PlayStation Hardware

Publication date: August 1998

Sony Computer Entertainment America
919 E. Hillsdale Blvd., 2nd floor
Foster City, CA 94404

Sony Computer Entertainment Europe
Waverley House
7-12 Noel Street
London W1V 4HH, England

Table of Contents

List of Figures

List of Tables

About This Manual

This manual is the latest release of specifications for PlayStation® hardware as of Run-Time Library release 4.3. The purpose of this manual is to describe the PlayStation's hardware architecture and to provide an overview of the PlayStation's subsystems.

For information about boards and development host connection, etc., please refer to the manual accompanying the board.

Changes Since Last Release

There have been no substantial changes to this document since its last release.

Related Documentation

You should read this manual in conjunction with the *Library Overview* and *Library Reference*.

Note: the Developer Support Website posts current developments regarding the Libraries and also provides notice of future documentation releases and upgrades.

Manual Structure

Section	Description
Ch. 1: System Hardware Summary	Describes the hardware composition of the PlayStation system and provides an overview of each component.
Ch. 2: CPU and Its Peripherals	Describes the CPU, memory, physical memory map and OS ROM.
Ch. 3: Graphics System	Describes the GPU, the graphics drawing processor. Topics include the frame buffer, CRT display, screen drawing capability and texture cache.
Ch. 4: Sound System	Describes the PlayStation sound system, which is composed of a sound reproduction processor (SPU) and a CD-ROM decoder.
App. A: Peripheral Configurations	Describes correspondence between bit location and buttons for various controllers.

Developer Reference Series

This manual is part of the *Developer Reference Series*, a series of technical reference volumes covering all aspects of PlayStation development. The complete series is listed below:

Manual	Description
PlayStation Hardware	Describes the PlayStation hardware architecture and overviews its subsystems.
PlayStation Operating System	Describes the PlayStation operating system and related programming fundamentals.
Run-Time Library Overview	Describes the structure and purpose of the run-time libraries provided for PlayStation software development.
Run-Time Library Reference	Defines all available PlayStation run-time library functions, macros and structures.

Inline Programming Reference	Describes in-line programming using DMPSX, GTE inline macro and GTE register information.
SDevTC Development Environment	Describes the SDevTC (formerly "Psy-Q") Development Environment for PlayStation software development.
3D Graphics Tools	Describes how to use the PlayStation 3D Graphics Tools, including the animation and material editors.
Sprite Editor	Describes the Sprite Editor tool for creating sprite data and background picture components.
Sound Artist Tool	Provides installation and operation instructions for the DTL-H800 Sound Artist Board and explains how to use the Sound Artist Tool software.
File Formats	Describes all native PlayStation data formats.
Data Conversion Utilities	Describes all available PlayStation data conversion utilities, including both stand-alone and plug-in programs.
CD Emulator	Provides installation and operation instructions for the CD Emulator subsystem and related software.
CD-ROM Generator	Describes how to use the CD-ROM Generator software to write CD-R discs.
Performance Analyzer User Guide	Provides general instructions for using the Performance Analyzer software.
Performance Analyzer Technical Reference	Describes how to measure software performance and interpret the results using the Performance Analyzer.
DTL-H2000 Installation and Operation	Provides installation and operation instructions for the DTL-H2000 Development System.
DTL-H2500/2700 Installation and Operation	Provides installation and operation instructions for the DTL-H2500/H2700 Development Systems.

Typographic Conventions

Certain Typographic Conventions are used through out this manual to clarify the meaning of the text. The following conventions apply to all narrative text except for structure and function descriptions:

Convention	*Meaning*
`courier`	Indicates literal program code.
Bold	Indicates a document, chapter or section title.

The following conventions apply within structure and function descriptions only:

Convention	*Meaning*
Medium Bold	Denotes structure or function types and names.
Italic	Denotes function arguments and structure members.

Developer Support

Sony Computer Entertainment America (SCEA)

SCEA developer support is available to licensees in North America only. You may obtain developer support or additional copies of this documentation by contacting the following addresses:

Order Information	Developer Support
In North America	*In North America*
Attn: Developer Tools Coordinator Sony Computer Entertainment America 919 East Hillsdale Blvd., 2nd floor Foster City, CA 94404 Tel: (650) 655-8000	E-mail: DevTech_Support@playstation.sony.com Web: http://www.scea.sony.com/dev Developer Support Hotline: (650) 655-8181 (Call Monday through Friday, 8 a.m. to 5 p.m., PST/PDT)

Sony Computer Entertainment Europe (SCEE)

SCEE developer support is available to licensees in Europe only. You may obtain developer support or additional copies of this documentation by contacting the following addresses:

