

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 non-disclosure and confidentiality agreement signed separately and /in the possession of SEGA. If you have not signed such a non-disclosure 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 describe 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 simply 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

SEGA Confidential

SATURN

System Library

User's Guide

ver. 1.0

Doc. # ST-162-R1-092994

© 1994-95 SEGA. All Rights Reserved.

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-162-R1-092994 Date _____

Document name Saturn System Library User's Guide, ver. 1.0

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

REFERENCES

In translating/creating this document, certain technical words and/or phrases were interpreted with the assistance of the technical literature listed below.

1. *KenKyusha New Japanese-English Dictionary*
1974 Edition
2. *Nelson's Japanese-English Character Dictionary*
2nd revised version
3. *Microsoft Computer Dictionary*
4. *Japanese-English Computer Terms Dictionary*
Nichigai Associates
4th version

Table of Contents

SYSTEM PROGRAM USER'S MANUAL	1
1.0 GUIDE	1
1.1 Explanation	1
2.0 REFERENCE	7
2.1 List of Functions	7
2.2 Function Specifications	8
SMPC I/F USER'S MANUAL	15
1.0 GUIDE	15
2.0 FEATURES	15
3.0 OVERVIEW	16
3.1 Function Overview	16
3.2 Process Overview	16
4.0 DETAILS	17
4.1 Functions	17
4.2 Process	18
4.3 Peripheral Control	22
5.0 CALLING SEQUENCE	23
6.0 DATA SPECIFICATIONS	24
6.1 List of Data	24
6.2 Data Specifications	25
7.0 FUNCTION SPECIFICATIONS	33
7.1 List of Functions	33
7.2 Function Specifications	33
BACKUP LIBRARY USER'S MANUAL	39
1.0 Guide	39
1.1 Purpose	39
1.2 Explanation	39
1.3 Program Example	40
2.0 Reference	41
2.1 Data List	41
2.2 Function List	42
2.3 Data Flow	43
2.4 Function Specification	43

System Program User's Manual

1.0 Guide

1.1 Explanation

Interrupt Process Routine Registration and Reference Operations

After booting up from the boot ROM, the master SH2 interrupt vector table is at the beginning of the work RAM, and the VBR (vector base register) indicates this address.

The slave SH2 interrupt vector table is the work RAM lead + 400H, and the slave SH2 VBR indicates that address. The interrupt vector (programmable) of every built-in SH2 module is assigned by the initial settings in the table below. Dummy routines that do nothing are set in the vector table. (With the exception of invalid commands and address errors, these are infinite loops.)

The FRT input capture interrupt is assigned for use in master and slave communications, and its initial priority is 15 (highest priority). In the table below, the priority of all interrupts, except for the FRT input capture interrupt, is set at 0 and interrupt is unauthorized.

In changing the priority of the built-in module interrupts, the content of the interrupt control register must change in response to the needs of the application.

Master SH2 Vector Initial Settings	Slave SH2 Vector Initial Settings
40H ~ SCU interrupt vector 5FH (set by hardware)	41H H-Blank In ** 43H V-Blank In
60H SCI receive error 61H SCI receive buffer full 62H SCI send buffer empty 63H SCI send quit * 64H FRT input capture 65H FRT compare match 66H FRT overflow 67H Free 68H WDT interval 69H BSC compare match 6AH Free 6BH Free 6CH DMACH1 (SH2 built-in) 6DH DMACH0 (SH2 built-in) 6EH DIVU (division) 6FH Free	60H SCI receive error 61H SCI receive buffer full 62H SCI send buffer empty 63H SCI send quit * 64H FRT input capture 65H FRT compare match 66H FRT overflow 67H free 68H WDT interval 69H BSC compare match 6AH Free 6BH Free 6CH DMACH1 (SH2 built-in) 6DH DMA CH0 (SH2 built-in) 6EH DIVU (division) 6FH Free

* for slave > master passing

* for master > slave passing

** IRL2, IRL6 level interrupts

This operation routine should be used to register the interrupt process routine to the interrupt vector and to reference the address of the current process routine. Furthermore, a SCU interrupt routine (the master SCU interrupt) that implements the interrupt process via a format that subroutine calls the C function is provided. The C function can be registered there and the registration address can also be referenced.

The function registered in the SCU interrupt routine is called whenever interrupt occurs. Before and after this call, register save and return are performed per register retention (save) protocol of the SHC compiler. If the routine is one that complies with the C function or that protocol, it can be registered and processed. If a separate interrupt process routine is registered in the SCU interrupt vector, the SCU interrupt process routine is bypassed and becomes invalid. However, it may not be suitable for an interrupt process requiring a rapid response such as HBlank.

SCU Interrupt Mask Set, Reference, and Change Operations

Because this register cannot be read, the mask value set to the SCU interrupt mask register cannot perform computation against the actually set values when changes, etc. are implemented. Therefore, this value is stored separately in memory and a service routine, which preserves and updates in consistency with the actual SCU interrupt mask register, is provided.

When this routine is used, setting and changing the SCU interrupt mask must always be done by the library and application through these functions. After the SCU interrupt mask register is set and changed, the SCU interrupt status register, and if necessary, the A-Bus interrupt acknowledge register, are cleared.

Simple Semaphore Operation

A service is available that enables memory (256 bytes) provided by the Boot ROM to be used as 256 bytes of simple semaphore. The first half (numbers 0 ~ 127) of semaphore can be used in any way. The second half (128 ~ 255) is used in operations related to the library. When the library uses a specific function such as DMA, it sets semaphore MSB(80H) to 1 and shows that it is in use. After that, it clears MSB and shows that it is free.

In a process that requires resources to be secured over a comparatively long period of time, the semaphore operation and reference procedure should be determined so that those resources are not accessed arbitrarily by an interrupt process during that period of time.

The SH2 TAS command is used when setting MSB for semaphore memory. This command allows only one process to reliably acquire semaphore since execution is indivisible (bus control [authorization] is not cleared). This must be cleared when the process that acquired semaphore is completed. All semaphore memory is cleared when reset.



System Clock Switching

System clock switching cannot be performed by issuing independent commands to the SMPC. Use of this system program is required. System clock switching entails a partial hardware reset.

CPU Clock	26 MHz	↔	28 MHz
Horizontal Resolution	320/640	↔	352/704

<u>Reset Devices</u>	<u>OFF or non guaranteed Devices</u>	<u>Unaffected Devices</u>
SCU	Slave SH (OFF)	Master SH * note
VDP1	DRAM (previous content destroyed)	SDRAM
VDP2	SCSP (OFF)	CD
SCSI/SCC (development devices only)		SIMM (development devices only)

* Note Because master SH is in the standby mode during clock switching, of the built-in SH modules, the FRT and the SCI within the SH must be reset. WDT is used during this process.

NMI goes to its existed status after the process; for example, the DMAC control goes to interrupt status by NMI. See the SH manual. If necessary, perform the reopen process.

Reinitializing process after reset:

SCU: Reinitializes the bus, interrupt mask, etc. However, the value of SYS_GETSCUIM is used for the interrupt mask value.

Postprocesses required with applications:

VDP2: The TV mode must be set comparatively fast. Because the 320/640 mode is used after the device itself is reset, especially when the system clock is changed to the 352/704 mode, the synchronous signal shifts in the TV and turbulence occurs on the screen.

VDP1, 2, SCSP: All previous settings are invalid. Must be reset.

SMPC: Hot reset must be enabled.

The clock change process time is about 110 ms, which includes the reset time of the device.

SCU Interrupt Routine Priority Change

The Boot ROM has an interrupt priority control table used for the SCU interrupt process service, making rewrite possible.

Note: This is a risky service. The system may hang up if priority relationship inconsistencies exist in the table contents.

With this, the interrupt process (items using SYS_SETUINT) can be optimized in the application.

To use this, prepare the same structural data for the application as the table, and call SYS_CHGUIPR.

Tables are of 32 long words. 1 long word has the following contents.

SH2 SR insignificant word value	SCU interrupt mask insignificant word value
---------------------------------	---

The value set to SR at the beginning of the interrupt process

ORed with the current mask set value and written to the SCU interrupt mask register at the beginning of the interrupt process.

