SGX Embedded Systems DDK for Linux

Software Functional Specification

Filename : SGX Embedded Systems DDK for Linux.Software Functional Specification (1.7 DDK)

Version : 1.0.9 External Issue

Issue Date : 13 Jun 2011

Author : PowerVR

Contents

List of Figures

Glossary of Terms

Khronos Group	Body that controls the OpenGL-ES, OpenVG and OpenMAX specifications
EGL 1.3	The Native Platform Graphics Interface implemented for this project.
OpenGL-ES™ 2.0	The programmable 3D API implemented for this project OpenGL-ES is a trademark of Silicon Graphics, Inc.
OpenGL-ES™ 1.x	The fixed function 3D API implemented for this project OpenGL-ES is a trademark of Silicon Graphics, Inc.
OpenMAX™ 1.0	The video acceleration API implemented for this project OpenMAX is a trademark of The Khonos Group, Inc.
OpenVG™ 1.0	The vector graphics API implemented for this project OpenVG is a trademark of The Khonos Group, Inc.
SGX	IP that provides 2D/3D/Video acceleration

1. Introduction

1.1. Scope

Since most of the software components within the SGX DDK have separate detailed Functional Specifications this document is largely an overview of the constituent components within the DDK with references to the more detailed documents. The exception is the SGX DDK Build System which is documented in detail.

1.2. Related Documents

OpenVG Specification, Khronos Group, Version 1.0
OpenMAX Specification, Khronos Group, Version 1.0
OpenGL-ES Specification, Khronos Group, Version 1.0, 1.1, 2.0
EGL Specification, Khronos Group, Version 1.4
SGX OpenGL ES 2.0 Reference Driver, Software Functional Specification
SGX OpenGL ES 1.1 Reference Driver, Software Functional Specification
SGX OpenVG Reference Driver, Software Functional Specification
SGX OpenMAX Reference Driver, Software Functional Specification
SGX EGL 1.x Reference Driver, Software Functional Specification
PVR2D Reference Driver, Software Functional Specification
Consumer Services Reference Driver, Software Functional Specification
Pluggable Window System.Implementation Guide
Pluggable Window System.API
Consumer Services.3rd Party Buffer API
Consumer Services.3rd Party Display API
PVR2D API Specification

2. Overview

The SGX Embedded Systems DDK for Linux integrates SGX drivers for OpenGL/ES, OpenVG, OpenMAX and customisable window system acceleration.

2.1. Functionality Summary

2.1.1. 3D Acceleration

The SGX ES-DDK for Linux contains both OpenGL ES 1.1 and an OpenGL ES 2.0 drivers, together with the associated EGL interfaces. These allow acceleration of 3D graphics through industry standard APIs.

2.1.2. 2D Acceleration

The SGX ES-DDK for Linux contains a PVR2D driver, which can be used by overlying window system to accelerate certain bit-blitting operations. The PVR2D API is also provided to Pluggable Window System (OPK) writers to provide 2D acceleration support (as well as surface creation and wrapping and potentially flipping).

2.1.3. Vector Graphics Acceleration

The SGX ES-DDK for Linux contains an OpenVG driver, together with the EGL 1.3 interface This allows acceleration of vector graphics through an industry standard API.

2.1.4. Video Acceleration

The SGX ES-DDK for Linux contains an OpenMAX driver. This allows acceleration of video through an industry standard API.

2.2. Packaging

The DDK software package will be delivered as a gzip compressed tar file. This package will contain the source code and makefile system to build the SGX DDK software. Additionally there will be driver binaries built for specific targets along with installer scripts, init.d scripts and unit tests.

3. Component Structure

The following diagram details a top-level overview of the major components of the SGX ES-DDK for Linux and their relationships.

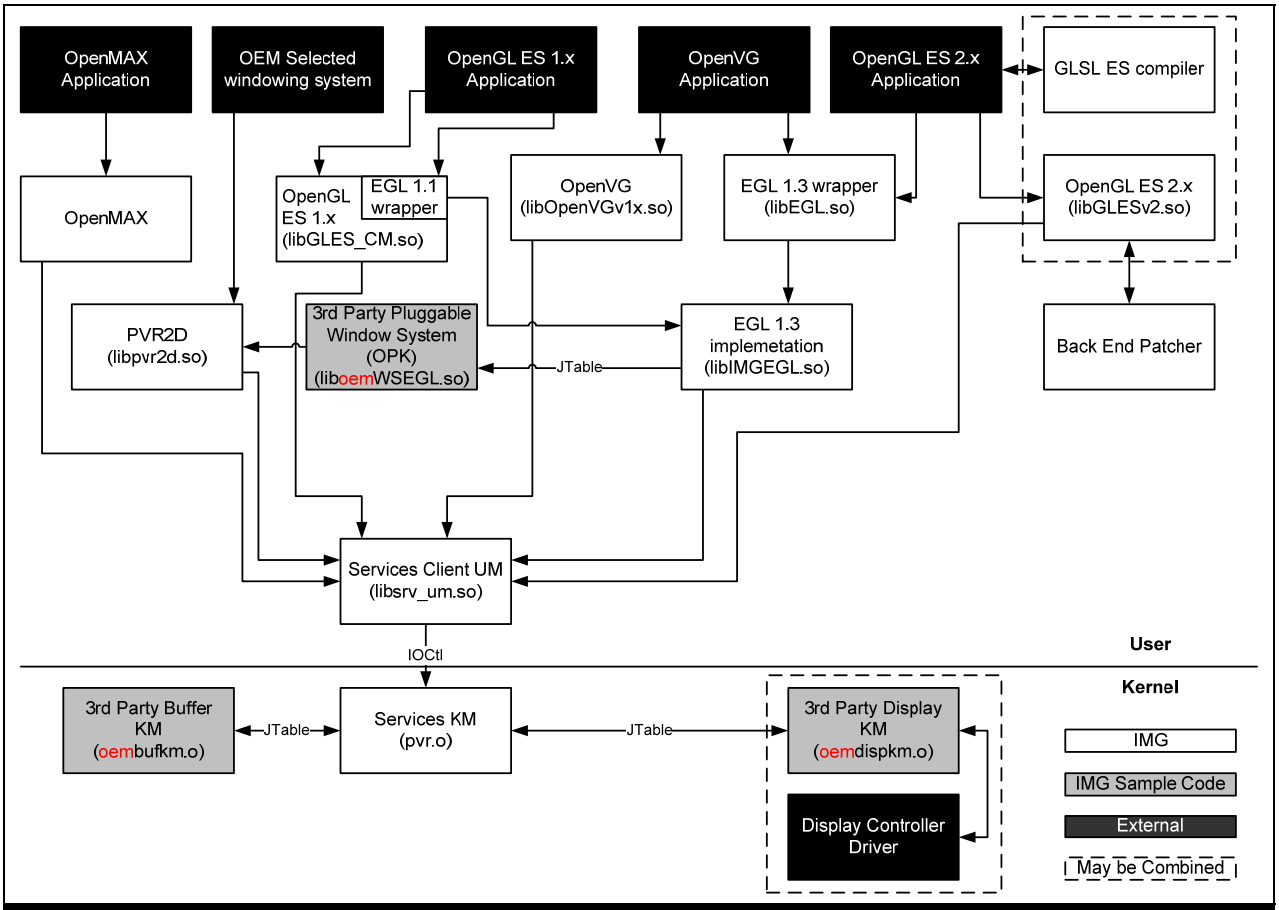

Figure 1 Major Component Relationships

3.1. EGL

The EGL 1.4 component is a platform binding API used to bind the rendering of other Khronos APIs into the target environment window (or display) system. It comprises a generic window system independent part and a window system interface part. The generic part exposes the Khronos EGL entry points and implements the semantics of the API specification. The lower level window system interface component implements a portable window system agnostic interface, suitable for customer to target their window system of choice. The interface between the generic EGL component and lower level component is detailed in the Pluggable Window System (OPK) API documentation. Display query support, surface creation and presentation mechanisms are discussed in the Pluggable Window System (OPK) implementation documentation.

