

## **General Notice**

When using this document, keep the following in mind:

1. This document is confidential. By accepting this document you acknowledge that you are bound by the terms set forth in the nondisclosure and confidentiality agreement signed separately and in the possession of SEGA. If you have not signed such a nondisclosure agreement, please contact SEGA immediately and return this document to SEGA.
2. This document may include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new versions of the document. SEGA may make improvements and/or changes in the product(s) and/or the program(s) described in this document at any time.
3. No one is permitted to reproduce or duplicate, in any form, the whole or part of this document without SEGA's written permission. Request for copies of this document and for technical information about SEGA products must be made to your authorized SEGA Technical Services representative.
4. No license is granted by implication or otherwise under any patents, copyrights, trademarks, or other intellectual property rights of SEGA Enterprises, Ltd., SEGA of America, Inc., or any third party.
5. Software, circuitry, and other examples described herein are meant merely to indicate the characteristics and performance of SEGA's products. SEGA assumes no responsibility for any intellectual property claims or other problems that may result from applications based on the examples described herein.
6. It is possible that this document may contain reference to, or information about, SEGA products (development hardware/software) or services that are not provided in countries other than Japan. Such references/information must not be construed to mean that SEGA intends to provide such SEGA products or services in countries other than Japan. Any reference of a SEGA licensed product/program in this document is not intended to state or imply that you can use only SEGA's licensed products/programs. Any functionally equivalent hardware/software can be used instead.
7. SEGA will not be held responsible for any damage to the user that may result from accidents or any other reasons during operation of the user's equipment, or programs according to this document.

NOTE: A reader's comment/correction form is provided with this document. Please address comments to :

SEGA of America, Inc., Developer Technical Support (att. Evelyn Merritt)  
150 Shoreline Drive, Redwood City, CA 94065

SEGA may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.



SEGA OF AMERICA, INC.  
Consumer Products Division

# Program Library User's Guide 3

Doc. # ST-135-R4-092295

## READER CORRECTION/COMMENT SHEET

### Keep us updated!

If you should come across any incorrect or outdated information while reading through the attached document, or come up with any questions or comments, please let us know so that we can make the required changes in subsequent revisions. Simply fill out all information below and return this form to the Developer Technical Support Manager at the address below. Please make more copies of this form if more space is needed. Thank you.

### General Information:

Your Name \_\_\_\_\_ Phone \_\_\_\_\_

Document number ST-135-R4-092295 Date \_\_\_\_\_

Document name Program Library User's Guide 3

### Corrections:

Chpt.	pg. #	Correction

Questions/comments: \_\_\_\_\_

---

---

---

### Where to send your corrections:

Fax: (415) 802-1717  
Attn: Evelyn Merritt,  
Developer Technical Support

Mail: SEGA OF AMERICA  
Attn: Evelyn Merritt,  
Developer Technical Support  
150 Shoreline Dr.  
Redwood City, CA 94065

---

## Table of Contents

<b>Sound Interface Library.....</b>	<b>6</b>
1.0 Guide .....	6
1.1 Objective .....	6
1.2 Overview .....	6
1.3 Sound System Startup .....	6
1.4 Performance Setup .....	7
1.5 Sound Control .....	7
1.6 Status Acquisition .....	8
2.0 Reference .....	9
2.1 Data List .....	9
2.2 Data Specifications .....	10
2.3 List of Functions .....	20
2.4 Function Specifications .....	21
<b>DMA Library.....</b>	<b>32</b>
1.0 Guide .....	32
1.1 Objective .....	32
1.3 Precautions .....	32
1.4 Calling Sequence .....	33
2.0 Reference .....	34
2.1 Data List .....	34
2.2 Data Specifications .....	35
2.3 List of Functions .....	41
2.4 Function Specifications .....	42
<b>Cache Library.....</b>	<b>50</b>
1.0 Guide .....	50
1.1 Objective .....	50
1.2 Overview .....	50
2.0 Reference .....	51
2.1 List of Functions .....	51
2.2 Function Specifications .....	51
2.3 List of Function Format Macros .....	52
2.4 Function Format Macro Specification .....	52
<b>Interrupt Management Library.....</b>	<b>55</b>
1.0 Guide .....	55
1.1 Objective .....	55
1.2 Overview .....	55
1.3 Overview of Functions .....	55
1.4 Usage Conditions .....	55
1.5 Calling Sequence .....	56
2.0 Reference .....	57
2.1 Data List .....	57
2.2 Data Specifications .....	58
2.3 List of Functions .....	61
2.4 Function Specifications .....	62

---

<b>Memory Management Library.....</b>	<b>66</b>
1.0 Guide .....	66
1.1 Objective .....	66
1.2 Overview .....	66
1.3 Calling Sequence .....	66
2.0 Reference .....	67
2.1 List of Functions .....	67
2.2 Function Specifications .....	67
<b>Timer Library.....</b>	<b>69</b>
1.0 Guide .....	69
1.1 Objective .....	69
1.2 Overview .....	69
1.3 Detailed Information .....	70
2.0 Reference .....	72
2.1 List of Function Format Macros .....	72
2.2 Function Specifications .....	73
<b>Debug Support Library.....</b>	<b>81</b>
1.0 Guide .....	81
1.1 Objective .....	81
1.2 Description .....	81
1.3 Coding Example .....	81
2.0 Reference .....	82
2.1 List of Functions .....	82
2.2 Function Specifications .....	82
<b>Compression Decompression Library.....</b>	<b>86</b>
Terminology .....	86
1.0 Guide .....	87
1.1 Application .....	87
1.2 Compression Method .....	87
1.3 Decompression Library .....	92
2.0 Reference .....	96
2.1 Data Specifications .....	96
2.2 List of Functions .....	97
2.3 Function Specifications .....	98
<b>PCM-ADPCM Playback Library.....</b>	<b>102</b>
1.0 Overview .....	102
1.1 Objective .....	102
1.2 Features .....	102
1.3 System Image .....	103
2.0 Specifications .....	104
3.0 Playback Sequence .....	107
4.0 Programming Precautions .....	109
5.0 Data Specifications .....	111
5.1 List of Data .....	111
5.2 Data Details .....	112
6.0 Functions Specifications .....	118
6.1 List of Functions .....	118
6.2 Function Details .....	120
<b>INDEX .....</b>	<b>131</b>

---

---

# Sound Interface Library

## 1.0 Guide

### 1.1 Objective

The Sound Interface Library interfaces the sound sub-system and the main hardware system (including the CD system), and enables control over sound playback and related processes.

### 1.2 Overview

The sound sub-system controls the playback of song sequences, PCM audio and CD audio. The sound system must be activated before the sound playback can be controlled. Sound data must then be transferred to the sound memory as a performance setup for song sequences and PCM audio. The resulting sound control is monitored by status information provided by the library. This chapter explains the details of the Sound Interface Library in the order shown in the following list. For more information, see also the sound driver system interface in the *Sound Development Manual* (ST-081-R5-062894).

- Sound system startup
- Performance setup
- Sound control
- Status acquisition

### 1.3 Sound System Startup

The sound system must be started at power on. The calling sequence is as follows.

```
void sndStart ()  
{  
    SndIniDt sys_ini; /* data area for system startup */  
  
    SND_INI_PRG_ADR(sys_ini) = (Uint16 *) 0x22002400 ;  
        /* set 68K program area start address */  
    SND_INI_PRG_SZ(sys_ini) = (Uint16 *) 0x4fc8 ;  
        /* set 68K program size */  
    SND_INI_ARA_ADR(sys_ini) = (Uint16 *) 0x22004400 ;  
        /* set sound area map area start address */  
    SND_INI_ARA_SZ(sys_ini) = 0x0550 ;  
        /* set sound area map size (word specified) */  
    SND_Init (&sys_ini) ; /* startup sound system */  
    ...  
}
```

---

## 1.4 Performance Setup

### Sequence

Sequence data (song data, sound effect data), executed by sound control functions, must be transferred to the sound memory. Transfers must be performed according to the current sound area map. (For more information about the sound area map, see the *Sound Development Manual*.) The calling sequence is shown in the *Sound Control* section below.

### PCM

Refer to the information on the sound driver system interface in the *Sound Development Manual* for setting up PCM performances.

---

## 1.5 Sound Control

### Functions

The sound control function writes commands to the sound memory. There are eight command buffers. The sound control function can be used without the need to monitor the command buffer.

- Sequence: Enables control such as playback start/end and volume settings over song sequences.
- PCM: Enables playback start and end of PCM data.
- CD: Enables control such as volume/pan settings, volume analysis, and effect changes over CD audio data.

### Calling Sequence

The following calling sequence is used until the start of a song sequence.

```
void sndCnt()
{
    sndStart;          /* sound system startup (see item above)      */
    SND_ChgMap(2);    /* change sound area map                      */
    SND_MoveData((Uint16 *)0x22005000, 0xffff, SND_KD_SEQ, 2);
                    /* sound data transfer (sequence)           */
    SND_MoveData((Uint16 *)0x22005500, 0xffff, SND_KD_TONE, 2);
                    /* transfer sound data (tone)            */
    SND_SetT1V1(15); /* Set master volume                         */
    SND_StartSeq(0, 2, 5, 0); /* start sequence                           */
    . . .
}
```

---

## 1.6 Status Acquisition

### Functions

The following provides an overview of functions by sound data type:

- Sequence: Enables acquisition of current status (play, pause, etc.).
- PCM: Enables acquisition of current PCM data address being used.
- CD: Enables acquisition of volume analysis results.

### Calling Sequence

The following calling sequence starts another sequence under sound control number 0 if a sound control number 0 sequence has stopped.

```
void sndHantei()
{
    SndSeqStat status;                      /* define sequence status area */
    .
    .
    SND_GetSeqStat (&status, 0);           /* get sequence status */
    if(SND_SEQ_STAT_MODE(status) == SND_MD_STOP){
        /* is sequence under sound control number 0 stopped? */
        SND_StartSeq (0, 2, 6, 0);         /* start sequence */
    }
}
```

---

## 2.0 Reference

### 2.1 Data List

	Data Type	Data Name	Number
Common	System startup data type	SndIniDt	1
	Sound area map number data type	SndAreaMap	2
	Total volume data type	SndTlVl	3
	Effect bank number data type	SndEfctBnkNum	4
	Tone bank number data type	SndToneBnkNum	5
	Mixer number data type	SndMixBnkNum	6
	Effect out select data type	SndEfctOut	7
	Level data type	SndLev	8
	Pan data type	SndPan	9
	Command execution status data type	SndRet	10
	Hardware check parameter data type	SndHardPrm	11
	Hardware check status data type	SndHardStat	12
Sequence	Sound control number data type	SndSeqNum	13
	Sequence bank number data type	SndSeqBnkNum	14
	Sequence song number data type	SndSeqSongNum	15
	Priority level data type	SndSeqPri	16
	Sequence volume data type	SndSeqVl	17
	Fade rate data type	SndFade	18
	Tempo data type	SndTempo	19
	Sequence status data type	SndSeqStat	20
	Sound control no. play position data type	SndSeqPlayPos	21
PCM	PCM start parameter data type	SndPcmStPrm	22
	PCM change parameter data type	SndPcmChgPrm	23
	PCM play address data type	SndPcmPlayAdr	24
	PCM address update interrupt status data type	SndPcmIntStat	25
CD	Volume analysis data type	SndCdVlAnl	26
	Discrete frequency band stereo volume analysis data type	SndCdHzSrVl	27

## 2.2 Data Specifications

- **Common**

Title	Data	Data Name	No.
Data Specifications	System start data type	SndIniDt	1

This data type indicates the transfer source of data required by the sound system. The transfer destination is the sound memory (fixed). Set each of the following:

- Address  
Refer to the link map file and set the address.
- Size  
Set the size of the 68K program object file and the sound area map object file.

### Access Macro

SndIniDt sys\_ini

Access Macro	Type	Description
SND_INI_PRG_ADR (sys_ini)	Uint16*	68K program start address
SND_INI_PRG_SZ (sys_ini)	Uint16	68K program start size
SND_INI_ARA_ADR (sys_ini)	Uint16*	Sound area map start address
SND_INI_ARA_SZ (sys_ini)	Uint16	Sound area map size (word specified)

Title	Data	Data Name	No.
Data Specifications	Sound area map data type	SndAreaMap	2

This data type displays the sound area map.

### Description of Values

Value	Description
0 ~ 255	Arbitrary

---

Title	Data	Data Name	No.
Data Specifications	Total volume data type	SndTlV1	3

This data type denotes the total volume.

**Description of Values**

Value	Description
0 ~15	0 is OFF, 15 is the maximum

Title	Data	Data Name	No.
Data Specifications	Effect bank number data type	SndEfctBnkNum	4

This data type indicates the effect bank number.

**Description of Values**

Value	Description
0 ~15	Arbitrary

Title	Data	Data Name	No.
Data Specifications	Sound tone bank number data type	SndToneBnkNum	5

This data type denotes the tone bank number.

**Description of Values**

Value	Description
0 ~15	Arbitrary

Title	Data	Data Name	No.
Data Specifications	Mixer number data type	SndMixBnkNum	6

This data type shows the mixer number.

**Description of Values**

Value	Description
0 ~127	Arbitrary

---

<b>Title</b>	<b>Data</b>	<b>Data Name</b>	<b>No.</b>
Data Specifications	Effect out select data type	SndEfctOut	7

This data type denotes the effect out selection setting.

**Description of Values**

Value	Description
0 ~ 15	Arbitrary

<b>Title</b>	<b>Data</b>	<b>Data Name</b>	<b>No.</b>
Data Specifications	Level data type	SndLev	8

This data type denotes the level.

**Description of Values**

Value	Description
0 ~ 7	Arbitrary

<b>Title</b>	<b>Data</b>	<b>Data Name</b>	<b>No.</b>
Data Specifications	Pan data type	SndPan	9

This data type shows pan. To pan the sound from left to right, increment up from - 15.

**Description of Values**

↑, ↓ : Adjust the volume in the direction of the arrow.

Value	Description	
	Left	Right
- 15	Maximum	OFF
-14 to -1	Maximum	↓
0	Maximum	Maximum
0 to 14	↑	Maximum
15	OFF	Maximum

---

Title	Data	Data Name	No.
Data Specifications	Command execution status data type	SndRet	10

This data type indicates the command execution condition.

Constant	Description
SND_RET_SET	Command normal end (command was set)
SND_RET_NSET	Command abnormal end (command was not set)

Title	Data	Data Name	No.
Data Specifications	Hardware check parameter data type	SndHardPrm	11

This data type denotes the hardware check parameters.

Constant	Description
SND_DRAM4	DRAM 4 Mbit read/write
SND_DRAM8	DRAM 8 Mbit read/write
SND_SCSP_MIDI	SCSP MIDI
SND_SOUND_SRC_LR	Sound generator output (L/R)
SND_SOUND_SRC_L	Sound generator output (L)
SND_SOUND_SRC_R	Sound generator output (R)

Title	Data	Data Name	No.
Data Specifications	Hardware check status data type	SndHardStat	12

This data type denotes the hardware check status.

Constant	Description
SND_HARD_OK	Normal
SND_HARD_ERR	Abnormal

---

- **Sequence**

Title	Data	Data Name	No.
Data Specifications	Sound control number data type	SndSeqNum	13

This data type indicates the sound control number.

**Description of Values**

Value	Description
0 ~7	Arbitrary

Title	Data	Data Name	No.
Data Specifications	Sequence bank number data type	SndSeqBnkNum	14

This data type denotes the sequence bank number.

**Description of Values**

Value	Description
0 ~15	Arbitrary

Title	Data	Data Name	No.
Data Specifications	Sequence song number data type	SndSeqSongNum	15

This data type denotes the sequence song number.

**Description of Values**

Value	Description
0 ~255	Arbitrary

Title	Data	Data Name	No.
Data Specifications	Priority level data type	SndSeqPri	16

This data type denotes the priority level.

**Description of Values**

Value	Description
0 ~7	0 is the highest priority level and 7 is the minimum

---

Title	Data	Data Name	No.
Data Specifications	Sequence volume data type	SndSeqVI	17

This data type denotes the sequence volume.

**Description of Values**

Value	Description
0 ~127	0 is off, 127 is the maximum (source sound level)

Title	Data	Data Name	No.
Data Specifications	Fade rate data type	SndFade	18

This data type denotes the fade rate.

**Description of Values**

Value	Description
0	Fade rate is disabled
1 ~255	Longest at 1, shortest at 255

Title	Data	Data Name	No.
Data Specifications	Tempo data type	SndTempo	19

This data type denotes the tempo.

**Description of Values**

Value	Description
+32767 ~ -32768	Relative value in terms of the present value + is Up and - is Down Up 2X at +4096, Down 2X at -4096

---

Title	Data	Data Name	No.
Data Specifications	Sequence status data type	SndSeqStat	20

This data type indicates the sequence status.

**SndSeqStat Status**

Access Macro	Type	Description
SND_SEQ_STAT_MODE(status)	Uint16	Song mode
SND_SEQ_STAT_STAT(status)	Uint16	Song status

**SND\_SEQ\_STAT\_MODE(status)**

Constant	Description
SND_MD_STOP	Stop
SND_MD_PLAY	Play
SND_MD_FADE	Fade
SND_MD_PLAY_PS	Play pause
SND_MD_FADE_PS	Fade pause

**SND\_SEQ\_STAT\_STAT(status)**

Value (Hexadecimal)	Description
00	Normal
01 ~ 7F	Error code
80 ~ FF	Timing flag

Title	Data	Data Name	No.
Data Specifications	Sound control number play position data type	SndSeqPlayPos	21

This data type denotes the sound control number play position.

**Description of Values**

Value (Hexadecimal)	Description
0 ~FFFF	0 ~FFFF

- PCM

Title	Data	Data Name	No.
Data Specifications	PCM start parameter data type	SndPcmStartPrm	22

This data type denotes the PCM start parameters.

**SndPcmStPrm prm**

Access Macro	Type	Description
SND_PRM_MODE(prm)	Uint8	Stereo and mono + sampling rate
SND_PRM_SADR(prm)	Uint16	PCM stream buffer start address
SND_PRM_SIZE(prm)	Uint16	PCM stream buffer size
SND_PRM_OFFSET(prm)	Uint8	PCM stream play start offset

**SND\_PRM\_MODE(prm)**

Specify the following constants by logical OR.

Bit Position	Constant	Description
Mono/stereo select bit	SND_MD_MONO	Mono
	SND_MD_STEREO	Stereo
Sampling rate selection bit	SND_MD_16	16-bit PCM
	SND_MD_8	8-bit PCM

**SND\_PRM\_SADR(prm)**

Value (Hexadecimal)	Description
0000 ~ FFFF	PCM stream buffer start address (upper 16 bits of 20 bit data)

**SND\_PRM\_SIZE(prm)**

Value (Hexadecimal)	Description
0000 ~ FFFF	PCM stream buffer size (number of samples for 1 channel)

**SND\_PRM\_OFFSET(prm)**

Value (Hexadecimal)	Description
0 ~ F (1000 [4k sample] units)	PCM stream play start offset (0000 ~ F000)

---

Title	Data	Data Name	No.
Data Specifications	PCM change parameter data type	SndPcmChgPrm	23

This data type denotes the PCM change parameters.

#### SndPcmChgPrm prm

Access Macro	Type	Description	
SND_PRM_NUM(prm)	SndPcmNum	PCM stream play number	
SND_PRM_LEV(prm)	SndLev	Direct sound level	
SND_PRM_PAN(prm)	SndPan	Direct sound pan	
SND_PRM_PICH(prm)	Uint8	PICH word	
SND_R_EFCT_IN(prm)	SndEfctIn	Effect in select	Right output (mono)
SND_R_EFCTLEV(prm)	SndLev	Effect send level	
SND_L_EFCT_IN(prm)	SndEfctIn	Effect in select	Left output (invalid for mono)
SND_L_EFCTLEV(prm)	SndLev	Effect send level	

The following constants and values can be used by each access macro.

#### SND\_PRM\_NUM(prm)

Value	Description
0 ~7	PCM stream play number

#### SND\_PRM\_PICH(prm)

Refer to the PICH register section in the *Saturn SCSP User's Manual* (ST-077-R2-052594).

Title	Data	Data Name	No.
Data Specifications	PCM play address data type	SndPcmPlayAddr	24

This data type denotes the PCM play address.

#### SndPcmPlayAddr (prm)

Access Macro	Type	Description
SND_PCM_RADR(prm)	Uint8	Right output (mono)
SND_PCM_LADR(prm)	Uint8	Left output

Title	Data	Data Name	No.
Data Specifications	PCM address update interrupt status data type	SndPcmIntStat	25

This data type indicates the PCM address update interrupt status. The following bit position constants can be specified by OR.

Bit Position Constant	Description
SND_PCM_PLAY0	PCM playback number 0
SND_PCM_PLAY1	PCM playback number 1
SND_PCM_PLAY2	PCM playback number 2
SND_PCM_PLAY3	PCM playback number 3
SND_PCM_PLAY4	PCM playback number 4
SND_PCM_PLAY5	PCM playback number 5
SND_PCM_PLAY6	PCM playback number 6
SND_PCM_PLAY7	PCM playback number 7

- **CD**

Title	Data	Data Name	No.
Data Specifications	Volume analysis volume data type	SndCdV1Anl	26

This data type denotes the analysis volume.

**Description of Values**

Value	Description
0 ~32767	0 is OFF, 32767 is maximum

Title	Data	Data Name	No.
Data Specifications	Discrete frequency band stereo volume analysis data type	SndCdHzSrV1	27

This data type denotes the discrete frequency band stereo volume analysis data. In the case of monaural, the same values are entered in left and right outputs.