This long word position inside the table corresponds to the SCU 30 interrupt factor. (V-Blank In is the start and V-Blank Out is the 2nd,...but 2 long word spaces that correspond to vectors 4EH and 4FH are included.)

Tables must be created very carefully so that there is no inconsistency between the SR, SCU interrupt mask, and interrupt factor.

For example, the Boot ROM uses the following table for its initial set values:

```

Uint32 PRITab[32] = {
    0x00F0FFFF, /* VBI SR=15 All prohibited (highest priority) */
    0x00E0FFFE, /* VBO SR=14 Only VBI is allowed */
    0x00D0FFFF, /* HBI SR=13 VBI and VBO are allowed */
    .
    .
    0x0070FE00, /* External 15 SR=7 SCU interrupt unique priority;
                all masked when 7 or less. */
                /* A bus interrupt unique priority that assumes 7, 4, and
                1 by cause factor, but because
                of the common use and 1 bit
                mask, it is set to 7. */
} ;

```



The creation example shown is one in which the SR value is always set to 0, and priorities are described only using SCU mask values (priority relationship). Here, SH always receives an interrupt and only the SCU mask controls authorization and prohibition.

```
Uint32 PRITab[32] = {
    0x0000FFF9,    /* during VBI process HBI and VBO authorization */
    0x0000FFFB,    /* during VBO process only HBI authorization */
    0x00D0FFFC,    /* during HBI process all prohibited (highest priority) */
    .
    .
    .
    0x00000000,    /* during external 15 process, all authorized (lowest priority) */
} ;
```

The inverse of the example above is prohibiting mask interrupt at SR value levels without changing the SCU mask value. (0 or 15 only are possible.)

In the example above, the interrupt authorize and prohibit register of each built-in module must also be operated for SH internal module interrupt authorization and prohibition.

Note: When the SCU factor interrupt is allowed and interrupt occurs, it is okay if the SH SR mask is higher than the interrupt unique level (value decided by SCU hardware) and if interrupt can never be refused by SH. (However, one exception is that all of SR mask 15 can be prohibited.)

CD Multiplayer Startup Execution

This is a service that activates and executes the CD multiplayer when an application ends. When this service is called, the CD multiplayer screen is displayed exactly the same as when the power-on sequence is activated. Regardless of the called status, the CD multiplayer screen is displayed and operation is enabled.

Power On Clear Memory Operation

This provides the 8 bytes of memory on the SDRAM controlled by the Boot ROM. The 8 bytes are initialized to 0 when power-on is activated, but the contents can be saved with the reset button (NMI).

(This page is blank in the original Japanese document.)

SEGA Confidential



2.0 Reference

2.1 List of Functions

Function	Function Name	No.
Interrupt Process Routine Registration / Reference		
Registers process routine to the interrupt vector	SYS_SETSINT	1.1
Registers function to SCU interrupt routine	SYS_SETUINT	1.2
Gets registration contents of interrupt vector	SYS_GETSINT	1.3
Gets registration contents of SCU interrupt routine	SYS_GETUINT	1.4
SCU Interrupt Mask Set, Reference, and Change Operations		
Sets SCU interrupt mask	SYS_SETSCUIM	2.1
Changes SCU interrupt mask	SYS_CHGSCUIM	2.2
References SCU interrupt mask	SYS_GETSCUIM	2.3
Simple Semaphore Operation		
Gets semaphore	SYS_TASSEM	3.1
Clears semaphore	SYS_CLRSEM	3.2
System Clock Switching		
Switches system clock	SYS_CHGSYSCK	4.1
References system clock value	SYS_GETSYSCK	4.2
SCU Interrupt Routine Priority Change		
Changes SCU interrupt routine priority	SYS_CHGUIPR	5.1
CD Multi-player Startup Execution		
Executes startup of CD multi-player	SYS_EXECDMP	6.1
Power On Clear Memory Operation		
Operates power-on clear memory	SYS_PCLRMEM	7.1

2.2 Function Specifications

Title	Function	Function Name	No.
Function Specifications	To the interrupt vector; registers interrupt process routine	SYS_SETSINT	1.1

Format: `void SYS_SETSINT (Uint 32 Num, void* Hdr);`

Input:
 Num : vector number (0..7FH)
 Hdr : interrupt process routine address (dummy routine when 0, or that interrupt process routine when Num is the SCU interrupt vector)

Output: None

Function Value: None

Function: Hdr must be a process routine that ends by register save, return, and RTE command (# pragma interrupt is added if using C language). When Hdr is 0, Num re-registers the SCU interrupt routine within the SCU interrupt (40H..4DH,50H..5FH), others register dummy routine. There is no range check. Values beyond restricted range must not be specified.

Remarks: This routine can be used with both master and slave SH2 and is registered to vector addresses based on each VBR.

Title	Function	Function Name	No.
Function Specifications	To SCU interrupt routine; registers process function	SYS_SETUINT	1.2

Format: `void SYS_SETUINT (Uint 32 Num, void* Hdr);`

Input:
 Num : vector number (SCU vector number)
 Hdr : function routine address (dummy routine when 0)

Output: None

Function Value: None

Function: Hdr must be a function by SHC. If the routine is created by the assembler, it must follow the SHC register save protocol. Num is limited to SCU interrupt vectors (40H..4DH,50H..5FH). A dummy routine is registered when Hdr is 0. There is no range check. Values beyond restricted range must not be specified.

Remarks: When a routine is registered to a vector by SYS_SETSINT, the SCU interrupt process routine of that vector becomes ineffective and the registration function is not called. Results are not guaranteed when this routine is called via SH2 slave.



Title	Function	Function Name	No.
Function Specifications	Gets registration contents of interrupt vector	SYS_GETSINT	1.3

Format: void (*) () SYS_GETSINT (Uint 32 Num) ;
Input: Num : vector number (0..7FH)
Output: None
Function Value: Vector registration contents (interrupt process routine address)
Function: Contents of the Num vector returned as function values.
There is no range check. Values beyond restricted range must not be specified.
Remarks: This routine can be used with both master and slave SH2 and refers to the vector addresses based on each VBR.

Title	Function	Function Name	No.
Function Specifications	Gets registration contents of SCU interrupt routine	SYS_GETUINT	1.4

Format: void (*) () SYS_GETUINT (Uint 32 Num) ;
Input: Num : vector number (SCU vector number)
Output: None
Function Value: Registration contents (function routine address)
Function: The registration contents of the SCU interrupt routine that pertains to Num returned as function values. There is no range check. Values beyond restricted range must not be specified.
Remarks: Results are not guaranteed when this routine is called via slave SH2.

Title	Function	Function Name	No.
Function Specifications	Sets SCU interrupt mask	SYS_SETSCUIM	2.1

Format: void SYS_SETSCUIM (Uint 32 MaskPat) ;
Input: MaskPat : SCU interrupt mask value
Output: None
Function Value: None
Function: Writes the MaskPat value to mask save memory and to the SCU interrupt register, and then writes the same value to the SCU interrupt status register. However, if the A-bus interrupt mask bit is allowed, the upper word of the status register is cleared, and the A-bus interrupt acknowledge register is cleared as well.
Remarks: This routine must not be used from the interrupt process (for SCU interrupt only) (The value of SYS_GETSCUIM becomes undefined during the SCU interrupt process.)
The SCU interrupt may disappear when it occurs during the process (SCU specifications). This possibility should either be avoided or the usage should occur under conditions where there is no related knowledge thereof. Results cannot be guaranteed when this routine is called via slave SH2.

Title	Function	Function Name	No.
Function Specifications	Changes SCU interrupt mask	SYS_CHGSCUIM	2.2

Format: void SYS_CHGSCUIM (Uint 32 AndMask, Uint32 OrMask) ;

Input: AndMask : Mask value used for authorizing
OrMask : Mask value used for denying

Output: None

Function Value: None

Function: Takes the logical product of the contents of the mask memory and the AndMask and writes the result of the logical sum of that and the OrMask to the mask save memory and the SCU interrupt. However, if the A-bus interrupt mask bit is allowed, the upper word of the status register is cleared, and the A-bus interrupt acknowledge register is cleared as well. These operations are executed inseparably.