For historical and backwards compatibility reasons the OpenGL ES 1.1 component also offers the EGL interface in its v1.1 form.

Configuration Generation

Generation of supported configurations is the responsibility of the window system interface component. The PowerVR services are queried to determine the current primary display pixel format.

A series of configurations is generated based on the primary display pixel format along with variations for different sample buffer configurations.

OpenGL-ES

The table below summarises the configuration permutations available for OpenGL ES supporting render targets.

Attribute	Values
EGL_RENDERABLE_TYPE	EGL_OPENGL_ES_BIT, EGL_OPENGL_ES2_BIT, EGL_OPENGVG_BIT
EGL_SURFACE_TYPE	EGL_WINDOW_BIT, EGL_PBUFFER_BIT
EGL_ALPHA_SIZE EGL_RED_SIZE EGL_GREEN_SIZE EGL_BLUE_SIZE	The window surfaces will have their color component support defined by the WSEGL module. PBuffer surfaces support 4444, 1555, 0565 and 8888 color depths.
EGL_DEPTH_SIZE	(0,24)
EGL_SAMPLE_BUFFERS	{0, 1}
EGL_STENCIL	(0,8)
EGL_SAMPLES	{0,4}
EGL_CONFIG_CAVEAT	EGL_NONE
EGL_NATIVE_RENDERABLE	EGL_FALSE
EGL_TRANSPARENT_TYPE	EGL_NONE

There are four base configurations including one with no depth buffer

- No Depth buffer

three sample buffer configurations:

- 24bit Depth Buffer, no samples buffers.
- 24bit Depth Buffer, one sample buffer, four samples.

OpenVG

The table below summarises the configuration permutations available for OpenVG supporting render targets.

Attribute	Values
EGL_BUFFER_SIZE EGL_ALPHA_SIZE EGL_RED_SIZE EGL_GREEN_SIZE EGL_BLUE_SIZE	The window surfaces will have their color component support defined by the WSEGL module. PBuffer surfaces support 4444, 1555, 0565 and 8888 color depths.
EGL_ALPHA_MASK_SIZE	0, 1, 4, 8
EGL_BIND_TO_TEXTURE_RGB EGL_BIND_TO_TEXTURE_RGBA	EGL_FALSE
EGL_CONFIG_CAVEAT	EGL_NONE
EGL_CONFIG_ID	-
EGL_DEPTH_SIZE	0
EGL_LEVEL	0
EGL_MAX_PBUFFER_WIDTH	1024
EGL_MAX_PBUFFER_HEIGHT	1024
EGL_MAX_PBUFFER_PIXELS	EGL_MAX_PBUFFER_WIDTH * EGL_MAX_PBUFFER_HEIGHT
EGL_MAX_SWAP_INTERVAL	Depends on platform capabilities. Defaults to 1
EGL_MIN_SWAP_INTERVAL	Depends on platform capabilities. Defaults to 1
EGL_NATIVE_RENDERABLE	EGL_FALSE
EGL_NATIVE_VISUAL_ID	0
EGL_NATIVE_VISUAL_TYPE	EGL_NONE
EGL_RENDERABLE_TYPE	EGL_OPENVG_BIT
EGL_SAMPLE_BUFFERS	0
EGL_SAMPLES	0
EGL_STENCIL_SIZE	0
EGL_SURFACE_TYPE	EGL_WINDOW_BIT and EGL_PBUFFER_BIT
EGL_TRANSPARENT_TYPE	EGL_TRANSPARENT_RGB
EGL_TRANSPARENT_RED_VALUE	0-255 depending on surface type
EGL_TRANSPARENT_GREEN_VALUE	0-255 depending on surface type
EGL_TRANSPARENT_BLUE_VALUE	0-255 depending on surface type

3.2. OpenGL ES 1.1

The OpenGL ES 1.1 component will implement the core functionality defined by the OpenGL ES 1.1 specification. This component is described in the SGX OpenGL ES 1.1 Reference Driver, Software Functional Specification.

3.3. OpenGL ES 2.0

The OpenGL ES 2.0 component will implement the core functionality defined by the OpenGL ES 2.0 specification. This component is described in the SGX OpenGL ES 2.0 Reference Driver, Software Functional Specification.

3.4. OpenVG 1.0

The OpenVG 1.0 component will implement the core functionality defined by the OpenVG 1.0 specification. This component is described in the SGX OpenVG Reference Driver, Software Functional Specification.

3.5. OpenMAX 1.0

The OpenMAX 1.0 component will implement the core functionality defined by the OpenMAX 1.0 specification. This component is described in the SGX OpenMAX Reference Driver, Software Functional Specification.

3.6. PVR2D and Window System Acceleration

PVR2D is a dual function library. One of its functions is to provide access to SGX 2D HW acceleration (where present). This functionality can be used customer to provide acceleration of blits for an overlying window system, and the OPK for presentation blits, and the synchronisation thereof. The second function that PVR2D supplies is creation and mapping into the SGX memory of surfaces used as render targets.

3.7. PowerVR Services

Although the PowerVR services are not an end user component, they are vital to the integration of each of the client drivers as they provide a number of common low level services to all SGX drivers.

PowerVR services provide the following functionality:

- SGX device initialisation.
- Low level SGX device management.
- Arbitration and access to common device resources, including:
  - Slave port(s)
  - Registers
  - Command queues
  - Device addressable memory.
  - Synchronisation facilities for above system components.
- Resource management of shared and high value resources to ensure that such resources are not lost.
- Integration with a 3rd party Display/Stream device, with synchronisation of operations.

The reference PowerVR Services implementation is described in the PowerVR Services Software Functional Specification document.

4. Build System

The Build System is responsible for compiling the source code into object files and linking those into shared libraries and executable binaries for a chosen target platform. The system also generates scripts used to install generated binaries onto a final target platform and init scripts that can be run on the platform to load the PowerVR drivers.

4.1. Build time configurations

4.1.1. Pre-requisites

In the accompanying Platform Guide for Linux, the exact system setup required to build the SGX DDK is detailed. This includes (but is not limited to) having a configured and built target system kernel present on the machine used to compile the SGX DDK.

There are also some environment variables that need to be set-up.

4.1.2. How to build

First it is assumed you have set up your environment variables as described in the Platform Guide, including $EURASIAROOT, $KERNELDIR, $CROSS_COMPILE, $TOOLCHAIN, $BISON and $DISCIMAGE.

You can now start building DDK components in one of two ways:

- **In a target build directory** - these are sub-directories of $(EURASIAROOT)/eurasiacon/build.

  This is the way most developers will build, by simply typing "make", e.g.

  ```
  $ cd build/linux2/nohw_linux
  $ make
  ```

  Options to "make" are described later.

