PlayStation™ Hardware

Developer

Reference

Series

September 1995

Foreword

This document gives an overview of the hardware structure and all sections of the PlayStation system. For information about boards and development host connection, etc, please refer to the manual accompanying the board

OS Hardware Guide

CD-ROM FILE STRUCTURE FORMAT
The PlayStation's CD-ROM logic file format conforms to the following specifications:

ISO-9660
Information processing - Volume and file structure of CD-ROM for information interchange.
(1988-09-01)

CD-ROM XA
SYSTEM DESCRIPTION CD-ROM XA
Copyright May 1991

Published December 1994.

Sony Electronic Publishing Ltd
13 Great Marlborough Street
London
W1V 2LP
Tel No: +44 (0) 171 911 8900 (switchboard)

Contents

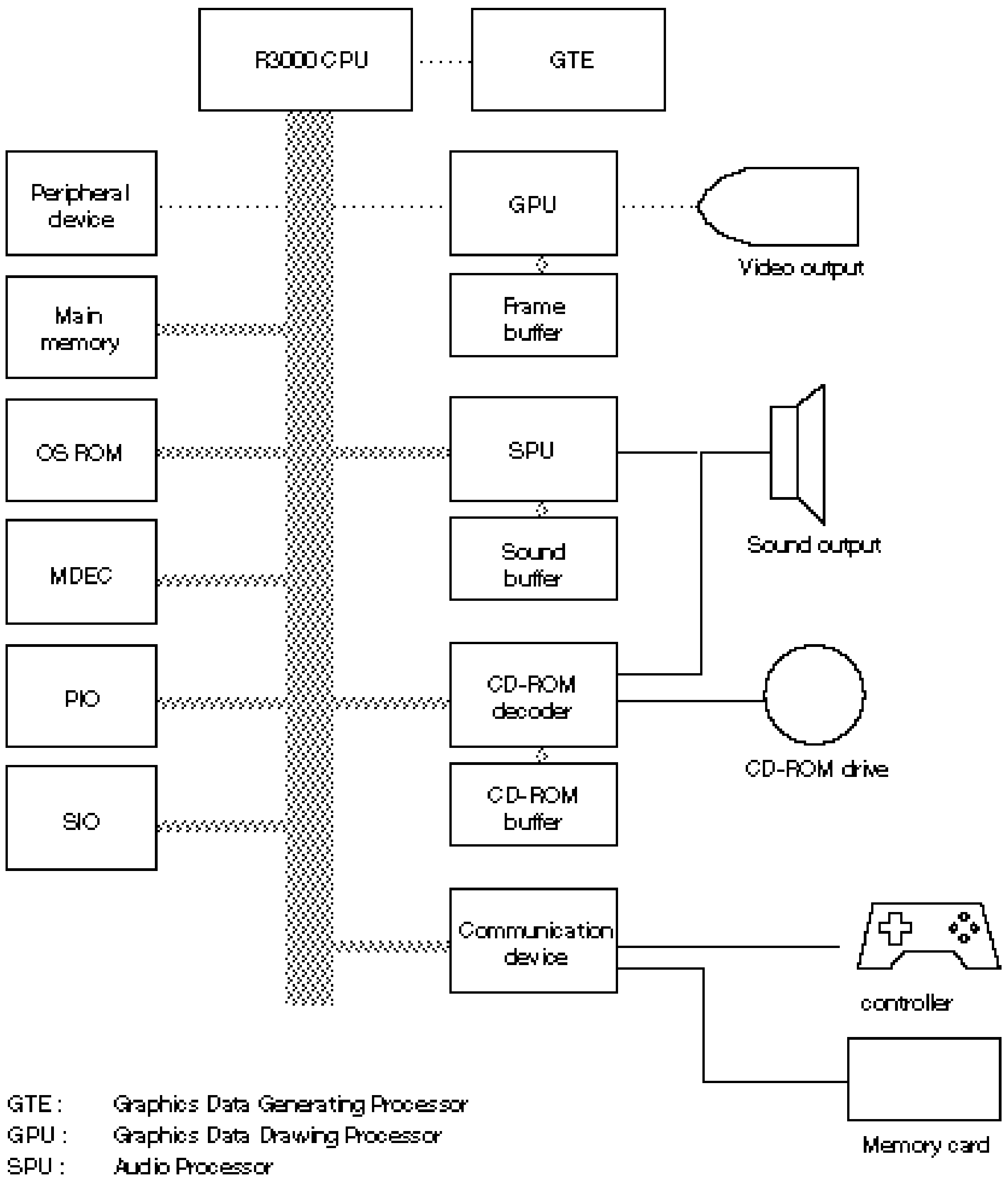
chapter 1

System Hardware Summary

This is an explanation of the hardware composition of the “PlayStation” system and an overview of each component.

System Architecture

Playstation is made up of processors and device groups which implement all functions, such as graphics and sound, centred on a 32 bit RISC CPU as in the figure below. In this document, each block is explained based on the figure below.



GTE :	Graphics Data Generating Processor
GPU :	Graphics Data Drawing Processor
SPU :	Audio Processor
MDEC :	Motion decoder
PIO :	Extended parallel port
SIO :	Extended serial port

Fig. 1.1 PlayStation block diagram

CPU and its Peripherals

This is the basic part of the system, composed of a memory and an interrupt controller, with a 32-bit RISC CPU as its core. The CPU mounts an "I" (instruction) cache and a scratchpad memory, and executes management of the actual memory.

Graphics System

The "PlayStation" graphics system is composed of a graphics data creation processor (GTE) and a graphics drawing processor (GPU).

The GTE executes coordinate transformation and light source calculation as a coprocessor to the CPU, for example fixed-point format matrix and vector operations, at high speed by a parallel processing mechanism.

The GPU operates according to polygon drawing instructions from the CPU. Also, the CPU holds a non-shared 2-dimensional address space, and the frame buffer is mapped in this space.

The basic principle of the "PlayStation" graphics system is the transmission of texture images and colour palettes ("CLUT") from the CPU to the frame buffer, and the causing of the GPU to draw polygons, based on coordinates and colour information obtained by the GTE.

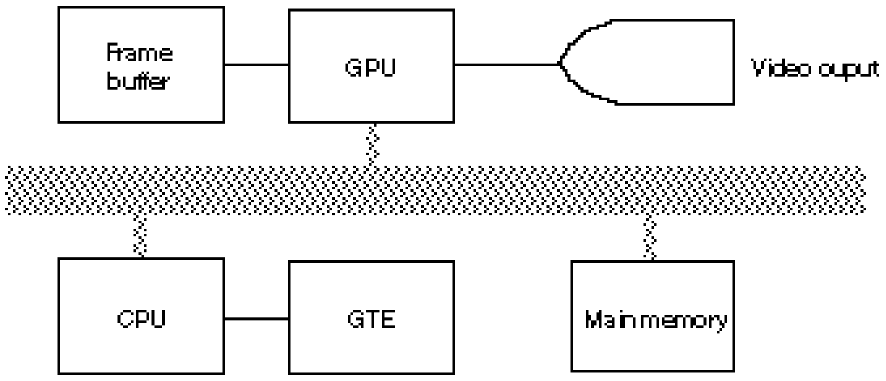


Fig. 1.2 Graphics System

Sound System

The “PlayStation” sound system is composed of a sound reproduction processor (SPU) and a CD-ROM decoder.