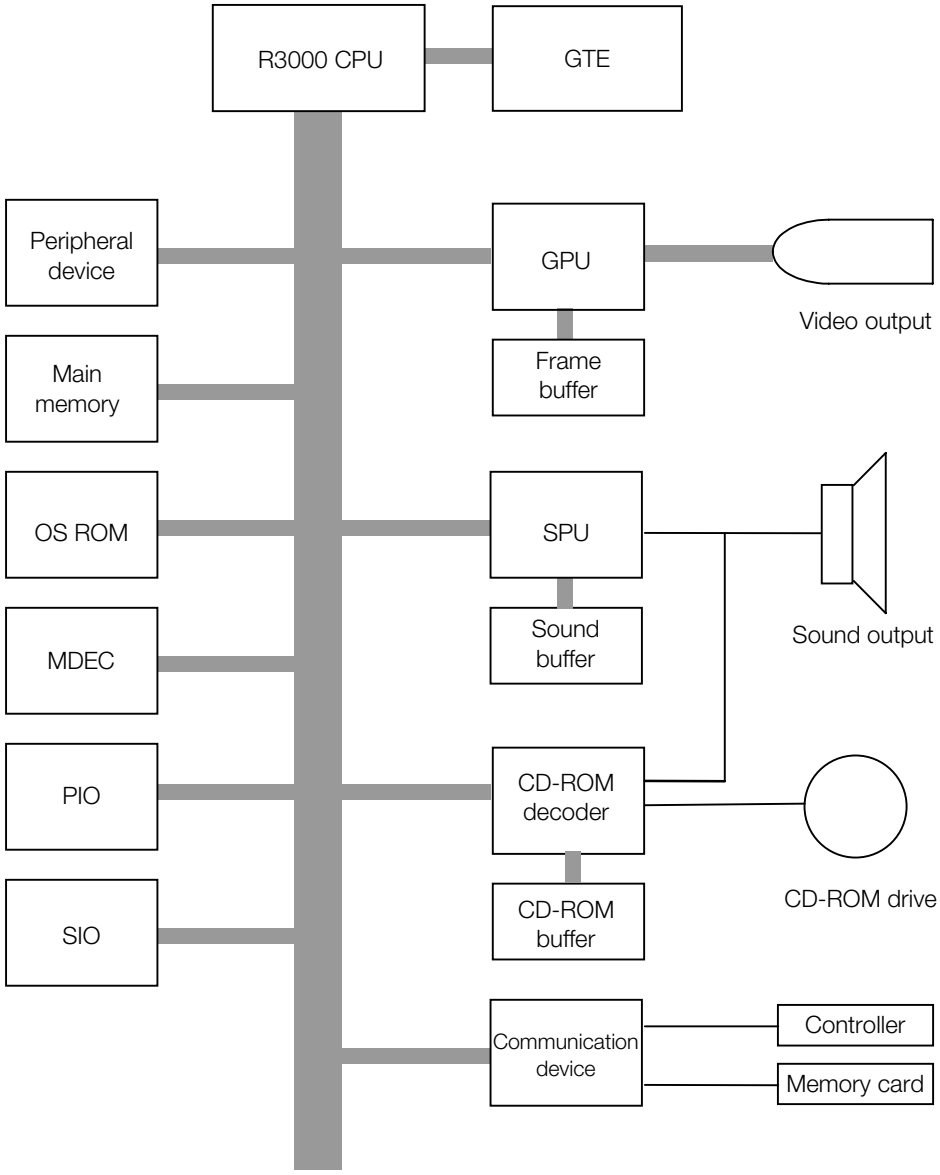
Order Information	Developer Support
In Europe	*In Europe*
Attn: Production Coordinator Sony Computer Entertainment Europe Waverley House 7-12 Noel Street London W1V 4HH Tel: +44 (0) 171 447 1600	E-mail: dev_support@playstation.co.uk Web: https://www-s.playstation.co.uk Developer Support Hotline: +44 (0) 171 447 1680 (Call Monday through Friday, 9 a.m. to 6 p.m., GMT or BST/BDT)

Chapter 1:
System Hardware Summary

System Architecture

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

Figure 1-1: PlayStation Block Diagram



GTE : Graphics Data Generating Processor
GPU : Graphics Data Drawing Processor
SPU : Audio Processor
MDEC : Motion Decoder
PIO : Extended Parallel Port
SIO : Extended Serial Port

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

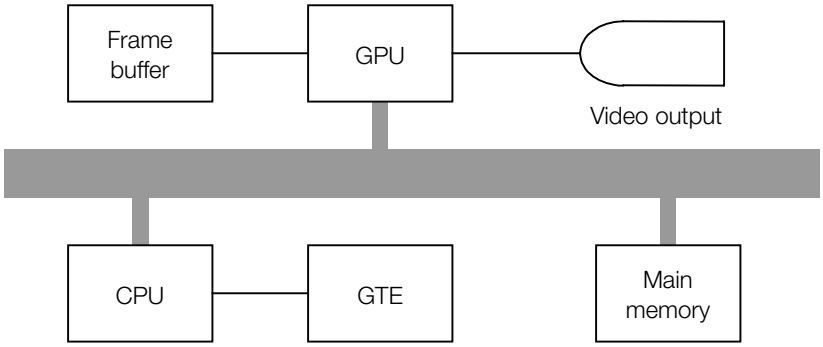
The PlayStation graphics system is composed of a graphics data creation processor (GTE) and a graphics drawing processor (GPU).

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 color palettes (CLUT) from the CPU to the frame buffer, and the causing of the GPU to draw polygons, based on coordinates and color information obtained by the GTE.

Figure 1-2: Graphics System



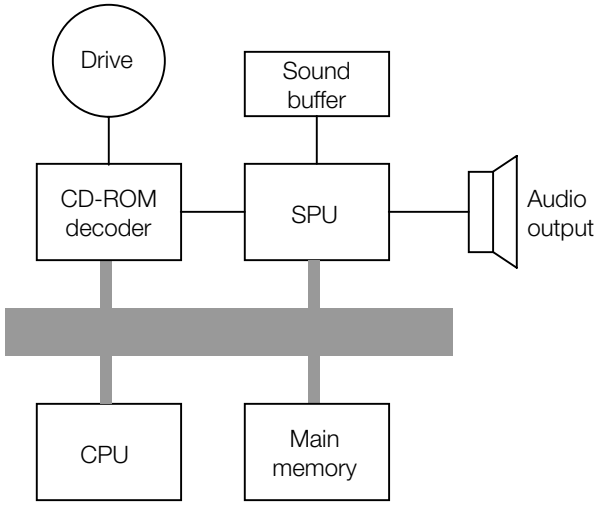
Sound System

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.

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.

Figure 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 that 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 Multi tap 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 Multi tap as well.

