NEC

User's Manual

μSAP77016-B03

G.729 Speech Codec Middleware

Target Device

μPD77016
μPD77017
μPD77018
μPD77018A
μPD77019
μPD77110
μPD77111
μPD77112
μPD77113
μPD77114
μPD77116

Document No. U13373EJ3V0UMJ1 (3rd edition)
Date Published October 2000 N CP(K)

[MEMO]

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

- **The information in this document is current as of November, 1999. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC's data sheets or data books, etc., for the most up-to-date specifications of NEC semiconductor products. Not all products and/or types are available in every country. Please check with an NEC sales representative for availability and additional information.**
- No part of this document may be copied or reproduced in any form or by any means without prior written consent of NEC. NEC assumes no responsibility for any errors that may appear in this document.
- NEC does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC semiconductor products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC or others.
- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of customer's equipment shall be done under the full responsibility of customer. NEC assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC endeavours to enhance the quality, reliability and safety of NEC semiconductor products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC semiconductor products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment, and anti-failure features.
- NEC semiconductor products are classified into the following three quality grades:
  "Standard", "Special" and "Specific". The "Specific" quality grade applies only to semiconductor products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of a semiconductor product depend on its quality grade, as indicated below. Customers must check the quality grade of each semiconductor product before using it in a particular application.

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

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

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

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

  (Note)

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

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

M8E 00.4

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC product in your application, pIease 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.l.
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.
Madrid Office
Madrid, Spain
Tel: 91-504-2787
Fax: 91-504-2860

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
Tel: 65-253-8311
Fax: 65-250-3583

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

NEC do Brasil S.A.
Electron Devices Division
Guarulhos-SP Brasil
Tel: 55-11-6462-6810
Fax: 55-11-6462-6829

J00.7

[MEMO]

PREFACE

Target Readers

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

The μPD77016 Family comprises the μPD77015, 77016, 77017, 77018, 77018A, 77019, 77110, 77111, 77112, 77113, 77114, and 77116[Note]. However, note that the μPD77015 is not among the target devices in this user's manual.

Note Under development

Purpose

This user's manual is intended to give users an understanding of the supported middleware available for the design and development of application systems using the μPD77016 Family.

Organization

This user's manual describes the basic numerical operation program, etc.

CHAPTER 1 GENERAL
CHAPTER 2 LIBRARY SPECIFICATIONS
CHAPTER 3 INSTALLATION
APPENDIX SAMPLE SOURCE

How to Use This Manual

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

To learn about the hardware functions of the μPD77016 Family:
→ Refer to μPD7701x Family User's Manual Architecture.

To learn about the instruction functions of the μPD77016 Family:
→ Refer to the μPD7701x Family User's Manual Instructions.

Conventions

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

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

Documents Related to Devices

Document Name / Part Number	Pamphlet	Data Sheet	User's Manual		Application Note
			Architecture	Instructions	Basic Software
μPD77016	U12395E	U10891E	U10503E	U13116E	U11958E
μPD77015		U10902E			
μPD77017					
μPD77018					
μPD77018A		U11849E			
μPD77019					
μPD77019-013		U13053E			
μPD77110		U12801E	Under preparation		
μPD77111					
μPD77112					
μPD77113		Under preparation			
μPD77114					

Documents Related to Development Tools

Document Name		Document No.
IE-77016-98/PC User's Manual	Hardware	EEU-1541
IE-77016-CM-LC User's Manual		U14139E
RX77016 User's Manual	Function	U14397E
	Configuration Tool	U14404E
RX77016 Application Note	HOST API Interface	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

[MEMO]

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.

The μSAP77016-B03 is middleware that provides speech data compression and decompression which supports ITU-T[Note] Recommendation G.729 (including Annex A and Annex B). The descriptions in this manual, unless specified otherwise, assume use of G.729 speech codec.

Note International Telecommunication Union - Telecommunication Standardization Sector

1.2 G.729 Speech Codec

The G.729 speech codec is an 8-Kbps speech compression and decompression codec recommended by ITU-T that consists of an algorithm for encoding speech signals using CS-ACELP[Note].

G.729 speech codec is designed to perform 8-kHz sampling of the analog input signal, which is band limited with a telephone band filter (ITU-T Recommendation G.712), and to use the digital signal obtained by converting this data to 16-bit linear PCM data as the encoder input. Similarly, the decoder output must be returned to an analog signal.

Other I/O format signals such as 64-Kbps PCM data prescribed by ITU-T Recommendation G.711 must be converted to a suitable format either to a 16-bit linear PCM before the encoder or from a 16-bit linear PCM after the decoder. The bit string passed from the encoder to the decoder is prescribed by ITU-T Recommendation G.729.

Note Conjugate Structure-Algebraic Code Excited Linear Prediction

1.3 G.729 Annex A Speech Codec

ITU-T Recommendation G.729 Annex A is the low-calculation-capacity version of the G.729 speech codec, and this codec can be connected with the standard G.729 speech codec. The low-calculation-capacity speech codec was developed for multimedia applications that use both speech and data at the same time, but it is not limited to such applications in particular.

1.4 G.729 Annex B Speech Codec

ITU-T Recommendation G.729 Annex B is the silence compression function of G.729 speech codec, and it is added to the standard G.729 speech codec or Annex A. Speech codec with an added silence compression function cannot be connected to a speech codec without an added silence compression function. This silence compression function is designed to lower the bit rate by raising the compression rate of silent parts.

1.5 System Overview

1.5.1 Features

- 8-Kbps compression and encoding
- High-quality speech encoding (quality equivalent to 32-Kbps ADPCM)
- Encoding/decoding of 80 samples as 1 frame at sampling frequency of 8 kHz
- Speech I/O data all 16-bit linear PCM data

1.5.2 Operating environment

(1) Target DSPs

μPD77016, 77017[Note 1], 77018, 77018A, 77019, 77110, 77111, 77112, 77113, 77114, and 77116[Note 2]

Notes 1. If Annex B is added, μPD77017 is excluded from the target DSPs.

2. Under development

Caution The μPD77015 is excluded from the target devices.

(2) Required memory

Table 1-1. Required Memory Capacities

Memory [word] \ Type		G.729	Annex A	Annex B	Annex AB[Note]	Annex B
		Single-channel version				Multichannel version
Instruction memory	–	8.8 K	7.7 K	12.2 K	11.2 K	7.5 K
X memory	RAM	1.8 K	1.7 K	1.9 K	1.8 K	942 × N + 1153
	ROM	3.1 K	2.8 K	3.1 K	2.8 K	2.6 K
Y memory	RAM	1.9 K	1.7 K	2.0 K	1.8 K	914 × N + 1388
	ROM	2.0 K	1.8 K	2.4 K	2.2 K	2.0 K

Note Annex AB: Annex A + Annex B

Remark N: Number of channels

(3) Software tool (Windows™ version)

DSP tools: Workbench WB77016 Ver. 2.32
High-speed simulator HSM77016 Ver. 2.3

1.5.3 Performance

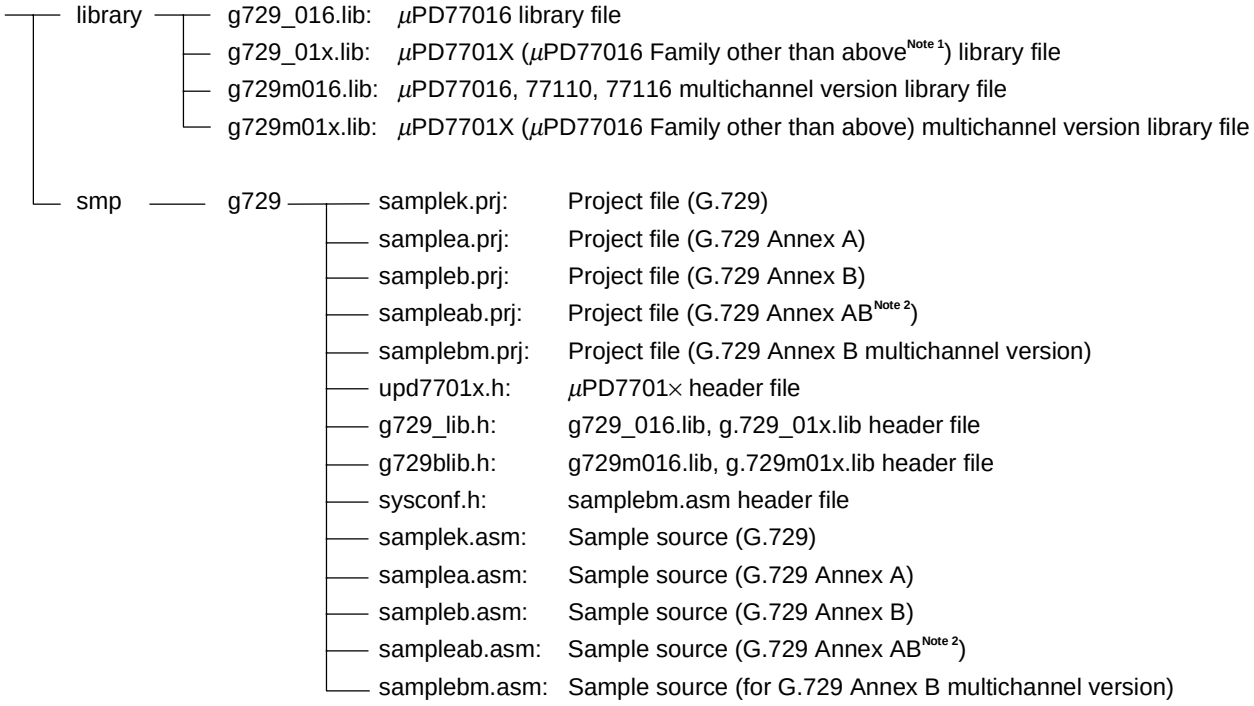
[Condition] DSP: μPD77016 Family (Operating frequency: 33 MHz, 33 MIPS)

[MIPS value required for processing 1 frame in real time (10 ms)]

(G.729)	Compression:	15.7 MIPS
	Decompression:	3.3 MIPS
(G.729 Annex A)	Compression:	9.0 MIPS
	Decompression:	2.2 MIPS
(G.729 Annex B)	Compression:	16.5 MIPS
	Decompression:	3.5 MIPS
(G.729 Annex A + Annex B)	Compression:	9.5 MIPS
	Decompression:	3.7 MIPS
(G.729 Annex B multichannel version)	Compression:	13.9 × N MIPS (N: Number of channels)
	Decompression:	2.8 × N MIPS (N: Number of channels)

1.5.4 Directory structure

The μSAP77016-B03 directory structure is as follows.



Notes 1. The μPD77016, 77110, and 77116 are not target DSPs. g729_01x.lib is the file for the μPD77017, 77018, 77018A, 77019, 77111, 77112, 77113, and 77114.

2. Annex AB: Annex A + Annex B

Library files g729_016.lib and g729_01x.lib are for the single-channel version. They include the following object files.

- g729kenc.rel (G.729 encoder)
- g729kdec.rel (G.729 decoder)
- g729aenc.rel (G.729 Annex A encoder)
- g729adec.rel (G.729 Annex A decoder)
- g729benc.rel (G.729 Annex B encoder)
- g729bdec.rel (G.729 Annex B decoder)
- g729abenc.rel (G.729 Annex A + Annex B encoder)
- g729abdec.rel (G.729 Annex A + Annex B decoder)
- g729getv.rel (Get version function)

Library files g729m016.lib and g729m01x.lib are for the multichannel version. They include the following object files.

- g729bmen.rel (G.729 Annex B multichannel version encoder)
- g729bmde.rel (G.729 Annex B multichannel version decoder)
- g729bmco.rel (G.729 Annex B multichannel version common part)

Caution g729m016.lib is a file for the μPD77016, 77110, and 77116. g729bm01x.lib is a file for the μPD77018, 77018A, 77019, 77111, 77112, 77113, and 77114.

[MEMO]

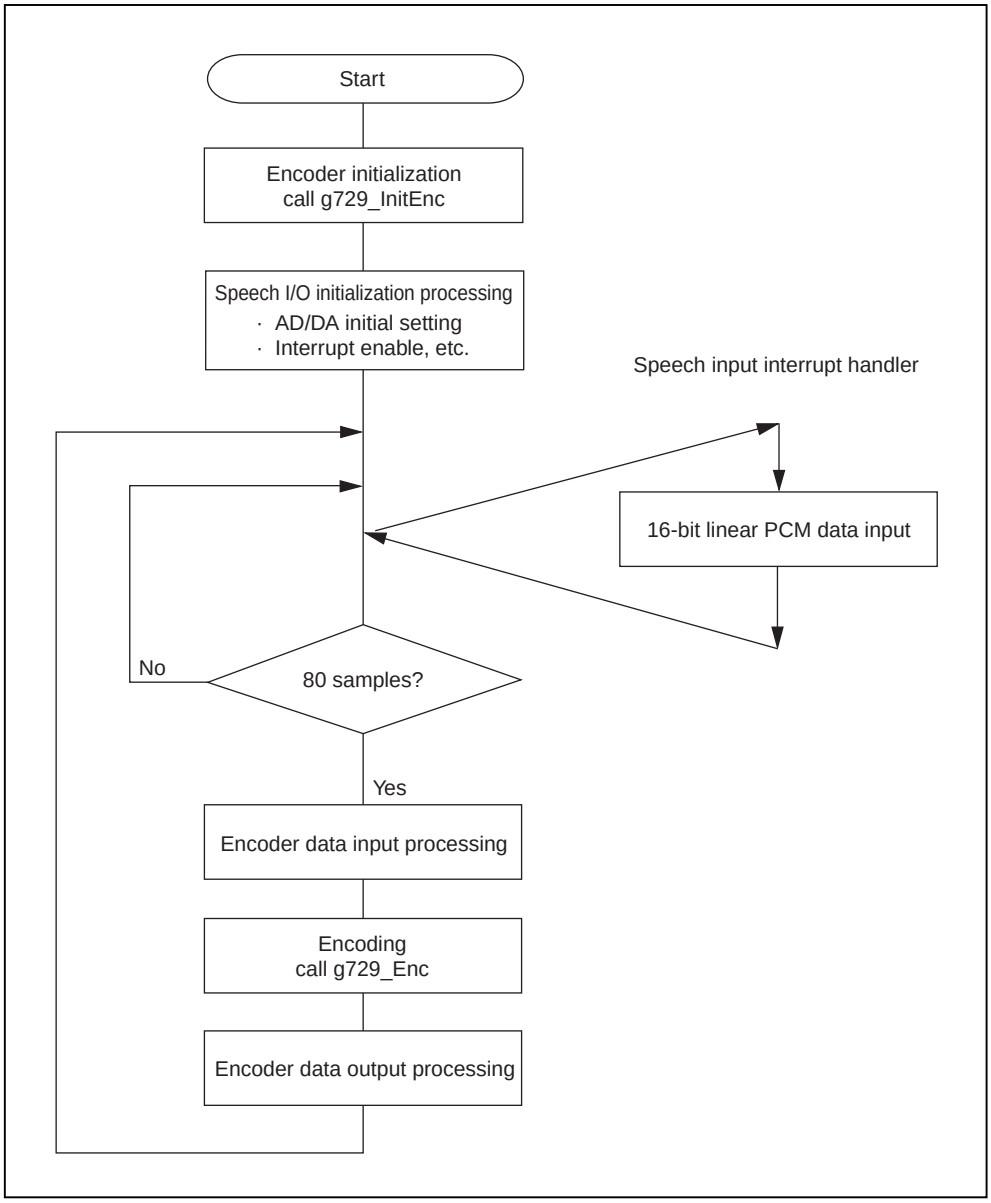
CHAPTER 2 LIBRARY SPECIFICATIONS

This chapter describes the G.729 speech codec function specifications and call rules.

