Radeon Southern Islands Acceleration

Trademarks

AMD, the AMD Arrow logo, Athlon, and combinations thereof, ATI, ATI logo, Radeon, and Crossfire are trademarks of Advanced Micro Devices, Inc.

Microsoft and Windows are registered trademarks of Microsoft Corporation.

Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Disclaimer

The contents of this document are provided in connection with Advanced Micro Devices, Inc. ("AMD") products. AMD makes no representations or warranties with respect to the accuracy or completeness of the contents of this publication and reserves the right to make changes to specifications and product descriptions at any time without notice. No license, whether express, implied, arising by estoppel, or otherwise, to any intellectual property rights are granted by this publication. Except as set forth in AMD's Standard Terms and Conditions of Sale, AMD assumes no liability whatsoever, and disclaims any express or implied warranty, relating to its products including, but not limited to, the implied warranty of merchantability, fitness for a particular purpose, or infringement of any intellectual property right. AMD's products are not designed, intended, authorized or warranted for use as components in systems intended for surgical implant into the body, or in other applications intended to support or sustain life, or in any other application in which the failure of AMD's product could create a situation where personal injury, death, or severe property or environmental damage may occur. AMD reserves the right to discontinue or make changes to its products at any time without notice.

1. Introduction

This guide is targeted at those who are familiar with GPU programming and the Radeon programming model. It is recommended that you read the r6xx/r7xx and evergreen/NI programming guides and ISA documents first as this guide builds on the information in those documents.

1.1 Major Hardware Changes

This section briefly describes the key SI hardware changes. Changes that have a major impact on the driver architecture will be described in more detail later.

- **Shader Core Next.** The hardware shader core has been completely redesigned. Highlights:
  - New, non-VLIW, clause-less instruction set architecture.
  - Distributed sequencer per compute unit.
  - Scalar ALU per compute unit.
  - Support for unlimited resources. Resource constants now read from memory.
  - Fetch shader stage removed.
  - Compute stage separated from LS.

- **JIT Constant Updates.** All resource descriptors (sometimes called "fetch constants" on previous hardware) are read from memory instead of registers. In order to improve performance, the CP block adds a new constant engine. The ***constant engine (CE)*** runs in parallel to the 3D engine, and allows constants to be written to memory ahead of the main command stream.

- **Unified Cache.** Most shader memory – vertex buffers, textures, constant buffers, UAVs, etc. – are read/written through a shared cache. Draw indices and the CB and DB blocks do not use the shared cache.

- **UAVs Written through Texture Pipe.** UAV writes are now done through the texture unit rather than the CB/DB RAT capabilities of prior hardware. Reads and writes both share the same cache, and the return buffer is no longer necessary.

- **Stateless Compute.** All render state for a given compute dispatch will be passed through the pipeline instead of reading context registers based on a state set ID. Therefore, compute dispatches do not use a hardware context as on previous hardware.

- **Color Export Packing Removed From SX.** On prior hardware, the PS would always export 4 32-bit values and the SX had fixed function hardware to pack the data into fewer bits depending on the RT format. On SI, this packing must be done by the PS itself.

- **Tile Index.** In order to reduce complexity, tiling parameters are now specified as an index into a 32-entry global tiling table initialized at boot.

- **Support for new OpenGL Extensions:**
  - **Partially Resident Textures.** Allows textures and their associated mipmaps to be accessed while only partially loaded into video memory.

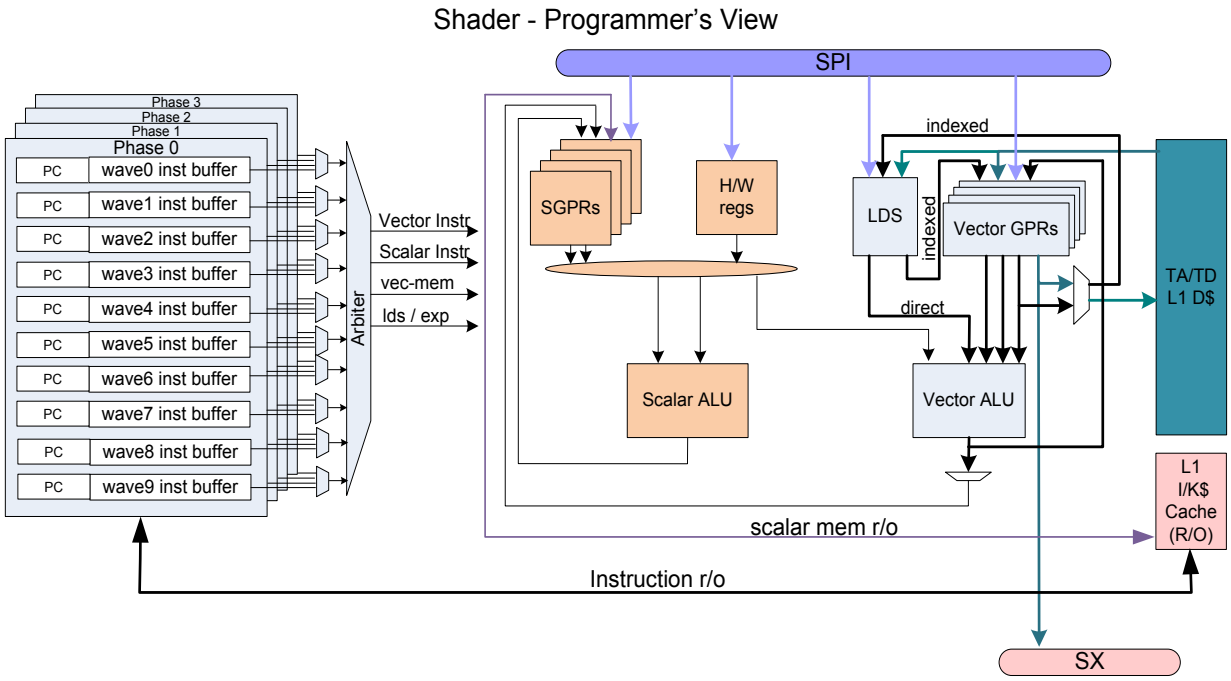
- **Depth Bounds Test.** The DB now supports functionality equivalent to the OpenGL EXT_depth_bounds_test extension.

