Software Development Seminar

Memory Card (Advanced)

Memory Card Programming

Memory Card Library

Contents	File name
Library	libcard.lib
Header	kernel.h
	sys\file.h

Hardware

Capacity: 120K bytes when formatted
(accessed as individual 128-byte sectors)

Communications method: Synchronous serial communications via the controller port

Access speed: (1) After one sector is written, access is not possible for 20ms.

(2) Maximum continuous read speed: approximately 10K bytes/second

Miscellaneous: Batteries not needed
Can be inserted/removed while the power is on
Guaranteed for 100,000 writes

BIOS

Access:	128-byte units for every two vertical retrace lines
Startup timing:	After the vertical retrace line interrupt and the controller read, the card connection is confirmed and handshaking is performed. The passing of the main body of data is driven by receive interrupts after each byte.
Effective speed:	30 sectors/second = 3.75K bytes/second
CPU load:	2.5% during consecutive reads from two cards 3.2% during consecutive writes to two cards

File System

Device name:	buX0 X: Connector number (0 or 1)
File name:	Up to 21 ASCIZ characters
Directory structure:	None
Unit of control:	Blocks 8K bytes (64 sectors) -> Units of file size
Number of blocks:	15 blocks/card (maximum number of files: 15)
File size:	Specified at time of CREATE; fixed thereafter.

Function List <File System>

File system

open	(references directory cache)
read	(asynchronous)
write	(asynchronous)
close	(does not access card)
firstfile	(references directory cache)
nextfile	(does not access card)
delete	(references directory cache)
rename	(references directory cache)
format	(occupies CPU completely for approximately 1.2 seconds)
lseek	(does not access card)

Known Bugs (partial list)

- After using open() to create a file, close the file immediately by calling close(). An error will result if read() or write() is issued.

- When an asynchronous access is performed using read(), the file pointer is updated to a value that is 128 bytes low. It is necessary to correct the pointer by issuing lseek().

Function List<BIOS>

BIOS

InitCARD	Initializes the memory card BIOS.
StartCARD	Starts the memory card BIOS.
StopCARD	Stops the memory card BIOS.
_bu_init	Initializes the memory card file system.
_card_info	Gets the card status.
_card_clear	Clears unchecked flags.
_card_load	Tests the logical format.
_card_auto	Sets the auto format function.
_new_card	Changes the unchecked flag test settings.
_card_status	Gets the memory card BIOS status.
_card_wait	Waits for completion of memory card BIOS processing.
_card_chan	Gets a memory card BIOS event.
_card_write	Writes one block to the memory card.
_card_read	Reads one block from the memory card.

Initialization Procedure

When used with a controller

```
InitPAD(&cbuf[0][0],34,&cbuf[1][0],34);
StartPAD();

InitCARD(1);              /* BIOS init. (when used with a control pad) */
StartCARD();              /* BIOS start */
_bu_init();               /* File system initialization */
ChangeClearPAD(0);        /* Enables processing of interrupt modules
                             with a lower priority than the controller driver.*/
OPEN SwCARD/HwCARD events
Enable SwCARD/HwCARD events
```

