PLAYSTATION DEVELOPER'S GUIDE

NOTE : DOCUMENT STILL UNDER DEVELOPMENT

Sony Computer Entertainment Europe

Developer Support Group

Version: 2.4 - 18 November 1996

About This Document

As a PlayStation Developer you have access to technical support from SCEE (Sony Computer Entertainment Europe) via the Developer Support group. This document is designed to supplement the standard PlayStation documentation, providing hints and tips for the whole development process from receiving your licensed development kit, to mastering you first gold disc, ready for submission.

We are grateful to all who contributed to this guide - especially all those who developers who provide tips to other developers on our BBS. We welcome any further information that you have.

The latest version of this document can be found on the BBS as DEVGUIDE.ZIP.

History

- October 1995 (Dave, Paul, Laura and others)

  Document first written.

- *20-Nov-1995 (*Paul*)*

  Version 1.33. Slight re-ordering, more references.

- *24-Nov-1995 (*Paul, *Dave)*

  Added more information on CD-Emulation and Disc Mastering, memory and peripherals.

- *11-Dec-1995 (*Paul, *Dave)*

  Add sections on XA, code overlay, tidy up

- *29-Feb-1996 (*Paul, *Dave)*

  More on memory cards and CPE2X.

- *29-Mar-1996 (*Paul*)*

  Addition of a Table of Contents, Index, more general information.

- *29-Apr-1996 (Harry)*

  Tips on Sound.

- 26th-July-1996 (Laura, Paul)

  Basic level sound and re-ordering. Split into two documents (Tools Guide and Developer's Guide).

- 19th-Aug-1996 (Paul)

  Added Section on multi-CD from SCEI

- 23-Sep-1996 (Paul, Dave)

  More on CD Emulator, DTL-H2500

- 18-Nov-96 (Paul)

  More on CD Access and contact information

Contacts

Department:	*Developer Support*
Technical Support:	*Paul Holman* *Stuart Ashley* *Dave Coombes* *Vince Diesi* *Mal Duffin* *Lewis Evans* *Colin Hughes* *Michael Braithwaite* *Kevin Thompson* *Dave Virapen* *Jason Page (Sound)*
Production Co-ordination:	*Sarah Bennett*
(Tools Ordering)	*Tim Flett*
Newsletter:	*Laura Smith*
Address:	*SCEE* *Waverley House* *7-12 Noel Street* *London W1V 4HH*
Telephone:	*+44 (0) 171 447 1649 (Production Co-ordination)* *+44 (0) 171 447 1650 (Production Co-ordination)* *+44 (0) 171 447 1680 (Hot line[1])*
FAX:	*+44 (0) 171 390 4324*
E-Mail:	*dev_support@interactive.sony.com* *prod_coord@interactive.sony.com* *ps_developer@interactive.sony.com*
BBS:	*+44 (0)171 390 4327* *+44 (0)171 390 4328* *+44 (0)171 390 4329*

[1] *For new developers, and titles in their last month before submission*

Table of Contents

2 SUPPORT DETAILS

2.1 The BBS

The Developer is issued a separate User ID and Password, yours will be sent to you in a separate letter.

Queries to Developer Support can be mailed to the user "*sce support*". You have access to two main upload/download areas - your own, and the PlayStation Common area.

In addition to the PlayStation Common Conference/File area, there are a number of Support BBS provides facilities for uploading and downloading software, and mail for queries and support requests. Each Developer other conference areas - their contents should be fairly intuitive:

1. **General Email**
2. **General Chat**
3. **PlayStation Common (whose file area contains all of the down-loadable libraries, documents etc.)**
4. **PlayStation C++**
5. **CD Programming**
6. **Known Bugs**
7. **Demo Disc - for producing demos for the European demo disc.**
8. **New Releases**
9. **Windows95**
10. **CD ROM Emulator**
11. **CD ROM burner**
12. **Graphic Artist Board**
13. **Sound Artist Board**

The last four categories (10 to 13) will only be visible if your have bought licenses for the corresponding tools.

You are recommended to check the BBS at regular intervals - to ensure you have the latest versions of development tools and any new useful utilities. It is probably worthwhile updating the development tools from the BBS as soon as possible, as the versions stored there are upgraded regularly.

If you have no way of accessing the BBS, contact with technical support by fax or E-Mail, to arrange an alternative method of receiving updates.

(i)The Best Way To Get Support

At the moment, we are encouraging developers to contact us with queries by E-Mail, fax or through the BBS, rather than directly by phone. There are several reasons for this, the most important being that often queries cannot be answered on the spot. This method also ensures fairness by enabling us to handle questions on a first come, first served basis..

(ii)BBS Etiquette

When entering a query onto the BBS, it helps if:

- you choose the most appropriate conference for the problem,
- you include all information about tools/libraries - including version numbers,
- if the query is hardware related, you provide as much information about the hardware you are using as possible (for example, the memory/processor configuration of your PC),
- if you can, upload pieces of code that demonstrate the problem (all code will be treated in strict confidence).

Remember, you can enter private messages if you are concerned about security.

2.2 Useful Reading

- “PlayStation Development Tools Guide” - SCEE Developer Support
  (on BBS as DEVTOOLS.ZIP)
- “Guidelines for PlayStation PAL Conversions” - SCEE Developer Support
  (on BBS as PALGUIDE.ZIP)
- ELSPA Anti-piracy screen, required for all titles approved by SCEE
  (on BBS as WARNING.ZIP)
- “QA Standards ...” - Technical Requirements for all SCE Territories
  (on BBS as QADOC.ZIP)
- MIPS RISC Architecture, by Gerry Kane and Joe Heinrich (Publisher: Prentice Hall, ISBN: 0-13-590472-2)
- MIPS Programmer’s Handbook by Erin Farquhar and Philip Bunce (Publisher: Morgan Kaufmann, ISBN: 1-55860-297-6)

2.3 Development Tools

2.3.1 Outline

PlayStation Development Tools consist of three sets of kit and some miscellaneous tools:

- ***PlayStation Development System***

Programmer tools which operate on a PC to provide a programming environment for developing PlayStation Games. A specialised PlayStation known as a 'Debugging Station' is used in conjunction with the Development System to test games from CD Write-Once and Production Discs.

- ***PlayStation Sound Artist Tool***

A system allowing musicians to create PlayStation music with known sound tools on the Apple Macintosh computer.

- ***PlayStation Graphic Artist Tool***

A system which operates in a PC environment to provide the Graphic Artist with 2D and 3D graphic data conversion and manipulation capabilities as well as emulation of PlayStation graphics processing capabilities. These tools enable the Graphic Artist to work independently from the PlayStation Development System.

- ***Miscellaneous Tools***

There are further accessories which can be used as part of the Development System, if required and consumer PlayStations and Games.

To develop games for the PlayStation, you need to use the PlayStation Development System. For sound and graphics, you can use any tool that you prefer. Obviously, SCEE recommend and support only the Sony proprietary Sound Artist and Graphic Artist tools.

2.3.2 General Details

2.3.2a Development System (Programmer Tools)

(i) Hardware

The majority of PlayStation development is centred around a standard PC. The recommended minimum is a 488/66 with 16Mb RAM, CD-ROM Drive and a half a gigabyte of local disk space (with a great deal more disk space, either local or accessible via a network, depending on the complexity of your game).

There are two forms of development systems - one with an ISA interface (the **DTL-H2000**, requiring a PC with 2 full length ISA slots)), and the newer PCI based version (the **DTL-H2500** requires one full length PCI slots, and one other slot). The DTL-H2000 system includes two special controllers, for the DTL-H2500 you will need to purchase a **DTL-H2080** and some standard controllers. Both systems have and video and audio out (via composite and S-video cables), connection for an optional PS CD-ROM drive **(**an external **DTL-H2010** for the DTL-H2000 or the internal **DTL-H2510** for the DTL-H2500**)** and a serial communications port for use with a Link Cable.)

If you are doing a lot of work with both PAL and NTSC, we recommend an RGB SCART cable that can be acquired from Lightwave Cables. (The part number is SP198, and the cable costs £5.08. Lightwave may be contacted on +44 (0)151-630-5003).

In addition, your PC should have room for at least one additional ISA board - for the **CD-ROM Emulator Board (DTL-H2020)** ISA Card which itself requires an additional dedicated SCSI AV hard disk. (Any good quality, recent (1995+) disk should be acceptable - we've had good experiences with IBM, Micropolis and Fireball drives).

You also require a television or colour monitor (ideally both PAL and NTSC compatible).

At the moment, SCEE support standard DOS/Windows, but are moving towards Windows '95 support.

In the latter stages of development (if not before), you also need to start writing PlayStation format CD-ROMS - this requires the **CD Write Once Drive (CDW-900E** *or* the newer **CDU921S)** unit, which is connected to your PC via a SCSI interface (you will need to buy an additional Adaptec AHA-154*x* SCSI card).

Test PlayStation CD-ROM "Gold" discs with either a **PlayStation CD-ROM Drive (DTL-H2010)** which attaches to the DTL-H2000, and/or one or more Debugging Stations. There are three types Debugging Stations: **Japanese NTSC (DTL-H1000), US NTSC (DTL-H1001)** or **UK PAL (DTL-H1002)** to use according to your target markets, and are similar to commercially available machines (with the exception that they allow Gold discs to be run from any territory).

(Note that the internal ROM varies slightly from territory to territory - which is why there are three types of debugging stations.)

You can use any good quality write-once Gold discs for development, but we supply a range of **Write Once Mastering Discs (CDR-71PBS)** for creating high quality masters to be used for submission to QA and Approvals.

For development of games that support connected PlayStations playing together, you can buy either a standard **Link Cable (SCP-H1040)** to connect Debugging Stations together, or a **Link Cable for DTL-H2000/H2500 (DTL-H2060)** that will allow two Development Board Sets to be linked (but will not connect to a PlayStation).

The Development Board Set's default Controllers and ISA Board have different plugs and sockets from commercially available Controllers - if you wish to test memory cards or use standard Controllers, then you need to buy from us a **Controller Box (DTL-H2080)** adapter.

(ii) Software

The Development Board Set comes with the complete range of GNU C-based development software and libraries required. Please note that we strongly recommend that all Developers connect to our BBS to download any updates to the libraries and tools.

On the BBS, in the "PlayStation Libraries" file are (PlayStation Common conference group), you can download a file called "BBS_VERS.TXT" which lists all the latest versions of software available.

The CD Write Once Drive requires **CD Write Once Generator** (**DTL-S2010**) software which is sold separately.

2.3.2b Sound and Graphic Artist Tools

In addition to the Development System, SCEE provides a range of tools that assist you in creating the sound and graphic effects that match the power of the PlayStation.

The **Graphic Artist Tool (DTL-K2)** consists of a range of PC based and software hardware (running on a PC to the same specification as the DTL-H2000), that is designed to run independently of the Development System. Connectors include 9-pin analogue RGB, S-video and Composite output.

The **Sound Artist Tool (DTL-K3)** provides hardware and software tools to independently perform sound data conversions from popular formats such as SMF and AIFF to the SEQ PlayStation format and to emulate the sound capabilities of the PlayStation. At the time of writing these tools are only supported for the Apple Macintosh (Recommended: a 68040/PowerPC with 16Mb memory and 100Mb hard disk with 1 Nubus slot) and an NTSC Television.

Unlike the Development System, described above, you do not *have to* use Sony proprietary Sound and Graphic Artist Tools to develop PlayStation games, you can use any tool that you prefer. (There are some DOS sound utilities available from the BBS for those who don't use PlayStation Sound Artist Tools.) Obviously, SCEE Developer Support recommend and support only the Sony proprietary tools.

2.3.2c Documentation

SCEE provide a full set of documentation in PDF (Adobe Acrobat) format with the PlayStation Development System Board Set and additional manuals for the CD Emulator (hard copy), CD-ROM Generator (hard copy) and Write-Once Drive (hard copy), Sound Artist Tool and Graphic Artist Tool . As with all things appertaining to PlayStation development, we recommend that you examine the BBS for new versions.

Documentation tends to change frequently, which is why we prefer the electronic versions - however hard copies can be printed out at your site using the Acrobat Reader software provided with the manuals.

There is a list of latest manual versions on the BBS called "BBSCAT.TXT" in the PlayStation Developer Manuals file area.

2.3.3 PlayStation Utilities

Data Converters

A suite of software utilities for 3D and 2D data conversion and manipulation are available. Check BBS_VERS.TXT on the BBS for the latest version. They include:

- **pict2tim** Converts PICT image files to TIM format.
- **rgb2tim** Converts RGP to TIM.
- **timpos** DOS based modification of TIM image & CLUT location in VRAM.
- **timutil** Converts 2D image formats (BMP, PICT, RGB) to & from TI .
- **MovConv** Converts 2D image & AVI into PlayStation movie format.

2.4 Development Tools Problems

(i)General Procedure

If you have difficulties getting your system working, Developer Support recommend the following steps.

1. Check the documentation both here and that provided with the hardware .. twice .. to make sure you none of the installation steps have been skipped.

2. Ensure you have the latest versions of the software component.

3. Check the BBS (using ‘search’) for any notice or discussion of similar problems,

4. If you can, retry the procedure on a different machine, or strip our other cards from your system (PC interrupt conflicts can be tough to pin down),

5. Attempt to isolate the problem to one card or component - try cleaning the edge connectors (observing the normal anti-static precautions) and, if you have two development systems, try swapping cards,

6. If all else fails contact SCEE Developer Support via the BBS (selecting the most appropriate Conference), detailing all the steps you have tried and the exact symptoms of the problem. Download the latest FAULTREP.ZIP.

7. If you do have a faulty piece of hardware, we can arrange to have the hardware sent back here ... upon delivery, we will loan you a new piece of hardware, whilst your original version is repaired. (But please ensure you send back **all** the contents - which includes cables, two controllers and memory cards for Debugging Stations).

(ii)Emulator Specific Diagnosis

The CD-ROM Emulator appears to be one of the most temperamental components of the Development Kit. Some of the problems seem to be linked to the quality of the PC you have installed the hardware in, other problems may suddenly occur after months of happy use. Our additional favourite tips are as follows.

- Try using ‘cddisk -n <scsi-id>‘ to recreate your emulator drive.
- Ensure you are using CDBOOT32.BIN
- Try switching off and on all your hardware. Twice.
- Remove any CDs from your connected PS CD-ROM Drive (you’ll have to select your CD to do this).