2.1 G.729 Speech Codec Processing Flow

The processing flow of applications using the G.729 speech codec is shown in Figures 2-1 and 2-2.

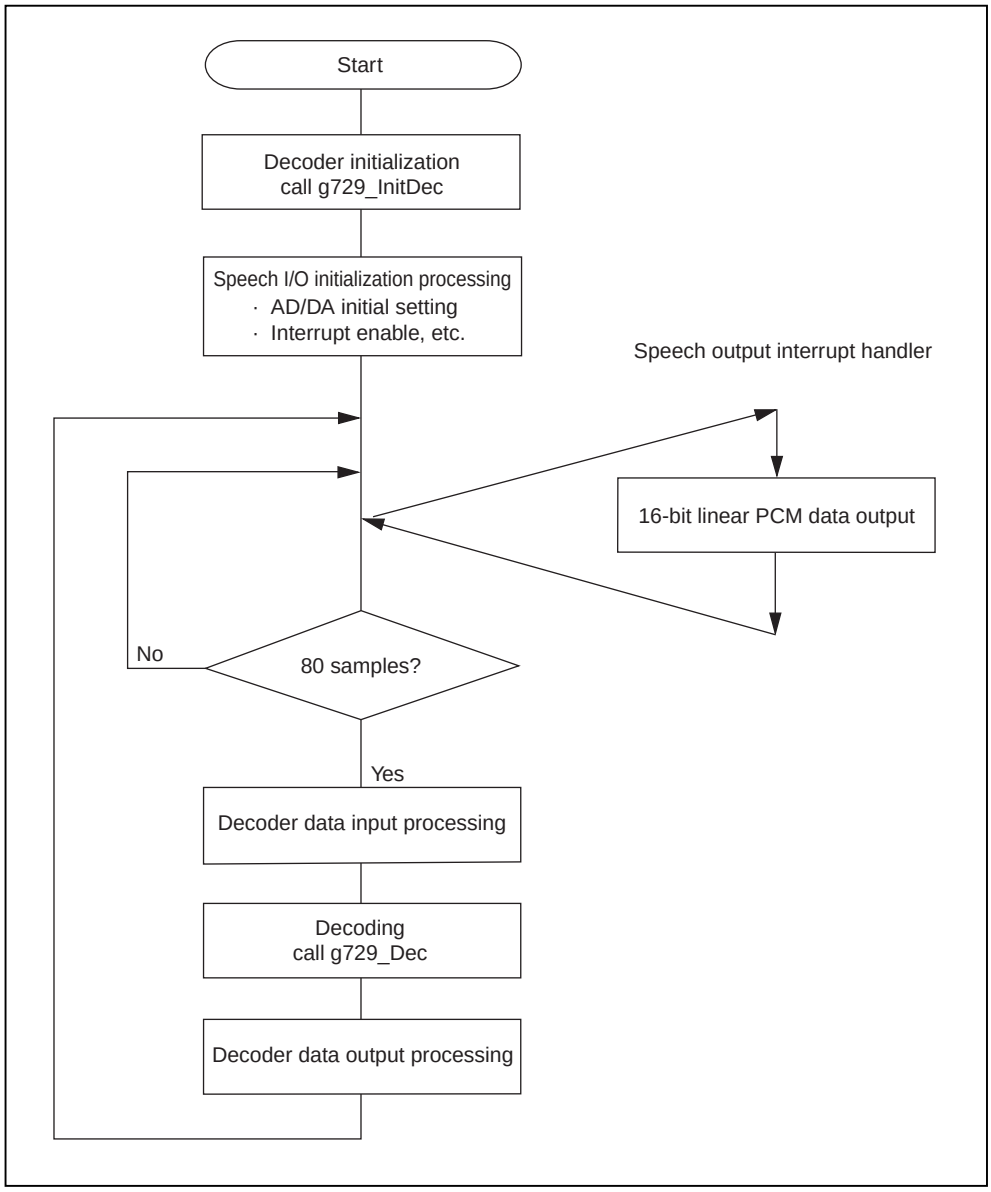
Figure 2-1. Application Processing Flow (Encoder)



Remark In the case of G.729 Annex A, Annex B, and Annex A + Annex B, the “g729_” part of the G.729 function name is “g729a_, “g729b_”, and “g729ab_”, respectively.

Moreover, in the case of the G.729 Annex B multichannel version, call “g729b_Start Codec” immediately after start, “g729b_InitEncM” in the “g729_InitEnc” part, and “g729b_EncM” in the “g729_Enc” part.

Figure 2-2. Application Processing Flow (Decoder)



Remark In the case of G.729 Annex A, Annex B, and Annex A + Annex B, the “g729_” part of the G.729 function name is “g729a_, “g729b_”, and “g729ab_”, respectively.

Moreover, in the case of the G.729 Annex B multichannel version, call “g729b_Start Codec” immediately after start, “g729b_InitDecM” in the “g729_InitDec” part, and “g729b_DecM” in the “g729_Dec” part.

2.2 Function Specifications (Single-Channel Version)

2.2.1 Initialize encoder function

The g729_InitEnc (g729a_InitEnc, g729b_InitEnc, g729ab_InitEnc) function is used to set encoder constants and perform initialization of coefficient tables and delay buffer.

(1) G.729 initialize encoder function

[Classification]	Encoder initialization processing
[Function name]	g729_InitEnc
[Summary of function]	Initializes the RAM area used by the G.729 encoder and sets parameters.
[Format]	call g729_InitEnc
[Arguments]	None
[Return value]	None
[Function]	Initializes the G.729 encoder and sets parameters.
[Registers used]	R0, DP0, DP1, DP2, DP3, DP4, DP7, DN0, DN2, DMX
[Hardware resources]	Maximum stack level: 2 Maximum loop stack level: 1 Maximum number of repeats: 240 Maximum number of cycles: 842

(2) G.729 Annex A initialize encoder function

[Classification]	Encoder initialization processing
[Function name]	g729a_InitEnc
[Summary of function]	Initializes the RAM area used by the G.729 Annex A encoder and sets parameters.
[Format]	call g729a_InitEnc
[Arguments]	None
[Return value]	None
[Function]	Initializes the G.729 Annex A encoder and sets parameters.
[Registers used]	R0, DP0, DP1, DP2, DP3, DP4, DP7, DN0, DN2, DMX
[Hardware resources]	Maximum stack level: 2 Maximum loop stack level: 1 Maximum number of repeats: 240 Maximum number of cycles: 860

(3) G.729 Annex B initialize encoder function

[Classification]	Encoder initialization processing
[Function name]	g729b_InitEnc
[Summary of function]	Initializes the RAM area used by the G.729 Annex B encoder and sets parameters.
[Format]	call g729b_InitEnc
[Arguments]	None
[Return value]	None
[Function]	Initializes the G.729 Annex B encoder and sets parameters.
[Registers used]	R0, R1, R2, DP0, DP1, DP2, DP3, DP4, DP7, DN0, DN2, DMX
[Hardware resources]	Maximum stack level: 2 Maximum loop stack level: 1 Maximum number of repeats: 240 Maximum number of cycles: 1177

(4) G.729 Annex A + Annex B initialize encoder function

[Classification]	Encoder initialization processing
[Function name]	g729ab_InitEnc
[Summary of function]	Initializes the RAM area used by the G.729 Annex A + Annex B encoder and sets parameters.
[Format]	call g729ab_InitEnc
[Arguments]	None
[Return value]	None
[Function]	Initializes the G.729 Annex A + Annex B encoder and sets parameters.
[Registers used]	R0, R1, R2, DP0, DP1, DP2, DP3, DP4, DP7, DN0, DN2, DMX
[Hardware resources]	Maximum stack level: 2 Maximum loop stack level: 1 Maximum number of repeats: 240 Maximum number of cycles: 1195

2.2.2 Initialize decoder function

The g729_InitDec (g729a_InitDec, g729b_InitDec, g729ab_InitDec) function is used to set decoder constants and perform initialization of coefficient tables and delay buffer.

(1) G.729 initialize decoder function

[Classification]	Decoder initialization processing
[Function name]	g729_InitDec
[Summary of function]	Initializes the RAM area used by the G.729 decoder and sets parameters.
[Format]	call g729_InitDec
[Arguments]	None
[Return value]	None
[Function]	Initializes the G.729 decoder and sets parameters.
[Registers used]	R0, R1, R2, DP0, DP1, DP2, DP4, DP7, DN0, DN2, DMX
[Hardware resources]	Maximum stack level: 2 Maximum loop stack level: 1 Maximum number of repeats: 154 Maximum number of cycles: 549

(2) G.729 Annex A initialize decoder function

[Classification]	Decoder initialization processing
[Function name]	g729a_InitDec
[Summary of function]	Initializes the RAM area used by the G.729 Annex A decoder and sets parameters.
[Format]	call g729a_InitDec
[Arguments]	None
[Return value]	None
[Function]	Initializes the G.729 Annex A decoder and sets parameters.
[Registers used]	R0, R1, R2, DP0, DP1, DP2, DP4, DP7, DN0, DN2, DMX
[Hardware resources]	Maximum stack level: 2 Maximum loop stack level: 1 Maximum number of repeats: 183 Maximum number of cycles: 939

(3) G.729 Annex B initialize decoder function

[Classification]	Decoder initialization processing
[Function name]	g729b_InitDec
[Summary of function]	Initializes the RAM area used by the G.729 Annex B decoder and sets parameters.
[Format]	call g729b_InitDec
[Arguments]	None
[Return value]	None
[Function]	Initializes the G.729 Annex B decoder and sets parameters.
[Registers used]	R0, R1, R2, DP0, DP1, DP2, DP4, DP7, DN0, DN2, DMX
[Hardware resources]	Maximum stack level: 2 Maximum loop stack level: 1 Maximum number of repeats: 154 Maximum number of cycles: 760

(4) G.729 Annex A + Annex B initialize decoder function

[Classification]	Decoder initialization processing
[Function name]	g729ab_InitDec
[Summary of function]	Initializes the RAM area used by the G.729 Annex A + Annex B decoder and sets parameters.
[Format]	call g729ab_InitDec
[Arguments]	None
[Return value]	None
[Function]	Initializes the G.729 Annex A + Annex B decoder and sets parameters.
[Registers used]	R0, R1, R2, DP0, DP1, DP2, DP4, DP7, DN0, DN2, DMX
[Hardware resources]	Maximum stack level: 2 Maximum loop stack level: 1 Maximum number of repeats: 183 Maximum number of cycles: 1120

2.2.3 Encoder function

The encoder functions generate an 80-bit or 16-bit signal by compressing 80 input samples of speech signal.

(1) G.729 encoder function

[Classification]	Encode processing block	
[Function name]	g729_Enc	
[Summary of function]	Compresses 80 samples × 16 bits into 80 bits.	
[Format]	call g729_Enc	
[Arguments]	G729E_PCM_BUF[80](X)	Input data
	*G729E_CMD_STS:y	Control/status register
[Return value]	G729E_ANA_BUF[5](Y)	Compressed signal
[Function]	Compresses the input signal from codec (80 samples × 16 bits) into 80 bits.	
[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	
[Hardware resources]	Maximum stack level:	4
	Maximum loop stack level:	2
	Maximum number of repeats:	240
	Maximum MIPS value:	15.7

(2) G.729 Annex A encoder function

[Classification]	Encode processing block	
[Function name]	g729a_Enc	
[Summary of function]	Compresses 80 samples × 16 bits into 80 bits.	
[Format]	call g729a_Enc	
[Arguments]	G729AE_PCM_BUF[80](X)	Input data
	*G729AE_CMD_STS:y	Control/status register
[Return value]	G729AE_ANA_BUF[5](Y)	Compressed signal
[Function]	Compresses the input signal from codec (80 samples × 16 bits) into 80 bits.	
[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	
[Hardware resources]	Maximum stack level:	4
	Maximum loop stack level:	2
	Maximum number of repeats:	240
	Maximum MIPS value:	9.0

(3) G.729 Annex B encoder function

[Classification]	Encode processing block	
[Function name]	g729b_Enc	
[Summary of function]	Compresses 80 samples × 16 bits into 80 bits or 16 bits.	
[Format]	call g729b_Enc	
[Arguments]	G729BE_PCM_BUF[80](X)	Input data
	*G729BE_CMD_STS:y	Control/status register
	*G729Bframe:y	Frame counter (0 to 0x7fff) (Reset to 0x100 after 0x7fff)
[Return value]	G729BE_ANA_BUF[5](Y)	Compressed signal
[Function]	Compresses the signal input from codec (80 samples × 16 bits) to 80 bits or 16 bits.	
[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	
[Hardware resources]	Maximum stack level:	6
	Maximum loop stack level:	3
	Maximum number of repeats:	240
	Maximum MIPS value:	16.5

(4) G.729 Annex A + Annex B encoder function

[Classification]	Encode processing block	
[Function name]	g729ab_Enc	
[Summary of function]	Compresses 80 samples × 16 bits into 80 bits or 16 bits.	
[Format]	call g729ab_Enc	
[Arguments]	G729ABE_PCM_BUF[80](X)	Input data
	*G729ABE_CMD_STS:y	Control/status register
	*G729ABframe:y	Frame counter (0 to 0x7fff) (Reset to 0x100 after 0x7fff)
[Return value]	G729ABE_ANA_BUF[5](Y)	Compressed signal
[Function]	Compresses the signal input from codec (80 samples × 16 bits) to 80 bits or 16 bits.	
[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	
[Hardware resources]	Maximum stack level:	6
	Maximum loop stack level:	3
	Maximum number of repeats:	240
	Maximum MIPS value:	9.5

2.2.4 Decoder function

The decoder function decompresses data compressed to 80 bits or 16 bits into speech data of 80 samples × 16 bits.