Remarks: This routine must not be used from the interrupt process (for SCU interrupt only) (The value of SYS_GETSCUIM becomes undefined during the SCU interrupt process.)
The SCU interrupt may disappear when it occurs during the process (SCU specifications). This possibility should either be avoided or the usage should occur under conditions where there is no related knowledge thereof. Results cannot be guaranteed when this routine is called via slave SH2.

Title	Function	Function Name	No.
Function Specifications	References SCU interrupt mask value	SYS_GETSCUIM	2.3

Format: Uint 32 SYS_GETSCUIM ;

Input: None

Output: None

Function Value: Mask save memory value

Function: This function reads the mask save memory value. If this value performs settings and changes of the SCU interrupt mask register using the aforementioned function, it is the same as the value actually set in the SCU interrupt mask register.

Remarks: During the SCU interrupt process, the change is made to the value that is set when the application uses SYS_SETSCUIM () and SYS_CHGSCUIM () where the change is made to the logical sum of the mask value by interrupt cause factor (when the application uses SYS_CHGUIPR() and setting has been made, the corresponding values within that table). However, the interrupt becomes multi-level and changes to the logical sum. Consequently, this value becomes undefined during the SCU interrupt process. During this procedure, a process that relies on referenced values should not be performed.



Title	Function	Function Name	No.
Function Specifications	Gets semaphore	SYS_TASSEM	3.1

Format: `Uint 32 SYS_TASSEM (Uint 32 Num) ;`
Input: `Num` : semaphore number (0 ~ FFH)
Output: None
Function Value: Result (1: acquired, 0: already acquiring another)
Function: TAS command to memory (1 byte) linked to Num number is executed and the results are returned to function values. There is no range check. Values beyond the restricted range must not be specified. This routine can be used with both master and slave SH2.

Title	Function	Function Name	No.
Function Specifications	Clears semaphore	SYS_CLRSEM	3.2

Format: `void SYS_CLRSEM (Uint 32 Num) ;`
Input: `Num` : semaphore number (0 ~ FFH)
Output: None
Function Value: None
Function: Clears the memory (1 byte) linked to Num number. There is no range check. Values outside restrictions must not be specified. This routine can be used with both master and slave SH2.

Title	Function	Function Name	No.
Function Specifications	Switches System Clock	SYS_CHGSYSCK	4.1

Format: `void SYS_CHGSYSCK (Uint 32 CkMode) ;`
Input: `CkMode`: 0: CPU 26 MHz, 320/640 Mode
 1: CPU 28 MHz, 352/704 Mode
Output: None
Function Value: None
Function: System clock is switched to the value specified by CkMode.
Remarks: See the overview and the SMPC Manual regarding reset devices and processing time. The system hangs up when this routine is called via slave SH2. Be sure to call via master SH2.

Title	Function	Function Name	No.
Function Specifications	References System Clock Value	SYS_GETSYSCK	4.2

Format: Uint 32 SYS_GETSYSCK ;
Input: None
Output: None
Function Value: 0 or 1: the final SYS_CHGSYSCK () parameter value
Function: Reads the system clock value. Parameter value when SYS_CHGSYSCK () is called for the last time.
Remarks: Please reference this value via master SH2.

Title	Function	Function Name	No.
Function Specifications	Change SCU Interrupt Routine Priority	SYS_CHGUIPR	5.1

Format: void SYS_CHGUIPR (Uint 32 *IprTab) ;
Input: IprTab : 32 long word data array
Output: None
Function Value: None
Function: The SCU interrupt routine priority table of the Boot ROM is rewritten as the table value specified by IprTab. If rewrite is performed once, interrupt processing by the SCU interrupt routine is executed in accordance with the priority of the table values pertaining to each factor.

Note: The table contents are not checked. If inconsistencies relating to priority exist in the table, the system may hang up.

Remarks: Table settings are valid until the next rewrite. During this interval, there is no need for the application to save the table specified in the parameters. Further, reset returns to the initial set value of the Boot ROM. The results cannot be guaranteed when this routine is called via slave SH2.

Title	Function	Function Name	No.
Function Specifications	Starts and Executes CD Multiplayer	SYS_EXECDMP	6.1

Format: void SYS_EXECDMP (void) ;
Input: None
Output: Does not return to call side.
Function Value: None
Function: Starts and executes the CD multiplayer.
Remarks: The system hangs up when this routine is called via slave SH2. Be sure to call via master SH2.



Title	Function	Function Name	No.
Function Specifications	Operates Power On Clear Memory	SYS_PCLRMEM	7.1

Format: Uint8 *SYS_PCLRMEM
Input: Perform normal memory access.
Output: Perform normal memory access.
Function Value: None
Function: 8 byte memory address controlled by the Boot ROM. This memory is initialized to 0 only when the power is turned on, but the contents are saved by the reset button (NMI).
Remarks: A check of the range is not performed. Be careful not to access outside the range.

SEGA Confidential

(This page is blank in the original Japanese version.)

SEGA Confidential



SMPC I/F User's Manual

1.0 Guide

This library uses functions of the SMPC (System Management and Peripheral Control).

2.0 Features

- Able to greatly decrease the main CPU burden.
- The main CPU and SMPC interface is a software handshake method using the SMPC register.

3.0 Overview

3.1 Function Overview

SMPC functions can be divided into the system management system and peripheral control system.

- **System Management System**
The system management system resets the hardware (CPU, sound, etc.), sets the clock, and performs acquisition.
- **Peripheral Control System**
The peripheral control system collects data automatically from peripherals connected to the peripheral I/F, and returns it to the main CPU.

3.2 Process Overview

The interface between the SMPC and the main CPU is a handshake. In executing a function (command), required parameters are written to the SMPC register. The command write procedure is followed. SMPC executes processing in response to commands when commands are written. Processes include IntBack and Non-IntBack.

- **IntBack**
Intback is the process of returning results through interrupts (SMPC interrupts) after commands are passed to the SMPC. SMPC interrupt processing and registration are performed in the library. Also, other interrupt processes are prohibited during the SMPC interrupt process. Timing of issuing command issuance differs according to the function.
- **Non-IntBack**
Non-IntBack is a process that only passes commands to the SMPC. The library function waits for the SMPC process to end. (See the SMPC User's Manual for details of each SMPC process time.) The functions (reset, etc.) that can be used by this process are functions which do not have to receive results after a command has been passed.

Shown below are the processing systems that can be used by each function.

O : Processing, X : Not processing

	IntBack	Non-IntBack
System Management	O	O
Peripheral Control	O	X



4.0 Details

4.1 Functions

- **System Management System**

The system management system has the following functions.

Function	IntBack	Non-IntBack
Master SH2 ON	X	O
Slave SH2 ON	X	O
Slave SH2 OFF	X	O
Sound ON	X	O
Sound OFF	X	O
CD ON	X	O
CD OFF	X	O
Entire system reset	X	O
NMI Request	X	O
Hot reset enable	X	O
Hot reset disable	X	O
Get cartridge code	O	X
Get area code	O	X
Get system status	O	X
Set SMPC memory	X	O
Get SMPC memory	O	X
Set time	X	O
Get time	O	X

(Note) Clock change 320,352 is provided by the system library.

- **Peripheral Control System**

The following peripherals are supported by the library.

Game Device	Peripheral Name
Saturn Peripherals	Digital Devices
	Analog Devices
	Pointing Devices (mouse)
	Keyboard
	Multitap (6P)
Mega Drive Peripherals	3-button Pad
	6-button Pad
	4P adapter
	Mouse

4.2 Process

4.2.1 Library Configuration

- **Non-IntBack Command Issuance**
This is the Non-IntBack system process.
- **IntBack Command Issuance**
Gets system data (except for time)
This process is performed once per game.

Gets peripheral data and time data
This is a process required for each frame.

4.2.2 Recommended Examples

Recommended examples of three patterns are shown below.