Expansion Ports

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

Chapter 2:
CPU and Its Peripherals

Memory

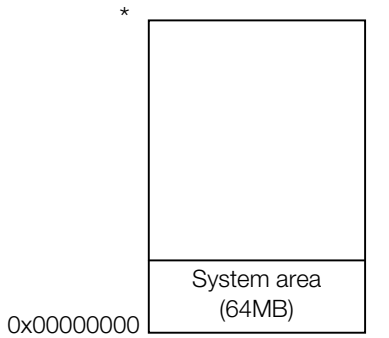
There is 2 MB of main memory installed in the PlayStation.

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:

Figure 2-1: Physical Memory Map



* : 0x1FFFFF (PlayStation platform)
0x7FFFFF (PlayStation developer)

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

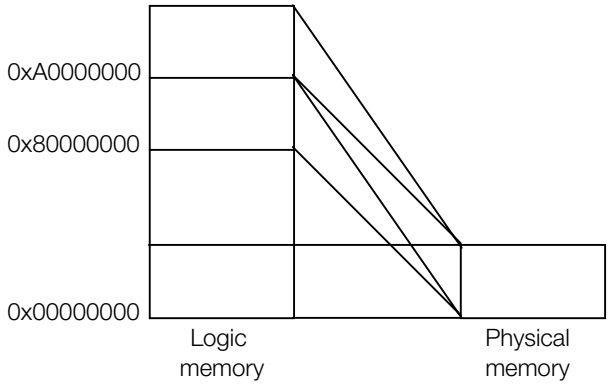
Item	Specification
System Clock	33.8688 MHz
Bus Width	32 bit
I cache	4KB
D cache	1KB (Scratch Pad)
Endian	Little

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.

Figure 2-2: 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	Segment title	Cache
0x00000000-Physical Space MAX	A	Effective
0x1F800000-0x1F8003FF	S	Not effective
0x1F801000-0x1FBFFFFF	X	Not effective
0x1FC00000-0x1FC7FFFF	P	Effective
0x80000000-0x80000000+Physical Space MAX	B	Effective
0x9FC00000-0x9FC7FFFF	Q	Effective
0xA0000000-0xA0000000+Physical Space MAX	C	Not effective
0xBFC00000-0xBFC7FFFF	R	Not effective

Logic Space	Corresponding Logic Memory Segment	Device
0x00000000-MAX	A+B+C	Main Memory (RAM)
0x1F800000-0x1F8003FF	S	Scratchpad (D-cache)
0x1F801000-0x1FBFFFFF	X	Hardware Register
0x1FC00000-0x1FC7FFFF	P+Q+R	Boot 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 that 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 that 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 game programmers 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) that do not comply with this rule will cause address errors.

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`.

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

Table 2-3: R3000 General-purpose Registers

Register #	Macro (1)	Macro (2)	Assembler	Assignment
0	R_ZERO	R_R0	Zero	0 fixed
1	R_AT	R_R1	AT	Assembler reserves
2	R_V0	R_R2	v0	Function return value
3	R_V1	R_R3	v1	Return value (for double type)
4	R_A0	R_R4	a0	Argument #0
5	R_A1	R_R5	a1	Argument #1
6	R_A2	R_R6	a2	Argument #2
7	R_A3	R_R7	a3	Argument #3
8	R_T0	R_R8	t0	Destroyed in function
9	R_T1	R_R9	t1	Destroyed in function
10	R_T2	R_R10	t2	Destroyed in function
11	R_T3	R_R11	t3	Destroyed in function
12	R_T4	R_R12	t4	Destroyed in function
13	R_T5	R_R13	t5	Destroyed in function
14	R_T6	R_R14	t6	Destroyed in function
15	R_T7	R_R15	t7	Destroyed in function
16	R_S0	R_R16	s0	Saved in function
17	R_S1	R_R17	s1	Saved in function
18	R_S2	R_R18	s2	Saved in function
19	R_S3	R_R19	s3	Saved in function
20	R_S4	R_R20	s4	Saved in function
21	R_S5	R_R21	s5	Saved in function
22	R_S6	R_R22	s6	Saved in function
23	R_S7	R_R23	s7	Saved in function
24	R_T8	R_R24	t8	Saved in function
25	R_T9	R_R25	t9	Saved in function
26	R_K0	R_R26	k0	Kernel reserves
27	R_K1	R_R27	k1	Kernel reserves
28	R_GP	R_R28	gp	Global pointer
29	R_SP	R_R29	sp	Stack pointer
30	R_FP	R_R30	fp	Frame pointer
31	R_RA	R_R31	ra	Return address

AT Register

The AT register is used as a work area by the assembler. The 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.

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 the “bss section”. Here, the center 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 General-purpose Registers

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

Figure 2-3: Program Counter Values

Timing	Program Counter
Immediately after power on reset	0xbfc00000 fixed
When an external interrupt is generated	0x00000080 fixed
When an exception interrupt is generated	ox00000080 fixed

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 (during syscall instruction execution)
Bp	Breakpoint (during break instruction execution)

Memory Access Timing

Three high speed memories consisting of an I-cache and two buffer memories are placed between the CPU and main memory for the purpose of reducing access time to memory. The buffer memories are transparent to a running program and have the side-effect of making it impossible to accurately predict the timing of the program.

Related Structures

(1) I-cache

The I-cache is 4K bytes deep and is divided into units known as lines. Each line is 16 bytes, or the equivalent of 4 instructions. Each line in the I-cache is addressed with 8 bits, and there are a total of 256 lines. A line also contains a management field known as a tag. The tag indicates whether the corresponding instructions have already been read into the line. The tag also stores information such as the upper 20 bits of the instruction memory address.