3 PLAYSTATION DEVELOPMENT BASICS

3.1 Getting Started on Games Programming

3.1.1 Introduction

This purpose of this section is to steer the novice developer with typically a PC programming or console background away from many of the more common problems that may occur during a project.

(i)How To run a program on a PlayStation

1. After you have edited, closed and saved your program (let's call it `bull.c`), compile it using psymake (see example makefile below):

   ```
   psymake bull.c
   ```

   to make a PlayStation executable file called `bull.cpe`.

2. Reset the PlayStation to stop the program which is currently running and prepare it to download another:

   ```
   rp
   ```

   Now the coloured bars are displayed on the PlayStation monitor.

3. Load the sound data files that your program will require into main RAM:

   ```
   psymake load
   ```

   These files will then be loaded into the PlayStation SPU (sound) RAM by your program when you run it.

4. Run your program:

   ```
   run bull
   ```

Example makefile

```
all:
        ccpsx -O -Xo$80100000 bull.c -obull.cpe,bull.sym

load:
       pqbload ..\simple\sample.vh 80020000
       pqbload ..\simple\sample.vb 80025000
       pqbload mozart.seq          80010000

clean:
      del *.cpe
        del *.sym
        del *.map
```

3.1.1b Update Tools and Libraries

We strongly recommend that you use the latest tools, utilities and documentation as the basis for any new title - you will have received a basic version with your development kit, and perhaps later upgrades, but you should also review the latest BBS_VERS.TXT (listing tools available on the BBS), and BBS_CAT.TXT (listing the documentation) to check if there are any newer versions of tools or libraries to download.

3.1.1c Features of PlayStation Development

PlayStation is very different to the previous generation of home consoles and to the PC. The PlayStation could be said to combine the best features of both.

PC-esque Features

It has a powerful main processor and a comparatively large amount of main memory, it has a mass storage data system (CD ROM) and it supports development in a high level language (C).

Console-esque Features

It contains several dedicated processors to speed up otherwise CPU intensive tasks.

It is standard. With PlayStation the only versions you have to worry about arePAL and NTSC. There are no multiple processor / memory configuration / graphics mode / sound card worries.

The PlayStation is in terms of graphics and audio as good if not slightly better than the most powerful PC and yet costs less than a decent motherboard. This is achieved by using dedicated hardware for many tasks, a dedicated processor can be engineered to provide a far superior level of performance at a lower cost than upgrading an inefficient general purpose CPU.

This means that to get the most from the PlayStation, and the current crop of games are only getting close to using everything the PlayStation has to offer, the developer should make use of the hardware as provided rather than trying to fight against it. This means that porting a game over from the PC could well be more difficult than writing the whole thing again from scratch. This is only true, however, if the developer goes about it in the wrong way.

3.1.1d Quality

The PlayStation sets new standards in home entertainment, not only is it winning over converts from other formats it is also exciting many people old and young is amazing graphics and sound capability which assist in making for absorbing gameplay. To maintain this quality we want our developers to have the best possible start in writing their game.

Dodgy ports of 256 colour 8 bit sound PC streaming games are not going to be approved, derivative racing games the do not look as good as Ridge Racer are not going to be approved, Doom clones that are not better than Doom are not going to be approved. Games with shoddy graphics, poor sound are not going to be approved. Games that do not comply with the Sony guidelines are not going to be approved.

Who wants to waste a significant portion of their life doing something that disappoints people after five minutes of playing. Go on, do something nice.

3.1.1e Designing Your Title

We recommend that early during your design stage, you should review the guidelines provided by the QA and Approvals groups of the territories for which you plan to submit your title.

The latest SCE QA requirements are available on our BBS as **QADOC.ZIP** (with **WARNING.ZIP** providing a template for the ELPSA Anti-Piracy screen required for all European titles).

(i)How To run a program on a PlayStation

1. After you have edited, closed and saved your program (let's call it `bull.c`), compile it using psymake (see example makefile below):

```
psymake bull.c
```

to make aPlayStation executable file called `bull.cpe`.

2. Reset thePlayStation to stop the program which is currently running and prepare it to download another:

```
rp
```

Now the coloured bars are displayed on the PlayStation monitor.

3. Load the sound data files that your program will require into main RAM:

```
psymake load
```

These files will then be loaded into the PlayStation SPU (sound) RAM by your program when you run it.

4. Run your program:

```
run bull
```

Example makefile

```
all:
        ccpsx -O -Xo$80100000 bull.c -obull.cpe,bull.sym

load:
       pqbload ..\simple\sample.vh 80020000
       pqbload ..\simple\sample.vb 80025000
       pqbload mozart.seq          80010000

clean:
      del *.cpe
        del *.sym
        del *.map
```

3.1.2 Debugging Tips

Developers tend to use two main strategies to debug their programs; using the PSY-Q Debugging tool (DBUGPSX) and inserting *printf's* into their code, which are displayed on the PC screen using TESTMESS.

3.1.2a Using CD-ROM Emulator (DTL-H2020) and CD-ROM Drive

Almost all titles need to simulate access to the files as they appear in the final game. Although it is possible to create games using local PC file storage during development and then switch at the end of development. However, using an Emulator or CD-ROM Drive from the start gives more efficient game development.

With your emulator you are given access to a new conference and file area. This file contains - the latter containing a number of essential tools and examples to down load.

The CD Emulator documentation (in Acrobat PDF format, you may wish to print out a copy) and the sample CDREAD.ZIP file on the BBS show how an emulator disk can be filled with data.

Accessing & Switching Between Units (for DTL-H2000)

Select the emulator.
For each process you need to run a specific CPE. You can use a batch file (for example, one called *runselemu.bat* which contains the line **run /w5 selemu.cpe**). (Incidentally, If you also have a PlayStation CD-ROM Drive, make sure that you take out any discs beforehand).

Always reset your development system between use of **Run**.
again you could use a batch file (called, for example, *rp.bat* that contains the lines **resetps** and **run snpath.cpe**).

To switch back to your PS CD-ROM Drive you can also use a batch file (called runselcd.bat which contains the line **run /w5 <pathname>\selcd.cpe**...) remember to **rp** again.

If you wish to run the CD (whether it's physically in the PS CD-ROM, or simply built into the emulator drive) there is another CPE which will read your SYSTEM.CNF file, and start the specified executable - Sony's runcdexec actually holds the line **run <pathname> \cdexec.cpe**. Try this with your copy of *WipEout* as an example....

The CD Emulator is prone to erratic behaviour at times, and sometimes will stubbornly refuse to `rp` correctly. One solution is to power-cycle your machine.

NB The DTL-H2500 uses a different mechanism using a *DESICONS* monitor.

3.1.2b DBUGPSX

Use MAKEFILEs (and PSYMAKE) to control the compilation of your program. Using macro references, it is easy to switch between different compiler options - using **-g**, will produce code that can be debugged at source code level.

When using DBUGPSX, remember the /E switch to allow the program to be run under the debugger!

While the current version of DBUGPSX is DOS based, a Windows95 version will soon be available.

3.1.2c PRINTF & TESTMESS

Developer Support recommend adding switches (e.g. **#ifdef DEBUG**) to your code which can be activated by a compile time flag (e.g. **-DDEBUG**) to introduce run time 'printf' messages. These messages can then be displayed on your PC by starting the MESS1[2] TSR, and then running the TESTMESS command.

One often used technique is to remove all printf statements during the final build with a -DFINAL compiler flag and a *#define printf* in game header files. There is a potential pitfall with this method: when running a final gold disk you may not be able to determine the cause of some problems.

[2] *MESS1 and TESTMESS can be found in the PSYQBIN file area in CODETOOLS.ZIP*

3.1.3 Working with PlayStation Memory

3.1.3a Main memory

Video RAM (VRAM) performs considerably better than main memory. Given this, you should avoid as far as possible operations involving processing large arrays in main memory, or copying from main memory to video ram. To make a game display fast you should use VRAM directly, loading as many of the textures that you will use in the game into VRAM at start up.

The development kit comes with 8 Mb of memory, the PlayStation only has 2 Mb of memory. It is sensible to try and forget the extra 6 Mb of memory right from the start. There is little point in making use of memory that cannot be used in the final machine, as this only means that parts of the game will have to be cut-down or modified to squeeze into less memory right at the end of development. This is not fun to do usually results in the quality of the product suffering.

The extra 6 Mb is for debugging: code compiled without any optimisation will take up far more space in memory than code compiled with optimisation.

Developers would be wise to have an option for debugging and an option with optimisation in their makefile and switch between them as appropriate.

3.1.3b Using Memory

This section explains how the memory of the PlayStation is organised and how it can best be used from within C with the libraries provided.

Main memory can be split into 3 distinct areas; the area used by the program at start-up, the area reserved for the stack and the remainder known as the ‘heap’ from which memory can be dynamically allocated by the program.

Global and static local variables are stored in the program area while volatile local variables are stored on the stack. Obviously then, large local variables are a bad idea as they may result in a stack overflow.

Explanation of the Addressing Scheme

The R3000 uses 32 bit addressing to access main RAM. The physical RAM is logically mapped into the 32 bit address space in several places - ie several sets of addresses map to the same physical RAM. Different sets of addresses mean different things to the processor. Most PlayStation applications will run using the address ranges 0x80000000 to 0x80200000 (2 Mb RAM). Using this set of addresses means that the instruction cache is turned on, and thus your code will run faster. An equivalent, but little used, set of addresses is 0x00000000 to 0x00200000. It is also possible (but not advisable) to run your code in the address ranges 0xA0000000 -> 0xA0200000. However, this set of addresses, while accessing the same RAM as 0x80...., has the instruction cache turned off, and thus is slower - this addressing range is used by the kernel which requires code to run at the same speed at all times, not guaranteed with a cache in operation.

(See the *OS Hardware Guide’s* section on memory management for a detailed explanation.)

Program Origin and the Kernel

The bottom 64 Kbytes of PlayStation main RAM is used by the kernel. The first valid program address, then, is 0x80010000. The program stack usually grows downwards from the top of RAM (0x80200000). The only exception to these address ranges is the development system, which has 8Mb of RAM (making the usual address space 0x80000000 -> 0x80800000).

Configuring Memory on the PlayStation

In the past there have been several approaches to ensuring that a program is running on the development kit in under 2 Megabytes. These included linking with the file 2Mbyte.obj and using the SetMem(2) function. Whilst these functions do work, there is an simpler, more easily understood way of ensuring the program is running within the acceptable environment space.

This is done by simply setting the global variables **__ramsize** and **__stacksize** (provided in *libsn.lib*) at the start of the program.

```
unsigned long __ramsize   = 0x00200000;  //2 megabytes
unsigned long __stacksize = 0x00004000;  //16 kilobytes
```

In this example the ramsize is set to 2 megabytes and the stacksize is set to 16 kilobytes. Setting the stack size merely ensures that at least 16K is reserved for the stack when the program begins execution.

3.1.3c How Programs Are Stored In Memory

A program has 3 parts:

- the program *'text'*, which is the actual code for your game;
- the *data*, which is global variables which you have initialised in you code;
- the *bss* segment which is the space your program needs for un-initialised data (or globals or statics).

Consider the following code, for example.

```
#include <stdio.h>

int numbers[5] =  { 0, 1, 2, 3, 4 };
int myNumber;

int Add(int a, int b)
{
  int temp;
  int max = 0xffffffff;
  static int odd;

  temp = a + b;
  return temp;
}
```

- The *text* section will contain the assembler version of the function (but with no variables).
- The array 'numbers' is stored in the *data* segment (because it must be initialised with the values specified, which are stored as part of the executable).
- The integer 'myNumber' is stored in the *bss* section (because it is not initialised).
- The variable 'temp' is created on the stack at run time, and so is not part of the executable segments.
- The variable 'odd' is stored in the *bss* section too, because it is static and must keep its value across function calls.
- The variable 'max' is actually created on the stack at run time too, and initialised with the value given to it at function entry time (so this is actually more expensive in terms of execution time than using a global, which is initialised at program start up, or the best case of all, using a #defined value).
- The heap addresses will be in kseg0 (cacheable memory), even though the data cache is not automatically used.

So the size of the heap at program start is this:

- 2 Mb (Main RAM size)
- 32k (standard stack)
- 64k (kernel RAM)
- (*text* length + *bss* length + *data* length (your program))

The length of the segments and their positions in memory can be obtained from the map file for the executable. To generate a map file add a map file to the -o section of the makefile command line, i.e.

```
mem.cpe: mem.c
        $(CC) $(OPTIONS) mem.c -omem.cpe,mem.sym,mem.map
```

The mapfile will then have a section in it that looks as the mapfile below.

```
Start     Stop   Length      Obj Group            Section name
80010000 800100AF 000000B0 80010000 text            .rdata
800100B0 80010607 00000558 800100B0 text            .text
80010608 80010633 0000002C 80010608 text            .data
80010634 80010647 00000014 80010634 text            .sdata
80010648 8001064B 00000004 80010648 bss             .sbss
```

Extract from Map File

The SDATA And SBSS Sections

The Mips compiler has an option to put small data items (e.g. <= 8 bytes) into special sections called .sdata and .sbss rather than the usual .data and .bss. The global register (gp) is set up at the start of the program to point to the base of these sections and variables are then accessed as offsets from this pointer rather than as absolute addresses. This can speed the code up considerably when you're accessing lots of global variables and also reduce the size of the code.

To use it you add the option -mgpopt to your **ccpsx** command line,

e.g.

```
ccpsx -g -c -mgpopt main.c
```

By default, this will place any data item of size 8 bytes or less in the .sdata/.sbss sections. You can override this size with the -G switch, as shown below.

```
ccpsx -g -c -mgpopt -G16 main.c
```

The -G switch will place any item of size 16 bytes or less in the .s... sections.

If you specify -G0 then no data will be placed in the .s... sections and the optimisation will be disabled. (Note : even if you do not specify the -mgpopt option the compiler may still place some variables in the .sdata/.sbss sections and the assembler will generate gp relative addressing modes to access them. It is therefore necessary to specify -G0 to completely disable this).

Because the size of an offset from the gp register is limited to 16 bits the total size of the .sdata and .sbss sections is limited too. If the sections grow too large then you will get "Illegal value" type errors at link time.

There is one problem in using this feature. In an event handler the GP register will be set to point to the kernel's data area rather than the main program's. This is for efficiency reasons to reduce the time taken to get into the event handler. It is therefore important to compile any event handler routines separately and to specify the -G0 option when you do so. You may also need to compile program overlays depending on the method you use to achieve overlaying.