The SPU houses an ADPCM 24-voice sound source which has functions such as automatic alteration of operating parameters taking looping and time as coefficients. It is operated by the CPU. The SPU manages its own address space in which a sound buffer is mapped. It transmits ADPCM data to the sound buffer from the CPU, and reproduces data by the direct delivery of “Key On/Key Off” and modulation information.

System Architecture (cont.)

The CD-ROM decoder reproduces audio while reading PCM or CD-ROM XA ADPCM data on a disk. The decoder audio output temporarily enters the SPU, is mixed with the SPU output, and becomes the final audio output via the reverb unit.

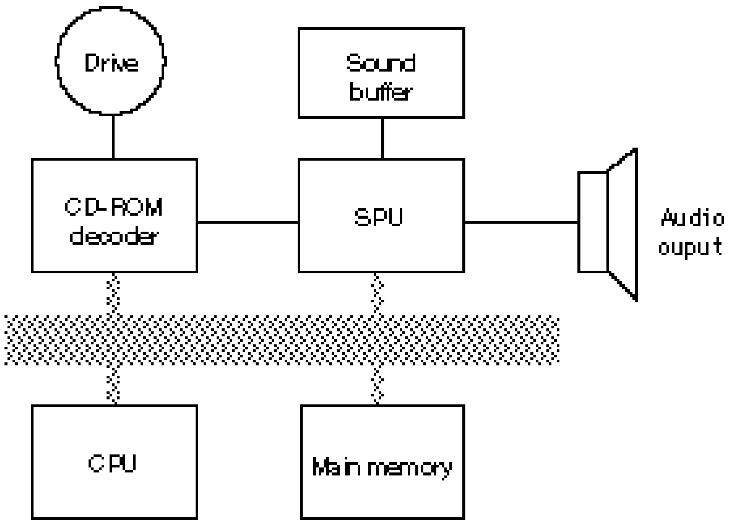


Fig. 1.3 Sound System

CD-ROM System

This is composed of the necessary components for reading CD-ROM, such as a drive and a decoder. It supports CD-DA, CD-ROM XA and “PlayStation” disk formats.

Data Expansion Engine

This is an engine which executes reverse DCT transformation at high speed. It executes expansion of JPEG and MPEG (compressed in frame only) data.

Controller

This is an interface which transmits a player’s intentions to the applications. Apart from the two connectors on the mainframe, it is possible to connect a large number of controllers by using a “multitap” as well.

Memory Card

This is a device for writing data which it is desired to save after “reset” or “power OFF”. Apart from the two connectors on the mainframe, it is possible to connect a large number of cards by using “multitap” as well.

Expansion Ports

Two type of expansion port are provided, serial and parallel.

CPU and its Peripherals

This is an explanation of the CPU which forms the central part of the system and its peripheral devices.

2.1 Memory

There is 2 MB of main memory installed in the PlayStation.

However, a “TLB”, which is the virtual memory management device for the R3000 CPU, is not mounted. Therefore, the relationship between the physical address and the logic address of the memory space is always fixed.

Physical Memory Map

The “PlayStation” physical memory map is as follows.

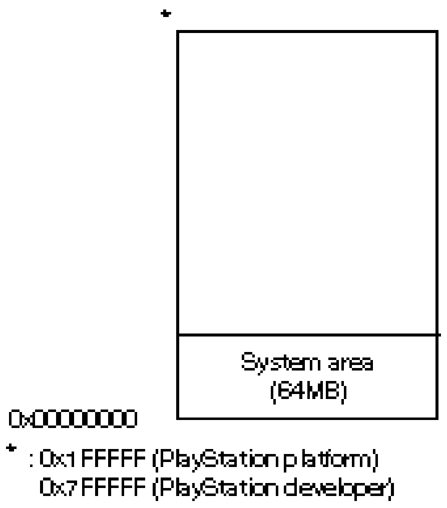

*
System area
(64MB)
0x00000000
* : 0x1FFFFF (PlayStation platform)
0x7FFFFF (PlayStation developer)

Fig. 2.1 Physical Memory Map

OS ROM

A 512KB OS ROM is mounted in the "PlayStation". "OS kernel" and "boot loader" are stored in this ROM.

Access to ROM is prohibited and addresses are not disclosed.

CPU

The PlayStation uses a customized CPU based on the R3000A, which is a 32 bit RISC CPU. The specifications are as below.

The D cache of the PlayStation is called a “scratch pad.” Fixed address areas may be accessed with the same block number as cache memory .

Table 2.1 CPU specifications

Item	Specification
System Clock	33.8688MHz
Bus Width	32 bit
I cache	4KB
Dcache	1KB (Scratch Pad)
Endian	Little

Memory Management

Memory Map

The space handled by the program is called “logic space”. “Logic space” is expressed by a 32-bit address, and is mapped in multiplex in physical space. With the “C” language, it is accessed by using a “long” pointer.

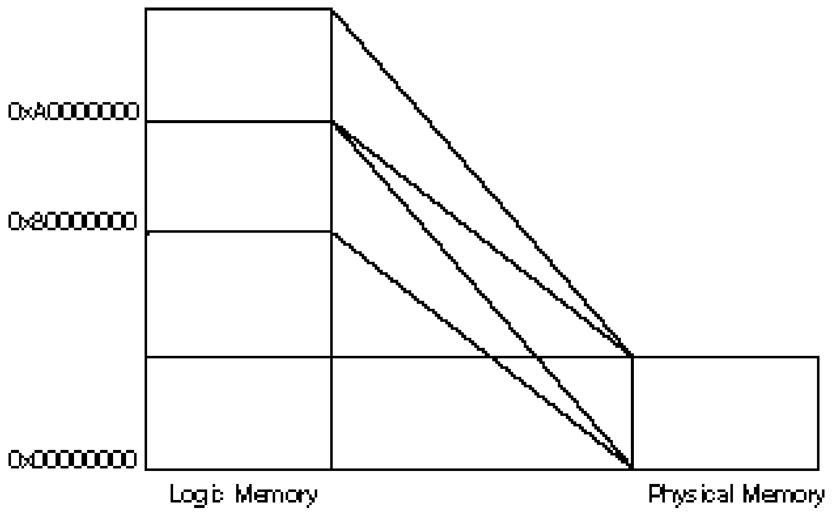


Fig. 2.1 Logic Memory/Physical Memory allocation

“Physical space” is an address in the packaged memory device. When accessing a logic space address which is not mapped in physical space, a “bus error exception” is generated.

Table 2.2 Memory maps

Logic Space (32 bits)	Segment Title	Cache
0x00000000-MAX(0x1FFFFF)	A	Effective
0x1F800000-0x1F8003FF	S	Not effective
0x1F801000-0x1FBFFFFF	X	Not effective
0x1FC00000-0x1FC7FFFF	P	Not effective
0x80000000-0x80000000+MAX	B	Effective
0x9FC00000-0x9FC7FFFF	Q	Effective
0xA0000000-0xA0000000+MAX	C	Not effective
0xBFC00000-0xBFC7FFFF	R	Not effective

Logic Space	Corresponding Logic Segment	Device
0x000000~MAX(0x1FFFFF)	A + B + C	RAM
0x1F800000~0x1F8003FF	S	Scratchpad
0x1F801000~0x1FBFFFFF	X	Device
0x1FC00000~0x1FC7FFFF	P + Q + R	ROM

I-Cache

“Effective/not effective” is determined for the instruction cache (hereafter, “I-cache”) by the top 4-bits of logic space. Out of the segments which correspond to the above-mentioned memory device, I-cache is effective with A, B, P and Q, but is not effective with C and R. A “segment” is a “block” unit which is divided from the functional plane of the memory space in the R3000. Note that this differs from those of other CPUs (8086 series)

When I-cache is effective, instruction codes are read into I-cache by collating them in specific units. If the target instruction is present in I-cache during execution, memory access via the bus is not necessary. By this means, application execution speed is increased.