Initialization Procedure

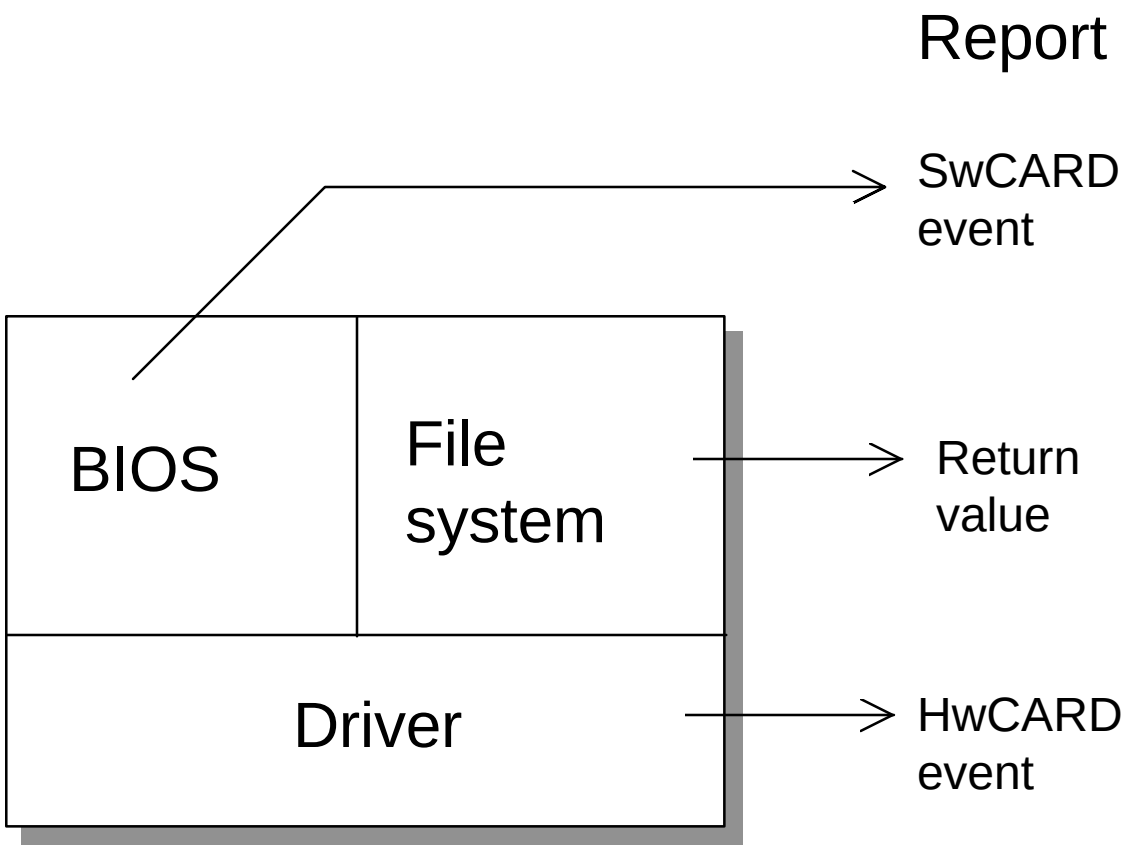
When used with other libraries

```
ResetCallback();          /* Callback function initialization*/
CdInit();                 /* CD initialization */
SsInit();                 /* Sound initialization */
ResetGraph(0);            /* Graphics initialization */
Pad initialization
Memory card initialization  /* ChangeClearPAD(0) must be executed */
```

Related Events

Source descriptor	Event type	Description
HwCARD	EvSpIOE	Processing end
	EvSpERROR	Card error
	EvSpTIMOUT	No card
	EvSpNEW	New card or uninitialized card
SwCARD	EvSpIOE	Processing end
	EvSpERROR	Card error
	EvSpTIMOUT	No card
	EvSpNEW	New card or uninitialized card

Error Report



File System and Error Reporting

Function name	Event sensing via TestEvent()		Result/Status determination
	HwCARD	SwCARD	
open(O_CREAT)	O	X	Function return value
open(O_RDONLY/BLOCK)	O	X	Function return value
open(O_RDONLY/NO_WAIT)	O	X	Function return value
open(O_WRONLY/BLOCK)	X	X	Function return value
open(O_WRONLY/NO_WAIT)	X	X	Function return value
close	X	X	----
read(BLOCK)	O	X	Function return value
read(NO_WAIT)	O	O	SwCARD (end determination)
write(BLOCK)	O	X	Function return value
write(NO_WAIT)	O	O	SwCARD (end determination)
format	O	X	HwCARD
firstfile	O	X	Function return value (HwCARD)
nextfile	X	X	----
delete	O	X	HwCARD
rename	O	X	HwCARD
_card_info	O	O	SwCARD
_card_load	O	O	SwCARD
_card_clear	O	X	HwCARD

Note: HwCARD is cleared at the second VSync (within 1/30 of a second) after the event is generated.

Asynchronous Access (READ/WRITE)

- The memory card device "bu" supports non-blocking mode.

  => Specify the macro O_NOWAIT when opening the device.