1.2 Shader Core Next

The shader core has changed dramatically from NI to SI. This section highlights some critical changes.

1.2.1 Critical Concepts

The following sections detail key shader architecture changes that affect the 3D driver. The following diagram is included as a reference. It is a nice programmer's view of an SI compute unit (CU):



1.2.1.1 Distributed Sequencer (SQ)

NI has a global SQ per shader engine that executes a *control flow program* determining the execution of *instruction clauses* across all SIMDs. On SI, the SQ is distributed per-CU (represented on the left hand side of the above diagram), managing a 4-phase execution of up to 40 waves, where 1 instruction of each type can be scheduled per-cycle (i.e., one vector ALU op, one scalar op, one vector texture fetch, one LDS read/write, etc). The new ISA does not use instruction clauses, and control flow is managed by the Scalar ALU instead of a NI-style control flow program.

1.2.1.2 Vector ALU

On NI's Very Long Instruction Word (VLIW) architecture, a SIMD processes waves of 16 threads, with each thread executing 4 scalar ops simultaneously. It is up to the compiler to schedule 4 scalar ops that do not have dependencies on each other.

On SI, the vector ALU processes waves of 64 threads, with each thread executing only 1 scalar op. This architecture simplifies scheduling for the compiler and is more efficient in the common case where full instruction vectors cannot be scheduled.

The vector unit mostly works with a set of *Vector GPRs*. These are referred to as VGPRs, and show up as v0, v1, etc. in text shader code. It is important to recognize that the vector is 64-wide, with each item in the vector corresponding to one thread (one pixel in a pixel shader, one vertex in a vertex shader, etc). They are *not* 4-wide vectors corresponding to the rgba/xyzw components of an IL register, which must be implemented with 4 VGPRs.

1.2.1.3 Scalar ALU

The scalar ALU does work that does not vary per-thread in a wave, working with a separate set of *Scalar GPRs* (SGPRs). SGPRs show up as s0, s1, etc. in text shader code.

For example, the resource descriptors (fetch constants) stored in memory must be fetched before the texture fetch itself can be executed. Typically, the resource descriptor fetch would be done using a scalar op into SGPRs, while the texture fetch itself would vary per thread (due to texture coordinates varying per-pixel), and is therefore done with a vector texture fetch into VGPRs.

The Scalar ALU also manages control flow. For intra-wave branching, this can be done by manipulating the EXEC architectural SGPR, a 64-bit register that qualifies which threads are active for each vector instruction. The scalar unit also supports a variety of full-wave branch operations, typically based on the VCC architectural SGPR, a 64-bit register containing the 1-bit result per-thread of a previous vector instruction.

1.2.1.4 Fetch Constants in Video Memory

On NI, "fetch constants", describing the address, format, and other resource properties, are written to 8-state context registers. On SI, all such descriptors are typically stored in video memory, and the shader must explicitly fetch the resource descriptor before it can read or write the resource.

1.2.1.5 Shader Initialization

At shader launch, the SPI loads various GPRs from a variety of sources. Understanding the shader launch state is important for driver debugging:

- **VGPR Initialization:** VGPRs are loaded with shader inputs that vary per-thread. The initialization depends heavily on the shader type. For example, VSes load VGPR0 with the thread's vertex index, PSes loads VGPRs with barycentric coordinates, etc. The full details are explained in the Shader Programming section.

- **SGPR Initialization**: SGPRs are loaded with shader inputs that do not vary per-thread, and this process also depends on the shader type.

