	PCM code interface (synchronous tandem coding not supported)	29 μs max.
	PCM code interface	31 μs max.

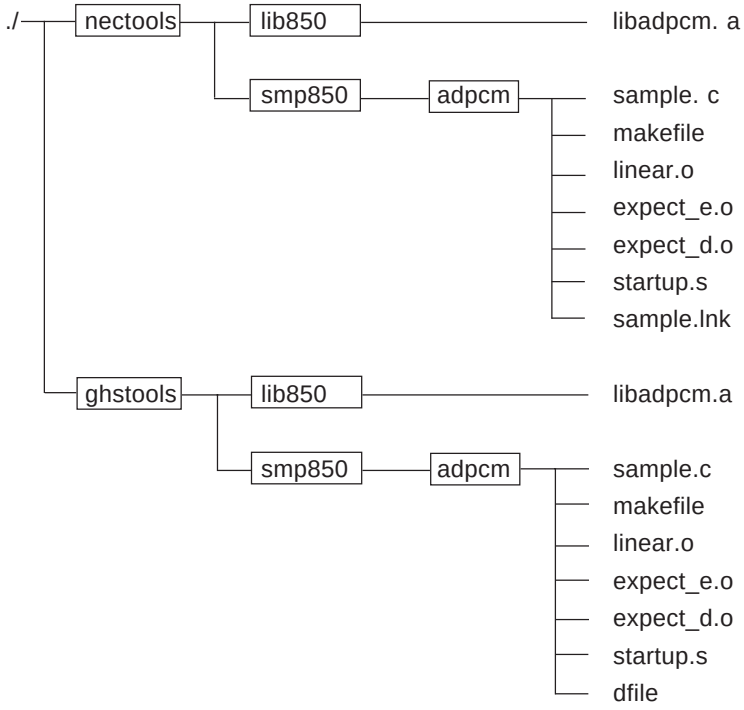
★ 1.3.4 Directory configuration

The directory configuration of the middleware library ADPCM is as follows:

(1) V810 family (software version: Ver. 1.00)



(2) V850 family (software version: Ver. 1.01)



[MEMO]

CHAPTER 2 LIBRARY SPECIFICATIONS

2.1 Function

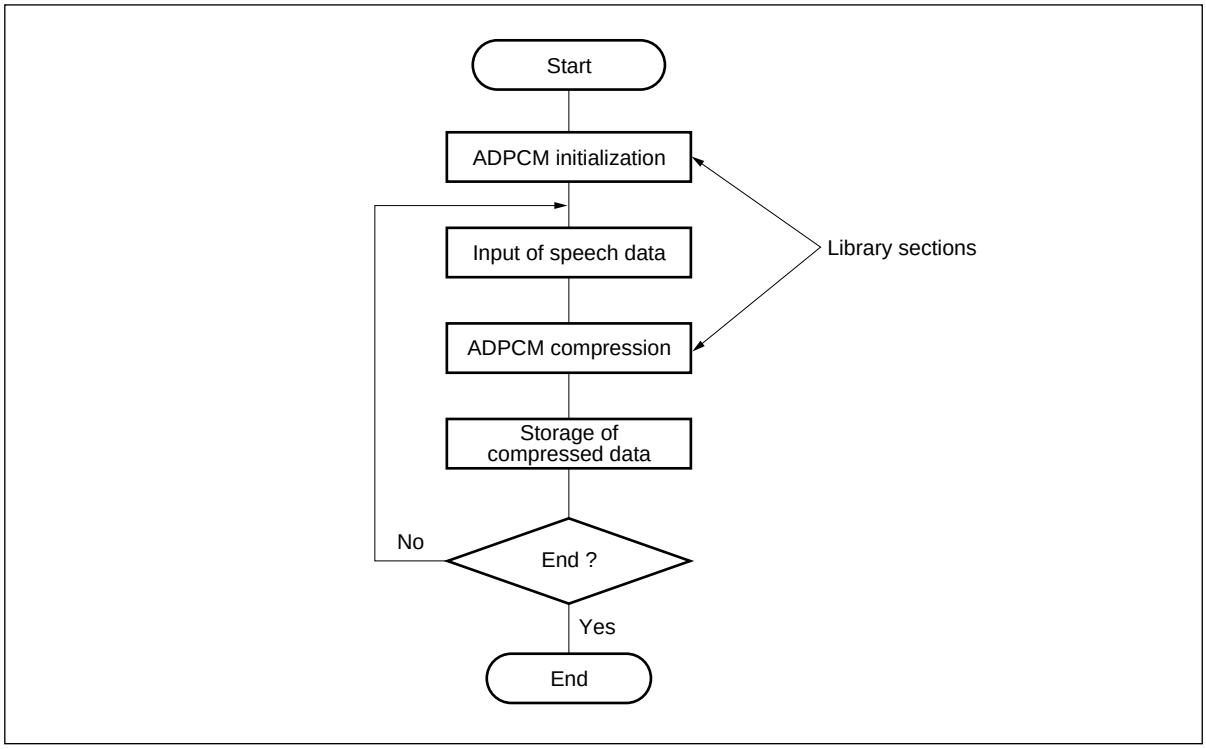
2.1.1 Compression processing

Speech data is compressed to ADPCM codes of 32 kbps or 16 kbps. The speech data which is input is classified according to functions corresponding to linear codes and 64-kbps PCM codes (μ-law, A-law).