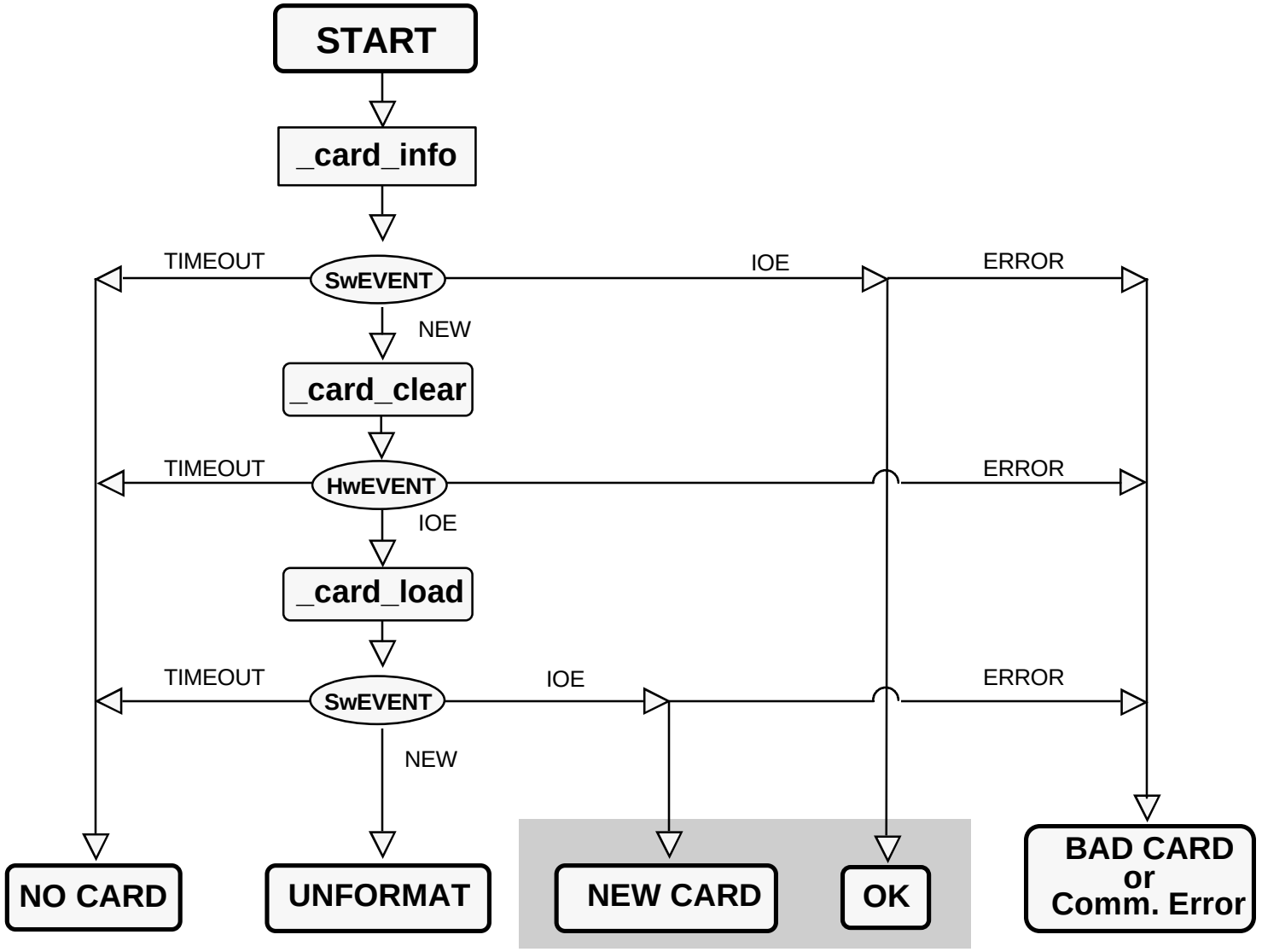
- read() and write() terminate immediately after the I/O request is registered with the driver.

- The completion of I/O is reported through events.

READ/WRITE processing in progress => HwCARD
Processing completed => SwCARD

- Each slot accepts only one I/O request at a time.

Memory Card Status Check



Status Check Example

(Example) sample.c

Checking the status of the specified slot

Format	int CheckCardStatus(int cnct, int slot)		
Parameters	cnct	Connector No. (0 or 1)	
	slot	Slot No. (normally 0; multitap: 0 to 3)	
Return values	CARDSTS_OK	1	Normal
	CARDSTS_NEW	2	New card
	CARDSTS_NONE	-1	No card inserted
	CARDSTS_UNFMT	-2	Card not formatted
	CARDSTS_ERR	-3	Unknown error[

Note 1: Even if the return value is "0" (normal), a format check is performed.
=> Processing is slow.
Note 2: The result of _card_clear() is used to monitor for HwCARD events.

Distributed Processing Check

(Example) card.c

Distributes the memory card status check processing within the main loop. =>Reduces the load in the main loop.