**SndCdHzSrV1 hz\_v1**

Access Macro	Type	Description	
SND_CD_LHIGH (hz_v1)	SndCdV1Anl	Treble analysis volume	Left output
SND_CD_LMID (hz_v1)	SndCdV1Anl	Midrange analysis volume	
SND_CD_LLLOW (hz_v1)	SndCdV1Anl	Bass analysis volume	
SND_CD_RHIGH (hz_v1)	SndCdV1Anl	Treble analysis volume	Right output
SND_CD_RMID (hz_v1)	SndCdV1Anl	Midrange analysis volume	
SND_CD_RLOW (hz_v1)	SndCdV1Anl	Bass analysis volume	

## 2.3 List of Functions

		Function	Name	No.
Start sound system		Start sound system	SND_Init	1
Interrupt		Set interrupt enable	SND_SET_ENA_INT	2
		Get interrupt enable	SND_GET_ENA_INT	3
		Set interrupt source	SND_SET_FCT_INT	4
		Reset interrupt	SND_RESET_INT	5
Performance setup		Transfer sound data	SND_MoveData	6
Sound control	Common	Change sound area map	SND_ChgMap	7
		Set total volume	SND_SetTlVl	8
		Change effect	SND_ChgEfct	9
		Change mixer	SND_ChgMix	10
		Change mixer parameters	SND_ChgMixPrm	11
		Check hardware	SND_ChkHard	12
	Sequence	Start sequence	SND_StartSeq	13
		Stop sequence	SND_StopSeq	14
		Pause sequence	SND_PauseSeq	15
		Continue sequence	SND_ContSeq	16
		Set sequence volume	SND_SetSeqVl	17
		Change tempo	SND_ChgTempo	18
		Direct MIDI control	SND_CtrldirMidi	19
	PCM	Start PCM	SND_StartPcm	20
		Stop PCM	SND_StopPcm	21
		PCM change	SND_ChgPcm	22
	CD	Start volume analysis	SND_StartVlAnl	23
		Stop volume analysis	SND_StopVlAnl	24
		Set CD-DA level	SND_SetCdDaLev	25
		Set CD-DA pan	SND_SetCdDaPan	26
Status Acquisition	Sequence	Get sequence status	SND_GetSeqStat	27
		Get sound control number play position	SND_GetSeqPlayPos	28
	PCM	Get PCM execute address	SND_GetPcmPlayAddr	29
		Get PCM address update interrupt status	SND_GET_INT_STAT	30
	CD	Get total stereo volume analysis	SND_GetAnlTlVl	31
		Get discrete frequency band stereo volume analysis	SND_GetAnlHzVl	32

## 2.4 Function Specifications

- Start Sound System

Title	Function	Function Name	No.
Function Specifications	Start sound system	SND_Init	1

**Format:** void SND\_Init(SndIniDt \*sys\_ini)  
**Input:** sys\_ini : Data for system startup  
**Output:** None  
**Function Value:** None  
**Function:** The sound system is started up based on the specified system startup data. Startup occurs after the program and sound area map data are transferred and the hardware registers are initialized. For more details, see the *Sound Driver System Interface* section of the *Sound Development Manual*.  
**Remarks:** SCU DMA transfer is used for these transfers.

- Interrupt

Title	Function	Function Name	No.
Function Specifications	Set interrupt enable	SND_SET_ENA_INT	2

**Format:** void SND\_SET\_ENA\_INT(UINT8 data)  
**Input:** data: Interrupt enable bit (1= enabled, 0= disabled)

### Constant

Bit Position Constant	Description
SND_INT_PCM_ADR	PCM address interrupt

**Output:** None  
**Function Value:** None  
**Function:** Enables interrupts. This setting enables or disables the interrupt signal from the sound system. This setting does not mask SCU interrupts.

Title	Function	Function Name	No.
Function Specifications	Get interrupt enable	SND_GET_ENA_INT	3

**Format:** `Uint8 SND_GET_ENA_INT(void)`

**Input:** None

**Output:** Interrupt enable bit (1= enabled, 0= disabled)

#### Constant

Bit Position Constant	Description
<code>SND_INT_PCM_ADR</code>	PCM address interrupt

**Function Value:** None

**Function:** Gets interrupt enable.

Title	Function	Function Name	No.
Function Specifications	Set interrupt source	SND_SET_FCT_INT	4

**Format:** `Uint8 SND_SET_FCT_INT(void)`

**Input:** None

**Output:** Interrupt enable bit

#### Constant

Bit Position Constant	Description
<code>SND_FCT_PCM_ADR</code>	PCM address update

**Function Value:** None

**Function:** Sets the interrupt source.

Title	Function	Function Name	No.
Function Specifications	Reset interrupt	SND_RESET_INT	5

**Format:** `void SND_RESET_INT(void)`

**Input:** None

**Output:** None

**Function Value:** None

**Function:** Resets interrupt. Execute after a sound interrupt occurs. If this function is not executed, the interrupt remains asserted.

---

- **Performance Setup**

Title	Function	Function Name	No.
Function Specifications	Transfer sound data	SND_MoveData	6

**Format:** void SND\_MoveData(Uint16 \*source, Uint32 size, Uint16 data\_kind, Uint16 data\_no)

**Input:**

source	: Sound data transfer source address
size	: Transfer size (byte units)
data_kind	: Data type
data_no	: Data number

#### Constant

Bit Position Constant	Description
SND_KD_TONE	Tone bank data
SND_KD_SEQ	Sequence data
SND_KD_DSP_PRG	DSP program
SND_KD_DSP_RAM	DSP work RAM

**Output:** None

**Function Value:** None

**Function:** Transfers sound data to the sound memory according to the map information that corresponds to the specified data type and number.

**Remarks:** SCU DMA transfer is used in transferring data.

---

- **Sound Control**

### Common

Title	Function	Function Name	No.
Function Specifications	Change sound area map	SND_ChgMap	7

**Format:** SndRet SND\_ChgMap(SndAreaMap area\_no)  
**Input:** area\_no : Sound area map number  
**Output:** None  
**Function Value:** Command execution status  
**Function:** Changes the current sound area map to the sound area map indicated by the sound area map number.

Title	Function	Function Name	No.
Function Specifications	Set total volume	SND_SetTlV1	8

**Format:** SndRet SND\_SetVlTl(SndTlV1 vol)  
**Input:** vol : Total volume  
**Output:** None  
**Function Value:** Command execution status  
**Function:** Changes the total volume.  
**Remarks:** Affects only sequence and PCM.

Title	Function	Function Name	No.
Function Specifications	Change effect	SND_ChgEfct	9

**Format:** SndRet SND\_ChgEfct(SndEfctBnkNum efct\_no)  
**Input:** efct\_no : Effect bank number  
**Output:** None  
**Function Value:** Command execution status  
**Function:** Executes the DSP program denoted by the effect bank number.

Title	Function	Function Name	No.
Function Specifications	Change mixer	SND_ChgMix	10

**Format:** SndRet SND\_ChgMix(SndToneBnkNum tone\_no, SndMixBnkNum mix\_no)  
**Input:** tone\_no : Tone bank number  
mix\_no : Mixer number  
**Output:** None  
**Function Value:** Command execution status  
**Function:** Changes mixers.

---

Title	Function	Function Name	No.
Function Specifications	Change mixer parameters	SND_ChgMixPrm	11

**Format:** SndRet SND\_ChgMixPrm(SndEfctOut efct\_out SndLev level,  
SndPan pan)  
**Input:** efct\_out : Effect out select  
level : Effect return level  
pan : Effect PAN  
**Output:** None  
**Function Value:** Command execution status  
**Function:** Changes mixer parameters.

Title	Function	Function Name	No.
Function Specifications	Check hardware	SND_ChkHard	12

**Format:** SndRet SndChkHard(SndHardStat \*stat, SndHardChk prm)  
**Input:** prm : Hardware check parameters  
**Output:** stat : Hardware check status  
**Function Value:** Command execution status  
**Function:** Checks hardware according to hardware parameters.

---

## Sequence

Title	Function	Function Name	No.
Function Specifications	Start sequence	SND_StartSeq	13

**Format:** SndRet SND\_StartSeq(SndSeqNum seq\_no, SndSeqBnkNum seq\_bk\_no, SndSeqSongNum song\_no, SndSeqPri pri\_lev)

**Input:** seq\_no : Sound control number  
seq\_bk\_no : Sequence bank number  
song\_no : Sequence song number  
pri\_lev : Priority level

**Output:** none

**Function Value:** Command execution status

**Function:** Starts the specified sequence.

Title	Function	Function Name	No.
Function Specifications	Stop sequence	SND_StopSeq	14

**Format:** SndRet SND\_StopSeq(SndSeqNum seq\_no)

**Input:** seq\_no : Sound control number

**Output:** None

**Function Value:** Command execution status

**Function:** Stops the specified sequence.

Title	Function	Function Name	No.
Function Specifications	Pause sequence	SND_PauseSeq	15

**Format:** SndRet SND\_PauseSeq(SndSeqNum seq\_no)

**Input:** seq\_no : Sound control number

**Output:** None

**Function Value:** Command execution status

**Function:** Pauses the selected sequence.

Title	Function	Function Name	No.
Function Specifications	Continue sequence	SND_ContSeq	16

**Format:** SndRet SND\_ContSeq(SndSeqNum seq\_no)

**Input:** seq\_no : Sound control number

**Output:** None

**Function Value:** None

**Function:** Cancels pause of the selected sequence.

Title	Function	Function Name	No.
Function Specifications	Set sequence volume	SND_SetSeqV1	17

**Format:** SndRet SND\_SetSeqV1(SndSeqNum seq\_no, SndSeqV1 seq\_vl,  
                  SndFade fade)

**Input:** seq\_no : Sound control number  
seq\_lev : Fade level  
fade : Fade rate

**Output:** None

**Function Value:** Command execution status

**Function:** Sets the sequence volume.

Title	Function	Function Name	No.
Function Specifications	Change tempo	SND_ChgTempo	18

**Format:** SndRet SND\_ChgTempo(SndSeqNum seq\_no, SndTempo tempo)

**Input:** seq\_no : Sound control number  
tempo : Tempo

**Output:** None

**Function Value:** Command execution status

**Function:** Sets the tempo of the selected sequence.

Title	Function	Function Name	No.
Function Specifications	Direct MIDI control	SND_CtrlDirMidi	19

**Format:** SndRet SND\_CtrlDirMidi(SndSeqNum seq\_no, SndSeqPri seq\_pri,  
                  Uint8 md\_com, Uint8 ch, Uint8 dt1, Uint8 dt2)

**Input:** seq\_no : Sound control number  
seq\_pri : MIDI command (0H~7H)  
ch : MIDI channel (0H~1FH)  
dt1 : MIDI data1 (00H~7FH)  
dt2 : MIDI data2 (00H~7FH)

**Output:** None

**Function Value:** Command execution conditions

**Function:** MIDI is controlled directly according to the selected parameters.

---

## PCM

Title	Function	Function Name	No.
Function Specifications	Start PCM	SND_StartPcm	20

**Format:** SndRet SND\_StartPcm(SndPcmStartPrm \*sprm, SndPcmChgPrm \*cprm)  
**Input:** sprm : PCM start parameter pointer  
cprm : PCM change parameter pointer  
**Output:** None  
**Function Value:** Command execution status  
**Function:** Plays PCM data according to the specified parameters.

Title	Function	Function Name	No.
Function Specifications	Stop PCM	SND_StopPcm	21

**Format:** SndRet SND\_StopPcm(SndPcmNum pcm\_num)  
**Input:** pcm\_num : Play stop PCM stream play number  
**Output:** None  
**Function Value:** Command execution status  
**Function:** Stops play of PCM data.

Title	Function	Function Name	No.
Function Specifications	PCM Change	SND_ChgPcm	22

**Format:** SndRet SND\_ChgPcm(SndPcmChgPrm \*cpcm)  
**Input:** cpcm : PCM change parameter pointer  
**Output:** None  
**Function Value:** Command execution status  
**Function:** Changes the playback status of PCM data.

---

## CD

Title	Function	Function Name	No.
Function Specifications	Start Volume Analysis	SND_StartVlAnl	23

**Format:** SndRet SND\_StartVlAnl(void)  
**Input:** None  
**Output:** None  
**Function Value:** Command execution status  
**Function:** Starts analysis of the frequency band volume and total volume. Volume analysis continues until it is stopped by the volume analysis stop function. Before analyzing the frequency band volumes with this function, the special DSP program for this task must be executed by effect change. The DSP program is not required when analyzing the total volume.  
**Remarks:** Other DSP programs cannot be executed while the frequency band volume analysis DSP program is being executed.

Title	Function	Function Name	No.
Function Specifications	Stop Volume Analysis	SND_StopVlAnl	24

**Format:** SndRet SND\_StopVlAnl(void)  
**Input:** None  
**Output:** None  
**Function Value:** Command execution status  
**Function:** Stops volume analysis.

Title	Function	Function Name	No.
Function Specifications	Set CD-DA Level	SND_SetCdDaLev	25

**Format:** SndRet SND\_SetCdDaLev(SndLev left, SndLev right)  
**Input:** left : Volume of left output  
right : Volume of right output  
**Output:** None  
**Function Value:** Command execution status  
**Function:** Changes the current stereo volume.

Title	Function	Function Name	No.
Function Specifications	Set CD-DA Pan	SND_SetCdDaPan	26

**Format:** SndRet SND\_SetCdDaPan(SndPan left, SndPan right)  
**Input:** left : Pan of left output  
right : Pan of right output  
**Output:** None  
**Function Value:** Command execution status  
**Function:** Changes the current stereo pan.

---

- **Status Acquisition**

### Sequence

Title	Function	Function Name	No.
Function Specifications	Get sequence status	SND_GetSeqStat	27

**Format:** void SND\_GetSeqStat(SndSeqStat \*status, SndSeqNum seq\_no)  
**Input:** seq\_no : Sound control number  
**Output:** status : Sequence status pointer  
**Function Value:** None  
**Function:** Gets the sequence status of the selected sound control number.

Title	Function	Function Name	No.
Function Specifications	Get sound control number play position	SND_GetSeqPlayPos	28

**Format:** void SND\_GetSeqPlayPos(SndSeqPlayPos \*pos, SndSeqNum seq\_no)  
**Input:** seq\_no : Sound control number  
**Output:** pos : Sound control number play position  
**Function Value:** None  
**Function:** Gets the play position of the specified sound control number.

### PCM

Title	Function	Function Name	No.
Function Specifications	Get PCM execute address	SND_GetPcmPlayAddr	29

**Format:** void SND\_GetPcmPlayAddr(SndPcmPlayAddr \*adr, SndPcmNum num)  
**Input:** num : PCM play number  
**Output:** adr : PCM execute address  
**Function Value:** None  
**Function:** Gets the PCM data address being played back.

Title	Function	Function Name	No.
Function Specifications	Get PCM address update interrupt status	SND_GET_INT_STAT	30

**Format:** SndPcmIntStat SND\_GET\_INT\_STAT(void)  
**Input:** None  
**Output:** None  
**Function Value:** PCM address update interrupt status  
**Function:** Gets the PCM address update interrupt status. Valid when the interrupt source is interrupted by the PCM address update. The PCM playback number address that was updated can be determined with this information.

---

## CD

Title	Function	Function Name	No.
Function Specifications	Get total stereo volume analysis	SND_GetAnlTlV1	31

**Format:** void SND\_GetAnlTlV1(SndChV1Anl \*left, SndCdAnl \*right)  
**Input:** None  
**Output:** left : Total volume of left output  
right : Total volume of right output  
**Function Value:** None  
**Function:** Gets the analysis of the total stereo volume. Execute the volume analysis start function before executing this function.  
**Remarks:** Volume is updated at 16 msec intervals.

Title	Function	Function Name	No.
Function Specifications	Get discrete frequency band stereo volume analysis	SND_GetAnlHzV1	32

**Format:** void SND\_GetAnlHzV1(SndCdHzSrV1 \*hz\_v1)  
**Input:** none  
**Output:** hz\_v1 : Frequency band stereo volume  
**Function Value:** None  
**Function:** Gets the stereo volume based on sound frequency bands. Execute the volume analysis start function before executing this function.  
**Remarks:** Volume is updated at 16 msec intervals.

---

# DMA Library

## 1.0 Guide

### 1.1 Objective

Direct Memory Access (DMA) functions decrease the overhead on the Main CPU's processing time by enabling it to perform work other than the transfer of data. This library provides basic DMA data transfer functions.

### 1.2 Overview

The DMA Library supports both the SH2 (CPU) and SCU-based DMA transfers. High-level and low-level library functions are provided for each device.

Descriptions of the high-level and low-level functions are as follows.

- **High-level functions**  
Provides basic functionality for byte, word, and longword transfers through easy-to-use functions.
- **Low-level functions**  
Provides functions that enable the detailed setup of device modes, interrupts, and status communication.

The main differences between the SH2 (CPU) and SCU DMA functions are explained next. For more detailed information, refer back to the hardware manual for each device.

- **CPU**  
Capable of transfers between the same bus (e.g., between work RAM areas). This is not possible with the SCU.
- **SCU**  
Because parallel operations by the CPU can be performed during data transfers, the SCU is effective in transfers that do not involve the CPU bus. The SCU is capable of high-speed transfers compared with the CPU.

### 1.3 Precautions

- Select the suitable function after examining the source and destination access units used for the transfer. See the appropriate hardware manual for information about access units.
- High-level DMA transfers purge the destination area after a transfer under the following condition.
  - Condition: When the destination area is the work RAM.
  - Reason: An assumption is made that the RAM area read using the cache is work RAM.
- Burst mode transfers cannot be used by the CPU.

---

## 1.4 Calling Sequence

The following example shows the DMA transfer calling sequence.

```
#define SCLA_CFRAME ((void *)0x5e60000)      /* VDP2 VRAM scroll coefficient address */
#define K_NUM      (424)                      /* Scroll coefficient data size          */
Uint32 scl_k_data[512];                      /* Work RAM scroll coefficient data area */

. . .

void copyToData()
{
    DMA_ScuCopyMem(SCLA_CFRAME, scl_k_data, K_NUM);
                                /* data transfer to VDP2 VRAM fromWork RAM */

    . . .
    if(DMA_ScuResult() == DMA_SCU_BUSY){ /* DMA_ScuCopyMem() completion check */
        . . .
    }
}
```

---

## 2.0 Reference

### 2.1 Data List

	Data		Data Name	No.
<b>Low-level</b>	SCU	DMA channel	None	1
		Transfer parameters	DmaScuPrm	2
	CPU	DMA channel	None	3
		Common transfer parameters	DmaCpuComPrm	4
		Transfer parameters	DmaCpuPrm	5

## 2.2 Data Specifications

- **Low-level SCU Transfers**

Title	Data	Data Name	No.
Data Specifications	DMA Channel	none	1

Use the following constants when specifying an SCU DMA channel.

Name of Constant	Description
DMA_SCU_CH0	Channel 0
DMA_SCU_CH1	Channel 1
DMA_SCU_CH2	Channel 2

Title	Data	Data Name	No.
Data Specifications	Transfer parameters	DmaScuPrm	2

Transfer parameters have the following structure.

```
struct {  
    Uint32 dxr;      /* Read address */  
    Uint32 dxw;      /* Write address */  
    Uint32 dxc;      /* Number of transfer bytes */  
    Uint32 dxad_r;   /* Read address add value */  
    Uint32 dxad_w;   /* Write address add value */  
    Uint32 dxmod;    /* Mode bit */  
    Uint32 dxrup;    /* Read address update bit */  
    Uint32 dxwup;    /* Write address update bit */  
    Uint32 dxft;     /* Start source selection bit */  
    Uint32 msk;      /* Mask bit */  
} DmaScuPrm;
```

The following tables show constants that can be used by each member.

dxad\_r /\* read address add value \*/

Name of Constant	Description
DMA_SCU_R0	Don't add
DMA_SCU_R4	Add 4 bytes

---

```
dxad_w /* write address add value */
```

Name of Constant	Description
DMA_SCU_W0	Don't add
DMA_SCU_W2	Add 2 bytes
DMA_SCU_W4	Add 4 bytes
DMA_SCU_W8	Add 8 bytes
DMA_SCU_W16	Add 16 bytes
DMA_SCU_W32	Add 32 bytes
DMA_SCU_W64	Add 64 bytes
DMA_SCU_W128	Add 128 bytes

```
dxmod /* mode bit */
```

Name of Constant	Description
DMA_SCU_DIR	Direct mode
DMA_SCU_IN_DIR	Indirect mode

```
dxrup /* read address update bit */
```

```
dxwup /* write address update bit */
```

Name of Constant	Description
DMA_SCU_KEEP	Hold
DMA_SCU_REN	Update

```
dxft /* start source select bit */
```

Name of Constant	Description
DMA_SCU_F_VBLK_IN	Receive V-blank-IN signal
DMA_SCU_F_VBLK_OUT	Receive V-blank-OUT signal
DMA_SCU_F_HBLK_OUT	Receive H-blank-IN signal
DMA_SCU_F_TIM0	Receive Timer 0 signal
DMA_SCU_F_TIM1	Receive Timer 1 signal
DMA_SCU_F SND	Receive Sound-Req signal
DMA_SCU_F SPR	Receive Sprite draw end signal
DMA_SCU_F DMA	Set DMA start source bit

---

```
mask /* mask bit */
```

(This is a write mask bit for the members above. Bits specified by the following constants are not written. Multiple specifications are possible by using logical OR)

Name of Constant	Description
DMA_SCU_M_DXR	Read address
DMA_SCU_M_DXW	Write address
DMA_SCU_M_DXC	Number of transfer bytes
DMA_SCU_M_DXAD_R	Read address add value
DMA_SCU_M_DXAD_W	Write address add value
DMA_SCU_M_DXMOD	Mode bit
DMA_SCU_M_DXRUP	Read address update bit
DMA_SCU_M_DXWUP	Write address update bit
DMA_SCU_M_DXFT	Start source selection bit

- **Low-Level CPU Transfers**

Title	Data	Data Name	No.
Data Specifications	DMA channel	None	3

Use the following constants when specifying a CPU DMA channel.

Name of Constant	Description
DMA_CPU_CH0	Channel 0
DMA_CPU_CH1	Channel 1

Title	Data	Data Name	No.
Data Specifications	Common transfer parameters	DmaCpuComPrm	4

Common transfer parameters have the following structure.

```
struct {
    Uint32 pr;      /* priority mode */
    Uint32 dme;     /* DMA master enable */
    Uint32 msk;     /* mask bit */
} DmaCpuComPrm;
```

The following tables show constants that can be used by each of these members.

pr /\* priority mode \*/

Name of Constant	Description
DMA_CPU_FIX	Priority sequence is fixed
DMA_CPU_ROR	Priority sequence based on round robin

dme /\* DMA master enable \*/

Name of Constant	Description
DMA_CPU_DIS	DMA transfer on all channels disabled
DMA_CPU_ENA	DMA transfer on all channels enabled

msk /\* mask bit \*/

This is a write mask bit for the members. Bits specified by the following constants are not written. Multiple specifications are possible by using logical OR.