    - **USER_DATA.** Up to 16 SGPRs can be loaded with data from the SPI_SHADER_USER_DATA_xx_0-15 hardware registers written by the driver with PM4 SET commands. User data is used for communicating several details to the shader, most importantly the memory address of resource constants in video memory.

    - **Shader type specific data.** For example, CSes can load the thread group ID into SGPRs, GSes can load a GS/VS ring offset, etc.

The full SGPR loading details are described in the Shader Programming section.

- **Internal Regs:** Several architectural registers will also be loaded by SPI. For example, the program counter (PC) will be loaded with the driver-written shader address, and the EXEC mask will be loaded with the valid threads.

1.2.1.6 SH Registers

A new class of hardware register has been introduced for communicating with the shader core. NI has config (1-state) registers and context (8-state) registers. SI adds the concept of SH registers, which allow more than 8 sets of state at once, limited by the total number of registers updates in flight. Frequently updated shader registers are SH regs – user data, program bases, etc. There is a new PM4 packet for setting SH regs, SET_SH_REG, analogous to SET_CONTEXT_REG and SET_CONFIG_REG.

1.3 Constant Management

This section gives a high level overview of the constant management scheme for SI.

1.3.1 Shader Resource Descriptors (SRD)

On SI, all constant hardware registers have been removed, and instead normally live in memory. This includes fetch constants (SRVs, UAVs, vertex buffers, constant buffers), samplers, and descriptors for internal surfaces like shader rings.

The layout of individual constants in memory (buffer, image, or sampler) is described in the register spec entries for SQ_BUF_RSRC_WORD0-3, SQ_IMG_RSRC_WORD0-7, and SQ_IMG_SAMP_WORD0-3. These are dummy entries; they do not correspond to real registers.

1.3.2 Shader Launch State

1.3.2.1 User Data

For each shader type, there are 16 ***user data*** hardware registers (e.g., SPI_SHADER_USER_DATA_PS_0-15). These 16 32-bit values are the only interface through which the driver can specify SRDs to the shader:

- These registers are *written* with SET_SH_REG PM4 commands in the driver's indirect buffer.
- At shader launch, SPI loads values from these 16 hardware registers into SGPRs 0 – 15 to be *read* by the shader. The actual number of values loaded by SPI is controlled by the USER_SGPR field in the SPI_SHADER_PGM_RSRC2_xx register.

1.3.2.2 User Element Table

SC specifies what the driver should write in each of the user data hardware registers with a ***user element table*** returned along with the compiled hardware shader. For example, consider the following vertex shader code:

One entry in the user element table specifies mmSPI_SHADER_USER_DATA_VS_8-9 should be loaded with a 64-bit GPU memory pointer to a table of all constant buffer SRDs:

```
;  userElements[3] = PTR_CONST_BUFFER_TABLE,    0, s[8:9]
```

At shader launch, SPI will load the address written by the driver into SGPRs 8 and 9. In the shader itself, s[8:9] is dereferenced to lookup SRDs for each constant buffer.

This user data scheme is ultimately flexible for getting SRDs to the shader. We could specify SRD tables hierarchically, in efficient sparse structures, etc. In practice, though, we currently only support two modes:

- ***Immediate Mode.*** In immediate mode, a single SRD is written directly into user data hardware registers. This makes the SRDs available for use (in SGPRs) immediately, without having to read it from memory first. This mode is severely limited since there are so few user data registers and SRDs take up 4 to 8 dwords.

- ***Flat Table Mode.*** In flat table mode, the specified user data registers are programmed with an address to GPU memory containing a table of all SRDs of the specified type. E.g., PTR_RESOURCE_TABLE requires a pointer to a table with all SRV SRDs stored consecutively. The shader will have to explicitly load SRDs from the table before performing a fetch.

1.3.3 JIT CP Constant Updates

Updating SRDs using the CPU would be too slow. The CP provides a constant update engine to accelerate the process.

Draw Engine (DE): The standard graphics engine is now referred to as the Draw Engine. Most PM4 commands continue to be submitted to the DE command buffer.

Constant Engine (CE): The constant engine uses a second, separate command buffer to control constant uploads. The engine runs in parallel with the draw engine, allowing constant updates to get sufficiently ahead of the draws/dispatches that will use them.

Additionally, CP has 64KB of on-chip RAM (CE RAM) that acts as a staging buffer for constant updates. Shaders cannot read directly from the CE RAM.

The 64KB is carved up between the 3 rings, with ring-0 (gfx) having 32KB. The driver is responsible for further subdividing the partitions to store an on-chip copy of the most up-to-date copy of every SRD.

The CE engine supports 3 main operations:

- **Writes:** Write operations update a specified location in the CE RAM with inline data from the CE command buffer. As described above, these writes are used at resource bind to keep the CE RAM image up to date.
- **Dumps:** Dump operations copy from CE RAM to GPU memory. Dumps are performed during validation to update a newly allocated chunk with the most up to date SRD table.
- **Loads:** Load operations copy data from GPU memory to CE RAM.