		1	2	3
Non-IntBack Command Issuance			0	
IntBack Command Issuance	Get system data			0
	Get peripheral data	0	0	0
	Get peripheral data and time data	0		0



- **Chart Descriptions**

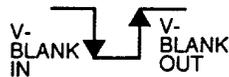
Main Process

- IntBack initialization (1): Specifies system data acquisition
- IntBack initialization (2): Specifies peripheral data acquisition
- IntBack initialization (3): Specifies peripheral data acquisition and time data acquisition

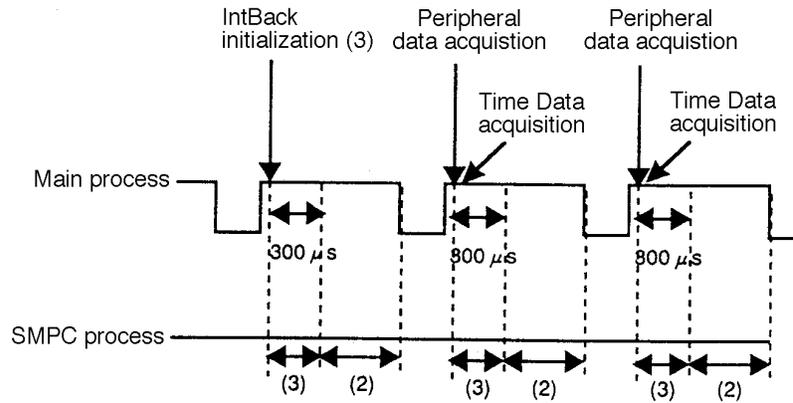
SMPC Process

- (1): System data collection process
- (2): Peripheral data collection process
- (3): Time data collection process

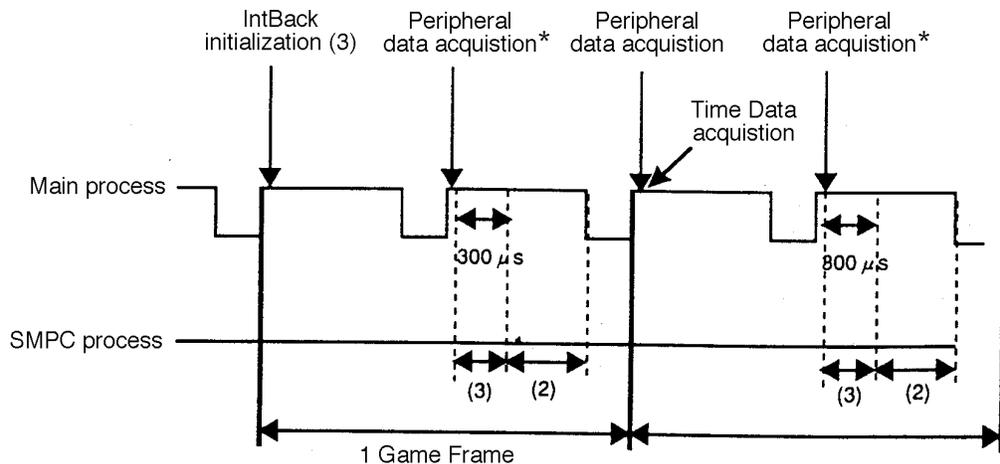
(1) Pattern 1



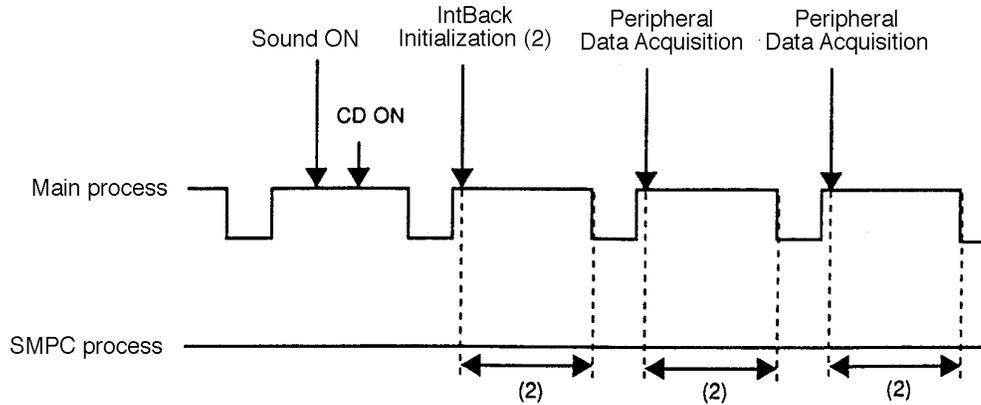
- When the game frame is 1 frame (when there is 1 game frame).



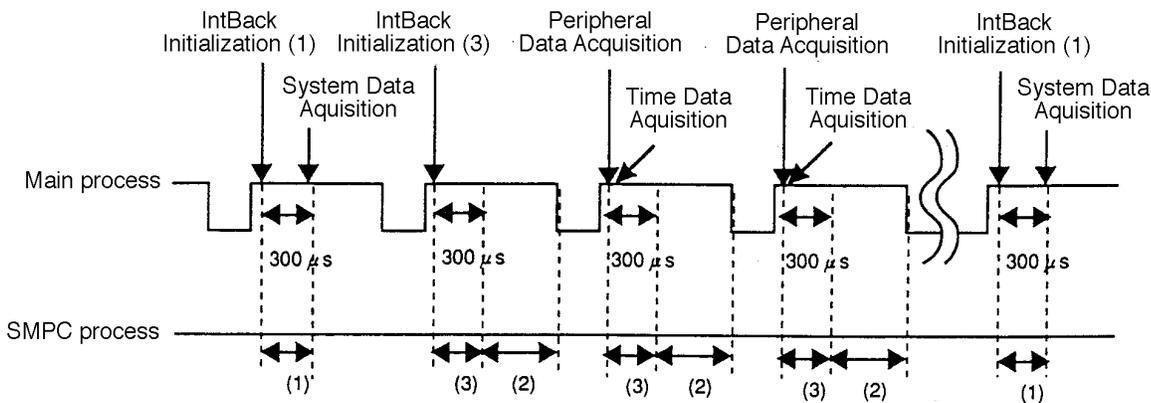
- When the game frame is 2 frame (when there are 2 game frames).
Peripheral data acquisition* does not perform peripheral data acquisition.



(2) Pattern 2



(3) Pattern 3



4.2.3 Restrictions

- **Common**
- Do not issue commands after V-Blank IN until 300 μs has elapsed.
- Do not issue commands after issuing the IntBack command until the next V-Blank IN.
- Cautions when executing in multiple tasks
 - In cases in which the interrupt task is changed when using the library with multiple tasks, because SMPC exclusive control is not performed in the library, SMPC deadlock may occur and SMPC operation may become abnormal. As a result, when executing with multiple tasks, the exclusive control of library must be performed by the user.
- For restrictions from slave SH2 see the SMPC User's Manual.
- Command process time may vary between 20 μs to 100 μs, because there is 1 SMPC internal process per second.
- **Non-IntBack Command Issue Function**
- There are no restrictions other than Common (above).



- **IntBack Command Issue Function**

System Data Acquisition (Excluding time)

- There are no restrictions other than Common (above).

Peripheral Data Acquisition, Time Data Acquisition

- Do not issue a command after issuing the IntBack command until the next V-Blank IN.
- Issue a IntBack command 14 ms before the specified V-Blank IN.
- The peripheral data collection process within SMPC is done during one vertical display period.
- (Time out) data that could not be acquired at the instructed V-Blank IN after the IntBack command is issued cannot be acquired thereafter.
- If a time out frequently occurs, peripherals are handled as unconnected.
- In SMPC, peripheral acquisition begins in such a way that the peripheral acquisition ends 1ms before the specified V-Blank.
- Number of V-Blank IN Skips

The number of V-Blank skips indicates the execution timing of the peripheral data collection process in SMPC after the IntBack command is issued. This is provided for games that change frames by 2 frames or more. The peripheral data collection process in SMPC is executed in V-DISP before the (V-Blank In skip number + 1)th V-Blank IN after the IntBack command is issued. Set 0 when executing in V-Blank IN immediately after issuing the IntBack command.

4.3 Peripheral Control

Peripheral controls must be created in accordance with game creation standards. The library is created intelligently, thus enabling them to be created easily.

4.3.1 Policy

The peripheral acquisition process is created according to the following policies for flexible response to the user.

- Able to accommodate peripherals to be sold in the future.
- Methods that do not provide multitap awareness.
- Perform corrective measures against problems.

4.3.2 Method

(1) Overview

- Each peripheral IDs and peripheral data is obtained by specifying as inputs the required number of peripherals, peripheral IDs and preferred size.
- Even when the peripheral acquisition ID and connected peripheral ID are different, data is obtained if peripherals are connected. For example, analog device data is obtained as mouse data.