- **A specific module sub-directory** – directories contribute files called Linux.mk which define one more more modules. You can build any single module (and all of its dependencies) with:

  ```
  $ cd build/linux2/nohw_linux
  $ make opengles2
  ```

4.1.3. Standard build command

When you are building from a target build directory, then the standard top level build command is:

```
$ make BUILD=release
```

The standard top level install command is:

```
# make install
```

4.1.4. What You Can Build

From a designated build directory you can give make the following targets.

Target name	Description
build	Builds all components
clean	Removes all intermediate files created by a build: objects, dependencies etc but not final binaries.
clobber	Removes all binaries for all builds as well as anything removed by make clean
install	Runs the install script generated by the build.
components	Builds only user-mode components
kbuild	Builds only kernel-mode components

4.1.5. Basic Command-line Build Options

The following variables are the most common ones that may be set on the command line to influence a build.

Name	Default	Description
BUILD	release	The type of build being performed. Alternatives selectable from command line are **timing** or **debug**.
KERNELDIR	empty	The directory containing the linux kernel sources.
TOP	root of work-space	Determines where components for the build (makefiles and sources files)are located.
V	0	Determines whether text of commands is produced during build. Using make V=1 produces text of executed commands

Naturally, other variables may be overridden too but the effect may be unpredictable in some cases.

A full list of available build options/variables is detailed below.

4.1.6. Frequently Used Build Options

The following are the most common ones that may be set on the command line to influence a driver build.

Setting Description	
BYPASS_SLC=1	Bypass the SGX second level cache, where present.
EGL_DEFAULT_PARAMETER_BUFFER_SIZE	Sets the size of the parameter size in bytes.
DC_NOHW_WIDTH=640	Specify the width of the no hardware display (default 640).
DC_NOHW_HEIGHT=480	Specify the height of the no hardware display (default 480).
LDM_PCI=1	Enables PCI support for the Linux Driver Model.
LDM_PLATFORM=1	Enables platform support for the Linux Driver Model.
NO_HARDWARE=1	Build the no hardware variant of the driver.
PDUMP=1	Build a PDUMP driver. A PDUMP driver is be used to dump parameters that can be run through a simulator.
PVR2D_ALT_2DHW=1	Enables 2D Core Emulation that is required to allow the PVR2D API to be used on SGX systems with no 2D core.
PVRSRV_USSE_EDM_STATUS_DEBUG=1	Enables support for the microkernel status buffer.
SGXCORE	e.g. SGXCORE=535 specifies what SGX core you are building for.
SGXCOREREV	e.g. SGXCOREREV=112 specifies what SGX core revision you are building for.
SGX_FEATURE_MP=1	Enable multi-core SGX support. This option is supported by the *nohw_linux* driver. The number of cores must also be specified, using SGX_FEATURE_MP_CORE_COUNT (described below).

Setting Description	
SGX_FEATURE_MP_PLUS=1	Enable multi-core SGX support with different TA and 3D core counts. This option is supported by the *nohw_linux* driver and must be used in conjunction with SGX_FEATURE_MP=1. This only applies to SGX554. The number of cores must also be specified, using SGX_FEATURE_MP_CORE_COUNT or SGX_FEATURE_MP_CORE_COUNT_TA and SGX_FEATURE_MP_CORE_COUNT_3D (described below).
SGX_FEATURE_MP_CORE_COUNT=*n*	For no hardware drivers with multi-core support enabled (SGX_FEATURE_MP=1), specify the number of SGX cores.
SGX_FEATURE_MP_CORE_COUNT_TA=*n*	For no hardware drivers with multi-core plus support enabled (SGX_FEATURE_MP_PLUS=1), specify the number of SGX TA cores.
SGX_FEATURE_MP_CORE_COUNT_3D=*n*	For no hardware drivers with multi-core plussupport enabled (SGX_FEATURE_MP_PLUS=1), specify the number of SGX 3D cores.
SUPPORT_ACTIVE_POWER_MANAGEMENT=0	Disables active power management.
SUPPORT_LINUX_X86_PAT=0	Prevents the driver from using the Page Attribute Table (PAT) to obtain memory page attributes on x86 based platforms. Disabling PAT support will disable write combining.
SUPPORT_LINUX_X86_WRITECOMBINE=0	Prevents the driver from using write combining in drivers on x86 based platforms.
SUPPORT_OPENGLES1_V1_ONLY=1	Enable building of the OpenGL-ES 1.1 libs/interfaces without the EGL entry points (GLESv1_C[LM] library names) only
SUPPORT_PERCONTEXT_PB=1	Enables the per context parameter buffer size. Use SUPPORT_PERCONTEXT_PB=0 to use shared parameter buffer size.
SUPPORT_SECURE_FD_EXPORT=0	Disables secure user-space handles for exported device memory allocations.
SUPPORT_SGX_HWPERF=1	Enables SGX hardware performance metrics.
SUPPORT_SGX_LOW_LATENCY_SCHEDULING=0	Disables low latency scheduling.
SUPPORT_SLC=1	Enable support for the SGX second level cache, where present.

4.1.7. Extensions Build Options

The following are the options that may be added to CFLAGS to influence driver extensions.
Please refer to OpenGL-ES specifications for detailed description.

Setting Description	
GLES1_EXTENSION_MATRIX_PALETTE	OpenGL-ES 1.x matrix pallete extension

Setting Description	
GLES1_EXTENSION_DRAW_TEXTURE	OpenGL-ES 1.x draw texture extension
GLES1_EXTENSION_QUERY_MATRIX	OpenGL-ES 1.x query matrix extension
GLES1_EXTENSION_TEX_ENV_CROSSBAR	OpenGL-ES 1.x tex crossbar extension
GLES1_EXTENSION_TEX_MIRRORED_REPEAT	OpenGL-ES 1.x tex mirrored extension
GLES1_EXTENSION_TEX_CUBE_MAP	OpenGL-ES 1.x tex cube map extension
GLES1_EXTENSION_BLEND_SUBTRACT	OpenGL-ES 1.x blend substract extension
GLES1_EXTENSION_BLEND_FUNC_SEPARATE	OpenGL-ES 1.x blend func separate extension
GLES1_EXTENSION_BLEND_EQ_SEPARATE	OpenGL-ES 1.x blend equation extension
GLES1_EXTENSION_STENCIL_WRAP	OpenGL-ES 1.x stencil wrap extension
GLES1_EXTENSION_EXTENDED_MATRIX_PALETTE	OpenGL-ES 1.x extended matrix palette extension
GLES1_EXTENSION_FRAME_BUFFER_OBJECTS	OpenGL-ES 1.x frame buffer objects extension
GLES1_EXTENSION_RGB8_RGBA8	OpenGL-ES 1.x RGB8 and RGBA8 extensions
GLES1_EXTENSION_DEPTH24	OpenGL-ES 1.x COMPONENT24 extension
GLES1_EXTENSION_DEPTH32	OpenGL-ES 1.x COMPONENT32 extension
GLES1_EXTENSION_STENCIL8	OpenGL-ES 1.x OES_stencil8 extension
GLES1_EXTENSION_ETC1	OpenGL-ES 1.x ETC extension
GLES1_EXTENSION_PVRTC	OpenGL-ES 1.x PVRTC extension
GLES1_EXTENSION_READ_FORMAT	OpenGL-ES 1.x read format extension
GLES1_EXTENSION_TEXTURE_STREAM	OpenGL-ES 1.x texture stream extension
GLES1_EXTENSION_VERTEX_PROGRAM	OpenGL-ES 1.x vertex program extension
GLES2_EXTENSION_FLOAT_TEXTURE	OpenGL-ES 2.x float texture extension
GLES2_EXTENSION_HALF_FLOAT_TEXTURE	OpenGL-ES 2.x half float texture extension