Format: int CheckCardStatus(int cnct, int slot)

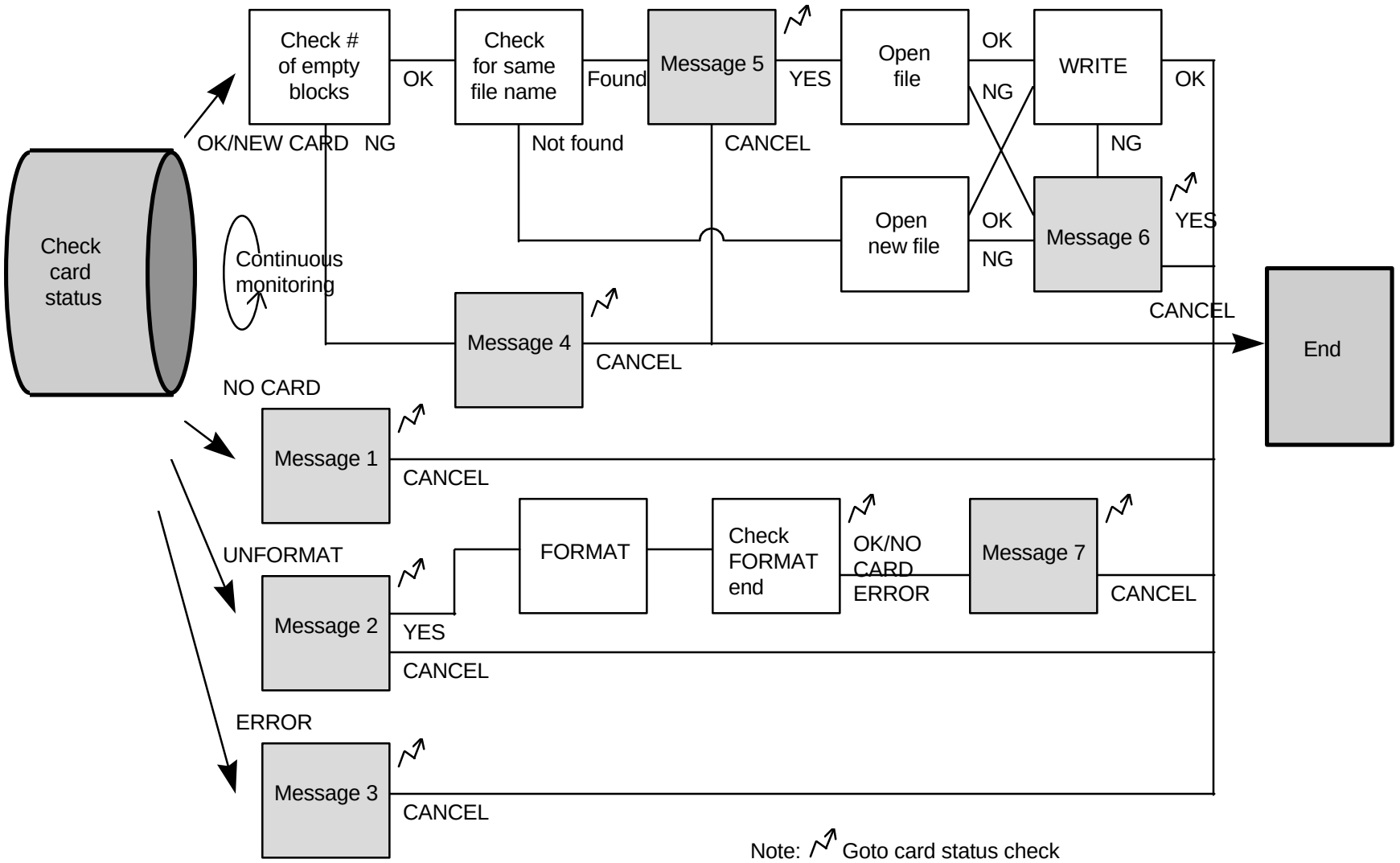
Parameters:

cnct		Connector No. (0 or 1)
slot		Slot No. (normally 0; multitap: 0 to 3)

Return values:

CARDSYS_BUSY	0	Status check in progress
CARDSTS_OK	1	Normal
CARDSTS_NEW	2	New card
CARDSTS_NONE	-1	No card inserted
CARDSTS_UNFMT	-2	Card not formatted
CARDSTS_ERR	-3	Unknown error