The driver must explicitly manage synchronization between the CE and DE command buffers. To handle this, there are two counters:

- **CE Counter:** The CE counter can be incremented with a packet in the CE command buffer. *Before* each draw/dispatch, we insert a CE increment packet.
- **DE Counter:** The DE counter can be incremented with a packet in the DE command buffer. We issue a DE counter increment *after* every draw/dispatch.

Finally, CP supports a PM4 command allowing the DE engine to wait for (CE – DE > 0). We issue this packet before each draw/dispatch, ensuring that the updated constants are ready in memory before the draw is executed.

1.4 Fetch Shader Changes

Fetch shader is no longer available as a first-class shader stage as it is on the 6xx through NI families. On SI, the vertex fetch must be performed as part of the API VS (hardware VS, ES, or LS).

1.4.1 Implementation Options

The following options are possible for supporting vertex fetching on SI:

- **Fetch Shader Subroutine (FS):** Implement a subroutine called by the VS to load all input VGPRs, based on input layout.
- **Monolithic VS with Embedded Vertex Fetch (MVS):** Compile vertex fetch as part of the VS. A new hardware VS would need to be compiled for every associated input layout.
- **Fetch Operation Per Vertex Element (FOPE):** Implement a set of small subroutines that each fetch a single input element, VS would call subroutines to fetch input elements as necessary.

1.4.2 FS Subroutine

The fetch shader subroutine attempts to emulate the fetch shader functionality on NI. As on NI, the driver allocates video memory for the FS and generates hardware shader instructions directly mapping the vertex layout state to the shader inputs. Key changes from NI:

- **FS is a true VS subroutine.**
  - No SQ_PGM_START_FS register. Compiler specifies a user data register to be programmed with the FS address.
  - No SQ_PGM_RESOURCES_FS register. Driver may need to update hardware VS registers based on which FS is bound (account for FS VGPR/SGPR usage, etc).
  - FS is responsible for explicitly returning to the VS.
- **No semantic fetch support.** Without semantic fetch, the compiler returns a map from logical input registers to physical GPRs. A unique FS is generated for every combination of vertex layout state and hardware VS.
- **No fixed function BaseVertexLocation or StartInstanceLocation support.** NI exposed SQ_VTX_BASE_VTX_LOC and SQ_VTX_START_INST_LOC CTL constant registers for these parameters which modify the vertex index and instance index, respectively, before invoking the FS. We must now specify these parameters through user data registers and compute the offset values in the FS.

1.5 VS/PS Semantic Mapping

NI and prior hardware support VS to PS semantic mapping, where the VS outputs are "named" with semantic values, PS inputs are "named" with similar semantic values, and the hardware automatically manages mapping everything to the right place in the parameter cache.

On SI, this responsibility falls on the driver, which must program the SPI_PS_INPUT_CNTL_0-31 registers with the absolute parameter cache location of each PS input. This must be handled at validation time by examining the VS output declarations and PS input declarations as returned by the compiler.

1.6 Color Export Packing

On pre-SI hardware, the pixel shader always exports 4-component 32-bit vectors per render target. Based on the format of the destination render target view, the SX has fixed function hardware to reduce precision of the exported

values as needed. This feature was controlled by the SOURCE_FORMAT field in the CB_COLORn_INFO register.

On SI, the SX hardware block no longer provides this fixed function support. Instead, the PS must issue shader instructions to reduce precision before executing export instructions. This requires slightly different versions of each PS based on the formats of the bound render targets.

At draw time, the driver will examine the bound RTs, and determine which format should be used for each export. There are 10 possible export formats, selected per-RT in the SPI_SHADER_COL_FORMAT register (e.g., 32_ABGR, UNORM16_ABGR, 32_R, etc). Based on the format chosen for each RT, the driver will potentially create a new version of the PS, patching all export sequences with the appropriate export format.

1.7 Tile Index

SI removes the CB, DB, and SRD fields allowing the explicit programming of tiling parameters (tile mode, num banks, macro tile aspect ratio, etc). Instead, these parameters are specified with a *tile index* that will select a pre-determined set of tiling parameters from a global, constant "menu" of configurations. This "menu" is specified in the GB_TILE_MODE0-31 registers.

1.7.1 Gotchas

There are some gotchas with the tile index scheme worth noting:

- CB and SRDs allow a 5-bit index, allowing the choice of any tile config in the table. DB only has a 3-bit index, so the Z/stencil tile configs have to be in the first 8 entries.
- There is no LINEAR_GENERAL entry. Linear general is implied for all buffer SRVs. When rendering to a buffer RT, an override bit, CB_COLORn_INFO.LINEAR_GENERAL, must be set telling the CB to ignore the programmed tile index.
- Z/stencil buffers have two tile indices:
    - DB_Z_INFO.TILE_MODE_INDEX defines all tile parameters for the depth plane, and *most* parameters for the stencil plane.
    - DB_STENCIL_INFO.TILE_MODE_INDEX specifies an entry from which the stencil's TILE_SPLIT parameter will be read from (others are shared with the depth plane).
- MSAA color surfaces also have two tile indices. CB_COLORn_ATTRIB.FMASK_TILE_MODE_INDEX specifies an entry from which the fmask's tile parameters will be read.