I-cache is packaged in 1K words (4K bytes), and is 1-way mapped. In other words, logic space is divided into 4K-byte units, and these are mapped in multiplex on the I-cache.

D-Cache

The D-cache is composed of a high-speed memory housed in the CPU. This has a scratchpad structure, and is mapped on the memory space as an “S” segment so that a “game programmer” can access it.

Both data and programs can be stored in this segment. However, it is not a subject for “DMA” transmission.

Address Alignment

R3000 requires strict address alignment.

This must start from an address which is a multiple of 4 for word-length (4 bytes) data, and a multiple of 2 for half word-length (2 bytes) data. Memory accesses (“data store”, “data load”, “instruction fetch”) which do not comply with this rule will cause bus errors.

2.6

Registers

The R3000 registers can be broadly divided into “general-purpose registers” and “program counters”.

General-Purpose Registers

There are 32 32-bit general-purpose registers. Each register is assigned to a specific use by the compiler, as shown below. These registers must be operated in accordance with this assignment when in thread database operation or when developing in assembler. The macros in the table are defined in “asm.h”.

Table 2.3 R3000 General Purpose Registers

Macro (1)	Macro (2)	Assignment
R_ZERO	R_R0	0 fixed
R_AT	R_R1	Assembler reserves
R_V0	R_R2	Return value
R_V1	R_R3	Return value (for double type)
R_A0	R_R4	Argument #1
R_A1	R_R5	Argument #2
R_A2	R_R6	Argument #3
R_A3	R_R7	Argument #4
R_T0	R_R8	destroyed in function (10)
R_T1	R_R9	〃 〃
R_T2	R_R10	〃 〃
R_T3	R_R11	〃 〃
R_T4	R_R12	〃 〃
R_T5	R_R13	〃 〃
R_T6	R_R14	〃 〃
R_T7	R_R15	〃 〃
R_S0	R_R16	Saved in function (8)
R_S1	R_R17	〃 〃
R_S2	R_R18	〃 〃
R_S3	R_R19	〃 〃
R_S4	R_R20	〃 〃
R_S5	R_R21	〃 〃
R_S6	R_R22	〃 〃
R_S7	R_R23	〃 〃
R_T8	R_R24	〃 〃
R_T9	R_R25	〃 〃
R_K0	R_R26	Kernel reserved #0
R_K1	R_R27	Kernel reserved #1
R_GP	R_R28	Global Pointer
R_SP	R_R29	Stack Pointer
R_FP	R_R30	Frame pointer
R_RA	R_R31	Return address

Items to be noted when using general-purpose registers and assignment contents are as shown below.

AT Register

The AT register is used as a work area by the assembler. “C compiler” and “assembler programmer” are not permitted to use this register.

Return Address

There is no concept of "sub-routine call" in R3000. Therefore the return address is substituted by a jump instruction which is held in a register. This register can be designated in the assembler, but in the compiler it is limited to the RA register.

Arguments of the "C Language" Function

When there are 4 or less arguments, they are stored, from the left, in the registers A0 to A3. When there are 5 or more arguments, they are stored on the "stack". In this case, the space ensured on the stack relating to the first 4 arguments is dummy, and their contents are delivered via the register.

Return Values for "C Language" Functions

When the return value of a function is less than 32 bits, it is stored in the V0 register. When the value is 64 bits ("double" type), the top 32 bits are stored in the V1 register.

Stack

There is no "stack" concept in R3000. Therefore, the compiler achieves a "stack" by storing a pointer to the SP register. Also, in order for "function frames" (memory areas for automatic variables and work areas) to operate efficiently, the head address of the frame for that function is stored in the FP register. This value is determined based on the SP. During module activation, the same value is set in the FP as in the SP.

Global Pointer

R3000 memory access is executed by "register indirect mode with coded 16-bit offset". In order to operate this effectively, the compiler collates variables up to 64K bytes in a block called "bss section". Here, the centre address of the block is stored in the GP register, and access is executed by a 1-word instruction using the above mode. This address value is called a "global pointer" and is not variable in the module.

Program Counters

Program counters cannot be directly accessed. The operation of program counters is as follows.

Immediately After "Power-On Reset"

The content of the program counter is 0xbfc00000.

External Interrupt and Exception Generation (except for "Debug Exceptions")

1) The content of the program counter is stored in the EPC register of co-processor 0 (the part in R3000 which manages exceptions and interrupts).
2) Then it jumps to 0x00000080.

Exceptions

If an error, such as a bus error or a reserved instruction execution, is detected, an exception will be generated. An exception triggers a jump to interrupt address 0x00000080 in the same way as an interrupt from a device. However, it differs from an interrupt from a device in that it cannot be masked. Exceptions are classified as follows according to the errors which cause them.

Table 2.4 Exception categories

Code	Content
AdEL	Address error (during data loading or instruction fetching)
AdES	Address error (during data storage)
IBE	Bus error (during instruction fetching)
DBE	Bus error (during data loading or data storing)
Sys	System call
Bp	Breakpoint

Graphics System

In the PlayStation there is the GPU, the graphics drawing processor, and the GTE, geometry transform engine. The GPU is equipped with a CRTC function for screen display and a polygon and sprite high speed drawing function for the frame buffer. The GTE carries out high speed calculation of coordinate data and light source data as the CPU co-processor.

Frame Buffer

The GPU has 1 MB of video memory (VRAM).

The frame buffer is dual ported, and may be accessed for drawing while display takes place. High speed DMA transfer may also be performed between the frame buffer and main memory.

CRT Display

The GPU displays the content of rectangular areas in the frame buffer on the CRT display. These areas are called “display areas.” The relationship between rectangular areas and CRT screen displays is as follows.

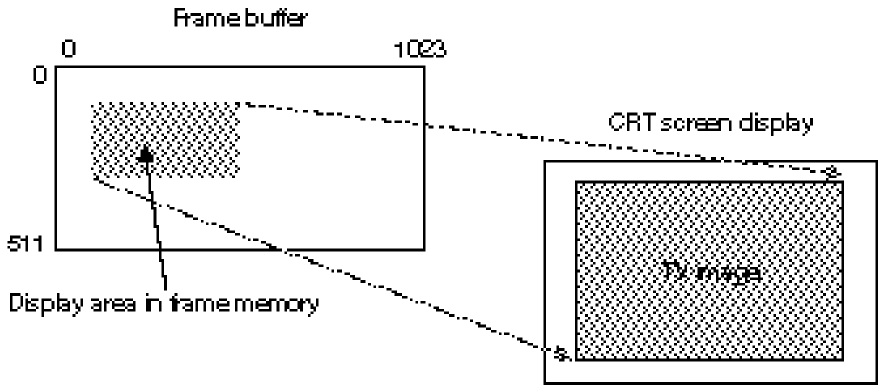


Fig. 3.1 Display areas

Screen mode and display location

The GPU supports the following screen modes.