4.1.8. Specialised Build Options

Setting Description	
DEBUG_LINUX_MEMORY_ALLOCATIONS	Enables some low level kernel memory allocation related debug statistics. (These can be viewed under /proc/pvr/meminfo)
DEBUG_LINUX_MEM_AREAS	LinuxMemAreas are a higher level structure for tracking memory allocations in the kernel space driver. This enables collection of mem area statistics which can be viewed under /proc/pvr/mem_areas
DEBUG_LINUX_MMAP_AREAS	The driver has to track areas of memory that may be mapped in to userspace and this option enables collection of some additional statistics for these areas, viewable under /proc/pvr/mmap

Setting Description	
DEBUG_LINUX_SLAB_ALLOCATIONS	If you enable slab debugging for your kernel then this can be enabled so the driver will pass some extra flags when calling kmem_cache_create which will activate the kernels slab debug features.

4.2. Makefiles

4.2.1. Overview of Makefiles

build/linux2/*/toplevel.mk

This file is an entry point for the core of the build system.

build/linux2/*/Makefile

This file contains build/compiler settings for a particular system build

Any of the Build Options described in section 4.1.6 and 4.1.7 can also be set in this file which saves you from otherwise specifying these options on the command line. (Though variables set in this file can still be overridden on the command line)

4.3. Listing of Makefile Variables

4.3.1. Variables affecting the toolchain flags

Name	Default	Description
SYS_CFLAGS	Empty	Compilation settings specific to this system
SYS_LDFLAGS	Empty	Flags to pass to the link stage of a UM module build
SYS_EXE_LDFLAGS	Empty	Linker flags specific to this system to be used for building executables
SYS_LIB_LDFLAGS	Empty	Linker flags specific to this system to be used for building libraries
OPTIM	Build dependent	Global optimization flags to use for all components. Generally one of -O2, -O3 or -Os.
LIBGCC	Empty	Path to toolchain's libgcc.a. This is often not required but some cross compilers will not automatically link it.

4.3.2. Variables that configure standard Linux commands

Name	Default
BISON	bison
CHMOD	chmod
CP	cp
DOS2UNIX	dos2unix
FLEX	flex
GAWK	gawk
GREP	grep
HOST_AR	ar
HOST_CC	gcc

Name	Default
HOST_CXX	g++
HOST_STRIP	strip
INSTALL	Install
JAR	jar
JAVA	java
JAVAC	javac
M4	m4
MKDIR	mkdir
MV	mv
RM	rm
SED	sed
TAR	tar
TOUCH	touch
ZIP	zip

4.3.3. Variables related to the cross-compile environment

These are all set in embedded/build/makefile.shared_conf and generally should not be overridden

Name	Default
AR	$(CROSS_COMPILE)ar
CC	$(CROSS_COMPILE)gcc
CXX	$(CROSS_COMPILE)g++
OBJCOPY	$(CROSS_COMPILE)objcopy
STRIP	$(CROSS_COMPILE)strip

4.3.4. Excluding APIs/Components From Default Build

The variable EXCLUDED_APIS can be set on the command line or in the environment to selectively disable components that normally build by default.

Previous drivers would use options like SUPPORT_OPENGLES1, SUPPORT_OPENGLES2, SUPPORT_XORG, etc. to control this.

The following tokens can be used:

Name	Default
opengles1	Disables OpenGL ES 1.x and dependent components
opengles2	Disables OpenGL ES 2.0 and dependent components
openvg	Disables OpenVG 1.x and dependent components
opencl	Disables OpenCL 1.x and dependent components
opengl	Disables OpenGL and dependent components
ews	Disables Example Window System and dependent components
unittests	Disables all API unit tests

These options can be specified together, delimited by spaces.

Where required, be careful to specify EXCLUDED_APIS with some escape characters so that the whole string is defined at once.

For example:

```
export EXCLUDED_APIS=”opengl opencl”
make EXCLUDED_APIS=opengles1
```

4.4. Overview

The SGX install.sh and rc.pvr scripts are generated using M4 in conjunction with a local configuration script install.sh. This section documents the the use of the generated install shell script, as well as detailing the generation system including supported directives for install.sh.m4 and rc.pvr.m4, and how GNU m4 is used to achieve this.

4.4.1. install.sh

The install.sh file has several options that allow control of how the driver is installed to the target machine disc image. These are:

Option	Description
-v	Verbose mode
-n	Dry-run mode
-u	Uninstall-only mode
--no-pvr	Don't install PowerVR driver components
--no-x	Reserved for future use.
--no-display	Don't install integrated PowerVR display module
--no-bcdevice	Don't install buffer class device module
--root path	Use path as the root of the install file system

4.4.2. rc.pvr

One of the files auto generated by the build process and installed by the install script is the rc.pvr script which controls loading and unloading of the PowerVR kernel services. This script can be run directly or incorporated into the target systems init sequence. If this is done it would be typically run as part of the run level 2 sequence. The file is by default installed to /etc/init.d.

Option	Description
start	Inserts the driver into the kernel and recreates the PVR services device
stop	Removes the driver from the kernel
reload	Unloads and loads the driver
restart	Unloads and loads the driver (same as reload)

4.5. GNU M4 Install Script Generation

M4 is a macro processing language. Macros are strings of characters using roughly the same conventions as C variable names. Text containing macros is substituted as soon as it is encountered. provided it is not quoted. Substitution recurs until no further substitution is possible.

Macros are defined using the construct

```
 define(name, value)
```

Macros can be given arguments which can be referred to using positional parameters $n (e.g. $1, $2 as in Bourne Shell) in the definition of value above.

If is advisable always to quote arguments to macros and definitions (define is itself a built-in macro) as M4 will attempt macro substitution as soon as it sees any unquoted text, which is not always

desirable. Hence quotations ([[and]] - see below) are used widely in the bulk of the text of install.sh and rc.pvr thus:

```
define([[foo]], [[bar]])
```

Were quotes not used around foo then a subsequent attempt to redefine foo would fail. The quotes around bar ensure that it is not substituted until foo is referenced.

Also note that M4 macro substitutes text on a word basis and NOT a line basis. This makes new-line characters significant: they will not ordinarily be removed to the output stream, but copied to it in addition to any new-lines generated by the macro substitution.