When an instruction is fetched, bits 5-12 of the memory address are first used to specify a unique line in the I-cache. The CPU inspects the tag associated with the line and determines whether the desired instruction is present in the line. If it is, the CPU reads the instruction from the line. The number of cycles required to perform this operation is 1.

If the desired instruction is not present in the line, starting with the next cycle, the CPU will read the line from memory which contains the desired instruction. If the memory address of the target instruction is not on a line boundary (i.e. it is not a multiple of 16), the instruction at the low address with the lowest possibility of execution cannot be read into the line. In this case, 4-7 cycles are required to read the data into the R-buffer. One additional cycle is required to read the line to the CPU.

(2) R-buffer

The R-buffer is a single 4-byte long register that is used to buffer read data. Loading data from memory into the R-buffer requires 4 cycles and reading data from the R-buffer to the CPU requires 1 cycle. Therefore, a pure data read requires 5 cycles.

The R-buffer cannot be accessed by bytes or halfwords. Reading a single word and dividing it into multiple bytes and halfwords requires (read instruction X 5) cycles.

(3) W-buffer

The W-buffer consists of a set of 4 registers, 4-bytes long and is implemented as a fifo (first in first out). It is used to buffer write data. Each register provides a status flag which can store 2 states (empty, non-empty). Each write is allocated one register, without regard to data length (byte, word, etc.).

If a write is attempted when there is no available register, the CPU first writes the contents of the non-empty register that was first to be written to main memory. This frees the register and it is available for storing the new write data.

Each write to the W-buffer requires 1 cycle. This provides a high rate of instruction execution throughput as the CPU does not have to wait for writes to main memory to complete.

When the W-buffer contains 4 words (16 bytes) and the CPU attempts to perform a write, the instruction waits until a register is freed in the W-buffer. Subsequently, data is written from the W-buffer to main memory and registers are freed in sequence. The timing of this operation does not interfere with instruction execution. However, it cannot be controlled from software.

When the CPU attempts to read data from a memory address for which there is a write pending in the W-buffer, the write is allowed to complete before the read is performed to ensure that the read obtains the newest data. In general, write timing is not guaranteed, however, a write is always guaranteed to complete first if a read was issued to the same address.

Writing to the W-buffer from the CPU requires 1 cycle to complete, whereas writing from the W-buffer to main memory requires 4 cycles. W-buffer accesses cannot be divided into bytes or halfwords. The required number of cycles is always (write instruction X 4) for writing one word divided into multiple bytes and halfwords.

When writing 2 or more words from the W-buffer to locations in main memory that are on the same 1K-byte page, 2 cycles are required after writing the first 2 words.

(4) ScratchPad

The ScratchPad is an area mapped into main memory space. The ScratchPad can read and write at high speed (once per cycle). However, the ScratchPad cannot be the target of a DMA transfer.

Access Timing

The table below shows access timing from the CPU to main memory

Table 2-5: Memory Access Timing

Contents	Number of Words	Penalty Cycles	Required Number of Cycles
Reads			
I-Cache → CPU	1	0	0
ScratchPad → CPU	1	0	0
Main Memory → I-Cache	1	5	4
	2	6	5
	3	7	6
	4	8	7
Main Memory → CPU	1	5	4
Writes			
CPU → W Buffer	1	0	0
CPU → ScratchPad	1	0	0
W-Buffer → Main Memory	1	0	4
(Continuous writes to same page)	2	0	6
	3	0	8
	4	0	10

Estimating Instruction Execution Timing

The I-cache together with the W and R buffers can improve performance by reducing access time from the CPU to main memory. A side-effect from the presence of these buffers is that it is no longer possible to predict the actual execution time of a program just by knowing the number of instructions. Furthermore, the cache and buffers are managed automatically by the main CPU and cannot be controlled from a program. Consequently, it is not possible to accurately predict the number of cycles required to execute a segment of code. However, from the underlying architecture, it is possible to compute best and worst case instruction execution times.

Chapter 3:
Graphics System

Overview

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 coprocessor

Frame Buffer

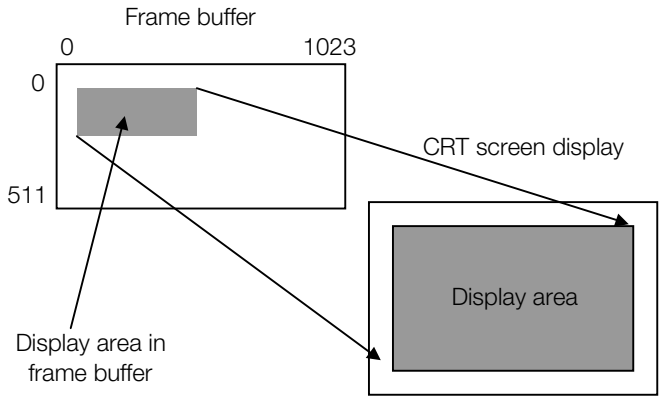
The GPU has 1 MB of video memory (VRAM).

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:

Figure 3-1: Display Areas



Screen Mode and Display Location

The GPU supports the following screen modes:

Table 3-1: Window Resolutions

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

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 Direct Mode

This mode can display up to 32,768 colors.