(1) G.729 decoder function

[Classification]	Decode processing block	
[Function name]	g729_Dec	
[Summary of function]	Decompresses 80 bits into 80 samples × 16 bits.	
[Format]	call g729_Dec	
[Arguments]	G729D_ANA_BUF[5](Y)	Input data
	*G729D_CMD_STS:y	Control/status register
[Return value]	G729D_PCM_BUF[80](X)	Decompressed data
[Function]	Decompresses data compressed to 80 bits into speech signal (80 samples × 16 bits).	
[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	
[Hardware resources]	Maximum stack level:	4
	Maximum loop stack level:	2
	Maximum number of repeats:	40
	Maximum MIPS value:	3.3

(2) G.729 Annex A decoder function

[Classification]	Decode processing block	
[Function name]	g729a_Dec	
[Summary of function]	Decompresses 80 bits into 80 samples × 16 bits.	
[Format]	call g729a_Dec	
[Arguments]	G729AD_ANA_BUF[5](Y)	Input data
	*G729AD_CMD_STS:y	Control/status register
[Return value]	G729AD_PCM_BUF[80](X)	Decompressed data
[Function]	Decompresses data compressed to 80 bits into speech signal (80 samples × 16 bits).	
[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	
[Hardware resources]	Maximum stack level:	4
	Maximum loop stack level:	2
	Maximum number of repeats:	40
	Maximum MIPS value:	2.2

(3) G.729 Annex B decoder function

[Classification]	Decode processing block	
[Function name]	g729b_Dec	
[Summary of function]	Decompresses 80 bits or 16 bits into 80 samples × 16 bits.	
[Format]	call g729b_Dec	
[Arguments]	G729BD_ANA_BUF[5](Y)	Input data
	*G729BD_CMD_STS:y	Control/status register
[Return value]	G729BD_PCM_BUF[80](X)	Decompressed data
[Function]	Decompresses data compressed to 80 bits or 16 bits into speech signal (80 samples × 16 bits).	
[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	
[Hardware resources]	Maximum stack level:	6
	Maximum loop stack level:	3
	Maximum number of repeats:	40
	Maximum MIPS value:	3.5

(4) G.729 Annex A + Annex B decoder function

[Classification]	Decode processing block	
[Function name]	g729ab_Dec	
[Summary of function]	Decompresses 80 bits or 16 bits into 80 samples × 16 bits.	
[Format]	call g729ab_Dec	
[Arguments]	G729ABD_ANA_BUF[5](Y)	Input data
	*G729ABD_CMD_STS:y	Control/status register
[Return value]	G729ABD_PCM_BUF[80](X)	Decompressed data
[Function]	Decompresses data compressed to 80 bits or 16 bits into speech signal (80 samples × 16 bits).	
[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	
[Hardware resources]	Maximum stack level:	4
	Maximum loop stack level:	3
	Maximum number of repeats:	40
	Maximum MIPS value:	3.7

2.2.5 Get version function

The get version function returns the library version.

[Classification]	Version information acquisition
[Function name]	g729_GetVersion
[Summary of function]	Returns the library version.
[Format]	call g729_GetVersion
[Arguments]	None
[Return value]	R0H Major version number R0L Minor version number
[Function]	Returns the version number of the G.729 speech codec library as a 32-bit value. Example: When R0 = 0x00'0x0001'0x0100 Version: V1.01
[Registers used]	R0
[Hardware resources]	Maximum stack level: 0 Maximum loop stack level: 0 Maximum number of repeats: 0 Maximum number of cycles: 6

2.3 Function Specifications (Multichannel Version)

2.3.1 Initialize scratch area function

The initialize scratch area function performs initialization of the scratch area of this middleware corresponding to multichannel specifications.

(1) Initialize scratch area function

[Classification]	Scratch area initialization processing	
[Function name]	g729b_StartCodec	
[Summary of function]	Clears to 0 the scratch area used by the G.729 Annex B multichannel version.	
[Format]	call G729b_Start_Codec	
[Arguments]	None	
[Return value]	None	
[Function]	Clears to 0 the scratch area used by the G.729 Annex B multichannel version.	
[Registers used]	R0, DP0, DP4, DP5	
[Hardware resources]	Maximum stack level:	1
	Maximum loop stack level:	1
	Maximum number of repeats:	597
	Maximum number of cycles:	1072

2.3.2 Initialize encoder function

The g729b_InitEncM function is used to set encoder constants and perform initialization of coefficient tables and delay buffer.

(1) G.729 Annex B multichannel version initialize encoder function

[Classification] Encoder initialization processing

[Function name] g729b_InitEncM

[Summary of function] Performs initialization of the G.729 Annex B multichannel version encoder and sets parameters.

[Format] call g729b_InitEncM

[Arguments] I/O buffer start address and parameters

Example:
```
/*---*/       dp0=g729b_IO_Table  ;
        clr(r1)                         ;
        r01=STATIC_X1                   ;
        /*---*/       *dp0++=r01          ;
        r01-STATIC_Y1                   ;
        /*---*/       *dp0++=r01          ;
        r01=EncPcmData1                 ;
        /*---*/       *dp0++=r01          ;
        r01=EncPrmData1                 ;
        /*---*/       *dp0++=r01          ;
        rep 4                           ;
            /*...*/   *dp0++=r1h          ;
        r01=DecPcmData1                 ;
        /*---*/       *dp0++=r01          ;
        r01=DecPrmData1                 ;
        /*---*/       *dp0++=r01          ;
        r01= 1                          ;
            /*   */   *dp0++=r01          ; fParityCheck = 1
        rep 5                           ;
            /*...*/   *dp0++=r1h          ;
        /*   */       *dp0++=r01          ; fVadEnable = 1
        rep 3                           ;
            /*...*/   *dp0++=r1h          ;
        /*---*/       dp0=g729b_IO_Table  ;
```

[Return value] None

[Function] Performs initialization of the G.729 Annex B multichannel version encoder and sets parameters.

[Registers used] R0, R1, DP0, DP1, DP2, DP3, DP4, DP5, DP6, DP7, DN0

[Hardware resources]
Maximum stack level: 2
Maximum loop stack level: 1
Maximum number of repeats: 240
Maximum number of cycles: 2642

2.3.3 Initialize decoder function

The g729b_InitDecM function is used to set decoder constants and perform initialization of coefficient tables and delay buffer.

(1) G.729 Annex B multichannel version initialize decoder function

[Classification] Decoder initialization processing

[Function name] g729b_InitDecM

[Summary of function] Initializes the G.729 Annex B multichannel version decoder and sets parameters.

[Format] call g729b_InitDecM

[Arguments] I/O buffer start address and parameters

Example:
```
/*---*/       dp0=g729b_IO_Table  ;
clr(r1)                         ;
r01=STATIC_X1                   ;
/*---*/       *dp0++=r01          ;
r01-STATIC_Y1                   ;
/*---*/       *dp0++=r01          ;
r01=EncPcmData1                 ;
/*---*/       *dp0++=r01          ;
r01=EncPrmData1                 ;
/*---*/       *dp0++=r01          ;
rep 4                           ;
    /*...*/   *dp0++=r1h          ;
r01=DecPcmData1                 ;
/*---*/       *dp0++=r01          ;
r01=DecPrmData1                 ;
/*---*/       *dp0++=r01          ;
r01= 1                          ;
    /*   */   *dp0++=r01          ; fParityCheck = 1
rep 5                           ;
    /*...*/   *dp0++=r1h          ;
/*   */       *dp0++=r01          ; fVadEnable = 1
rep 3                           ;
    /*...*/   *dp0++=r1h          ;
/*---*/       dp0=g729b_IO_Table  ;
```

[Return value] None

[Function] Performs initialization of the G.729 Annex B multichannel version decoder and sets parameters.

[Registers used] R0, R1, DP0, DP1, DP2, DP3, DP4, DP5, DP6, DN0

[Hardware resources]
Maximum stack level: 2
Maximum loop stack level: 1
Maximum number of repeats: 234
Maximum number of cycles: 1741

2.3.4 Encoder function

The encoder function is used to generate an 80-bit or 16-bit signal by compressing 80 input samples of speech signal.

(1) G.729 Annex B multichannel version encoder function

[Classification] Encode processing block

[Function name] g729b_EncM

[Summary of function] Compresses 80 samples × 16 bits into 80 bits or 16 bits.

[Format] call g729b_EncM

[Arguments] I/O buffer start address and parameters

Example:
```
/*---*/       dp0=g729b_IO_Table  ;
clr(r1)                         ;
r01=STATIC_X1                   ;
/*---*/       *dp0++=r01          ;
r01-STATIC_Y1                   ;
/*---*/       *dp0++=r01          ;
r01=EncPcmData1                 ;
/*---*/       *dp0++=r01          ;
r01=EncPrmData1                 ;
/*---*/       *dp0++=r01          ;
rep 4                           ;
    /*...*/   *dp0++=r1h          ;
r01=DecPcmData1                 ;
/*---*/       *dp0++=r01          ;
r01=DecPrmData1                 ;
/*---*/       *dp0++=r01          ;
r01= 1                          ;
    /*   */   *dp0++=r01          ; fParityCheck = 1
rep 5                           ;
    /*...*/   *dp0++=r1h          ;
/*   */       *dp0++=r01          ; fVadEnable = 1
rep 3                           ;
    /*...*/   *dp0++=r1h          ;
/*---*/       dp0=g729b_IO_Table  ;
```

[Return value] Compression data: Output buffer specified by argument

[Function] Compresses the input signal from codec (80 samples × 16 bits) to 80 bits or 16 bits.

[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

[Hardware resources]
Maximum stack level: 7
Maximum loop stack level: 1
Maximum number of repeats: 223
Maximum MIPS value: 13.9

2.3.5 Decoder function

The decoder function decompresses a signal compressed to 80 bits or 16 bits to speech data of 80 × 16 bits.

(1) G.729 Annex B multichannel version decoder function

[Classification] Decode processing block

[Function name] g729b_DecM

[Summary of function] Decompresses 80 bits or 16 bits into 80 samples × 16 bits.

[Format] call g729b_DecM

[Arguments] I/O buffer start address and parameters

Example:
```
/*---*/       dp0=g729b_IO_Table  ;
clr(r1)                         ;
r01=STATIC_X1                   ;
/*---*/       *dp0++=r01          ;
r01-STATIC_Y1                   ;
/*---*/       *dp0++=r01          ;
r01=EncPcmData1                 ;
/*---*/       *dp0++=r01          ;
r01=EncPrmData1                 ;
/*---*/       *dp0++=r01          ;
rep 4                           ;
    /*...*/   *dp0++=r1h          ;
r01=DecPcmData1                 ;
/*---*/       *dp0++=r01          ;
r01=DecPrmData1                 ;
/*---*/       *dp0++=r01          ;
r01= 1                          ;
    /*   */   *dp0++=r01          ; fParityCheck = 1
rep 5                           ;
    /*...*/   *dp0++=r1h          ;
/*   */       *dp0++=r01          ; fVadEnable = 1
rep 3                           ;
    /*...*/   *dp0++=r1h          ;
/*---*/       dp0=g729b_IO_Table  ;
```

[Return value] Compression data: Output buffer specified by argument

[Function] Decompresses 80 bits or 16 bits into 80 samples × 16 bits.

[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

[Hardware resources]
Maximum stack level: 4
Maximum loop stack level: 1
Maximum number of repeats: 203
Maximum MIPS value: 2.8

2.4 Control/Status Registers (Single-Channel Version)

G729E_CMD_STS (G729AE_CMD_STS, G729BE_CMD_STS, G729ABE_CMD_STS) is the control/status register for the G.729 encoder, and G729D_CMD_STS (G729AD_CMD_STS, G729BD_CMD_STS, G729ABD_CMD_STS) is the control/status register for the decoder. The higher 8 bits are a status word and are read-only. The lower 8 bits are a control word and are read/write-enabled.

2.4.1 G729E_CMD_STS (G729AE_CMD_STS, G729BE_CMD_STS, G729ABE_CMD_STS) register

The G729E_CMD_STS (G729AE_CMD_STS, G729BE_CMD_STS, G729ABE_CMD_STS) register is a memory mapped register that performs encoder operation control and status display. Changes made prior to the g729_Enc (g729a_Enc, g729b_Enc, g729ab_Enc) function call are effective.

However, bits D9, D8, and D5 support Annex B, so if using G.729 and G.729 Annex A, set (D9, D8, D5) = (0, 1, 0).

Table 2-1. G729E_CMD_STS Register

Bit	Name	Initial Value	0	1	Remark
D15	Busy Flag	–	Not started up	Started up	Encoder Indicates whether or not encoder (g729_Enc/g729a_Enc/g729b_Enc/g729ab_Enc) is in started up status.
D14 to D10	Reserved	–	–	–	–
D9	Ftyp1	–	**Note**	**Note**	–
D8	Ftyp0	–	**Note**	**Note**	–
D7	Operation	–	Don't process	Process	Determines whether or not to perform encode processing.
D6	Test Mode	–	Normal	Test	Inputs test data located in ROM to encoder.
D5	VAD/DTX	–	Don't process	Process	Specifies ON/OFF for silence compression function.
D4	Muting	–	Normal	Silence	Inputs silent data (0) to encoder.
D3 to D0	Reserved	–	–	–	–

Note Ftyp(1,0) = 00: Untransmitted frame
01: Active Speech frame
10: SID frame
11: N/A

2.4.2 G729D_CMD_STS (G729AD_CMD_STS, G729BD_CMD_STS, G729ABD_CMD_STS) register

The G729D_CMD_STS (G729AD_CMD_STS, G729BD_CMD_STS, G729ABD_CMD_STS) register is a memory mapped register that performs decoder operation control and status display. Changes made prior to the g729_Dec (g729a_Dec, g729b_Dec, g729ab_Dec) function call are effective.

However, bits D5, D1, and D0 support Annex B, so if using G.729 and G.729 Annex A, set (D5, D1, D0) = (0, 0, 1).