Name of Constant	Description
DMA_CPU_M_PR	Priority mode
DMA_CPU_M_AE	Address error flag
DMA_CPU_M_NMIF	NMI flag
DMA_CPU_M_DME	DMA master enable

---

Title	Data	Data Name	No.
Data Specifications	Transfer parameters	DmaCpuPrm	5

Transfer parameters have the following structure.

```
struct {
    Uint32 sar;      /* DMA source address */ */
    Uint32 dar;      /* DMA destination address */ */
    Uint32 tcr;      /* DMA transfer count */ */
    Uint32 dm;       /* destination address mode */ */
    Uint32 sm;       /* source address mode bit */ */
    Uint32 ts;       /* transfer size */ */
    Uint32 ar;       /* auto request mode */ */
    Uint32 ie;       /* interrupt enable */ */
    Uint32 drcr;     /* DMA request/response select control */ */
    Uint32 msk;      /* mask bit */ */
} DmaCpuPrm;
```

The following tables show constants that can be used by each member.

dm /* destination address mode	*/
sm /* source address mode bit	*/

Name of Constant	Description
DMA_CPU_AM_NOM	Fixed
DMA_CPU_AM_ADD	Add
DMA_CPU_AM_SUB	Subtract

ts /* transfer size	*/
---------------------	----

Name of Constant	Description
DMA_CPU_1	Byte unit
DMA_CPU_2	Word (2 byte) unit
DMA_CPU_4	Long word (4 byte) unit
DMA_CPU_16	16 byte unit

ar /* auto request mode	*/
-------------------------	----

Name of Constant	Description
DMA_CPU_MOD	Module request
DMA_CPU_AUTO	Auto request

ie /* interrupt enable	*/
------------------------	----

Name of Constant	Description
DMA_CPU_INT_ENA	Enable interrupt request
DMA_CPU_INT_DIS	Disable interrupt request

---

```
drcr /* DMA request/response select control */
```

Name of Constant	Description
DMA_CPU_DREQ	DREQ (external request)
DMA_CPU_RXI	RXI (built-in SCI receive data full interrupt transfer request)
DMA_CPU_TXI	TXI (built-in SCI send data empty interrupt transfer request)

```
msk /* mask bit */
```

Name of Constant	Description
DMA_CPU_M_SAR	DMA source address
DMA_CPU_M_DAR	DMA destination address
DMA_CPU_M_TCR	DMA transfer count
DMA_CPU_M_DM	Destination address mode
DMA_CPU_M_SM	Source address mode bit
DMA_CPU_M_TS	Transfer size
DMA_CPU_M_AR	Auto request mode
DMA_CPU_M_IE	Interrupt enable
DMA_CPU_M_DRCSR	DMA request/response select control
DMA_CPU_M_TE	Transfer end flag

This is a member write mask bit. Bits specified by the following constants are not written. Multiple settings are possible by using logical OR.

## 2.3 List of Functions

		Function	Function Name	No.
High-level	SCU	DMA data transfer	DMA_ScuMemCopy	1
		DMA data transfer result check	DMA_ScuResult	2
	CPU	DMA byte data transfer	DMA_CpuMemCopy1	3
		DMA word data transfer	DMA_CpuMemCopy2	4
		DMA long word data transfer	DMA_CpuMemCopy4	5
		DMA 16-byte data transfer	DMA_CpuMemCopy16	6
		DMA data transfer end check	DMA_CpuResult	7
Low-level	SCU	Set DMA transfer parameters	DMA_ScuSetPrm	8
		Start DMA transfer	DMA_ScuStart	9
		Stop DMA transfer	DMA_ScuStop	10
		Stop DMA transfer on all channels	DMA_ScuAllStop	11
	CPU	Set common DMA transfer parameters	DMA_CpuSetComPrm	12
		Set DMA transfer parameters	DMA_CpuSetPrm	13
		Start DMA transfer	DMA_CpuStart	14
		Stop DMA transfer	DMA_CpuStop	15
		Stop DMA transfer on all channels	DMA_CpuAllStop	16
		Get common DMA status	DMA_CpuGetComStatus	17
		Get DMA status	DMA_CpuGetStatus	18

---

## 2.4 Function Specifications

- **High-level SCU Transfers**

Title	Function	Function Name	No.
Function Specifications	DMA data transfer	DMA_ScuMemCopy	1

**Format:**

void DMA\_ScuMemCopy(void \*dst, void \*src, Uint32 cnt)

**Input:**

\*dst : Destination address  
\*src : Source address  
cnt : Transfer byte count

**Output:**

None

**Function Value:**

None

**Function:**

DMA transfer is done by the direct mode of the SCU DMA mode 0. For more information, see the *SCU User's Manual* (ST-097-R5-072694). Operation of this transfer differs depending on whether the destination address is in work RAM or not. The differences are explained below.

- Work RAM

The destination area is purged after transfer in order to eliminate conflict with the cache. In addition, because the CPU bus is used, control does not return to the CPU until DMA execution ends.

- Areas other than the Work RAM

After transfer begins, control returns to the CPU immediately because the CPU bus is not used. In this case, a check of the transfer result is done by `DMA_ScuResult()`.

Although transfer units are basically long word units, but the source and destination addresses can be specified in byte units. However, when the device requests data in word boundary units, make sure to do so. For more information, see the *SCU User's Manual*.

---

Title	Function	Function Name	No.
Function Specifications	DMA data transfer result check	Dma_ScuResult	2

**Format:**            Uint 32 DMA\_ScuResult(void)

**Input:**            None

**Output:**          None

**Function Value:** Result of DMA\_ScuMemCopy( )

**Function:**         Checks the results of DMA\_ScuMemCopy( ). The following table lists the return values.

#### Execution Result Constant Names

Name of Constant	Description
DMA_SCU_END	Normal end
DMA_SCU_FAIL	Abnormal end
DMA_SCU_BUSY	Busy

**Remarks:**         When the CPU\_Bus is specified as the transfer source or destination with DMA\_ScuMemCopy( ), the CPU cannot run while the DMA is executing.

Accordingly, the execution results in that case cannot be "busy" (DMA\_SCU\_BUSY).

---

- **High-level CPU Transfers**

Title	Function	Function Name	No.
Function Specifications	DMA byte data transfer	DMA_CpuMemCopy1	3

**Format:** void DMA\_CpuMemCopy1(void \*dst, void \*src, Uint32 cnt)

**Input:**

*dst	: Destination address
*src	: Source address
cnt	: Transfer count (0 - 16777215)
	0x00000001 1 time
	0x00fffff 16777215 times
	0x00000000 16777216 times

**Output:** None

**Function Value:** None

**Function:** Transfers byte data cnt times from the src address to the dst address. The transfer amount is cnt bytes. Data is transferred without the use of channel 0, cycle steal mode, and completion interrupt. When the destination address is the work RAM, the destination area is purged after transfer in order to eliminate conflicts with the cache. There are no restrictions on addressing.

Title	Function	Function Name	No.
Function Specifications	DMA word data transfer	DMA_CpuMemCopy2	4

**Format:** void DMA\_CpuMemCopy2(void \*dst, void \*src, Uint32 cnt)

**Input:**

*dst	: Destination address
*src	: Source address
cnt	: Transfer count (0 - 16777215)
	0x0000000001 1 time
	0x00fffff 16777215 times
	0x00000000 16777216 times

**Output:** None

**Function Value:** None

**Function:** Transfers byte data cnt times from the src address to the dst address. The transfer amount is cnt \* 2 bytes. Data is transferred without the use of channel 0, cycle steal mode, and completion interrupt. When the destination address is the work RAM, the destination area is purged after transfer in order to eliminate conflicts with the cache. The addressing must be word boundary-based.

Title	Function	Function Name	No.
Function Specifications	DMA long word Data Transfer	DMA_CpuMemCopy4	5

**Format:** void DMA\_CpuMemCopy4(void \*dst, void \*src, Uint32 cnt)

**Input:**

- \*dst : Destination address
- \*src : Source address
- cnt : Transfer count (0 - 16777215)
 

0x00000001	1 time
0x00ffffff	16777215 times
0x00000000	16777216 times

**Output:** None

**Function Value:** None

**Function:** Transfers long word data cnt times from the src address to the dst address. The transfer amount is cnt \* 4 bytes. Data is transferred without the use of channel 0, cycle steal mode, and completion interrupt. When the destination address is the work RAM, the destination area is purged after transfer in order to eliminate conflicts with the cache. Addressing must be long word boundary-based.

Title	Function	Function Name	No.
Function Specifications	DMA 1-byte data transfer	DMA_CpuMemCopy16	6

**Format:** void DMA\_CpuMemCopy16(void \*dst, void \*src, Uint32 cnt)

**Input:**

- \*dst : Destination address
- \*src : Source address
- cnt : Transfer count (0 - 16777215) (long word units)
 

0x00000001	1 time
0x00ffffff	16777215 times
0x00000000	16777216 times

**Output:** None

**Function Value:** None

**Function:** Transfers long word data of data cnt times from the src address to the dst address. The transfer amount is cnt \* 4 bytes. Data is transferred without the use of channel 0, the cycle steal mode, and completion interrupt. When the destination address is the work RAM, the destination area is purged after transfer in order to eliminate conflicts with the cache. Addressing must be long word boundary-based.

---

Title	Function	Function Name	No.
Function Specifications	DMA data transfer end check	DMA_CpuResult	7

**Format:** `Uint 32 DMA_CpuResult(void)`

**Input:** None

**Output:** None

**Function Value:** Results of `DMA_CpuMemCopy 1, 2, 4, 16 ()`

**Function:** Checks the results of `DMA_CpuMemCopy 1, 2, 4, 16 ()`. The following table shows returned values.

#### Execution Result Constant Names

Name of Constant	Description
<code>DMA_CPU_END</code>	Normal end
<code>DMA_CPU_BUSY</code>	Busy

---

- **Low-level SCU Transfers**

Title	Function	Function Name	No.
Function Specifications	Set DMA transfer parameters	DMA_ScuSetPrm	8

**Format:** void DMA\_ScuSetPrm(DmaScuPrm \*prm, Uint32 ch)  
**Input:** prm : Transfer parameters  
           ch : DMA channel  
**Output:** None  
**Function Value:** None  
**Function:** Sets the values of transfer parameters to the specified DMA channel.

Title	Function	Function Name	No.
Function Specifications	Start DMA transfer	DMA_ScuStart	9

**Format:** void DMA\_ScuStart(Uint32 ch)  
**Input:** ch : DMA channel  
**Output:** None  
**Function Value:** None  
**Function:** Starts DMA transfer on the specified DMA channel.

Title	Function	Function Name	No.
Function Specifications	Stop DMA transfer	DMA_ScuStop	10

**Format:** void DMA\_ScuStop(Uint32 ch)  
**Input:** ch : DMA channel  
**Output:** None  
**Function Value:** None  
**Function:** Stops DMA transfer on the specified DMA channel.

Title	Function	Function Name	No.
Function Specifications	Stop DMA transfer on all channels	DMA_ScuAllStop	11

**Format:** void DMA\_ScuAllStop(void)  
**Input:** None  
**Output:** None  
**Function Value:** None  
**Function:** Stops DMA transfer on all DMA channels.

---

- **Low-level CPU Transfers**

Title	Function	Function Name	No.
Function Specifications	Set common DMA transfer parameters	DMA_CpuSetComPrm	12

**Format:** void DMA\_CpuSetComPrm(DmaCpuComPrm \*com\_prm)  
**Input:** com\_prm : Common transfer parameters  
**Output:** None  
**Function Value:** None  
**Function:** Sets specified common DMA transfer parameters, but does not set parameters masked by DmaCpuComPrm.msk.  
**Remarks:** When DMA\_CPU\_M\_AE is not masked by DmaCpuComPrm.msk, the address error flag is cleared. When DMA\_CPU\_M\_NMIF is not masked, the NMI flag is cleared.

Title	Function	Function Name	No.
Function Specifications	Set DMA transfer parameters	DMA_CpuSetPrm	13

**Format:** void DMA\_CpuSetPrm(DmaCpuPrm \*prm Uint32 ch, )  
**Input:** prm : Transfer parameters  
ch : DMA channel  
**Output:** None  
**Function Value:** None  
**Function:** Sets specified channel transfer parameters, but does not set parameters masked by DmaCpuPrm.msk.  
**Remarks:** When DMA\_CPU\_TE is not masked by DmaCpuPrm.msk, the transfer end flag bit is cleared.

Title	Function	Function Name	No.
Function Specifications	Start DMA transfer	DMA_CpuStart	14

**Format:** Uint32 DMA\_CpuStart(Uint32 ch)  
**Input:** ch : DMA channel  
**Output:** None  
**Function Value:** None  
**Function:** Starts DMA transfer on the specified DMA channel.

Title	Function	Function Name	No.
Function Specifications	Stop DMA transfer	DMA_CpuStop	15

**Format:** void DMA\_CpuStop(Uint32 ch)  
**Input:** ch : DMA channel  
**Output:** None  
**Function Value:** None  
**Function:** Stops DMA transfer on the specified DMA channel.

Title	Function	Function Name	No.
Function Specifications	Stop DMA transfer on all channels	DMA_CpuAllStop	16

**Format:** void DMA\_CpuAllStop(void)  
**Input:** None  
**Output:** None  
**Function Value:** None  
**Function:** Stops DMA transfer on all DMA channels.

Title	Function	Function Name	No.
Function Specifications	Get common DMA status	DMA_CpuGetComStatus	17

**Format:** void DMA\_CpuGetComStatus(DmaCpuComStatus \*status)  
**Input:** None  
**Output:** status : status pointer  
**Function Value:** None  
**Function:** Gets the specified common status.

Title	Function	Function Name	No.
Function Specifications	Get DMA status	DMA_CpuGetStatus	18

**Format:** void DMA\_CpuGetStatus(DmaCpuStatus \*status, Uint32 ch)  
**Input:** ch : DMA channel  
**Output:** status : Status pointer  
**Function Value:** None  
**Function:** Gets the status on specified DMA channel.

---

# Cache Library

## 1.0 Guide

### 1.1 Objective

The Cache Library provides functions that are required after the execution of a DMA as well as function-format macros for cache register operations.

### 1.2 Overview

The main purpose of this library is to provide functions that purge the cache after DMA transfers.

When the CPU accesses RAM memory space via a cache address, a conflict occurs between the RAM and cache memory contents if the RAM is used as a destination for DMA transfers.

For more details on SH2 cache operations, refer to the SH2 hardware manual.

---

## 2.0 Reference

### 2.1 List of Functions

Function		Function Name	No.
High-level	Initialize cache	CSH_Init	1
	Clear all caches	CSH_AllClr	2
	Associative purge of target area	CSH_Purge	3

### 2.2 Function Specifications

- **High-level**

Title	Function	Function Name	No.
Function Specifications	Initialize cache	CSH_Init	1

**Format:** void CSH\_Init(Uint16 sw)

**Input:** sw : Way mode

#### Way Mode Constant Names

Name of Constant	Description
CSH_4WAY	4-way set associative
CSH_2WAY	2-way set associative

**Output:** None

**Function Value:** None

**Function:** Initializes the cache. Clears valid bits of all cache lines, enables instruction fill and data fill, and enables the cache.

Title	Function	Function Name	No.
Function Specifications	Clear all caches	CSH_AllClr	2

**Format:** void CSH\_AllClr(void)

**Input:** None

**Output:** None

**Function Value:** None

**Function:** Clears valid bits of all caches of all ways. This function executes internally after disabling the cache and re-enables the cache after cache clear.

**Remarks:** Address array is updated, however, the data array is not updated.

Title	Function	Function Name	No.
Function Specifications	Associative purge of target area	CSH_Purge	3

**Format:** void CSH\_Purge(void \*address, Uint32 p\_size))  
**Input:** \*address : Start address  
               p\_size   : Purge byte size  
**Output:** None  
**Function Value:** None  
**Function:** Purges the area specified by address and p\_size.

## 2.3 List of Function Format Macros

Function		Function Format Macro Name	No.
Low-level	Get cache register	CSH_GET_CCR	1
	Set cache register	CSH_SET_CCR	2
	Cache enable control	CSH_SET_ENABLE	3
	Instruction fill control	CSH_SET_CODE_FILL	4
	Data fill control	CSH_SET_DATA_FILL	5
	2, 4-way set associative mode switch	CSH_SET_WAY_MODE	6
	Select address array access way	CSH_SET_ACS_WAY	7

## 2.4 Function Format Macro Specification

### Low-level

Title	Function	Function Name	No.
Function Specifications	Get cache register	CSH_GetCcr	1

**Format:** Uint16 CSH\_GetCcr(void)  
**Input:** None  
**Output:** None  
**Function Value:** Cache register contents  
**Function:** Gets the CCR contents and returns as a function value.

Title	Function	Function Name	No.
Function Specifications	Set cache register	CSH_SetCcr	2

**Format:**            `Uint16 CSH_SetCcr(Uint16 reg)`

**Input:**            `reg` : Cache register value

**Output:**          none

**Function Value:** Returns contents of register

**Function:**         Sets `reg` to CCR. Can set each state of the cache in one setting.

Title	Function	Function Name	No.
Function Specifications	Cache enable control	CSH_SetEnable	3

**Format:**            `void CSH_SetEnable(Uint16 sw)`

**Input:**            `sw` : Cache enable switch

#### Cache Enable Switch Constant Names

Name of Constant	Description
<code>CSH_DISABLE</code>	Disable cache
<code>CSH_ENABLE</code>	Enable cache

**Output:**          None

**Function Value:** None

**Function:**         Controls operation of the cache by the specified cache enable switch.

Title	Function	Function Name	No.
Function Specifications	Instruction fill control	CSH_SetCodeFill	4

**Format:**            `void CSH_SetCodeFill(Uint16 sw)`

**Input:**            `sw` : Instruction fill disable switch

#### Command Fill Disable Switch Constant Names

Name of Constant	Description
<code>CSH_CODE_ENABLE</code>	Enable instruction fill
<code>CSH_CODE_DISABLE</code>	Disable instruction fill

**Output:**          none

**Function Value:** none

**Function:**         Controls the cache fill operation during instruction fetches.

Title	Function	Function Name	No.
Function Specifications	Data fill control	CSH_SetDataFill	5

**Format:** void CSH\_SetDataFill(Uint16 sw)  
**Input:** sw : Data fill disable switch

#### Disable Switch Constant Names

Name of Constant	Description
CSH_DATA_ENABLE	Enable data fill
CSH_DATA_DISABLE	Disable data fill

**Output:** None  
**Function Value:** None  
**Function:** Controls the cache fill operation when reading data.

Title	Function	Function Name	No.
Function Specifications	2, 4-way set associative mode switch	CSH_SetWayMode	6

**Format:** void CSH\_SetWayMode(Uint16 sw)  
**Input:** sw : Way Mode

#### Way Mode Constant Names

Name of Constant	Description
CSH_4WAY	4-way set associative mode
CSH_2WAY	2-way set associative mode

**Output:** None  
**Function Value:** None  
**Function:** Selects 2-way or 4-way mode.

Title	Function	Function Name	No.
Function Specifications	Select access way of address array	CSH_SetAcsWay	7

**Format:** void CSH\_SetAcsWay(Uint16 way)  
**Input:** way : Way (0~3)  
**Output:** None  
**Function Value:** None  
**Function:** Used in selecting the access way when reading or writing an address array.

---

# Interrupt Management Library

## 1.0 Guide

### 1.1 Objective

The Interrupt Management Library controls interrupts.

### 1.2 Overview

The library manages access to interrupt registers and interrupt processing routines.

### 1.3 Overview of Functions

#### Interrupt RegisterAccess Control

Because the interrupt mask register is write-only, its value cannot be read. Therefore, values written to the interrupt mask register are held by the library. Specifically, the following methods are used.

- (1) The interrupt mask register state is always held in memory (mask state memory hereafter).
- (2) The same data is written to the mask state memory when a write occurs to the interrupt mask register.
- (3) The contents of the mask state memory are read when there is a need to read the interrupt mask register.

#### Interrupt Processing RoutineAccess

The library may be used to set the interrupt function or SCU function to the interrupt vector table. In addition, the current interrupt vector settings can be checked. The SCU interrupt function is set as the initial value in the SCU interrupt vector table. The SCU interrupt function has a feature that enables a set SCU function to be called as a subroutine. The SCU function is executed each time the SCU interrupt function is executed. As default initial settings, a dummy function (which executes only the return process) is set as the interrupt function, and a SCU dummy function (which executes only the return process) is set as the SCU interrupt function.

### 1.4 Usage Conditions

#### Interrupt RegisterAccess Control

Because the value of the interrupt mask register is held by the library, the library should be used to set the interrupt mask register. If the register is written directly, the interrupt mask register will not correspond with the mask state memory. As a result, correct processing will not occur.

---

## 1.5 Calling Sequence

### Interrupt RegisterAccess Control

```
void sysInit()
{
    INT_SetMsk((INT_MSK_HBLK_IN | INT_MSK_VBLK_OUT),
                (INT_MSK_SPR | INT_MSK_DMA1));
    /* Enable "H-Blank-IN" and "V-Blank-OUT" */
    /* Disable "Sprite draw complete" and "Level 0-DMA" */

    . . .

}

void anyPrg()
{
    Uint32 status_bit;
    status_bit = INT_GET_STAT(); /* Get status */
    if(INT_JudgeStat(status_bit, INT_ST_VBLK_OUT, INT_ST_VBLK_IN) == TRUE) {
        /* When V-Blank-OUT interrupt occurs, and */
        /* V-Blank-IN interrupt does not occur */

        . . .
    }
}
```

### Interrupt Processing RoutineAccess

#### File A

```
#pragma interrupt (vblkIn) /* vblkIn function is specified as the
                           interrupt function */

void vblkIn(void)
{
    .
    .
}
```

---

**File B**

```
void systemInit(void)
{
    INT_SetFunc(INT_SCU_VBLK_IN, vblkIn);
        /* vblkIn interrupt function is set to the */
        /* V-Blank-IN interrupt vector table.      */
    INT_SetScuFunc(INT_SCU_VBLK_OUT, vblkOut);
        /* vblkOut function is set to the SCU interrupt */
        /* function of the V-Blank-OUT interrupt vector. */
    . . .
}

void vblkOut(void)
{
    . . .
}
```

## 2.0 Reference

### 2.1 Data List

Data	Data Name	Number
<b>Interrupt Register Access Control</b>		
Interrupt mask bit value constant	None	1
Interrupt status bit value constant	None	2
<b>Interrupt Processing Routine Access</b>		
Vector number constant	None	3

## 2.2 Data Specifications

### Interrupt RegisterAccess Control

Title	Data	Data Name	No.
Data Specifications	Interrupt mask bit value constant	None	1

The following table shows constants that can be set as the interrupt mask bit value. The value of each constant is 1 for the 1 bit that the constant represents, and everything else is set as 0. Multiple numbers can be set once by using logical OR.

Constant	Description
INT_MSK_NULL	No specification
INT_MSK_ALL	Set all (specifies all bits described below)
INT_MSK_ABUS	A-Bus
INT_MSK_SPR	Sprite draw complete
INT_MSK_DMAI	Illegal DMA
INT_MSK_DMA0	Level 0 DMA
INT_MSK_DMA1	Level 1 DMA
INT_MSK_DMA2	Level 2 DMA
INT_MSK_PAD	PAD
INT_MSK_SYS	System manager
INT_MSK SND	Sound request
INT_MSK_DSP	DSP end
INT_MSK_TIM1	Timer 1
INT_MSK_TIM0	Timer 0
INT_MSK_HBLK_IN	H-Blank-IN
INT_MSK_VBLK_OUT	V-Blank-OUT
INT_MSK_VBLK_IN	V-Blank-IN

---

Title	Data	Data Name	No.
Data Specifications	Interrupt Status Bit Value Constant	None	2

The following table shows the constants that can be set as the interrupt status bit value. The value of each constant is 1 for the 1 bit that the constant represents, and everything else is set as 0. Multiple numbers can be set once by using logical OR.