2. PM4

2.1 Overview

PM4 is the packet API used to program the GPU to perform a variety of tasks. The driver does not write directly to the GPU registers to carry out drawing operations on the screen. Instead, it prepares data in the format of PM4 Command Packets in either system or video (a.k.a. local) memory, and lets the Micro Engine to do the rest of the job.

Three types of PM4 command packets are currently defined. They are types 0, 2 and 3 as shown in the following figure. A PM4 command packet consists of a packet header, identified by field HEADER, and an information body, identified by IT_BODY, that follows the header. The packet header defines the operations to be carried out by the PM4 micro-engine, and the information body contains the data to be used by the engine in carrying out the operation. In the following, we use brackets [.] to denote a 32-bit field (referred to as DWord) in a packet, and braces {.} to denote a size-varying field that may consist of a number of DWords. If a DWord consists of more than one field, the fields are separated by "|". The field that appears on the far left takes the most significant bits, and the field that appears on the far right takes the least significant bits. For example, DWord LO_WORD denotes that HI_WORD is defined on bits 16-31, and LO_WORD on bits 0-15. A C-style notation of referencing an element of a structure is used to refer to a sub-field of a main field. For example, MAIN_FIELD.SUBFIELD refers to the sub-field SUBFIELD of MAIN_FIELD.

2.1.1 Type-0 Packets

Type-0 Packets are discouraged, but can be used if absolutely required. Type-3 packets should be used instead. Write N DWords in the information body to the N consecutive registers, or to the register, pointed to by the BASE_INDEX field of the packet header. Does check for context roll. This packet supports a register memory map up to 64K DWords (256K Bytes).

Type-0 Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet [15:0] BASE_INDEX; DWord Offset into the 64KB register address space. [29:16] COUNT; Count of DWords in the information body. Its value should be N-1 if there are N DWords in the information body. [31:30] TYPE; Packet identifier. It should be zero.
2	REG_DATA_x	[31:0] REG_DATA_x; The bits correspond to those defined for the relevant register. Note the suffix x of REG_DATA_x stands for an integer ranging from 1 to N.

The use of this packet requires the complete understanding of the registers to be written. The register address is split into two areas: the first 32K bytes is system registers and beyond that is graphics and multi-media. For graphics and multi-media registers there is an alternative, called SET_*. For the first 32KB of register space (system registers) there is no SET_* type packet and TYPE-0 packets should be used.

2.1.2 Type-1 Packets

Type-1 packets are not supported.

2.1.3 Type-2 Packets

This is a filler packet. It has only the header, and its content is not important except for bits 30 and 31. It is used to fill up the trailing space left when the allocated buffer for a packet, or packets, is not fully filled. This allows the CP to skip the trailing space and to fetch the next packet.

Type-2 Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet [29:0] Reserved; This field is undefined, and is set to zero by default. [31:30] TYPE; Packet identifier. It should be 2.

2.1.4 Type-3 Packets

Carry out the operation indicated by field IT_OPCODE.

Type-3 Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet [0] PREDICATE; Predicated version of packet when bit 0 is set. [1] SHADER_TYPE; (0: Graphics, 1: Compute Shader). Unless otherwise noted, Shader_Type should be programmed to zero. [7:2] Reserved; This field is undefined, and is set to zero by default. [15:8] IT_OPCODE; Operation to be carried out. [29:16] COUNT; Number of DWords -1 in the information body. It is N-1 if the information body contains N DWords. [31:30] TYPE; Packet identifier. It should be 3.
2,3,..	IT_BODY	[31:0] Body; The information body "IT_BODY" will be described extensively in the following sections

Type-3 packets have a common format for their headers. However, the size of their information body may vary depending on the value of field IT_OPCODE. The size of the information body is indicated by field COUNT. If the size of the information is N DWords, the value of COUNT is N-1. In the following packet definitions, we will describe the field IT_BODY for each packet with respect to a given IT_OPCODE, and omit the header. .

2.1.5 Packet Restrictions

Packets that must be immediately after a Draw_*

- EVENT_WRITE_EOS

2.1.6 Ring Packet Support

There are two CP engines on SI: the Constant Engine (CE) and the Drawing Engine (DE). Previous asics only had a DE (previously refered to as the Micro-Engine or ME). Certain packets are only allowed on the DE or the CE. Packets can be executed in the DE via the ring or via an INDIRECT_BUFFER packet. Packets can only be executed on the CE by using the INDIRECT_BUFFER_CONST packet.