Table 2-2. G729D_CMD_STS Register

Bit	Name	Initial Value	0	1	Remark
D15	Busy Flag	–	Not started up	Started up	Indicates whether or not decoder (g729_Dec/g729a_Dec/g729b_Dec/g729ab_Dec) is in started up status.
D14	Parity Error	–	No error	Error	Displays parity check result.
D13 to D8	Reserved	–	–	–	–
D7	Operation	–	Don't process	Process	Determines whether or not to perform decode processing.
D6	Test Mode	–	Normal	Test	Inputs test data located in ROM to decoder.
D5	DTX/CNG	–	Don't process	Process	Specifies ON/OFF for silence compression function.
D4	Muting	–	Normal	Silence	Outputs silent data (0) from decoder.
D3	Frame Eras.	–	Don't process	Process	Determines whether or not to perform processing for erase frames.
D2	Parity Check	–	No check	Check	Determines whether or not to perform parity check.
D1	Ftyp1	–	**Note**	**Note**	–
D0	Ftyp0	–	**Note**	**Note**	–

Note Ftyp(1,0) = 00: Untransmitted frame
01: Active Speech frame
10: SID frame
11: N/A

2.5 Multichannel Version External Interface

This section describes the parameters passed when calling the G729b_EncM, g729_DecM functions for the G.729 Annex B multichannel version.

2.5.1 Function transfer parameters for G.729 Annex B multichannel version

Table 2-3 shows the contents of the tables.

Table 2-3. Function Transfer Parameters for the G.729 Annex B Multichannel Version

Offset	Description	Access Timing		
		Initialization	Encode	Decode
0x00	Static variable area x address (Don't change after initialization processing)	C	C	C
0x01	Static variable area y address (Don't change after initialization processing)	C	C	C
0x02	Address of PCM data input buffer for encoding		C	
0x03	Address of encoding data output buffer for encoding		C	
0x04	Encoding frame count Setting value: 0x0001 to 0x7fff at the beginning only (+1 for each frame), then 0x0100 to 0x7fff (+1 for each frame) repeatedly		C	
0x05 to 0x07	Reserved			
0x08	Address of PCM data output buffer for decoding			C
0x09	Address of encoding data input buffer for decoding			C
0x0a	Parity check specification (0: No parity check, 1: Parity check)			C
0x0b	Parity error (0: No parity error, 1: Parity error)			S
0x0c	Erase frame processing specification (0: Normal frame, 1: Erase)			C
0x0d to 0x0f	Reserved			
0x10	Specifies whether silent compression is used or not (0: not used, 1: used)		C	
0x11	Encoding output frame type		S	
0x12	Decoding input frame type			S
0x13	Reserved			

Caution Secure the scratch area in the following manner. 'align at 4' is required for scratch area x.
Scratch area x: 1153-word size at label name "lib_Scratch_x" in the X memory
Scratch area y: 1257-word size at label name "lib_Scratch_y" in the Y memory

Remark Access types C: Control information
S: Result status

2.6 Compressed Data Formats

2.6.1 Speech data compression format

The 80-bit data format used for encoder function output and decoder function input is shown in Tables 2-4 and 2-5. For more information about compression data, refer to ITU-T recommendations.

Table 2-4. Bit Allocation of Speech Compression Data

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
word0	L0	L1	L1	L1	L1	L1	L1	L1	L2	L2	L2	L2	L2	L3	L3	L3
word1	L3	L3	P1	P1	P1	P1	P1	P1	P1	P1	P0	C1	C1	C1	C1	C1
word2	C1	C1	C1	C1	C1	C1	C1	C1	S1	S1	S1	S1	GA1	GA1	GA1	GB1
word3	GB1	GB1	GB1	P2	P2	P2	P2	P2	C2	C2	C2	C2	C2	C2	C2	C2
word4	C2	C2	C2	C2	C2	S2	S2	S2	S2	GA2	GA2	GA2	GB2	GB2	GB2	GB2

Table 2-5. Meaning of Speech Compression Data

Symbol	Description	Number of Bits
L0	Switched MA predictor index of LSP quantizer	1
L1	1st stage vector of LSP quantizer	7
L2	2nd stage lower vector of LSP quantizer	5
L3	2nd stage higher vector of LSP quantizer	5
P1	Pitch delay 1st subframe	8
P0	Parity bit for pitch delay	1
C1	Fixed codebook 1st subframe	13
S1	Signs of fixed-codebook pulses 1st subframe	4
GA1	Gain codebook (stage 1) 1st subframe	3
GB1	Gain codebook (stage 2) 1st subframe	4
P2	Pitch delay 2nd subframe	5
C2	Fixed codebook 2nd subframe	13
S2	Signs of fixed-codebook pulses 2nd subframe	4
GA2	Gain codebook (stage 1) 2nd subframe	3
GB2	Gain codebook (stage 2) 2nd subframe	4

2.6.2 SID frame compression format

The 16-bit SID frame data format used for encoder function output and decoder function input is shown in Tables 2-6 and 2-7. For more information about compression data, refer to ITU-T recommendations.

Table 2-6. Bit Allocation of Silent Compression Data

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word	L0	L1	L1	L1	L1	L1	L2	L2	L2	L2	E	E	E	E	E	0

Table 2-7. Meaning of Silent Compression Data

Symbol	Description	Number of Bits
L0	Switched predictor index of LSF quantizer	1
L1	1st stage vector of LSF quantizer	5
L2	2nd stage vector of LSF quantizer	4
E	Gain (Energy)	5

CHAPTER 3 INSTALLATION

3.1 Installation Procedure

This middleware is supplied on a 3.5-inch floppy disk (1.44MB). The procedure for installing in the host machine is described below.

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

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

(2) Confirm that the files have been copied. Refer to **1.5.4 Directory structure** for details on the directories.

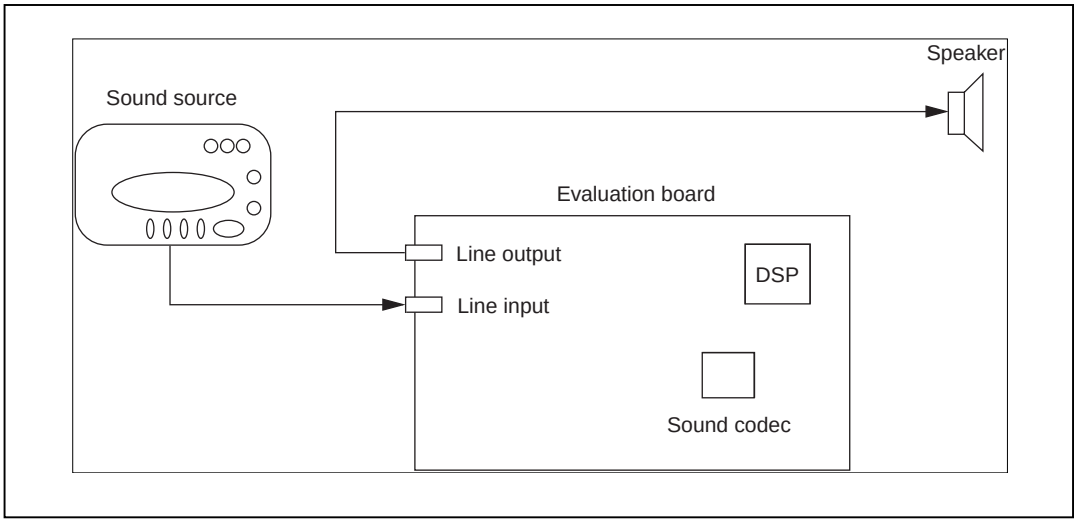
```
a:\>dir c:\DSPTools <CR>
```

3.2 Sample Creation Procedure

A sample program is stored to the sample directory (smp) (for the sample.asm source program, refer to **APPENDIX SAMPLE SOURCE**).

The sample program can be used to compress/decompress a sound source such as a CD or digital audio tape (DAT) in real time by connecting speakers, etc. The sample program operates on a μPD77016 speech codec evaluation board.

Figure 3-1. Sample Program Evaluation System



Caution However, the sample program for the G.729 Annex B multichannel version cannot be used with this evaluation system.

How to build the sample program for this middleware is described below.

(1) Start up the WB77016 (workbench).

(2) Open the project file (sample.prj).
Example: Specify sample.prj with the Open Project command on the Project menu.

(3) Execute Build and confirm that sample.lnk has been created.
Example: When the Build All command is selected in the Make menu, sample.lnk is created.

(4) Execute the sample program by downloading sample.lnk to the target DSP using the IE77016 (debugger).

3.3 Symbol Naming Regulations

The symbol names used in this library follow the rules listed below. Be careful to avoid overlap if using other applications in combination.

Table 3-1. Symbol Names

Classification	G.729	ANNEX A	ANNEX B	ANNEX AB[Note]	ANNEX B
	Single-Channel				Multichannel
Function name	g729_xxxx	g729a_xxxx	g729b_xxxx	g729ab_xxxx	g729b_xxxx
Macro, constant name	G729xxxx	G729Axxxx	G729Bxxxx	G729ABxxxx	g729bxxxx
Section name	_ _G729_xxxx	_ _G729A_xxxx	_ _G729B_xxxx	_ _G729AB_xxxx	_ _G729B_xxxx

Note Annex AB: Annex A + Annex B

[MEMO]

APPENDIX SAMPLE SOURCE

A.1 Sample Source for Single-Channel Version (sample.asm)

The sample source for the single-channel version described below is samplek.asm (sample source for G.729). In samplea.asm (sample source for G.729 Annex A) the G729 part becomes G729A, and the g729_ part becomes g729a_. In sampleb.asm (sample source for G.729 Annex B), the G729 part becomes G729B, and the g729_ part becomes g729b_. In sampleab.asm (sample source for G.729 Annex A + Annex B), the G729 part becomes G729AB, and the g729_ part becomes g729ab_.

Moreover, in sampleb.asm and sampleab.asm, a part related to frame counters (G729Bframe, G729ABframe) is added.

```
;; ********************************************************************
;;
;;  CS-CELP[G.729] CODEC control program Ver1.0
;; ********************************************************************

#DEFINE     MODEB_TYPE
#IFDEF      MODEB_TYPE
#DEFINE     MODE_BIT_SET
#ENDIF

#include "g729_lib.h"
#include "upd7701x.h"

#IF CHIPTYPE == TYPE77017
#DEFINE IN_OR_EXT
#ELSE
#DEFINE IN_OR_EXT   EXTERNAL
#ENDIF

;; ********************************************************************
;;  G729 Encoder, Decoder processing core Parts
;; ********************************************************************
EXTRN   g729_Enc           ;; G729 Encoder Main
EXTRN   g729_InitEnc       ;; G729 Encoder Initialize
EXTRN   g729_Dec           ;; G729 Decoder Main
EXTRN   g729_InitDec       ;; G729 Decoder Initialize

EXTRN   G729E_ANA_BUF      ;; Encoder Output Buffer
EXTRN   G729D_ANA_BUF      ;; Decoder Input Buffer
EXTRN   G729E_PCM_BUF      ;; Serial Input Buffer
EXTRN   G729D_PCM_BUF      ;; Serial Output Buffer
EXTRN   G729E_CMD_STS      ;; Encoder status
EXTRN   G729D_CMD_STS      ;; Decoder Status
EXTRN   G729ana            ;; Encoder internal output                    } Note 1
;; ********************************************************************


;; ====================================
;;  X/Y Data ROM Table (NEC)
;; ====================================
__G729_COMROMY YROMSEG IN_OR_EXT
;; ------------------------------------
;;  Encoder Mute Data (for ana[])
;; ------------------------------------
```