Dynamic Memory Allocation and the Heap

Memory allocation functions allow the programmer to increase the depth and variety of the game world whilst making the best use of the PlayStation’s relatively modest memory size.

In C, memory allocation is achieved using the *malloc* and *free* functions. Unfortunately there have been several problems with these functions on PlayStation.

The standard *malloc*/*free* combination supplied as part of *libc* fragments memory due to a bug in the free function. This means that large chunks on the machine memory become inaccessible even though they are not holding any valid data.

There is a patch for this bug which is used by linking with a file called mmgm.obj (available from the SCEE BBS). This supplies a new function for initialising the heap and replaces the buggy malloc and free with more reliable functions. In this scheme the heap base is unconventionally located at the address ramsize minus the stacksize and the heap "grows" toward the base of memory.

Whilst these functions are better than the originals, they are still comparatively untested. Also they are rather generic. You are **strongly** recommended to consider developing a memory allocation scheme that is tailored to your specific application, rather than relying on those supplied in the libraries.

An example showing how to implement a simple malloc/free scheme is presented in the Kernighan & Ritchie book *The C Programming Language*. This text notes that the optimal memory allocation scheme depends on the nature of the application.

The Stack

The stack starts at the top of main memory and works down. The stack may not exceed its defined size. The stacksize can be defined from within the program using the **__stacksize** variable.

3.1.4 Caches

3.1.4a I Cache

The instruction cache is a conventional direct mapped 4 kilobyte cache. As a result it is inefficient when executing large loops, or loops with function calls in them. This is one of the reasons why many of the GET functions are now provided as inline statements rather than function calls.

3.1.4b D Cache

The data cache, however, is 1 kilobyte of fast RAM built into the CPU to be used by the developer to improve the performance of their product.

Some developers have successfully placed their stack on the data cache. While this produces a useful speed increase, it is not as straight forward as it might first appear because the data cache does not provide the space that is usually required for a stack. Attempting to read an address below the lowest value of the data cache (remember the stack grows downward) will result in garbage.

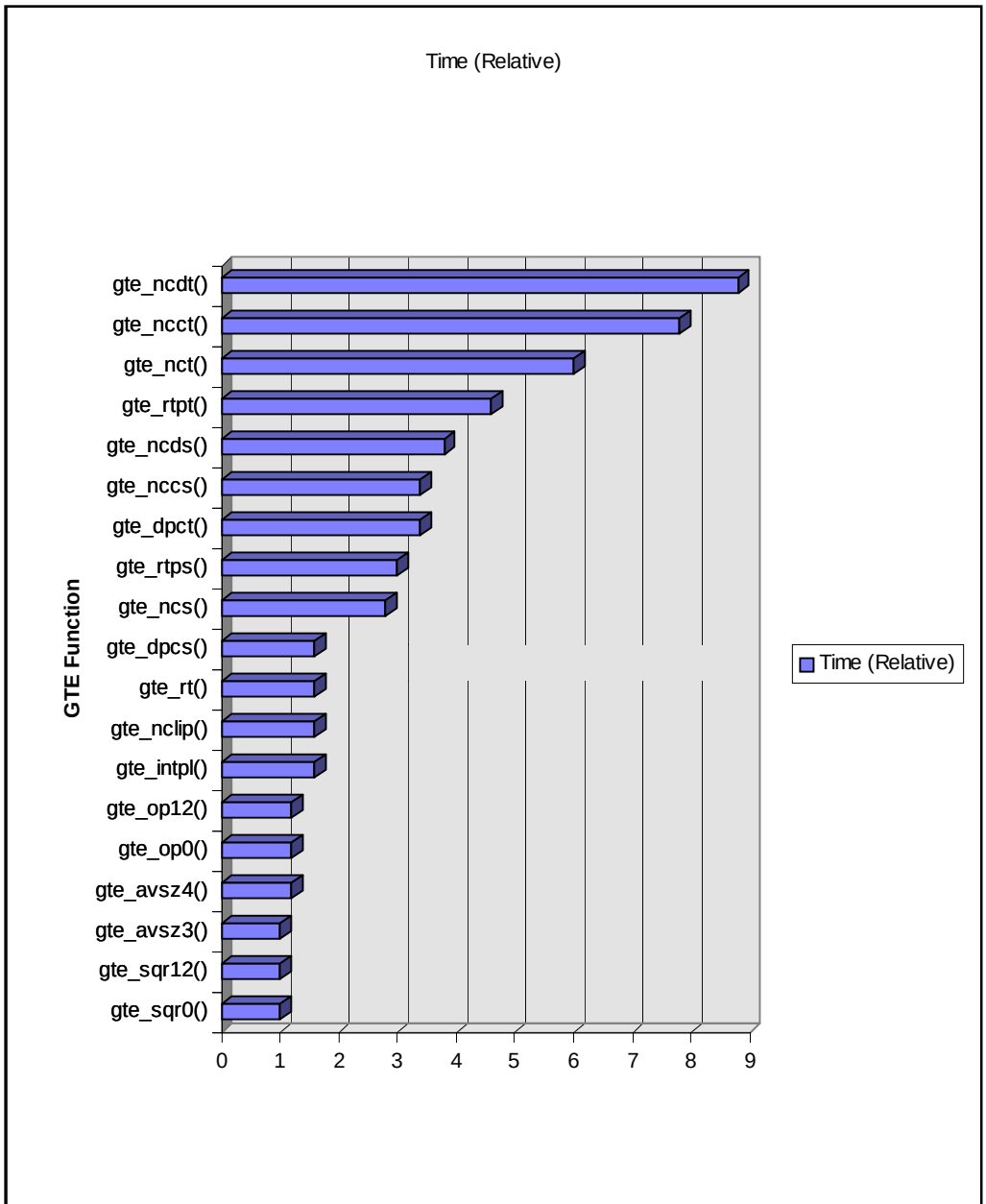
Placing frequently used variables in the data cache is a safe and efficient way to produce fast code. An example of this is the ordering tables, if the ordering table is small enough, it can be placed in the data cache. Reading from the data cache is approximately four times faster than main memory.

To obtain the base address of the data cache use the macro getScratchAddr(off) where off is an offset of 0. This macro is in libetc.h

3.1.5 The GTE

The GTE is a fast maths coprocessor attached to the PlayStation's R3000 (the main processor). It provides the 3D calculation power which makes 3D games on the PlayStation so fast. The chip provides a variety of services at hardware level dealing with rotation, translation and perspective transform of points, lighting, fogging and depth cueing calculations, linear interpolation and various matrix and vector functions. Many of these services are much faster than a single R3000 multiply or divide. It is important to distinguish the GTE from the GPU - the GTE is responsible for the maths involved in generating a 2D picture from a 3D world, and the GPU is responsible for actually drawing the polygons in the picture.

The following graph shows the relative timing for GTE functions. DMPSX.ZIP provides information and example code for using in-line assembler calls to these functions.



Relative timing for GTE functions

3.2 CD-R on The PlayStation

(See *Beginner's Guide to Sound* section in this document for a simple) explanation of data/sound storage on PlayStation CD).

The PlayStation is fitted with a dual speed CD ROM reader providing continuous data transfer rate of 300 kilobytes per second at dual speed or 150 kilobytes a second at single speed.

3.2.1 How much space is there on a PlayStation Disc ?

The standard PlayStation CD, played at normal speed can be up to 71 minutes, 59 seconds and 74 sectors long (so not quite 72 minutes). This gives a total capacity of 624Mb:

71(minutes) x 60 (seconds) = 4260 (seconds total length)

4260 (seconds) x 75 (sectors per second) + 74 (sectors) = 319574 (total sectors on the CD)

319574 (sectors) x 2048 (bytes of data storage in a sector) = 654487552 (bytes)

= 639148 (Kb)

= **624 (Mb)**

How much room can you use ? Well, there will be some overhead for each DA track you have (allowing for gaps), and you have to be careful to organise your disc to avoid seeking in the last three minutes of data (see QADOC.ZIP).

3.2.2 Organisation of a PlayStation CD

The data on a CD is arranged in a spiral, much like a record, except that on a CD the data starts at the centre and works out toward the edge.

The disc is divided logically into tracks, the first track is called the lead-in track and the last track is called the lead out track.

There can be up to 99 data tracks. The lead in track contains the CD's table of contents. The TOC (table of contents) details the position of all the tracks in terms of an offset from the start of the disk.

On PlayStation the rest of the CD can be neatly divided into two sections. Track 1 will contain all of the PlayStation specific information, such as the executable and any data that it should require. Other tracks (2-99) may hold standard CD audio files.

There are limitations on the maximum number of files and directories allowed on a PlayStation disc - roughly 40 directories, each with a maximum of 30 files. The actual limits depend on the physical length of the filenames. Known Symptoms - *cdSearchFile* will fail/ CD ROM Generator won't allow disc to be created.

One caveat - although CD-DA tracks are physically not in track 1, they do appear in the disc's Table of Contents. The advantage is that you can use cdSearchfile to find them - the disadvantage is that they impact on the number of files per directory limitation.

3.2.3 Data Storage on PlayStation

The PlayStation uses standard Phillips' *White Book* formats to store data on CD (Standard Data Format 'Frames', XA and XA-ADPCM). These are listed below.

One sector (one 75th of a second) of CD holds 2352 bytes in total (if it were storing straight Digital Audio data.)

3.2.3a Standard Data Formats

Mode 0 Frames

Sectors in Mode 0 are ignored by the CD reader so this format tends to be used (if it is used at all) to hold empty sectors.

Mode 1 Frames

This Mode has 2048 bytes of data available per sector. The remaining 304 used for a CD reader header (sector synchronisation and identification) and a CRC (cyclic redundancy check) with error correction. This format is typically used for items requiring a high level of data integrity such as program executables.

Mode 2 Frames

Similar to Mode 1, this looses 16 bytes to a header but leaves 2336 bytes of data available. As this mode has no error correction, it is best used for non-critical data such as sound which can tolerate an error rate.

For audio, however the subset of Mode 2 Frames, XA Form 2 or its subset XA ADPCM is more usually used - see below.

Mode 2 Frames is 14% faster in loading data than Mode 1 Frames.

3.2.3b XA Frames

XA Frames is an extension of the standard frame types listed above to allow interleaving of data. It is specifically as subset of Mode 2 Frames.

XA Form 1

This mode has 2048 bytes of data per sector and includes automatically CRC (cyclic redundancy check) and error correction. It tends to be used for video and audio interleaving.

XA Form 2

This has 2324 bytes per sector available for data storage with no error correction. This could be used for video storage but as there is no error correction it is not recommended. It tends to be used to store audio data, although the specific XA Form 2 audio storage mode 'XA-ADPCM' is that most often used

3.2.3c XA-ADPCM

XA ADPCM (eXtended Audio Adaptive Differential Pulse Code Modulation) is a sub-mode of *Mode 2 Frames* and *XA Form 2* data storage mode which is specifically for audio data as opposed to any data (code, video or audio) of the storage modes listed above

It is a compression technique that can be used to provide near CD-DA quality sound, with a level of compression.

A 37.8Khz stereo XA ADPCM file only takes up one quarter of the space as the same data stored as a DA track. However this data is spaced on the disk to use every fourth sector, leaving 75% of the CD blank. This provides a useful method to interleave tracks - for example four XA-ADPCM tracks interleaved or video streams interleaved with one XA-ADPCM track.

(See *Beginners' Guide to PlayStation Sound* in this document for a more detailed explanation.)

3.2.4 Working With Data Storage on the PlayStation

3.2.4a Playing XA on the PlayStation

Ingredients

- *Movconv* 1.96e or better
- Extract.exe
- Some WAV files. (16bit Stereo with a high sampling rate).

A sample rate of 37.8Khz (not a standard WAV frequency) was used in the following example.

There may be some kind of limited frequency conversion within *movconv*, but this should not be relied upon.

Do not use 8 bit WAVs. These are unsuitable.

Preparation

1. Use movconv scripting facility to convert the WAV files to .XA files, 37.8 Khz stereo.

   Example Script to convert a WAV to an XA file:

```
Wav2xa(   c:\data\wav\tomtom.wav,       # Input file name
\data\wav\da4.xa,       # Output file name
37.8KHz,                # Frequency of xa audio
Stereo                  # Stereo or Mono for xa audio
);
```

2. Remove the sub-header from the file using the extract program.

   Example DOS command line:

```
c:\data\wav\da4.xa c:\data\wav\d4.xa
```

   The output file d4.xa is now ready to be interleaved.

3. Interleave the files.

   The interleaving process depends on the final application for the file. The following table shows what is possible. The example presented here is designed to be played at single speed and has 37.8Khz stereo channels. The table shows that the XA channel must occur once every four sectors on the disk.

   One advantage of XA-ADPCM over DA is that more data can be stored on the disk. In the example presented four channels are interleaved for this purpose

Double Speed CD, 37.8KHz, Stereo:	**8**
Double Speed CD, 37.8KHz, Mono:	**16**
Double Speed CD, 18.9KHz, Stereo:	**16**
Double Speed CD, 18.9KHz, Mono:	**32**
Normal Speed CD, 37.8KHz, Stereo:	**4**
Normal Speed CD, 37.8KHz, Mono:	**8**
Normal Speed CD, 18.9KHz, Stereo:	**8**
Normal Speed CD, 18.9KHz, Mono:	**16**

3.2.4b Buildcd

Buildcd can be used in two ways; to both create files on the Development System's CD Emulator for development and testing purposes, and to dump output to the PC's hard disk without any sub-header information.

This is useful as these files can then be copied around on the PC filing system and then eventually be used to burn the final CD.

Note: Because of the unusual size of the sectors used by XA-ADPCM, it is not possible to copy these sectors using DOS. This means that you cannot copy an XA-ADPCM file from a CD on to a PC easily.

3.2.4c How To Force XA Output To The PC Filing System

Firstly ensure that there is a PC path in addition to the Emulator path for an the file you require in the .CTI file. For example:

```
XAInterleavedFile  even.xa    c:\dave\even.xa
```

Note that neither of the files have version numbers. Do not add version numbers to XA-ADPCM files as this will cause an error to occur.

Then run the Buildcd program with the -g command line option, specifying an output file name must (required with the -g option). This is the output for a .CCS file (format of the CD ROM Generator configuration/script language file). You can disregard this output file as it is not needed in this example.

Example running build cd program.

```
Buildcd xagen.cti -s0:1 -gwaster.ccs
```

(See the *CDROM Emulator* manual for more information on how to use Buildcd and sample scripts. Further samples and examples are available on the BBS.)