Constant	Description
INT_ST_NULL	No specification
INT_ST_ALL	Specify all (specifies all bits described below)
INT_ST_ABUS	A-Bus (specifies A-Bus 01 ~ 16 bits)
INT_ST_ABUS_01 ~ ABUS_16	A-Bus 01 ~ 16
INT_ST_SPR	Sprite draw complete
INT_ST_DMA1	Illegal DMA
INT_ST_DMA0	Level 0 DMA
INT_ST_DMA1	Level 1 DMA
INT_ST_DMA2	Level 2 DMA
INT_ST_PAD	PAD
INT_ST_SYS	System manager
INT_ST SND	Sound request
INT_ST_DSP	DSP end
INT_ST_TIM1	Timer 1
INT_ST_TIM0	Timer 0
INT_ST_HBLK_IN	H-Blank-IN
INT_ST_VBLK_OUT	V-Blank-OUT
INT_ST_VBLK_IN	V-Blank-IN

## Interrupt Processing Routine Access

Title	Data	Data Name	No.
Data Specifications	Vector number constant	None	3

The following table shows the constants that can be set as the vector number. The SCU constant name can be set only by the master SH2. The CPU constant name can be set by the master and slave SH2.

Interrupt Types	Constant Name	Description
SCU	INT_SCU_ABUS	A-Bus
	INT_SCU_SPR	Sprite draw complete
	INT_SCU_DMAI	Illegal DMA
	INT_SCU_DMA0	Level 0 DMA
	INT_SCU_DMA1	Level 1 DMA
	INT_SCU_DMA2	Level 2 DMA
	INT_SCU_PAD	PAD
	NT_SCU_SYS	System manager
	NT_SCU SND	Sound request
	NT_SCU_DSP	DSP end
	NT_SCU_TIM1	Timer 1
	NT_SCU_TIM0	Timer 0
	NT_SCU_HBLK_IN	H-Blank-IN
	NT_SCU_VBLK_OUT	V-Blank-OUT
	NT_SCU_VBLK_IN	V-Blank-IN
CPU	INT_CPU_DIVU	Divider
	NT_CPU_DMAC0	DMAC channel 0
	NT_CPU_DMAC1	DMAC channel 1
	NT_CPU_WDT	WDT interval
	NT_CPU_BSC	BSC compare match
	NT_CPU_SCI_ERI	SCI receive error
	NT_CPU_SCI_RXI	SCI receive data fill
	NT_CPU_SCI_TXI	SCI receive data empty
	NT_CPU_SCI_TEI	SCI send end
	NT_CPU_FRTICI	FRT input capture
	NT_CPU_FRTOCI	FRT output compare
	NT_CPU_FRT_OVI	FRT overflow

---

## 2.3 List of Functions

Function	Function Name	Number
<b>Interrupt Register Access Control</b>		
Get interrupt mask register	INT_GetMsk	1
Set interrupt mask register	INT_SetMsk	2
Change interrupt mask register	INT_ChgMsk	3
Get interrupt status register	INT_GetStat	4
Reset interrupt status register	INT_ResStat	5
Set A-Bus interrupt acknowledge	INT_SetAck	6
Get A-Bus interrupt acknowledge	INT_GetAck	7
<b>Interrupt Processing Routine Access</b>		
Set interrupt function	INT_SetFunc	8
Set SCU function	INT_SetScuFunc	9
Get interrupt function address	INT_GetFunc	10
Get SCU function address	INT_GetScuFunc	11

## 2.4 Function Specifications

### Interrupt RegisterAccess Control

Title	Function	Function Name	No.
Function Specifications	Get interrupt mask register	INT_GetMsk	1

**Format:** `Uint32 INT_GetMsk(void)`

**Input:** None

**Output:** None

**Function Value:** Interrupt mask bit value (0= not masked, 1= masked)

**Function:** Mask state memory is fetched as the interrupt mask bit value.

Title	Function	Function Name	No.
Function Specifications	Set interrupt mask register	INT_SetMsk	2

**Format:** `void INT_SetMsk(Uint32 msk_bit)`

**Input:** `msk_bit : Interrupt mask bit value  
(0= not masked, 1= masked)`

**Output:** None

**Function Value:** None

**Function:** Sets the interrupt mask bit value to the interrupt mask register and mask storage memory.

Title	Function	Function Name	No.
Function Specifications	Change interrupt mask register	INT_ChgMsk	3

**Format:** `void INT_ChgMsk(Uint32 ena_msk_bit, Uint32 dis_msk_bit)`

**Input:** `ena_msk_bit : Interrupt mask bit enable value  
dis_msk_bit : Interrupt mask bit disable value`

**Output:** None

**Function Value:** None

**Function:** Changes only the interrupt mask register and mask state memory of the specified bit. The values of unspecified sets are preserved. Set that the value so that the "1" bit of the interrupt mask bit enable value is not masked and the "1" bit of the interrupt mask bit disable value is masked. The unspecified bit will preserve its value.

Interruptions are temporarily disabled during this process. In addition, the interrupt status register that corresponds to the "1" bit of the interrupt mask bit enable value is reset. The interrupt acknowledge register is enabled when the A-Bus interrupt mask is enabled.

**Remarks:** When neither interrupt mask bit is set, specify `INT_MSK_NULL`.

Title	Function	Function Name	No.
Function Specifications	Get interrupt status register	INT_GetStat	4

**Format:**            `Uint32 INT_GetStat(void)`  
**Input:**            None  
**Output:**           None  
**Function Value:**   Interrupt status register value  
                       (0= interrupt not generated, 1= interrupt generated)  
**Function:**          Gets the interrupt status register value.

Title	Function	Function Name	No.
Function Specifications	Reset interrupt status register	INT_ResStat	5

**Format:**            `void INT_ResStat(Uint32 status_bit)`  
**Input:**            `status_bit : Interrupt status bit value`  
                       (0: reset, 1: preserve value)  
**Output:**           None  
**Function Value:**   None  
**Function:**          Resets the "0" bit of the interrupt status bit value.

Title	Function	Function Name	No.
Function Specifications	Set A-Bus interrupt acknowledge	INT_SetAck	6

**Format:**            `void INT_SetAck(Uint32 ack)`  
**Input:**            `ack : Acknowledge value`

#### Acknowledge Value Constant Names

Name of Constant	Description
<code>INT_ACK_ENA</code>	Enable
<code>INT_ACK_KEEP</code>	Preserve current state

**Output:**           None  
**Function Value:**   None  
**Function:**          Sets the specified acknowledge value to the A-Bus interrupt acknowledge register.

Title	Function	Function Name	No.
Function Specifications	Get A-Bus interrupt acknowledge	INT_GetAck	7

**Format:**            `Uint32 INT_GetAck(void)`

**Input:**            None

**Output:**           None

**Function Value:** Acknowledge value

#### Acknowledge Value Constant Names

Name of Constant	Description
<code>INT_ACK_ENA</code>	Enabled
<code>INT_ACK_DIS</code>	Disabled

**Function:**         Gets the acknowledge value of the A-Bus interrupt acknowledge register.

- **Interrupt Process RoutineAccess**

Title	Function	Function Name	No.
Function Specifications	Set interrupt function	INT_SetFunc	8

**Format:**            `void INT_SetFunc (Uint32 num, void* hdr)`

**Input:**            `num` : Vector number

`hdr` : Interrupt function address

**Output:**           None

**Function Value:** None

**Function:**         Sets the interrupt function address to the interrupt vector specified by the vector number. The set interrupt function is executed when an interrupt occurs. When the interrupt function address is NULL, functions set to the interrupt vector by based on the vector number are as follows:

- When the vector number is SCU:        Resets the SCU interrupt function.
- When the vector number is not SCU:    Sets a dummy function. The dummy function only performs return processing.

**Remarks:**          Interrupt functions can be specified with save register, return, and RTE instructions. (In C, functions using the `#pragma interrupt` declaration can be specified) This function can be used with both the master and slave SH2. This function must be executed with the CPU to be set.

Title	Function	Function Name	No.
Function Specifications	Set SCU function	INT_SetScuFunc	9

**Format:** void INT\_SetScuFunc (Uint32 num, void\* hdr)  
**Input:** num : Vector number  
           (Only SCU vector numbers can be specified)  
           hdr : SCU function address  
**Output:** None  
**Function Value:** None  
**Function:** Sets only the SCU function address to the SCU interrupt function of the specified vector number. The specified SCU function is executed when an SCU interrupt occurs. When the specified interrupt routine address is NULL, an SCU dummy function is set. The SCU dummy function performs only return processing.  
**Remarks:** This function may only be used on the master SH2. When setting the interrupt function with INT\_Setfunc, the vector of the SCU interrupt routine becomes invalid and the set function cannot be called.

Title	Function	Function Name	No.
Function Specifications	Get interrupt function address	INT_GetFunc	10

**Format:** void\* INT\_GetFunc(Uint32 num)  
**Input:** num : Vector number  
**Output:** None  
**Function Value:** Interrupt function address  
**Function:** Gets the interrupt function address set in the specified vector number.  
**Remarks:** This function can be used with both the master and slave SH2. Execute this function on the CPU to get the address for that CPU.

Title	Function	Function Name	No.
Function Specifications	Get SCU function address	INT_GetScuFunc	11

**Format:** void\* INT\_GetScuFunc(Uint32 num)  
**Input:** num : Vector number (SCU vector number only can be specified)  
**Output:** None  
**Function Value:** SCU function address  
**Function:** Gets the SCU function address set in the SCU interrupt function of the specified vector number.  
**Remarks:** This function can be used with the master SH2 only. Execute this function with the master SH2.

---

# Memory Management Library

## 1.0 Guide

### 1.1 Objective

Use the Memory Management Library for simplifying memory management tasks.

### 1.2 Overview

Various functions that allocate and free memory are provided. Functions that free up memory check adjacent memory space for additional free memory. When free memory is detected, it is linked with the memory being freed. These functions reduce the burden of memory management on the user. Memory Management Library functions have function interfaces similar to standard C library functions related to memory management.

### 1.3 Calling Sequence

The following example shows the calling sequence for allocating and freeing memory blocks.

```
void sysInit()
{
    . .
    MEM_Init(0x6050000, 0x10000); /* Sets 10000H byte from 6050000H
                                     to memory management area */
    . .
}

void userFunc()
{
    Uint32 *mem_areal;
    Uint8 *mem_area2;

    . .
    mem_areal = (Uint32 *)MEM_Malloc(4); /* allocates a 4 byte area */
    if(mem_areal == NULL) {
        return(ERR);
    }
    mem_area2 = (Uint8 *)MEM_Malloc(1); /* allocates a 1 byte area */
    if(mem_area2 == NULL) {
        return(ERR);
    }
    . .
    . .
    MEM_Free(mem_areal); /* frees mem_areal memory */
    . .
}
```

## 2.0 Reference

### 2.1 List of Functions

Function	Name	No.
Set memory management area	MEM_Init	1
Allocate array area	MEM_Calloc	2
Free memory blocks	MEM_Free	3
Allocate memory blocks	MEM_Malloc	4
Reallocate memory blocks	MEM_Realloc	5

### 2.2 Function Specifications

Title	Function	Function Name	No.
Function Specifications	Set memory management area	MEM_Init	1

**Format:** void MEM\_Init(Uint32 top\_address, Uint32 mem\_size)  
**Input:** top\_address: Memory management area start address  
mem\_size : Memory management area size (specify in bytes)  
**Output:** None  
**Function Value:** None  
**Function:** Sets the memory management area used by MEM\_Calloc, MEM\_Malloc, MEM\_Realloc, and MEM\_Free. The memory space defined by the start address and size is handled as the memory management area.  
**Remarks:** Before using MEM\_Calloc, MEM\_Malloc, MEM\_Realloc, and MEM\_Free, use this function only once. If used twice or more, the contents of the previously allocated memory cannot be guaranteed.

Title	Function	Function Name	No.
Function Specifications	Allocate array area	MEM_Calloc	2

**Format:** void \*MEM\_Calloc(Uint32 arg\_num, Uint32 arg\_size)  
**Input:** arg\_num : Number of array elements  
**Output:** arg\_size : Array element byte size  
**Function Value:** Returns a pointer to the allocated memory block when the function executes correctly. Returns NULL when an error occurs.  
**Function:** Allocates memory space to the memory management area equivalent to the specified array element byte size for the specified number of array elements. Allocated memory blocks are cleared to 0.

Title	Function	Function Name	No.
Function Specifications	Free memory blocks	MEM_Free	3

**Format:** void MEM\_Free(void \*mem\_ptr)  
**Input:** mem\_ptr : Pointer to the memory block  
**Output:** None  
**Function Value:** None  
**Function:** Frees specified memory blocks. Memory blocks that can be specified are those allocated by MEM\_Calloc, MEM\_Malloc, and MEM\_Realloc.

Title	Function	Function Name	No.
Function Specifications	Allocate memory blocks	MEM_Malloc	4

**Format:** void \*MEM\_Malloc(UINT32 mem\_size)  
**Input:** mem\_size : Requested memory block size (specified in bytes)  
**Output:** None  
**Function Value:** Returns a pointer to the allocated memory area when the function executes. Returns NULL when an error occurs.  
**Function:** Allocates blocks of the requested memory size to the memory management area, and returns a pointer for those blocks. When the requested memory block size is 0, NULL is returned as the function value. Initialization is not performed by setting this function to 0.

Title	Function	Function Name	No.
Function Specifications	Reallocate memory blocks	MEM_Realloc	5

**Format:** void \*MEM\_Realloc(void \*mem\_ptr, UINT32 mem\_size)  
**Input:** mem\_ptr : Pointer to the previous memory block  
**Output:** mem\_size : New memory block size (specified in bytes)  
**Function Value:** None  
**Function:** Normally returns a pointer to reallocated memory. Returns NULL when an error occurs.  
**Function:** The memory block size of the pointer belonging to the memory blocks prior to reallocation is updated. The contents of the previous memory block is transferred to a new area through reallocation.

---

# Timer Library

## 1.0 Guide

### 1.1 Objective

The Timer Library provides function format macros that access the programmable wait and processing timer values.

### 1.2 Overview

The timer uses the free running timer (FRT) in the SH2 and the timer interrupt in the SCU. Function format macro are provided for each device in consideration of the following applications.

#### SCU

- Timing that is in sync with V-Blanks and H-Blanks can be obtained. This timing information can be used for performing graphics processing tasks.

#### CPU

- The correct wait time can be set as required when programming. Relatively accurate waits within a C language program can be executed.
- Elapsed time can be determined. This can be used to profile code execution times.

## 1.3 Detailed Information

- SCU

### Usage Method

For more details on how to use the SCU's Timer Function, refer to the *Timer Interrupt* section in the *SCU User's Manual*.

### Calling Sequence

The following calling sequence example shows timer 1 being used to generate a timer 1 interrupt at the 10th bit of every line that is drawn. If the line is odd, it is flipped.

```
Uint32 time_flg;           /* interrupt flag */
. . .
void vblankOut ()          /* execute V-BLANK OUT function */
{
    Uint32 intr_count = 0;   /* interrupt counter */
. . .
    TIM_T1_DISABLE ():      /* disable timer 1 interrupt */
    TIM_T1_SET_MODE(TIM_MD_LINE); /* specify interrupt of
                                   every line */
    TIM_T1_SET_DATA (10);
        /* specify interrupt timing at the 10th bit of
           the line */
    time_flg = OFF;          /* turns off the interrupt flag */
    TIM_T1_ENABLE();         /* enable timer 1 interrupt */
. . .
    for(intr_count < ALL_LINE_NUM) {
        /* until interrupt of all lines has been
           executed */
        changeLine(intr_count); /* flip odd line */
        intr_count++;          /* line count */
    }
}
void timeIntr()             /* execute interrupt function */
{
    time_flg = ON;           /* turn on the interrupt flag */
}
```

- CPU

### Basic Items

- Counter Value

Counter values are used when the FRT is used as a timer. The counter value is explained below.

The count up cycle of the counter value varies depending on the period specification and graphics mode. Use the following formula to calculate the count-up cycle.

$$\text{Count Up Cycle (sec)} = (\text{cycle period}) \times 1 / \text{clock frequency (Hz)}$$

---

Conversion from the counter to microseconds and visa versa can be done conveniently if the following function format macro is used.

Function	Function Format Macro	No.
Convert counter value to microseconds	TIM_FRT_CNT_TO_MCR	15
Convert microseconds to counter value	TIM_FRT_MCR_TO_CNT	16

#### Usage Method

- **Initialization**

Execute the following function format macro before using the wait time (WAIT) function or fetching elapsed time.

Function	Function Format Macro	No.
Initialize FRT	TIM_FRT_INIT	8

- **Fetching Elapsed Time**

Use the following functions to obtain the elapsed time.

Function	Function Format Macro	No.
Set counter value (16 bit)	TIM_FRT_SET_16	9
Get counter value (16 bit)	TIM_FRT_GET_16	10

```
void sysInit()
{
    TIM_FRT_INIT(8);           /* Set period to 8 */
    .
}

void writeFrameBuff()
{
    Uint16 count;             /* store fetched counter value here */
    float micro_sec;          /* store microseconds here */
    TIM_FRT_SET_16(0);        /* set counter value to 0 */
    WriteAllVram();           /* execute process to be timed */
    count = TIM_FRT_GET_16()  /* get counter value */
    /* count shows the WriteAllVram() execution time */
    micro_sec = TIM_FRT_CNT_TO_MRC(count); /* convert counter value
                                              to microseconds */
    printDisplay(micro_sec); /* display elapsed time on the screen */
}
```

---

## 2.0 Reference

### 2.1 List of Function Format Macros

	Function	Function Format Macro	No.
SCU	Enable timer 0 interrupt	TIM_T0_ENABLE	1
	Disable timer 0 interrupt	TIM_T0_DISABLE	2
	Enable timer 1 interrupt	TIM_T1_ENABLE	3
	Disable timer 1 interrupt	TIM_T1_DISABLE	4
	Set timer 0 compare register	TIM_T0_SET_CMP	5
	Set timer 1 set data register	TIM_T1_SET_DATA	6
	Set timer 1 mode register	TIM_T1_SET_MODE	7
CPU	Initialize FRT	TIM_FRT_INIT	8
	Set counter value (16-bit)	TIM_FRT_SET_16	9
	Get counter value (16 bit)	TIM_FRT_GET_16	10
	Delay Time (16 bit)	TIM_FRT_DELAY_16	11
	Convert counter value to microseconds	TIM_FRT_CNT_TO_MCR	12
	Convert microseconds to counter value	TIM_FRT_MCR_TO_CNT	13
	Set timer interrupt enable register	TIM_FRT_SET_TIER	14
	Set timer control/status register	TIM_FRT_SET_TCSR	15
	Set free running counter	TIM_FRT_SET_FRC	16
	Set output compare register A	TIM_FRT_SET_OCRA	17
	Set output compare register B	TIM_FRT_SET_OCRB	18
	Set timer control register	TIM_FRT_SET_TCR	19
	Set timer output compare control register	TIM_FRT_SET_TOCR	20
	Get timer interrupt enable register	TIM_FRT_GET_TIER	21
	Get timer control/status register	TIM_FRT_GET_TCSR	22
	Get free running counter	TIM_FRT_GET_FRC	23
	Get output compare register A	TIM_FRT_GET_OCRA	24
	Get output compare register B	TIM_FRT_GET_OCRB	25
	Get timer control register	TIM_FRT_GET_TCR	26
	Get timer output compare control register	TIM_FRT_GET_TOCR	27
	Get input capture register A	TIM_FRT_GET_ICRA	28

## 2.2 Function Specifications

- SCU

Title	Function	Function Name	No.
Function Specifications	Enable timer 0 interrupt	TIM_T0_ENABLE	1

**Format:** void TIM\_T0\_ENABLE(void)

**Input:** None

**Output:** None

**Function Value:** None

**Function:** Enables timer 0 interrupt.

Title	Function	Function Name	No.
Function Specifications	Disable timer 0 interrupt	TIM_T0_DISABLE	2

**Format:** void TIM\_T0\_DISABLE(void)

**Input:** None

**Output:** None

**Function Value:** None

**Function:** Disables timer 0 interrupt.

Title	Function	Function Name	No.
Function Specifications	Enable timer 1 interrupt	TIM_T1_ENABLE	3

**Format:** void TIM\_T1\_ENABLE(void)

**Input:** None

**Output:** None

**Function Value:** None

**Function:** Enables timer 1 interrupt.

Title	Function	Function Name	No.
Function Specifications	Disable timer 1 interrupt	TIM_T1_DISABLE	4

**Format:** void TIM\_T1\_DISABLE(void)

**Input:** None

**Output:** None

**Function Value:** None

**Function:** Disables timer 1 interrupt.

Title	Function	Function Name	No.
Function Specifications	Set timer 0 compare register	TIM_T0_SET_CMP	5

**Format:** void TIM\_T0\_SET\_CMP(Uint32 time\_cmp)  
**Input:** time\_cmp : Timer 0 compare register value  
**Output:** None  
**Function Value:** None  
**Function:** Sets the timer compare register value to the timer 0 compare register.

Title	Function	Function Name	No.
Function Specifications	Set timer 1 set data register	TIM_T1_SET_DATA	6

**Format:** void TIM\_T1\_SET\_DATA(Uint32 time\_data)  
**Input:** time\_data : Timer 1 set data register value  
**Output:** None  
**Function Value:** None  
**Function:** Sets the timer 1 set data register value to the timer 1 set data register.

Title	Function	Function Name	No.
Function Specifications	Set timer 1 mode register	TIM_T1_SET_MODE	7

**Format:** void TIM\_T1\_SET\_MODE(Uint32 time\_mode)  
**Input:** time\_mode : Timer 1 mode register value  
**Output:** None  
**Function Value:** None  
**Function:** Sets the timer 1 mode register value to the timer 1 mode register.