Note 1. In sampleb.asm (sampleab.asm), this part is as follows. However, in sampleab.asm, the G729B part becomes G729AB.

```
   EXTRN   G729Bana          ;; Encoder internal output
   EXTRN   G729Bframe        ;;
;;*****************************************************************
```

```
E_MUTE_ANA:                ;;
    DW  0b01111000         ;; ana[0]
    DW  0b0011111010       ;;     1
    DW  0b10111101         ;;     2
    DW  0b0                ;;     3
    DW  0b0000000000000    ;;     4
    DW  0b1111             ;;     5
    DW  0b1010110          ;;     6
    DW  0b11011            ;;     7
    DW  0b0000000000000    ;;     8
    DW  0b1111             ;;     9
    DW  0b1010110          ;;    10

;; ============================================
;;  Register Store Area
;; ============================================

__G729_X_REG_SAVE_SEG  XRAMSEG

SaveRegAreaX:
__R0_SAVE:      DS  3       ;; SaveRegAreaX+0x0000
__R1_SAVE:      DS  3       ;; SaveRegAreaX+0x0003
__R2_SAVE:      DS  3       ;; SaveRegAreaX+0x0003
__R3_SAVE:      DS  3       ;; SaveRegAreaX+0x0003
__R4_SAVE:      DS  3       ;; SaveRegAreaX+0x0003
__R5_SAVE:      DS  3       ;; SaveRegAreaX+0x0003
__R6_SAVE:      DS  3       ;; SaveRegAreaX+0x0003
__R7_SAVE:      DS  3       ;; SaveRegAreaX+0x0003
__DP0_SAVE:     DS  1       ;; SaveRegAreaX+0x0006
__DP1_SAVE:     DS  1       ;; SaveRegAreaX+0x0006
__DP2_SAVE:     DS  1       ;; SaveRegAreaX+0x0006
__DP3_SAVE:     DS  1       ;; SaveRegAreaX+0x0006
__DP4_SAVE:     DS  1       ;; SaveRegAreaX+0x0007
__DP5_SAVE:     DS  1       ;; SaveRegAreaX+0x0006
__DP6_SAVE:     DS  1       ;; SaveRegAreaX+0x0006
__DP7_SAVE:     DS  1       ;; SaveRegAreaX+0x001f

;; ======================================
;;  SI1 Save/Recover registers MACRO
;; ======================================

%DEFINE (G729SAVE_REG)(
    *SaveRegAreaX+0x0000:X = R0L   ;;
    *SaveRegAreaX+0x0001:X = R0H   ;;
    *SaveRegAreaX+0x0002:X = R0E   ;;

    *SaveRegAreaX+0x0003:X = R1L   ;;
    *SaveRegAreaX+0x0004:X = R1H   ;;
    *SaveRegAreaX+0x0005:X = R1E   ;;

    R0L = DP0                      ;;
    *SaveRegAreaX+0x00018:X = R0L  ;;
    R0L = DP1                      ;;
    *SaveRegAreaX+0x00019:X = R0L  ;;
)

%DEFINE (G729RECOVER_REG)(
    R0L = *SaveRegAreaX+0x0019:X   ;;
    DP1 = R0L                      ;;
    R0L = *SaveRegAreaX+0x0018:X   ;;
```

```
    DP0 = R0L                       ;;

    R1E = *SaveRegAreaX+0x0005:X    ;;
    R1H = *SaveRegAreaX+0x0004:X    ;;
    R1L = *SaveRegAreaX+0x0003:X    ;;

    R0E = *SaveRegAreaX+0x0002:X    ;;
    R0H = *SaveRegAreaX+0x0001:X    ;;
    R0L = *SaveRegAreaX+0x0000:X    ;;
)

;; ============================================
;; ENCODER SYMBOL
;; ============================================


__G729_ENC_DSEG_Y  YRAMSEG

E_SI1_CNT:      DS  1       ;; E_SI1_BUF index
E_FRAME_CNT:    DS  1       ;; Frame Number
E_SKIP_FLAG:    DS  1       ;; Skip flag for Encoder Startup
E_NEW_DATA:     DS  1       ;; New Data input flag

__G729_ENC_DSEG_X  XRAMSEG

E_SI1_BUF:      DS  80      ;; Serial input buffer  (Speech Signal)

;; ------------------------------------
;;  DECODER SYMBOL
;; ------------------------------------


__G729_DEC_DSEG_Y  YRAMSEG
D_SO1_CNT:      DS  1       ;; D_SO1_BUF index
D_NEW_DATA:     DS  1       ;; Input New Data flag
D_SKIP_FLAG:    DS  1       ;; Skip Flag for Startup Decoder
D_MUTE_DATA:    DS  1       ;; Mute Data

__G729_DEC_DSEG_X  XRAMSEG

D_SO1_BUF:      DS  80      ;; Output CODEC Frame Buffer

;; ********************************************************************
;;  G729 TEXT SEGMENT
;; ********************************************************************

SPX_START   IMSEG   at 0x200
;; -------------------------------------
;;  SPX Interrupt Vector Code
;; -------------------------------------
Reset_Entry:        ;;
    JMP Main        ;; Jump to Main
    NOP             ;;
    NOP             ;;
    NOP             ;;
    ;
    ; Reserved_Vector_Area [0x0204 - 0x020F]
    ;
    NOP             ;;
    NOP             ;;
```

```
    NOP             ;;
    NOP             ;;
    NOP             ;;
    NOP             ;;
    NOP             ;;
    NOP             ;;
    NOP             ;;
    NOP             ;;
    NOP             ;;
    NOP             ;;
INT1_Entry:             ;; -----------------------------------
    JMP FATAL_INT       ;; [INT1:External]
    NOP             ;;
    NOP             ;;
    NOP             ;;
INT2_Entry:             ;; -----------------------------------
    JMP FATAL_INT       ;; [INT2:External]
    NOP             ;;
    NOP             ;;
    NOP             ;;
INT3_Entry:             ;; -----------------------------------
    JMP FATAL_INT       ;; [INT3:External]
    NOP             ;;
    NOP             ;;
    NOP             ;;
INT4_Entry:             ;; -----------------------------------
    JMP FATAL_INT       ;; [INT4:External]
    NOP             ;;
    NOP             ;;
    NOP             ;;
SI1_Entry:              ;; -----------------------------------
    JMP Enc_SI1_Handler ;; [SI1:Internal]
    NOP             ;;
    NOP             ;;
    NOP             ;;
SO1_Entry:              ;; -----------------------------------
    JMP FATAL_INT       ;; [SO1:Internal]
    NOP             ;;
    NOP             ;;
    NOP             ;;
SI2_Entry:              ;; -----------------------------------
    JMP FATAL_INT       ;; [SI2:Internal]
    NOP             ;;
    NOP             ;;
    NOP             ;;
SO2_Entry:              ;; -----------------------------------
    JMP FATAL_INT       ;; [SO2:Internal]
    NOP             ;;
    NOP             ;;
    NOP             ;;
HI_Entry:               ;; -----------------------------------
    JMP FATAL_INT       ;; [HI:Internal]
    NOP             ;;
    NOP             ;;
    NOP             ;;
HO_Entry:               ;; -----------------------------------
    JMP FATAL_INT       ;; [HO:Internal]
    NOP             ;;
    NOP             ;;
    NOP             ;;
```

```
    ;
    ; Reserved_Vector_Area [0x0238 - 0x023F]
    ;
    NOP             ;;
    NOP             ;;
    NOP             ;;
    NOP             ;;
    NOP             ;;
    NOP             ;;
    NOP             ;;
    NOP             ;;

;; ***************************************************************************
;;
;;

NECMAIN IMSEG at 0x240
Main:
    ;; ------------------------------
    ;; Initialize SPX
    ;; ------------------------------
    R0L = 0xF3FF        ;; Disable All Interrupt
    SR  = R0L           ;;
    CLR(R0)             ;; Clear General Registers
    CLR(R1)             ;;
    CLR(R2)             ;;
    CLR(R3)             ;;
    CLR(R4)             ;;
    CLR(R5)             ;;
    CLR(R6)             ;;
    CLR(R7)             ;;
    DP0 = R0L           ;;
    DN0 = R0L           ;;
    DP1 = R0L           ;;
    DN1 = R0L           ;;
    DP2 = R0L           ;;
    DN2 = R0L           ;;
    DP3 = R0L           ;;
    DN3 = R0L           ;;
    DP4 = R0L           ;;
    DN4 = R0L           ;;
    DP5 = R0L           ;;
    DN5 = R0L           ;;
    DP6 = R0L           ;;
    DN6 = R0L           ;;
    DP7 = R0L           ;;
    DN7 = R0L           ;;
    DMX = R0L           ;;
    DMY = R0L           ;;
```

```
    *SST1:X = R0L       ;; SI1/SO1 MODE <= 16bit/MSB first
    *SST2:X = R0L       ;; SI2/SO2 MODE <= 16bit/MSB first
    *DWTR:X = R0L       ;; All Data Memory is NO WAIT
    *IWTR:X = R0L       ;; All Inst Memory is NO WAIT
    R1L = 0x300         ;;
    *HST:X  = R1L       ;; Disable HDT In/Out                          } Note 2

    ;; -------------------------------
    ;; Clear Memory[X/Y:0x0000-0x07FF]
    ;; -------------------------------
    DP0 = R0L                       ;; X Memory Pointer <= 0x0000
    DP4 = R0L                       ;; Y Memory Pointer <= 0x0000

    REP 0x800                       ;; -- loop [clear] --
            *DP0++=R0H  *DP4++=R0H  ;; Clear X/Y
                                    ;; -- loop end -------------------
    CLR(R0)                         ;;
    R3L = 0x55FF                    ;; R3L = A-law Mute Data
    *D_MUTE_DATA:y = R3L            ;; D_MUTE_DATA = R3L

;; *********************************************************************
;;
;; *********************************************************************
MAIN2:

_ENC_INI:
    ;; -------------------------------
    ;;  Encoder Initialize
    ;; -------------------------------

     CALL    Initialize_Encoder     ;;

    ;; -------------------------------
    ;;  SST1[BIT-9,8] <- [1,0]
    ;; -------------------------------
    R1L = 0x200                     ;;
    *SST1:X = R1L                   ;;

_DEC_INI:
    ;; -------------------------------
    ;;  Decoder Initialize
    ;; -------------------------------

     CALL    Initialize_Decoder     ;;

    ;; -------------------------------
    ;;  SO1 Dummy W
    ;; -------------------------------
    R1L = *D_MUTE_DATA:y            ;; Read D_MUTE_DATA
    *SDT1:X = R1L                   ;; Dummy Write to SO1
```

Note 2. In sampleb.asm (sampleab.asm), this part is as follows. However, in sampleab.asm, the G729B part becomes G729AB.

```
    *HST:X  = R1L       ;; Disable HDT In/Out
    *G729Bframe:y = R0L ;;

    ;; -------------------------------
```

```
    ;; ------------------------------
    ;;  Enable Interrupt
    ;; ------------------------------
    R0L = SR              ;; READ Status Register

    R0 = R0 & 0x7FEF      ;; Open Current Int Mask
    SR = R0L              ;;
    nop                   ;;

;; *********************************************************************
;;  G729 Encoder/Decoder Main Loop
;; *********************************************************************

MainLoop:
    ;; ------------------------------
    ;;  Check Mode
    ;; ------------------------------
    CLR(R1)                         ;;

    ;; ################################
    ;;   Normal Process
    ;; ################################

    ;; ------------------------------
    ;;  Check E_SKIP_FLAG
    ;; ------------------------------
    R0 = *E_SKIP_FLAG:y             ;; IF E_SKIP_FLAG == ON then SKIP
    IF (R0 == 0) JMP chk_E_NEW_DATA ;;

    ;; ------------------------------
    ;;  Encoder init
    ;; ------------------------------
Enc_Skip:                           ;;
    Call g729_InitEnc               ;;
    DP4 = E_MUTE_ANA                ;;
    DP0 = G729ana                   ;;
        R3=*DP4++                   ;;
    Rep 10                          ;;
        R3=*DP4++   *DP0++=R3H      ;;
                *DP0++=R3H          ;;

    ;; ------------------------------
    ;;  Set E_SKIP_FLAG
    ;; ------------------------------
    CLR(R0)                         ;;
    *E_SKIP_FLAG:y = R0L            ;; Clear E_SKIP_FLAG

    JMP EndEncStage                 ;;

    ;; ------------------------------
    ;;  Check E_NEW_DATA
    ;; ------------------------------
chk_E_NEW_DATA:                     ;;
    R0 = *E_NEW_DATA:y              ;;
    IF (R0 == 0) JMP EndEncStage    ;;
```

```
    ;; -------------------------------
    ;;  Run Encoder
    ;; -------------------------------
    CALL    g729_Enc                ;; Do Encoder

    CLR ( R0 )                      ;;
    *E_NEW_DATA:y = R0L             ;;
    R0L = 0x1                       ;;
    *D_NEW_DATA:y = R0L             ;;

EndEncStage:                        ;;

    DP4 = G729E_ANA_BUF             ;;
    DP5 = G729D_ANA_BUF             ;;
    loop 5 {                        ;;
        R2 = *DP4++                 ;;
        *DP5++=R2H                  ;;
    }                               ;;


    ;; -------------------------------
    ;;  Check D_SKIP_FLAG
    ;; -------------------------------
    R0 = *D_SKIP_FLAG:y             ;; IF D_SKIP_FLAG == ON then SKIP;
    IF (R0 == 0) JMP chk_D_NEW_DATA ;;

    ;; -------------------------------
    ;;  Decoder Init
    ;; -------------------------------
Dec_Skip:                           ;;

    Call    g729_InitDec            ;;
    R0 = *D_MUTE_DATA:y             ;;
    DP1 = G729D_PCM_BUF             ;;
    Rep 80                          ;;
        *DP1++=R0H                  ;;
    jmp EndDecStage                 ;;
```