Table 3.1 Window Resolutions

Mode	Standard Resolution	Remarks
Mode 0 Mode 1 Mode 2 Mode 3	256(H) x 240(V) 320 x 240 512 x 240 640 x 240	non-interlace
Mode 4 Mode 5 Mode 6 Mode 7	256 x 480 320 x 480 512 x 480 640 x 480	interlace
Mode 8	384 x 240	non-interlace
Mode 9	384 x 480	interlace

CRT Display (cont.)

Screen size, in other words the number of pixels on the CRT screen is variable and as is shown in the diagram below, the display initial location (DTX, DTY) and the display final location (DBX, DBY) may be independently specified for horizontal and vertical directions.

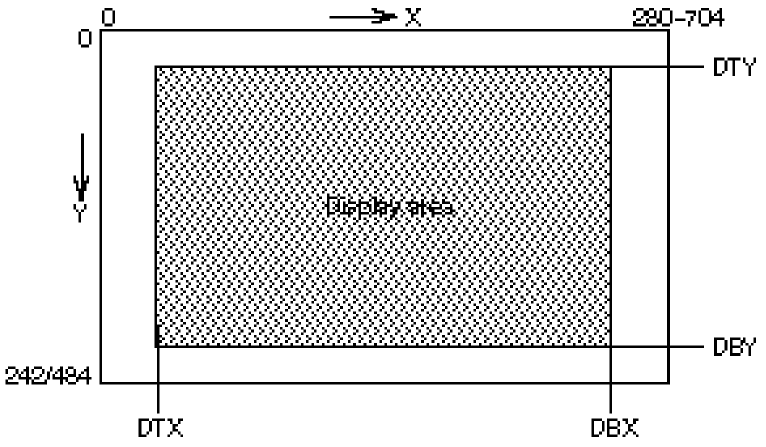


Fig. 3.2 CRT display designation

The relationship between screen mode and the values that can be specified at each coordinate is shown below.

Also, DTX and DBX must be set so that they are multiples of 4. Therefore, the minimum screen size is 4 pixels horizontal, 2 pixels vertical (when non-interlaced), or 4 pixels (when interlaced).

Table 3.2 Possible Designation Ranges for X Coordinate

Mode	DTX	DBX
0, 4	0-276	4-284
1, 5	0-348	4-352
2, 6	0-556	4-560
3, 7	0-700	4-704
8, 9	0-396	4-400

Table 3.3 Possible Designation Ranges for Y Coordinate

Mode	DTY	DBY
0-3, 8	0-241	2-243
4-7, 9	0-480	4-484

Number of display colors

The GPU supports 2 modes for color display, 15 bit Direct (32,768 colors) mode and 24 bit Direct (full color).

15 bit mode

This mode can display up to 32,768 colors.

The number of display colors is limited compared to 24 bit mode, but color computation in the GPU is done in 24 bits. Because of this and the fact that this mode is loaded with a dither feature, quasi-full color (24 bit color) display is possible.

24 bit mode

This mode can display up to 16,777,216 colors (full color).

However, it is only possible to display image data transferred to the frame buffer in 24 bit mode. The GPU cannot execute drawing functions in this mode, because it assumes 16 bits per pixel.

Though the bit length of one pixel is 24 bits, the values of the coordinates and display locations in the frame buffer must be specified as being 16 bit based. In other words, 640 x 480, 24 bit image data is treated as 960 x 480 in the frame buffer.

Also, the previously mentioned DBX must be designated so that it will be a multiple of 8. Thus the minimum screen size in this mode would be horizontal 8 x vertical 2 pixels.

Screen Drawing Capability

GPU has following screen drawing features.

Table 3.4 GPU Drawing Feature

Sprite Drawing	1 x 1 pixel - 256 x 256 pixel 4 bit CLUT (16 colours/Sprite) 8 bit CLUT (256 colours/Sprite) 15 bit DIRECT (32768 colours/Sprite)
Polygon Drawing	Flat shading Gouraud shading Texture mapping
Line Drawing	Gradation possible
Image Transfer	CPU ⟶ Frame buffer Frame buffer ⟶ CPU Frame buffer ⟶ Frame buffer
Other	α Blending (translucency) Dithering Viewport clipping Offset specification

Drawing and coordinate system

The coordinate system for drawing uses a coded 11 bit unit and X and Y take values from -1024 - 1023. Since the size of the frame buffer is 1024 x 512, the over flow portion is returned.

The drawing origin may be freely changed in the frame buffer by setting the coordinate offset value as desired. Drawing may be done only in the desired rectangular area in the frame buffer, because the GPU will clip drawing to the current drawing area.

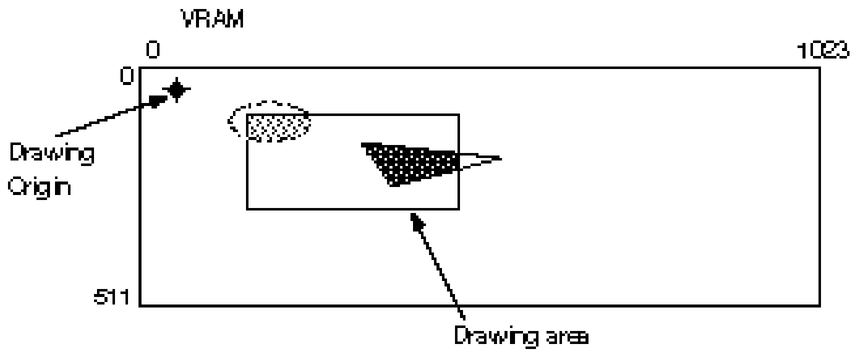


Fig. 3.3 Clipping & Drawing Areas

Sprite drawing

The GPU supports up to 256 x 256 pixel sprites and width and height can vary up to this value.

Sprite & Texture page

The image data associated with sprites is placed in a frame buffer non-display area. Sprite images must be transferred in advance to the frame buffer before executing sprite drawing commands.

Sprite patterns are 256 x 256 pixels and are placed in the frame buffer. The 256 x 256 pixel areas are called texture pages. The size of one texture page in the frame buffer varies with the texture mode.

The position of texture pages in the frame buffer is determined by specifying the page number in the TSB parameter of the drawing command.

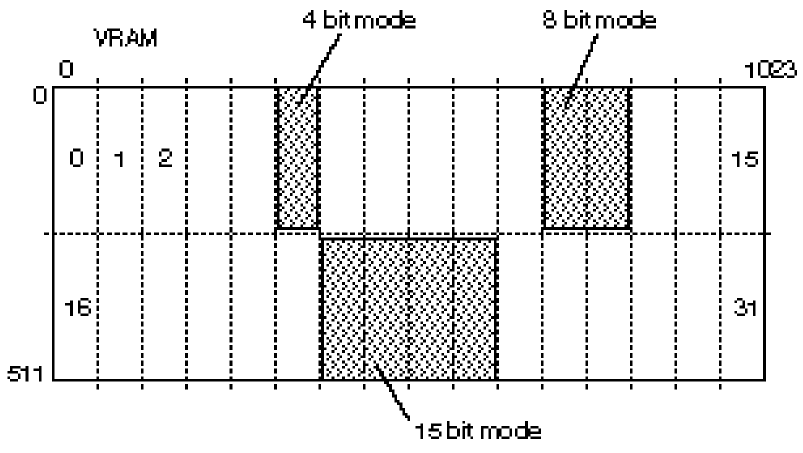


Fig. 3.4 Texture page

Sprite color modes

There are three sprite color modes: 4 bit CLUT, 8 bit CLUT and 15 bit direct. Sprites use a CLUT in 4 bit and 8 bit modes. A CLUT (color lookup table) is a table of RGB values that express the color ultimately displayed. For a 4 bit mode sprite, the table will have 16 sets of RGB components. For an 8 bit mode sprite, the CLUT will have 256 entries. Each RGB value has a number assigned in order from the left (i.e. a table index) in the frame buffer and sprite patterns express the color of each pixel with this number. CLUT may be selected by sprite unit and sprite patterns may have a separate CLUT for all the sprites.