(Example)

Input

Required number of peripherals = 3
Peripheral ID = digital device
Peripheral size = digital device size

Connection Status

Main connector 1 = multi-tap 6P
Multi-tap connector 1 = digital device
Multi-tap connector 3 = analog device
Main connector 2 = keyboard device
(Other devices are not connected to peripherals)

Output

	Peripheral ID	Peripheral Data
No. 1	Digital	Digital Format
No. 2	Unconnected	Invalid
No. 3	Analog	Digital Format



5.0 Calling Sequence

Recommended example of pattern 3 calling sequence.

```
Uint32 work[10];      /* peripheral data acquisition work area */
PerGetSys *sys_data; /* system data */

PerDgtInfo *get_per; /* peripheral output data pointer */
Uint8 *get_tim;     /* time output data pointer */
```

- Initial Process (Immediately after the V-Blank Process).

```
PER_Init (PER_KD_SYS, 0, 0, 0, NULL, 0); /* get system data is specified*/
. . .
sys_data = PER_GET_SYS (); /* get system data */
```

- Normal (Immediately after the V-Blank Process).

```
PER_Init (PER_KD_PERTIM, 3, PER_ID_DGT, PER_SIZE_DGT, work, 0);
/* Peripheral data and time data acquisition are specified */
...
```

- Normal for 2 or more times (Immediately after the V-Blank Process).

```
PER_GetPer (PerGetPer **) &get_per); /* get peripheral data */
get_tin = PER_GET_TIM (); /* get time */
if (((get_per[2].data) & PER_DGT_U) == PER_DGT_U) { /* current pe-
ripheral data of connector 3 is UP ? */
. . .
. . .
```

6.0 Data Specifications

6.1 List of Data

Function		Function name	No.	
IntBack	Peripheral ID data type	PerId	1	
	Peripheral size data type	PerSize	2	
	IntBack Kind	PerKind	3	
	Required number of peripherals	PerNum	4	
	System data output data type	PerGetSys	5	
	Peripheral data output data type	PerGetPer	6	
	Device information data type	Digital device info data type	PerDgtInfo	7
		Analog device info data type	PerAnlInfo	8
		Pointing device info data type	PerPntInfo	9
		Keyboard device info data type	PerKbdInfo	10
		Mega Drive 3-button pad info data type	PerM3bpInfo	11
		Mega Drive 6-button pad info data type	PerM6bpInfo	12
	Device Data Type	Digital device data type	PerDgtData	13
		Analog device data type	PerAnlData	14
		Pointing device data type	PerPntData	15
		Keyboard device data type	PerKbdData	16
		Mega Drive 3-button pad data type	PerM3bpData	17
		Mega Drive 6-button pad data type	PerM6bpData	18



6.2 Data Specifications

IntBack command issuance

Title Data Specifications	Data Peripheral ID data type	Data Name PerId	No. 1
---------------------------------	---------------------------------	--------------------	----------

This data type shows the peripheral ID.

Constant name	Description
PER_ID_NCON	Unconnected
PER_ID_UNKNOW	Peripherals that cannot be processed by SMPC
PER_ID_DGT	Digital Device
PER_ID_ANL	Analog Device
PER_ID_PNT	Pointing Device (Mega Drive mouse)
PER_ID_KBD	Keyboard
PER_ID_M3BP	Mega Drive 3-button pad
PER_ID_M6BP	Mega Drive 6-button pad

Title Data Specifications	Data Peripheral size data type	Data Name PerSize	No. 2
---------------------------------	-----------------------------------	----------------------	----------

This data type shows the peripheral size.

Constant name	Description
PER_SIZE_DGT	Digital Device
PER_SIZE_ANL	Analog Device
PER_SIZE_PNT	Pointing Device (Mega Drive mouse)
PER_SIZE_KBD	Keyboard
PER_SIZE_M3BP	Mega Drive 3-button pad
PER_SIZE_M6BP	Mega Drive 6-button pad

Title Data Specifications	Data IntBack kind data type	Data Name PerKind	No. 3
------------------------------	--------------------------------	----------------------	----------

This data type shows the IntBack kind.

Constant name	Description
PER_KD_SYS	System data acquisition (except time)
PER_KD_PER	Peripheral data acquisition
PER_KD_PERTIM	Peripheral data acquisition + time data acquisition

Title Data Specifications	Data Required number of peripherals data type	Data Name PerNum	No. 4
------------------------------	--	---------------------	----------

This data type shows the required number of peripherals.

Value	Meaning
0 ~ 31	1P ~ 32P

Title Data Specifications	Data System data output data type	Data Name PerGetSys	No. 5
------------------------------	--------------------------------------	------------------------	----------

This data type shows the system data output.

PerGetSys *data

Access Macro	Type	Description
PER_GS_AC (data)	UInt8	Area code
PER_GS_SS (data)	UInt16	System status
PER_GS_SM (data)	UInt32	SMPC memory
PER_GS_SMPC_STAT (data)	UInt8	SMPC status



Shown below are the constants and values that can be used by each access macro.

PER_GS_CC (data)

bit 7							bit 0
0	0	0	0	0	0	CTR1	CTR0

PER_GS_AC (data)

bit 7							bit 0
0	0	0	0	ACODE 3	ACODE 2	ACODE 1	ACODE 0

See the hardware manual for areas indicated an area codes.

PER_GS_SS (data)

Bit Position Constant	Acquisition Value
PER_SS_DOTSEL	DOTSEL signal status
PER_SS_SYSRES	SYSRES signal status
PER_SS_MSHNMI	MSHNMI signal status
PER_SS_SNDRES	SNDRES signal status
PER_SS_CDRES	CDRES signal status

Description of Acquisition Values

Value	Meaning
0	OFF
1	ON

PER_GS_SM (data)

Areas which are used in common by applications and Boot ROM.

Bit Position Constant	Acquisition Value
PER_MSK_LANGU	Language (see below)
PER_MSK_SE	SE (0: ON, 1: OFF)
PER_MSK_STEREO	STEREO or MONO (0: STEREO, 1: MONO)
PER_MSK_HELP	HELP (0: ON, 1: OFF)

Language Constant

Constant	Description
PER_JAPAN	Japanese
PER_ENGLISH	English
PER_FRANCAIS	French
PER_DEUTSCH	German
PER_ITALIANO	Italian
PER_ESPNOL	Spanish

PER_GS_SMPC_STAT (data)

Bit Position Constant	Acquisition Value
PER_SS_RESET	Reset mask condition
	0: reset enable
	1: reset disable (default)
PER_SS_SETTIME	Time set condition
	0: time is not set after SMPC cold reset
	1: time is set after SMPC cold reset

Title	Data	Data Name	No.
Data Specifications	Peripheral data output data type	PerGetPer	6

This data type shows peripheral data output.

• Device Information Data Type

Title	Data	Data Name	No.
Data Specifications	Digital device information data type	PerDgtInfo	7

This data type shows digital device information.

```
typedef struct { /* digital device */
    PerDgtData data; /* current peripheral data */
    PerDgtData push; /* previously not pressed currently pressed button */
    PerId id; /* peripheral ID */
} PerDgtInfo;
```

Title	Data	Data Name	No.
Data Specifications	Analog device information data type	PerAnlInfo	8

This data type shows analog device information.

```
typedef struct { /* analog device */
    PerAnlData data; /* current peripheral data */
    PerAnlData push; /* previously not pressed currently pressed button */
    PerId id; /* peripheral ID */
} PerAnlInfo;
```



Title	Data	Data Name	No.
Data Specifications	Pointing device information data type	PerPntInfo	9

This data type shows pointing device information.

```
typedef struct {          /* pointing device          */
    PerPntData data;      /* current peripheral data  */
    PerPntData push;     /* previously not pressed  */
    PerId id;            /* peripheral ID            */
} PerPntInfo;
```

Title	Data	Data Name	No.
Data Specifications	Keyboard device information data type	PerKbdInfo	10

This data type shows keyboard device information.

```
typedef struct {          /* keyboard device          */
    PerKbdData data;     /* current peripheral data  */
    PerKbdData push;     /* previously not pressed  */
    PerId id;            /* peripheral ID            */
} PerKbdInfo;
```

Title	Data	Data Name	No.
Data Specifications	Mega Drive 3-button pad information data type	PerM3bpInfo	11

This data type shows Mega Drive 3-button pad information.

```
typedef struct {          /* Mega Drive 3-button pad */
    PerM3bpData data;    /* current peripheral data  */
    PerM3bpData push;    /* previously not pressed  */
    PerId id;            /* peripheral ID            */
} PerM3bpInfo;
```

Title	Data	Data Name	No.
Data Specifications	Mega Drive 6-button pad information data type	PerM6bpInfo	12

This data type shows Mega Drive 6-button pad information.

```
typedef struct {          /* Mega Drive 6-button pad */
    PerM6bpData data;    /* current peripheral data  */
    PerM6bpData push;    /* previously not pressed  */
    PerId id;            /* peripheral ID            */
} PerM6bpInfo;
```

• **Device Data Type**

The meaning of the device data bit acquisition value is explained below.