The number of display colors is limited compared to 24-bit direct 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 Direct 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 direct mode *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 the following screen drawing features:

Figure 3-2: GPU Drawing Capabilities

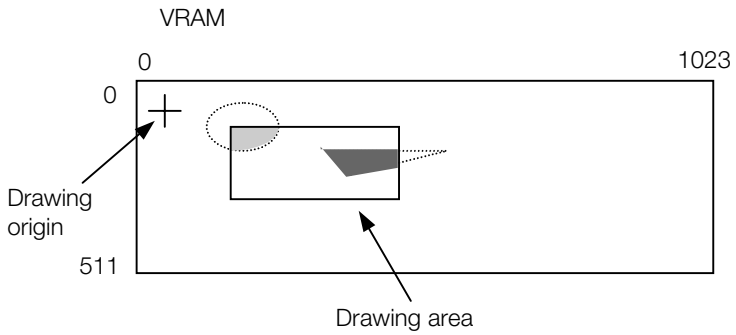
Name	Description
Sprite Drawing	1 x 1 pixel - 256 x 256 pixel
	4-bit CLUT (16 colors/sprite)
	8-bit CLUT (256 colors/sprite)
	15-bit DIRECT (32,768 colors/sprite)
Polygon drawing	Flat shading
	Gouraud shading
	Texture mapping
Line drawing	Gradation possible
Image transfer	CPU → Frame buffer
	Frame buffer → CPU
	Frame buffer → Frame buffer
Other	∝ Blending (translucency)
	Dithering
	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 to 1023. Since the size of the frame buffer is 1024 x 512, the overflow 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.

Figure 3-3: Clipping and Drawing Areas



Sprite Drawing

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

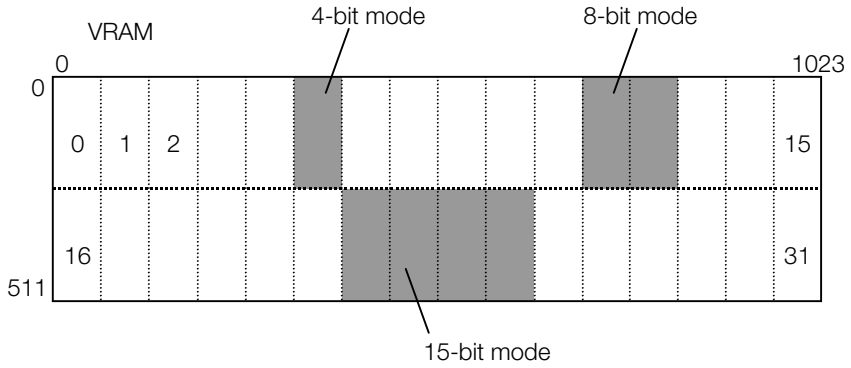
Sprite and Texture Page

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.

Figure 3-4: Texture Page



The page number can be calculated from the coordinate values in the upper left corner of the texture page (tx, ty) as follows:

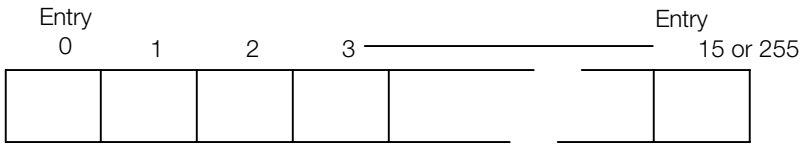
tpage = ty/16 + tx/64

However, the actual tpage will contain data such as CLUT modes (4/8/16 bits), semi-transparency rates, etc.

Sprite Pattern 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.

Figure 3-5: CLUT Structure



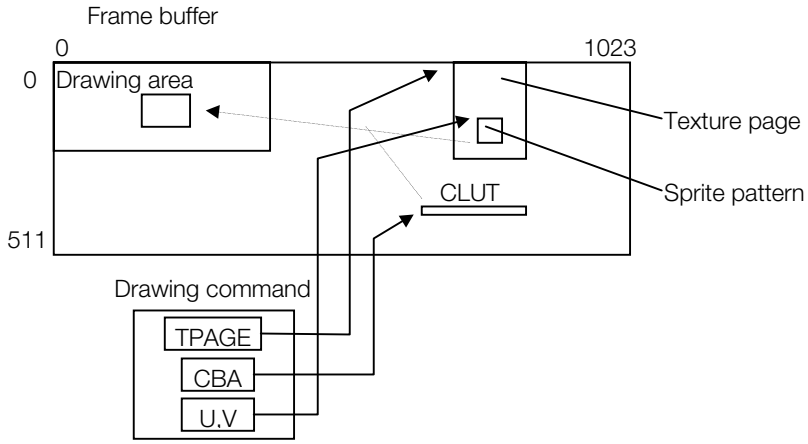
One entry has the same structure as a 15-bit single pixel.

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.

Conceptual Diagram of Sprite Drawing

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

Figure 3-6: Sprite Drawing Operation



Drawing 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.

Texture Cache

The texture cache, or T-cache, is a high-speed buffer memory located between the GPU and the frame buffer. The primary function of the T-Cache is to improve drawing speed.

Mechanism

The T-cache is accessed from the GPU. It is a small-scale memory that is faster than the frame buffer.

The GPU scans a rendered polygon in the horizontal direction in order to determine the textures of the corresponding pixels of the polygon. Although each pixel of a polygon is drawn only once, there is a high probability that each pixel of the texture pattern will be referred to more than once. Consequently, reading the texture pattern into the T-cache in advance speeds up texture mapping of the polygon, compared to accessing the texture pattern directly from the frame buffer.

Pixel information that is obtained from the texture is tested to see whether it is already in the T-cache. If the pixel info is present in the T-cache, the frame buffer is not accessed and the pixel information is read from the T-cache instead.

Data in the T-cache is saved in polygon intervals. Therefore, when drawing polygons continuously, the drawing speed will be faster when the polygons use textures of a size which can fit in the T-cache.

Size

The size of the T-Cache is 2K bytes. Both the frame buffer, which stores the texture, as well as the T-cache contain a 2-dimensional address.

This address changes according to the pixel mode of the polygon which is being drawn. The size of the T-cache for each pixel mode is shown below.

