PS4-GPU-DOC

DRAFT 0.1.0

Table of Contents

1. Introduction

This document contains information on the PS4s GPU, more specifically the higher-level components of it. All information comes from publicly available sources and my own reverse engineering of the PS4s firmware 9.0.0 version of the GNM driver (the Graphics API the PS4 uses to issue commands to the GPU, like OpenGL, Direct-X, or Vulkan).

The hope is that this documentation will lead to furthering of Homebrew applications on the PS4, and hopefully emulation (of which I think there is a *lot* of potential for).

1.1 What is the PS4s GPU Anyway

The PS4 pro GPU is an AMD GPU based off of the Radeon GCN 4 Polaris 10 (Ellesmere) architecture[1] [2]. The GCN architecture is the underlying specification the hardware is based off of, though surprisingly isn't very important to this document except for shader machine-code, as the GCN ISA[3] (Instruction Set Architecture) is needed to understand it.

> 1. The original PS4s GPU actually is based off GCN 1.1, but from here on out all of the information is assuming the PS4 pros GPU (The distinction doesn't matter much for this higher-level document)

> 2. I'm actually not sure exactly if it was precisely Polaris 10 or maybe some slightly modified version, but the distinction is irrelevant as, if there are any changes, they are purely hardware based and doesn't change the software at all.

> 3. The link is actually to the GCN 3 ISA, but it's mostly the same to GCN 4, there just isn't a version of the documentation that documents GCN 4 specifically, as all of the changes are hardware optimizations and not actual ISA changes.

1.2 Where Did You Find This Out Anyhow

You may be asking yourself, how do you even reverse engineer something as *complicated* as a relatively modern GPU that has so many moving parts no single engineer that even worked on the damn thing probably knows how the entire design interconnects and works with itself. Well the answer is simple really... AMD *told* me how it works. No really. I'm not even breaching any NDAs or anything like that. AMD is actually a fairly notoriously open-source company, and so I'd like you to meet someone who will be helping significantly with this documentation...

1.2.1 Meet Your PAL

The Platform Abstraction Library... rolls right off the tounge doesn't it? Okay I'll be honest, this is actually only *one* part to a broader open-source driver that actually is the *real* documentation. I just couldn't resist the play on words. PAL, however, is an essential part to that driver, as it explains how many components of the GPU works including the things that get submitted to the lower-level driver that interacts with the GPU via PCI-E directly, like the PM4 format command buffers. Coincidentally also what the PS4s own GNM API directly submits to the GPU. However that isn't enough sometimes to figure out everything, at that point it might be useful to look at something that *uses* PAL to implement a well known API... say... Vulkan?

1.2.2 XGL

That's where XGL comes in. XGL is an implementation of the Vulkan API, using PAL as an abstraction over the hardware. However, crucially, PAL is similar in abstraction scope to GNM, the PS4s API. Thus a lot of what you can learn about how PAL is used, will teach you about how GNM is used. And the best part is, XGL is an implementation of the *full* Vulkan API, meaning that everything that games will pretty much ever use, is right there.

> Note: One more thing I'd like to fit in here. Another invaluable tool is the Radeon GPU Analyzer software, it allows you to write GLSL and translate it directly into GCN machine-code, allowing you to see exactly how it works. Can be useful for generally just figuring out where certain GCN instructions are used where.

1.3 So Why Mention All of This

Because I don't want this document to just be about the technical information I've gathered (though we'll get to that part shortly), but also about *how* to look for the information I've gathered. So that hopefully collaboration can improve documentation over time, and correct errors that I've made in my own research.

Nothing I'm presenting here is necessarily ground-breaking or new, anyone could've figured this stuff out pretty easily. But I am the first (I think?) to put it all in one place that is easily accessible and maintainable when new information gets discovered.

2. Orbis OS

The PS4s operating system is named Orbis, but it's actually based off FreeBSD 9.0, thus most of its functionality you can expect is very POSIX like.

The OS is split up into different libraries, *.prx files. Well actually *.sprx files before decryption that are signed and can only be decoded using a PS4 (at time of writing). But really *.prx files are just ELF (Executable Linkable Format) files, a well documented format that needs no introduction.

We won't go too in-depth into the OS in this document, basically the only library we care about is `libSceGnmDriver.sprx`[1] (And the libraries that support that one, but most of the calls it makes are just standard library stuff).

> 1. There are actually two versions of this library. One for the Original PS4 / Slim, and one for "Neo mode" aka the PS4 pro. The version I reverse engineered was the PS4 pro version.