---

- CPU

Title	Function	Function Name	No.
Function Specifications	Initialize FRT	TIM_FRT_INIT	8

**Format:** void TIM\_FRT\_INIT(Uint32 mode)

**Input:** mode : Period

Constant	Description
TIM-CKS-8	8
TIM-CKS-32	32
TIM-CKS-128	128

**Output:** None

**Function Value:** None

**Function:** Sets the selected period and initializes the FRT.

Title	Function	Function Name	No.
Function Specifications	Set counter value (16 bit)	TIM_FRT_SET_16	9

**Format:** void TIM\_FRT\_SET\_16(Uint16 cnt)

**Input:** cnt : Counter value

**Output:** None

**Function Value:** None

**Function:** Sets the 16-bit counter value to the FRT. The FRT counts from this value.

Title	Function	Function Name	No.
Function Specifications	Get counter value (16 bit)	TIM_FRT_GET_16	10

**Format:** Uint16 TIM\_FRT\_GET\_16(void)

**Input:** None

**Output:** None

**Function Value:** counter value

**Function:** Gets the 16-bit counter value of the FRT.

Title	Function	Function Name	No.
Function Specifications	Delay time (16 bit)	TIM_FRT_DELAY_16	11

**Format:** void TIM\_FRT\_DELAY\_16(Uint16 cnt)  
**Input:** cnt : Delay time counter value  
**Output:** None  
**Function Value:** None  
**Function:** Waits for the specified delay time counter value. Executes TIM\_FRT\_SET\_16 (0) internally and loops on TIM\_FRT\_GET\_16 (w\_cnt) until cnt == w\_cnt.

Title	Function	Function Name	No.
Function Specifications	Convert counter value to microseconds	TIM_FRT_CNT_TO_MCR	12

**Format:** Float TIM\_FRT\_CNT\_TO\_MCR(Uint32 count)  
**Input:** count : Counter value  
**Output:** None  
**Function Value:** Microseconds value  
**Function:** Converts specified counter value to a microsecond value.

Title	Function	Function Name	No.
Function Specifications	Convert microseconds to counter value	TIM_FRT_MCR_TO_CNT	13

**Format:** Uint32 TIM\_FRT\_MCR\_TO\_CNT(Float micro)  
**Input:** micro : microsecond value  
**Output:** None  
**Function Value:** Counter value  
**Function:** Converts microsecond value to a counter value.

Title	Function	Function Name	No.
Function Specifications	Set Timer Interrupt Enable Register	TIM_FRT_SET_TIER	14

**Format:** void TIM\_FRT\_SET\_TIER(Uint8 reg)  
**Input:** reg : Value  
**Output:** None  
**Function Value:** None  
**Function:** Sets a value to the timer interrupt enable register.

Title	Function	Function Name	No.
Function Specifications	Set timer control/status register	TIM_FRT_SET_TCSR	15

**Format:** void TIM\_FRT\_SET\_TCSR(Uint8 reg)  
**Input:** reg : Value  
**Output:** None  
**Function Value:** None  
**Function:** Sets the setting value to the timer control /status register.

Title	Function	Function Name	No.
Function Specifications	Set free running counter	TIM_FRT_SET_FRC	16

**Format:** void TIM\_FRT\_SET\_FRC(Uint16 reg)  
**Input:** reg : Value  
**Output:** None  
**Function Value:** None  
**Function:** Sets the setting value to the free running counter.

Title	Function	Function Name	No.
Function Specifications	Set output compare register A	TIM_FRT_SET_OCRA	17

**Format:** void TIM\_FRT\_SET\_OCRA(Uint16 reg)  
**Input:** reg : Value  
**Output:** None  
**Function Value:** None  
**Function:** Sets the setting value to output compare register A.

Title	Function	Function Name	No.
Function Specifications	Set Output Compare Register B	TIM_FRT_SET_OCRB	18

**Format:** void TIM\_FRT\_SET\_OCRB(Uint16 reg)  
**Input:** reg : Value  
**Output:** None  
**Function Value:** None  
**Function:** Sets the setting value to output compare register B.

Title	Function	Function Name	No.
Function Specifications	Set timer control register	TIM_FRT_SET_TCR	19

**Format:** void TIM\_FRT\_SET\_TCR(UINT8 reg)  
**Input:** reg : Set value  
**Output:** None  
**Function Value:** None  
**Function:** Sets the setting value to the timer control register.

Title	Function	Function Name	No.
Function Specifications	Sets timer output compare control register	TIM_FRT_SET_TOCR	20

**Format:** void TIM\_FRT\_SET\_TOCR(UINT8 reg)  
**Input:** reg : Set value  
**Output:** None  
**Function Value:** None  
**Function:** Sets the setting value to the timer output compare control register.

Title	Function	Function Name	No.
Function Specifications	Get timer interrupt enable register	TIM_FRT_GET_TIER	21

**Format:** UINT8 TIM\_FRT\_GET\_TIER(void)  
**Input:** None  
**Output:** None  
**Function Value:** Register value  
**Function:** Gets the timer interrupt enable register value.

Title	Function	Function Name	No.
Function Specifications	Get timer control/status register	TIM_FRT_GET_TCSR	22

**Format:** UINT8 TIM\_FRT\_GET\_TCSR(void)  
**Input:** None  
**Output:** None  
**Function Value:** Register value  
**Function:** Gets the timer control/status register value.

Title	Function	Function Name	No.
Function Specifications	Get free running counter	TIM_FRT_GET_FRC	23

**Format:** `Uint16 TIM_FRT_GET_FRC(void)`

**Input:** None

**Output:** None

**Function Value:** Register value

**Function:** Gets the free running counter value.

Title	Function	Function Name	No.
Function Specifications	Get output compare register A	TIM_FRT_GET_OCRA	24

**Format:** `Uint16 TIM_FRT_GET_OCRA(void)`

**Input:** None

**Output:** None

**Function Value:** Register value

**Function:** Gets the output compare register A value.

Title	Function	Function Name	No.
Function Specifications	Get output compare register B	TIM_FRT_GET_OCRB	25

**Format:** `Uint16 TIM_FRT_GET_OCRB(void)`

**Input:** None

**Output:** None

**Function Value:** Register value

**Function:** Gets the output compare register B value.

Title	Function	Function Name	No.
Function Specifications	Get timer control register	TIM_FRT_GET_TCR	26

**Format:** `Uint8 TIM_FRT_GET_TCR(void)`

**Input:** None

**Output:** None

**Function Value:** Register value

**Function:** Gets the timer control register value.

---

Title	Function	Function Name	No.
Function Specifications	Get timer output compare control register	TIM_FRT_GET_TOCR	27

**Format:** `Uint8 TIM_FRT_GET_TOCR(void)`

**Input:** None

**Output:** None

**Function Value:** Register value

**Function:** Gets the timer output compare control register value.

Title	Function	Function Name	No.
Function Specifications	Get input capture register A	TIM_FRT_GET_ICRA	28

**Format:** `Uint16 TIM_FRT_GET_ICRA(void)`

**Input:** None

**Output:** None

**Function Value:** Register value

**Function:** Gets the interrupt capture register A value.

---

# Debug Support Library

## 1.0 Guide

### 1.1 Objective

Incorporating the Debug Support Library in the application during code development enables various debug functions such as the display of character strings to the screen, data input using a software “keyboard”, and read / writes to memory.

### 1.2 Description

- This library uses the VDP2 graphics library and constructs a simple debug environment using the normal scroll screen NBG0. The Debug Support Library cannot be used if the VDP2 library is not used.
- This library only supports the standard SATURN control pad. This library cannot be used if the peripheral library is not used.

### 1.3 Coding Example

The following is an example of an actual C language program.

```
#include    "sega_scl.h"
#include    "sega_dbg.h"

void      err(Uint8 *mess);
Uint16 PadData;

main()
{
    SCL_Vdp2Init();
    DBG_Initial(&PadData,RGB16_COLOR(31, 31, 31),0);
    . . . . . . . . . . /*  program   */
    if (. . . . . . .)
        err("error occurred in main() ~ function");
    . . . . . . . . . . /*  program   */

}

void err(Uint8 *mess)
{
    DBG_Printf("%s\n",mess); /*  display message   */
    DBG_Monitor(): /*  go to simple debug environment */
}
```

## 2.0 Reference

### 2.1 List of Functions

Function	Name	No.
Initialize debug environment	DBG_Initial	1
Turn on debug screen display	DBG_DisplayOn	2
Turn off debug screen display	DBG_DisplayOff	3
Clear debug screen	DBG_ClearScreen	4
Set debug screen cursor	DBG_SetCursor	5
Display character string on debug screen	DBG_Printf	6
Key input by software keyboard	DBG_GetKeyStr	7
Simple debug process	DBG_Monitor	8

### 2.2 Function Specifications

Title	Function	Function Name	No.
Function Specifications	Initialize debug environment	DBG_Initial	1

**Format:** void DBG\_Initial (Uint16 \*PadData, Rgb16 charColor, Rgb16 backColor)

**Input:** apPadData : Control pad data pointer  
Specify the pad data area that is used to set the pad data obtained by using the peripheral library within a user V-Blank routine. Pad data should be set by flipping 0 and 1.  
charColor : The character display color is specified by 16-bit RGB values.  
backColor : The color of the background display is specified by 16-bit RGB values. Transparent when 0 is specified.

**Output:** None

**Function Value:** None

**Function:** Sets the program error display routine entry to the Debug Support Library initialization and the following program exception interrupt vectors.

- General invalid instruction interrupt
- Invalid slot instruction interrupt
- CPU address error interrupt
- DMA address error interrupt

When an error occurs, the program error display routine displays the source of the error, the current register content, the stack area content, and then passes control to a simple monitor.

**Remarks:** Users using the Debug Support Library must execute this function first.

Title	Function	Function Name	No.
Function Specifications	Turn on debug screen display	DBG_DisplayOn	2

**Format:** void DBG\_DisplayOn(void)

**Input:** None

**Output:** None

**Function Value:** None

**Function:** Displays the debug screen.

Title	Function	Function Name	No.
Function Specifications	Turn off debug screen display	DBG_DisplayOff	3

**Format:** void DBG\_DisplayOff(void)

**Input:** None

**Output:** None

**Function Value:** None

**Function:** Does not display the debug screen.

Title	Function	Function Name	No.
Function Specifications	Clear debug screen	DBG_ClearScreen	4

**Format:** void DBG\_ClearScreen(void)

**Input:** None

**Output:** None

**Function Value:** None

**Function:** Erases data written on the debug screen.

Title	Function	Function Name	No.
Function Specifications	Set debug screen cursor	DBG_SetCursor	5

**Format:** void DBG\_SetCursor(Uint16 x, Uint16 y)

**Input:** x : x coordinate

y : y coordinate

**Output:** None

**Function Value:** None

**Function:** Specifies the cursor position.

Title	Function	Function Name	No.
Function Specifications	Display character string on debug screen	DBG_Printf	6

**Format:** void DBG\_Printf(const char \*control, . . .)  
**Input:** control : Pointer to character string that represents the expression  
... : Parameter list  
**Output:** None  
**Function Value:** None  
**Function:** Writes character strings to the debug screen. Used in the same manner as printf().  
**Example:** sample ()  
{  
    DBG\_Printf("%s\n", "SEGA");  
}

Title	Function	Function Name	No.
Function Specifications	Key input by software keyboard	DBG_GetKeyStr	7

**Format:** char \*DBG\_GetKeyStr(char \*KeyStr)  
**Input:** KeyStr : Input key string return area pointer.  
**Output:** KeyStr : Input key string data.  
**Function Value:** Specified input key string return area pointer.  
**Function:** A software "keyboard" is displayed on the debug screen and a character string is retrieved.

Key pad operation:

R	: Move key select cursor right
L	: Move key select cursor left
U	: Move key select cursor up
D	: Move key select cursor down
B + R	: Move key window right
B + L	: Move key window left
B + U	: Move key window up
B + D	: Move key window down
A	: Input key select cursor position character
C	: Delete input character (backspace)
START	: End input (return)
X	: Display / don't display sprite screen
Y	: Display / don't display scroll screen (other than NBG0 screen)

---

Title	Function	Function Name	No.
Function Specifications	Simple debug process	DBG_Monitor	8

**Format:** void DBG\_Monitor(void)  
**Input:** None  
**Output:** None  
**Function Value:** None  
**Function:** The following commands are used to display and edit memory. Data within square brackets [ ] can be omitted, and items separated by a back slash (/) can be selected.

- **Display Memory**

```
D fromAddr [toAddr/@size] [;B/W/L] <RTN>
    fromAddr      : Dump start address
    toAddr        : Dump end address
    @size         : Dump size (HEX)
    B             : Byte (8 bit) display
    W             : Word (16 bit) display
    L             : Long (32 bit) display
```

After inputting the D command, the memory dump may be continued by entering <RTN>.

- **Change Memory Contents**

```
D toAddr [data] [;B/W/L] <RTN>
    toAddr        : Memory address
    data          : Edit data (HEX)
    B             : Byte (8 bits) data
    W             : Word (16 bits) data
    L             : Long (32 bits) data
```

If data is omitted, input sequential data from the specified address when the data entry prompt is displayed. If “^” is input at this time, the edit address is decremented; if <RTN> is input, the address is incremented. The memory edit ends when “.” is input.

- **Display Register Contents**

R<RTN>

Valid only when calling from the exception process routine of the program.

- **Quit Simple Debug Process**

Q<RTN>

Returns to the calling point of this routine. Invalid when called from the exception processing routine of the program.

- **Display Command Help**

H<RTN>

---

# Compression Decompression Library

## Terminology

Term	Description
Compression rate	Equal to (Size after compression / Size before compression) X 100 [%]
Symbol (character)	1 processing unit of input data for the compression process. BYTE, WORD, DWORD can be specified in this library.
Code	Smallest unit of compression process output data. Defined per method.
Run Length Encoding (RLE)	Technique for compressing data based on replacing input data with a run length value and a corresponding value for that run length.
Run length	Number of occurrences of the same input value.
Non-pattern phrase	A set of data with a short “run” that results in the creation of more data under the conventional RLE method.

---

## 1.0 Guide

### 1.1 Application

The Compression-Decompression Library is used to decode run length encoded (compressed) data. Run length encoding is done via a utility called `CMPRUN.EXE` (provided separately).

### 1.2 Compression Method

This implementation of RLE has additional processing for handling non-pattern phrases. Since a simple RLE algorithm has a tendency of having reduced compression efficiency when processing non-pattern phrases, those data are organized and processed together.

Pattern phrases are represented by a run length and a value. Non-pattern phrases are represented as non-pattern lengths and data values (i.e. uncompressed data). In order to differentiate between code that represent pattern and non-pattern data, non-pattern data are expressed as negative values (two's complement).

Data with short run lengths tend to reduce the processing efficiency of non-pattern phrases as well as the overall compression rate. Those data are handled in the same manner as non-pattern phrases.

#### Non-Pattern Phase Definition

A non-pattern phrase data set is defined as the following start and end conditions.

- Start condition of non-pattern phrase data:  
Symbol (character) with a run length of 1.
- End condition of non-pattern phrase data.
- Non-pattern phrase with a run length of 3 or greater  
OR

When the pattern phrase length can be expressed in terms of the processing byte number unit.

---

### Example of Non-Pattern Phrase

Figure 1.1 shows an example of a non-pattern phrase data.

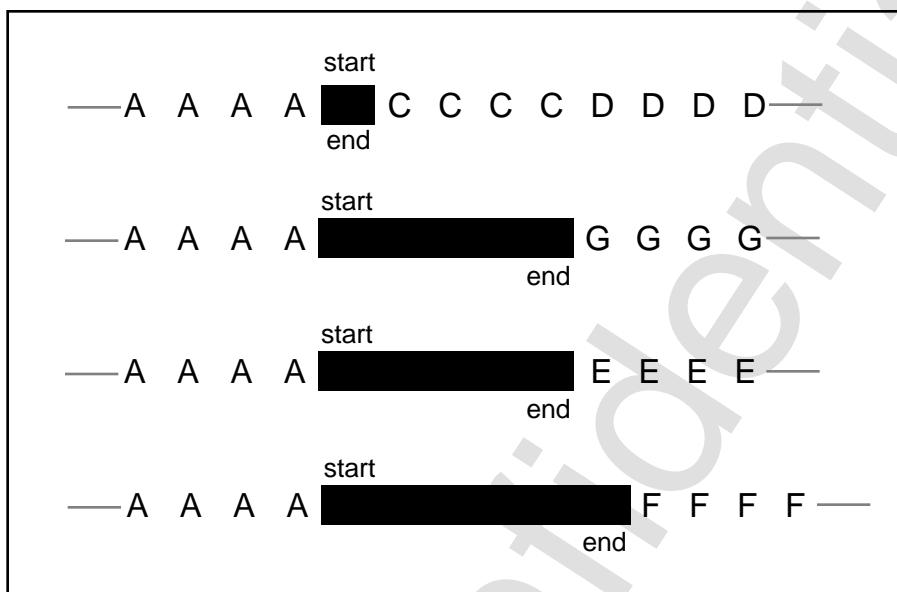


Figure 1.1 Non-pattern phrase data

### Processing Unit

As shown in the following table, there are three processing unit types. One character (symbol) is input from the start and broken up into the processing unit size.

Table 1.1 Process Units

Unit	Representation of compression processing units for input data
BYTE (1 byte)	[■] [■] [■] [■] [■] [■] [■] [■] [■] [■] [■] [■] —
WORD (2 bytes)	[■■] [■■] [■■] [■■] [■■] [■■] [■] —
DWORD (4 bytes)	[■■■■] [■■■■] [■■■■] [■] —

\* Values are in hexadecimal. ■ corresponds to 1 byte.

### Expression of Run Length

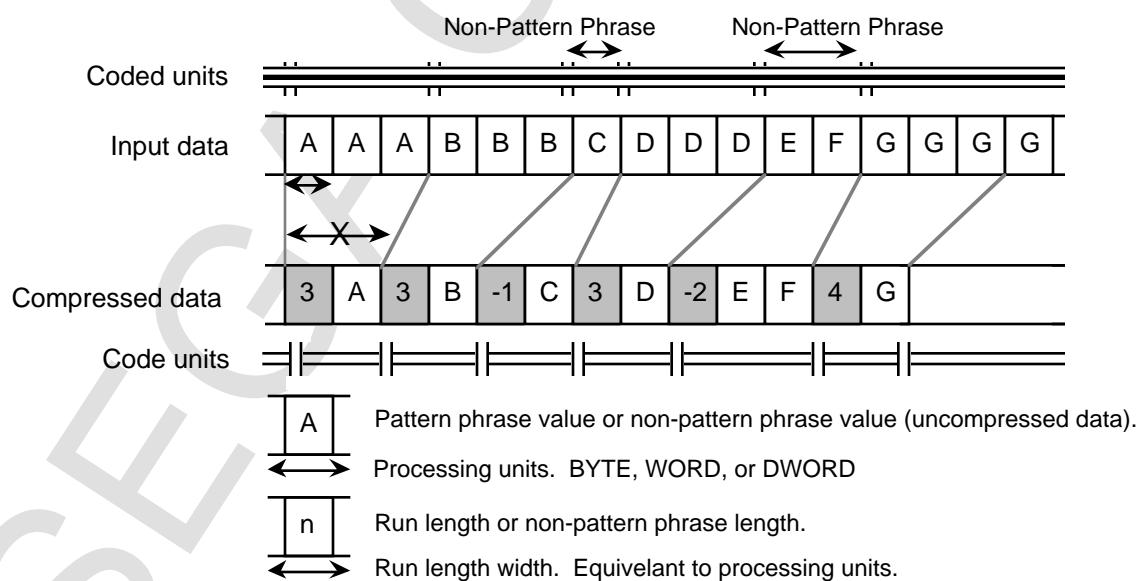
A normal run length (pattern phrase length) is expressed as a positive number, and the non-pattern phrase length is expressed as a negative number. Because a run length is always 2 or greater, it is expressed as a value that is *value* - 2. The byte number used for expressing run length is the same as the processing unit byte number. The actual amount of data for the run length is decoded by the processing units. That is, a processing unit in words with a run length of 5 is equivalent to the encoding process for 10 bytes of data. The following table shows the range of run length values that can be expressed.

**Table 1.2 Run Length Widths and Expressions**

Processing Units	Run Length (Pattern Phrase Length)	(Non-Pattern Phrase Length)	
	Representation (Description)	Representation (Description)	
BYTE 1 byte	2 ~ 129 (00h ~ 7Fh)	2 ~ -128 (FFh ~ 80h)	
WORD 2 bytes	2 ~ 32769 (0000h ~ 7FFFh)	2 ~ -32769 (FFFFh ~ 80000h)	
DWORD 3 bytes	2 ~ $2^{31} + 2$ (00000000h ~ 7FFFFFFFh)	2 ~ $-2^{31}$ (FFFFFFFh ~ 8000000h)	

### Processing Image Diagram

Figure 1.2 is a processing image of the compression algorithm using the run length/non-pattern phrase process.



**Figure 1.2 Run length/non-pattern phrase processing**

## Compressed File Format

Figure 1.3 and the table on the next page show the run length compression file format.

Compressed File (2 Byte)	Header 0 (K BYTE)	Source size 1 (Variable Length)	Code (Variable Length)	Code Z		Code (Variable Length)
--------------------------------	-------------------------	--	------------------------------	-----------	--	------------------------------

K=2: When source size width is 2 bytes.

K = 6: When source size width is 4 bytes. (The last 2 bytes of the header are omitted to keep the data in the 4 byte boundary.)

Two types of codes exist, and are differentiated by whether the run length is a positive or a negative value.

Code (When run length $\geq 0$ ) Add 2 and interpret as pattern phrase length	Run length n-2 <pattern phrase length> (M BYTE)	Pattern phrase length (M BYTE)
--	---	-----------------------------------

M = Processing unit byte number

n: number of units			
Code (When run length $< 0$ ) Flip negative sign and interpret as pattern phrase length	Run length -n <non-pattern phrase length> (M BYTE)	Non-pattern phrase value 0 <uncompressed data> (M BYTE)	Non-pattern phrase value n-1 <uncompressed data> (M BYTE)

M = Processing unit byte number

Figure 1.3 Run length compression file format

**Table 1.3 Run length compression file header**

Meaning	Bit Position	MSB	LSB
		15 12 11 8 7 4 3 0 □□□□ □□□□ □□□□ □□□□ □□□□ □□□□ □□□□	
Compression Algorithm	15 .. 12	[■■■■] [■■■■] [■■■■] [■■■■] [■■■■] [■■■■]	0 0 0 1
Run length Reserved			
Processing Units	11 .. 10	[■] [■■■■] [■■■■] [■■■■] [■■■■] [■■■■]	00 01 11
1 byte			
2 bytes			
4 bytes			
Reserved	9 .. 4		0 0 0 0 [■■]
Source size width	3		[■] 0 1
2 bytes			
4 bytes			
Reserved	2 .. 0		0 0 0

## 1.3 Decompression Library

### Overview

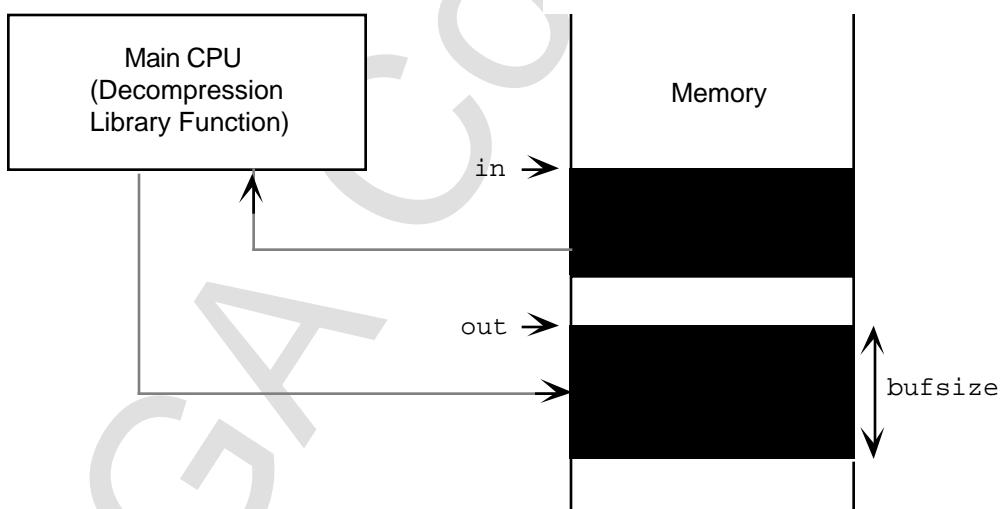
The Decompression Library expands data compressed by the compression tool. Compressed data may be handled as discrete files on the CD-ROM itself or stored by other means. However, when the data is decompressed using the Decompression Library, an input and output data buffer must be set up in a main CPU accessible memory area.