3.2.4d Example CTI files for XA Interleaving

The following two examples are CTI files to create interleaving.

Create an interleaved XA file with 4 channels

```
Disc XA_PSX

MapFile xa_int.map

LeadIn  XA
        Empty   1000
        PostGap 150
EndTrack

Track XA
        Pause   150
        Volume ISO9660
PrimaryVolume

    SystemIdentifier      "PLAYSTATION"      ;required indetifier
    VolumeIdentifier      "DEMO"             ; app specific identifiers
    VolumeSetIdentifier   "DEMO"
    PublisherIdentifier   "SCEE"
    DataPreparerIdentifier "BJF"
    ApplicationIdentifier  "DEMO"

    Lpath
    OptionalLPath
    Mpath
    OptionalMPath

    Hierarchy

     XAFileAttributes  Form1  Audio
     XAVideoAttributes ApplicationSpecific
          XAAudioAttributes ADPCM_C Stereo

          XAInterleavedFile  even.xa       c:\gamedata\even.xa
```

```
        XAChannelInterleave TimeCritical 1-2-3-4

        XAChannel       1
          XAFileAttributes   Form2  Audio
          Source          c:\data\wav\d1.xa        ;put your xa file here
          MinLength 270000   ;note this is the length of the longest of the 4
                ;tracks being interleaved
        XAEndChannel

        XAChannel       2
          XAFileAttributes   Form2  Audio
          Source          c:\data\wav\d2.xa  ;put your xa file here
          MinLength 270000
      XAEndChannel

        XAChannel       3
          XAFileAttributes   Form2  Audio
          Source          c:\data\wav\d3.xa        ;put your xa file here
      MinLength 270000
        XAEndChannel

        XAChannel       4
          XAFileAttributes   Form2  Audio
          Source          c:\data\wav\d4.xa        ;put your xa file here
      MinLength 270000
        XAEndChannel

      XAEndInterleavedFile
    EndHierarchy
  EndPrimaryVolume
 EndVolume

EndTrack
LeadOut  XA
 Empty   150
EndTrack
EndDisc
```

Interleaving video and DA elements

```
Disc XA_PSX     ;disk format

  LeadIn XA             ;lead in track, track 0

    Empty       1000    ;defines lead in size min (150)
    PostGap     150     ;required gap at end of lead in

  EndTrack              ; end of lead in track

  Track XA                      ;start of XA (data) track

    Pause      150              ;required pause in first track after lead in

    Volume          ISO9660     ;define ISO 9660 volume
    PrimaryVolume               ;start point of primary volume


    SystemIdentifier        "PLAYSTATION"       ;required indetifier
    VolumeIdentifier        "PSXTEST"           ; app specific identifiers
    VolumeSetIdentifier     "PSXTEST"
    PublisherIdentifier     "SONY"
    DataPreparerIdentifier  "SONY"
    ApplicationIdentifier   "SONY"

    Lpath               ; Path tables as specified for PlayStation
    OptionalLpath
    Mpath
    OptionalMpath

    Hierarchy           ;start point of root directory definition


     XAInterleavedFile dixlogo.str C:\DD\movies\dixlogo.str
```

```
        XAChannelInterleave TimeCritical 1-1-1-1-1-1-1-2

        XAChannel 1
          XAFileAttributes Form1 Video
            Source C:\DD\str\dixlogo.str
         XAEndChannel

         XAChannel 2
           XAFileAttributes Form2 Audio
          XAAudioAttributes ADPCM_C Stereo
           Source  C:\DD\newxa\sixty.xxa
         XAEndChannel

       XAEndInterleavedFile

       File  MARKER0.STR;1
         XAFileAttributes  Form1 Video
         Source  C:\DD\movies\mark2.str
       EndFile

      EndHierarchy        ;ends root directory definition

      EndPrimaryVolume    ;ends primary volume definition

      EndVolume           ;ends ISO 9660 definition

      PostGap  150        ;required to change track type

    EndTrack              ;ends track definition


    LeadOut  XA        ;required lead out track (must match previous track
type)
      Empty  150          ;required minimum lead out
    EndTrack              ;ends track definition

  EndDisc ;ends disk definition
```

3.2.4e MovPack (interleaved Audio Technique)

To create an interleaved XA file using MovPack:

- Take the individual XA channel files.
- Do not extract the sub headers from these files.
- Use MovPack to merge the files. The input and output sub-header check boxes should be selected for all input channels and the output channel.
- Terminate with null sectors button should not be checked.
- As this technique does not allow as great a level of control as Buildcd, this method works best when the samples are of similar lengths.

3.3 Getting Started on Sound

3.3.1 Sound Basics

3.3.1a Sound and Your Dev Kit

(i) Sound and PlayStation Memory

Running a program with sound usually requires more than one file: the executable program (the normal PlayStation's `.cpe` file) and its associated sound files.

The R3000 is the PlayStation's CPU. The PlayStation itself has 2MB of main RAM while the development kit has 8 Mb of main RAM.

The SPU (Sound Processing Unit) which manages PlayStation sound has its own SPU/sound RAM of 512 KB. This is not accessible via your PC so files to be used by the SPU must first be loaded into main RAM so that your program can move them to the SPU RAM.

The command *`pqbload`* loads sound files into PlayStation main RAM. This command may be in the *makefile* called when you use *`psymake`* `load`. So you can use either.

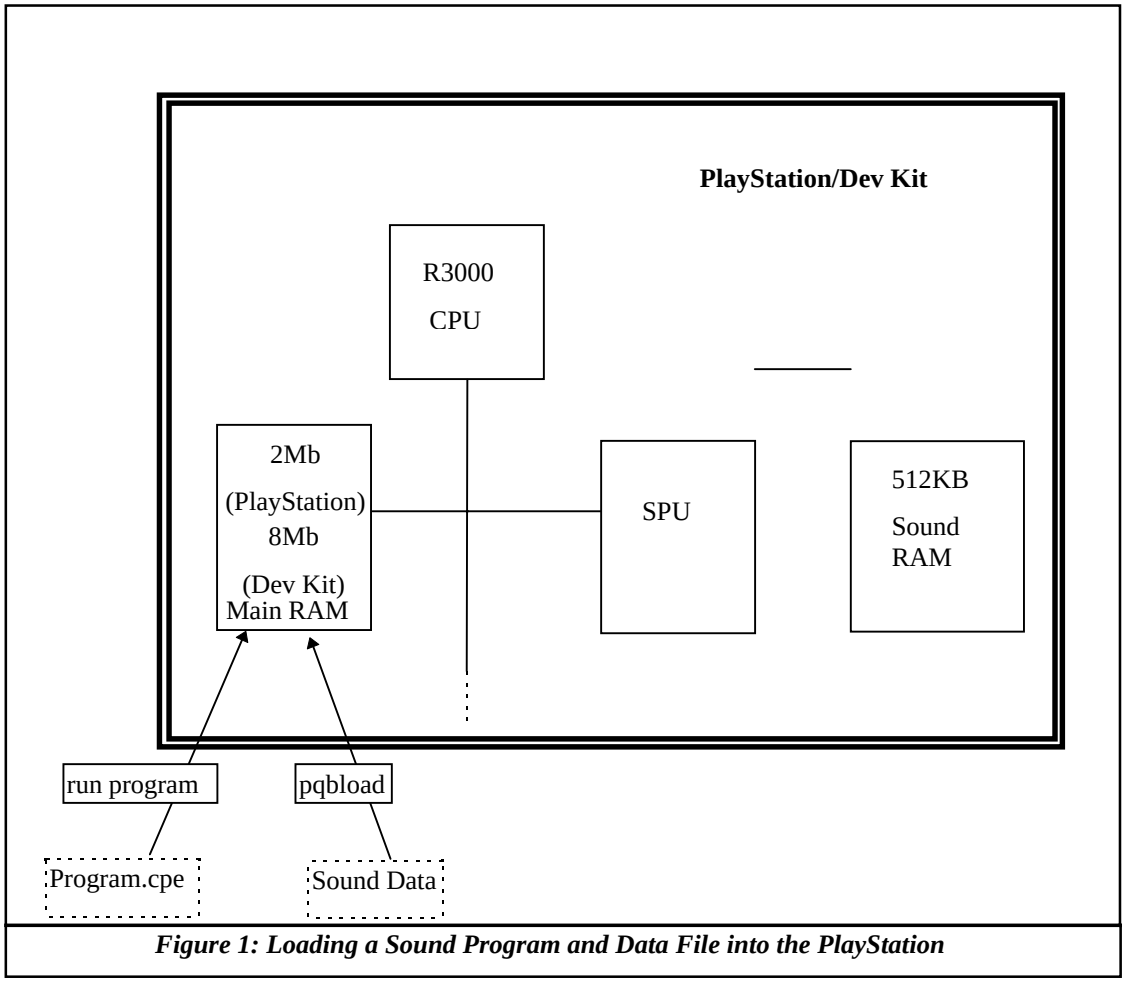


Figure 1: Loading a Sound Program and Data File into the PlayStation

(ii) Sound Files for the PlayStation SPU

VAG - a sound sample (a snippet of sound stored in digital format)

VAB - Used by the SPU, it is a collection of VAGs. A VAB file is sometimes split into a VB file (body) and a VH file (header).

SEQ - A sequence file defines a list of VAGs. To be played from VAG files. The Sequence file defines the order and speed that the VAGs will be played at.

SEP - A collection of SEQs.

3.3.1b CD Sound Storage

(i)How a CD Stores Sound

Sound is a digitised for storage on a CD.

Taking a sound wave (figure 2), between the points A-B are a complete wave.

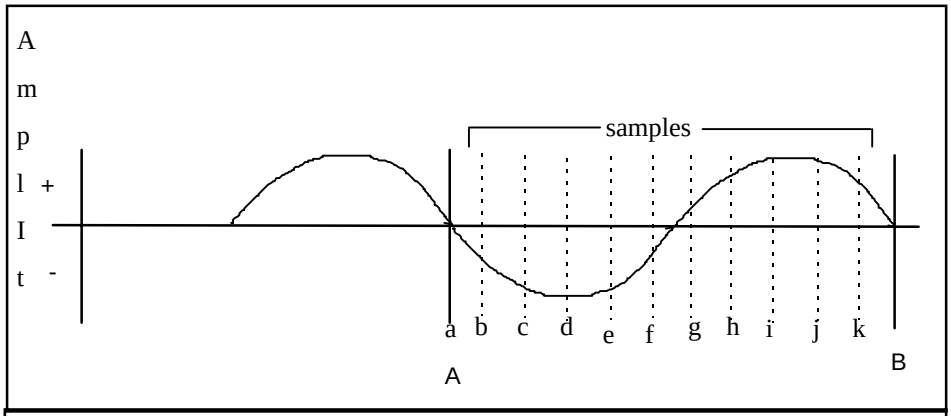


Figure 2: Digitising a sound by measuring its amplitude at discrete points.

Sound is digitised by measuring the amplitude of the wave at discrete intervals (a->k in figure 2) to get a list of values. In the simplified example above, this would be:

a = 0, b = -1, c = -2, d = -2.5, e = -2, f = -1, g = +0.5, h =+2, i = +2.5, j = +2.5, K= +2

The resulting digitised sound wave wound be a list as follows:

0 | -1 | -2 | -2.5 | -2 | -1 | +0.5 | +2 | +2.5 |+2.5|+2

In reality the values can be anything between -32767 and +32767. Each value is called a ‘sample’ and the number of samples a second is the ‘sample rate’. Normal CD (DA sound) is 44100 samples per second, expressed in Kilo Herz that is 44.1Khz.

A low sample rate will give a poorer sound quality as the recreated sound wave will not be very true to the original.

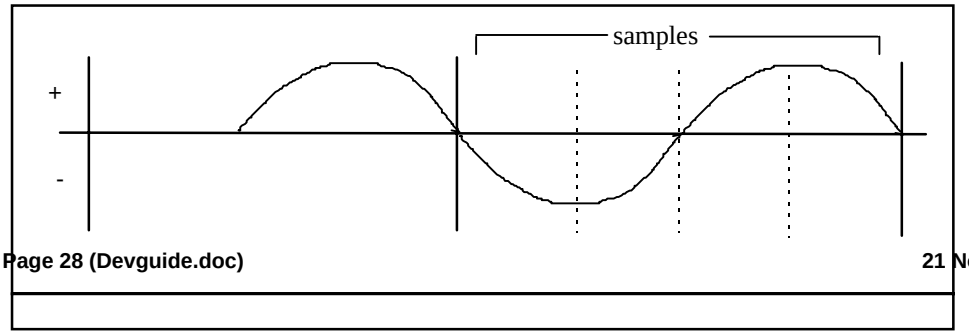


Figure 3: Low Sample Rate

The digitised sound from this is:
0 | -2 | 0 | +2

The recreated sound wave will be:

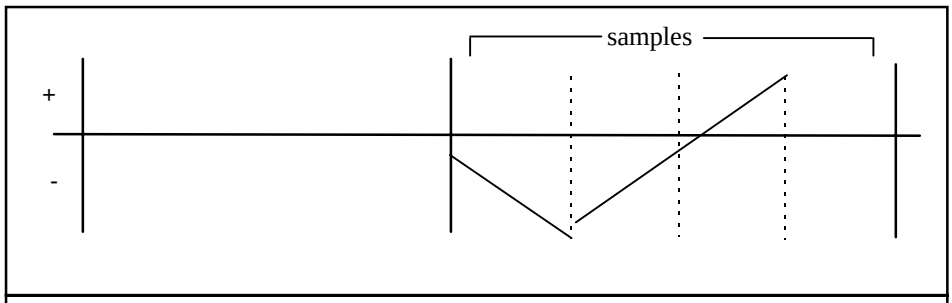


Figure 4: Sound Reproduction form a Low Sample Rate

Because the human ear can hear different sound timbres, some sound can be stored with a low sample rate - bassy sounds or drums for example.

(ii)CD Tracks - Space Available

While data is physically stored on the CD in a spiral track, there can be between one and ninety nine conceptual 'tracks' on a PlayStation CD which are like individual songs on a normal audio CD (or vinyl record or magnetic cassette).

Each track is a certain number of minutes and seconds long in time. Each second of playtime is divided into 75 sectors. If a sector is used to store data, it can hold 2Kbytes (or 2048 bytes).