Table 2-1. Compression Functions

Function Classification	Input Data Type → Compressed Data Type
adpcm_132_enc ( )	Linear code → 32-kbps ADPCM
adpcm_pm32_enc ( )	64-kbps PCM code (μ-law) → 32-kbps ADPCM
adpcm_pa32_enc ( )	64-kbps PCM code (A-law) → 32-kbps ADPCM
adpcm_I16_enc ( )	Linear code → 16-kbps ADPCM
adpcm_pm16_enc ( )	64-kbps PCM code (μ-law) → 16-kbps ADPCM
adpcm_pa16_enc ( )	64-kbps PCM code (A-law) → 16-kbps ADPCM

Figure 2-1. Compression Flow



2.1.2 Expansion processing

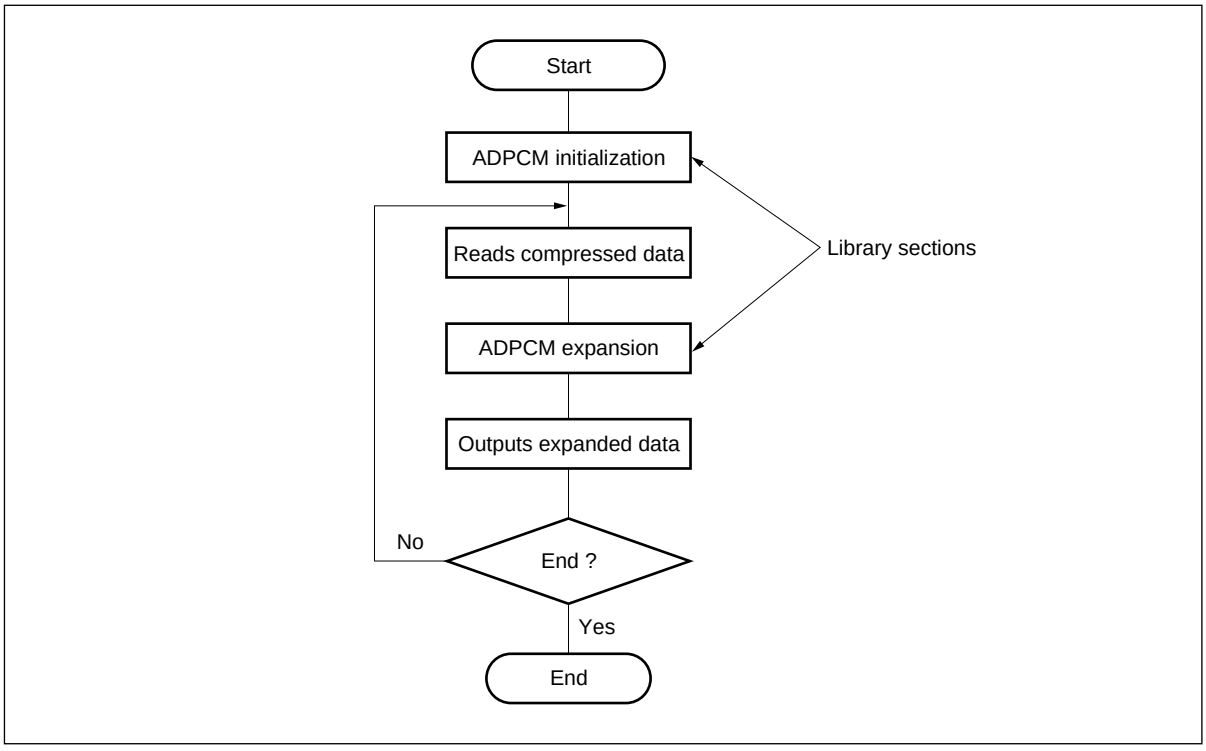
Compressed data (ADPCM codes of 32 kbps or 16 kbps) is expanded to speech data. The speech data which is output is classified according to functions corresponding to linear codes and 64-kbps PCM codes (μ-law, A-law).

Table 2-2. Expansion Functions

Function Classification	Compressed Data Type → Output Data Type
adpcm_l32_dec ( )	32-kbps ADPCM → linear code
adpcm_pm32_dec ( )	32-kbps ADPCM → 64-kbps PCM code (μ-law)
adpcm_pa32_dec ( )	32-kbps ADPCM → 64-kbps PCM code (A-law)
adpcm_tpm32_dec ( )[Note]	32-kbps ADPCM → 64-kbps PCM code (μ-law)
adpcm_tpa32_dec ( )[Note]	32-kbps ADPCM → 64-kbps PCM code (A-law)
adpcm_l16_dec ( )	64-kbps ADPCM → linear code
adpcm_pm16_dec ( )	16-kbps ADPCM → 64-kbps PCM code (μ-law)
adpcm_pa16_dec ( )	16-kbps ADPCM → 64-kbps PCM code (A-law)
adpcm_tpm16_dec ( )[Note]	16-kbps ADPCM → 64-kbps PCM code (μ-law)
adpcm_tpa16_dec ( )[Note]	16-kbps ADPCM → 64-kbps PCM code (A-law)

Note These functions support tandem synchronous coding.

Figure 2-2. Expansion Processing Flow



2.2 RAM

Be sure to secure the determined memory capacity (resident area) for each of the compression and expansion processing with this library. To secure the resident area, perform the following in the user application:

int work[16]

Manage the resident area in the user application, and specify a pointer (first address) with the input parameter of the compression/expansion function. Do not destroy the contents of the resident area until the compression/expansion processing sequence is completed. The operation is not guaranteed if the resident area is destroyed. This library also uses the following stack area for the compression/expansion functions:

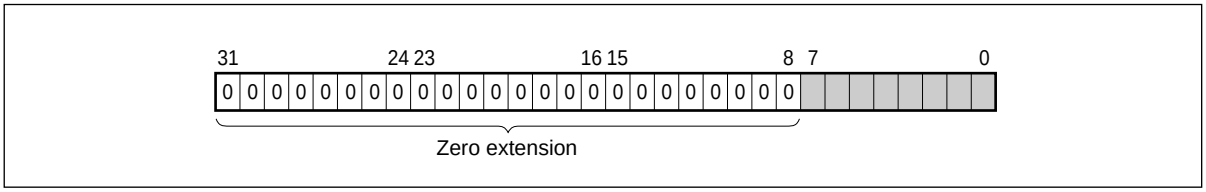
- V810 family version: 3 words (12 bytes)
- V850 family version: 2 words (8 bytes)

2.3 Data Type

This section describes the data used with this library. Each data type is standardized to int width (32 bits) to increase the processing speed. Note that the valid range is not checked in each function.