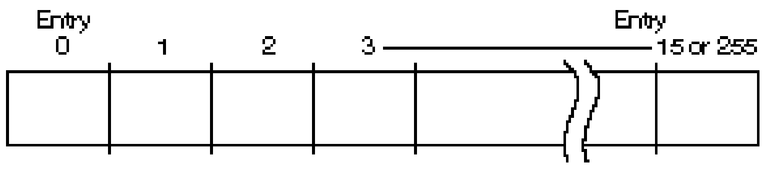


Fig. 3.5 CLUT structure

One entry has the same structure as a 15 bit single pixel.

The CLUT storage location in the frame buffer is determined by specifying the left coordinate of the CLUT being used in the CBA parameter of the drawing command.

Screen Drawing Capability (cont.)

Conceptual diagram of Sprite drawing

Based on the above the concept of sprite drawings would typically be as follows:

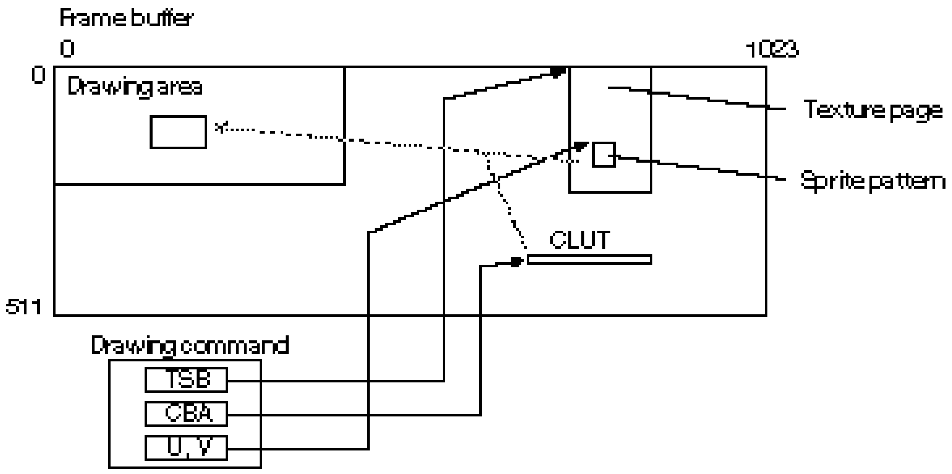


Fig. 3.6 Sprite Drawing Operation

Depiction of polygons

The GPU provides a polygon drawing feature. The polygons handled by GPU are triangular and rectangular. Specify each vertex's screen coordinates to draw.

The GPU has also has the following features.

Gouraud shading

You may specify a different color for each vertex of a polygon, and the GPU will interpolate colors across the polygon.

Texture Mapping

This is a feature which applies 2 dimensional image data (a texture) to a polygon.

As with sprite drawings, texture pattern (elements) are placed in the frame buffer non-display areas. Texture patterns are handled the same way as sprite patterns.

Frame Double Buffering

The GPU uses a method called frame double buffering as a technique of displaying animation.

Frame double buffering uses 2 rectangular areas on the frame buffer. While drawing is being done in one area the other area is displayed and when drawing is completed the 2 areas are swapped. Thus it is possible to avoid drawing to the area which is currently displayed.

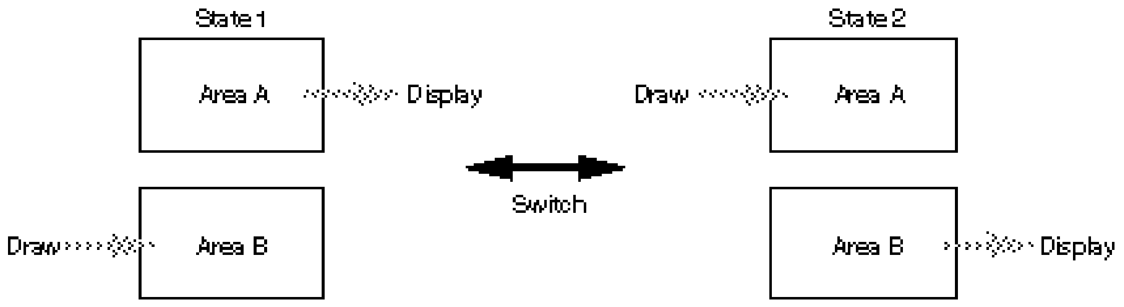


Fig. 3.7 Frame Double Buffering

Buffer switching operations are normally done during the monitor vertical blanking period.

Since the origin and coordinate systems of drawing and display areas may be freely assigned, multiple buffers may be implemented by simply changing the GPU area data.

Sound System

The “PlayStation” sound system is composed of a sound reproduction processor (SPU) and a CD-ROM decoder.

The SPU has a built-in ADPCM 24-voice sound source which has functions such as automatic modification of operating parameters which take pitch transformation and time as coefficients.

The CD-ROM decoder reproduces audio while reading CD-DA or CD-ROM XA data on a disk. The audio output of the decoder temporarily enters the SPU, is mixed with the SPU output, and then becomes the final audio output via a reverb unit.

CD-ROM Decoder

This reads data from a compact disc (CD) and executes sound reproduction or transmission to the main memory.

Sound reproduction is executed based on CD-DA 16-bit PCM data and ADPCM data determined by CD-ROM XA. The data read into the CD-ROM buffer from the drive is processed by the sound source in the CD-ROM decoder after error correction. The audio signal obtained is transmitted to the SPU via a mixer built into the decoder.

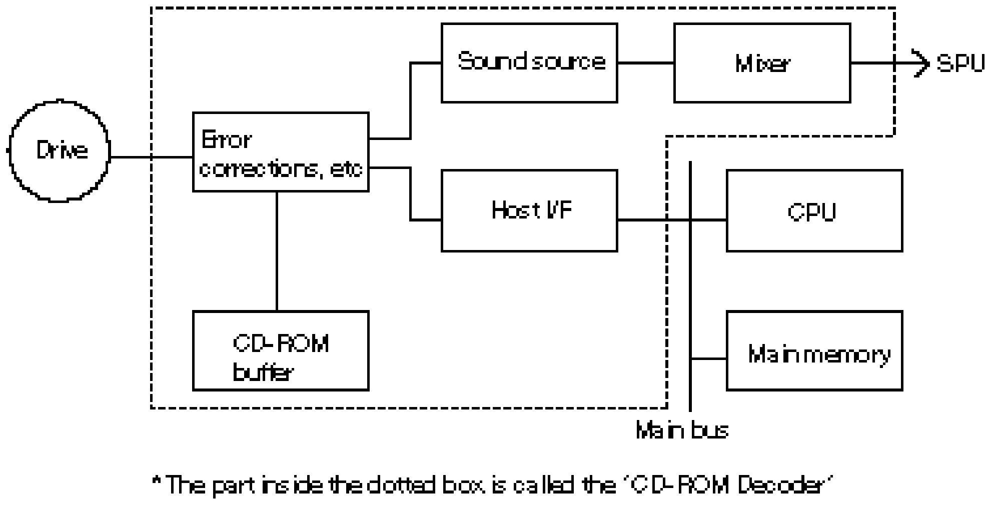


Fig. 4.1 CD-ROM Decoder

The decoder's built-in mixer is composed of 4 attenuators. It executes mixing of the stereo output by varying the attenuation of each attenuator.