3. PM4

PM4 is the format that the PS4 uses for its Command Buffers. The format consists of packets of data, which are essentially commands that tell the GPU to do things like change the current state of the GPU, and draw geometry, even instancing and compute are supported.

3.1 Commands

This actually won't be a long section as all the documentation you need is right here[1], it'd also be advisable to look at the XGL source code to see what the commands are used for. However one thing that the PM4 document doesn't tell you is all the opcodes for the commands listed in it (for some reason). So here they are. (Taken from public sources)

Name	Opcode
NOP	0x10
SET_BASE	0x11
CLEAR_STATE	0x12
INDEX_BUFFER_SIZE	0x13
DISPATCH_DIRECT	0x15
DISPATCH_INDIRECT	0x16
ATOMIC_GDS	0x1D
ATOMIC	0x1E
OCCLUSION_QUERY	0x1F
SET_PREDICATION	0x20
COND_EXEC	0x22
PRED_EXEC	0x23
DRAW_INDIRECT	0x24
DRAW_INDEX_INDIRECT	0x25
INDEX_BASE	0x26
DRAW_INDEX_2	0x27
CONTEXT_CONTROL	0x28
INDEX_TYPE	0x2A
DRAW_INDIRECT_MULTI	0x2C
DRAW_INDEX_AUTO	0x2D
DRAW_INDEX_IMMED[2]	0x2E

Name	Opcode
NUM_INSTANCES	0x2F
DRAW_INDEX_MULTI_AUTO	0x30
INDIRECT_BUFFER_CONST	0x33
STRMOUT_BUFFER_UPDATE	0x34
DRAW_INDEX_OFFSET_2	0x35
WRITE_DATA	0x37
DRAW_INDEX_INDIRECT_MULTI	0x38
MEM_SEMAPHORE	0x39
MPEG_INDEX[2]	0x3A
COPY_DW	0x3B
WAIT_REG_MEM	0x3C
MEM_WRITE[2]	0x3D
INDIRECT_BUFFER	0x3F
COPY_DATA	0x40
PFP_SYNC_ME	0x42
SURFACE_SYNC	0x43
ME_INITIALIZE[2]	0x44
COND_WRITE	0x45
EVENT_WRITE	0x46
EVENT_WRITE_EOP	0x47
EVENT_WRITE_EOS	0x48
PREAMBLE_CNTL	0x4A
ONE_REG_WRITE[2]	0x57
LOAD_SH_REG	0x5F
LOAD_CONFIG_REG	0x60
LOAD_CONTEXT_REG	0x61
SET_CONFIG_REG	0x68
SET_CONTEXT_REG	0x69
SET_CONTEXT_REG_INDIRECT	0x73

Name	Opcode
SET_SH_REG	0x76
SET_SH_REG_OFFSET	0x77
LOAD_CONST_RAM	0x80
WRITE_CONST_RAM	0x81
DUMP_CONST_RAM	0x83
INCREMENT_CE_COUNTER	0x84
INCREMENT_DE_COUNTER	0x85
WAIT_ON_CE_COUNTER	0x86
WAIT_ON_DE_COUNTER	0x87
WAIT_ON_DE_COUNTER_DIFF	0x88

Any not listed are unknown at this time and will be updated as more information is gathered.

> 1. Here's a mirror just incase the original link goes down.

> Note: Unless otherwise specified values are from PAL si_ci_vi_merged_pm4_it_opcodes.h

> 2. Taken from Mesa3d source code

4. Registers

PAL source code (There are so many we can't hope to list them all but the ones that will be listed here are the important ones that have been figured out their meaning and how to use them)

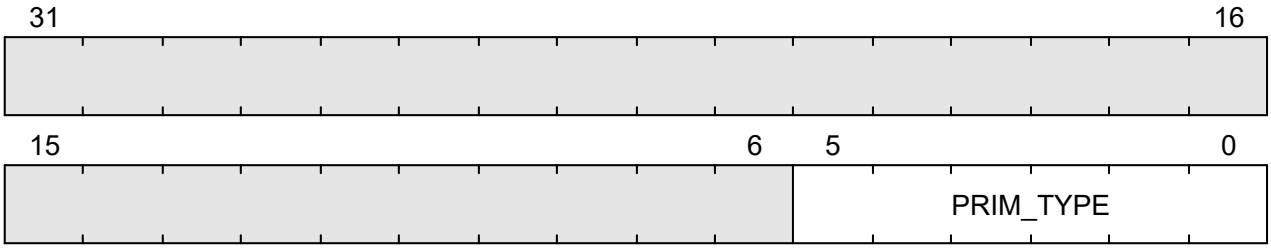
4.1 Context Registers

TODO

4.2 Config Registers

Config registers are present at offset `0x2000`.