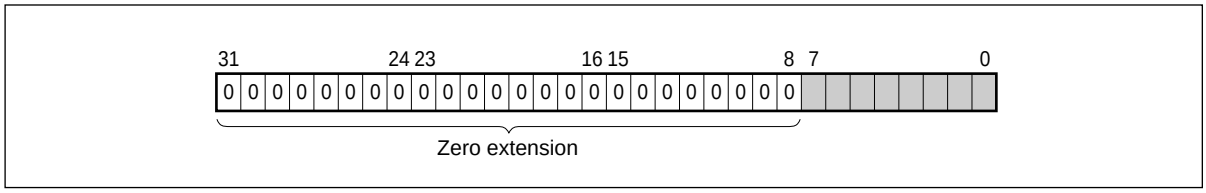
2.3.1 int pcm_mlaw

[Contents] 64-kbps PCM code (μ-law)
[Description] 8-bit data (Zero-extend up to bit 31.)
[Valid range] 0 to 255



2.3.2 int pcm_alaw

[Contents] 64-kbps PCM code (A-law)
[Description] 8-bit data (Zero-extend up to bit 31.)
[Valid range] 0 to 255

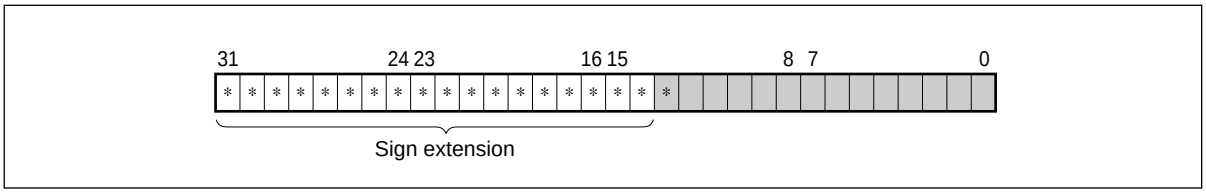


2.3.3 int linear_enc

[Contents] Linear code (compression processing argument)

[Description] 14-bit data (expressed as 2's complement. Sign-extend up to bit 31.)

[Valid range] –8192 to 8191

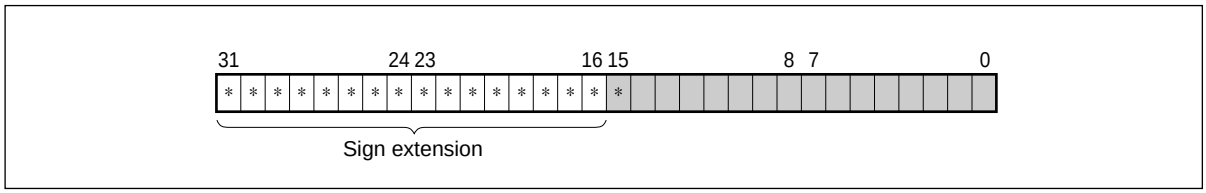


2.3.4 int linear_dec

[Contents] Linear code (expansion processing return value)

[Description] 16-bit data (expressed as 2's complement. Sign-extended up to bit 31)

[Valid range] –32768 to 32767

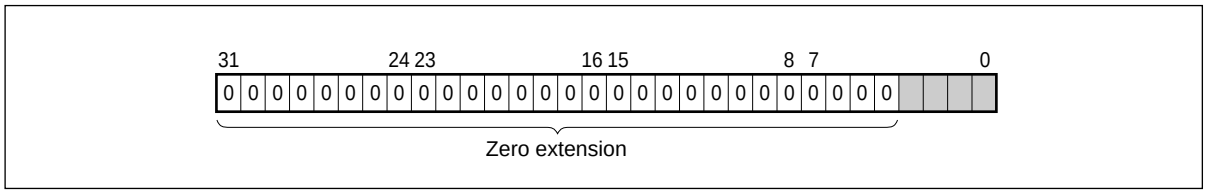


2.3.5 int adpcm_32kbps

[Contents] 32-kbps ADPCM code

[Description] 4-bit data (Zero-extend up to bit 31.)

[Valid range] 1 to 15

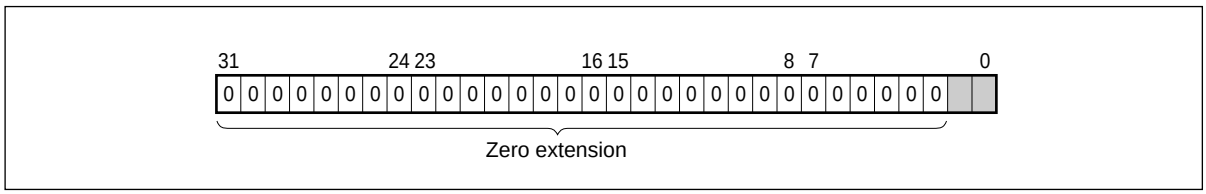


2.3.6 int adpcm_16kbps

[Contents] 16-kbps ADPCM code

[Description] 2-bit data (Zero-extend up to bit 31.)

[Valid range] 0 to 3



2.4 Error Processing

The functions of the ADPCM library do not have an error processing functions like outputting an error code as a return value. This is because the value outside the valid range in the middle of calculation is limited (saturated) by the Recommendation and therefore, errors do not exist. However, errors, such as NMI, may occur. If these errors occur, deal with them in the user application by managing flags before and after function calls.

2.5 Function Specifications

The ADPCM library supplies a total of 17 functions including an initialization function, six compression (encoder) functions, and 10 expansion (decoder) functions. These functions are described next.

2.5.1 Initialization function

(1) adpcm_init function

This function initializes the encoder and decoder.
Secure a resident area (64 bytes) for the processing of the encoder and decoder in the application program, and use the addresses of that area as arguments.
Be sure to call the adpcm_init function before performing an encoder/decoder processing sequence.