Both of these buffers must be reserved for the Decompression Library, and their addresses specified in the library functions.

For example, when input data is read off the CD to be decompressed, the Decompression Library functions must be executed after all of the input data used for a single decompression pass is read into memory.

The output data buffer must also be specified in the Decompression Library functions. When the function exceeds this buffer size, it will halt processing and stop the output of data. It is possible to use this feature as a means to perform partial decompression processing. However, note that there is no feature to resume the interrupted decompression process.

Figure 1.4 gives an overview of the decompression process.



**in:** Input buffer start address. Specify in terms of a 4-byte boundary.

**out:** Output buffer start address. Specify in terms of a 4-byte boundary.

**bufsize:** Output buffer size. Integer multiple of the processing unit byte number.

**Figure 1.4 Decompression Process**

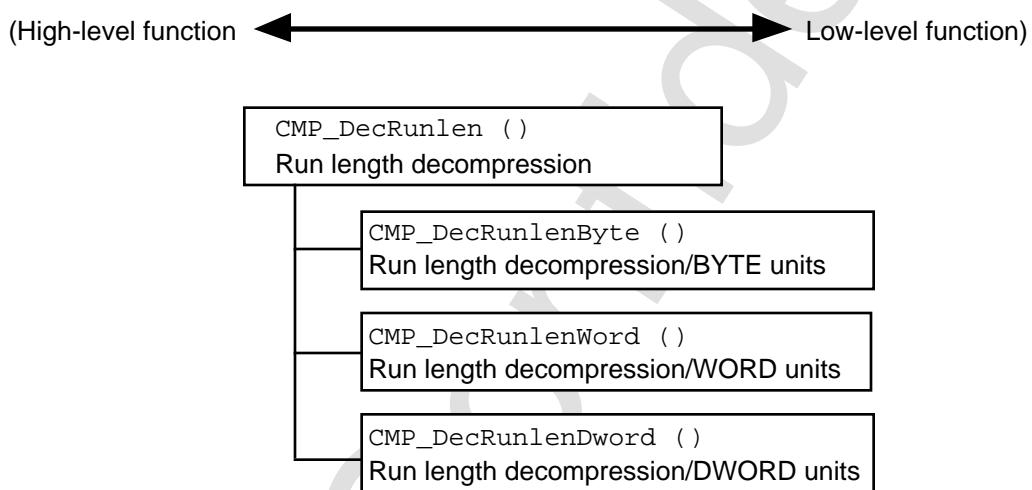
---

## Module Configuration

The Decompression Library function has a hierarchical module configuration in terms of the features of each function. The high-level functions are easier to use since they interpret the compressed data's algorithm and parameter attributes and perform decompression processing. The downside is that additional object code must be linked.

The low-level functions may be used when it is desirable to link only the minimum amount of object code that is necessary. If this approach is taken, it is necessary to make sure that there are no functional discrepancies between the output of the compression tool and the processing of the library function.

Figure 1.5 shows the configuration of the Decompression Library modules.



**Figure 1.5 Module Configuration**

---

## Usage Examples

The following programming examples show the use of the library functions included in CMPLIB.LIB. The use of these functions assumes that there is compression data in memory that can be directly accessed from the main CPU.

### Example 1

Figure 1.6 shows an example in which compressed data is included in the source file as a C language data array.

```
#include "cmplib.h"
/* RLE compressed data converted from binary to text data

char cmpdata [ ] = {
    0x10, 0x01, 0x04, . . . .
}
/* decompressed data buffer
char outputbuf[4096];

main()
{
    /* decompressed data pointer
    char *bufp;
    /* set to start of decompressed data buffer
    bufp = outputbuf;
    /* run length dictionary decompression
    CMP_DecRunlen (cmpdata, &bufp, sizeof(outputbuf));
    /* use of expanded data
}
```

Figure 1.6 Decompression Library Usage Example 1

---

### **Example 2**

Figure 1.7 shows an example in which compressed data is read from the CD-ROM and decompressed.

```
#include "sega_gfs.h"
#include "cmplib.h"

/* file read buffer
Uint8 readbuf[READ_SIZE]
/* expanded data buffer
Uint8 outputbuf[4096]

main()
{
    GfsFid fid;      /* file identifier
    Sint32 fsize;   /* file size
    char *bufp;  */
    fid = 5;        /* Set compressed data file identifier
    fsize=GFS_Load(fid, 0, readbuf, READBUF_SIZE) file batch read

    /* set to start of decompression data buffer
    bufp = outputbuf;

    /* run length decompression
    CMP_DecRunlen (cmpdata, &bufp, sizeof (outputbuf));

    /* use of decompressed data
    |
}
```

**Figure 1.7 Decompression Library Usage Example 2**

---

## 2.0 Reference

### 2.1 Data Specifications

The following table shows the basic types used in this library.

**Table 2.1 Basic Types**

Name	Description
Uint8	Unsigned 1-byte integer
Sint8	Signed 1-byte integer
Uint16	Unsigned 2-byte integer
Sint16	Signed 2-byte integer
Uint32	Unsigned 4-byte integer
Sint32	Signed 4-byte integer
Bool	Boolean type. Takes the following values: FALSE TRUE

The following table shows the constant macros defined in this library.

**Table 2.2 Constant Macros**

Macro Name	Value	Description
CMP_DEC_OK	0	Decompression function return value. Normal end.
CMP_DEC_STOP	1	Decompression function return value. Normal end. Halt decompression process.
CMP_DEC_ERR	-1	Decompression function return value. Abnormal end. Input data error.
CMP_DEC_ERR_H_ALGO	-2	Decompression function return value. Abnormal end. Unsupported algorithm.
CMP_DEC_ERR_H_UNIT	-3	Decompression function return value. Abnormal end. Unsupported processing units.

---

## 2.2 List of Functions

The following table lists the functions of the Decompression Library.

**Table 1.6 List of Data Decompression Library Functions**

Function	Name	No.
Run length decompression	CMP_DecRunlen	1
Run length decompression/BYTE units	CMP_DecRunlenByte	1.1
Run length decompression/WORD units	CMP_DecRunlenWord	1.2
Run length decompression/DWORD units	CMP_DecRunlenDword	1.3

## 2.3 Function Specifications

Title	Function	Function Name	No.
Function Specifications	Run length decompression	CMP_DecRunlen	1
<b>Format:</b>	Sint32 CMP_DecRunlen(void *in, void **out, Sint32 size)		
<b>Argument:</b>	in (input) : Compressed data input buffer pointer out (input) : Address of decompressed data output buffer pointer (output) : Decompressed data output end pointer bufsize (input) : Output buffer size [BYTE]		
<b>Return Value:</b>	Processing results: CMP_DEC_OK (0) : Normal end. Input data decompression completed. CMP_DEC_STOP (1) : Decompressed data to output buffer size. CMP_DEC_ERR (-1) : Abnormal end. Input data error. CMP_DEC_ERR_H_ALGO (-2) : Abnormal end. Unsupported algorithm. CMP_DEC_ERR_H_UNIT (-3) : Abnormal end. Unsupported processing units.		
<b>Function:</b>	Expands compressed data. This function processes the compressed data created by the run length compression tool <b>CMPRUN.EXE</b> . This function interprets the header of the input data and executes the decompression function that supports the compression parameters. Compressed data is read from <b>in</b> . Decompressed data is written to <b>*out</b> . *Out performs a post increment for each 1 byte written. The size of decompressed data can be determined by comparing the difference between the <b>*out</b> value set before and after the function is executed. The process is stopped if decompressed data exceeds <b>bufsize</b> [byte]. Decompressed data is written to the available memory in the buffer even when the process is stopped.		
<b>Note:</b>	The input buffer and the output buffer must be reserved. The start address of each buffer must be specified in terms of a 4-byte boundary. The output buffer must be large enough to store decompressed data and be an integer multiple of the processing unit number.		

### Remarks:

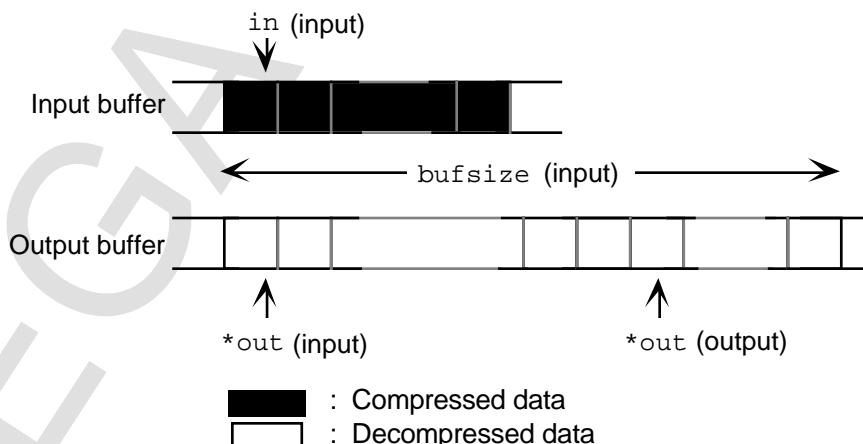


Figure 2.1 Description of Decompression Function Parameters

Title	Function	Function Name	No.
Function Specifications	Run length decompression / BYTE units	CMP_DecRunlenByte	1.1

**Format:** Sint32 CMP\_DecRunlenByte(void \*in, void \*\*out, Sint32 size)

**Argument:**

in	(input)	: Compressed data input buffer pointer
out	(input)	: Address of decompressed data output buffer pointer
	(output)	: Decompressed data output end pointer
bufsize	(input)	: Output buffer size [BYTE]

**Return Value:** Processing results

CMP_DEC_OK	(0)	: Normal end. Input data decompression completed.
CMP_DEC_STOP	(1)	: Decompressed data to output buffer size.
CMP_DEC_ERR	(-1)	: Abnormal end. Input data error.
CMP_DEC_ERR_H_ALGO	(-2)	: Abnormal end. Unsupported algorithm.
CMP_DEC_ERR_H_UNIT	(-3)	: Abnormal end. Unsupported processing units.

**Function:** Expands compressed data.  
 Processes byte unit RLE compressed data.  
 Compressed data is read from in. Decompressed data is written to \*out.  
 \*Out performs a post increment for each 1 byte written. The size of decompressed data can be determined by comparing the difference between the \*out value set before and after the function is executed. The process is stopped if decompressed data exceeds bufsize [byte]. Decompressed data is written to the available memory in the buffer even when the process is stopped.

**Note:** The input buffer and the output buffer must be reserved. The start address of each buffer must be specified in terms of a 4-byte boundary. The output buffer must be large enough to store decompressed data and be an integer multiple of the processing unit number.

**Remarks:** The function interface is the same as CMP\_DecRunlen().  
 See Figure 2.1.

Title	Function	Function Name	No.
Function Specifications	Run length decompression / WORD units	CMP_DecRunlenWord	1.2

**Format:** Sint32 CMP\_DecRunlenWord(void \*in, void \*\*out, Sint32 size)

**Argument:**

in	(input)	: Compressed data input buffer pointer
out	(input)	: Address of decompressed data output buffer pointer
	(output)	: Decompressed data output end pointer
bufsize	(input)	: Output buffer size [BYTE]

**Return Value:**

Processing results		
CMP_DEC_OK	( 0 )	: Normal end. Input data decompression completed.
CMP_DEC_STOP	( 1 )	: Decompressed data to output buffer size.
CMP_DEC_ERR	( -1 )	: Abnormal end. Input data error.
CMP_DEC_ERR_H_ALGO	( -2 )	: Abnormal end. Unsupported algorithm.
CMP_DEC_ERR_H_UNIT	( -3 )	: Abnormal end. Unsupported processing units.

**Function:**

Expands compressed data.  
 Processes word unit RLE compressed data.  
 Compressed data is read from *in*. Decompressed data is written to *\*out*.  
*\*out* performs a post increment for each 1 byte written. The size of decompressed data can be determined by comparing the difference between the *\*out* value set before and after the function is executed. The process is stopped if decompressed data exceeds *bufsize* [byte]. Decompressed data is written to the available memory in the buffer even when the process is stopped.

**Note:** The input buffer and the output buffer must be reserved. The start address of each buffer must be specified in terms of a 4-byte boundary. The output buffer must be large enough to store decompressed data and be an integer multiple of the processing unit number.

**Remarks:**

The function interface is the same as CMP\_DecRunlen().  
 See Figure 2.1.

Title	Function	Function Name	No.
Function Specifications	Run length decompression / DWORD units	CMP_DecRunlenDword	1.3

**Format:** Sint32 CMP\_DecRunlenDword(void \*in, void \*\*out, Sint32 size)

**Argument:**

in	(input)	: Compressed data input buffer pointer
out	(input)	: Address of decompressed data output buffer pointer
	(output)	: Decompressed data output end pointer
bufsize	(input)	: Output buffer size [BYTE]

**Return Value:**

Processing results		
CMP_DEC_OK	( 0 )	: Normal end. Input data decompression completed.
CMP_DEC_STOP	( 1 )	: Decompressed data to output buffer size.
CMP_DEC_ERR	( -1 )	: Abnormal end. Input data error.
CMP_DEC_ERR_H_ALGO	( -2 )	: Abnormal end. Unsupported algorithm.
CMP_DEC_ERR_H_UNIT	( -3 )	: Abnormal end. Unsupported processing units.

**Function:**

Expands compressed data.  
 Processes DWord unit RLE compressed data.  
 Compressed data is read from *in*. Decompressed data is written to *\*out*.  
*\*out* performs a post increment for each 1 byte written. The size of decompressed data can be determined by comparing the difference between the *\*out* value set before and after the function is executed. The process is stopped if decompressed data exceeds *bufsize* [byte]. Decompressed data is written to the available memory in the buffer even when the process is stopped.

**Note:** The input buffer and the output buffer must be reserved. The start address of each buffer must be specified in terms of a 4-byte boundary. The output buffer must be large enough to store decompressed data and be an integer multiple of the processing unit number.

**Remarks:**

The function interface is the same as *CMP\_DecRunlen()*. See Figure 2.1.

---

# PCM-ADPCM Playback Library

## 1.0 Overview

### 1.1 Objective

The PCM-ADPCM Playback Library enables simplified playback of PCM audio on the SEGA SATURN.

### 1.2 Features

#### Easy PCM Playback

The user simply makes calls to the PCM task function periodically in order to perform audio playback. The library reads from the CD, controls the ring buffer, manages timing, supplies data to the PCM buffer, and issues commands to the sound driver.

#### ADPCM Support

This library is capable of decompressing ADPCM compressed data on-the-fly while playback occurs. ADPCM is a CD-ROM XA audio data format standard that provides superior sound quality at a compression ratio of approximately 4:1.

#### Three Types of File Formats

- (1) AIFF format (uncompressed PCM)
- (2) CD-ROM XA audio format (ADPCM compression)
- (3) User-specified ADPCM format (OptImage Interactive Services AudioStack file output format)

#### Pause Function

Although the hardware has no function to pause PCM playback, this library makes pausing simple.

#### Multiple Stream Playback

This feature allows four PCM streams to be played at the same time. For example, this can be used for a game where overlapping voice-overs from three characters can be played back asynchronously with an additional background soundtrack.

### 1.3 System Image

Figures 1.1 and 1.2 show the configuration of the PCM-ADPCM Playback Library.

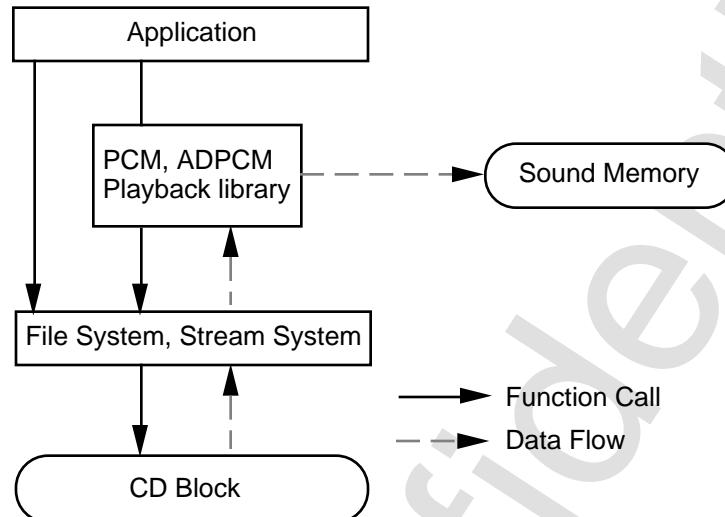


Figure 1.1 Playback of Uncompressed PCM Data

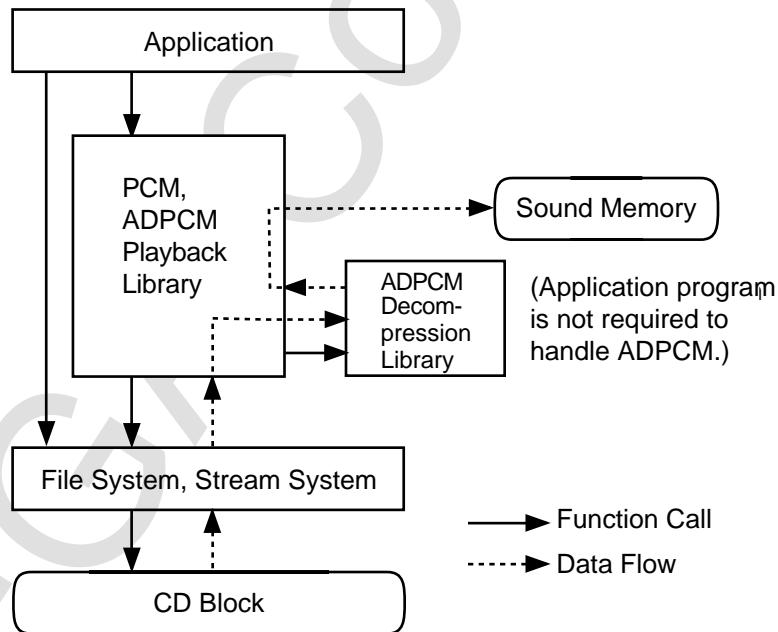


Figure 1.2 Playback of Compressed ADPCM Data

## 2.0 Specifications

The following two tables list the specifications of the PCM-ADPCM Playback Library.

**Table 2.1 Library Specifications (1)**

Item	Specification	Remarks
Sampling frequency	Maximum 44.1 kHz	
Bit resolution	8 bit, 16 bit	
Number of channels	Mono, stereo	
File format	AIFF User-specified ADPCM (1) CD-ROM XA Audio (2)	PCM Uncompressed ADPCM Compressed ADPCM Compressed
CD-ROM format	Mode 2 Form 2 for CD-ROM XA Audio format Mode 1 or Mode 2 Form 1 for other cases	
Playback modes	Memory Playback Mode (plays data in memory) File Playback Mode (reads and plays from CDs) Stream Playback Mode (reads and plays from CDs)	(3) Uses the File System Uses the Stream System
Functions	Branching playback, continuous playback Multiple stream playback (can play up to 4 streams at the same time) Pause, stop, volume/pan setting	Only 1 ADPCM stream can be played.
Continuous play time	Maximum 1 hour	
Libraries used	File System, Stream System, DMA Library, ADPCM Decompression Library	These libraries must be linked along with the PCM-ADPCM Playback Library (4)
CPU timer (FRT) usage	This library uses the CPU timer (FRT) set to the clock count of 128. Initialization is done only once within the <code>PCM_Init</code> function. User FRT initialization and settings are prohibited.	Values can be obtained by <code>TIM_FRT_GET_16</code>
Sound driver	The application performs initialization settings (68000 reset, sound driver transfer, etc.). This library issues commands for PCM playback (PCM start, stop, parameter change). When the application issues a command, interrupts must be disabled from the point prior to the clearing of the command block until the completion of the command settings.	See the <i>Saturn Sound Driver System Interface</i> manual (Doc. # ST-166-R2-091394).  Both this library and the Sound Interface Library are structured so that they do not override each other's commands.

**Table 2.2 Library Specifications (2)**

Item	Specification			Remarks
Buffer requirements	Work buffer: Work structure (about 530 bytes) Ring buffer: Sector size * 10 bytes ~ PCM buffer: 4096 * 2 ~ 4096 * 4 sample/1 ch Pause processing work: 4096 samples ~ PCM buffer size			Fixed size (5) (6) Required only when pause is executed.
CPU overhead (7) (CPU task function overhead ratio after play starts)	PCM uncompressed 44 kHz stereo 16 bit : 10% PCM uncompressed 44 kHz mono 16 bit : 6% ADPCM compressed 44 kHz stereo 16 bit : 33% ADPCM compressed 37.8 kHz stereo 16 bit : 29% ADPCM compressed 22 kHz stereo 16 bit : 17% ADPCM compressed 11 kHz stereo 16 bit : 8% ADPCM compressed 11 kHz mono 16 bit : 4%			These values covers all processing times, such as data transfers from the CD block through work memory to sound memory, and in the case of ADPCM, decompression processing.
PCM_Task (task function) specification	<ul style="list-style-type: none"> <li>Called at a frequency equal to or greater than the V-Blank interrupt frequency (once every 16 ms).</li> <li>Single-session task function maximum processing time</li> </ul> PCM uncompressed stereo 16 bit : 15 ms PCM uncompressed mono 16 bit : 8 ms ADPCM compressed stereo 16 bit : 34 ms ADPCM compressed mono 16 bit : 15 ms			If called at an appropriate frequency, longer processing occurs once per several times of processing.