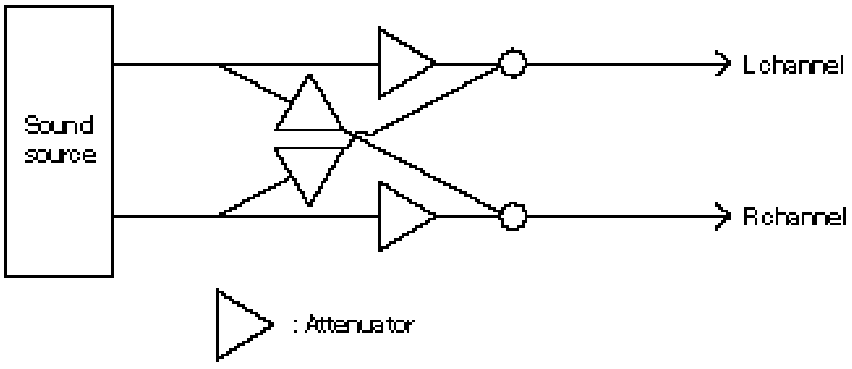


Figure 4.2 CD-ROM Decoder Built-In Mixer

SPU

The sound reproduction processor (SPU) executes reproduction at a sampling frequency of 44.1KHz, taking the ADPCM sample data in local memory as the sound source.

The SPU mounts a digital reverb as an effector, and it is possible to give ample effect through the ADPCM sound data. Furthermore, it houses a mixer which mixes the audio output of the CD-ROM decoder with its own output.

Table 4.1 SPU specifications

Item	Specification
Sound data format	ADPCM
Number of simultaneous sounds (number of voices	24
Sampling frequency	44.1 kHz

Table 4.1 SPU Specifications

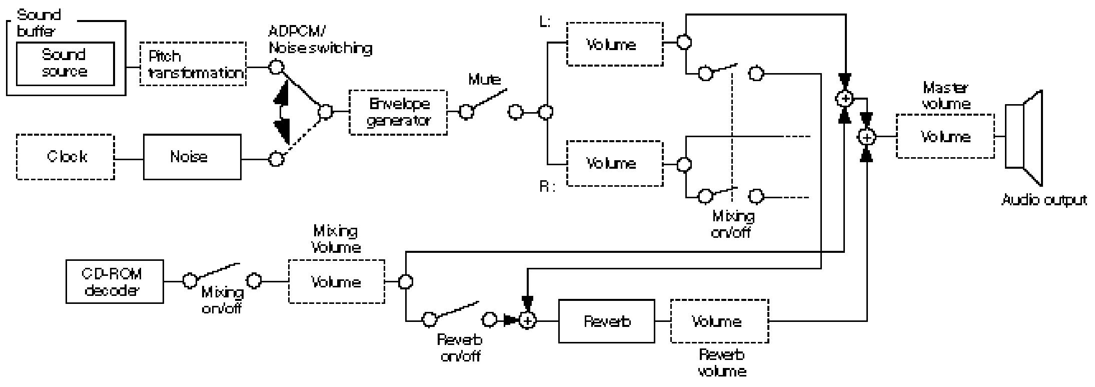


Fig. 4.3 SPU

This indicates a 1-voice relationship for convenience, and ADPCM/Noise switching, L/R volume and reverb ON/OFF can be set individually, voice by voice.

Also, from now on, only the L channel is described.

4.3 Sound Buffer

This is a 512KB local memory which supplies waveform data to the sound source. It is not mapped in the CPU address space.

Sound data encoded by ADPCM which is used by the SPU when generating sound is placed in the sound buffer. The sound buffer is also used as a reverb work area which functions as an effector, or as a temporary buffer when transmitting sound data, created by CD input or the SPU, to the main memory.

DMA transmission from main memory to the sound buffer can be executed. Also, it is not necessary to stop SPU sound generation during transmission.

Voice Functions

24 voices are mounted in the SPU, and the following functions can be set for each voice.

Pitch Transformation

The pitch of the sound data encoded by ADPCM can be varied. The pitch is variable within the range of -12 octaves to +2 octaves, and values of half-tone or less can also be set.

Pitch Variation by Time

Using 2 adjacent voices, the pitch of one voice can be varied with the volume value of the other voice. When this is expressed as an equation, it is as follows

```
    NewPitch(n) = (1 + V(n-1)) Pitch(n)
```

```
NewPitch(n):        the final pitch of voice n
V(n-1):             volume of voice (n-1) (varies by time)
Pitch(n):           pitch originally set for voice n
```

Noise Source

The SPU houses a noise generator which uses pseudo random numbers. This noise source can be designated for any voice instead of the ADPCM sound data in the sound buffer. It is also possible to vary the noise tone by changing the noise generation clock.

Voice Functions (cont.)

Envelope

Any envelope can be set. Separate rates can be set for attack, decay, sustain and release. Linear curves and exponential curves can be designated for the variation with time. It is also possible to set the level for sustain.

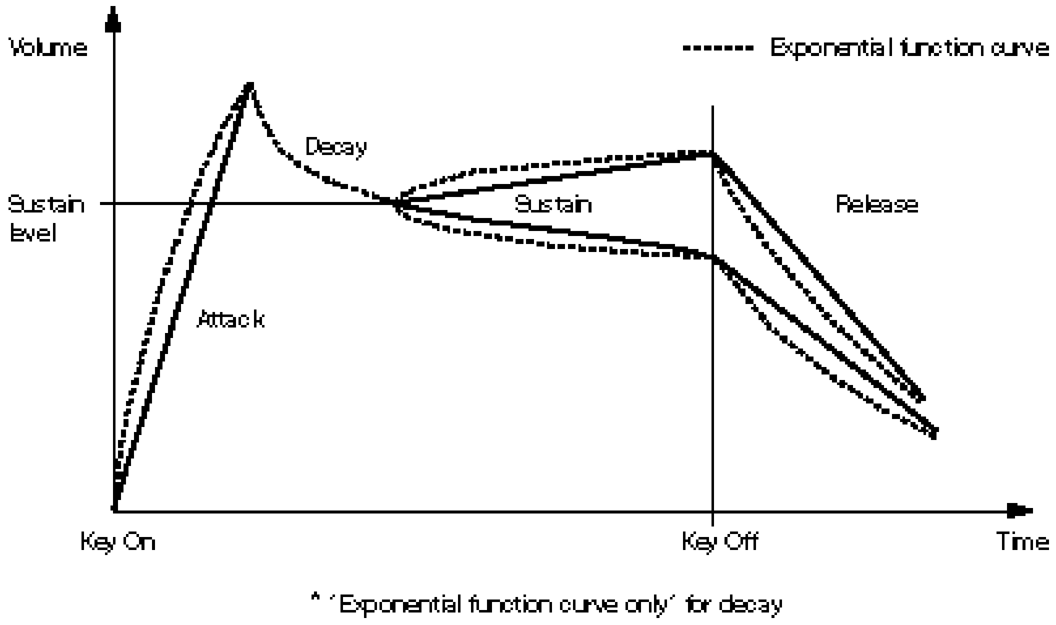

Volume
------- Exponential function curve
Decay
Sustain level
Sustain
Release
Attack
Key On
Key Off
Time
* 'Exponential function curve only' for decay

Figure 4-4 Envelope

Volume

Volume variation can be set separate from the envelope. The four types of linear increment, linear decrement, reverse indexed increment and indexed decrement can be set for the variation with time, apart from constant value.