Description of Acquisition Values

Value	Meaning
0	Button is pressed
1	Button is not pressed

Title	Data	Data Name	No.
Data Specifications	Digital device data type	PerDgtData	13

This data type shows the digital device.

```
typedef Uint16 PerDgtData; /* digital device data type */
```

Bit Position Constant	Acquisition Value
PER_DGT_U	UP
PER_DGT_D	DOWN
PER_DGT_R	RIGHT
PER_DGT_L	LEFT
PER_DGT_A	A
PER_DGT_B	B
PER_DGT_C	C
PER_DGT_S	START
PER_DGT_X	X
PER_DGT_Y	Y
PER_DGT_Z	Z
PER_DGT_TR	TRG-RIGHT (upper right of the device)
PER_DGT_TL	TRG-LEFT (upper left of the device)

Title	Data	Data Name	No.
Data Specifications	Analog device data type	PerAnlData	14

This data type shows the analog device.

```
typedef struct {
    PerDgtData dgt; /* analog device data type */
    Sint16 x; /* digital device data type */
    Sint16 y; /* X axis absolute value (0 ~ 255) */
    Sint16 z; /* Y axis absolute value (0 ~ 255) */
} PerAnlData; /* Z axis absolute value (0 ~ 255) */
```



Title	Data	Data Name	No.
Data Specifications	Pointing device data type	PerPntData	15

This data type shows the pointing device.

```
typedef struct { /* pointing device data type */
    PerPntData dgt; /* digital device data type */
    Sint16 data; /* data */
    Uint16 x; /* amount of X axis movement (-128 ~ 127) */
    Sint16 y; /* amount of Y axis movement (-128 ~ 127) */
} PerPntData; /*
```

Shown below are constants and values that can be used by each member.

Data	
Bit Position Constant	Acquired value
PER_PNT_R	RIGHT
PER_PNT_L	LEFT
PER_PNT_MID	MIDDLE
PER_PNT_CNT	CENTER
PER_PNT_X0	X axis overflow (0: overflows, 1: does not overflow)
PER_PNT_Y0	Y axis overflow (0: overflows, 1: does not overflow)

Title	Data	Data Name	No.
Data Specifications	Keyboard device data type	PerKbdData	16

This data type shows the keyboard device.

```
typedef struct { /* keyboard device data type */
    PerDgtInfo dgt; /* digital device data type */
    Uint8 skey; /* special key */
    Uint8 key; /* key */
} PerKbdData; /*
```

Shown below are constants and values that can be used by each member.

skey	
Bit Position Constant	Acquired value
PER_KBD_CL	Caps Lock
PER_KBD_NL	Num Lock
PER_KBD_SL	Scroll Lock
PER_KBD_MK	Make (0: key pressed, 1: key not pressed)
PER_KBD_BR	Break (0: key released, 1: key not released)

Title	Data	Data Name	No.
Data Specifications	Mega Drive 3-button pad data type	PerM3bpData	17

This data type shows the Mega Drive 3-button pad.

```
typedef Uint8 PerM3bpData; /* Mega Drive 3-button pad data type */
```

PER_M3BP_U ~ PER_M3BP_S is the same as PER_DGT_U ~ PER_DGT_S. PER_DGT_X ~ PER_DGT_TL is the condition when the button is not pressed.

Bit Position Constant	Acquired Value
PER_M3BP_U	UP
PER_M3BP_D	DOWN
PER_M3BP_R	RIGHT
PER_M3BP_L	LEFT
PER_M3BP_A	A
PER_M3BP_B	B
PER_M3BP_C	C
PER_M3BP_S	START

Title	Data	Data Name	No.
Data Specifications	Mega Drive 6-button pad data type	PerM6bpData	18

This data type shows the Mega Drive 6-button pad.

```
typedef Uint16 PerM6bpData; /* Mega Drive 6-button pad data type */
```

PER_M6BP_U ~ PER_M6BP_MD is the same as PER_DGT_U ~ PER_DGT_TR. PER_DGT_TL is the condition when the button is not pressed.

Bit Position Constant	Acquisition Value
PER_M6BP_U	UP
PER_M6BP_D	DOWN
PER_M6BP_R	RIGHT
PER_M6BP_L	LEFT
PER_M6BP_A	A
PER_M6BP_B	B
PER_M6BP_C	C
PER_M6BP_S	START
PER_M6BP_X	X
PER_M6BP_Y	Y
PER_M6BP_Z	Z
PER_M6BP_MD	MODE (upper right of device)



7.0 Function Specifications

7.1 List of Functions

		Function	Function Name	No.
Non-IntBack	Command Issue	Master SH2 ON	PER_SMPC_MSH_ON	1
		Slave SH2 ON	PER_SMPC_SSH_ON	2
		Slave SH2 OFF	PER_SMPC_SSH_OFF	3
		Sound ON	PER_SMPC_SND_ON	4
		Sound OFF	PER_SMPC_SND_OFF	5
		CD ON	PER_SMPC_CD_ON	6
		CD OFF	PER_SMPC_CD_OFF	7
		Reset entire system	PER_SMPC_SYS_RES	8
		NMI request	PER_SMPC_NMI_REQ	9
		Hot reset enable	PER_SMPC_RES_ENA	10
		Hot reset disable	PER_SMPC_RES_DIS	11
		SMPC memory set	PER_SMPC_SET_SM	12
		Time set	PER_SMPC_SET_TIM	13
IntBack	Command Issue	IntBack Initialization	PER_Init	14
		Peripheral data acquisition	PER_GetPer	15
	Other	Time acquisition	PER_GET_TIM	16
		System data acquisition	PER_GET_SYS	17
Other		Hot reset acquisition	PER_GET_HOT_RES	18

7.2 Function Specifications

Non-IntBack

Title	Function	Function Name	No.
Function Specifications	Master SH2 ON ~ hot reset disable	PER_SMPC_MSK_ON ~ PER_SMPC_RES_DIS	1 ~ 11

Format: void PER_SMPC_XXX (void)

Input: None

Output: None

Function Value: None

Function: As per the list of function specifications. See the SMPC hardware manual for details.

Title	Function	Function Name	No.
Function Specifications	Sets SMPC memory	PER_SMPC_SET_SM	12

Format: void PER_SMPC_SET_SM (Uint32 input_dt)
Input: input_dt : SMPC memory
 See System data output data type for the meaning of each bit value.
Output: None
Function Value: None
Function: Sets the SMPC memory. Because the SMPC memory is an area that is used in common by applications and the Boot ROM, the format must be observed.

Title	Function	Function Name	No.
Function Specifications	Sets time	PER_SMPC_SET_TIM	13

Format: void PER_SMPC_SET_TIM (Uint8 * input_dt)
Input: input_dt: time

Input Format

	bit7	bit4	bit3	bit0
*(input_dt)	second (10 digit)		second (1 digit)	
*(input_dt + 1)	minute (10 digit)		minute(1 digit)	
*(input_dt + 2)	hour (10 digit)		hour (1 digit)	
*(input_dt + 3)	day (10 digit)		day (1 digit)	
*(input_dt + 4)	day of week (0 ~ 6)		month (1H ~ CH)	
*(input_dt + 5)	year (10 digit)		year (1digit)	
*(input_dt + 6)	year (1000 digit)		year (100 digit)	

- Day data: Sunday is 0, Monday is 1, Tuesday is 2 . . .
- Month data is hexadecimal data.