Packet Name	Ring	DE	CE
Initialization Packets			

ME_INITIALIZE	0	X	X
PREAMBLE_CNTL	0-2	X	
Command Buffer Packets			
INDIRECT_BUFFER	0-2	X	
INDIRECT_BUFFER_CONST	0-2		X
Draw/Dispatch Packets			
DRAW_INDEX	0	X	
DRAW_INDEX_2	0	X	
DRAW_INDEX_AUTO	0	X	
DRAW_INDEX_MULTI_AUTO	0	X	
DRAW_INDEX_IMMED	0	X	
DRAW_INDEX_INDIRECT	0	X	
INDEX_BUFFER_SIZE	0	X	
DRAW_INDEX_OFFSET	0	X	
DRAW_INDEX_OFFSET_2	0	X	
DRAW_INDIRECT	0	X	
INDEX_BASE	0	X	
INDEX_TYPE	0	X	
NUM_INSTANCES	0	X	
MPEG_INDEX	0	X	
DISPATCH_DIRECT	0-2	X	
DISPATCH_INDIRECT	0-2	X	
State Management Packets			
CLEAR_STATE	0-2	X	
CONTEXT_CONTROL	0-2	X	
LOAD_CONFIG_REG	0	X	
LOAD_CONTEXT_REG	0-2	X	
LOAD_SH_REG	0-2	X	
ALLOC_GDS	0-2	X	
SET_BASE	0-2	X	X
SET_CONFIG_REG	0	X	
SET_CONTEXT_REG	0-2	X	
SET_CONTEXT_REG_INDIRECT	0-2	X	
SET_SH_REG	0-2	X	
LOAD_CONST_RAM	0-2		X
WRITE_CONST_RAM	0-2		X
WRITE_CONST_RAM_OFFSET	0-2		X
DUMP_CONST_RAM	0-2		X
SET_CE_DE_COUNTERS	0-2		X
INCR_DE_COUNTER	0-2	X	
INCR_CE_COUNTER	0-2		X
WAIT_ON_DE_COUNTER	0-2		X
WAIT_ON_CE_COUNTER	0-2	X	
Command Predication Packets			
COND_EXEC	0-2	X	

COND_WRITE	0-2	X	
SET_PREDICATION	0-2	X	
PRED_EXEC	0-2	X	
OCCLUSION_QUERY	0-2	X	
Synchronization Packets			
EVENT_WRITE	0-2	X	
EVENT_WRITE_EOP	0-2	X	
EVENT_WRITE_EOS	0-2	X	
MEM_SEMAPHORE	0-2	X	
PFP_SYNC_ME	0-2	X	
STRMOUT_BUFFER_UPDATE	0-2	X	
SURFACE_SYNC	0-2	X	
WAIT_REG_MEM	0-2	X	
Atomic			
ATOMIC	0-2	X	
ATOMIC_GDS	0-2	X	
Misc Packets			
COPY_DW	0-2	X	
COPY_DATA	0-2	X	
ME_WRITE	0-2	X	
CE_WRITE	0-2		X
MEM_WRITE	0-2	X	
NOP	0-2	X	X
ONE_REG_WRITE	0-2	X	

2.2 Configuration Packets

2.2.1 ME_INITIALIZE

The usage rules for the ME_INITIALIZE packet are:

- The ME_INITIALIZE packet should be sent to the CP immediately after loading the microcode and enabling the Micro Engine (ME).
- This Type-3 packet is used by the ME to initialize internal state information that is used by other packets.
- If the ME_INITIALIZE packet changes the MAX_CONTEXT value, then it needs to be followed by a CONTEXT_CONTROL packet with a full load mask to force a reload of shadowed registers and constants.
- If the device supports more than one ring buffer for a single GPU, only the primary ring (3D ring) should have this packet.

ME_INITIALIZE Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet.
2	Default Reset Control	[0] - Default Reset Control; Resets specific areas of the Scratch memory to known values and Resets the Current and Last contexts
3	Reserved	[31:0] Reserved; Program to zero.
4	MAX_CONTEXT	[2:0] MAX_CONTEXT; Maximum Context in Chip. Values are 1 to 7.

		Max context of 0 is not valid since that context is now used for the clear state context. For example, 3 means the GPU uses contexts 0-3, i.e., it utilizes 4 contexts.
5	DEV_ID | EXTERNAL_MEM_SWAP	31:24 - Reserved 23:16 - One-hot Device-ID 15:2 - Reserved 1:0 - Swap Code Used for the following transactions: Load_* , Set_* , PM4 headers - debug
6	Header_Dump_Base | Header_Dump_Swap	31:4 - Header_Dump_Base : a 4 Kbyte aligned address, i.e. base memory address [47:12] of the external memory location where CP will dump PM4 Headers. 3:2 - Reserved: should be set to zero. 1:0 - Header_Dump_Swap: the 2 bit Swap Code used when writing headers to memory.
7	Header_Dump_Enable | Header_Dump_Size	31 - Header_Dump_Enable: Enable Writing PM4 Headers to Memory for Debug (Degrades Performance). 30 - Reserved. 29:0 - Header_Dump_Size: Size in DWords for the Header Dump Ring in External Memory.