4.5.1. Supported Directives in install.sh.m4 and rc.pvr.m4

Directive & Usage (if different)	Affects	Description
INTERPRETER	install.sh rc.pvr	This causes a #! line to be written at the top of the generated script to give the path to the shell used to interpret it. (This can be overridden by platform versions of install.sh.m4 and rc.pvr.m4 if /bin/sh is located somewhere else)
SET_DDK_INSTALL_LOG_PATH	install.sh	This causes a line setting the path for the install log to be generated. (This can be overridden by platform versions of install.sh.m4 if the install log needs to be located somewhere else)
DEMO **DEMO(program_name)**	install.sh	This causes the program program_name to be copied to /usr/local/bin by install.sh.
DISPLAY_CONTROLLER **DISPLAY_CONTROLLER(name)**	install.sh rc.pvr	This causes install.sh to install a kernel module and a shared library (called libname.so) in the appropriate places. It also causes the kernel module to be loaded by rc.pvr and the appropriate entry crated for it in /dev.
FREEDESKTOP **FREEDESKTOP(path)**	install.sh	This causes install.sh to copy any X11 related files to appropriate places under path. It must be the first FREEDESKTOP macro invoked.
FREEDESKTOP_APPS	install.sh	This causes install.sh to copy into /bin all X applications produced by the build process (provided the names don't start with X).
FREEDESKTOP_HEADERS	install.sh	This causes install.sh to copy into FREEDESKTOP root (under /include) all X header files generated by the build process. (Actually the entire contents of the include hierarchy are copied.)
FREEDESKTOP_LIBRARIES	install.sh	This causes install.sh to copy into /lib all X libraries generated by the build process. (Actually the entire contents of the lib hierarchy are copied.)
FREEDESKTOP_MISC	install.sh	This causes install.sh to copy into FREEDESKTOP root the contents of the man and share hierarchies.

Directive & Usage (if different)	Affects	Description
FREEDESKTOP_SUPPORT	install.sh	This causes install.sh to copy any support packages required by the X window system build. Currently this comprises only the Z compression library (zlib).
FREEDESKTOP_XSERVERS	install.sh	This causes install.sh to copy any compiled X servers into /bin under Free desk-top root (set by FREEDESKTOP). The specific servers copied will depend upon the build in question but would nominally include Xfbdev and Xsgx.
FREEDESKTOP_XSERVER_LIBS	install.sh	This causes install.sh to copy into /lib (under FREEDESKTOP root) only those shared libraries required by the X servers.
LOAD_MODULES	rc.pvr	This directive causes the standard kernel modules to be loaded at boot time via rc.pvr. The standard modules are: PVR services, buffer class driver and the 3rd party display controller. The last two are conditional on the platform configuration.
MAKEDEV_FB	rc.pvr	This causes rc.pvr to contain commands to create the character special node /dev/fb0 when invoked. These are required on some platforms by the X server.
MAKEDEV_DISPLAY_CONTROLLER	rc.pvr	This causes rc.pvr to contain commands to create automatically the character special node in /dev for the display controller when invoked at boot time.
MAKEDEV_PVRSRV	rc.pvr	This causes rc.pvr to contain commands to create automatically the character special node /dev/pvrsrv when invoked at boot time.
MAKEDEV_TTYS	rc.pvr	This causes rc.pvr to contain commands to create the character special nodes /dev/tty0../dev/tty4 when invoked. These are required on some platforms by the X server.
STANDARD_SCRIPTS	install.sh	This causes install.sh to contain commands to copy rc.pvr to /etc/init.d.
STANDARD_KERNEL_MODULES	install.sh	This causes install.sh to contain commands to copy rc.pvr to /etc/init.d.
STANDARD_LIBRARIES	install.sh	This causes install.sh to contain commands to copy the standard libraries to /usr/lib. Currently these are: libGLES_CM.so, libpvrmmap.so plus any defined 3rd-party driver libraries (e.g. buffer class or display class drivers).

Directive & Usage (if different)	Affects	Description
TARGET_HAS_DEPMOD **TARGET_HAS_DEPMOD(TRUE|FALSE)**	install.sh	Determines whether the install script install.sh attempts to create the module dependencies manually or gets /sbin/depmod to do this. If TRUE is supplied, depmod will be invoked; otherwise install.sh will manually modify modules.dep to get the dependencies set correctly. (See also: TARGET_RUNS_DEPMOD_AT_BOOT)
TARGET_RUNS_DEPMOD_AT_BOOT **TARGET_RUNS_DEPMOD_AT_BOOT(TRUE|FALSE)**	install.sh	Determines whether depmod is invoked at installation time to fix the module dependencies or not. If TRUE then depmod is invoked at installation time by install.sh; otherwise, it is not invoked and assumed to be invoked by the targets native boot process. (See also: TARGET_HAS_DEPMOD)

VERSION
 	
VERSION(string)
	rc.pvr
	install.sh
		This causes an identifier string to be emitted containing string. Thus VERSION(abc) causes this text to be emitted:
		`# Auto generated from`
		`#  abc`

4.5.2. Quoting Conventions

By default in M4, open quotation is ` and close quotation is '. These have been changed deliberately to [[ and ]] respectively in order to allow ` and ' to be used in shell scripts as needed.

Note that positional parameters are *always* expanded whether quoted or not. This is awkward, but occasionally useful. Note also that it is NOT possible (according to the M4 manual) to escape an open-quotation character. This is inconvenient and seems to serve no useful purpose.

4.5.3. White Space

Use of white space in macro invocations is significant as follows:

- leading white space before a macro argument is ignored.
- trailing white space in a macro is consider part of the argument

Thus:

```
$ m4
ifelse(A, A, equal, not equal)
equal
ifelse(  A,     A, equal, not equal)
equal
ifelse(A, A ,equal, not equal)
not equal
^D
$
```

Note: White-space include new-line/line-feed characters as well as space and tab characters.

dnl

The dnl macro is used to swallow remaining text to the end of the current line of the input stream, which includes any terminating new-line characters. This is used regularly in macro definitions to avoid extra new lines in text that is intentionally diverted to the output stream.

```
define(mymacro?, dnl
[[do some stuff
do some more stuff
]])dnl
```

Here the first dnl invocation ensures that the newline at the end of the line is not included in the macro definition, i.e. it will not be copied to the output stream if someone expands mymacro. (As it happens, white-space conventions ensure this for us.) The second dnl invocation (on the last line) stops the new line at the end of the macro definition line from being copied to the output stream. Thus if the input stream contains a line

```
mymacro
```

then the output stream will contain

```
do some stuff
do some more stuff
<blank line>
```

The final blank line is caused by the newline character at the end of the input stream line containing the mymacro invocation. To remove that blank line use

```
mymacro()dnl
```

Correct use of dnl is critical to getting correct output from m4 macros and invocations.

4.5.4. Diversions

For anything more than a few script line replacements (for which one could easily use sed(1)) good diversion discipline is a must, simply in order to smoothly control how the output stream is generated. This partly comes from a need to make m4 scripts readable (well, almost) and thus the need to curtail unwanted text. M4's concept of diversions comprises temporary areas to which the output stream can be diverted until a later time. The following diversions are standardised as parameters to the divert built-in macro:

Diversion	Description
<0	all output is discarded
0	all output goes directly to the standard output stream (stdout - file descriptor #1)
>0	all output goes to a temporary place.

Diversions with positive numbers are diverted back to the standard output stream automatically and in increasing numerical order when m4 exits normally. Alternatively then can be diverted back to the standard output stream using the undivert built-in macro. Subsequent diversions to the same diversion number always append new output to the existing diverted output. Once undiverted, a diversion's contents are lost.

Understanding install.sh.m4 and rc.pvr.m4

The quotation characters have been specifically set to [[and]] to allow ' and ` and [and] to be used in shell scripts.

The following diversion numbers are allocated.

Diversion#	Owner	Use
1	common.m4	shell script interpreter directive (#!/bin/sh)
2	common.m4	the IMG copyright notice
3	common.m4	version trace log
4-49	common.m4	reserverd
50-99	*.m4	reserverd for use by libraries
50	install.sh.m4	script functions bail() and install_it(), invocation checking
50	rc.pvr.m4	check for $1=start
100	any	main output of generated text

Diversion#	Owner	Use
200	any	trailer - tidy up text from libraries
201	common.m4	trailer - tidy up text from common.m4

Diversion Macros

The following macros have been added for convenience to supplement the built-in divert and undivert macros.