Output: None
Function Value: None
Function: Sets the time.



IntBack

Title	Function	Function Name	No.
Function Specifications	Initializes IntBack	PER_Init	14

Format: Uint32 PER_Init (PerKind kind, PerNum num, PerId id, PerSize size, Uint32 work[n], Uint8 v_blank)

Input:

- kind : IntBack type
- num : Required number of peripherals
- id : Peripheral ID
- size : Peripheral size
- work : Work area (Use for getting peripheral data. Must be declared by a global variable.)
- v_blank : Number of V-Blank skip

- Method for calculating the work area

$$n = (\text{num} \times \text{data A}) / 4 + \text{data B}$$

round up to nearest digit

Data A and data B change depending on the peripheral.

Peripheral	Data A	Data B
Digital device	12	1
Analog device	36	2
Pointing device	36	2
Keyboard device	20	1
Mega Drive 3-button pad	6	1
Mega Drive 6-button pad	12	1

Output: none
Function Value: execution condition

Constant	Description
PER_INT_ERR	Could not issue the IntBack command
PER_INT_OK	Could issue the IntBack command

Function: Initializes IntBack and issues the IntBack command. Execution rules must be observed. Set Null or 0 to unneeded parameters. Execute at least 1 time before executing PER_GetPer (), PER_GetTim(), PER_GetSys().

Remarks: Generally, this should be executed immediately after the V-Blank process. Other interrupts are prohibited during SMPC interrupts.

Note: DO NOT perform this function within the interrupt process. Be sure to perform via the main process.

Title	Function	Function Name	No.
Function Specifications	Peripheral data acquisition	PER_GetPer	15

Format: Uint32 PER_GetPer (PerGetPer **output_dt)
Input: none
Output: output_dt: peripheral output address (Null = cannot get)
Function Value: Execution conditions.

Constant	Description
PER_INT_ERR	Could not issue the IntBack command.
PER_INT_OK	Could not issue the IntBack command.

Function Value: Issues the IntBack command and gets peripheral data. Execution rules must be observed. Before this function is executed, specify “peripheral data get” to PER_Init() and execute at least once. Null is output to the peripheral data address when peripheral data get is not specified.

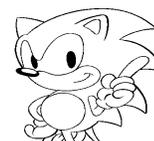
Remarks: Generally, this should be executed immediately after the V-Blank process. Other interrupts are prohibited during SMPC interrupt.

Title	Function	Function Name	No.
Function Specifications	Gets time	PER_GET_TIM	16

Format: Uint8 *PER_GET_TIM (void)
Input: none
Output: none
Function Value: Time data address
Function: Gets time data. Before this function is executed, instructs peripheral data get and time get to PER_Init() and execute at least once.

Title	Function	Function Name	No.
Function Specifications	Gets system data	PER_GET_SYS	17

Format: PerGetSys *PER_GET_SYS (void)
Input: none
Output: none
Function Value: System data address (NULL = could not get)
Function: Gets system data. Instructs system data get to PER_Init () before this function is executed and execute at least once. Execute this function about 300 μs after executing PER_Init.



Other

Title	Function	Function Name	No.
Function Specifications	Gets hot reset	PER_GET_HOT_RES	18

Format: Uint8 PER_GET_HOT_RES (void)

Input: none

Output: none

Function Value: Hot reset condition address

Constant	Description
PER_HOT_RES_ON	Hot reset ON
PER_HOT_RES_OFF	Hot reset OFF

Function: Gets the hot reset condition. This function can be executed at any time. Update is performed by PER_GetPer().

(This page is blank in the original Japanese version.)

SEGA Confidential



Backup Library User's Manual

1.0 Guide

1.1 Purpose

In addition to the built-in memory in this game machine, several types of storage devices are being planned for storing information during a game.

This library provides functions for reading, writing and searching these backup storage devices.

1.2 Explanation

1.2.1 Introduction

Always use this library when accessing storage devices for backup.

1.2.2 How to Use This Library

The library itself is compressed and stored in the boot ROM. The application programmer expands the library and uses it. Expansion is performed by securing a program expansion area and executing BUP_Init(). Once this is done, each function can be used.

1.2.3 Storage Capacity

The object of this library is to facilitate access of devices to be supported in the future through a common interface, and therefore the capacity of the storage devices to be developed in the future is not known. Also, depending on the device, it will be divided up into multiple areas. Each one of these unit areas is called a partition (the capacity of each partition may be different).

When storing data, execute BUP_SelPart() and BUP_Stat() and confirm the capacity before writing. The built-in backup memory is 32 Kbyte.

1.2.4 Date Setting

In order to preserve the uniqueness of this library, it does not have a function for acquiring the date and time. Set the date and time data by having the application use BUP_SetDate().

1.2.5 Precautions

This library will destroy data if writing is interrupted. Before executing BUP_Init(), BUP_Format(), BUP_Write() and BUP_Delete, disable the reset button by using PER_SMPC_RES_DIS() in the system library.

1.3 Program Example

An example of a program written in C is shown below.

```
#include "sega_per.h"
#define BUP_START_ADDR 0x60????0 /*sets write address for library */
#include "sega_bup.h"

Unit32 BackUpRamWork[2048];

main()
[
    BupConfig  conf[3]
    BupStat    sttb;
    BupDir     writetb;
    BupDate    datatb;
    Unit8      *time;

    PER_SMPC_RES_DIS(); /*disables reset button */
    BUP_Init(BUP_START_ADDR, BackUpRamWork, conf);
    if(BUP_Stat(0, &sttb)==BUP_UNFORMAT) {
        BUP_Format(0);
    }
    PER_SMPC_RES_ENA(); /*enables reset button */
    BUP_Stat(0, &sttb);
    if(sttb.freeblock > 0) {
        strcpy((char *)writetb.filename, "FILE_NAME01");
        STRCPY((char *)writetb.comment, "test");
        writetb.language = BUP_JAPANESE;
        time = PER_GET_TIM(); /*get date and time */
        datetb.year = (UInt8 )( (UInt16 )(time[6]>>4) * 1000
            + (UInt16 )(time[6] & 0x0F) * 100
            + (UInt16 )(time[5]>>4) * 10
            + (UInt16 )(time[5] & 0x0F) - 1980);
        datetb.month = time[4] & 0x0F;
        datetb.day = (time[3]>>4)*10 + (time[3] & 0x0F);
        datetb.time = (time[2]>>4)*10 + (time[2] & 0x0F);
        datetb.min = (time[1]>>4)*10 + (time[1] & 0x0F);
        writetb.date = BUP_SetDate(&datetb);
        writetb.datasize = 10;
        PER_SMPC_RES_DIS(); /*disable reset button */
        BUP_Write(0, &writetb, "Dummy Data");
        PER_SMPC_RES_ENA(); /*enable reset button */
    }
}
```



2.0 Reference

2.1 Data List

Title Data specification	Data Storage device connection information	Data Name BupConfig	No.
-----------------------------	---	------------------------	-----

```
typedef struct BupConfig {
    Uint16 unit_id;          /*unit ID          */
    Uint16 partition;       /*number of partitions */
} BupConfig;
```

*When unit_id is "0", it indicates non-connection.

Type of device	unit_id	partition
Built-in memory	1	1
External cartridge	2	1

Title Data specification	Function Status information	Function Name BupStat	No.
-----------------------------	--------------------------------	--------------------------	-----

```
typedef struct BupStat {
    Uint32 totalsize;       /*total capacity (bytes) */
    Uint32 totalblock;     /*number of blocks       */
    Uint32 blocksize;      /*size of one block (bytes) */
    Uint32 freesize;       /*open space             */
    Uint32 freeblock;      /*number of open blocks   */
    Uint32 datanum;        /*number of items that can be written */
} BupStat;
```

The value for the size specified by BUP_Stat() for datasize is stored in datanum.