So, one second of CD must be 150Kb of data as:

1 (second) x 75 (sectors) x 2 (Kb) = **150 (Kb)**

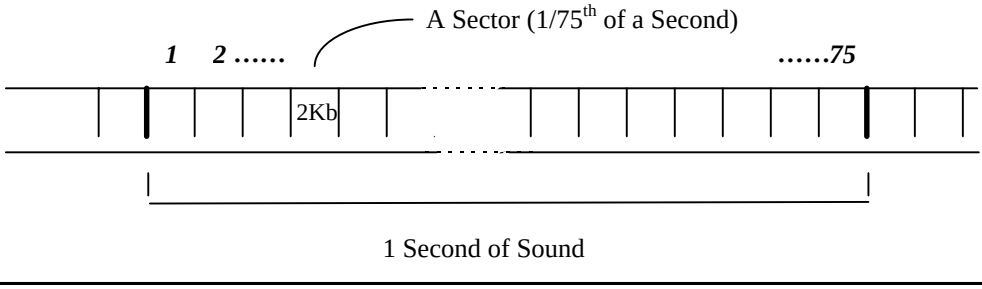


Figure 5: A Second of DA Sound has 75 Sectors

The PlayStation does have a dual speed CD ROM reader which can give continuous data transfer rate of twice the 150 Kb per second as well. (That is 300 kilobytes per second at dual speed.)

(See section 3.2.1 *How much space is there on a PlayStation Disc ?* 3.2.120)

This is the space when using XA Data Storage for programs. Called XA - Form 1, the 2Kb (2048 bytes) of storage space in each section is sandwiched between a header which announces the storage type and a CRC (Cyclic Redundancy Check) error check.

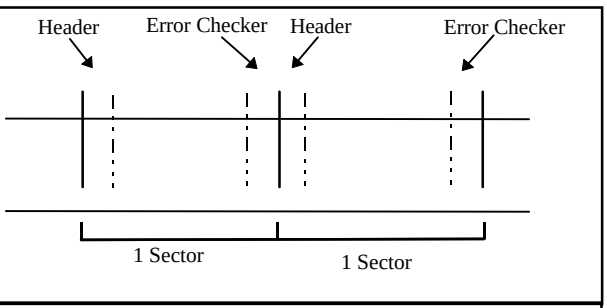


Figure 6: A Second of DA Sound has 75 Sectors

When storing certain types of sound, however, this space for the header and CRC is not always required. This space, then, can be used to store sound data too. Thus there can be an extra 288 or 304 Bytes available per sector for sound storage (depending on the type of sound stored, see *Types of Sound*, later)

(iii)Which Sampling Rate?

The important thing to remember here is that:

INCREASE in SAMPLING RATE = INCREASE in QUALITY + INCREASE in SIZE.

So there is a cost to higher quality, that is more memory use. It is important, then, to ensure you use the most appropriate sampling rate.

For ordinary CD (DA) tracks on the PlayStation, this is fixed at 44.1 KHz which gives sound at the highest quality that the human ear can detect. (According to Shannon's rule, in order to reproduce sound accurately, the sampling rate must be over twice the 15 - 20 KHz that is human hearing.)

For the other sorts of sound on the ***PlayStation*** (really SPU-ADPCM), the sampling rate is variable. To make things simple we can use multiples of the 44.1KHz to get some idea of the achievable quality (see Figure 7).

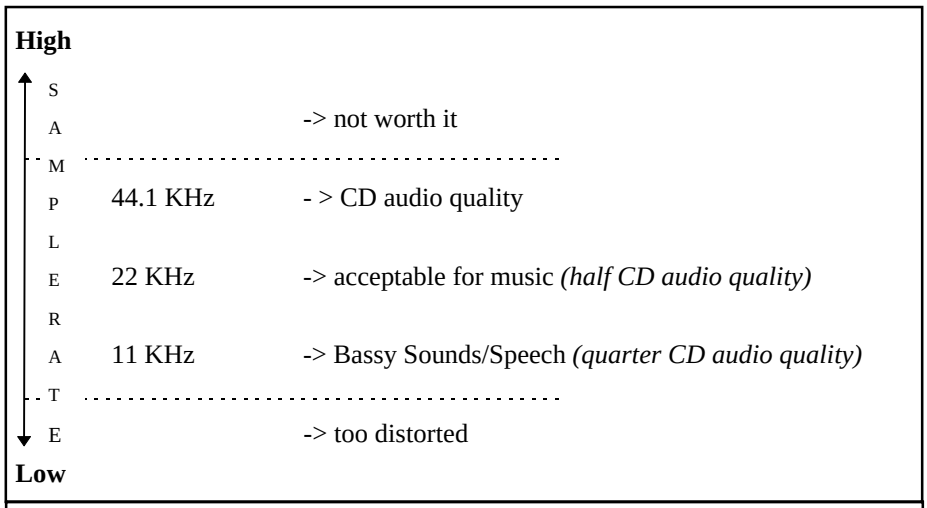


FIGURE 7: Sampling Rates and Their Sound Quality

3.3.2 Sound Types On The PlayStation

3.3.2a Simple List

The first two types of PlayStation sound listed here (DA and XA-ADPCM) are played directly in real time by the CD sub-system. The third, SPU-ADPCM, is a data type processed by the PlayStation's SPU.

- ***DA (Digital Audio)***

  DA is 'normal' sound that you get from in your CD player. It plays at 44.1 KHz and uses the full 2352 bytes of a sector in CD storage as it has neither header nor error checking.

  This is used on the ***PlayStation*** for background music, like that on Ridge Racer.

- ***XA- ADPCM (eXtended Audio -Adaptive Differential Pulse Code Modulation)***

  This is a type of Mode 2 Frames, XA From 2 data storage format. Compressed audio to approximately 25% of DA. It can be either:

  KHz stereo or mono or 18.9 KHz stereo or mono and uses 2336 bytes of a sector in CD storage as it has header but no error checking.

  This is used for video and music or, more usually in PlayStation, for a pre-rendered sequence and music - like the introduction to Worms, for example.

- ***SPU-ADPCM (Sound Processing Unit -Adaptive Differential Pulse Code Modulation)***

  Another compressed sound proprietal to Sony, this can be played at any speed - usually somewhere below 44.1 KHz. A data type, it is not expressed in terms of sector size as in XA and DA.

  This is used for sound effects in a game which can happen at any time - such as the car engine in Ridge Racer which changes pitch as you accelerate.

3.3.2b DA - The Basic Sound Store

As the ***PlayStation*** executable Program lies in track 1 of a ***PlayStation*** CD (holding data like pictures, 3D models and Sound SPU RAM data), you can store simple DA in later tracks on the ***PlayStation*** CD.

DA goes straight to playback, just like a normal CD player.

DA plays at 44.1 KHz. (To reiterate: there are 44,100 samples - in the form of numerical values - every second. This gives the best quality that the human ear can detect.) Each sample requires a 16 bit memory space to be stored being a value of between -32767 and +32767 (as described in *How a CD Stores Sound*, above).

As DA can be stereo, there is a left ('L') and right ('R') channel and a sample value for each, so:

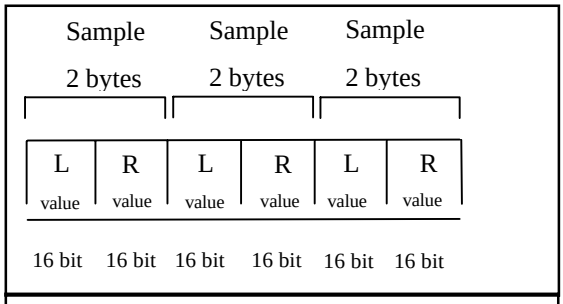


Figure 8: Stereo DA

This fits into the space available for one second of space on a CD (as described in *CD Tracks*, above).

For each of the 44,100 samples in a second, a 16 bit storage space is used for the left and a 16 bit for the right (16 bits = 2 bytes).

2 (bytes) x 2 (channels) x 44100 (number of samples) = 176400 bytes

DA data on CD is not error corrected. (If one of the sample values on the CD is read badly, it is not important as it represents only 1/44100 of a second of bad sound - not really likely to be a problem). So there is some additional space is available for DA: a DA audio sector is 2352 bytes per sector, rather than 2048 in XA Data Form 1.

So, in one DA sector there (2352 bytes) there are 1176 samples of 16 bits (2 bytes):

2352 (bytes space) / 2 (bytes per) = 1176 samples per sector.

If we remember that there are 75 sectors a second and 2 channels (left and right) we get the magic number 44,1000 samples per second:

1176 (samples per sector) x 75 (samples per second) = 882000 samples a second) / 2 (channels) = 44,1000 (samples per second)

3.3.2c XA - ADPCM - Compressed Sound and Interleaving

While a normal audio CD player plays the DA track, as if it were a continuous list of sound sample data (see Figure 9).

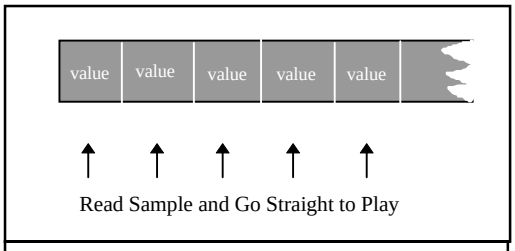


Figure 9: DA Sound - A Continuous List

XA-ADPCM is compressed and so a given amount of sound data can be squashed into a smaller amount of CD space. The PlayStation has the capacity to spin the CD twice as fast as a normal audio CD (300 Kb per second at double speed). So using this double speed play and compression together, the ***PlayStation*** can achieve continuous sound. It need only read the compressed data at intervals, playing what it has collected at each read until the next read. (See Figure 10)

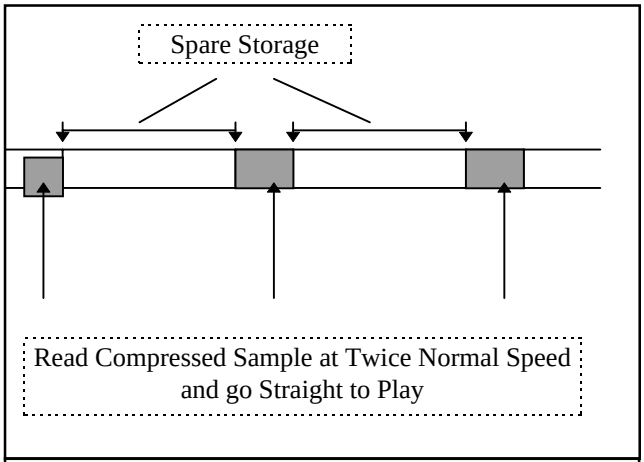


Figure 10: XA- ADPCM Compressed Data Samples are Read at Twice the Speed of DA

This gives the space for extra storage of video or pre-rendered sequences so it can be viewed while sound is played. The Sound Sample is said to be 'interleaved' with the video sample. (See Figure 11)

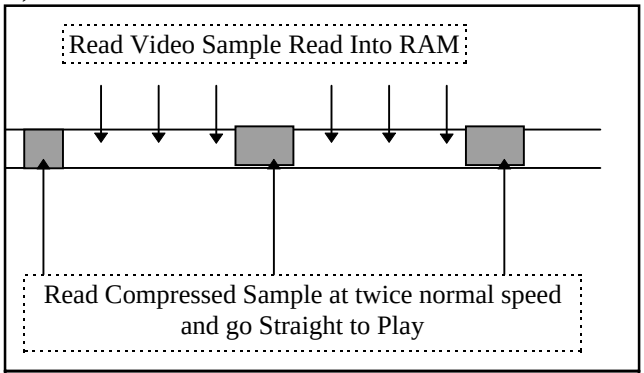


Figure 12: Video Samples Interleaved with

The interleaved sound sectors must be regular enough to give unbroken sound - so their interval depends on the compression ratio of the data. The interval must also take account of stereo or mono sound. If we bear in mind that XA - ADPCM sound can be 37.8 or 18.9 MHz, this sound (with a sample rate of say, 37.8 Mh mono has an interleave of one in every sixteen sectors. The same sample rate in stereo with two channels and twice as much data requires an interleave of 7 in every 8 sectors.

An XA-ADPCM sound sample takes approximately one quarter of the CD space if a DA sample - that is approximately 4 bits.

3.3.2d SPU-ADPCM: Changeable Sound

SPU - ADPCM is Sony proprietary compressed sound. Unlike DA and XA-ADPCM, this sound is loaded from the CD into SPU sound) RAM. Pieces of sound for use by the SPU RAM are called sound samples (confusingly this is the same name as the individual value used in storing sound digitally - so here they will be 'ssamples' to ease confusion).

A ssample can be held in the SPU RAM and replayed in loop, but at each play its pitch can change, simply because the SPU can play a ssample at any speed.

So the sound of a car revving in Ridge Racer is that same sample in a loop but being played at a faster and faster speed to give a higher and higher pitch. This change in pitch is called a 'pitch bend'.