Note 3

Note 3. In sampleb.asm (sampleab.asm), this part is as follows. However, in sampleab.asm, the G729B part becomes G729AB, and the g729B_ part becomes g729ab_.

```
    ;; -------------------------------
    CLR(R0)                         ;;
        R0L = *G729Bframe:y         ;;
        R0 = R0 + 1                 ;;
        R1 = R0 - 0x7fff            ;;
        IF(R1 < 0)JMP over_frm_cnt  ;;
        R0L = 0x100                 ;;
ovr_frm_cnt:
        *G729Bframe:y = R0L         ;;

    CALL    g729B_Enc               ;; Do Encoder
```

```
    ;; ------------------------------
    ;;  Check D_NEW_DATA
    ;; ------------------------------
chk_D_NEW_DATA:                     ;;
    R0 = *D_NEW_DATA:y              ;;
    IF (R0 == 0) JMP EndDecStage    ;;

    ;; ------------------------------
    ;;  Run Decoder
    ;; ------------------------------
    CALL    g729_Dec                ;; Do Decoder
    CLR ( R0 )                      ;;
    *D_NEW_DATA:y = R0L             ;;

EndDecStage:

    ;; ------------------------------
    ;;  Set D_SKIP_FLAG
    ;; ------------------------------
    CLR(R0)                         ;; Clear D_SKIP_FLAG;
    *D_SKIP_FLAG:y = R0L            ;;

    JMP MainLoop                    ;; Infinite Loop


;; ************************************************
;; **** G729 Encoder/Decoder Subroutine Segment ****
;; ************************************************
SUBR_SEG    IMSEG
;; --------------------------------------
;;  Encoder Initialize
;; --------------------------------------
Initialize_Encoder:
                                    ;; note! R0,R1,DP0,DP4 ONLY
    R0L = 1                         ;;
    *E_SKIP_FLAG:y = R0L            ;; E_SKIP_FLAG = 1

    R0L = 0x0080                    ;;
    *G729E_CMD_STS:y = R0L          ;;
    CLR(R0)                         ;;
    *E_NEW_DATA:y  = R0L            ;;

    R0L = 3                         ;;
    *E_FRAME_CNT:y = R0L            ;; E_FRAME_CNT = 3

    R0L = 0                         ;;
    *D_SO1_CNT:y   = R0L            ;; D_SO1_CNT   = 0

    R0L = 0                         ;;
    *E_SI1_CNT:y   = R0L            ;; E_SI1_CNT   = 0

    RET                             ;;

;; --------------------------------------
;;  Decoder Initialize
;; --------------------------------------

Initialize_Decoder:
                                    ;; note! R0,R1,DP0,DP4 ONLY
    R0L = 0x0085                    ;;
```

```
    *G729D_CMD_STS:y = R0L         ;;
    CLR(R0)                        ;;
    *D_NEW_DATA:y  = R0L           ;; D_NEW_DATA  = 0
    R0L = 1                        ;;
    *D_SKIP_FLAG:y = R0L           ;; D_SKIP_FLAG = 1

    R0 = *D_MUTE_DATA:y            ;;
    DP1 = G729D_PCM_BUF            ;;
    Rep 80                         ;;
        *DP1++=R0H                 ;;

    RET                            ;;

;; ********************************************************************
;;  Serial Import CH-1 Interrupt Handler Segment (Encoder)
;; ********************************************************************
EncSI1SEG   IMSEG   at  0x4000
;; -------------------------------------
;;  Encoder Serial-In Interrupt Flow
;; -------------------------------------
Enc_SI1_Handler:
    ;; -------------------------------
    ;;  Save Registers
    ;;     R0/R1/DP0/DP4
    ;; -------------------------------
    %G729SAVE_REG                  ;;

    ;; -------------------------------
    ;;  Check D_SO1_CNT
    ;; -------------------------------
    clr(R0)                        ;;
    R0L = *D_SO1_CNT:y             ;;

    R0 = R0 - 80                   ;; 0 <= D_SO1_CNT <= 79
    if(R0 < 0 ) jmp skip_D_SO1_RST ;;
    clr( R0)                       ;;
    *D_SO1_CNT:y = R0L             ;;

skip_D_SO1_RST:

    ;; -------------------------------
    ;;  Check E_SI1_CNT
    ;; -------------------------------
    clr(r1)                        ;;
    R1L = *E_SI1_CNT:y             ;;
    R1 = R1 - 80                   ;;
    if(R1 < 0 )  jmp not_full_SMPL ;;

    *E_SI1_CNT:y = R1L             ;; E_SI1_CNT=0

    ;; -------------------------------
    ;;  E_SI1_BUF -> E_PCM_BUF
    ;; -------------------------------
    DP1 = E_SI1_BUF                ;;
    R1L = G729E_PCM_BUF            ;;
    DP0 = R1L                      ;;

    Loop 80 {                      ;;
        R1=*DP1++                  ;;
        *DP0++=R1H                 ;;
```

```
    }                               ;;

    ;; ----------------------------;;
    ;;  Output Data Copy
    ;; ----------------------------;;
    DP0 = G729D_PCM_BUF             ;;
    DP1 = D_SO1_BUF                 ;;
    Loop 80 {                       ;;
            R0 = *DP0++             ;;
            *DP1++ = R0H            ;;
    }                               ;;

    ;; ------------------------------
    ;;  E_NEW_DATA <- 1
    ;; ------------------------------
    R1L = 1                         ;;
    *E_NEW_DATA:y = R1L             ;;


not_full_SMPL:
    ;; ------------------------------
    ;;  Input Serial Port
    ;; ------------------------------
    R0L = *E_SI1_CNT:y              ;;
    R1 = R0 + E_SI1_BUF             ;;
    DP0 = R1L                       ;;
    R1L = *SDT1:X                   ;;
    *DP0 = R1L                      ;;

    ;; ----------------------------;;
    ;;  Output Serial Port
    ;; ----------------------------;;

    R0L = *D_SO1_CNT:y              ;;

    R0 = R0 + D_SO1_BUF             ;;
    DP0 = R0L                       ;;

    nop                             ;;
    R0L = *DP0                      ;;

    *SDT1:X = R0L                   ;;

    ;; ------------------------------
    ;;  Increment E_SI1_CNT
    ;; ------------------------------
    R0L = *E_SI1_CNT:y              ;;
    R0 = R0 + 1                     ;;
    *E_SI1_CNT:y = R0L              ;;

    ;; ------------------------------
    ;;  D_SO1_CNT:Update
    ;; ------------------------------
    clr(R0)                         ;;
    R0L = *D_SO1_CNT:y              ;;
    R0 = R0 + 1                     ;;
    *D_SO1_CNT:y = R0L              ;;

    ;; ------------------------------
    ;;  Recover Registers
```

```
    ;; ------------------------------
    %G729RECOVER_REG              ;;
    RETI                          ;;


;; ============================================================================
;;  Fatal Error!!
;;  Disable All Interrupt & infinite loop
;;  without reset Watch Dog Timer.
;; ============================================================================
%DEFINE (FATAL_ERROR(Label))
(
Label:  R7L = 0xFFFF              ;; Disable Interrupt
    SR  = R7L                     ;;
    JMP $-2                       ;; Infinite Loop
)

;;/*************************************************/


%FATAL_ERROR(ERR_SO2_CNT)         ;;
%FATAL_ERROR(ERR_SI1_CNT)         ;;
%FATAL_ERROR(ERR_E_BUSY)          ;;

%FATAL_ERROR(ERR_D_BUSY)          ;;

%FATAL_ERROR(ERR_E_COD_CNT)       ;;
%FATAL_ERROR(ERR_D_COD_CNT)       ;;
%FATAL_ERROR(ERR_E_EXT_CNT)       ;;
%FATAL_ERROR(ERR_D_EXT_CNT)       ;;
%FATAL_ERROR(ERR_MEM)             ;;
%FATAL_ERROR(ERR_MODE)            ;;
%FATAL_ERROR(ERR_MODE2)           ;;
%FATAL_ERROR(ERR_HDT1)            ;;
%FATAL_ERROR(ERR_HDT2)            ;;

%FATAL_ERROR(ERR_E_FRAME_CNT)     ;;
%FATAL_ERROR(ERR_E_SI1_CNT)       ;;

;; ============================================================================
;;  Illegal interruption handling routine. (DI HALT)
;; ============================================================================
FATAL_INT:                        ;;
    R0L = 0xFFFF                  ;; Disable interrupt
    SR = R0L                      ;;
FATAL_INT_LOOP:                   ;;
    HALT                          ;; Halt loop
    JMP FATAL_INT_LOOP            ;;

    END
;; ---------------------- end of file ---------------------------------------
```

A.2 Sample Source for Multichannel Version (samplebm.asm)