[Classification] ADPCM initialization (common to encoder and decoder)
[Function name] adpcm_init
[Outline] Initializes the resident area of the ADPCM encoder and decoder.
[Format] void adpcm_init (init *_work_);

[Argument]

Type	Argument	Description
int	*work* [*16*]	First address of resident area (64 bytes)

[Return value] None
[Feature] Initializes the resident area for the encode and decode processing.

Caution After calling the adpcm_init function, do not destroy the resident RAM area until the encode and decode processing sequence is completed. If the resident area is destroyed, the operation is not guaranteed.

2.5.2 Compression function

(1) adpcm_I32_enc function

This function performs 32-kbps ADPCM encode processing of the linear code interface.
It encodes a linear code given by the input parameter, and returns a 32-kbps ADPCM code as a return value.
Of the parameters to be passed to the adpcm_I32_enc function, specify the same address as that passed to the adpcm_init function as the address of the resident area for encode processing.

[Classification] 32-kbps ADPCM encode processing (linear code interface)
[Function name] adpcm_I32_enc
[Outline] Encodes a specified linear code to a 32-kbps ADPCM code.
[Format] int adpcm_I32_enc (int linear_enc, int **work*)

[Argument]

Type	Argument	Description
int	*work* [*16*]	First address of resident area (64 bytes)
int	linear_enc	Linear code

[Return value]

Type	Return Value	Description
int	adpcm_32	32-kbps ADPCM code

[Feature] Encodes sampling data given by a linear code to a 32-kbps ADPCM code.
[Remark] The following function is deleted from the 32-kbps ADPCM encoder of ITU-T Recommendation.

- Conversion from 64-kbps PCM code to linear code

(2) adpcm_pm32_enc function

This function performs 32-kbps ADPCM encode processing of the μ-law PCM code interface.
It encodes a 64-kbps μ-law PCM code given by the input parameter, and returns a 32-kbps ADPCM code as a return value.
Of the parameters to be passed to the adpcm_pm32_enc function, specify the same address as that passed to the adpcm_init function as the address of the resident area for encode processing.

[Classification] 32-kbps ADPCM encode processing (μ-law PCM code interface)
[Function name] adpcm_pm32_enc
[Outline] Encodes a specified 64-kbps μ-law PCM code to a 32-kbps ADPCM code.
[Format] int adpcm_pm32_enc (int pcm_mlaw, int **work*)

[Argument]

Type	Argument	Description
int	*work* [*16*]	First address of resident area (64 bytes)
int	pcm_mlaw	64-kbps μ-law PCM

[Return value]

Type	Return Value	Description
int	adpcm_32	32-kbps ADPCM code

[Feature] Encodes sampling data given by a 64-kbps μ-law PCM code to a 32-kbps ADPCM code.

(3) adpcm_pa32_enc function

This function performs 32-kbps ADPCM encode processing of the A-law PCM code interface.
It encodes a 64-kbps A-law PCM code given by the input parameter, and returns a 32-kbps ADPCM code as a return value.
Of the parameters to be passed to the adpcm_pa32_enc function, specify the same address as that passed to the adpcm_init function as the address of the resident area for encode processing.

[Classification] 32-kbps ADPCM encode processing (A-law PCM code interface)
[Function name] adpcm_pa32_enc
[Outline] Encodes a specified a 64-kbps A-law PCM code to a 32-kbps ADPCM code.
[Format] int adpcm_pa32_enc (int pcm_alaw, int **work*)

[Argument]

Type	Argument	Description
int	*work* [*16*]	First address of resident area (64 bytes)
int	pcm_alaw	64-kbps A-law PCM code

[Return value]

Type	Return Value	Description
int	adpcm_32	32-kbps ADPCM code

[Feature] Encodes sampling data given by a 64-kbps A-law code to a 32-kbps ADPCM code.

(4) adpcm_I16_enc function

This function performs 16-kbps ADPCM encode processing of the linear code interface.
It encodes a linear code given by the input parameter, and returns a 16-kbps ADPCM code as a return value.
Of the parameters to be passed to the adpcm_I16_enc function, specify the same address as that passed to the adpcm_init function as the address of the resident area for encode processing.

[Classification] 16-kbps ADPCM encode processing (A-law PCM code interface)
[Function name] adpcm_I16_enc
[Outline] Encodes a specified linear code to a 32-kbps ADPCM code.
[Format] int adpcm_I16_enc (int pcm_alaw, int **work*)

[Argument]

Type	Argument	Description
int	*work* [*16*]	First address of resident area (64 bytes)
int	linear_enc	Linear code

[Return value]

Type	Return Value	Description
int	adpcm_16	16-kbps ADPCM code

[Feature] Encodes sampling data given by a linear code to a 16-kbps ADPCM code.
[Remark] The following function is deleted from the 16-kbps ADPCM encoder of ITU-T Recommendation.

- Conversion from 64-kbps PCM code to linear code

(5) adpcm_pm16_enc function

This function performs 16-kbps ADPCM encode processing of the μ-law PCM code interface.
It encodes a 64-kbps μ-law PCM code given by the input parameter, and returns a 16-kbps ADPCM code as a return value.
Of the parameters to be passed to the adpcm_pm16_enc function, specify the same address as that passed to the adpcm_init function as the address of the resident area for encode processing.

[Classification] 16-kbps ADPCM encode processing (μ-law PCM code interface)
[Function name] adpcm_pm16_enc
[Outline] Encodes a specified a 64-kbps μ-law PCM code to a 16-kbps ADPCM code.
[Format] int adpcm_pm16_enc (int pcm_mlaw, int **work*)

[Argument]

Type	Argument	Description
int	*work* [*16*]	First address of resident area (64 bytes)
int	pcm_mlaw	64-kbps μ-law PCM code

[Return value]

Type	Return Value	Description
int	adpcm_16	16-kbps ADPCM code

[Feature] Encodes sampling data given by a 64-kbps μ-law PCM code to a 16-kbps ADPCM code.