Table 3-2: T-Cache Size

Pixel Mode	Size (width x length)
4 bits/pixel	64 x 64 pixels
8	64 x 32
16	32 x 32

Frame Buffer Access Timing

Please refer to the “libgpu Library Overview” for information on frame buffer access timing.

Chapter 4:
Sound System

Overview

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.

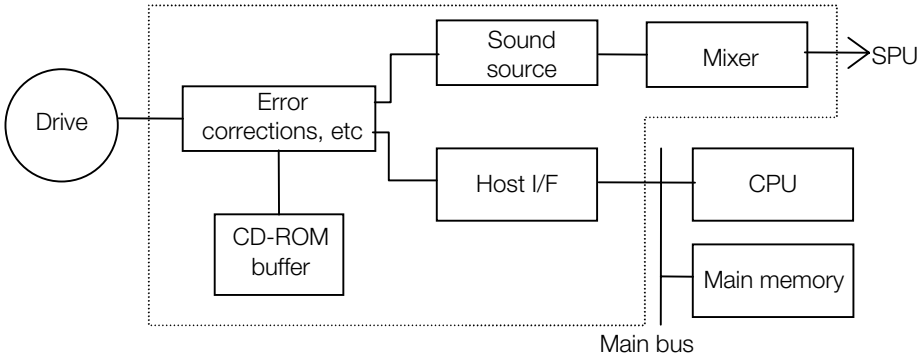
The CD-ROM decoder reproduces audio while reading CD-DA or CD-ROM XA data on a disk. The audio output of the decoder temporarily enters the SPU, is mixed with the SPU output, and then becomes the final audio output via a reverb unit.

CD-ROM Decoder

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

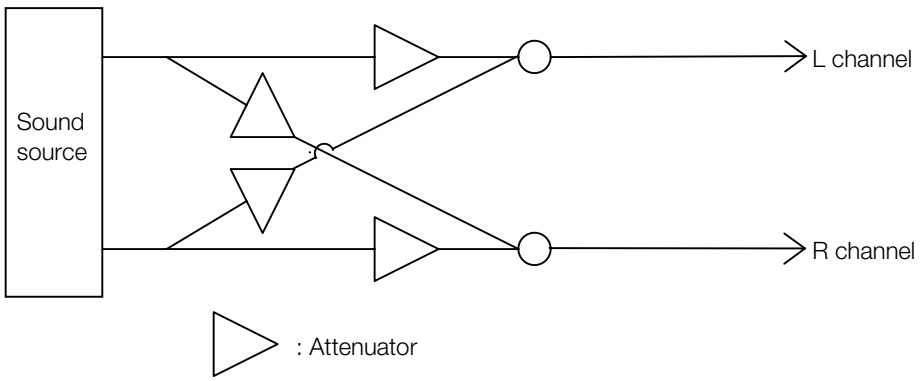
Sound reproduction is executed based on CD-DA 16-bit PCM data and ADPCM data determined by CD-ROM XA. The data read into the CD-ROM buffer from the drive is processed by the sound source in the CD-ROM decoder after error correction. The audio signal obtained is transmitted to the SPU via a mixer built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

Figure 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

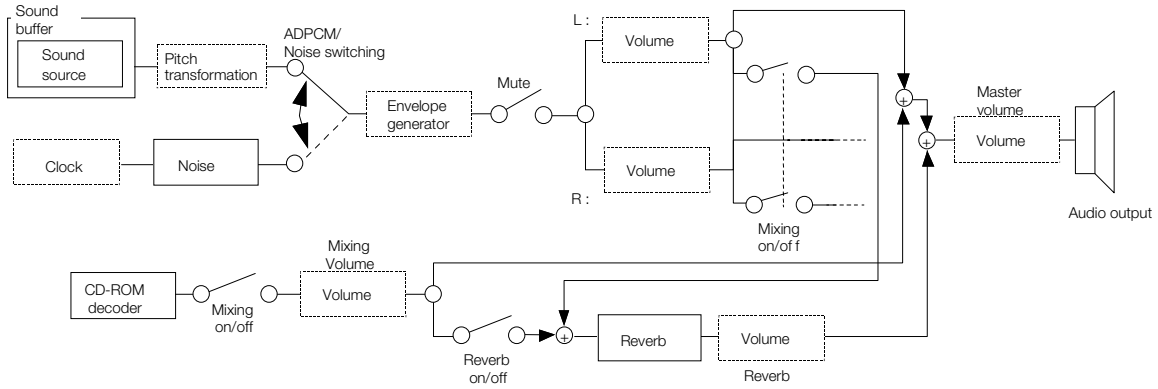
The sound reproduction processor (SPU) executes reproduction at a sampling frequency of 44.1 KHz, taking the ADPCM sample data in local memory as the sound source.

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

Figure 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.