SAVE Program flow



Messages Displayed by SAVE Program (Examples)

Message 1: No memory card found.
Insert a memory card in slot 1. (Return)

Message 2: Memory card not formatted.
Format card? (OK/Return))

Message 3: Memory card damaged.
Insert a different memory card. (Return)

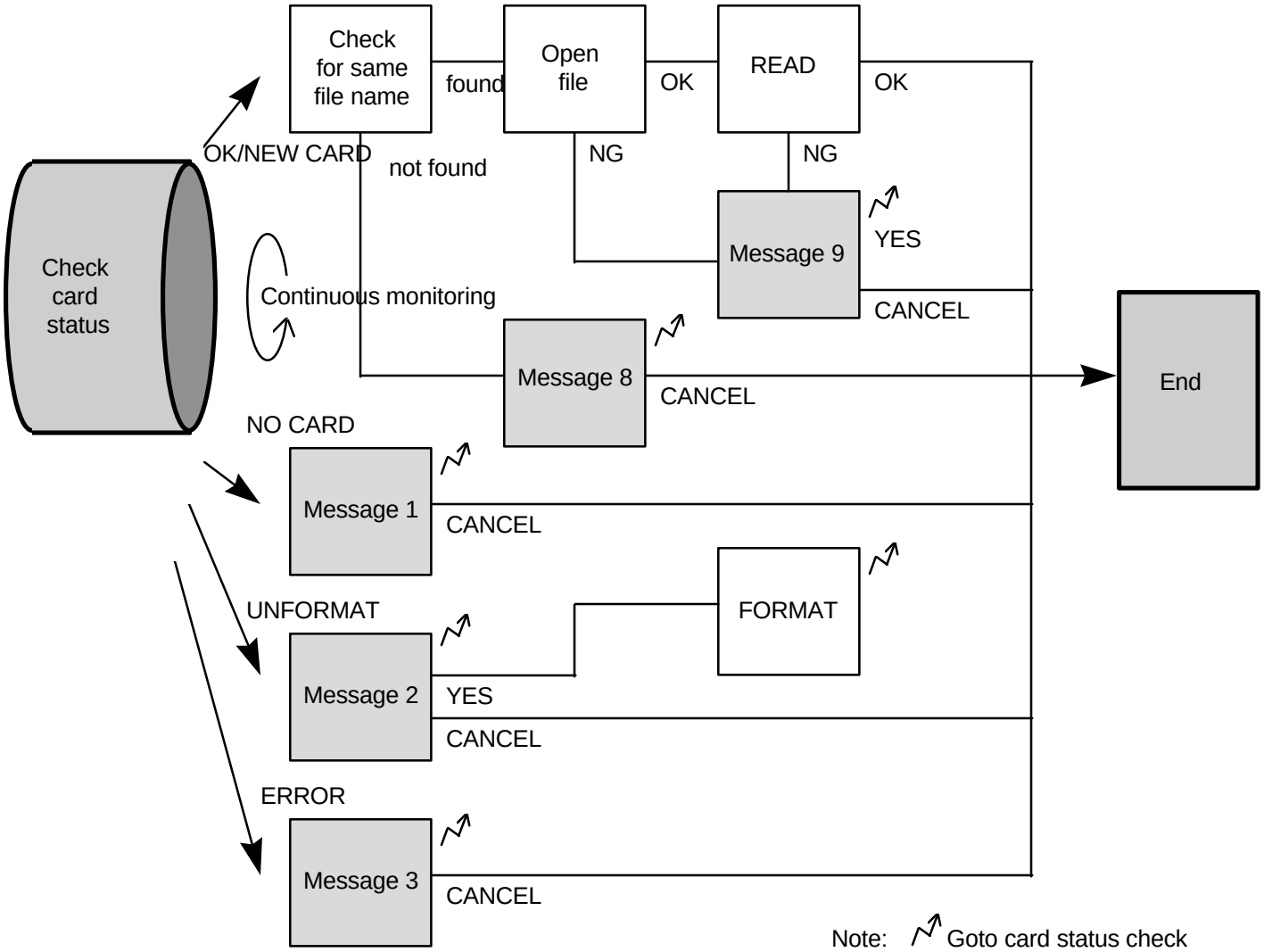
Message 4: Not enough available blocks.
Either insert a different card or delete unnecessary data using the memory card control screen. (OK/Return)

Message 5: File with the same name already exists.
Overwrite? (OK/Return)

Message 6: An error occurred while writing the data.
Save again? (Yes/Return)

Message 7: Formatting failed. (Return)

LOAD Program Flow



Messages Displayed by LOAD Program (Examples)

Message 8: File not found on memory card.
Insert the proper memory card (in slot 1). (Return)

Message 9: An error occurred while reading the data.
Load again? (Yes/Return)

Note: Refer to the messages displayed by the SAVE program for messages 1 to 3.