(6) adpcm_pa16_enc function

This function performs 16-kbps ADPCM encode processing of the A-law PCM code interface.
It encodes a 64-kbps A-law PCM code given by the input parameter, and returns a 16-kbps ADPCM code as a return value.
Of the parameters to be passed to the adpcm_pa16_enc function, specify the same address as that passed to the adpcm_init function as the address of the resident area for encode processing.

[Classification] 16-kbps ADPCM encode processing (A-law PCM code interface)
[Function name] adpcm_pa16_enc
[Outline] Encodes a specified a 64-kbps A-law PCM code to a 16-kbps ADPCM code.
[Format] int adpcm_pa16_enc (int pcm_alaw, int **work*)

[Argument]

Type	Argument	Description
int	*work* [*16*]	First address of resident area (64 bytes)
int	pcm_alaw	64-kbps A-law PCM code

[Return value]

Type	Return Value	Description
int	adpcm_16	16-kbps ADPCM code

[Feature] Encodes sampling data given by a 64-kbps A-law PCM code to a 16-kbps ADPCM code.

2.5.3 Expansion function

(1) adpcm_I32_dec function

This function performs 32-kbps ADPCM decode processing of the linear code interface.
It decodes a 32-kbps ADPCM code given by the input parameter, and returns a linear code as a return value.
Of the parameters to be passed to the adpcm_I32_dec function, specify the same address as that passed to the adpcm_init function as the address of the resident area for decode processing.

[Classification] 32-kbps ADPCM decode processing (linear code interface)
[Function name] adpcm_I32_dec
[Outline] Decodes a specified 32-kbps ADPCM code to a linear code.
[Format] int adpcm_I32_dec (int adpcm-32, int **work*)

[Argument]

Type	Argument	Description
int	*work* [*16*]	First address of resident area (64 bytes)
int	adpcm-32	32-kbps ADPCM code

[Return value]

Type	Return Value	Description
int	linear_dec	Linear code

[Feature] Decodes sampling data given by a 32-kbps ADPCM code to a linear code.
[Remark] The following function is deleted from the 32-kbps ADPCM decoder of ITU-T Recommendation.

- Conversion from linear code to 64-kbps PCM code
- Synchronous coding correction

(2) adpcm_pm32_dec function

This function performs 32-kbps ADPCM decode processing of the μ-law PCM code interface.
It decodes a 32-kbps ADPCM code given by the input parameter, and returns a 64-kbps μ-law PCM code as a return value.
Of the parameters to be passed to the adpcm_pm32_dec function, specify the same address as that passed to the adpcm_init function as the address of the resident area for decode processing.

[Classification] 32-kbps ADPCM decode processing (μ-law PCM code interface)
[Function name] adpcm_pm32_dec
[Outline] Decodes a specified a 32-kbps ADPCM code to a 64-kbps μ-law PCM code.
[Format] int adpcm_pm32_dec (int adpcm-32, int **work*)

[Argument]

Type	Argument	Description
int	*work* [*16*]	First address of resident area (64 bytes)
int	adpcm_32	32-kbps ADPCM code

[Return value]

Type	Return Value	Description
int	pcm_mlaw	64-kbps μ-law PCM code

[Feature] Decodes sampling data given by a 32-kbps ADPCM code to a 64-kbps μ-law PCM code.
[Remark] The following function is deleted from the 32-kbps ADPCM decoder of ITU-T Recommendation.

- Synchronous coding correction.

(3) adpcm_pa32_dec function

This function performs 32-kbps ADPCM decode processing of the A-law PCM code interface.
It decodes a 32-kbps ADPCM code given by the input parameter, and returns a 64-kbps A-law PCM code as a return value.
Of the parameters to be passed to the adpcm_pa32_dec function, specify the same address as that passed to the adpcm_init function as the address of the resident area for decode processing.

[Classification] 32-kbps ADPCM decode processing (A-law PCM code interface)
[Function name] adpcm_pa32_dec
[Outline] Decodes a specified a 32-kbps ADPCM code to a 64-kbps A-law PCM code.
[Format] int adpcm_pa32_dec (int adpcm-32, int **work*)

[Argument]

Type	Argument	Description
int	*work* [*16*]	First address of resident area (64 bytes)
int	adpcm_32	32-kbps ADPCM code

[Return value]

Type	Return Value	Description
int	pcm_alaw	64-kbps A-law PCM code

[Feature] Decodes sampling data given by a 32-kbps ADPCM code to a 64-kbps A-law PCM code.
[Remark] The following function is deleted from the 32-kbps ADPCM decoder of ITU-T Recommendation.

- Synchronous coding correction.

(4) adpcm_tpm32_dec function

This function performs 32-kbps ADPCM decode processing of the μ-law PCM code interface.
It decodes a 32-kbps ADPCM code given by the input parameter, and returns a 64-kbps μ-law PCM code as a return value.
Of the parameters to be passed to the adpcm_tpm32_dec function, specify the same address as that passed to the adpcm_init function as the address of the resident area for decode processing.

[Classification] 32-kbps ADPCM decode processing (μ-law PCM code interface)
[Function name] adpcm_tpm32_dec
[Outline] Decodes a specified a 32-kbps ADPCM code to a 64-kbps μ-law PCM code.
[Format] int adpcm_tpm32_dec (int adpcm-32, int **work*)

[Argument]

Type	Argument	Description
int	*work* [*16*]	First address of resident area (64 bytes)
int	adpcm_32	32-kbps ADPCM code

[Return value]

Type	Return Value	Description
int	pcm_mlaw	64-kbps μ-law PCM code

[Feature] Decodes sampling data given by a 32-kbps ADPCM code to a 64-kbps μ-law PCM code.

(5) adpcm_tpa32_dec function

This function performs 32-kbps ADPCM decode processing of the A-law PCM code interface.
It decodes a 32-kbps ADPCM code given by the input parameter, and returns a 64-kbps A-law PCM code as a return value.
Of the parameters to be passed to the adpcm_tpa32_dec function, specify the same address as that passed to the adpcm_init function as the address of the resident area for decode processing.