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
```

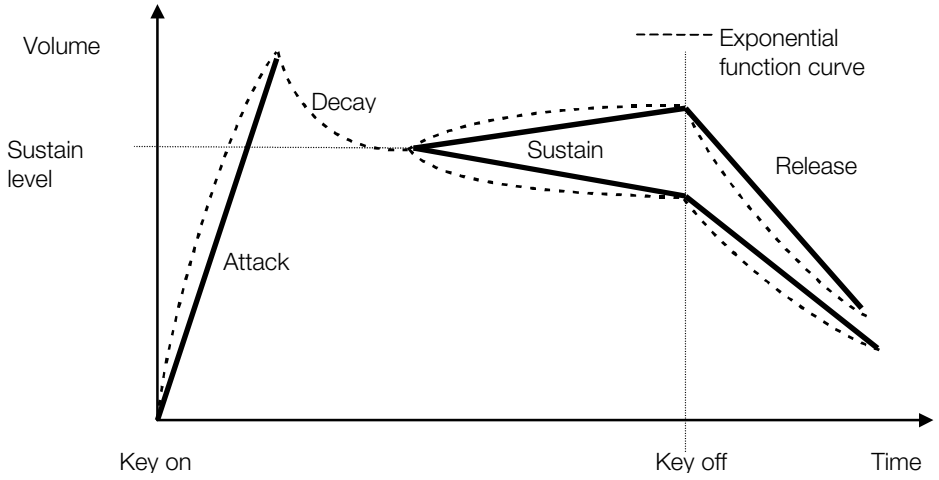
Noise Source

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.

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.

Figure 4-4: Envelope

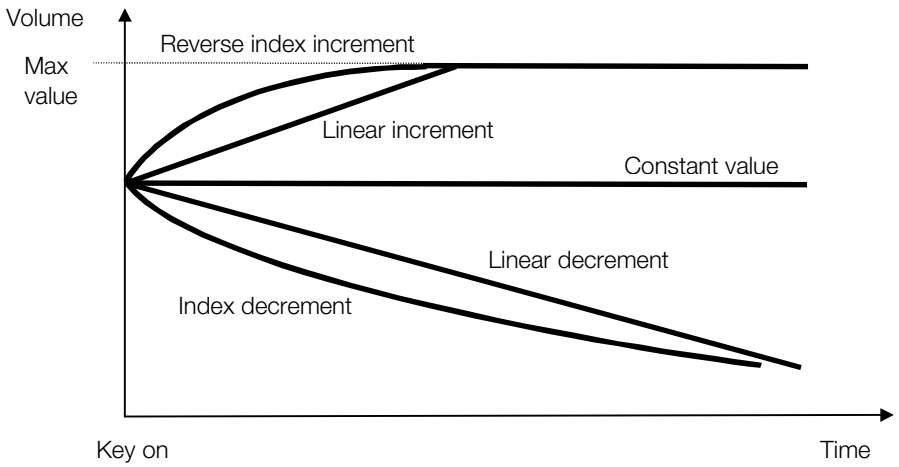


Exponential function curve only for decay

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



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).

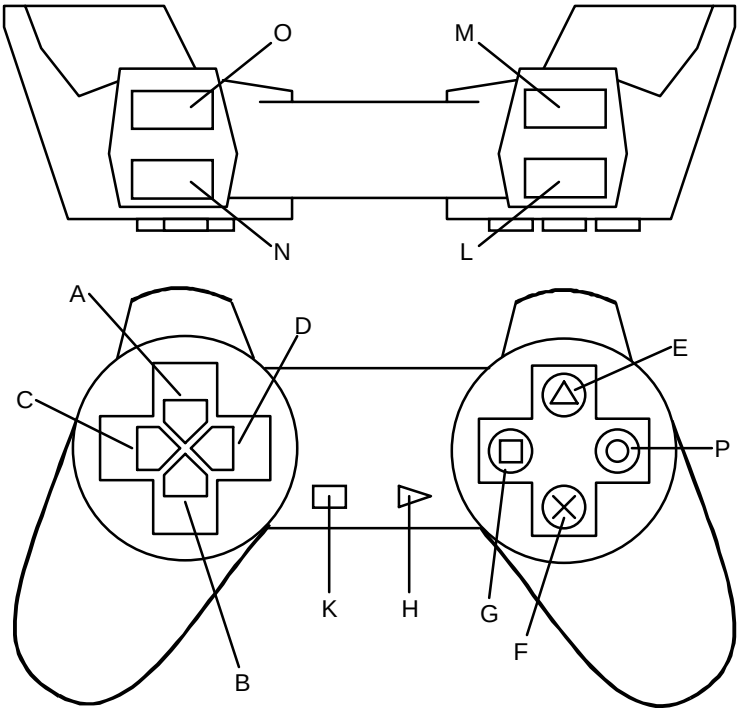
The data written to the sound buffer can be transmitted to the main memory by DMA transmission. Sound data created by CD input or SPU can be processed by processing this data.

Appendix A:
Peripheral Configurations

Controller Button Configuration

The controller provided with the PlayStation is illustrated below.

Figure A-1: Controller Buttons and Bit Correspondence



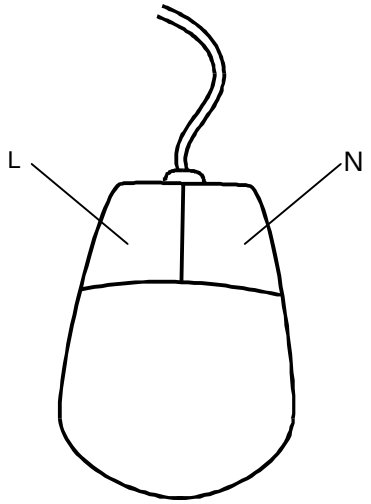
Bit number	Button
15	C
14	B
13	D
12	A
11	H
10	
9	
8	K
7	G
6	F
5	P
4	E
3	L
2	M
1	N
0	0

Bit values
0: Pressed 1: Released

Mouse

The mouse sold with the PlayStation is illustrated below.