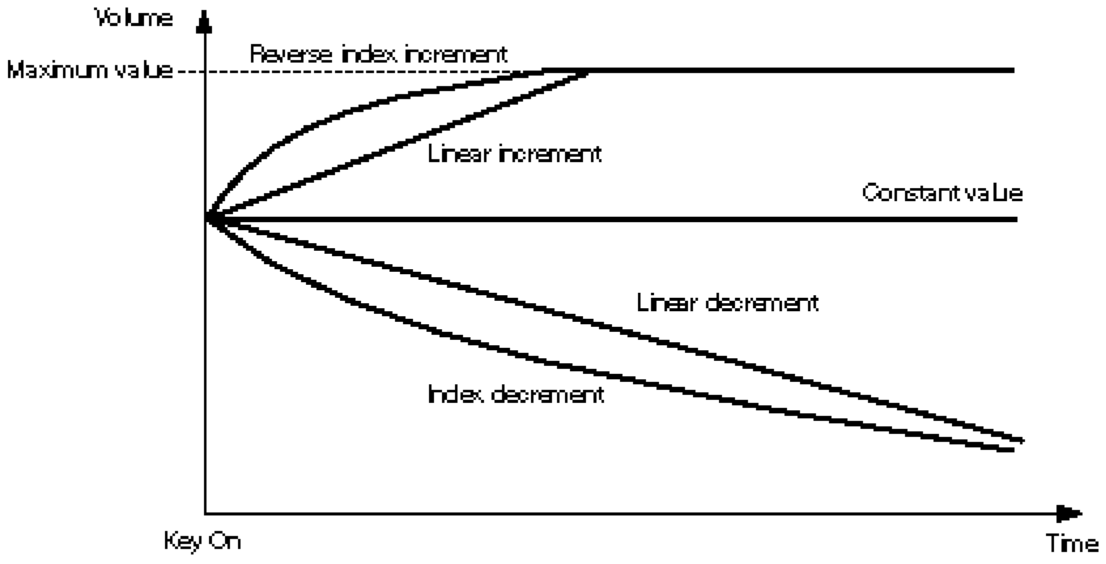


Figure 4-5 Volume

4.5 Reverb Function

A digital reverb which takes the sound buffer as its work area, is mounted in the SPU.

Reverb ON/OFF can be set voice by voice. ON/OFF can be set for CD input as well.

Sound Data Transmission to Main Memory

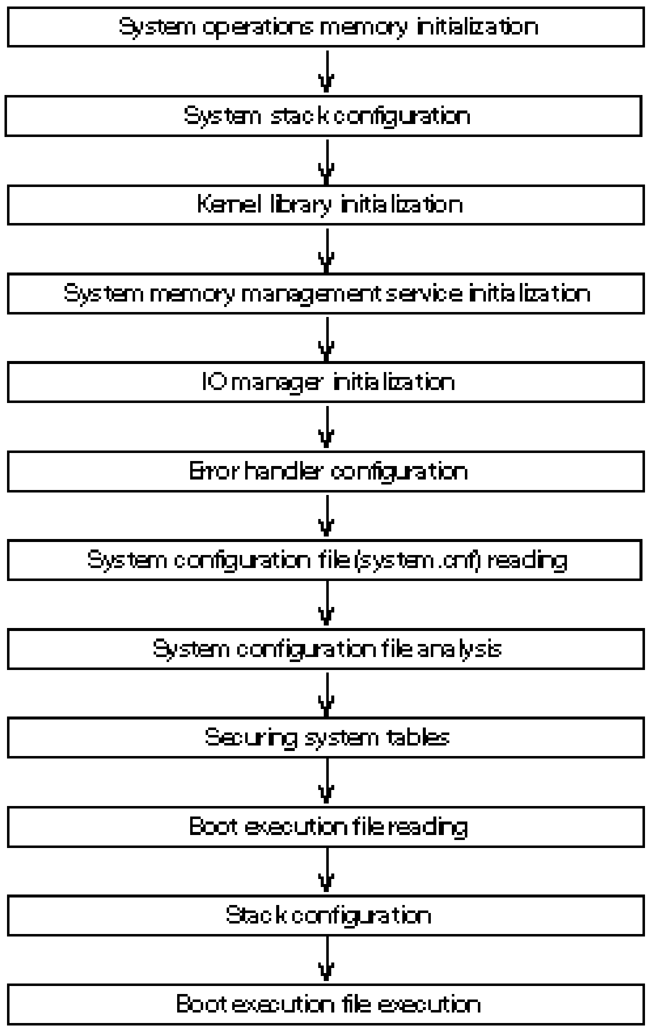
The SPU always writes sound data after CD input volume variation and sound data after specific voice (number of voices: 2) envelope variation to a specific area of the sound buffer every “fs” (fs = 44.1kHz).

The data written to the sound buffer can be transmitted to the main memory by DMA transmission. Sound data created by CD input or SPU can be processed by processing this data.

Appendices

Boot Sequence

The operations of the boot sequence are as follows.


System operations memory initialization
System stack configuration
Kernel library initialization
System memory management service initialization
IO manager initialization
Error handler configuration
System configuration file (system.cnf) reading
System configuration file analysis
Securing system tables
Boot execution file reading
Stack configuration
Boot execution file execution

System Configuration File SYSTEM.CNF

Here we describe the PlayStation system configuration, execution file format etc.

Parameters are described in the format from the beginning of a line “<key word> = <content>.”

Keywords are always in alphanumeric, upper case characters, and on either side of the equals sign a space is required. Lower case characters may not be used. Numbers are in hexadecimal. When the same parameter exists multiple times in a file, the first found takes precedence.

The parameters that can be specified are as follows.

Table B.1 Keywords for system configuration file

Keyword	Content
BOOT	Specify the name of file to be started. Dafault is cdrom : PSX.EXE; 1. Description of file name follows the rule for determining the file system. Example: BOOT = PSXAPP.EXE
TCB	Specify the number of tasks. Default is 4. Example: TCB = 5
EVENT	Specify the number of events. The default is 16. Example: EVENT = 10
STACK	The stack pointer value when the file designated by BOOT is started. Default is 0x8001FF00. Example: STACK = 800FFFF0

ROM Monitor Commands

With the ROM monitor mounted in the development system, debugging can be carried out using the commands below. Notation is as follows:

Table C.1 Notation Examples

Notation	Explanation
Command name	Only the command. No argument.
Command name <argument>	Following the command. Specify an argument
Command name [<argument>]	Following the command. Specify an argument. The portion in brackets may be omitted.