[Classification] 32-kbps ADPCM decode processing (A-law PCM code interface)
[Function name] adpcm_tpa32_dec
[Outline] Decodes a specified a 32-kbps ADPCM code to a 64-kbps A-law PCM code.
[Format] int adpcm_tpa32_dec (int adpcm_32, int **work*)

[Argument]

Type	Argument	Description
int	*work* [*16*]	First address of resident area (64 bytes)
int	adpcm_32	32-kbps ADPCM code

[Return value]

Type	Return Value	Description
int	pcm_alaw	64-kbps A-law PCM code

[Feature] Decodes sampling data given by a 32-kbps ADPCM code to a 64-kbps A-law PCM code.

(6) adpcm_I16_dec function

This function performs 16-kbps ADPCM decode processing of the linear code interface.
It decodes a 16-kbps ADPCM code given by the input parameter, and returns a linear code as a return value.
Of the parameters to be passed to the adpcm_I16_dec function, specify the same address as that passed to the adpcm_init function as the address of the resident area for decode processing.

[Classification] 16-kbps ADPCM decode processing (linear code interface)
[Function name] adpcm_I16_dec
[Outline] Decodes a specified a 16-kbps ADPCM code to a linear code.
[Format] int adpcm_I16_dec (int adpcm_16, int **work*)

[Argument]

Type	Argument	Description
int	*work* [*16*]	First address of resident area (64 bytes)
int	adpcm_16	16-kbps ADPCM code

[Return value]

Type	Return Value	Description
int	linear_dec	Linear code

[Feature] Decodes sampling data given by a 16-kbps ADPCM code to a linear code.
[Remark] The following function is deleted from the 32-kbps ADPCM decoder of ITU-T Recommendation.

- Conversion from linear code to 64-kbps PCM code.
- Synchronous coding correction

(7) adpcm_pm16_dec function

This function performs 16-kbps ADPCM decode processing of the μ-law PCM code interface.
It decodes a 16-kbps ADPCM code given by the input parameter, and returns a 64-kbps μ-law PCM code as a return value.
Of the parameters to be passed to the adpcm_pm16_dec function, specify the same address as that passed to the adpcm_init function as the address of the resident area for decode processing.

[Classification] 16-kbps ADPCM decode processing (μ-law PCM code interface)
[Function name] adpcm_pm16_dec
[Outline] Decodes a specified 16-kbps ADPCM code to a μ-law PCM code.
[Format] int adpcm_pm16_dec (int adpcm_16, int **work*)

[Argument]

Type	Argument	Description
int	*work* [*16*]	First address of resident area (64 bytes)
int	adpcm_16	16-kbps ADPCM code

[Return value]

Type	Return Value	Description
int	pcm_mlaw	64-kbps μ-law PCM code

[Feature] Decodes sampling data given by a 16-kbps ADPCM code to a 64-kbps μ-law PCM code.
[Remark] The following function is deleted from the 16-kbps ADPCM decoder of ITU-T Recommendation.

- Synchronous coding correction

(8) adpcm_pa16_dec function

This function performs 16-kbps ADPCM decode processing of the A-law PCM code interface.
It decodes a 16-kbps ADPCM code given by the input parameter, and returns a 64-kbps A-law PCM code as a return value.
Of the parameters to be passed to the adpcm_pa16_dec function, specify the same address as that passed to the adpcm_init function as the address of the resident area for decode processing.

[Classification] 16-kbps ADPCM decode processing (A-law PCM code interface)
[Function name] adpcm_pa16_dec
[Outline] Decodes a specified 16-kbps ADPCM code to a 64-kbps A-law PCM code.
[Format] int adpcm_pa16_dec (int adpcm_16, int **work*)

[Argument]

Type	Argument	Description
int	*work* [*16*]	First address of resident area (64 bytes)
int	adpcm_16	16-kbps ADPCM code

[Return value]

Type	Return Value	Description
int	pcm_alaw	64-kbps A-law PCM code

[Feature] Decodes sampling data given by a 16-kbps ADPCM code to a 64-kbps A-law PCM code.
[Remark] The following function is deleted from the 16-kbps ADPCM decoder of ITU-T Recommendation.

- Synchronous coding correction

(9) adpcm_tpm16_dec function

This function performs 16-kbps ADPCM decode processing of the μ-law PCM code interface.
It decodes a 16-kbps ADPCM code given by the input parameter, and returns a 64-kbps μ-law PCM code as a return value.
Of the parameters to be passed to the adpcm_tpm16_dec function, specify the same address as that passed to the adpcm_init function as the address of the resident area for decode processing.

[Classification] 16-kbps ADPCM decode processing (μ-law PCM code interface)
[Function name] adpcm_tpm16_dec
[Outline] Decodes a specified 16-kbps ADPCM code to a μ-law PCM code.
[Format] int adpcm_tpm16_dec (int adpcm_16, int *_work_)

[Argument]

Type	Argument	Description
int	*work* [*16*]	First address of resident area (64 bytes)
int	adpcm_16	16-kbps ADPCM code

[Return value]

Type	Return Value	Description
int	pcm_mlaw	64-kbps μ-law PCM code

[Feature] Decodes sampling data given by a 16-kbps ADPCM code to a 64-kbps μ-law PCM code.

(10) adpcm_tpa16_dec function

This function performs 16-kbps ADPCM decode processing of the A-law PCM code interface.
It decodes a 16-kbps ADPCM code given by the input parameter, and returns a 64-kbps A-law PCM code as a return value.
Of the parameters to be passed to the adpcm_tpa16_dec function, specify the same address as that passed to the adpcm_init function as the address of the resident area for decode processing.