This sample source serves to transmit/receive the data of 4 channels using the host interface. The channel number of the data that is transmitted in and encoder (6) and decoder (7) execution commands are required. For these execution commands, refer to **A.2.1 samplebm.asm header file (sysconf.h)**.

```
;; ***************************************************************************
;;  CS-ACELP[G.729 multi-channel] CODEC control program for 4-channel Ver3.0
;; ***************************************************************************

#include "g729blib.h"
#include "sysconf.h"                /* Application definition file */
#include "upd7701x.h"               /* SPX definition file */
public      lib_Scratch_x, lib_Scratch_y
#define L_FRAME    80                    ; Frame size.
#define PRM_SIZE    5                    ; Size of vector of analysis parameters.
#define UPD77116
extrn   g729b_StartCodec

/*---- Memory allocation-----------*/
__G729B_SCRATCHX    xramseg     align at 4
lib_Scratch_x:
            ds      G729B_SCRATCH_X_BUFSIZE

__G729B_SCRATCHY    yramseg
lib_Scratch_y:
            ds      G729B_SCRATCH_Y_BUFSIZE

USER_RAMX       xramseg     Internal
/*-- channel 1 --*/
g729b_IO_Table1:
             ds      G729B_IOTABLE_SIZE
STATIC_X1:   ds      G729B_STATIC_X_BUFSIZE

EncPcmData1: ds      G729B_ENCODE_PCM_BUFSIZE
EncPrmData1: ds      G729B_ENCODE_PRM_BUFSIZE
DecPcmData1: ds      G729B_DECODE_PCM_BUFSIZE       ;
DecPrmData1: ds      G729B_DECODE_PRM_BUFSIZE       ;

/*-- channel 2 --*/
g729b_IO_Table2:
             ds      G729B_IOTABLE_SIZE
STATIC_X2:   ds      G729B_STATIC_X_BUFSIZE

EncPcmData2: ds      G729B_ENCODE_PCM_BUFSIZE
EncPrmData2: ds      G729B_ENCODE_PRM_BUFSIZE
DecPcmData2: ds      G729B_DECODE_PCM_BUFSIZE       ;
DecPrmData2: ds      G729B_DECODE_PRM_BUFSIZE       ;

/*-- channel 3 --*/
g729b_IO_Table3:
```

```
            ds      G729B_IOTABLE_SIZE
STATIC_X3:  ds      G729B_STATIC_X_BUFSIZE

EncPcmData3: ds     G729B_ENCODE_PCM_BUFSIZE
EncPrmData3: ds     G729B_ENCODE_PRM_BUFSIZE
DecPcmData3: ds     G729B_DECODE_PCM_BUFSIZE        ;
DecPrmData3: ds     G729B_DECODE_PRM_BUFSIZE        ;

/*-- channel 4 --*/                                 ;
g729b_IO_Table4:
            ds      G729B_IOTABLE_SIZE
STATIC_X4:  ds      G729B_STATIC_X_BUFSIZE

EncPcmData4: ds     G729B_ENCODE_PCM_BUFSIZE
EncPrmData4: ds     G729B_ENCODE_PRM_BUFSIZE
DecPcmData4: ds     G729B_DECODE_PCM_BUFSIZE
DecPrmData4: ds     G729B_DECODE_PRM_BUFSIZE


USER_RAMY      yramseg
/*-- channel 1 --*/
STATIC_Y1:  ds      G729B_STATIC_Y_BUFSIZE

/*-- channel 2 --*/
STATIC_Y2:  ds      G729B_STATIC_Y_BUFSIZE

/*-- channel 3 --*/
STATIC_Y3:  ds      G729B_STATIC_Y_BUFSIZE

/*-- channel 4 --*/
STATIC_Y4:  ds      G729B_STATIC_Y_BUFSIZE


%define(MakeIOTable(ch))
(
    /*---*/                 dp0=g729b_IO_Table@ch@  ;
    clr(r1)                                         ;
    r0l=STATIC_X@ch@                                ;
    /*---*/                 *dp0++=r0l              ;
    r0l=STATIC_Y@ch@                                ;
    /*---*/                 *dp0++=r0l              ;
    r0l=EncPcmData@ch@                              ;
    /*---*/                 *dp0++=r0l              ;
    r0l=EncPrmData@ch@                              ;
    /*---*/                 *dp0++=r0l              ;
    rep 4                                           ;
        /*...*/             *dp0++=r1h              ;
    r0l=DecPcmData@ch@                              ;
    /*---*/                 *dp0++=r0l              ;
    r0l=DecPrmData@ch@                              ;
    /*---*/                 *dp0++=r0l              ;
    r0l= 1                                          ;
```

```
        /*   */             *dp0++=r0l              ; fParityCheck = 1
    rep 5                                          ;
        /*...*/             *dp0++=r1h              ;
    /*   */                 *dp0++=r0l              ; fVadEnable = 1
    rep 3                                          ;
        /*...*/             *dp0++=r1h              ;
)

USER_TEXT_SEG     imseg
g729b_MkIO_Table:
        %MakeIOTable(1)                            ;
        %MakeIOTable(2)                            ;
        %MakeIOTable(3)                            ;
        %MakeIOTable(4)                            ;
        ret                                        ;

/*---- Symbol definition----*/
%define ( DefVectNop ) (              /* Macro for vector definition (no processing) */
    reti;
    nop;
    nop;
    nop;
    )
%define ( DefVectJmp(iproc) ) (       /* Macro for vector definition (jmp iproc) */
    jmp  iproc;
    nop;
    nop;
    nop;
    )

/*---- Functions entry----*/
public  ClrIntMask                    /* Interrupt mask change clear */
public  SetIntMask                    /* Interrupt mask change set */
public  ErrorDump                     /* Error data display  */

$EJECT
/**/
/************************************************************************/
/* Make interrupt vector settings                                       */
/************************************************************************/
Sys_V  imseg at 0x0200
    %DefVectJmp(StartUp)              /*- Reset -------------------------*/
    %DefVectNop                       /*- Reserved 1---------------------*/
    %DefVectNop                       /*- Reserved 2---------------------*/
    %DefVectNop                       /*- Reserved 3---------------------*/
    %DefVectJmp(Codec1CK)             /*- INT1  -------------------------*/
    %DefVectJmp(Codec1FM)             /*- INT2  -------------------------*/
    %DefVectJmp(Codec2CK)             /*- INT3  -------------------------*/
    %DefVectJmp(HostIntProc)          /*- INT4  -------------------------*/
    %DefVectJmp(SI1Proc)              /*- SI1   -------------------------*/
    %DefVectJmp(SO1Proc)              /*- SO1   -------------------------*/
    %DefVectJmp(SI2Proc)              /*- SI2   -------------------------*/
```

```
    %DefVectJmp(SO2Proc)             /*- SO2   --------------------------*/
    %DefVectJmp(HostIn)              /*- HI    --------------------------*/
    %DefVectJmp(HostOut)             /*- HO    --------------------------*/
    %DefVectNop                      /*- Reserved 4----------------------*/
    %DefVectNop                      /*- Reserved 5----------------------*/

/*************************************************************************/
/* Register save are for vectored interrupt processing                   */
/*************************************************************************/
Sys_XE xramseg
HipSave:
    ds  4;

/*************************************************************************/
/* Vectored interrupt processing according to PIO value                  */
/*************************************************************************/
Sys_II1 imseg
HostIntProc:
    *HipSave+0:x = r0e;               /* Register save */
    *HipSave+1:x = r0h;
    *HipSave+2:x = r0l;
    r0l= dp0;
    *HipSave+3:x = r0l;

    r0l= *PDT:x;                      /* PIO value registration */
    r0 = r0 & MAX_HIP_NO;             /* Execution of registered interrupt processing
*/
    r0 = r0 + HipJumpTbl;
    /*   */             dp0 = r0l;
    call dp0;

    r0l= *HipSave+3:x;                /* Register restore */
    /*   */             dp0 = r0l;
    r0l= *HipSave+2:x;
    r0h= *HipSave+1:x;
    r0e= *HipSave+0:x;
    reti;

  HipJumpTbl:
    jmp  StartErrDump;                /* For error processing */
    jmp  HintProc1;                   /* jump to registered interrupt processing */
    jmp  HintProc2;
    jmp  HintProc3;
    jmp  HintProc4;
    jmp  HintProc5;
    jmp  HintProc6;
    jmp  HintProc7;

$EJECT
/**/
/*************************************************************************/
/* Perform initialization processing, wait for command                   */
/*************************************************************************/
```

```
Sys_II2 imseg
StartUp:
    call InitMode;                  /* mode */
    call InitWait;                  /* Wait controller setting  */
    call InitPort;                  /* General-purpose I/O port setting  */
    call InitInt;                   /* Interrupt initialization  */
    call SetHifBusy;                /* Busy setting */
    WaitCom:
        call InitPort;              /* General-purpose I/O port setting  */

        call SetHifReady;           /* Ready setting */

        clr(r1);                    /* Command ID input */
        r1l= *HDT:x;
        call SetHifBusy;            /* Busy setting */

        r0 = r1 - 11;               /* Command ID check */
        if( r0>0 ) call ErrorDump;

        r0 = r1 + ComTbl;           /* Call command processing */
        /*---*/         dp0=r0l;
        nop;
        /*   */         r0l=*dp0;
        /*---*/         dp0=r0l;
        call dp0;

        jmp  WaitCom;


/*************************************************************************/
/* Function table used to perform command processing                     */
/*************************************************************************/
#ifdef  UPD77116
Sys_XC      xramseg
#else
Sys_XC      xromseg
#endif
ComTbl:
    dw  NoProc                  /* ComId=0  : No processing */
    dw  PowerDown               /* ComId=1  : Power down  */
    dw  SetData                 /* ComId=2  : Data set */
    dw  PutData                 /* ComId=3  : Data put */
    dw  Com4                    /* ComId=4  : Virtual parameter */
    dw  Com5                    /* ComId=5  : Virtual parameter */
    dw  Com6                    /* ComId=6  : Virtual parameter */
    dw  Com7                    /* ComId=7  : Virtual parameter */
    dw  Com8                    /* ComId=8  : Virtual parameter */
    dw  Com9                    /* ComId=9  : Virtual parameter */
    dw  Com10                   /* ComId=10 : Virtual parameter */
    dw  Com11                   /* ComId=11 : Virtual parameter */
```

```
/************************************************************************/
/*                                                                      */
/************************************************************************/
Sys_II3 imseg
InitMode:
    r0l= 0x440;
    *HST:x=r0l;
;    r0l= 0x0F0;               Use HWE,HRE
;    *ICR:x = r0l;
    ret;

/************************************************************************/
/* Initialize wait controller                                           */
/************************************************************************/
Sys_II3 imseg
InitWait:
    r0l= DWTR_WAIT0;                    /*  External data memory setting */
    *DWTR:x = r0l;                      /*  External instruction memory setting */
    r0l= IWTR_WAIT0;
    *IWTR:x = r0l;
    ret;

/************************************************************************/
/* Set general-purpose I/O ports (all input)                            */
/************************************************************************/
Sys_II4 imseg
InitPort:
    r0l= PIO_ALL_IN;
    *PCD:x = r0l;
    ret;

$EJECT
/**/
/************************************************************************/
/* Initialize interrupt mask flag etc.                                  */
/*                                                                      */
/*  others: r_      [*, , , , , , , ]   dmx,dmy         [ , ]           */
/*          dp_     [ , , , , , , , ]   loops/stacks    [0/0]           */
/*          dn_     [ , , , , , , , ]   cycles          8               */
/************************************************************************/
Sys_II5 imseg
InitInt:
    clr(r0);
    r0l= 0xFFFF;                        /* EIR initialization */
    EIR= r0l;

    fint;                               /* Interrupt request annulled */

    r0l= SR;                            /* Interrupt enable flags change */
    r0 = r0 | 0x03FF;                   /*  All disabled  */
    SR = r0l;
```

```
    ret;

/*****************************************************************************/
/* Set interrupt mask flag                                                   */
/* Change interrupt status                                                   */
/*                                                                           */
/*  input : r0l     change-bit setting                                       */
/*          r1l     Interrupt status setting 0x8000: Change, 0x0000: No change */
/*  others: r_      [ , ,*,*, , , , ]   dmx,dmy         [ , ]                */
/*          dp_     [ , , , , , , , ]   loops/stacks    [0/0]                */
/*          dn_     [ , , , , , , , ]   cycles          10                   */
/*****************************************************************************/
Sys_II6 imseg
SetIntMask:
    r2l= EIR;                           /* Interrupt disable setting */
    r3 = r2 | 0x8000;
    EIR= r3l;
    r3 = r2 ^ r1;
    r2l= SR;                            /* Interrupt enable flag change */
    r2 = r2 | r0;
    SR = r2l;
    EIR= r3l;                           /* Interrupt status change  */
    ret;

/*****************************************************************************/
/* Clear interrupt mask flag                                                 */
/* Change interrupt status                                                   */
/*                                                                           */
/*  input : r0l     Change-bit setting                                       */
/*          r1l     Interrupt status setting 0x8000: Change, 0x0000: No change */
/*  others: r_      [*, ,*,*, , , , ]   dmx,dmy         [ , ]                */
/*          dp_     [ , , , , , , , ]   loops/stacks    [0/0]                */
/*          dn_     [ , , , , , , , ]   cycles          11                   */
/*****************************************************************************/
Sys_II7 imseg
ClrIntMask:
    clr(r2);
    r2l= EIR;                           /* Interrupt disable setting */
    r3 = r2 | 0x8000;
    EIR= r3l;
    r3 = r2 ^ r1;
    r2l= SR;                            /* Interrupt enable flag change */
    r0 = r0 ^ 0xFFFF;
    r2 = r2 & r0;
    SR = r2l;
    EIR= r3l;                           /* Interrupt status change */
    ret;

$EJECT
/**/
/*****************************************************************************/
/* Set host IF to Busy                                                       */
```

```
/*                                                                          */
/*  others: r_      [*, , , , , , , ]   dmx,dmy         [ , ]               */
/*          dp_     [ , , , , , , , ]   loops/stacks    [0/0]               */
/*          dn_     [ , , , , , , , ]   cycles          6                   */
/****************************************************************************/
Sys_II8 imseg
SetHifBusy:
    r0l= *HST:x;                        /*  Flag-bit save */
    R0 = R0 & 0x003F;                   /*  16-bit, with wait, busy */
    r0 = r0 | 0x0440;
    *HST:x = r0l;
    ret;

/****************************************************************************/
/* Set host IF to Ready                                                     */
/*  * Caution: HST-related error flag will be cleared                       */
/*                                                                          */
/*  others: r_      [*, , , , , , , ]   dmx,dmy         [ , ]               */
/*          dp_     [ , , , , , , , ]   loops/stacks    [0/0]               */
/*          dn_     [ , , , , , , , ]   cycles          4                   */
/****************************************************************************/
Sys_II9 imseg
SetHifReady:
    r0l= HST_WAIT;                      /*  16-bit, with wait, ready */
    *HST:x = r0l;
    ret;

$EJECT
/**/
/****************************************************************************/
/* Download to external instruction and data memories                       */
/*                                                                          */
/*  others: r_      [*, , , , , , ,*]   dmx,dmy         [ , ]               */
/*          dp_     [*, , ,*,*, , , ]   loops/stacks    [2/1]               */
/*          dn_     [ , , , , , , , ]   cycles          ...                 */
/****************************************************************************/
Sys_II10 imseg
DownLoadProc:
    clr(r7);                            /* Data set to external instruction memory */
    r7l= *HDT:x;                        /*  Instruction count */
    if( r7==0 ) jmp Xmem;               /*                    */
    r0l= *HDT:x;                        /*  Segment address   */
    /*---*/             dp3=r0l;       /*                    */
    call HostReBoot;

  Xmem:
    clr(r0);
    r0l= *HDT:x;                        /* Data set to x memory */
    if( r0==0 ) jmp  Ymem;              /*  Segment count */
    loop r0l {                          /*                */
        r0l= *HDT:x;                    /*  Segment data count */
        r1l= *HDT:x;                    /*                     */
```

```
        /*---*/         dp0=r1l;          /*  Segment address */
        loop r0l {                        /*                  */
            r0l= *HDT:x;                  /*  Setting data */
            /*   */     *dp0++=r0l;
        }
        clr(r0);
    }

  Ymem:
    r0l= *HDT:x;                          /* Data set to y memory */
    if( r0==0 ) ret;                      /*  Segment count */
    loop r0l {                            /*               */
        r0l= *HDT:x;                      /*  Segment data count */
        r1l= *HDT:x;                      /*                     */
        /*---*/         dp4=r1l;          /*  Segment address */
        loop r0l {                        /*                  */
            r0l= *HDT:x;                  /*  Setting data */
            /*   */     *dp4++=r0l;
        }
        nop;
    }
    ret;

/************************************************************************/
/*  Check operation status                                              */
/*                                                                      */
/*  others: r_      [*,*, , , , , , ]   dmx,dmy         [ , ]           */
/*          dp_     [ , , , , , , , ]   loops/stacks    [0/0]           */
/*          dn_     [ , , , , , , , ]   cycles          14              */
/************************************************************************/
Sys_IE1 imseg
HealthCheck:
    clr(r0);
    r0l= ESR;
    if( r0!=0 ) call  ErrorDump;

    r0l= LSP;
    if( r0!=0 ) call  ErrorDump;

    r0l= EIR;
    r1 = r0 - 0xFFFF;
    if( r1!=0 ) call  ErrorDump;

    r0l= SP;
    r1 = r0 - 0x0001;
    if( r1!=0 ) call  ErrorDump;

    r0l= *HST:x;                        /* Wait for host to */
    r1 = r0 & HST_ERRMASK;              /* read out parameter */
    if( r1!=0 ) call  ErrorDump;
    r1 = r0 & HST_SENMASK;
    if( r1!=0 ) jmp  $-4;
```

```
    nop;
    ret;

$EJECT
/**/
/************************************************************************/
/* For error Dump                                                       */
/************************************************************************/
Sys_XE xramseg
EdLock:
    ds  1;
CoreImage:
    ds  0x20;

/************************************************************************/
/*  Dump error data, end command processing                             */
/************************************************************************/
Sys_IE2 imseg
ErrorDump:
    *CoreImage+0x02:x = r0l;              /* Interrupt disable setting */
    *CoreImage+0x03:x = r0h;              /* Register value dump */
    call InitInt;
    *CoreImage+0x04:x = r1l;
    *CoreImage+0x05:x = r1h;
    *CoreImage+0x06:x = r2l;
    *CoreImage+0x07:x = r2h;
    *CoreImage+0x08:x = r3l;
    *CoreImage+0x09:x = r3h;
    *CoreImage+0x0A:x = r4l;
    *CoreImage+0x0B:x = r4h;
    *CoreImage+0x0C:x = r5l;
    *CoreImage+0x0D:x = r5h;
    *CoreImage+0x0E:x = r6l;
    *CoreImage+0x0F:x = r6h;
    *CoreImage+0x10:x = r7l;
    *CoreImage+0x11:x = r7h;
    r0l= dp0;
    *CoreImage+0x12:x = r0l;
    r0l= dp1;
    *CoreImage+0x13:x = r0l;
    r0l= dp2;
    *CoreImage+0x14:x = r0l;
    r0l= dp3;
    *CoreImage+0x15:x = r0l;
    r0l= dp4;
    *CoreImage+0x16:x = r0l;
    r0l= dp5;
    *CoreImage+0x17:x = r0l;
    r0l= dp6;
    *CoreImage+0x18:x = r0l;
    r0l= dp7;
```

```
    *CoreImage+0x19:x = r0l;
    r0l= STK;                           /* Error occurrence address dump */
    *CoreImage+0x00:x = r0l;
    r0l= ESR;                           /* ESR value dump */
    *CoreImage+0x01:x = r0l;

    r0l= *HST:x;                        /* Error flag set */
    r0 = r0 & 0x003F;                   /*  Flag-bit save */
    r0 = r0 | 0x04C0;                   /*  16 bit, with wait, busy, error */
    *HST:x = r0l;

    clr(r0);                            /* Initialize stack(etc.) */
    SP = r0l;
    LSP= r0l;
    ESR= r0l;

    clr(r0);                            /* Wait for INT4(Vect:7) */
    *EdLock:x = r0l;
    r0l= INTFLG_INT4;
    r1l= 0x8000;
    call ClrIntMask;
    nop;
    r0 = *EdLock:x;
    if( r0==0 ) jmp  $-2;
    call InitInt;

    /*---*/          dp0=CoreImage;   /* Error data reported to Host */
    loop 0x1A {
        /*...*/      r0l=*dp0++;
        *HDT:x = r0l;
    }

    r0l= *HST:x;                        /* Clear host input */
    r0 = r0 & HST_LENMASK;              /* Return to command wait status */
    if( r0!=0 ) jmp  WaitCom;
    r0l= *HDT:x;
    jmp  WaitCom;

StartErrDump:                           /* INT2(Vect:7) processing */
    r0l= 1;
    *EdLock:x = r0l;
    ret;

$EJECT
/**/
/************************************************************************/
/* No processing <for interrupt processing>                             */
/************************************************************************/
Sys_IE3 imseg
IntNoProc:
    reti;
```

```
/************************************************************************/
/* No processing                                                        */
/************************************************************************/
Sys_IE4 imseg
NoProc:
    ret;


/************************************************************************/
/* Power down                                                           */
/*  *1) No return to normal operation from power down mode              */
/************************************************************************/
Sys_IE5 imseg
PowerDown:
    r0l= EIR;                       /* All interrupts:disabled */
    r0 = r0 | 0x8000;               /* Disable Power Down release */
    EIR= r0l;
    nop;

      PdLoop:
        halt;                       /* PowerDown */
        jmp  PdLoop;

$EJECT
/**/
/************************************************************************/
/* Read data from host IF, store in specified area                      */
/*  1. Receive stored parameters from host IF                           */
/*      Memory classification 0) x memory 1) y memory                   */
/*      Stored addresses                                                */
/*      Stored data count                                               */
/*  2. Receive stored data from host IF, store in specified memory      */
/*                                                                      */
/*  others: r_      [*,*,*,*, , , , ]   dmx,dmy         [ , ]          */
/*          dp_     [*, , , ,*, , , ]   loops/stacks    [1/1]          */
/*          dn_     [ , , , , , , , ]   cycles          ...            */
/************************************************************************/
Sys_IE6 imseg
SetData:
    clr(r2);                                /* Stored parameter introduction  */
    clr(r3);                                /*                              */
    r1  = *HDT:x;                           /*  Memory classification */
    r2l = *HDT:x;                           /*  Stored address */
    r3l = *HDT:x;                           /*  Stored data count */

    if( r3==0 ) ret;                        /* Stored parameter analysis */
    if( r1!=0 ) jmp  SetYdata;

  SetXdata:
    /*---*/         dp0=r2l;              /* Stored to x memory */
    loop r3l {
```

```
        r0  = *HDT:x;
        /*...*/     *dp0++=r0h;
    }
    ret;

  SetYdata:
    /*---*/         dp4=r2l;                    /* Stored to y memory */
    loop r3l {
        r0  = *HDT:x;
        /*...*/     *dp4++=r0h;
    }
    ret;

/************************************************************************/
/* Output data stored in specified area from host IF                    */
/*  1. Receive stored area specification value from host IF             */
/*       Memory classification 0) x memory 1) y memory                  */
/*       Stored addresses                                               */
/*       Output data count                                              */
/*  2. Output stored data to host IF                                    */
/*                                                                      */
/*  others: r_      [*,*,*,*, , , , ]   dmx,dmy         [ , ]          */
/*          dp_     [*, , , ,*, , , ]   loops/stacks    [1/1]          */
/*          dn_     [ , , , , , , , ]   cycles          ...            */
/************************************************************************/
Sys_IE7 imseg
PutData:
    clr(r2);                                /* Stored parameter introduction */
    clr(r3);                                /*                               */
    r1  = *HDT:x;                           /*  Memory classification */
    r2l = *HDT:x;                           /*  Store addresses */
    r3l = *HDT:x;                           /*  Stored data count */

    if( r3==0 ) ret;                        /* Stored parameter analysis */
    if( r1!=0 ) jmp  PutYdata;

  PutXdata:
    /*---*/         dp0=r2l;                    /* Output from x memory */
    loop r3l {
        /*...*/     r0l=*dp0++;
        *HDT:x = r0l;
    }
    ret;

  PutYdata:
    /*---*/         dp4=r2l;                    /* Output from y memory  */
    loop r3l {
        /*...*/     r0l=*dp4++;
        *HDT:x = r0l;
    }
    ret;
```

```
/*-------------------------------------------------------------------------*/
/*   Function setting definition                                           */
/*-------------------------------------------------------------------------*/
#define Dec                 ; Use decoder function
#define Enc                 ; Use encoder function

#ifdef  Enc
extrn   g729b_InitEncM
extrn   g729b_EncM
#endif

#ifdef  Dec
extrn   g729b_InitDecM
extrn   g729b_DecM
#endif

;public  Encode
;public  Decode

;public  InitG729b

Mchannel_X  xramseg
cur_ch:     ds  1

/*************************************************************************/
/* Input processing of PCM data for Coder                                */
/*************************************************************************/
Ld8k_Ix  imseg
GetPcmData:
    r0l= dp0                        ;
    r0 = r0+2                       ;
    /*---*/             dp1=r0l      ;
    nop                             ;
    /*---*/             r0l=*dp1     ;
    /*---*/             dp1=r0l      ;
    loop L_FRAME {                  ;

        r0l= *HDT:x                 ;
        /*   */         *dp1++ =r0l  ;
    }                               ;
    ret                             ;

/*************************************************************************/
/* Output processing of compression data for Coder                       */
/*************************************************************************/
Ld8k_Ix  imseg
PutPrmData:
    r0l= dp0                        ;
    r0 = r0 + 3                     ;
    /*---*/             dp1=r0l      ;
    nop                             ;
    /*---*/             r1l=*dp1     ;
```

```
    /*---*/             dp1=r1l       ;
    r0 = r0 + 14                       ; dp0 + 3+14 -> dp0 + 0x11
    dp2 = r0l                          ;
    nop                                ;
    r0l = *dp2                         ; Get FTYP
    *HDT:x  = r0l                      ; put FTYP

    loop PRM_SIZE {                    ;
        /*   */         r0l=*dp1++    ;
        *HDT:x = r0l                   ;
    }                                  ;
    ret                                ;

/*************************************************************************/
/* Output processing of PCM data for Decode                              */
/*************************************************************************/
Ld8k_Ix imseg;
PutPcmData:
    r0l= dp0                           ;
    r0 = r0 + 8                        ;
    /*---*/             dp1=r0l       ;
    nop                                ;
    /*---*/             r0l=*dp1      ;
    /*---*/             dp1=r0l       ;
    loop L_FRAME {                     ;
        /*   */         r0l=*dp1++    ;
        *HDT:x = r0l                   ;
    }                                  ;
    ret                                ;

/*************************************************************************/
/* Input processing of compression data for Decode                       */
/*************************************************************************/
Ld8k_Ix imseg
GetPrmData:
    r0l= dp0                           ;
    r0 = r0 + 9                        ;
    /*---*/             dp1=r0l       ;
    nop                                ;
    /*---*/             r0l=*dp1      ;
    /*---*/             dp1=r0l       ;
    loop PRM_SIZE {                    ;
        r0l= *HDT:x                    ;
        /*   */         *dp1++=r0l    ;
    }                                  ;

    ret                                ;


/*************************************************************************/
/* Initialization                                                        */
/*************************************************************************/
```

```
Ld8k_Ix imseg
InitG729b:
    call g729b_MkIO_Table                   ;
#ifdef  Enc
    call    g729b_StartCodec                ;
    /*---*/         dp0=g729b_IO_Table1     ;
    call    g729b_InitEncM                  ;
#endif
#ifdef  Dec
    /*---*/         dp0=g729b_IO_Table1     ;
    call    g729b_InitDecM                  ;
#endif
#ifdef  Enc
    /*---*/         dp0=g729b_IO_Table2     ;
    call    g729b_InitEncM                  ;
#endif
#ifdef  Dec
    /*---*/         dp0=g729b_IO_Table2     ;
    call    g729b_InitDecM                  ;
#endif
#ifdef  Enc
    /*---*/         dp0=g729b_IO_Table3     ;
    call    g729b_InitEncM                  ;
#endif
#ifdef  Dec
    /*---*/         dp0=g729b_IO_Table3     ;
    call    g729b_InitDecM                  ;
#endif

#ifdef  Enc
    /*---*/         dp0=g729b_IO_Table4     ;
    call    g729b_InitEncM                  ;
#endif
#ifdef  Dec
    /*---*/         dp0=g729b_IO_Table4     ;
    call    g729b_InitDecM                  ;
#endif

;    call    InitCycleDt                    ; *
;    call    StMipsDt                       ; *
    ret                                     ;

#ifdef  Enc
/*************************************************************************/
/* Encode 1 frame                                                        */
/*************************************************************************/
Encode:
    r0l= *HDT:x                         ; Get channel No.
    *cur_ch:x=r0l                       ;
    call getCurIO_Table                 ;
    r0l= *HDT:x                         ; Get Vad Mode
    call setVadMode                     ;
```

```
    call IncFrameCount                ;
    call getCurIO_Table               ;
    call GetPcmData                   ;
    call getCurIO_Table               ;
;    call ClrMipsDt                   ; *
    call g729b_EncM                   ;
;    call PutMipsDt                   ; *
    call getCurIO_Table               ;
    call PutPrmData                   ;
  ret                                 ;

/*************************************************************************/
/*  setup VAD flag(Encode)                                               */
/*************************************************************************/
setVadMode:
    r1l= dp0                          ;
    r1 = r1 + 16                      ;
    /*---*/             dp1=r1l      ;
    nop                               ;
    /*---*/             *dp1=r0l     ;
    ret                               ;

/*************************************************************************/
/*  Incriment frame counter(Encode)                                      */
/*************************************************************************/
IncFrameCount:
    r0l= dp0                          ;
    r0 = r0 + 4                       ;
    /*---*/             dp1=r0l      ;
    nop                               ;
    /*...*/             r0 = *dp1    ;
    r0 = r0 sra 16                    ;
    r1 = r0 - 32767                   ;
    if(r1 != 0) jmp $+2               ;
        r0l = 255                     ;
    r0 = r0 + 1                       ;
    /*...*/              *dp1=r0l     ;
    ret                               ;
#else
Encode:
    ret                               ; Dummy function
#endif

#ifdef  Dec
/*************************************************************************/
/*  Decode 1 frame                                                       */
/*************************************************************************/
Decode:
    r0 = *HDT:x                       ; ch No.
    *cur_ch:x = r0h                   ;
    call getCurIO_Table               ;
    r0 = *HDT:x                       ; erase
```

```
    r1 = *HDT:x                     ; ftype
    call DecSetup                   ;

    call GetPrmData                 ;
;    call ClrMipsDt                 ;
    call g729b_DecM                 ;
;    call PutMipsDt                 ;
    call getCurIO_Table             ;
    call PutPcmData                 ;
    ret                             ;

/*************************************************************************/
/*  setup decoder parameter                                              */
/*************************************************************************/
DecSetup:
    r2l= dp0                        ;
    /*---*/             dp1=r2l    ;
    nop                             ;
    /*...*/             r2 =*dp1##12  ;
    /*...*/             *dp1##6=r0h   ;
    /*...*/             *dp1=r1h      ;
    ret                             ;

#else
Decode:
    ret                             ; Dummy function
#endif
/*************************************************************************/
/*  get current IO Control Table pointer                                 */
/*************************************************************************/
;; Step 2-6 (d)
getCurIO_Table:
    r0 = *cur_ch:x                  ;

        clr(r1);
        r1l = g729b_IO_Table2;
        r1 = r1 - g729b_IO_Table1;
;;      r1l = g729b_IO_Table2 - g729b_IO_Table1;
        r1 = r1 sll 16;
        r0 = r0h * r1h;
        r0 = r0 sra 1;
        r0 = r0 + g729b_IO_Table1;
        dp0 = r0l;
        ret;

#if 0
    r0 = r0 sra 16                  ;
    if(r0 != 0) jmp $+3             ;
        dp0=g729b_IO_Table1         ;
        ret                         ;
    r0 = r0 - 1                     ;
    if(r0 != 0) jmp $+3             ;
```

```
        dp0=g729b_IO_Table2           ;
        ret                           ;
    dp0=g729b_IO_Table3               ;
    ret                               ;
#endif


$EJECT
/**/
/************************************************************************/
/* Data for Debug                                                       */
/************************************************************************/
Debug_Ye yramseg
NextAdr:    ds  1
DebDat:     ds  1024

/************************************************************************/
/* Initialization for Debug                                             */
/************************************************************************/
Debug_Ix imseg
IniDebDat:
    r0l= DebDat;
    *NextAdr:y = r0l;
    ret;

/************************************************************************/
/* Debug data output                                                    */
/************************************************************************/
Debug_Ix imseg
PutDebDat:
    clr(r0);
    r0l= *NextAdr:y;
    r0 = r0 - DebDat;
    *HDT:x = r0l;
    if( r0==0 ) ret;
    /*---*/                dp4=DebDat;
    loop r0l {
        /*   */             r0l=*dp4++;
        *HDT:x = r0l;
    }
    r0l= DebDat;
    *NextAdr:y = r0l;
    ret;

end
```