This is (again) simple physics :

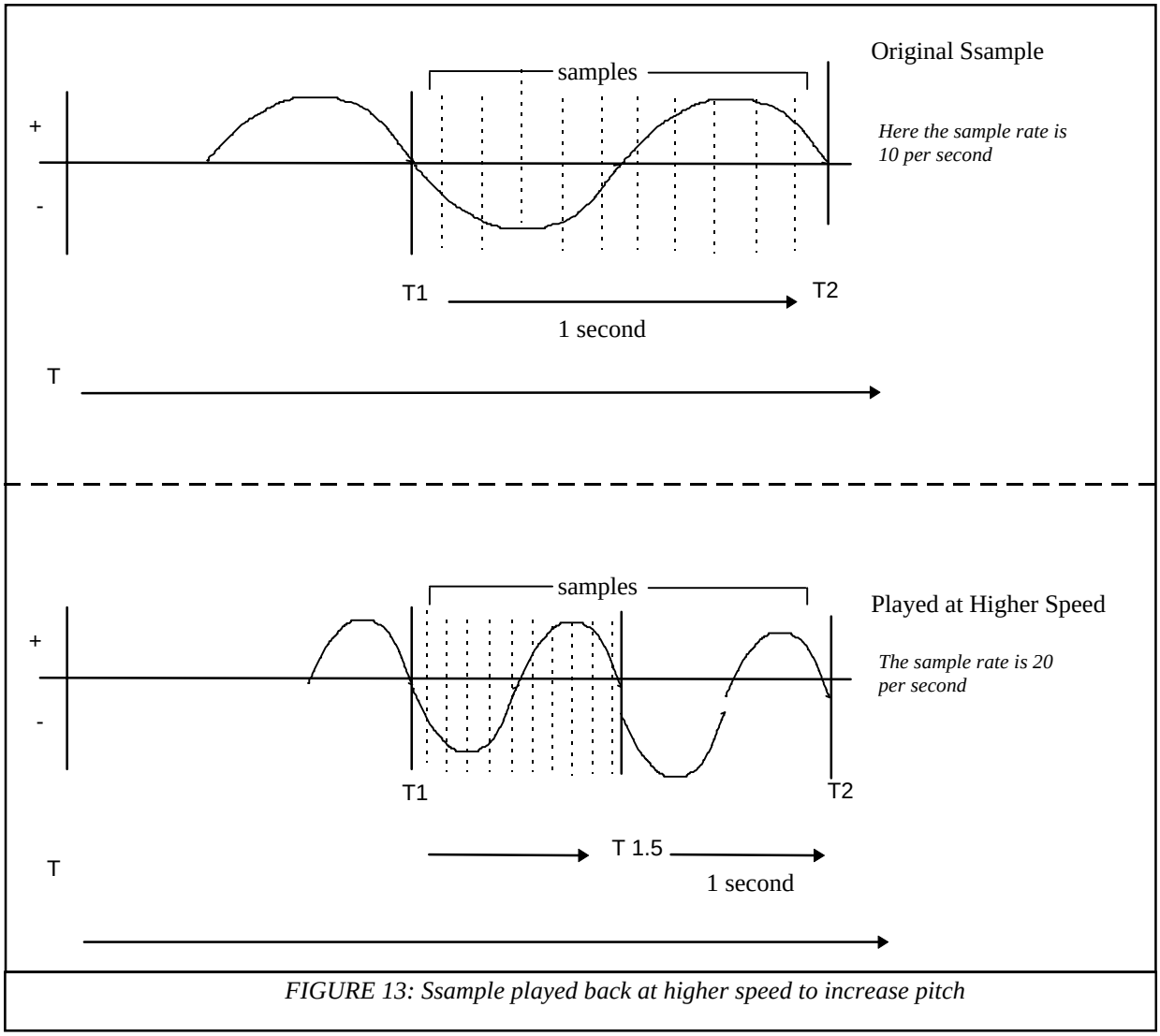


FIGURE 13: Ssample played back at higher speed to increase pitch

Here, the original samples are all played in half a second, giving a shorter sound wave and therefore higher pitch.

Or to give a simpler example: when you used to play your vinyl LPs of U2 (33 RPM) at single speed (45 RPM), U2's Bono had a high pitched voice like he had been on the helium.

Similar to XA-ADPCM, an SPU-ADPCM sound sample takes up approximately one quarter of the size of DA - about 4 bits per sample.

However, an SPU-ADPCM ssample recorded and replayed at 44.1Mhz will be of a poorer quality than a normal DA track (recorded and played at the same speed). This is because the compression loses some data - this is said to be 'lossy'.

So a sample recorded at 22 Mhz can be looped at different speeds:

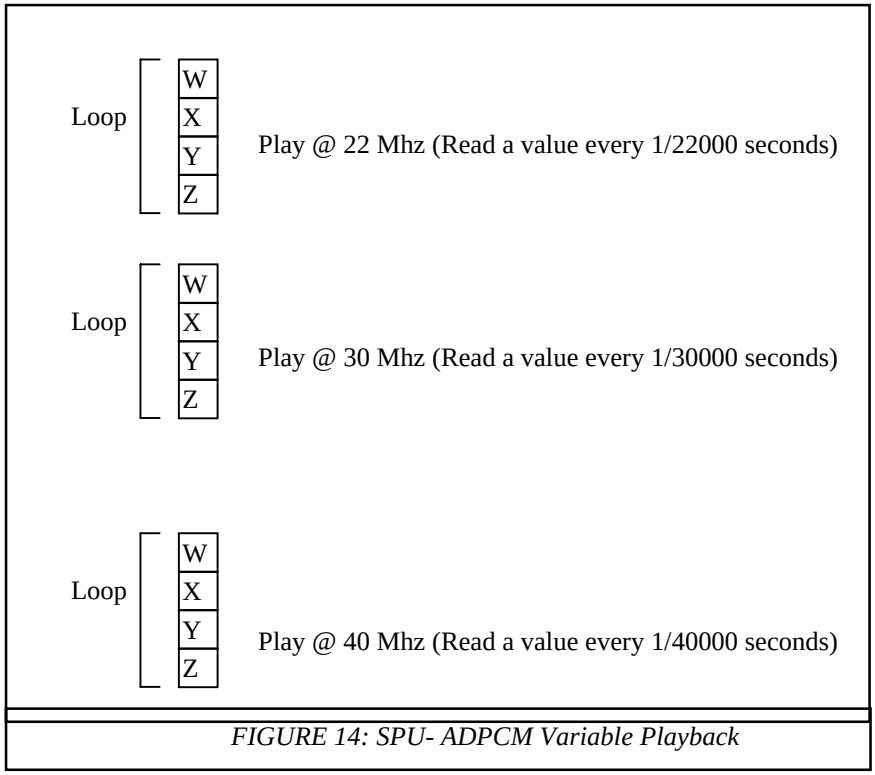


FIGURE 14: SPU- ADPCM Variable Playback

In figure 14: Loop 3 will be over sooner than 2 which will be over sooner than 1……

Loop 1 takes 4 x 1/22000 of a second
Loop 2 takes 4 x 1/30000 of a second
Loop 3 takes 4 x 1/40000 of a second

3.3.3 Practical Sound Tips

3.3.3a File Formats

The VAG file format is a proprietary file format, based upon the AD-PCM compression technique. This provides a compression ratio of around 4:1. Samples are converted into the VAG file format using the program AIFF2VAG, on either the Mac (it is provided as a part of Sony's Sound Artist Tool) or the PC (where it takes the form of a command-line DOS converter, and is provided free). AIFF is a standard file format for sound, and can be produced by most sample editing software. You can use samples of any sample rate up to 48KhZ, so long as they are mono.

As well as the sample data itself, it is necessary to combine VAGS into one file, called a VAB, or Voice Bank file. This is produced using the program SoundDelicatessen, which is part of the Mac Sound Artist Tool. VABs control the ADSR (attack, decay, sustain and release), or envelope of a sound, as well as specifying information about their tuning and other factors.

3.3.3b Sample Editing

Samples should be edited before conversion into VAG format. For this a package such as Sound Designer II, Sound Forge or Cool Edit should be used. It is important that the sample starts and ends cleanly - that is, the sample's volume at the start and end should be zero. This is because the TV speaker will move in and out depending upon the pattern of the wave-form being played back. If, when a sample is played, its amplitude is not zero at the start, the speaker is rapidly moved into a new position, which results in an unpleasant sounding 'click' sound, and can damage the speaker. This can be avoided by fading the sample in and out before conversion (only over a few milliseconds is fine).

3.3.3c Looping

For certain sounds, particularly certain musical sounds such as drum beats or strings, it is often desirable to 'loop' the sound. This involves the setting of loop start and end points in the sample. When the sample is playing back, as it encounters the loop end point, playback will move back to the loop start point. This will continue until the sample is stopped.

Again, to avoid the speaker 'clicking' the amplitude at the loop start and end must be identical. This can usually be achieved in a sample editing program with a 'crossfade' function.

Due to the ADPCM compression, loop points must be placed on 28 sample boundaries. That is, they can exist only on 1 in every 28 sample positions. Thus, a loop point can be placed at sample 0, sample 28, 56 and so on. When looping a sample therefore, it is advisable to loop it manually (so it sounds about right), then adjust the loop points until they are exactly divisible by 28. The loop can then be cross-faded (over a few samples) to make sure it sounds OK.

3.3.3d MIDI sequencing

As well as playing individual sound samples for sound effects or speech, LIBSND allows the playback of MIDI sequences. MIDI sequences are a very efficient way of storing sound, as they consist of 'note-on' and 'note-off' messages which are used to trigger individual sound samples. In this way, a very long piece of music could be played using just a few instrument sound samples, thus using very little sound RAM.

MIDI sequences are produced using a sequencing package, such as Cubase, Vision or Notator. Many people have had trouble getting the Sound Artist Card to work with a sequencer. This seems to be because SoundDelicatessen supports the Apple Midi Manager standard, while many sequencers do not behave reliably under this. Vision seems to give the best performance - details of setting up sequencers to work are given in the Sound Artist Tool documentation.

The best solution is to use two computers in your music setup. One (PC, Mac, Atari etc) should be running a sequencer program, and outputting via MIDI to the Mac containing the Sound Artist Tool. The MIDI Manager and Patchbay (see SA Board documentation) can then patch the MIDI input of the Mac to play back on the sound artist tool.

If this is not possible, it is advisable to produce samples and music on an external sampler, such as a Digidesign SampleCell II (the system used in house at SCEE) or an Akai S1100. Once the music has been produced, it can then be converted into PlayStation format for testing on a development system.

MIDI files should be saved in Standard MIDI format. They should not contain any aftertouch, and pitch bend should be kept minimal (continuous controllers produce large amounts of data, which can cause playback to slow down on PlayStation).

The SMF2SEQ program on the Mac is used to convert the MIDI files into PlayStation's SMF format. Full details about this format and how to use it are in the Sound Artist and Library Overview documentation.

The SAMPLE directories which come with the PlayStation libraries give a good overview of how to use the important sound functions on PlayStation.

3.3.3e Reverb

PlayStation can apply reverb and echo to sounds. Reverb is an algorithm which simulates the reverberant effect of making a sound in a room or hall. It is useful for making sounds 'bigger' and more impressive. Reverb can also be applied to CD audio and XA-ADPCM audio. The most well-known example of this is moving through the tunnel in Ridge Racer - in the tunnel, reverb effects are applied to all sounds in the game.

Reverb uses a buffer in sound RAM. Depending upon the size of the reverb (ie the size of the room you are simulating), more or less memory is required. Large reverbs tend to use up too much sound RAM to be very useful. As SoundDelicatessen allows you to test the different reverb algorithms on your VAB files, you can find the one most suited to your situation

4 IMPLEMENTATION ISSUES

4.1 Streaming

The CD-ROM reads at a maximum of 300K per second. This means that for a stream to be played back at 15 frames per second, there is 20K available per frame.

A 640 by 480 24bit image is (640*480*24) around 900k. For the PlayStation movie player to work with images of this size, it would require a compression ratio of around 45:1. Unsurprisingly this is not really feasible.

4.1.1a MDEC Compression

The PlayStation has a decompression system built into its hardware. This is the MDEC (Motion DECompressor) which uses a 'lossy' (i.e there is a lot of data loss) compression technique to compress individual frames of video.

Two techniques are used: DCT (Discrete Cosine Transformation), and VLC (Variable Length Coding). The actual processes involved are discussed in the *Libpress* chapter of the *Library Overview* manual.

In brief, the DCT function makes an image more suitable for run level encoding. The VLC is a form of run-length compression. This means that images with a narrow range of colours will compress far better than images that are dithered or contain noise.

There are various image processing programs available that allow the user to use a low pass filter to reduce the amount of noise in the image.

4.1.1b Streams Tips

Tips on improving your streaming techniques, gleaned from the European Developers.

- If it's a linear 'Play_stream(<my_filename>)' type, try running with the stack in the DCache, like this.

  ⇒ `unsigned long old_stack;`

  ⇒ `old_stack = SetSp(0x1f8003fc);`

  ⇒ `Play_stream(<mr_streamy>);`

  ⇒ `SetSp(old_stack);`

- When you swap frame buffers, you've probably got a VSync(0) in there just before. Try commenting it out, if you're dropping frames.
- Make sure, under library 3.3, that you're using the new definition of CdlModeStream (now 0x120 instead of 0x100), when using CdRead2.
- Build your stream with MDEC Version 2 in MovConv (not V3), or run some sort of image processing over the sequence to eliminate all those nasty high frequencies.
- Add several dummy frames at the end to absorb the time gap.

4.2 Title that use Multiple CDs

This section explains the procedures related to CD swapping during a game. (August 9, 1996, SCEI R & D Division

Regarding the titles consisting of multiple CDs and CD swapping is required during game play, make sure to follow the steps below:

1) Prior to CD Swapping

(MUST) Prior to CD swapping(before displaying the prompt message for CD swapping), set the CD subsystem to "Normal Speed Mode".

(OPTIONAL) After setting the mode, stop the CD rotation using "**CdlStop**".

Example of "Normal Speed Mode" setting

```
        :
com = 0;
CdControlB( CdlSetmode, &com, result );
        :
```

2) How to Detect Completion of CD Swapping

Follow the two steps (A) Detect Open/Close status and (B) Detect spindle rotation, in this order to detect the completion of CD swapping.

The command "CdlNop" is used in both detection tests.

```
CdControlB( CdlNop, 0, result );      /* char result[ 8 ]; */
```

(A) The Open/Close status of the cover is reflected to the Bit 4 (0x10) of result[0]. This flag is used to detect that the cover was opened and then closed and has the following states;

Cover being opened:	Always 1
Cover was closed:	1 for the first detection, 0 after the first detection

Therefore, when the flag changes from 1 to 0, it can be assumed that the CD was changed.

(B) Wait till Bit 1 (0x02) of result[0] becomes 1 after issuing the "CdlNop" command.

3) Immediately After CD Swapping

After CD swapping followed by the cover close, the CD subsystem starts reading the TOC data. During this read, do not issue any command except for "CdlNop" and "CdlGetTN".

To detect the completion of TOC data read, use "**CdlGetTN**" command described below. After successful execution of "**CdlGetTN**", the CD subsystem finishes TOC data read and is able to issue standard command. Thus repeat issuing the "**CdlGetTN**" command till detect the completion of TOC data read.

Name	Number	Execution Result	Description	type
CdlGetTN	0x13	status	Obtained number of TOC	non-block

First Track No.
Last Track No.

4) PlayStation Disk Check

Check whether the CD placed is a PlayStation disk (black disk) by issuing the logical access command (**CdlReadS**/N, **CdlSeekL**, etc.)

Logical access to the disk that was not recognised as a PlayStation disk will result in a "Command Error".

This "Command Error" differs form the ordinal DiskError in the following manner.

Right after DiskError, the bits below are set to 1;

```
Bit 0 ( 0x01 ) of result[ 0 ]
Bit 6 ( 0x40 ) of result[ 1 ]
```

After detecting a "Command Error", any logical access will not be accepted. The reason for this error may be due to setting an incorrect type of CD (e.g. CD-DA etc.), or placing a CD improperly.

The only way to recover from this "Command Error" is to open the cover and place the CD again. Thus display appropriate message and then repeat steps starting from Step 1).