[Classification] 16-kbps ADPCM decode processing (A-law PCM code interface)
[Function name] adpcm_tpa16_dec
[Outline] Decodes a specified 16-kbps ADPCM code to a 64-kbps A-law PCM code.
[Format] int adpcm_tpa16_dec (int adpcm_16, int *_work_)

[Argument]

Type	Argument	Description
int	*work* [*16*]	First address of resident area (64 bytes)
int	adpcm_16	16-kbps ADPCM code

[Return value]

Type	Return Value	Description
int	pcm_alaw	64-kbps A-law PCM code

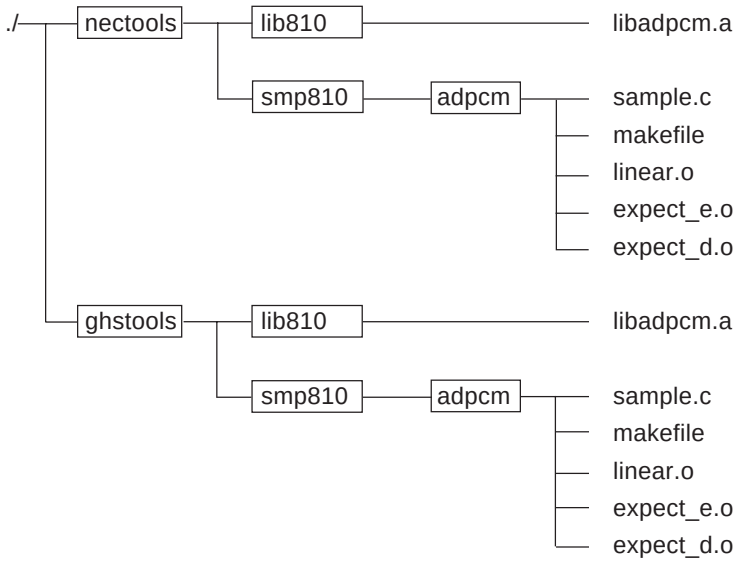
[Feature] Decodes sampling data given by a 16-kbps ADPCM code to a 64-kbps A-law PCM code.

CHAPTER 3 INSTALLATION

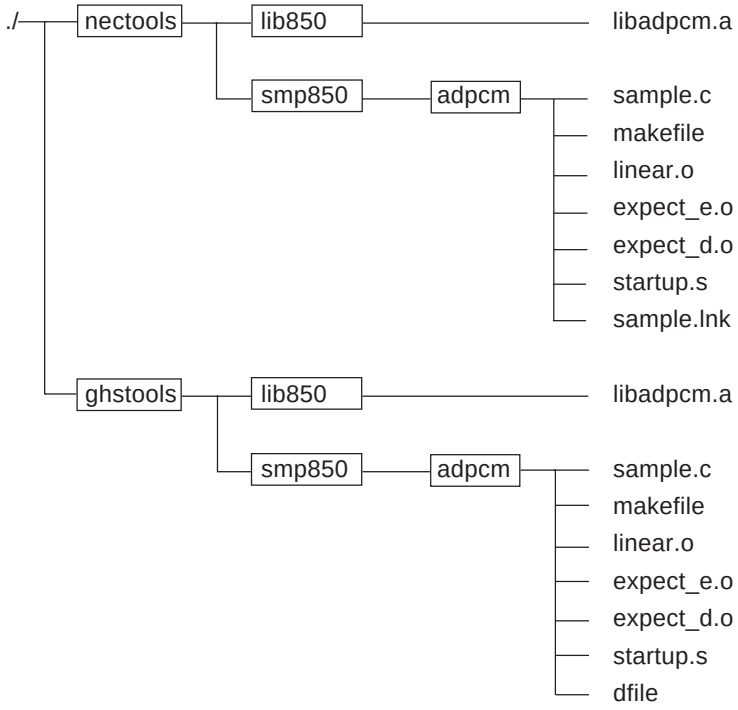
3.1 Supply Format

The middleware library ADPCM is a library for developing applications using NEC or GHS tools. Therefore, it is available in two media, one for NEC tools and the other for GHS tools. The contents of the supply media are as follows:

★ **3.1.1 V810 family**



★ **3.1.2 V850 family**



3.1.3 Directory and file

The functions of each directory and file are as follows:

★ **(1) nectools**

Contains the ADPCM program used to develop applications using NEC tools.

★ **(2) ghstools**

Contains the ADPCM program used to develop applications using GHS tools.

(3) lib850

Contains the library of the ADPCM.

(4) smp810/adpcm, smp850/adpcm

Contains a sample program using the ADPCM and sample speech data.

3.2 File Expansion to Host Machine

This section describes the procedure, for the UNIX version (SUN4) and MS-DOS™/PC DOS™ version, to transfer the files from the supply media to the host machine.

3.2.1 UNIX version

The UNIX version is available in two media: CGMT and 3.5" FD. The ADPCM files are stored in these media in the tar format. The procedure for installing ADPCM to the host machine is as follows.

<1> Create a directory to install the ADPCM. In this example, a directory named mw_adpcm is created.

```
% mkdir mw_adpcm <CR>
```

<2> Move to the created directory.

```
% cd mw_adpcm <CR>
```

<3> Insert the supply media.
- Insert the CGMT in the magnetic tape unit.
- Insert the 3.5" FD in the floppy disk drive.

<4> Execute the tar command to expand the files on disk. The special file name to be specified differs depending on the host machine.
In this example, only the files for NEC tools are expanded, assuming that the special file name is /dev/rst8.

★
```
% tar -xvof /dev/rst8 nectools <CR>
```

To expand the files for GHS tools, input the following:

★
```
% tar -xvof /dev/rst8 ghstools <CR>
```

<5> Confirm that the files have been installed. For each directory, refer to **3.1 Supply Format**.

```
% ls -CFR <CR>
```

3.2.2 MS-DOS/PC DOS version

The MS-DOS/PC DOS version is supplied on a 3.5” FD. Install the ADPCM on the host machine in the following procedure:

<1> Create a directory to which the ADPCM is to be installed. In this example, a directory named mw_adpcm is created in drive A.

```
A> md mw_adpcm <CR>
```

<2> Move to the created directory.

```
A> cd mw_adpcm <CR>
```

<3> Insert the supply media in the floppy disk drive. In this example, insert the disk in drive C.