Title Data specification	Function Date and time	Function Name BupDate	No.
-----------------------------	---------------------------	--------------------------	-----

```
typedef struct BupStat {
    Uint8 year;            /*year (1980 subtracted from year) */
    Uint8 month;          /*month (1-12)                    */
    Uint8 day;            /*day (1-31)                      */
    Uint8 time;          /*hour (0-23)                     */
    Uint8 min;           /*minute (0-59)                   */
    Uint8 week;          /*day of week (sunday 0- saturday 6) */
} BupDate;
```

Title	Function	Function Name	No.
Data specification	Directory information	BupDir	

```

typedef struct BupStat {
    Uint8  filename[12]; /*file name */
                        /*total 12 bytes ASCII 11 characters
                        + NUL */
    Uint8  comment[11]; /*comment */
                        /*total 11 bytes ASCII 10 characters
                        + NUL */
    Uint8  language; /*language of comment */
                        /* Japanese BUP_JAPANESE */
                        /* English BUP_ENGLISH */
                        /* French BUP_FRACAIS */
                        /* German BUP_DEUTSCH */
                        /* Spanish BUP_ESPANOL */
                        /* Italian BUP_ITALIANO */
    Uint32 date; /*date and time data */
    Uint32 datasize; /*data size (unit: byte) */
    Uint16 blocksize; /*data size (unit: block) */
} BupDir;

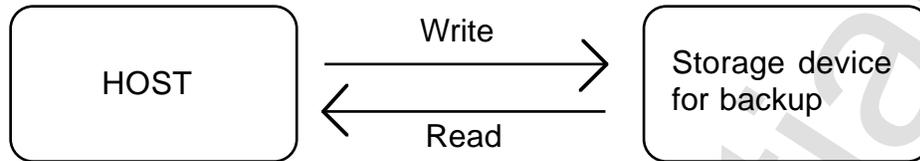
```

2.2 Function List

Function	Function Name	No.
Backup library		
Initialize backup library	BUP_Init	1
Select partition	BUP_SelPart	2
Execute format	BUP_Format	3
Get status	BUP_Stat	4
Write data	BUP_Write	5
Read data	BUP_Read	6
Delete data	BUP_Delete	7
Get directory information	BUP_Dir	8
Verify data	BUP_Verify	9
Open date and time data	BUP_GetDate	10
Compress data and time data	BUP_SetDate	11



2.3 Data Flow



2.4 Function Specifications

Title	Function	Function Name	No.
Function specification	Initializes backup library	BUP_Init	1

Format void BUP_Init(Uint32 *libaddr, Uint32 *workbuff, BupConfig conf[3])

Input libaddr : specifies address to which library is loaded.
The size of the library is 16 Kbytes.
workbuff : point for library work area
A work area size of 8192 bytes is required.
Long word access may also be performed, so be sure to secure with Uint32.

Output conf : Gets information on the connected storage device.

Function Loads the backup library to the specified memory area and prepares it for use. Gets information on the connected storage device.

The following device numbers correspond to three tables.

Device No.	Device Type
0	Built-in memory cartridge
1	Memory cartridge or parallel interface
2	Serial interface

Notes Always prepare three storage device connection information tables.

Example

```

#define BUP_START_ADDR 0x6??????
#include "sega_bup.h"
Uint32 workmemory[2048]

void sample()
{
    BupConfig conf[3]

    BUP_Init(BUP_START_ADDR, workmemory, conf);
    .....
}
  
```

Title	Function	Function Name	No.
Function specification	Partition selection	BUP_SelPart	2

Format Sint32 BUP_SelPart(Uint32 device, Uint16 num)

Input device : device number
0: built-in memory
1: memory cartridge or parallel interface
2: serial interface
num : partition number
0 - (number of partitions - 1)

Output none

Function value 0 : success
other : failure

Function Selects a partition. In the initial condition, partition 0 is selected.

Title	Function	Function Name	No.
Function specification	Executes format	BUP_Format	3

Format Sint32 BUP_Format(Uint32 device)

Input device : device number
0: built-in memory
1: memory cartridge or parallel interface
2: serial interface

Output none

Function value 0 : success
BUP_WRITE_PROTECT : write protected
other : failure

Function Initializes the backup storage device.
Formats only the current partition in a partitioned backup storage device.



Title	Function	Function Name	No.
Function specification	Gets status	BUP_Stat	4

Format Sint32 BUP_Stat(Uint32 device, Uint32 datasize, BupStat *stat)

Input device: device number
0: built-in memory
1: memory cartridge or parallel interface
2: serial interface

datasize : specify size of data to be written in byte units

Output stat : status information

Function value returns device status
0 : success
BUP_NON : not connected
BUP_UNFORMAT : not formatted

Function Gets status information.

Title	Function	Function Name	No.
Function specification	Writes data	BUP_Write	5

Format Sint32 BUP_Write(Uint32 device, BupDir *dir, Uint8 *data, Uint8 osw)

Input device : device number
0: built-in memory
1: memory cartridge or parallel interface
2: serial interface

dir : file control information (input other than dir.blocksize)

data : pointer for write data

osw: overwrite check mode
ON: does not write if file of same name exists
OFF: writes on file if file of same name exists

Output none

Function value 0 : success
BUP_NON : not connected
BUP_UNFORMAT : not formatted
BUP_WRITE_PROTECT : write protect exists
BUP_FOUND : file of same name exists
Other : failure

Function Writes data to the backup storage device.

Title	Function	Function Name	No.
Function specification	Loads data	BUP_Read	6

Format Sint32 BUP_Read(Uint32 device, Uint8 *fname, Uint8 *data)

Input device : device number
0: built-in memory
1: memory cartridge or parallel interface
2: serial interface
fname : file name specification (11 characters are required in ASCII)

Output data : pointer for load destination buffer

Function value 0 : success
BUP_NON : not connected
BUP_UNFORMAT : not formatted
BUP_NOT_FOUND : file not found
BUP_BROKEN : file is damaged
Other : failure

Function Loads data from the backup storage device.

Title	Function	Function Name	No.
Function specification	Deletes data	BUP_Delete	7

Format Sint32 BUP_Delete(Uint32 device, Uint8 *fname)

Input device : device number
0: built-in memory
1: memory cartridge or parallel interface
2: serial interface
fname : file name (11 characters is required in ASCII)

Output none

Function value 0 : success
BUP_NON : not connected
BUP_UNFORMAT : not formatted
BUP_NOT_FOUND : file not found
BUP_WRITE_PROTECT : write protect exists
Other : failure

Function Deletes data on the backup storage device.



Title	Function	Function Name	No.
Function specification	Gets directory information	BUP_Dir	8

Format Sint32 BUP_Dir(Uint32 device, Uint8 *fname, Uint16 dirsize, BupDir *dir)

Input

- device : device number
 - 0: built-in memory
 - 1: memory cartridge or parallel interface
 - 2: serial interface
- fname : file name specification (within 11 ASCII characters)
- dirsize : specifies the number of directory info secured
- dir : stores directory information

Output

Function value number of directory information hits

Function The file name is done by searching forward, and the directory information is stored in a table. The number of directories hit by the search is returned as the function value. If negative, it indicates that the table is too small, and the number of hits can be confirmed by inverting the sign (if the result is 11 when the tbsize is set to 10 and a search is performed, -11 is returned).

Title	Function	Function Name	No.
Function specification	Verify data	BUP_Verify	9

Format Sint32 BUP_Verify(Uint32 device, Uint8 *filename, Uint8 *data)

Input

- device : device number
 - 0: built-in memory
 - 1: memory cartridge or parallel interface
 - 2: serial interface
- filename : file name (11 characters are required in ASCII)
- data : pointer for data verification

Output

Function value

- 0 : success
- BUP_NON : not connected
- BUP_UNFORMAT : not formatted
- BUP_NO_MATCH : data do not agree
- BUP_NOT_FOUND : file not found
- BUP_BROKEN : file is damaged

Function Verifies data written to the backup file.

Title	Function	Function Name	No.
Function specification	Expands data and time data	BUP_GetDate	10

Format void BUP_GetDate(Uint32 pdate, BupDate *date)
Input pdate : data and time data of directory information
date : date and time table
Output none
Function value none
Function Expands the date and time data in the directory information table.

Title	Function	Function Name	No.
Function specification	Compress date and time data	BUP_SetDate	11

Format Uint32 BUP_SetDate(BupDate *date)
Input date : date and time table
Output none
Function value Data in date and time data form in the directory information table.
Function Compresses the date and time data in the directory information table.