Name	Use
_current_divert	hidden - retain current diversion number
pushdivert	saves current diversion number on top of stack and sets diversion to parameter $1
popdivert	reverts back to diversion number on top of stack
pushmain	saves current diversion number on top of stack and sets diversion to 100 (main body)

The main purpose of these diversion macros is to allow macros to invoke other macros without concerning themselves greatly about which diversion is currently in use.

Other Defined Macros

The following (key among others) are defined in build/scripts/common.m4.

Name	Default value	Description
FALSE	0	Represents boolean false
TRUE	1	Represents boolean true
SHLIB_DESTDIR	/usr/lib	Where shared libraries are placed
RC_DESTDIR	/etc/init.d	Where install scripts are placed
X11BIN_DESTDIR	/usr/X11R6	Where X11 binaries are placed - unused
X11LIB_DESTDIR	/usr/X11R6	Where X11 libraries are placed - unused
DEMO_DESTDIR	/usr/local/bin	Where demonstration programs (egl_test etc.) are placed

4.5.5. Example Macros

The following are taken from install.sh.m4

```
define([INSTALL SHARED LIBRARY]?, [[pushmain()dnl
install_it $1 SHLIB_DESTDIR "shared library"
popdivert()]])
```

The pushmain() call ensures output is directed to diversion #100 (the main output body); a dnl invocation ensures the newline character at the end-of-line is not copied to the output stream. The second line is the actual text that will be output; here the entirety of the line (text and newline) is copied. The final line redirects output to which ever stream whas in use before the pushmain() invocation. No dnl invocation is needed at the end because at the point of the macro definition output is already discarded.

```
define([INSTALL KERNEL MODULE]?, [[pushmain()dnl
install_it $1 MOD_DESTDIR "kernel module"
ifelse(TARGET_RUNS_DEPMOD_AT_BOOT,TRUE,,TARGET_HAS_DEPMOD,FALSE,[[dnl
FIXUP_MODULES_DEP($1, $2)dnl
]],[[dnl
if [ "$host" = 1 ]; then
        FIXUP_MODULES_DEP($1, $2)dnl
fi
]])dnl
popdivert()]])
```

This is a much more complex example, particularly because of the use of the ifelse macro.

- dnl is used on all lines where there is a macro invocation which supplies its own new-line characters.
- dnl is also used on lines which are part of macros and don't contribute to generated text.
- don't confuse the "if [$host" text with the "ifelse(" macro. The former is actually generated code; the latter is an m4 macro invocation.
- where possible, quotes and brackets and dnl invocations are kept as seperate from text that contributes to the final script.
- The use of ifelse() here is particularly unpleasant and is a form of embedded two conditions in a single if clause.
- If TARGET_RUNS_DEPMOD_AT_BOOT is TRUE then no text is generated (hence two consecutive commas);
- If TARGET_RUNS_DEPMOD_AT_BOOT is not TRUE then the TARGET_HAS_DEPMOD is compared with FALSE.
- If that comparison succeeds then the FIXUP_MODULES_DEP text is used; otherwise the "if [$host" text is used.

More detailed information on m4 programming can be found in the tex info pages, obtaining on linux using

```
$ info m4
```

5. Proc File System

The PowerVR services kernel module uses the proc file system to report back run time information about its activity. This information can be found in /proc/pvr. For example:

```
 # ls -l /proc/pvr/
-r--r--r-- 1 root root 0 May 16 12:29 bridge_stats
-rw-r--r-- 1 root root 0 May 16 12:29 debug_level
-r--r--r-- 1 root root 0 May 16 12:29 mem_areas
-r--r--r-- 1 root root 0 May 16 12:29 meminfo
-r--r--r-- 1 root root 0 May 16 12:29 mmap
-r--r--r-- 1 root root 0 May 16 12:29 nodes
-r--r--r-- 1 root root 0 May 16 12:29 queue
-r--r--r-- 1 root root 0 May 16 12:29 ra_info_2D
        ……
-r--r--r-- 1 root root 0 May 16 12:29 ra_info_TestChipDeviceMemory
-r--r--r-- 1 root root 0 May 16 12:29 ra_segs_2D
        ……
-r--r--r-- 1 root root 0 May 16 12:29 ra_segs_TestChipDeviceMemory
-r--r--r-- 1 root root 0 May 16 12:29 version
```

The different files are documented below. In the case of the *ra_info* and *ra_segs* files, some files are specific to particular platforms. You can read any file using cat as shown. (Note that using *more* or *less* commands can be problematic when viewing these pseudo files)

5.1. bridge_stats

This shows some statistics relating to the interface between the user and kernel mode parts of the driver.

```
# cat /proc/pvr/bridge_stats
Total ioctl call count = 147957
Total number of bytes copied via copy_from_user = 23483760
Total number of bytes copied via copy_to_user = 1208068
Total number of bytes copied via copy_*_user = 24691828

Bridge Name                                  | Wrapper Function                          |
Call Count | copy_from_user Bytes | copy_to_user Bytes
PVRSRV_BRIDGE_ENUM_DEVICES                     PVRSRVEnumerateDevicesBW                    4
0                    800
PVRSRV_BRIDGE_ACQUIRE_DEVICEINFO               PVRSRVAcquireDeviceDataBW                   4
48                   32
PVRSRV_BRIDGE_RELEASE_DEVICEINFO               DummyBW                                     0
0                    0
PVRSRV_BRIDGE_CREATE_DEVMEMCONTEXT             PVRSRVCreateDeviceMemContextBW              2
16                   1304
PVRSRV_BRIDGE_DESTROY_DEVMEMCONTEXT            PVRSRVDestroyDeviceMemContextBW             0
0                    0
PVRSRV_BRIDGE_GET_DEVMEM_HEAPINFO              PVRSRVGetDeviceMemHeapInfoBW                4
48                   2592
PVRSRV_BRIDGE_ALLOC_DEVICEMEM                  PVRSRVAllocDeviceMemBW                      206
4944                 16480
PVRSRV_BRIDGE_FREE_DEVICEMEM                   PVRSRVFreeDeviceMemBW                       0
0                    0
<snip bridge name details>
```

5.2. debug_level

This allows read and write to the global debug level variable (C global variable gPVRDebugLevel) used by kernel-mode services (BUILD=debug only).

```
 # cat /proc/pvr/debug_level
 2
 #
```

The debug_level file is only accessible for writing by root.

```
# echo "3" >/proc/pvr/debug_level
# cat /proc/pvr/debug_level
3
#
```

5.3. meminfo

This details all memory and IO allocations or reservations done using, vmalloc, kmalloc, ioremap, alloc_pages, kmem_cache_alloc.