Getting Available Blocks

(Example) sample.c

```
/* Gets all directory entries for the specified slot */
FILE_NUM = GetCardEntry(CONECT_NO, SLOT_NO);


/* Gets the available blocks from a directory entry */
AVAIL_BLOCK = GetCardSpace(FILE_NUM);
```

Writing Files

(Example)sample.c

```
/* Writes a file to the specified slot */
ret = WriteCardFile(CONECT_NO, SLOT_NO, FNAME,
                    BUF, BLOCK_NUM);
```

- After opening a new file, immediately issue CLOSE once.
- This function supports both synchronous and asynchronous mode.

- write function return values:

Asynchronous mode	=>	Normally 0 Write end -> SwCARD event
Synchronous mode	=>	Number of bytes written

Reading Files

(Example)sample.c

```
/* Reads a file from the specified slot */
ret = WriteCardFile(CONECT_NO, SLOT_NO, FNAME, BUF,
                    BLOCK_NUM);
```

- This function supports both synchronous and asynchronous mode.
- read function return values:

Asynchronous mode	=>	Normally 0 Read end => SwCARD event
Synchronous mode	=>	Number of bytes read

Function Examples (sample.c)

```
void InitMemCard(long val);              /* Initializes memory card control module*/
void StopMemCard(void);                  /* Terminates memory card control module */
int _card_event(void);                   /* Checks SwCARD event (BLOCK) */
int _card_event_NW(void);                /* Checks SwCARD event (NON BLOCK)*/
void _clear_event(void);                 /* Clears SwCARD event */
int _card_event_x(void);                 /* Checks HwCARD event */
void _clear_event_x(void);               /* Clears HwCARD event */
int GetCardEntry(int cnct, int slot);
                                         /* Gets all directory entries from specified slot */
int GetCardSpace(int file_num);
                                         /* Gets available blocks from directory entry */
int CheckFileExist(int filenum, char *fname);  /* Checks if same file name exists */
int ReadCardFile(int cnct, int slot, char *fname, char *buf, long block);
                                         /* Reads file from specified slot*/
int WriteCardFile(int cnct,int slot,char *fname,char *buf, long block,int new);
                                         /* Writes file to specified slot*/
int CheckCardStatus(int cnct, int slot);          /* Checks memory card status */
int FormatCard(int cnct, int slot);      /* Formats card in specified slot */
```

File Names

Bytes	Contents	Remarks
0	Magic	Always B
1	Region	Japan: "I"; North America: "A"; Europe: "E" (*1)
2-11	Title	SCE product number (*2)
12-20	Available to user	Use any ASCII characters, except for "0x00"; end with "0x00"

*1: Not checked by system.
*2: In the case of multi-disc titles, the title of the first disc is used.
Example: If the product code is "SLPS-00001", the first twelve characters of the file name are "BISLPS-00001". (The numeric portion is always padded out to five digits with zeroes ("0").)

File Headers

Placed at the start of the data area

Item	Size (bytes)
Header	128
Magic	2 (always "SC")
Type	1
Number of blocks	1
Document names	64 (Shift-JIS, *1)
pad	28 (0 padding)
CLUT	32
Icon image(1)	128 (16 x 16 x 4 bits)
Icon image(2)	128 (Type == 0x12, 0x13 only)
Icon image(3)	128 (Type == 0x13 only)

*1: 32 full-size characters; non-kanji and first level kanji only. If less than 32 full-size characters, terminate with the null character.

File Headers

Type	Number of icon images (auto substitution animation)
0x11	1
0x12	2
0x13	3

* Icon data for the memory card control screen.
  The number of icon images can range from 1 to 3 patterns.