4.2.1 VGTPrimitiveType



- PRIM_TYPE: The Primitive type to render when submitting Draw Calls
  - POINT_LIST: `0x00`
  - LINE_LIST: `0x01`
  - LINE_STRIP: `0x02`
  - TRIANGLE_LIST: `0x03`
  - TRIANGLE_STRIP: `0x04`
  - RECT_LIST: `0x05`
  - QUAD_LIST: `0x06`
  - QUAD_STRIP: `0x07`
  - LINE_LIST_ADJ: `0x08`
  - LINE_STRIP_ADJ: `0x09`
  - TRIANGLE_LIST_ADJ: `0x0A`
  - TRIANGLE_STRIP_ADJ: `0x0B`
  - PATCH: `0x0C`
  - TRIANGLE_FAN: `0x0D`
  - LINE_LOOP: `0x0E`
  - POLYGON: `0x0F`
  - TWOD_RECT_LIST: `0x10`

Register Offset: `0x256`

Description: Describes the Primitive type to use when rendering geometry.

4.3 Shader Registers

TODO

5. Events

TODO

6. GNM

The GNM API is the API used by the PS4 to abstract over the lower-level device driver (which it issues commands to via the ioctl function), think of it as like OpenGL, Direct-X, or Vulkan. However unlike those APIs, GNM is a bit lower-level, actually low-level enough to give games higher-level driver-like control over the GPU.

> Note: Quick note about GNMX. You may have heard about GNMX as an even higher-level API, well unfortunately there's not much we can do to get information about that API as it seems to be statically linked within games. Thus there is no information about function names or anything of the sort. But luckily that doesn't really matter as GNMX just boils down to calling GNM functions, so we can safely ignore it.

6.1 API Functions

All functions are currently untested on real hardware. All information is purely based off decompiled code from 9.0.0 firmware.

> Note: All function names come from the names embedded in the ELF file. Some parameter names come from error messages embedded in the code.

6.1.1 sceGnmSubmitCommandBuffers

```
int __stdcall sceGnmSubmitCommandBuffers(uint32_t length,
    void* dcbGpuAddrs[], uint32_t dcbSizesInBytes[],
    void* ccbGpuAddrs[], uint32_t ccbSizesInBytes[]);
```

- length: The amount of Command Buffers (for both the DE and CE) to submit.
- dcbGpuAddrs: A list of pointers to the addresses of the Command Buffers to submit to the DE. (Cannot be null)
- dcbSizesInBytes: A list of sizes of each Command Buffer to submit to the DE. (Cannot be null)
- ccbGpuAddrs: A list of pointers to the addresses of the Command Buffers to submit to the CE. (Can be null)
- ccbSizesInBytes: A list of sizes of each Command Buffer to submit to the CE. (Can be null if ccbGpuAddrs is also null)

Description: Submits `length` amount of Command Buffers to the DE (Draw Engine), and optionally the CE (Constant Engine).

Returns: `0` if successful. Otherwise it returns some other value (Error codes not documented yet)

Exceptions: Unknown at this time.

6.1.2 sceGnmSubmitCommandBuffersForWorkload

```
int __stdcall sceGnmSubmitCommandBuffersForWorkload(
    uint32_t unused, uint32_t length,
    void* dcbGpuAddrs[], uint32_t dcbSizesInBytes[],
    void* ccbGpuAddrs[], uint32_t ccbSizesInBytes[])
```

Description: Same as `sceGnmSubmitCommandBuffers` but with an unused parameter. Perhaps was used for debugging on Devkit firmware? Complete speculation. `sceGnmSubmitCommandBuffers` actually just directly calls this function with the length parameter copied to the unused parameter.

Special Thanks

- Inori developer of GPCS4 for writing nice and readable code 😃
- The NSA for open-sourcing Ghidra, the reverse-engineering tool. And to all the developers who have now contributed to the project.
- AMD for open-sourcing some of their Radeon drivers which helped immensely with learning the hardware.
- The Mesa developers for also writing quite nice and readable code 😃
- The Wavedrom developers for the wonderful bitfield SVG renderer.