A.2.1 samplebm.asm header file (sysconf.h)

```
/************************************************************************/
/* SPX/evaluation board, application definition file                    */
/************************************************************************/

/*---- Parameter definition ----*/
#define SI1Proc   IntNoProc        /* SIO1(IN)       <Interrupt processing>  */
#define SO1Proc   IntNoProc        /* SIO1(OUT)      <Interrupt processing>  */
#define SI2Proc   IntNoProc        /* SIO2(IN)       <Interrupt processing>  */
#define SO2Proc   IntNoProc        /* SIO2(OUT)      <Interrupt processing>  */
#define HostIn    IntNoProc        /* Host IF input  <Interrupt processing>  */
#define HostOut   IntNoProc        /* Host IF output <Interrupt processing>  */
#define Codec1CK  IntNoProc        /* INT1           <Interrupt processing>  */
#define Codec1FM  IntNoProc        /* INT2           <Interrupt processing>  */
#define Codec2CK  IntNoProc        /* INT3           <Interrupt processing>  */

#define MAX_HIP_NO      3          /* When [P0..2] is host programmable 7    */
                                   /* When [P0..1] is host programmable 3    */

#define HintProc1 NoProc           /* INT4(Vect:1) processing                */
#define HintProc2 NoProc           /* INT4(Vect:2) processing                */
#define HintProc3 NoProc           /* INT4(Vect:3) processing                */
#define HintProc4 NoProc           /* INT4(Vect:4) processing. Use prohibited when
                                      [P0..1] */
#define HintProc5 NoProc           /* INT4(Vect:5) processing. Use prohibited when
                                      [P0..1] */
#define HintProc6 NoProc           /* INT4(Vect:6) processing. Use prohibited when
                                      [P0..1] */
#define HintProc7 NoProc           /* INT4(Vect:7) processing. Use prohibited when
                                      [P0..1] */

#define Com4      NoProc           /* ComID=4      processing                */
#define Com5      NoProc           /* ComID=5      processing                */
#define Com6      Encode           /* ComID=6      processing                */
#define Com7      Decode           /* ComID=7      processing                */
#define Com8      NoProc           /* ComID=8      processing                */
#define Com9      NoProc           /* ComID=9      processing                */
#define Com10     NoProc           /* ComID=10     processing                */
#define Com11     InitG729b        /* ComID=11     processing                */
/* uPD77116 only */
ICR               equ 0x3828       /* Interrupt control register             */


/*---- Register setting value definition ----*/
DWTR_WAIT0        equ 0x0000       /* Wait setting value (data)         0wait*/
DWTR_WAIT1        equ 0x5454       /*                                   1wait*/
DWTR_WAIT3        equ 0xA8A8       /*                                   3wait*/
DWTR_WAIT7        equ 0xFCFC       /*                                   7wait*/
IWTR_WAIT0        equ 0x0000       /* Wait setting value (instruction)  0wait*/
IWTR_WAIT1        equ 0x0054       /*                                   1wait*/
IWTR_WAIT3        equ 0x00A8       /*                                   3wait*/
IWTR_WAIT7        equ 0x00FC       /*                                   7wait*/

PIO0_INP          equ 0x2001       /* General-purpose I/O register setting value   */
PIO0_HIGH         equ 0xF001
PIO0_LOW          equ 0xB001
PIO1_INP          equ 0x2002
PIO1_HIGH         equ 0xF102
```

```
PIO1_LOW          equ 0xB102
PIO2_INP          equ 0x2004
PIO2_HIGH         equ 0xF204
PIO2_LOW          equ 0xB204
PIO3_INP          equ 0x2008
PIO3_HIGH         equ 0xF308
PIO3_LOW          equ 0xB308
PIO_ALL_IN        equ 0x200F

HST_WAIT          equ 0x0400       /* Host status register setting value       */
HST_NOWAIT        equ 0x0000
HST_MASK          equ 0x00FF
HST_LENMASK       equ 0x0001
HST_SENMASK       equ 0x0002
HST_ERRMASK       equ 0x003C

SST_WAIT          equ 0x0F00       /* Serial status register setting value     */
SST_NOWAIT        equ 0x0300
SST_HALT          equ 0x0000
SST_ERRMASK       equ 0x000C
SST_LENMASK       equ 0x0001
SST_SENMASK       equ 0x0002
INTFLG_INT1       equ 0x0001       /* Interrupt register setting value         */
INTFLG_INT2       equ 0x0002
INTFLG_INT3       equ 0x0004
INTFLG_INT4       equ 0x0008
INTFLG_SI1        equ 0x0010
INTFLG_SO1        equ 0x0020
INTFLG_SI2        equ 0x0040
INTFLG_SO2        equ 0x0080
INTFLG_HI         equ 0x0100
INTFLG_HO         equ 0x0200

/*---- External bus definition ----*/

/*---- Function entry ----*/
HostReBoot        equ 0x0005       /* Host reboot                              */

$LIST
```

[MEMO]

[MEMO]

NEC

Facsimile Message

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.

From:

Name

Company

Tel. FAX

Address

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: +55-11-6462-6829

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

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

Taiwan
NEC Electronics Taiwan Ltd.
Fax: 02-2719-5951

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

Japan
NEC Semiconductor Technical Hotline
Fax: 044-435-9608

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	❑	❑	❑	❑