**Notes:**

- (1) The “user-specified ADPCM format” refers to the data file format output by an AIFF to ADPCM file converter utility called **AudioStack** by OptImage Interactive Services.
- (2) CD-ROM XA Audio has 16-bit data ADPCM compressed to 4 bits and has the following formats:  
 Mode B (37.8 kHz) stereo/mono  
 Mode C (18.9 kHz) stereo/mono  
 Only the Stream Playback Mode can be used when CD-ROM XA Audio is used.  
 For more information, see the *CD-ROM XA Standards*.
- (3) The Memory Playback Mode can be used in two ways:
  - To play small files, prepared as memory-based files.
  - To play data read by the application from the CD through a series of ring buffers.
- (4) There are also object files that no not need to be linked depending on the functions used.  
 The ADPCM Decompression Library does not need to be linked if the ADPCM usage declaration `PCM_DeclareUseAdpcm` is not made. In Memory Playback Mode, the File System and Stream System are not linked. In File Playback Mode, the Stream System is not linked. The File System and the Stream System are linked if the Stream Playback Mode is used.

- 
- (5) During branching playback or when data is supplied to the ring buffer using memory playback, the danger of playback interruptions are reduced by using a larger ring buffer. The larger the ring buffer, the greater the safety margin for playing PCM audio. However, the drawback is that each task's processing time becomes somewhat unstable.

When all files are kept in memory for memory playback, the addresses and files sizes for each file must be specified. These values need not be integer multiples of the sector size.

- (6) A PCM buffer size of 4096 \* 2 [sample/1 ch] is adequate if there are no special reasons for a larger buffer.) By allocating enough memory, playback will complete without glitches even when the task function cannot be called at the required intervals.
- (7) The CPU overhead (CPU overhead ratio for the task function after playback begins) is defined as follows.

$$R = (100 \times T_{task}) / T_{play}$$

R = CPU overhead [%]

T<sub>task</sub> = Total PCM\_Task  
processing time from start to end of playback.

T<sub>play</sub> = Playback time

### 3.0 Playback Sequence

#### Playback Sequence

Figure 3.1 shows the sequence for audio playback using the Stream System.

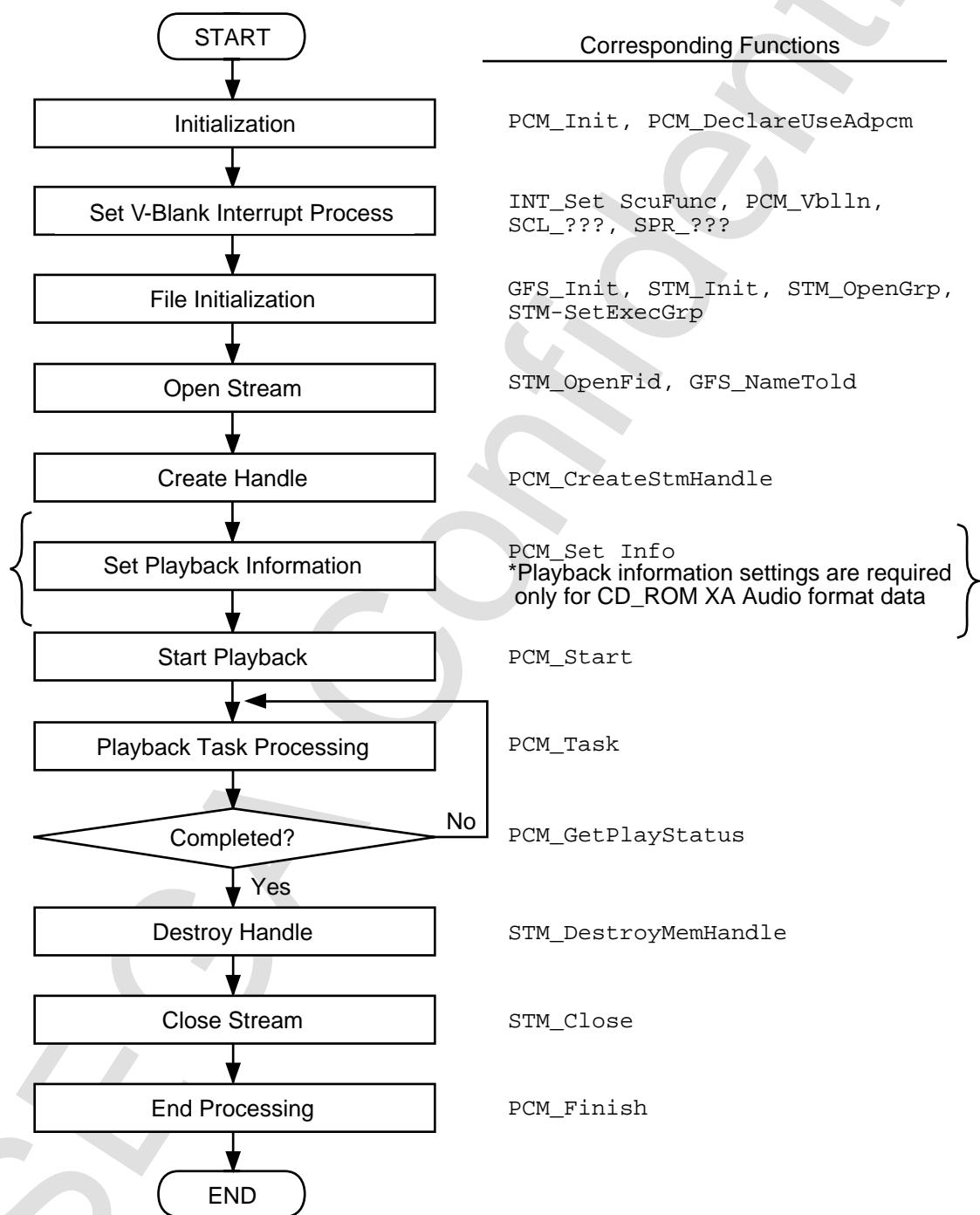


Figure 3.1 Sequence for Stream Playback

---

## Sample Program

The following example is a program that plays either AIFF or ADPCM format files using the Stream System Library.

```
#define RING_BUF_SIZE  (2048L*10)
#define PCM_ADDR   ((void*)0x25a20000)
#define PCM_SIZE   (4096L*2)

/* Work */
PcmWork  pcm_work;

/* Ring Buffer */
Uint32  ring_buf[RING_BUF_SIZE / sizeof(Uint32)];
StmHn      stm;
PcmHn      pcm;

/* Initialization */
PCM_Init();

/* ADPCM Use Declaration (required when using ADPCM) */
PCM_DeclareUseAdpcm();

/* Set Interrupt Process */

Use INT_??? and set V-Blank IN interrupt.
Call PCM_VblIn(); within V-Blank IN interrupt.

/* File initialization */
GFS_Init(..);
STM_Init(..);
STM_OpenGrp();
STM_SetExecGrp(..);
/* Open Stream */
stm = STM_OpenFid(..);

/* Create Handle */
PCM_PARA_WORK(&para) = &pcm_work;
PCM_PARA_RING_ADDR(&para) = ring_buf;
PCM_PARA_RING_SIZE(&para) = RING_BUF_SIZE;
PCM_PARA_PCM_ADDR(&para) = PCM_ADDR;
PCM_PARA_PCM_SIZE(&para) = PCM_SIZE;
pcm = PCM_CreateStmHandle(&para, stm);

/* Start Playback */
PCM_Start(pcm);
while(TRUE) {
    /* Playback task processing */
    PCM_Task(pcm);

    /* End condition */
    if (PCM_GetPlayStatus(pcm) == PCM_STAT_PLAY_END) break;
}
/* Destroy Handle */
PCM_DestroyStmHandle(pcm);

/* Close stream */
STM_Close(stm);

/* End Processing */
PCM_Finish();
```

---

## 4.0 Programming Precautions

This section lists the precautions that must be taken when implementing the PCM-ADPCM Playback Library in your application.

### 1. Application Development Issues

- This library uses the CPU's DMA and timer functions. The CPU's DMA and timer functions may not be used when the library is in operation.
- The frequency with which `PCM_Task` is called should be more than the V-Blank interrupt frequency (1 time/16 ms). Calling `PCM_Task` at every V-Blank interrupt may not be adequate when the same sound is played back twice.

### 2. Relationship with Other Libraries

- This library uses the File System, Stream System, DMA, and ADPCM Decompression Libraries. Be sure to link these libraries.

### 3. Creating a Buffer

- Reserve the following memory areas:

Work buffer:	Size of the work structure
Ring buffer:	Minimum (sector size) * 10 [byte]. Integer multiple of the sector size. 1 sector = 2324 bytes during CD-ROM XA audio playback. In all other cases, 1 sector = 2048 bytes.
PCM buffer:	4096*2 ~ 4096*4 [sample/1 ch].
Pause processing work:	Minimum 4096 [sample]. Optimal when it is the same size as the PCM buffer.

### 4. Seamless Branching

- The PCM volume and pan settings are the same as the prior handle when seamless branching is performed by `PCM_EntryNext`.
- Up to 60 minutes of continuous playback is possible when seamless branching is performed with `PCM_EntryNext`. Continuous playback longer than this cannot be done.
- Use file pre-read processing in the application program when seamless branching is performed.
- The seamless branching function cannot be used when the File System is used.
- The seamless branching function can be performed only during single playback, and not during multiple system playback.

### 5. Multiple System Playback

- Use different PCM stream playback numbers when using this feature.
- When playing more than one CD file at the same time, be sure to use file interleaving or channel interleaving. Use the Stream System in this case.
- Multiple System Playback cannot be done with the same file. Different files must be specified.
- Multiple System Playback is able to play up to four audio channels at the same time.
- Only one channel of ADPCM compressed audio data can be played.

---

## 6. ADPCM Support

- Link the ADPCM Decompression Library `SEGA_ADP.LIB`.
- Call the ADPCM usage declaration `PCM_DeclareUseAdpcm` after the initialization function.

### User-SpecifiedADPCM Format Playback

- It is possible to play back audio with the same process (same program) as for the AIFF format. However, be aware that the CPU overhead increases compared with the AIFF format.

### CD-ROM XA Audio Format Playback

- The library determines the playback frequency and the number of channels (stereo/mono) from the coding information contained in the subheader area. Confirm that the information is entered correctly.
- The library user sets the stream key.
- Set the ring buffer to an integer multiple of the sector size 2324 bytes.
- Set playback information using `PCM_SetInfo`.
- Use only the Stream Playback mode for CD-ROM XA audio.
- See the sample program `smp_pcm5.c`.

## 7. Using the Functions

- Be sure to call `PCM_Init` at the start of the program.
- Be sure to call `PCM_VblIn` within the V-Blank IN interrupt.
- `PCM_Start` can be executed only once for the handle.  
If the same file is played more than once, create a handle for each time it is played.
- Call `PCM_SetLoadNum`, `PCM_SetPcmCmdBlockNo`, `PCM_SetPcmStreamNo` before beginning playback.
- For handles created using `PCM_CreateMemHandle`, be sure to communicate the data size to the ring buffer using `PCM_NotifyWriteSize`.
- When using the pause function, pause processing work must first be specified by `PCM_SetPauseWork`. This work need not be prepared for every handle.  
Because this work is a work area that is used temporarily during pause-on processing, the area can be reserved immediately prior to the pause-on processing and can be released immediately thereafter.
- CPU DMA is used as the default method to transfer data from the CD block to the ring buffer.  
Use `PCM_SetTrModeCd` to specify program transfer, CPU DMA, or SCU DMA.  
Call `PCM_SetTrModeCd` after creating a handle and before calling the task function for the first time.
- Be sure to set playback information with `PCM_SetInfo` when playing back CD-ROM XA audio.

---

## 5.0 Data Specifications

### 5.1 List of Data

Table 5.1 lists the data used by this library.

**Table 5.1 List of Data**

Function	Function Name	No.
Basic Data		1.0
Constants		2.0
Error code	PCM_ERR_~	2.1
Playback status	PCM_STAT_PLAY_~	2.2
Pause control command	PCM_PAUSE_~	2.3
Forced switch enabled check value	PCM_CHANGE_~	2.4
Data transfer mode select value	PCM_TRMODE_~	2.5
Data Type		3.0
Handle	PcmHn	3.1
Create parameters	PcmCreatePara	3.2
Error registration function	PcmErrFunc	3.3
PCM information	PcmInfo	3.4

---

## 5.2 Data Details

- **Basic Data**

Title	Data	Data Name	No.
Data Specifications	Basic data		1.0

Type	Description
Uint8	Unsigned 1 byte integer
Sint8	Signed 1 byte integer
Uint16	Unsigned 2 byte integer
Sint16	Signed 2 byte integer
Uint32	Unsigned 4 byte integer
Sint32	Signed 4 byte integer
Bool	Boolean type. The following values are taken: FALSE TRUE

---

- **Constants**

Title	Data	Data Name	No.
Data Specifications	Error Codes	PCM_ERR_~	2.1

The following constants represent error codes.

Constant Name	Description
PCM_ERR_OK	No error has occurred
PCM_ERR_OUT_OF_HANDLE	Out of handles
PCM_ERR_NO_INIT	Initialization function was not called
PCM_ERR_INVALID_HN	Destroyed by invalid handle
PCM_ERR_ILL_CREATE_MODE	Differs than Create mode
PCM_ERR_TOO_LARGE_HEADER	Header is too large (buffer size is too small)
PCM_ERR_HEADER_DATA	Header data error
PCM_ERR_AFI_NO_COMMON	No CommonChunk
PCM_ERR_AFI_COMPRESS	Unsupported compression type
PCM_ERR_NOT_DECLARE_ADPCM	ADPCM usage declaration not made
PCM_ERR_ILLEGAL PARA	Mistake in argument specification
PCM_ERR_ILLEGAL_HANDLE	Illegal handle
PCM_ERR_NEXT_HN_STATUS	Error status of continuous play handle
PCM_ERR_NEXT_HN_AUDIO	Audio conditions do not match
PCM_ERR_CHANGE_NO_ENTRY	Changed without entries
PCM_ERR_PAUSE_STATUS	Called by conditions other than PCM_STAT_PLAY_TIME or PCM_STAT_PLAY_PAUSE
PCM_ERR_PAUSE_WORK_NULL	Pause processing work error
PCM_ERR_PAUSE_WORK_SIZE	Pause processing work error
PCM_ERR_PAUSE_WORK_SET	Illegal pause processing work specification
PCM_ERR_DMA_MODE	Unsupported transfer mode
PCM_ERR_DMA_CPU_PCM	Abnormal end of DMA
PCM_ERR_GFS_READ	GFS read failure
PCM_ERR_RING_SUPPLY	Supplied data to the ring buffer after playback was finished (data was not supplied in time and processing was stopped)

Title	Data	Data Name	No.
Data Specifications	Playback status	PCM_STAT_PLAY_~	2.2

The following table shows the playback status data.

Constant Name	Description
PCM_STAT_PLAY_ERR_STOP	Abnormal stop
PCM_STAT_PLAY_CREATE	Creation status
PCM_STAT_PLAY_PAUSE	Pause
PCM_STAT_PLAY_START	Start playback
PCM_STAT_PLAY_HEADER	Header processing status
PCM_STAT_PLAY_TIME	Playing (timer start)
PCM_STAT_PLAY_END	End playback

Title	Data	Data Name	No.
Data Specifications	Pause control command	PCM_PAUSE_~	2.3

The following tables shows pause control data.

Constant Name	Description
PCM_PAUSE_ON_AT_ONCE	Immediate pause
PCM_PAUSE_OFF	Cancel pause

Title	Data	Data Name	No.
Data Specifications	Forced switch enabled check value	PCM_CHANGE_~	2.4

The following table shows forced switching enabled status data.

Constant Name	Description
PCM_CHANGE_OK_AT_ONCE	Forced switching can be done immediately
PCM_CHANGE_NO_DATA	Insufficient data. Cannot do continuous playback smoothly if forced switching (PCM_Change) is executed.
PCM_CHANGE_NO_ENTRY	The next handle is not registered by PCM_EntryNext. Forced switching (PCM_Change) cannot be executed.

---

Title	Data	Data Name	No.
Data Specifications	Data transfer mode select value	PCM_TRMODE_~	2.5

The following table shows data transfer methods.

Constant Name	Description
PCM_TRMODE_CPU	Software transfer
PCM_TRMODE_SDMA	CPU direct memory access (cycle steal)
PCM_TRMODE_SCU	SCU direct memory access

---

- **Data Type**

Title	Data	Data Name	No.
Data Specifications	Handle	PcmHn	3.1

The following represents information on each playback unit.

```
typedef void *PcmHn;
```

Title	Data	Data Name	No.
Data Specifications	Create parameters	PcmCreatePara	3.2

The following represents handle creation information. All types of parameter values are set to `PCM_Create~` as structures.

```
typedef struct {
    PcmWork *work;          /* Start address of work           */
                           /* Work area when this library plays audio */
    Sint32  *ring_addr;    /* Ring buffer address            */
                           /* This is the address when playing data
                           /* from memory                   */
    Sint32  ring_size;     /* Ring buffer size [byte]        */
                           /* File size when playing data from memory */
    Sint32  *pcm_addr;    /* PCM buffer address of sound memory */
                           /* PCM buffer size [sample/1 ch] of sound
                           /* memory                         */
    Sint32  pcm_size;     /* PCM buffer size [sample/1 ch] of sound
                           /* memory                         */
} PcmCreatePara;
```

#### Cautions Concerning the PCM Buffer

- Set up the PCM buffer in sound memory.
- Specify the integer multiple of 4096 for the PCM buffer size (minimum 4096 X 2).
- If PCM buffer size is considered to be S [sample/1 ch], then—as required by playback conditions—it is expressed in bytes as shown in the following table:

Playback Conditions	PCM Buffer Size (bytes)
8 bit monaural	S [byte]
8 bit stereo	2 X S [byte]
16 bit monaural	2 X S [byte]
16 bit stereo	4 X S [byte]

- Specify the same area (address, size) when doing smooth continuous playback.
- Specify a different area when doing multiple stream playback.

---

Title	Data	Data Name	No.
Data Specifications	Error registration function	PcmErrFunc	3.3

This function is called when an error occurs.

**Format:** void (\*PcmErrFunc)(void \*obj, Sint32 err\_code)  
**Input:** obj : Registered object  
          err\_code : Error code  
**Output:** None

Title	Data	Data Name	No.
Data Specifications	Playback information	PcmInfo	3.4

The following represents PCM playback information.

```
typedef struct {
    PcmFileType file_type;      /* File type */          */
    PcmDataType data_type;      /* Data type */          */
    Sint32     file_size;       /* File size [byte] */   */
                           /* Permits the supply of more data than this to */
                           /* to the ring buffer, but does not process it */ */
    Sint32     channel;        /* Number of channels */ */
    Sint32     sampling_bit;    /* Sampling bit resolution */ */
    Sint32     sampling_rate;   /* Sampling rate [Hz] */ */
    Sint32     sample_file;    /* Number of samples in the file [sample/1ch] */ */
    Sint32     compression_type; /* Compression type */ */
} PcmInfo;
```

## 6.0 Functions Specifications

### 6.1 List of Functions

Tables 6.1 and 6.2 list the functions of this library.

**Table 6.1 List of Functions**

Function	Function Name	No.
Initialization and End Processes		1.0
Initialize Library	(1) PCM_Init	1.1
Library end process	(1) PCM_Finish	1.2
ADPCM use declaration	(2) PCM_DeclareUseAdpcm	1.3
Handle Operations		2.0
Create handle (memory)	(3) PCM_CreateMemHandle	2.1
Destroy handle (memory)	(3) PCM_DestroyMemHandle	2.2
Create handle (file system)	(4) PCM_CreateGfsHandle	2.3
Destroy handle (file system)	(4) PCM_DestroyGfsHandle	2.4
Create handle (stream system)	(5) PCM_CreateStmHandle	2.5
Destroy handle (stream system)	(5) PCM_DestroyStmHandle	2.6
Playback task	(1) PCM_Task	2.7
) V-Blank IN processing function	(1) PCM_VblIn	2.8
Playback Controls		3.0
Start playback	(6) PCM_Start	3.1
Stop playback	PCM_Stop	3.2
Pause	(7) PCM_Pause	3.3
Register next play handle	(6) PCM_EntryNext	3.4
Forced playback handle change	PCM_Change	3.5
Get handle change status	PCM_CheckChange	3.6

#### Notes

- (1) Required
- (2) Required when using ADPCM
- (3) Required for Memory Playback Mode
- (4) Required for File Playback Mode
- (5) Required for Stream Playback Mode
- (6) One or the other is required
- (7) Required when using the pause function

**Table 6.2 List of Functions (2)**

Function	Function Name	No.
Information Setting Functions		4.0
Set maximum number of transfer sectors	PCM_SetLoadNum	4.1
Set playback pan position	PCM_SetPan	4.2
Set playback volume	PCM_SetVolume	4.3
Change PCM playback parameter	PCM_ChangePcmPara	4.4
Set PCM command block number	PCM_SetPcmCmdBlockNo	4.5
Set PCM stream playback number	(4) PCM_SetPcmStreamNo	4.6
Set pause processing work	(5) PCM_SetPauseWork	4.7
Set data transfer mode (CD block→ Ring buffer)	PCM_SetTrModeCd	4.8
Set playback information	(6) PCM_SetInfo	4.9
Get Information Functions		5.0
Get playback time	PCM_GetTime	5.1
Get playback status	(1) PCM_GetPlayStatus	5.2
Buffer Controls		6.0
Get write buffer	(3) PCM_GetWriteBuf	6.1
Notify write size	(2) PCM_NotifyWriteSize	6.2
Error Controls		7.0
Register error function	PCM_SetErrFunc	7.1
Get error information	PCM_GetErr	7.2

**Notes:**

- (1) Required
- (2) Required for Memory Playback Mode
- (3) Required when supplying data to the ring buffer in the Memory Playback Mode
- (4) Required for Multiple Stream Playback Mode
- (5) Required when using the pause function
- (6) Required for CD-ROM XA audio playback

## 6.2 Function Details

- **Initialization and End Processes**

Title	Function	Function Name	No.
Function Specifications	Initialize Library	PCM_Init	1.1

**Format:** Bool PCM\_Init(void)  
**Input:** None  
**Output:** None  
**Value:** When able to initialize normally TRUE  
When unable to initialize normally FALSE  
**Function:** To enable use of this library, perform this initialization immediately after the program starts.  
**Remarks:** Be sure to call this function at the start of the program when using the library.

Title	Function	Function Name	No.
Function Specifications	Library end process	PCM_Finish	1.2

**Format:** void PCM\_Finish(void)  
**Input:** None  
**Output:** None  
**Value:** None  
**Function:** Performs the PCM library end process.  
**Remarks:** Call this function when no longer performing PCM playback.