Overview of Sample Program

- Implements asynchronous READ/WRITE
- Senses the card state in real time

Overview of Sample Program

```
ResetCallback();
Initialize pad;
Initialize memory card;
ChangeClearPAD(0);
Initiailize graphics;
Initialize primitives
main() {
        Input from pad;
        Process graphics drawing;
        Check memory card status;
        Execute memory card processing
        (SAVE/LOAD/FORMAT);
        Sense end of memory card processing
(SAVE/LOAD);
        Display memory card status;

}
Stop memory card;
Stop pad;
end
```

Structure of Sample Program

Revision of \psx\sample\graphics\balls

File organization

main.c	Added function group for asynchronous access of memory card
card.c	Memory card-related function group
card.h card.c	#define group referenced by "card.c"
cardicon.h	Icon data for the memory card control screen
balltex.h	Texture data for balls

<makefile.mak>

```
all:
        ccpsx -G 0 -g -Xo$80010000 main.c card.c -omain.cpe,main.sym
```

Programming Notes

File names for multi-disc titles

In the case of a multi-disc title or a limited edition, it is not necessary to assign a different product number for each disc.

Ex.: Product numbers for a three-disc title

	OK	NG
SLPS-11111	BISLPS-11111~	BISLPS-11111~
SLPS-11112	BISLPS-11111~	BISLPS-11112 ~
SLPS-11113	BISLPS-11111~	BISLPS-11113 ~

Note: When creating a master disc with a CD-ROM generator, input the corresponding product numbers for the PVD volume descriptors and the disc name in the master information.

Programming Notes

Document names

Always use Shift-JIS codes (up to a maximum of 32 full-size characters) for the document names registered in the data header field for the memory card.

Do not use ASCII only or a mixture of Shift-JIS and ASCII codes.

If the document name is less than 32 characters long, end the name with the null code (0x00).

Always use zeroes for padding directly after the document name.

Pay careful attention to these points when development work is conducted overseas!

For the ASCII => Shift-JIS conversion tool, see the library
"CD-ROM\psx\kanji\asc2sjis".

Programming Notes

Format function return value

Because the return value of the format function is always "1", the return value can not be used to determine whether the formatting operation was
completed successfully or not. The result of the formatting operation is determined by the HwCARD event value.

Programming Notes

Event values immediately after _card_info() is issued

In some cases, when noise is generated during serial communications or when the load is high due to background processing, such as graphics drawing,
CD-ROM accesses, or sound playback, the event value may return an error immediately after _card_info() is issued.

Issue a retry in the case of a memory card status check that includes _card_info() or _card_load().

Programming Notes

Incorrect determination of unformatted card

When it is unclear whether or not a memory card was inserted or removed beforeSAVE/LOAD processing, or if an unformatted memory card was already inserted,
it is impossible to open a file normally, even if the event value was "IOE" after_card_info() was issued.

In this situation, always execute the unformatted check
_card_load()
after _card_info().

Reference: Creating an unformatted memory card

Execute \psx\sample\etc\cman\cman and select "UNFORMAT".

Programming Notes

Pad input

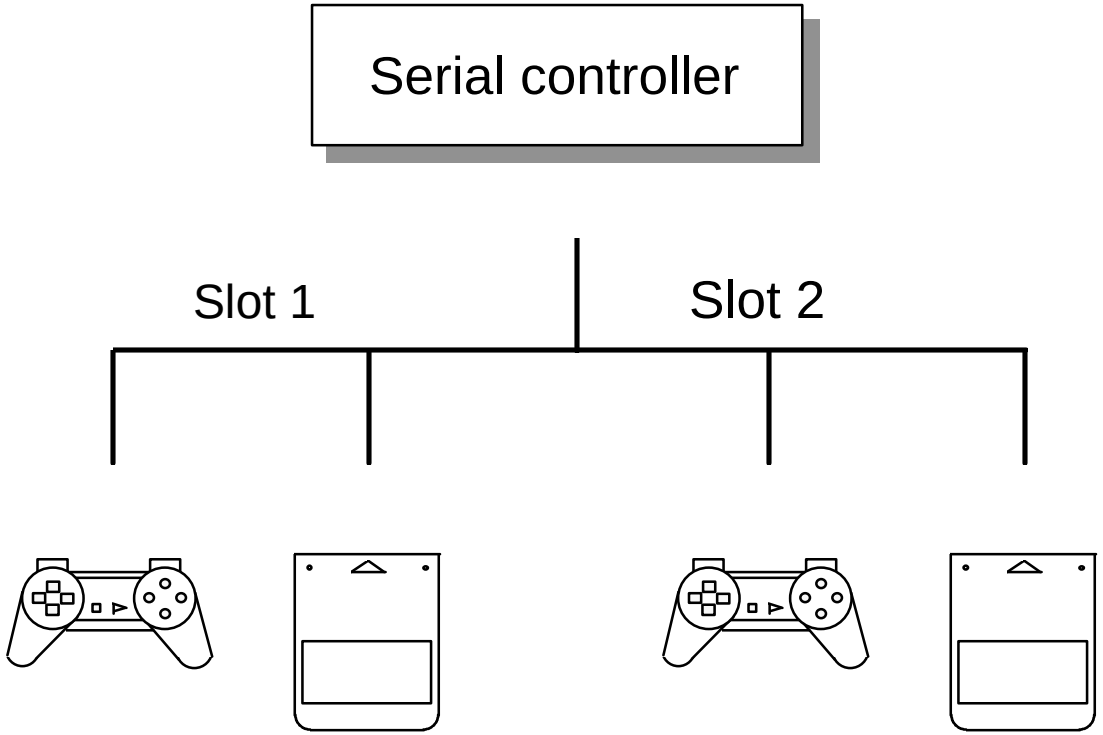
When pad initialization is performed by means of InitPAD(), pad input is enabled by executing StartCARD(), even if StartPAD() wasn't executed. Pad input is stopped by executing StopCARD().