When a logical access such as data load exists immediately after the data load, as game was designed, checking if the disk is a PlayStation disk or not can be done by issuing the logical access command.

For other cases (e.g. playing DA), check the disk by issuing a dummy read.

Note 1) Since Step 1) through Step 3) can be successfully executed even for the CD-DA or other ordinal CD-ROM, make sure to check whether the disk is a PlayStation disk or not.

Note 2) Although ordinal CD-ROM is also recognised as a PlayStation disk on the Debugging Station, only the black disk is recognised as a PlayStation disk on the PlayStation.

5) Others

- Be sure to use the "Normal Speed Mode" thorough-out Steps 1)- 3).
- Be sure to detect the command completion using "**CdControlB**" during Step 1) through 3). Although examples above are simplified for easy reading, be sure to include the execution result checking for each command.

■ Please display appropriate messages while checking CDs.

4.2.1a Important Information: Sector Drop Out Problem during CD Read

We would like to inform you of special programming notes in regards to CD access.

During CD data read by issuing CdRead()•CdRead2() of CdlReadN or CdlReadS command, issuing other commands below may result in dropping out one sector of data:

CdlNop
CdlGetlocL
CdlGetlocP
CdlSetloc

When the drop out occurs, the data obtained by CdReady() orDataReady callback is that of the NEXT sector, not the one expected.

The Reason for this problem is that issuing other commands during CD read overloads the CD subsystem, and thus finish reading the next sector before completion of the previous command process. This problem occurs more frequently in case of issuing several other commands.

To avoid the drop out problem, please make sure not to issue other command during data read.

CdRead() or Streaming Library of the Library Version 3.5 or later monitors the sector continuity from the sub-header location information. In case of drop out, CdReadSync is designed to return an error code -1 (read failed).

4.3 Overlays

PlayStation has only 2 Megabytes of main memory, although many games have been written that use far smaller amounts of memory and still provide a quality experience, for a lot of applications, the 2Mb will simply not be enough.

Large amounts of Data can be loaded from the CD when required (actually it is better if it is loaded before it is needed).

Code, however, is a different matter. It is feasible to have several executables present in main memory that share data and pass control between them. (This is how programs like the *Demo Disc* work). However this is probably not a convenient way for most game engines. Overlays allow the program to have more functions than will fit into main RAM and page them in when required.

In the simplest case, the main core program would always be memory resident, with a number of other modules (for example one for CD loading, Memory card access) introduced as overlays since these functions are not interdependent, so they are not required to be in main RAM simultaneously.

The process of creating overlays then, is to trick the compilation process into creating two modules that occupy the same area in RAM, and then doing some processing to ensure that the correct module is loaded when the functions are called. Because the modules are quite small, they do not take long to load, so time taken to swap overlays is minimised.

```
Start    Stop     Length   Obj Group              Section name
80040000 800409B7 000009B8 80040000 text          .rdata
800409B8 800498FB 00008F44 800409B8 text          .text
800498FC 8004A99F 000010A4 800498FC text          .data
8004A9A0 8004A9AB 0000000C 8004A9A0 text          .sdata
8004A9AC 8004A9AF 00000004 8004A9AC bss           .sbss
8004A9B0 8005372B 00008D7C 8004A9B0 bss           .bss
8005372C 80053762 00000037 8005372C l1            l1.rdata
80053763 8005382F 000000CD 80053763 l1            l1.text
8005372C 8005376D 00000042 8005372C l2            l2.rdata
8005376E 8005399B 0000022E 8005376E l2            l2.text
```

Map File Extract From A Program Using Overlays

In this example the overlay sections (l1 and l2) are placed in main memory after the *bss* sections.

Note that both sections start at the same point in memory as far as the calling program is concerned.

It is important to realise at this point that this now makes the heap_base variable used by Malloc() incorrect since this points to then end of the *bss* section.

Since there is no guarantee that the two modules (l1 and l2) are of the same size the heap_base must be assigned a value that is clear of the biggest section, to prevent the heap and the code being overwritten with disastrous consequences.

```
main.cpe: main.obj overlay1.obj overlay2.obj address.obj
        psylink /v /c /m @main.lnk,main.cpe,main.sym,main.map
main.obj: main.c
        ccpsx -c -g -comments-c++ main.c
overlay1.obj: overlay1.c
```

```
        ccpsx -c -g -G0 -comments-c++ -Wa,sl1 overlay1.c
overlay2.obj: overlay2.c
        ccpsx -c -g -G0 -comments-c++ -Wa,sl2 overlay2.c
address.obj: address.s
        asmpsx /l address,address
```

Note: A complete example using overlays is available on the BBS. This includes more notes on how to use overlays.

4.3.1a Hints on programming Overlays

If any of the code in any of the modules is changed, the positions of the sections may also change. This means that if the program is being run from the Emulator the image must be rebuilt each time, because their position in RAM may have altered

Overlays are thus one of the few areas of code where it may be sensible to use the PC filing system during development. Just bear in mind that the PC filing system operates in a different way and with a faster data transfer rate to the CD

- It is important to ensure that the Instruction cache is flushed when a code overlay is loaded into ram, else if the previous overlay is still in the cache this will be executed instead, with disastrous consequences.
- Ensuring the overlays are of equal size minimises wasted memory: if one overlay is much smaller than the other, the extra space reserved for it is wasted, so either redesign the structure of the game to equalise the overlays or fill the wasted space with something nice.
- Consider the advantages of having lots of small overlay sections rather than a few big ones. Small overlays will load faster.
- The newest version of the C compiler and assembler (listed below) should be employed. Earlier versions had problems with handling overlays.

```
aspsx 1.12+
ccpsx 1.06+
psylink 2.34+
dbugpsx 4.81+
```

4.4 General Implementation Tips

1. Reducing On-Screen Flicker when using *ResetGraph(0)*

   After using ResetGraph(0) to rest the graphic environment, code should be structured to any non-screen relevant functions (such as PadInit or CdReads) before performing initialisation of your display environment with setRECT/PutDispEnv followed by a SetDispMask(1). This will allow the GPU to sufficiently 'warm-up' and will reduce unsightly flicker.

2. Required use of ResetCallback() Function

   You should ensure that you programs perform correct initialisation of the system by using ResetCallback(). This function initialises all of the System Callbacks, and without its use symptoms such as system hang - especially on newer versions of the PlayStation hardware, and libraries after 3.1.

   Please see Section 5 (*Controller Library* - libetc) of the Library Reference manual.

3. Use VSync to lock the frame rate:

   VSync(0), when the frame rate is 50 (60 for NTSC) frame/sec.

   VSync(2), when the frame rate is 25 (30 for NTSC)frame/sec.

   VSync(3), when the frame rate is 20 .. for NTSC frame/sec.

   VSync(4), when the frame rate is 15 .. for NTSC frame/sec.

5 PLAYSTATION PERIPHERALS

We provide a consistent way of writing code to handle the various controllers designed for the PlayStation - see CTRLLER.ZIP, for an include file that will simplify coding**.**

There are three types of peripherals to consider:

Controllers

Link Cable

Memory Card

Further Reading

Additional Information for using Controllers is available for the following sources.

Library Overview Libapi psxman

Library Reference Libapi psxman

cntrller.zip psxlibs

5.2 Controllers

The PlayStation has several different control options available at the current time. This number will expand over time. Currently there are five types of Controller that should be considered in title development (compulsory compatability with the standard Controller only), others on the market need not be explicitly developed for as they conform to the standards for the Controllers listed below.

CTRLLER.ZIP explicitly supports :

- Standard Controller (called a 'Controller')
  This is the only Controller/Peripheral that your title ***MUST*** work with.
- negCon
- Mouse
- Multi tap
- Analog Joystick

  NOTE: Two new Controllers still ***under consideration*** are a gun and a steering wheel. Please contact your Account Manager for details of these.

  The operation of most of the Controllers listed above are described in the *Library Overview* manual (Lib API). neGcon and Analog Joystick are described below.

Nomenclature of Peripherals

Right	Wrong
Memory card	RAM card
Controller	Pad
Link cable	Combat cable
Multi tap	Multi-tap, Multitap
Analog Joystick	Flight Stick
Mouse	
Directional button	D-PAD, Cross button
Controller port	Controller slot
Memory card slot	Memory card port
neGcon	Negcon

(ii) Controller Conditions

The following conditions should be met by the game for it to pass the tests placed upon it by the European Quality Assurance Group (the latest QA standards should be obtained from the BBS as QADOC.ZIP).

- The game must be playable with the standard Controller. Controller Port 1 always takes precedence over Port 2. So in the case of a single player game, the game only responds to Controllers connected to Port 1.
- If a Controller is plugged in that is not supported by the game it should be ignored. For example, if a neGcon is plugged into a game which requires a standard pad, the neGcon input should be ignored. (Under the current rules, for example Tekken would have failed QA on this point.)
- The game should recognise specific Controllers from the Controller option screen and include calibration options, if they are required. For example the, analogue controls should be adjustable for sensitivity. It should be possible to reconfigure a game to any attached peripheral
- If a controller is removed during play the game should go into a pause state, and maintain that state while the Controller is reconnected or a different Controller connected. If the newly connected Controller needs configuring, this should be an option.

(iii)Multiple Controllers and Different Controller Types

Most of the sample code supplied only supports a single Controller and does not detect the type of Controller connected - in a title this would fail QA tests.

Standard code uses the PadInit and PadRead functions. However Libapi supports a more sophisticated interface to the Controllers, and SCEE provides an additional header file cntrller.h which allows access to these facilities easily and tidily.

5.2.2 Multi Tap

The *Multi tap* allows the connection of up to four individual Controllers and/or memory cards to each PlayStation controller port.

Titles which use a *Multi tap* should ensure that libtap.lib is used.

There are certain requirements for using Multi tap in a title. These are listed below.

- Make sure you use the correct and authorised naming conventions (especially the ports):

- When a Multi tap is connected to console controller port 1, the names of each port on Multi tap are 1A, 1B, 1C and 1D.
- When a Multi tap is connected to console controller port 2, the names of each port on the Multi tap are 2A, 2B, 2C and 2D.

5.2.3 Analog Joystick (SCPH-1110)

Availability - quarter 3, 1996 for commercial version.

Physically, the Analog Joystick consist of two standard joysticks, linked by a control panel with standard switches. The joystick can act in either digital or analogue mode.

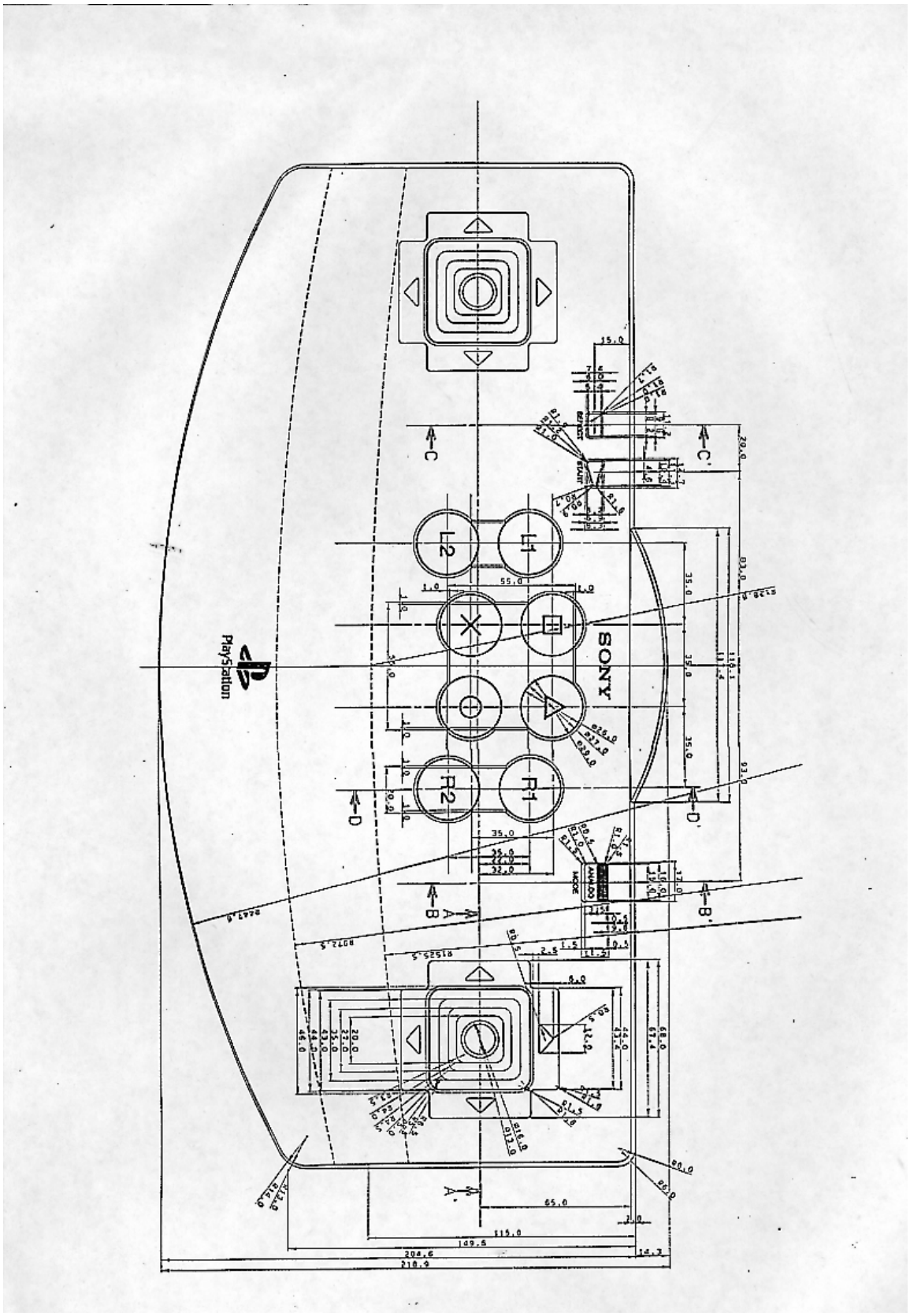
Make sure you include a Calibration menu (0 position calibration, idle movement, sensitivity etc.) in your title.

(See diagram of the Joystick on the next page.)

The 8 byte Controller packet (giving the status of the Controller) which can be returned at every vsynch call back consists of the following for the Analog Joystick.