2.2.2 SET_CONFIG_REG

The SET_CONFIG_REG packet loads the single-context-configuration register data, which is embedded in the packet, into the chip. The REG_OFFSET field is a DWord-offset from the starting address. All the register data in the packet is written to consecutive register addresses beginning at the starting address. The starting address for register data is computed as follows:

- Reg_Start_Address[17:2] = 0x2000 + REG_OFFSET (Note: Byte Offset 0x8000; DWord Offset 0x2000)

The CP will write the data to external memory if the corresponding shadow enable is set. This allows the register data to be reloaded into the chip later with the LOAD_CONFIG_REG packet. The LOAD_CONFIG_REG packet sets the REG_CONFIG_BASE and the CONTEXT_CONTROL packet enables/disables write shadowing to external memory (see these packets for more details). The starting external memory address that the register data is written to is computed as follows:

- Mem_Start_Address[47:2] = CONFIG_REG_ BASE[47:2] + REG_OFFSET

SET_CONFIG_REG Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet.
2	REG_OFFSET	[31:16] - Must be programmed to zero for legacy support [15:0] - Offset in DWords from the register base address (0x2000 in DWs) and memory base address (CONFIG_REG_BASE).
3 to N	REG_DATA	DWord Data for Registers.

2.2.3 SET_BASE

The SET_BASE packet is used to generically specify the starting, or base, memory address of a buffer use by the CP for unrelated features.

BASE_INDEX detail:

- 0010 (GDS_Partition_Bases), the packet sets the partition boundaries for each section in the GDS. There are 3 sections: Ring 0 Gfx/CS0, CS1 & CS2. GDS_RING0_INDEX is always 0, but the other boundaries are programmable as indicated in the table below.
- 0011 (CE_Partition_Bases), the packet sets the partition boundaries for each section in the CE. There are 3 sections: Ring 0 Gfx/CS0, CS1 & CS2. CE_RING0_INDEX is always 0, but the other boundaries are programmable as indicated in the table below.

SET_BASE Packet Description

DW	Field	Description
1	SET_BASE	Header with Shader_Type in bit[1]
2	BASE_INDEX	[3:0] BASE_INDEX – specifies which base address is specified in the following DWs. 0000 - Display List Patch Table Base. 0001 - DrawIndexIndir Patch Table Base. 0010 - GDS_Partition_Bases. 0011 - CE_Partition_Bases.
3	ADDRESS0	If BASE_INDEX 0000 or 0001 then [31:3] ADDRESS_LO [2:0] Reserved – zero. Else If BASE_INDEX 001X then [31:16] Reserved – zero. [15:0] CS1_INDEX – applies to BASE_INDEX 001X. Specifies the beginning of the CS1 partition as an index in bytes from '0'. Bits[5:0] should be zero since only mod 64byte boundaries are supported to match the TC L2 interface granularity. Else illegal programming.
4	ADDRESS1	[31:16] Reserved – zero. If BASE_INDEX 0000 or 0001then [15:0] ADDRESS_HI Else If BASE_INDEX 001X then [15:0] CS2_INDEX. Specifies the beginning of the CS2 partition as an index in bytes from '0'. Bits[5:0] should be zero since only mod 64byte boundaries are supported to match the TC L2 interface granularity. Else illegal programming.

2.2.4 LOAD_CONFIG_REG

The LOAD_CONFIG_REG packet will only be processed if the CP processes a CONTEXT_CONTROL packet with the appropriate shadow bit set.

This packet provides the ability to have the CP:

- Initialize the CONFIG_REG_BASE (BASE_ADDR_* fields) internally for later use when a SET_CONFIG_REG packet is processed and shadowing is enabled (via the CONTEXT_CONTROL packet). For this case, there are 5 DWs in the packet and DWs 4 and 5, REG_OFFSET and NUM_DWords are programmed to zero.
- Fetch single-context-configuration register data from external memory into the chip that was previously shadowed. For this case, there are 5 or more DWs and all are meaningful.

The CP computes the DWord-aligned external memory read address as follows:

- Mem_Start_Address[47:2] = CONFIG_REG_BASE[47:2] + REG_OFFSET

The CP writes the loaded data to consecutive register addresses. The starting address is computed as shown below:

- Reg_Start_Address[17:2] = 0x2000 + REG_OFFSET (Note: Byte Offset 0x8000 = DWord Offset 0x2000)

To preserve coherency between shadowed Set_* packet writes and Load_* packet reads from external memory, the CP first waits until all prior SET_* data has been shadowed to memory before issuing the Load's memory read requests for the register data. The CP will fetch Num_DWords from external memory. If, however, the driver only needs to change the Base_Addr, then Num_DWords can be set to zero and no data will be fetched.