Programming Notes

Determining Free Space

When determining how much free space a memory card has, use the total number of blocks used by all of the files, not the total number of files stored in the memory card, as the basis for determining the free space. (Some data uses only 15 blocks/file.) Assume 15 as the maximum number of blocks per slot.

Controller and Memory Cards



Communications failure with a memory card

Communications failure with the controller

Notes on Methods

Memory Card Formatting (Initialization)

Be sure to require key input to confirm the player's intentions before formatting a memory card.

When providing an original screen within a game for the purpose of controlling functions such as copying or erasing memory cards individually, do not provide a separate "format" button or command. (Do not create an environment that allows the player to format a memory card at his discretion.)

If a memory card is found to be unformatted when an attempt is made to save data, then give the player the ability to perform a format at that point.

Notes on Methods

Continuation of game regardless of whether a memory card is present

Even if a memory card is required in order to start a game, the design should allow the game to proceed (with the player's permission) without a memory card.

Allow saves via a Save screen while the game is in progress if a memory card is inserted in a slot.

The design of the game must never be such that the game cannot be played at all if there is no memory card in a slot.

Notes on Methods

Memory card control screen display on the PlayStation

There must always be at least one icon data pattern stored in the save data portion of the header in order to display an icon on the memory card management screen display on the PlayStation. Either 1, 2, or 3 patterns may be stored.

Notes on Methods

When creating an original memory card control screen

The control screen should be capable of displaying the icons and document names for other titles correctly.

Do not provide a separate "format" button or command for initializing memory cards.

Note: The screen should be able to display all ASCII code characters, a mixture of ASCII code and Shift-JIS code characters, and all Shift-JIS code characters when displaying document names from other titles. Also limit the maximum number of characters that can be displayed to 32 (whether full-size or half-size).

Notes on Methods

Variable length files

Make every effort to stop the practice of allowing a variable number of blocks in files that are being saved. We recommend establishing a fixed number of blocks required for a save. With this type of approach, it is essential to make adjustments for empty blocks, so be certain to set up a memory card control screen (a screen on which data can be copied or deleted).

Notes on Methods

When supporting both memory card slots 1 and 2

Allow the user to select one memory card slot or another when saving or loading data. Do not have the application automatically access the slots in sequence (slot 1 and then slot 2).

Note: Supporting slot 1 only is also acceptable.

Notes on Methods

Miscellaneous

We have distributed a pamphlet entitled, "Notes on Creating Master Discs;" refer to this pamphlet for further information. This pamphlet is also available on SCE-NET.

When delivering a master CD-ROM, it should be accompanied by a "Master Contents Confirmation" form. Carefully check the items noted on this form before submitting it.

Sample Program Using Graphics, CD-ROM, and Sound

Special Features of the Sample Program

• Memory card is accessed while simultaneously in the background, graphics are being drawn, files are being read from the CD-ROM, and sound (SEQ) playback is being performed. (The memory card processing is block-based.)

=> The drawing of graphics is not interrupted even when block-type functions such as format() are being executed.

• Error processing in memory card accesses when other processing loads are high.

• Memory card processing that is part of the user interface.
Menu format, messages, etc.