<4> Execute the xcopy command to expand the files. In this example, only the files for NEC tools are expanded.

★

```
A> xcopy c: \nectools . /s /e /v <CR>
```

Similarly, expand the files for GHS tools as follows:

★

```
A> xcopy c: \ghstools . /s /e /v <CR>
```

<5> Confirm that the files have been installed. For each directory, refer to **3.1 Supply Format**.

★

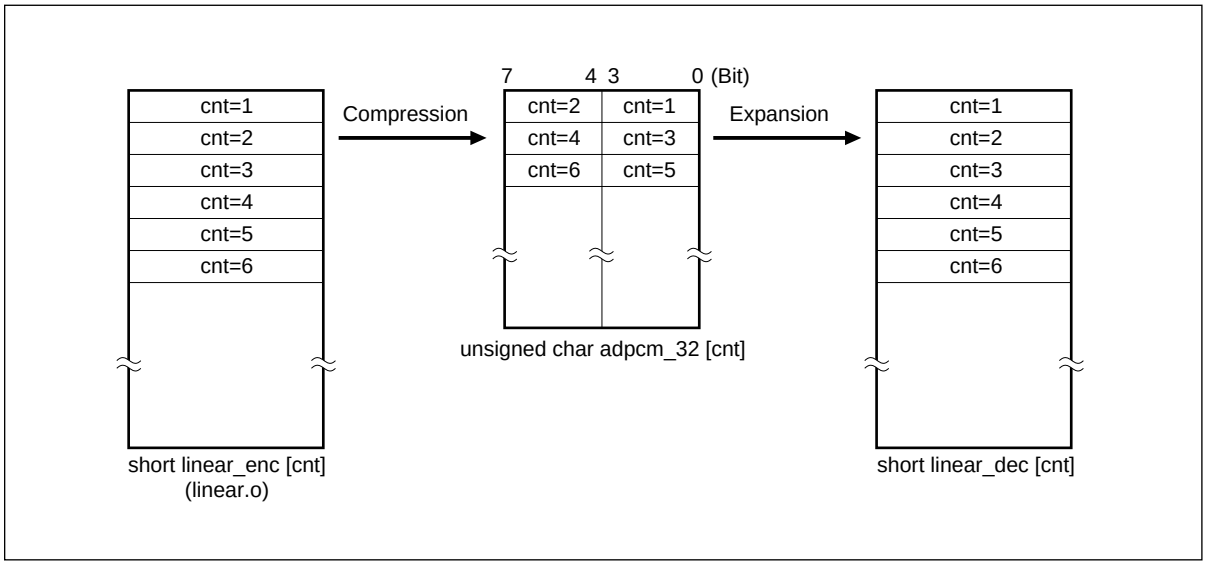
```
A> dir a: \nectools <CR>
```

3.3 Creating Sample Program

The directory smp810 (V810 family version) or smp850 (V850 family version) stores a sample program using the compression function and expansion function of 32 kbps ADPCM (linear code interface) and speech data (refer to **APPENDIX** for the source program of sample.c).

The sample program compresses a data array called linear_enc stored in the file linear.o, and the result of compression is stored in an array called adpcm_32. Next, this compressed data is expanded and the result of decoding (expansion) is stored in an array called linear_dec.

Figure 3-1. Organization of Sample Program



In the example given below, sample program “make”, NEC tools, and UNIX version are used.

<1> Move to the directory containing the sample program.

★

```
% cd mw_adpcm/nectools/smp810/adpcm <CR>
```

<2> Change startup.c in accordance with the target, by using an editor. startup.c supplied as a sample describes only initialization of the stack pointer.

★ <3> Execute the make command to create sample.elf

```
% make sample.elf <CR>
```

★ <4> Confirm that sample.elf has been created.

```
% ls - l sample.elf <CR>
```

<5> Download the sample program to the target using an in-circuit emulator and execute it.

3.4 Changing Location

The ADPCM library is given the following section name. The location can be changed in accordance with the user's target.

Section Name	Type	Description
.ad_text	.text	Text (instruction code)
.ad_data	.data	Table data (constant)

3.5 Symbol Name Convention

The symbol names used in the ADPCM library are in compliance with the following convention. When using the symbol names in combination with other applications, be careful not to duplicate them.

Classification	Convention
Function/symbol	"adpcm_" is prefixed.

★ APPENDIX SOURCE PROGRAM OF sample.c

```
--- sample.c---
extern short linear_enc[0x4000];
unsigned char adpcm_32[0x4000/2];
short linear_dec[0x4000];

int work[16];

void
encoder ()
{
      unsigned register int cnt;
      adpcm_init( work );
      for( cnt=0; cnt<0x4000; cnt++ ) {
          if(cnt & 1){
            adpcm_32[cnt>>1]|=((unsigned char)adpcm_132_enc((int)linear_enc[cnt],
            work)) << 4;
          }
          else{
            adpcm_32[cnt>>1]=(unsigned char)adpcm_132_enc((int)linear_enc[cnt],
            work);
          }
      }
}

void
decoder ()
{
      unsigned register int cnt;
      adpcm_init( work );
      for( cnt = 0; cnt<0x4000; cnt++ ){
          if(cnt & 1){
              linear_dec[cnt] = (short)adpcm_132_dec( ((int)adpcm_32[cnt>>1]) >> 4,
              work );
          }
          else{
              linear_dec[cnt] = (short)adpcm_132_dec( ((int)adpcm_32[cnt>>1]) & 0xf,
              work );
          }
      }
}

main ()
{
    encoder();
    decoder();
}
```

[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
Technical Documentation Dept.
Fax: +49-211-6503-274

South America
NEC do Brasil S.A.
Fax: +55-11-6465-6829

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

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

Taiwan
NEC Electronics Taiwan Ltd.
Fax: 02-719-5951

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

Japan
NEC Semiconductor Technical Hotline
Fax: 044-548-7900

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	❑	❑	❑	❑