Figure A-2: Mouse buttons and bit correspondence



Bit number	Button
15	
14	
13	
12	
11	
10	
9	
8	
7	
6	
5	
4	
3	L
2	N
1	
0	

Bit values

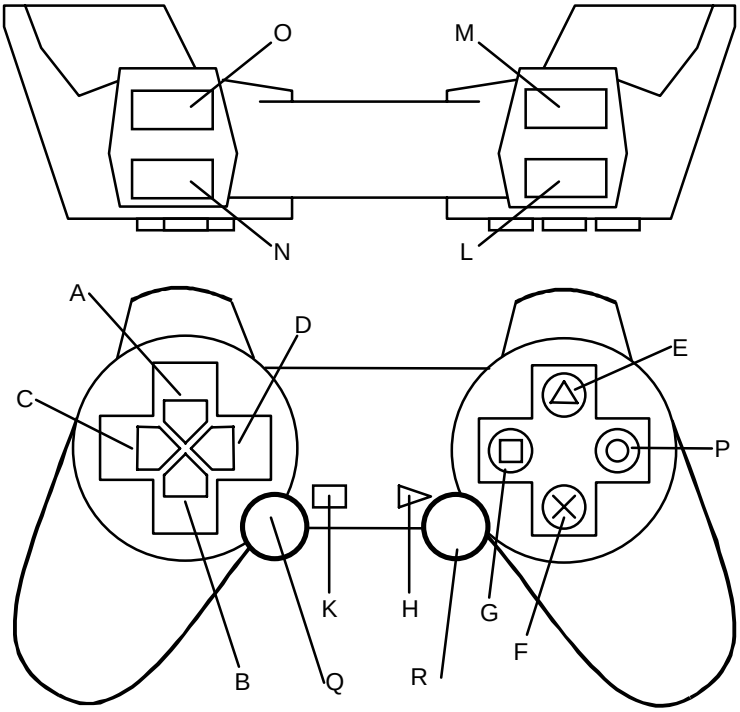
0: Pressed 1: Released

Analog Controller Button Locations

The following diagram shows the analog controller sold by Sony Computer Entertainment for the PlayStation.

Analog Controller Mode

Figure A-3: Correspondence between bit number and controller button

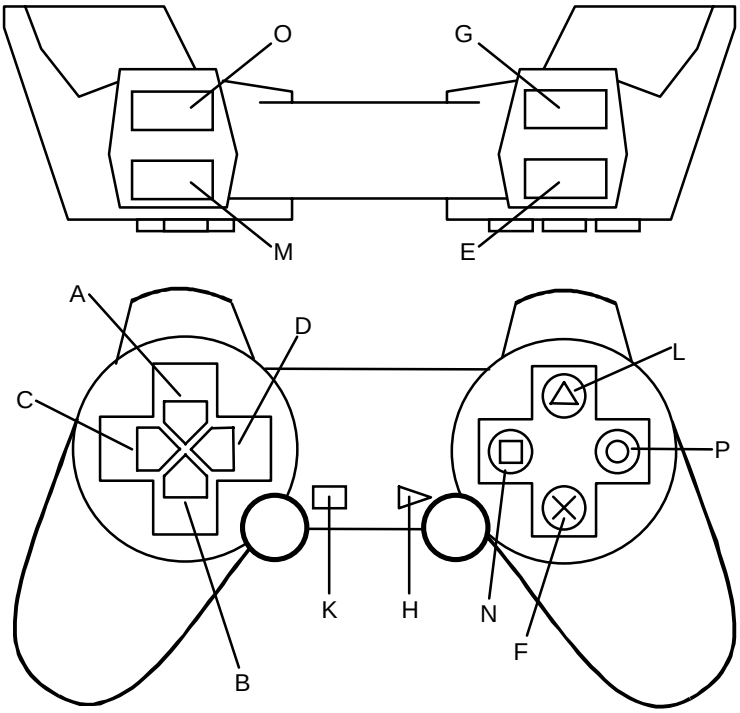


Bit number	Button
15	C
14	B
13	D
12	A
11	H
10	R
9	Q
8	K
7	G
6	F
5	P
4	E
3	L
2	N
1	M
0	O

Bit values
0: Pressed 1: Released

Analog Joystick Mode

Figure A-4: Correspondence between bit number and controller button

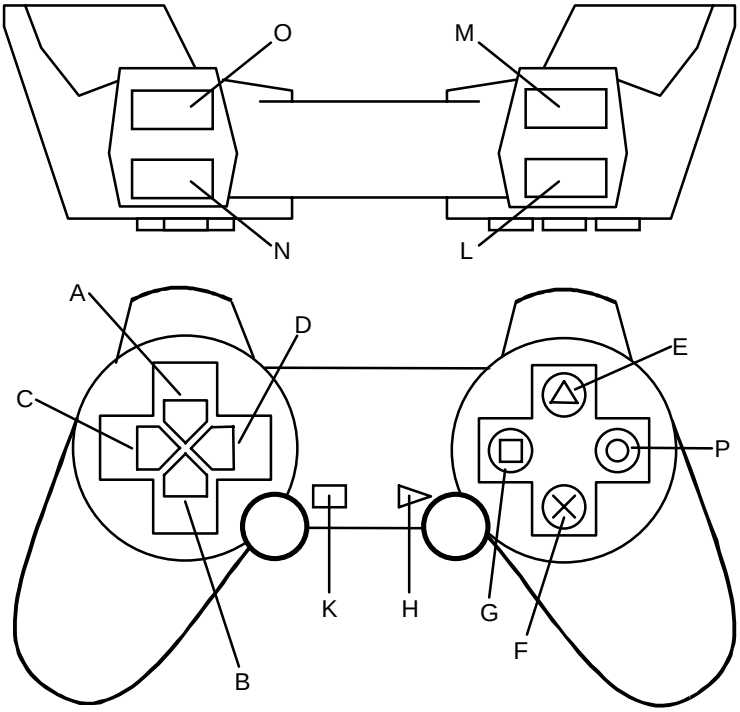


Bit number	Button
15	C
14	B
13	D
12	A
11	H
10	
9	
8	K
7	G
6	F
5	P
4	E
3	L
2	N
1	M
0	O

Bit values
0: Pressed 1: Released

Digital Controller Mode

Figure A-5: Correspondence between bit number and controller button



Bit number	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

Bit values
0: Pressed 1: Released

Index