Title	Function	Function Name	No.
Function Specifications	ADPCM use declaration	PCM_DeclareUseAdpcm	1.3

**Format:** void PCM\_DeclareUseAdpcm(void)  
**Input:** None  
**Output:** None  
**Value:** None  
**Function:** This function enables ADPCM playback.  
**Remarks:** This function is required when the user-specified ADPCM or CD-ROM XA Audio format data is played back.

Call this function immediately after the initialization function `PCM_Init`. Calling this function links the ADPCM Decompression Library module. Calling this function does not restrict playback functions for data formats other than ADPCM. If this function is not called, the execution file size becomes somewhat smaller.

---

- **Handle Operations**

Title	Function	Function Name	No.
Function Specifications	Create handle (Memory)	PCM_CreateMemHandle	2.1

**Format:** PcmHn PCM\_CreateMemHandle(PcmCreatePara \*para)  
**Input:** para : Creation parameters  
**Output:** None  
**Value:** Handle (NULL when unable to generate handle)  
**Function:** Creates handles for playing files that exist in memory.  
**Remarks:**

- 32 handles can be created at the same time.
- Load files with the application program into memory.
- For handles produced by PCM\_CreateMemHandle, be sure to communicate the file size using PCM\_NotifyWriteSize.

Title	Function	Function Name	No.
Function Specifications	Destroy handle (Memory)	PCM_DestroyMemHandle	2.2

**Format:** void PCM\_DestroyMemHandle(PcmHn pcm)  
**Input:** pcm : handle  
**Output:** None  
**Value:** None  
**Function:** Destroys the handle.  
**Remarks:** Once a handle is destroyed, it can no longer be used.

Title	Function	Function Name	No.
Function Specifications	Create handle (File System)	PCM_CreateGfsHandle	2.3

**Format:** PcmHn PCM\_CreateGfsHandle(PcmCreatePara \*para, GfsHn gfs)  
**Input:** para : Creation parameters  
gfs : File handle  
**Output:** None  
**Value:** Handle (NULL when unable to generate handle)  
**Function:** Creates a handle for playing files with the File System.  
**Remarks:**

- 32 handles can be created at the same time.
- Get the file handle in advance with the application.

Title	Function	Function Name	No.
Function Specifications	Destroy handle (File System)	PCM_DestroyGfsHandle	2.4

**Format:** void PCM\_DestroyGfsHandle(PcmHn pcm)  
**Input:** pcm : handle  
**Output:** None  
**Value:** None  
**Function:** Destroys the handle.  
**Remarks:** Once a handle is destroyed, it can no longer be used.

Title	Function	Function Name	No.
Function Specifications	Create handle (Stream System)	PCM_CreateStmHandle	2.5

**Format:** PcmHn PCM\_CreateStmHandle(PcmCreatePara \*para, StmHn stm)  
**Input:** para : Creation parameters  
stm : Stream handle  
**Output:** None  
**Value:** Handle (NULL when unable to generate handle)  
**Function:** Creates handles for playing files with the Stream System.  
**Remarks:**

- 32 handles can be created at the same time.
- Get the stream handle in advance with the application.

Title	Function	Function Name	No.
Function Specifications	Destroy handle (Stream System)	PCM_DestroyStmHandle	2.6

**Format:** void PCM\_DestroyStmHandle(PcmHn pcm)  
**Input:** pcm : Handle  
**Output:** None  
**Value:** None  
**Function:** Destroys handles  
**Remarks:** Once a handle is destroyed, it can no longer be used.

Title	Function	Function Name	No.
Function Specifications	Playback task	PCM_Task	2.7

**Format:** void PCM\_Task(PcmHn pcm)  
**Input:** pcm : Handle  
**Output:** None  
**Value:** None  
**Function:** Reads files from the CD and transfers data to sound memory.  
**Remarks:** This function must be called periodically during playback. This function should be called with a frequency greater than the V-Blank interruptfrequency (1 time/16 ms).

Title	Function	Function Name	No.
Function Specifications	V-Blank IN processing function	PCM_VblIn	2.8

**Format:** void PCM\_VblIn(void)  
**Input:** None  
**Output:** None  
**Value:** None  
**Function:** Manages the playback time.  
**Remarks:** Be sure to call this function with a V-Blank IN interrupt process when using this library.

---

- **Playback Controls**

Title	Function	Function Name	No.
Function Specifications	Start playback	PCM_Start	3.1

**Format:** void PCM\_Start(PcmHn pcm)  
**Input:** pcm : Handle  
**Output:** None  
**Value:** None  
**Function:** Starts playback.  
**Remarks:** Playback is possible only once per generated handle.  
 To play the same file repeatedly, a new handle must be generated each time.

Title	Function	Function Name	No.
Function Specifications	Stop playback	PCM_Stop	3.2

**Format:** void PCM\_Stop(PcmHn pcm)  
**Input:** pcm : Handle  
**Output:** None  
**Value:** None  
**Function:** Stops playback.

Title	Function	Function Name	No.
Function Specifications	Pause	PCM_Pause	3.3

**Format:** void PCM\_Pause(PcmHn pcm, PcmPauseCmd cmd)  
**Input:** pcm : Handle  
 cmd : Pause control command  
**Output:** None  
**Value:** None  
**Function:** Pauses playback. Also cancels pause.

Title	Function	Function Name	No.
Function Specifications	Register next play handle	PCM_EntryNext	3.4

**Format:** void PCM\_EntryNext( PcmHn pcm )  
**Input:** pcm : Handle  
**Output:** None  
**Value:** None  
**Function:** Registers the next playback handle.  
**Remarks:** When a handle is registered by this function, the next handle automatically begins playing when the handle currently playing completes its playback. The entry is canceled if NULL is specified.  
A handle registered by this function begins playback even if there is a forced change by PCM\_Change.  
Only one handle can be registered. It becomes unregistered when the switch occurs to the registered handle. In addition, the next handle can be registered.

Title	Function	Function Name	No.
Function Specifications	Forced playback handle change	PCM_Change	3.5

**Format:** void PCM\_Change(void)  
**Input:** None  
**Output:** None  
**Value:** None  
**Function:** Stops the handle currently being played and starts playback of the handle registered by PCM\_EntryNext.

Title	Function	Function Name	No.
Function Specifications	Get handle change status	PCM_CheckChange	3.6

**Format:** PcmChangeStatus PCM\_CheckChange(void)  
**Input:** None  
**Output:** None  
**Value:** Forced change enabled check value  
**Function:** Enables the application to check whether it can force a change with PCM\_Change.

- **Information Setting Functions**

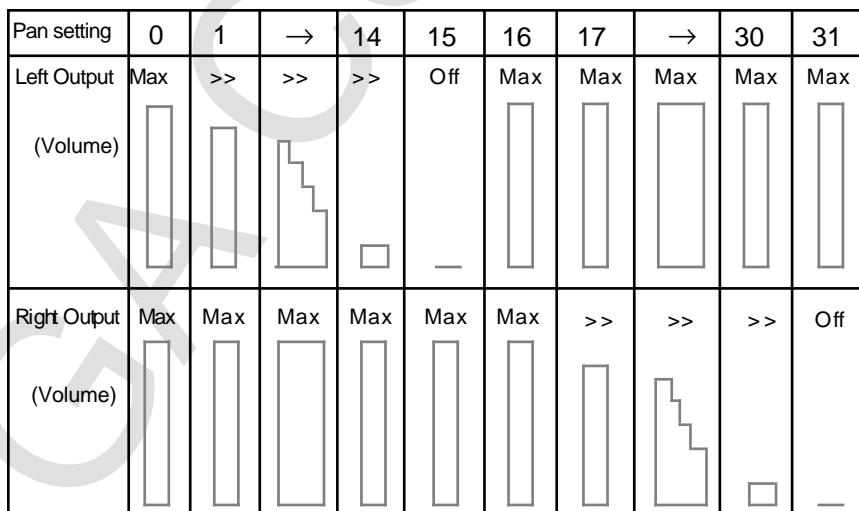
Title	Function	Function Name	No.
Function Specifications	Set maximum number of transfer sectors	PCM_SetLoadNum	4.1

**Format:** void PCM\_SetLoadNum(PcmHn pcm, Sint32 load\_sct)  
**Input:** pcm : Handle  
**Output:** None  
**Value:** None  
**Function:** Sets the maximum number of sectors transferred from the CD buffer to the ring buffer of the library. Default is 20 sectors.  
**Remarks:** Call this function before starting playback.

Title	Function	Function Name	No.
Function Specifications	Set playback pan position	PCM_SetPan	4.2

**Format:** void PCM\_SetPan(PcmHn pcm, Sint32 PAN)  
**Input:** pcm : Handle  
**Output:** None  
**Value:** None  
**Function:** Specifies the audio pan position. Default is 0. Pan settings are valid for mono playback, but are ignored when playing in stereo.

The following figures shows pan settings.



Title	Function	Function Name	No.
Function Specifications	Set playback volume	PCM_SetVolume	4.3

**Format:** void PCM\_SetVolume(PcmHn pcm, Sint32 volume)

**Input:** pcm : Handle

volume : Volume (0 ~ 7)

**Output:** None

**Value:** None

**Function:** Specifies the volume (no output at 0, maximum volume at 7). Default is 7.

Title	Function	Function Name	No.
Function Specifications	Change PCM playback parameter	PCM_ChangePcmPara	4.4

**Format:** void PCM\_ChangePcmPara(PcmHn pcm)

**Input:** pcm : Handle

**Output:** None

**Value:** None

**Function:** Changes the PCM playback parameter for the sound driver during playback.

**Remarks:** The actual parameter change processing is performed by calling this function after the pan and volume setting are made.

Title	Function	Function Name	No.
Function Specifications	Set PCM command block number	PCM_SetPcmCmdBlockNo	4.5

**Format:** void PCM\_SetPcmCmdBlockNo(PcmHn pcm, Sint32 blk\_no)

**Input:** pcm : Handle

blk\_no : PCM command block number (0 ~ 7)

**Output:** None

**Value:** None

**Function:** Sets the PCM command block number that is set in the sound driver. Default is 1.

**Remarks:** See *SATURN Sound Driver System Interface* manual for details on the PCM command block number. This function must be called before starting playback.

Title	Function	Function Name	No.
Function Specifications	Set PCM stream playback number	PCM_SetPcmStreamNo	4.6

**Format:** void PCM\_SetPcmStreamNo( PcmHn pcm, Sint32 stream\_no )  
**Input:** pcm : Handle  
stream\_no : PCM stream playback number (0 ~ 7)  
**Output:** None  
**Value:** None  
**Function:** Sets the PCM stream playback number to set in the sound driver. Default is 1.  
**Remarks:** See the *SATURN Sound Driver System Interface* manual for details on PCM stream playback numbers. Different stream playback numbers must be set to each stream for multiple stream playback. This function must be called before beginning playback.

Title	Function	Function Name	No.
Function Specifications	Set pause processing work	PCM_SetPauseWork	4.7

**Format:** void PCM\_SetPauseWork( Sint32 \*addr, Sint32 size )  
**Input:** addr : Work address  
size : Work size  
**Output:** None  
**Value:** None  
**Function:** Sets the work area used by the pause-on processing.  
**Remarks:** When using the pause function, a pause processing work area must be specified by PCM\_SetPauseWork in advance. It is not necessary to set the work area up for each handle. Because this work area is used temporarily for the pause-on processing, the area must be reserved immediately before the pause-on processing, and can be released immediately thereafter.

**Example:**

```
#define PAUSE_WORK_SIZE ( 4096L*2 )

pause_work_addr = malloc(PAUSE_WORK_SIZE);
PCM_SetPauseWork(pause_work_addr, PAUSE_WORK_SIZE);
PCM_Pause(pcm, PCM_PAUSE_ON_AT_ONCE);
free(pause_work_addr);
```

Title	Function	Function Name	No.
Function Specifications	Set data transfer mode (CD block → ring buffer)	PCM_SetTrModeCd	4.8

**Format:** void PCM\_SetTrModeCd(PcmHn pcm, PcmTrMode mode)  
**Input:** pcm : Handle  
**Output:** mode : Data transfer method  
**Value:** None  
**Function:** Sets the data transfer mode for transfers from the CD block to the ring buffer.  
Default is CPU DMA. It is possible to specify software-based transfer.  
**Remarks:** After the handle is created, PCM\_SetTrModeCd must be called before calling the task function.

Title	Function	Function Name	No.
Function Specifications	Set playback information	PCM_SetInfo	4.9

**Format:** void PCM\_SetInfo(PcmHn pcm, PcmInfo \*info)  
**Input:** pcm : Handle  
info : Play information  
**Output:** None  
**Value:** None  
**Function:** Sets information for PCM playback.  
**Remarks:** Because it gets information for playback from the file header, this function is normally not used. Currently, it is required only for playing CD-ROM XA audio.  
After the handle is created, PCM\_SetInfo must be called before the task function is called for the first time.  
**Example:** Playback information should be set as shown in the following example for CD-ROM XA audio playback.

```
PcmInfo info;

PCM_INFO_FILE_TYPE (&info) = PCM_FILE_TYPE_NO_HEADER; /* without a header */
PCM_INFO_DATA_TYPE (&info) = PCM_DATA_TYPE_ADPCM_SCT; /* XA audio sector processing */
PCM_SetInfo(pcm, &info);
```

---

- **Get Information Functions**

Title	Function	Function Name	No.
Function Specifications	Get play time	PCM_GetTime	5.1

**Format:** Sint32 PCM\_GetTime(PcmHn pcm)  
**Input:** pcm : Handle  
**Output:** None  
**Value:** Current time  
**Function:** Gets the current time (number of samples played from the start of this file until now.)

Title	Function	Function Name	No.
Function Specifications	Get playback status	PCM_GetPlayStatus	5.2

**Format:** PcmPlayStatus PCM\_GetPlayStatus(PcmHn pcm)  
**Input:** pcm : Handle  
**Output:** None  
**Value:** Playback status  
**Function:** Returns the current playback status.

- **Buffer Controls**

Title	Function	Function Name	No.
Function Specifications	Get write buffer	PCM_GetWriteBuf	6.1

**Format:** Uint32 \*PCM\_GetWriteBuf(PcmHn pcm, Sint32 \*free\_size, Sint32 \*free\_total)  
**Input:** pcm : Handle  
**Output:** free\_size : Number of bytes of continuous writeable area  
free\_total : Total number of bytes of writeable area including noncontiguous areas.  
**Value:** Start address of contiguous writeable area (NULL when not writeable)  
**Function:** Gets the write destination buffer address and writeable number of bytes.  
**Remarks:** This function is used by the user to provide data to the ring buffer.

Title	Function	Function Name	No.
Function Specifications	Notify write size	PCM_NotifyWriteSize	6.2

**Format:** void PCM\_NotifyWriteSize(PcmHn pcm, Sint32 write\_size)  
**Input:** pcm : Handle  
write\_size : Number of bytes written  
**Output:** None  
**Value:** None  
**Function:** Notifies the library of the number of data bytes actually output to the ring buffer.  
This function is used for implementing user-controlled ring buffer data transfers.  
**Remarks:** Reports the buffer size (file size) when playing files in memory.

---

- **Error Controls**

Title	Function	Function Name	No.
Function Specifications	Register error function	PCM_SetErrFunc	7.1

**Format:** void PCM\_SetErrFunc(PcmErrFunc func, void \*obj)  
**Input:** func : Function called when an error occurs  
**Output:** obj : Registered object  
**Function Value:** None  
**Function:** Sets the function called when an error occurs. The registered object is passed to the first argument of the registered function.

Title	Function	Function Name	No.
Function Specifications	Get error information	PCM_GetErr	7.2

**Format:** PcmErrCode PCM\_GetErr(void)  
**Input:** None  
**Output:** None  
**Value:** Error code  
**Function:** Returns the error code that occurred most recently.

---

# INDEX

## C

CMP\_DecRunlen ..... 98  
CMP\_DecRunlenByte ..... 99  
CMP\_DecRunlenDword ..... 101  
CMP\_DecRunlenWord ..... 100  
CSH\_AllClr ..... 51  
CSH\_GetCcr ..... 52  
CSH\_Init ..... 51  
CSH\_Purge ..... 52  
CSH\_SetAcsWay ..... 54  
CSH\_SetCcr ..... 53  
CSH\_SetCodeFill ..... 53  
CSH\_SetDataFill ..... 54  
CSH\_SetEnable ..... 53  
CSH\_SetWayMode ..... 54

## D

DBG\_ClearScreen ..... 83  
DBG\_DisplayOff ..... 83  
DBG\_DisplayOn ..... 83  
DBG.GetKeyStr ..... 84  
DBG\_Initial ..... 82  
DBG\_Monitor ..... 85  
DBG\_Printf ..... 84  
DBG\_SetCursor ..... 83  
DMA\_CpuAllStop ..... 49  
DMA\_CpuGetComStatus ..... 49  
DMA\_CpuGetStatus ..... 49  
DMA\_CpuMemCopy 1 ..... 44  
DMA\_CpuMemCopy 16 ..... 45  
DMA\_CpuMemCopy 2 ..... 44  
DMA\_CpuMemCopy 4 ..... 45  
DMA\_CpuResult ..... 46  
DMA\_CpuSetComPrm ..... 48  
DMA\_CpuSetPrm ..... 48  
DMA\_CpuStart ..... 48  
DMA\_CpuStop ..... 49  
DMA\_ScuAllStop ..... 47

DMA\_ScuMemCopy ..... 42  
DMA\_ScuResult ..... 43  
DMA\_ScuSetPrm ..... 47  
DMA\_ScuStart ..... 47  
DMA\_ScuStop ..... 47  
DmaCpuComPrm ..... 38  
DmaCpuPrm ..... 39  
DmaScuPrm ..... 35

## I

INT\_ChgMsk ..... 62  
INT\_GetAck ..... 64  
INT\_GetFunc ..... 65  
INT\_GetMsk ..... 62  
INT\_GetScuFunc ..... 65  
INT\_GetStat ..... 64  
INT\_ResStat ..... 64  
INT\_SetAck ..... 64  
INT\_SetFunc ..... 65  
INT\_SetMsk ..... 63  
INT\_SetScuFunc ..... 66

## M

MEM\_Calloc ..... 67  
MEM\_Free ..... 68  
MEM\_Init ..... 67  
MEM\_Malloc ..... 68  
MEM\_Realloc ..... 68

## P

PCM\_Change ..... 124  
PCM\_CHANGE ~ ..... 114  
PCM\_ChangePcmPara ..... 126

---

PCM_CheckChange .....	124
PCM_CreateGfsHandle .....	121
PCM_CreateMemHandle .....	121
PCM_CreateStmHandle .....	122
PCM_DeclareUseAdpcm .....	120
PCM_DestroyGfsHandle .....	121
PCM_DestroyMemHandle .....	121
PCM_DestroyStmHandle .....	122
PCM_EntryNext .....	124
PCM_ERR_~ .....	113
PCM_Finish .....	120
PCM_GetErr .....	130
PCM_GetPlayStatus .....	129
PCM_GetTime .....	129
PCM_GetWriteBuf .....	129
PCM_Init .....	120
PCM_NotifyWriteSize .....	129
PCM_Pause .....	123
PCM_PAUSE_~ .....	114
PCM_SetErrFunc .....	130
PCM_SetInfo .....	128
PCM_SetLoadNum .....	125
PCM_SetPan .....	125
PCM_SetPauseWork .....	127
PCM_SetPcmCmdBlockNo .....	126
PCM_SetPcmStreamNo .....	127
PCM_SetTrModeCd .....	128
PCM_SetVolume .....	126
PCM_Start .....	123
PCM_STAT_PLAY_~ .....	114
PCM_Stop .....	123
PCM_Task .....	122
PCM_TRMODE_~ .....	115
PCM_VblIn .....	122
PcmCreatePara .....	116
PcmErrFunc .....	117
PcmHn .....	116
PcmInfo .....	117

## S

SND_ChgEfct .....	24
SND_ChgMap .....	24
SND_ChgMix .....	24
SND_ChgMixPrm .....	25
SND_ChgPcm .....	28
SND_ChgTempo .....	27
SND_ChkHard .....	25
SND_ContSeq .....	26
SND_CtrlDirMidi .....	25
SND_GET_ENA_INT .....	22
SND_GET_INT_STAT .....	30
SND_GetAnlHzVl .....	31
SND_GetAnlTIVl .....	31
SND_GetPcmPlayAdr .....	30
SND_GetSeqPlayPos .....	30
SND_GetSeqStat .....	30
SND_Init .....	21
SND_MoveData .....	23
SND_PauseSeq .....	26
SND_RESET_INT .....	22
SND_SET_ENA_INT .....	21
SND_SET_FCT_INT .....	22
SND_SetCdDaLev .....	29
SND_SetCdDaPan .....	29
SND_SetSeqVl .....	27
SND_SetTIVl .....	24
SND_StartPcm .....	28
SND_StartSeq .....	26
SND_StartVIAnl .....	27
SND_StopPcm .....	28
SND_StopSeq .....	26
SND_StopVIAnl .....	29
SndAreaMap .....	10
SndCdHzSrVl .....	19
SndEfctBnkNum .....	11
SndEfctOut .....	12
SndFade .....	15
SndHardPrm .....	13
SndHardStat .....	13
SndIniDt .....	10
SndLev .....	12

---

SndMixBnkNum .....	11
SndPan .....	12
SndPcmChgPrm .....	18
SndPcmIntStat .....	19
SndPcmPlayAdr .....	18
SndPcmStartPrm .....	17
SndRet .....	13
SndSeqBnkNum .....	14
SndSeqNum .....	14
SndSeqPlayPos .....	16
SndSeqPri .....	14
SndSeqSongNum .....	14
SndSeqStat .....	16
SndSeqVi .....	15
SndTempo .....	15
SndTIVI .....	11
SndToneBnkNum .....	11
T	
TIM_Frt_Cnt_To_Mcr .....	76
TIM_Frt_Delay_16 .....	76
TIM_Frt_Get_16 .....	75
TIM_Frt_Get_Frc .....	79
TIM_Frt_Get_Icra .....	80
TIM_Frt_Get_Ocra .....	79
TIM_Frt_Get_Ocrb .....	79
TIM_Frt_Get_Tcr .....	79
TIM_Frt_Get_Tcsr .....	78
TIM_Frt_Get_Tier .....	78
TIM_Frt_Get_Tocr .....	80
TIM_Frt_Init .....	75
TIM_Frt_Mcr_To_Cnt .....	76
TIM_Frt_Set_16 .....	75
TIM_Frt_Set_Frc .....	77
TIM_Frt_Set_Ocra .....	77
TIM_Frt_Set_Ocrb .....	77
TIM_Frt_Set_Tcr .....	78
TIM_Frt_Set_Tcsr .....	77
TIM_Frt_Set_Tier .....	76
TIM_Frt_Set_Tocr .....	78
TIM_T0_Disable .....	73
TIM_T0_Enable .....	73