Command list display

```
help
```

Displays the list of commands explained in this appendix.

```
help2
```

Displays the list of commands for hardware test.

Monitor reset

```
re
restart
exit
hot
```

Reset (hot start) the monitor.

```
start
quit
```

Reset (hot start) the monitor.

Register Setting & Display

```
dr[<reg>]
```

displays <reg> register value.
When <reg> is omitted, displays values of entire register.

```
sr[<reg>]
```

resets value in <reg> register.
When <reg> is omitted, asks register name.

```
ds
```

Displays the R3000 status.

Executing the Program

```
ca[<adr>]
call [<adr>]
```

Program is executed from <adr> address and after completion, control returns to monitor. When <adr> is omitted, execution begins from the current value in the EPC register. Breakpoints are ignored with this command.

```
go [<adr>]
```

Executes the program from the <adr> address. When <adr> is omitted, begins execution from the present value of the EPC register.

If a breakpoint is set processing will be stopped with this command. Control will not revert to the monitor upon completion of execution.

Memory operations

```
dw[<adr>[count]]
dh[<adr>[count]]
db[<adr>[count]]
```

The contents of <adr> address is displayed in hexadecimal for <count> bytes. dw is displayed in word units, dh in half word units and db in byte units.

Both <adr> and <count> may be omitted. When <adr> is omitted, memory is displayed starting at the end of the last block displayed. When <count> is omitted, 64 bytes of memory are shown.

```
sw<adr>
sh<adr>
sb<adr>
```

Allows values to be set in ascending order in memory. Press CTRL-C to stop entering values.

```
fw<adr>[<count>]
fh<adr>[<count>]
fb<adr>[<count>]
```

Sets <count> bytes starting at <adr> address to value <data>. fw sets in word units, fh in half word units and fb in byte units. Both <data> and <count> may be omitted. When <data> is omitted, the memory is cleared. When count is omitted, 16 bytes are set. <adr> cannot be omitted.

ROM Monitor Commands (cont.)

Debug

`di[<adr>]`

`dis[<adr>]`

Disassemble from <adr> address.
Where <adr> is abbreviated, it disassembles from the number of the previous series.

`ld2`

Load an “S-format” file into memory. Receives EPC and GP register value.

`smon[<key>]`

Switches to high-speed transmission mode and receives download.

`bp[<adr>]`

Sets a breakpoint in <adr> address. A maximum of 10 breakpoints may be designated.
When <adr> is omitted, set breakpoints will be displayed with an index.

`ub<num>`

Breakpoint <num> will be cancelled. If <num> is omitted all breakpoints will be cancelled.

`cont`

`c`

Continues execution of a program stopped by a breakpoint.

`step`

`s`

Performs a single step action at the machine language level.

`Step`

`S`

Same as step command. However, a subroutine call is considered 1 step.

`qdebug`

Enters “debugs taboo” mode for “remote debugger”. Once this mode is entered, return to the monitor is not possible.

User-Defined Commands

`showcom`

Displays the content of user-defined monitor commands (maximum 20 commands).

`addcom <name> <adr>`
Registers user-defined monitor commands. Designates command name to <name> and execution start address to <adr>.

`delcom <name>`
Cancels registration of user-defined monitor command <name>.

Kernel and Boot

`boot <drv>`
Boots an application program from a drive <drv>. Executes subsets, with the exception of “loading demo” etc, from the “PlayStation” mainframe boot sequence.

`rc <ev> <tcb>`
Resets the kernel configuration in the same way as in the system setting files with contents called EVENT = <ev> and TCB = <tcb>.

`sc`
Displays the current kernel configuration.

`load <file> <sp>`
Reads PS-X EXE format file <file> in memory. Zero-clears data areas without initial values and sets the EPC and GP register values to those in the file. Therefore, it can be executed immediately by the “ca” command.

`loadexec <file> <sp>`
Reads PS-X EXE format file <file> in memory, sets the initial value of the start pointer to <sp>, and executes.

`mem <mbyte>`
When <mbyte> is 2, 8 or 16, sets the effective memory capacity to that value (megabyte units). When there is no <mbyte> designation, displays the set value at that time.

ROM Monitor Commands (cont.)

PC Commands for Debug Monitor

`psxcons`

This is a console program. It has the following functions.

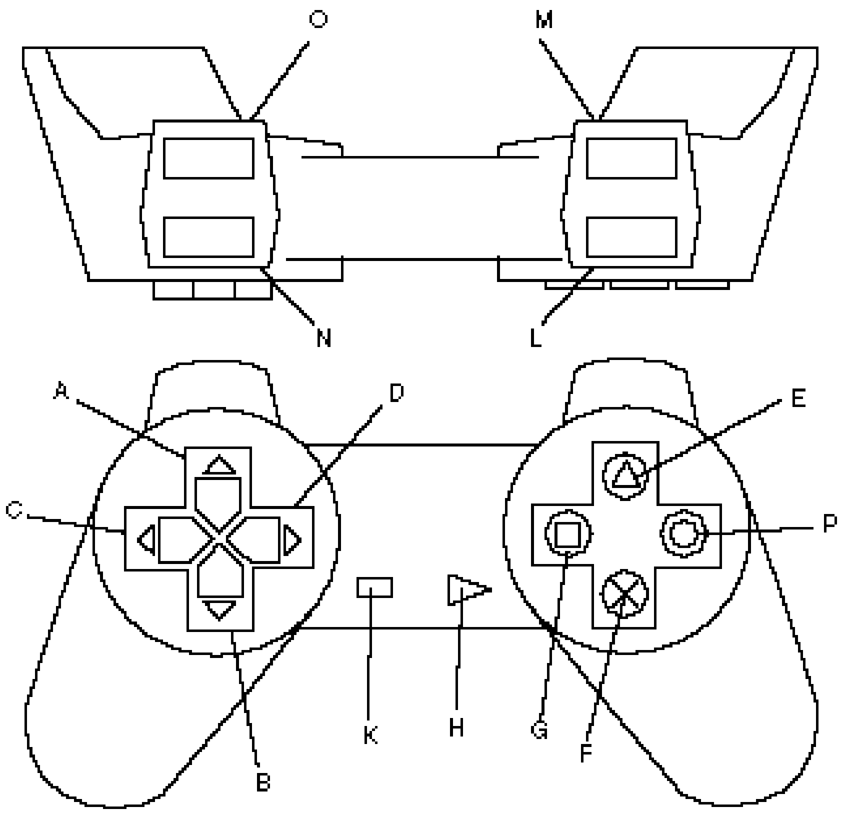
a) Display and operation of pseudo LED and pseudo dipswitch.
b) Download of S-format, execution format and binary files.
c) User interrupt generation.

`psxpt`

This is the file server for the “pt” device. It operates in the background of “psxcons”.

Controller Button Layout

The buttons shown below are to be provided as the controllers attached to the Playstation unit.



Bit no.	Corresponding button
15	C
14	B
13	D
12	A
11	H
10	
9	
8	K
7	G
6	F
5	P
4	E
3	L
2	N
1	M
0	O

Value of each bit
0: Not Pressed
1: Pressed

Table D.1 Controller button layout/Bit correspondence table