These statistics are enabled via the DEBUG_LINUX_MEMORY_ALLOCATIONS build option.

```
-bash-2.05b# cat /proc/pvr/meminfo |head -21
Current Water Mark of bytes allocated via kmalloc           : 35972 bytes
Highest Water Mark of bytes allocated via kmalloc           : 65312 bytes
Current Water Mark of bytes allocated via vmalloc           : 2363392 bytes
Highest Water Mark of bytes allocated via vmalloc           : 14146812 bytes
Current Water Mark of bytes allocated via alloc_pages       : 0 bytes
Highest Water Mark of bytes allocated via alloc_pages       : 7598080 bytes
Current Water Mark of bytes allocated via ioremap           : 65840 bytes
Highest Water Mark of bytes allocated via ioremap           : 65840 bytes
Current Water Mark of bytes reserved for "IO" memory areas  : 0 bytes Highest Water Mark of
bytes allocated for "IO" memory areas : 1228800 bytes Current Water Mark of bytes allocated
via kmem_cache_alloc  : 1192 bytes Highest Water Mark of bytes allocated via kmem_cache_alloc
: 3392 bytes

The Current Water Mark for memory allocated from system RAM : 2400556 bytes
The Highest Water Mark for memory allocated from system RAM : 21813596 bytes
The Current Water Mark for memory allocated from IO memory  : 65840 bytes
The Highest Water Mark for memory allocated from IO memory  : 1294640 bytes

Details for all known allocations:
Type             CpuVAddr CpuPAddr Bytes      PID   PrivateData Filename:Line
KMALLOC          c7652800 00000000 512        345   <NULL>      hash.c:183
<snip per allocation details>
```

The fields are as follows:

Field	Description
Type	Possible values: KMALLOC, VMALLOC, ALLOC_PAGES, IOREMAP, IO, KMEM_CACHE_ALLOC. This details what API was used to allocate/reserve the memory. Note “IO” memory doesn’t have a corresponding Linux API, it simply represents IO memory that we own (VRAM) which we have reserved.
CpuVAddr	The first cpu (kernel) virtual address associated with the memory or NULL if none exists.
CpuPAddr	The first CPU physical address associated with the first page of the corresponding memory.
Bytes	The size of the allocation in bytes.
PID	The process ID that was current at the time of allocation.
PrivateData	The intention was to have the kmem cache name here where appropriate but that’s currently not available.
Filename:Line	The code filename and line where the memory was allocated.

5.4. mem_areas

LinuxMemAreas are a higher level structure used for tracking memory allocated on behalf of the Buffer Manager and for memory that needs to be mapped into userspace.

These statistics are enabled via the DEBUG_LINUX_MEM_AREAS build option.

```
-bash-2.05b# cat /proc/pvr/mem_areas |head -7
Number of Linux Memory Areas: 71
At the current water mark these areas correspond to 2363396 bytes (excluding SUB areas)
At the highest water mark these areas corresponded to 22966276 bytes (excluding SUB areas)

Details for all Linux Memory Areas:
psLinuxMemArea LinuxMemType             CpuVAddr CpuPAddr Bytes    Pid   Flags
c7bb74d8       LINUX_MEM_AREA_IOREMAP   c8da0000 c0231000 65536    318
00041804=(WRITECOMBINE|KERNEL_ONLY)
<snip per mem area details>
```

The fields are as follows:

Field	Description
psLinuxMemArea	The kernel virtual address of the meta structure for this area of memory.
LinuxMemType	The types of memory area
CpuVAddr	The first cpu (kernel) virtual address associated with the memory area or NULL if none exists.
CpuPAddr	The first CPU physical address associated with the first page of the corresponding memory area.
Bytes	The size of the allocation in bytes.
PID	The process ID that was current at the time of allocation.
Flags	The heap attributes as passed down from the Buffer Manager.

5.5. mmap

This lists all memory areas that are available to user-mode services.

These statistics are enabled via the DEBUG_LINUX_MMAP_AREAS build option.

```
-bash-2.05b# cat /proc/pvr/mmap
Allocations registered for mmap: 288
In total these areas correspond to 13373440 bytes
psLinuxMemArea UserVAddr KernelVAddr CpuPAddr MMapOffset ByteLength LinuxMemType
Pid   Name     Flags
edeb3180       b7cc7020 ef46a020 a0006020 000a0006   60         LINUX_MEM_AREA_SUB_ALLOC 2213
LINUX_MEM_AREA_IOREMAP 04024200(WRITECOMBINE|MULTI_PROCESS)
edeb3120       b7cc6000 ef4be000 a0046000 000a0046   16         LINUX_MEM_AREA_SUB_ALLOC 2213
LINUX_MEM_AREA_IOREMAP 04024200(WRITECOMBINE|MULTI_PROCESS)
edeb3480       b76eb000 0        a0047000 000a0047   6139904    LINUX_MEM_AREA_SUB_ALLOC 2213
LINUX_MEM_AREA_IO 04014200(WRITECOMBINE|SINGLE_PROCESS)
<snip per mmap area details>
```

The fields are as follows:

Field	Description
psLinuxMemArea	MMapable areas are all backed by a LinuxMemArea structure, and this is the kernel space address to that structure.
UserVAddr	CPU user mode virtual address for this mmapable area. If an area has been mapped into the address space of several processes, the area will be shown multiple times, once for each process.
KernelVAddr	A convenience that is also available from /proc/pvr/mem_areas by cross referencing the psLinuxMemArea field showing the CPU kernel virtual address associated with this mmapable area or NULL if none exists.
CpuPAddr	A convenience that is also available from /proc/pvr/mem_areas by cross referencing the psLinuxMemArea field showing the CPU physical address of the first page associated with this mmapable memory area.
MMapOffset	The unique page aligned offset that must be supplied to the mmap(2) system call from user-mode services in order to map the chunk.

Field	Description
ByteLength	A convenience that is also available from /proc/pvr/mem_areas by cross referencing the psLinuxMemArea field showing the size of the mmapable area in bytes.
LinuxMemType	A convenience that is also available from /proc/pvr/mem_areas by cross referencing the psLinuxMemArea field, showing the type of memory area backing the mmapable area.
Pid	The ID of the process associated with this mapping.
Name	A string indicating why the memory has been mapped.
Flags	The heap attributes as passed down from the Buffer Manager.

5.6. nodes

This contains a list of all known nodes in the services device node table.

```
# cat /proc/pvr/nodes
Registered nodes
Addr     Type     Class    Index Ref pvDev     Size Res
c036cbe0 extern   display      1   1  c6cdd1e0  120  00000000
c036cb20 mbx1     3D           0   1  cc9e7038    0  00000000
#
```

The fields are as follows:

Field	Description
Addr	the address of the PVRSRV_DEVICE_NODE instance in kernel memory.
Type	the type of the node from sDevId memory of the PVRSRV_DEVICE_NODE structure.
Class	the type of the node from sDevId memory of the PVRSRV_DEVICE_NODE structure.
Index	the device of the node from the PVRSRV_DEVICE_NODE structure.
Ref	the number of other entities references this node.
pvDev	the pvDevice pointer from the PVRSRV_DEVICE_NODE structure.
Size	the size of the data associated with pvDev.
Res	the RESMAN_ITEM pointer for the node.

5.7. queue

This lists all defined command queues in the system and information about activity on each queue.

```
# cat /proc/pvr/queue
Command Queues
Queue    CmdPtr?   Command Size DevInd?  DSC  SSC  #Data ...
cc9e85d0 cc9e8728       0  256      0    1    1    196
cc9e7d50 <empty>
cc9e74d0 <empty>
#
```

The fields are as follows:

Field	Description
Queue	The address of the Queue in kernel memory.
CmdPtr	The address of the next command in the queue. If the queue has no pending or active commands then the text <empty> is shown.

Field	Description
Command	The command number.
Size	The size of the data supplied for the command.
DevInd	The index of the device node for which this command is queued.
DSC	The number of destination sync items.
SSC	The number of source sync items.
#Data	The number of data items in the command.