Byte No	Type	Bit No./Value	Button Assignment
1	Buffer Status	8 bits	0x00: success; 0xff: failure
2	Data length Controller type	bits 0 - 3 bits 4 - 7 (8 bits total)	data length/2 (= 0x03) Controller type (0x0**5**) (Analog Joystick is 5)
3, 4	Status of 14 digital buttons	16 bits	0x00: push ; 0x01 : release (as per standard Controller - however L1/L2/R1/R2 are not present on stick)
5 - 8	Analogue value (4 values)	4 x 8 bits	range 0x00 - 0xff. (x, y) , (x, y)

Table 8.1c - 1: The Controller Packet for an Analog JoyStick



SCPH-100 Analog Joystick

5.2.4 Namco negCon

NegCon Controller Button Assignment

The input data is transmitted as 6-byte serial data and the contents are as follows.

Make sure you include a Calibration menu (0 position calibration, idle movement, sensitivity etc.) in your title.

The 8 byte Controller packet (giving the status of the Controller) which can be returned at every vsynch call back consists of the following for the neGcon.

Note: The value varies by twisting NegCon not by pressing a button.

Byte No	**Type**	**Bit No./Value**	**Button Assignment**
1	Digital No. 1	bit 7	Left (on cross button)
		bit 6	Down (on cross button)
		bit 5	Right (on cross button)
		bit 4	Up (on cross button)
		bit 3	S (Start)
		bit 2	none
		bit 1	none
		bit 0	none
		(8 bits total)	
2	Digital No. 2	bit 7	none
		bit 6	none
		bit 5	A
		bit 4	B
		bit 3	R (side button)
		bit 2	none
		bit 1	none
		bit 0	none
		(8 bits total)	
3	Analog	0-255	Centre (1)
4	Analog	0-255	I
5	Analog	0-255	II
6	Analog	0-255	L (side button)
The Controller Packet for an Analog JoyStick			

NegCon Controller Data Format

1 - Switch Data

0 is returned when the button is pressed otherwise 1 is returned. Unused bits return 1. The return values comply with the standard Controller specifications, however, they can be changed by BIOS if necessary.

2 - Switch Data

As above.

3 - Twist Data

The title needs to be designed so that neGcon's twist data value is around 128 when it is released. However, a variation of +/-8 should be allowed for to take account of variations that may occur between different neGcons and the different effect gravity may have on Controller buttons if the Controller is held in a different position. (The specified +/-8 allowance may be changed later if it proves unsuitable but this does not affect your current title.)

Remember that you are recommended to include a calibration option in any title which uses a neGcon Controller as individual neGcons may differ slightly.

4 - I button Data

The more this button is pressed, the bigger value is returned. The title should be designed so that a value of 16 or less is returned when it is released.. The software should consider 16 or less as the button not pressed. This error margin allows for variations between different neGcons, as described in *3 - Twist Data*, above - the specified value of 16 or less may be changed later if it proves unsuitable but this does not affect your current title.) The maximum value returned when pressed should be not less than 192(C0H).

5 - II button Data

Same as I button, above.

6 - L button Data

Same as I button, above.

5.3 Link Cable

Originally known as the ‘Combat Cable’, the Link Cable allows two PlayStations to be connected together, and is supported by the **libcomb** library.

A special version of the cable is provided to link development systems together **(DTL-H2060).** Be warned - you are advised to connect this cable when the development PCs are powered down. There have been cases reported where connecting (or reconnecting) to “live” PCs has resulted in damage to the development board itself.

Link cable tips:

- Make sure you do an initial **read** to enable interrupts
- Read and write in 8 byte chunks (this is the size of the hardware buffer)
- Intersperse **write**() with **read**()

5.4 Memory Cards

Memory cards are the removable storage medium used by the PlayStation to save and recover game sessions (game save file). A Memory card contains 120K of static non volatile RAM. This is divided conceptually into 15 slots, each of 8k.

Memory cards have the advantage of small size and ruggedness (there are no moving parts and no batteries) but they are slow to access.

General Features of Memory Cards

- Capacity 120 Kilobytes
- Access Speed 10 Kilobytes per second
- Life span. 100,000 reads guaranteed

As well a being accessed by the application that created the game save file, each file will also need to be compatible with the OSD memory card management program built into the ROM of each PlayStation.

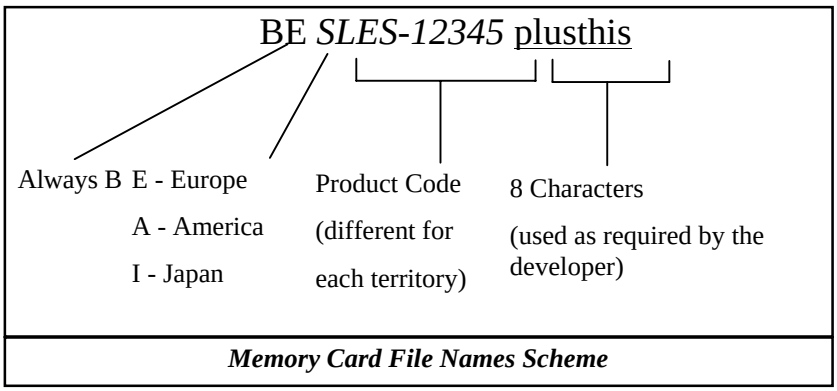
Different versions of the OSD are available for each PlayStation territory (i.e. European, Japanese, American).

The OSD program displays each save game file as a number of icons plus a textual title. Each icon displayed relates to a slot used. This information is provided by a header that must be included at the start of the save game file.

5.4.2 Memory Card Filenames

Filenames used for memory card save games are based upon the product code for your game. The file name is in the following format:

Always B



Memory Card File Names Scheme

Some Rules on Title Names

- Choose either Shift-JIS or ASCII code. Do not mix them.
- Use a maximum 32 ASCII letters (32 Byte). If less than 32 characters, please put null letter (0x00) at the end or fulfil the vacant part with blank letter.

5.4.2b Save Game File Header Format

The Save Game File Header format is as follows.

Name	Size (Bytes)
Magic Number	2 (always 'SC')
Type(see table 1)	1 (always 0x11 for Europe)
No. of Slots	1 (1..15, each being 8k)
Text Name	64 (32 Shift JIS (Must be Kanji, not ASCII)
Pad	28 (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)

In the header file there is provision for three icon images, as below, which allows animation.

Type	Number of Icon Images (automatically replaced animation)
0x11	1
0x12	2
0x13	3

Use Kanji fonts

The Memory card save game file header *Text Name* **must** be stored in Kanji (Japanese Shift-JIS) type character coding and not the normal ASCII.

(If ASCII were used the game header file, the title would not be displayed correctly if it is longer than 16 characters - 32 is the total possible.)

There is an example program (*writecrd*.c) for this on the BBS called CARD2.ZIP in the ***PlayStation*** code area.

Animated Icons

As can be seen from the table above it is possible to have animated icons in the save game. There are three frames available and they are cycled automatically by the OSD.

(If two slots are used per save, a two frame animation is allowed; with three slots, a three frame animation).

The Icon is a 16 * 16 image with 4 bit colour depth.

Note: The European OSD has a bug in it which gives on-screen corruption if animated icons are used. Therefore do not use *animated* icons for European titles (i.e. only use type 0x11) if your save occupies a single slot.

5.4.3 Hints on Using Memory Cards

Memory cards are slow to recognise and access. Memory card accesses, therefore, should be kept to a minimum and when they are made be as efficient as possible.

As a Memory card is divided into 8k slots, the minimum possible number should be used. Writes to the card system are always done as multiples of 8K, so if the intended game save is 8K + 1 bytes, it will use two slots, taking twice as long to save as one slot and take twice as much space as a save that is only 1 byte smaller.

In this example, there is only 1 byte over so some compression/economising of the data could probably be achieved. For over spill which cannot be squeezed into 8k, the alternative is to save more data in the in the file and use up the wasted space. It costs nothing in terms of resources or performance.

5.4.3a Notes for those writing RPG's and Strategy games

Many games in these genres benefit from allowing the user to save several sessions concurrently. This allows the user to backup before undertaking hazardous stages, or to approach a complex problem from several angles simultaneously.

This can be achieved in two ways, firstly by having several saves within the same file. For example if the data required by the save game is only 2K, you can save 4 games within the same slot.

Alternatively, the format of the save game filename allows for multiple saves from the same game, because there are nine characters in the memory card file name left available for user allocation. This could be used to store a user entered string that is a description of the game save at that point in the game.

Better still, add an internal character field that allows for a textual description of the current stage of the game and display this when displaying a list of all the save games on the memory card.

`BESLES-12345level2`

In some games the memory card facilities play a very important role. A classic example of this is XCOM: Enemy Unknown, which would be very hard, if not impossible to finish without frequent memory card access.

These unfortunately are slow and so disrupt the flow of play. One solution would be to save games into RAM as a quick save, with the option to commit the game to the card at any time. This is only feasible if the game never crashes out and there is room in memory for the save game(s).

As Memory cards are quite expensive, games that use a large amount of space on the memory card and will take a long time to complete (e.g. a RPG) may prove unpopular to the user group (author's opinion).

It is not possible to lengthen a file saved on a Memory card once it is created, therefore when the file is created it should as large as it will ever need to be.

5.4.4 Testing Memory Cards

1. Is the Memory card operation display working? (Display of icon, or file name, etc.)

2. Can Memory card be formatted? (On-screen instructions and statements are correct?)

3. The “formatting menu” options should include a dialogue such as "The Memory card *ID* is not formatted yet. Do you want to format that? Yes? or No?". Then, when "Yes" is chosen, saving should start and when "No" is chosen, the menu should exit from save mode.

4. Check what happens when there aren’t enough empty blocks in the Memory card for a game save.

5. Check what happens when the Memory card is not inserted into memory card slot.

More information about memory cards is available on the BBS.

5.5 CD-ROM GENERATION

5.5.1 Master Disc Creation

Final master disks must be created on an approved Sony CD-ROM burner, with the Sony CD Generator software.

There are no exceptions to this - any discs created by any other CD-ROM burner are not suitable for the DADC pressing factory - Only these burners and software can write the necessary territory and copy protection information.

Similarly, the master discs must be produced using CDR-71PS mastering discs.

You'll be able to store approximately 71 minutes of data (roughly 624 MB) of a combination of data (in track 1) and audio tracks (other tracks).

There are many variables as to the exact quantity of information you'll be able to squeeze on a disc - for example the total will be reduced by the number of audio (DA) tracks you have, and how you wish to access the outer tracks (you can't seek around the last few minutes). These issues are discussed below.

License files

You will require a license file for each territory (Japan, Europe and America) in which your game is to appear. You can get a license file for Europe from us (specifically via your Account Manager). License files for other territories must be obtained from SCEA and SCEJ.

CPE to EXE converter

The CPE2X.EXE program is supplied with the license file on floppy disc *on request from your Account Manager*. This program can create a different executable for each territory. Please ensure that the appropriate command line switch is used to select the correct territory.

5.5.1b Mastering Procedures for PlayStation products

Full details may be found in QADOC.ZIP..

Procedure

Place in the root directory of CD-ROM, the file "SYSTEM.CNF;1which specifies the boot file.

Naming rule for the boot file:

Assuming that the product code of the disc is XXXX-AAAAA, make the name of the boot file XXXX_AAA.AA;1 (note that the hyphen '-' is replaced with an underscore '_' for ISO-9660 compatibility) by inserting the period (.) between 8th and 9th character.

Example:	**Disc Type Number**	**File Name**
	SLPS-12345	SLPS_123.45;1
	SLUS-12345	SLUS_123.45;1
	SLES-12345	SLES_123.45;1

Contents of SYSTEM.CNF;1

(Assuming the name of boot file is XXXX-AAAAA. It is necessary to specify the full path with the name of the device)

```
BOOT = cdrom:\XXXX_AAA.AA;1
TCB = 4
EVENT = 10
STACK = 801fff00
```

5.5.2 Using the CD-Generator Software

In general, add files (including audio tracks) via the directory menu, and organise them in the layout menu. (Note that you can drag files and directories from the standard Windows file manager menu.)

(Although Buildcd can generate CCS files, the process is problematic and is not recommended).

5.5.2a Writing DA Tracks to a CD-ROM

The process of writing DA tracks involves two main tasks.

1. Preparation of DA source files.

2. Creating DA tracks using the CD-ROM Generator.

DA Source Files

Before you can write a DA track onto a CD you need to prepare a DA source containing the DA data to be written onto the CD. The structure of DA source files are described in section 3.1.3 of the CD-ROM Generator manual. It is also possible to use 16-bit PCM/44.1kHz WAV files as DA source files.

In the case of Macintosh RAW2DA tool conversions, input data format must be Sound Designer II. If you use AIFF format, some noises may appear before and after data.

Also, when transmitting data from Macintosh to PC, remember not to use MacBinary format ! (Your resource parts will be turned into noise!).

5.5.2b Creating CD-DA Tracks

Once the DA source files have been prepared they can be placed onto the CD in one of three ways: the directory method, Drag and Drop (both described below) or by using “Put Files”, also discussed below.

Directory

Using the Directory mode, which can be selected via the “Directory” button, you can add DA to a CD by simply copying the DA source file to a directory on the CD. This is achieved in the same way as copying any other file.

For example (using the “Put Files and Directories” function):

1. Once the DA source file has been copied into the desired directory you need to set its file type to CD-DA using the “File Type” dialog box (accessed from the File Type button).

   In the “File Type” dialog box select the “CD-DA File(s)” radio button, and click “Ok”. If this has worked correctly the DA file type icon in the directory structure window will have changed to a musical note icon.

2. You can now view your new DA track by using the Layout mode (selected via the “Layout” button).
   If you have inserted the new DA track using the method described above its location defaults to track 2. By selecting this track in the “Track Window” the name of the DA source file associated with it will be displayed in the “Location Window”.

3. If you wish to change the order of the DA tracks you can either alter the order of the tracks using the “Move” operation, or move the actual DA source files into other tracks. Both of these methods are described in the *CD-ROM Generator* manual.

Drag and Drop

1. From “Windows File Manger” and select the DA source file to be copied.

2. Drag the selected DA source file into the desired track by using the CD-ROM Generator's "Track Window".
   If this has worked correctly the name of the newly copied source file will appear in the "Location Window". Please remember that when a DA file is copied in this way it will not appear in the directory structure of the CD as a standard file would. This is completely normal, as CD-DA tracks are not visible in a CD's directory structure.

Put files

DA source files can also be copied directly into a track via the "Put Files" function, which can be invoked from the Edit Menu. This method is fully described in section 3.6.2 of *the CD-ROM Generator* manual.