5.8. ra_info_*

Those shows collected statistical information about a given resource allocator (arena). Note that some statistics are collected globally, whereas others are collected on a per process basis. For example, *ra_info_KernelData* is global:

```
 # cat /proc/pvr/ra_info_KernelData
quantum                 4096
import_handle           EDE1D5E0
span count              1
live segment count      12
free segment count      0
free resource count     82976768 (0x4f22000)
total allocs            12
total frees             0
import count            0
export count            0
```

Per process statistics are located in a subdirectory named with the process ID of the associated process. For example, the 3D parameter statistics for the process with ID *2213*, would be located in subdirectory *2213*:

```
# cat /proc/pvr/2213/ra_info_3DParameters
quantum                 4096
import_handle           EDE17580
span count              1
live segment count      3
free segment count      0
free resource count     255885312 (0xf408000)
total allocs            3
total frees             0
import count            0
export count            0
```

The fields are as follows:

Field	Description
Quantum	The size to which all allocations are rounded.
import_handle	The data required to be passed to the import function supplied.
span_count	The number of spans in the arena. This should equal the import count minus the export count.
live segment count	The number of allocations made from the arena. This should equal the total allocations minus the total frees.
free segment count	The number of free areas from which allocations can be made (presuming a size match).
free resource count	The total amount of resource that is considered free in the arena.
total allocs	The number of successful calls to RA_Alloc() for this arena.
total frees	The number of calls to RA_Free() for this arena.
import count	The number of times the resource allocator has had to import resource to satisfy a call to RA_Alloc().

Field	Description
export count	The number of times the resource allocator has freed previously imported resources.

5.9. ra_segs_*

This shows the current state of the segments in a given resource allocator (BUILD=debug only). As with the *ra_info* statistics, some statistics are collected globally, whereas others are collected on a per process basis, the latter being found in a subdirectory named with the process ID of the associated process.

```
 # cat /proc/pvr/ra_segs_KernelData
 Arena "KernelData"
Base         Size Type Ref
fa000000     1000 live 00000000
fa001000     1000 live 00000000
fa002000     1000 live 00000000
fa003000     4000 live 00000000
fa007000    40000 live 00000000
fa047000    40000 live 00000000
fa087000     1000 live 00000000
fa088000    11000 live 00000000
fa099000    11000 live 00000000
fa0aa000    11000 live 00000000
fa0bb000    11000 live 00000000
fa0cc000    11000 live 00000000
fa0dd000  4f22000 free 00000000
```

The fields are as follows:

Field	Description
Base	The base of the item in the addressing space of the specific arena.
Size	The size of the item in whatever units the specific arena uses.
Type	The type of item.
Ref	Some void * value associated with the underlying resource from which the item was allocated.

The item types are as follow:

Type	Description
live	This item has been allocated and returned to a caller of RA_Alloc().
free	This item is available to form the basis of future allocations. Freed resource is always merged with neighbouring freed resource in order to make a single large resource chunk; thus, a single call to RA_Free() might not be immediately obvious in the output.
span	This is a marker separating resource items all belonging to a single import. There are two such markers - one before and one after all the resources made against that import. The first span item to be the size of the import; the size of the second span item is always set to zero.

5.10. version

This shows the running services's version, build target and build type.

```
 # cat /proc/pvr/version
 Linux version 2.6.14 (user@buildhost) (gcc version 3.4.3) 11:12:20 GMT 2006
 #
```

5.11. /proc/slabinfo

While this isn't actually in /proc/pvr, it's worth mentioning because all of the POOL objects created by the buffer manager appear here.

```
 # head /proc/slabinfo
 slabinfo - version: 2.0
 # name    <active_objs> <num_objs> <objsize> <objperslab> <pagesperslab>
 : tunables <batchcount>   <limit> <sharedfactor>
 : slabdata <active_slabs> <num_slabs> <sharedavail>
 img-mmap      5          113         32        113           1
 : tunables     120           60          0
 : slabdata        1            1          0
#
```

All the slabs created by creating POOL objects begin with the "img" string; in most embedded environments they'll always be at the top of the file since new slabs are added at the front of a linked list. While the description of the output here is limited, it's worth noting that when pool objects are freed, the active_objs count won't immediately decrease because recently freed objects are cached for quick re-allocation for a short time after freeing; the active_objs count isn't adjusted until the object is removed from the cache and returned to the slab it was taken from. The delay is usually only a few seconds.

6. Application hints file

Possible run time configuration parameters are specified in the appropriate software functional specification for the software component in question. The run time configuration parameters are controlled form a file named “powervr.ini” which is stored in the system /etc directory.

The format of this file is:

```
[default]
Parameter1=Value
Parameter2=Value
…

[AnotherAppExeName]
…
```

If `Default` is used as the AppExeName then the values apply to all applications unless otherwise overridden by an application specific section. Please note a mandatory empty line at the end of the file.

6.1. Run time configuration settings

The table below provides some example settings with their defaults. Client APIs may have further settings that use this common mechanism, and those will be documented in the appropriate client API documentation.

Name	Default setting	Description
ForceExternalZBuffer	0	Forces driver to use external depth buffer.
ParamBufferSize	EGL_DEFAULT_PARAMAMETER_BUFFER_SIZE	Sets the size of the parameter buffer in bytes.
ExternalZBufferXSize	50 (100 when desktop OpenGL is supported)	Sets the depth buffer X size.
ExternalZBufferYSize	50 (100 when desktop OpenGL is supported)	Sets the depth buffer Y size.

7. Running the Drivers

After the drivers have been built, installed to the target system, and loaded on the target system as described in sections “Build System” and “Installation Script System” applications that use the driver can be run. The platform guide for a specific target system will give any specific installation instructions for a given system, but in general there will be a small number of unit tests installed to /usr/local/bin on the target system that can be quickly used to test driver functionality.

7.1. Running PDUMP Drivers

The drivers have a build variant documented in section “Build System”, that is called ‘pdump’, this mode of operation can be used to generate parameters usable for HW debug and HW simulation.

Running applications on pdump drivers involves one extra step versus running normally. This step involves running an application installed by the driver installation process (to /usr/local/bin) called “pdump”, this requires arguments to control which application frames are grabbed into the output files. The argument is of the form –cX-Y where X is the frame to start capture and Y is the frame where capture stops (the –Y may be omitted to capture only frame X). Pdump should be run interactively, and not in the background.

Once the pdump application is running, the test application should then be run. When the test application has exited, pressing ‘x’ in the interactive shell running pdump will cause it to exit if it has nor exited automatically.

7.2. PDUMP Output

Pdump will produce the following files in the directory where it was executed:

- out2.prm,
- out2.txt,
- out_drvinfo.txt,

7.3. Kernel module parameters

Some kernel modules can take parameters at module load time.

7.3.1. dc_nohw display driver

The width, height, and depth of the display can be overridden by setting the *PVR_DISPLAY_PARAMS* environment variable before loading the driver. For example, for a display size of 1280 by 1024, 32 bits deep:

```
PVR_DISPLAY_PARAMS=”width=1280 height=1024 depth=32” /etc/init.d/rc.pvr start
```

To use the default width and height (640x480), but with a depth of 16 bits:

```
PVR_DISPLAY_PARAMS=”depth=16” /etc/init.d/rc.pvr start
```

The only display depths currently supported are 16 and 32 bits, with pixel formats *RGB565* and *ARGB888*, respectively.