LOAD_CONFIG_REG Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet.
2	BASE_ADDR_LO	[31:2] - CONFIG_REG_BASE (31:2) for the block in Memory from where the CP will fetch the state.
3	WAIT_FOR_IDLE| BASE_ADDR_HI	[31] - If set the CP will wait for the graphics pipe to be idle by writing to the GRBM Wait Until register with Wait for 3D idle". [15:0] - CONFIG_REG_BASE (47:32) for the block in Memory from where the CP will fetch the state.
4	REG_OFFSET	[15:0] - REG_OFFSET[15:0] in DWords from the register base address (Fixed at 0x2000 in DWs) and memory base address (CONFIG_REG_BASE).
5	NUM_DWords	NUM_DWords[13:0] - Number of DWords that the CP will fetch and write into the chip. A value of zero will cause no constants to be loaded.
N	REG_OFFSET	31:16- Reserved REG_OFFSET[15:0] - Same Definition as Above.
N+1	NUM_DWords	31:12 - Reserved NUM_DWords[13:0] - Same Definition as Above.

2.3 Constant Engine Packets

2.3.1 DUMP_CONST_RAM

Sent only to the Constant Engine to instruct the CE to move data from one or more RAM slot lines to L2 Memory.

DUMP_CONST_RAM Packet Description

DW	Field	Description
1	HEADER	Header of the packet.
2	OFFSET	[15:0] Starting byte offset into the Constant RAM. The minimum granularity is 4 bytes, so bits[1:0] must be zero.
3	NUM_DW	[14:0] Number of DWs to read from the constant RAM. The minimum granularity is DWs, so any bit can be '1'.
4	ADDR_LO	[31:0] Byte Address[31:0]. The Address granularity is 4 bytes, so bits[1:0] must be zero. bits[1:0] are zero.
5	ADDR_HI	[31:0] Address[63:32].

2.3.2 INCREMENT_CE_COUNTER

In the ME this packet creates a pipelined event that causes the CP to EOP block to increment it counter. If the packet command specifies to clear the counter, the ME does this at the top of pipe and clears the associated counter. The counter is double buffered.

INCREMENT_CE_COUNTER Packet Description

DW	Field	Description
1	HEADER	Header of the packet.
2	DUMMY	[31:0] Dummy Data.

2.3.3 INDIRECT_BUFFER_CONST

This packet is used for dispatching Constant Indirect Buffers, new in SI. A separate constant buffer allows the CP to process constants ahead of and concurrently with the "draw command buffer". The driver effectively creates two command buffers where it created one previous to SI. The command buffer pointed to by this packet has new packets specifying the constants.

The KMD will specify the VMID in each Indirect_Buffer_Const packet. The KMD is required to include the VMID in the Indirect_Buffer_Const packet in the ring buffer.

INDIRECT_BUFFER_CONST Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	IB_BASE_LO	[31:2] - Indirect Buffer Base Address[31:2] - DW-Aligned [1:0] - Swap function used for data write.
3	IB_BASE_HI	[15:0] - Upper bits of Address [47:32]
4	VMID |	[31:24] - VMID[7:0], Virtual Memory Domain ID for the command buffer.
	IB_SIZE	[19:0] - Indirect Buffer Size [19:0], size of the Indirect Buffer in DWORDs.

2.3.4 LOAD_CONST_RAM

Can be conditionally executed to initialize or re-prime the CE's constant RAM from memory at the beginning of an app or at a switch from one application to another.

LOAD_CONST_RAM Packet Description

DW	Field	Description
1	HEADER	Header of the packet.
2	ADDR_LO	[31:0] Byte Address[31:0]. The Address granularity is 32 bytes, so bits[4:0] must be zero.
3	ADDR_HI	[31:0] Address[63:32].
	NUM_DW	[14:0] Number of DW to read from "memory". The minimum granularity is 32 bytes (8 DWs), so bits[2:0] must be zero.
	OFFSET	[15:0] Starting byte offset into the constant RAM. The minimum granularity is 32 bytes so bits[4:0] must be zero.

2.3.5 SET_CE_DE_COUNTERS

This packet initializes the ME called DE_COUNT and the CE version called CE_COMPARE_COUNT. The WAIT_ON_DE_COUNTER packet only compares to the DE_COUNT, where the WAIT_ON_DE_-COUNTER_DIFF subtracts the DE_COUNT from the CE_COMPARE_COUNT. It does not clear the CE's up/down counter called CE_COUNTER since that will always be zero already by design.

SET_CE_DE_COUNTERS Packet Description

DW	Field	Description
1	HEADER	Header of the packet.
2	COUNTER_LO	[31:0] Lower 32 bits of the counter.
3	COUNTER_HI	[31:0] Upper 32 bits of the counter.

2.3.6 WAIT_ON_DE_COUNTER

Instructs the CE to wait on counter from the DE to be greater than or equal to COUNTER_HI:COUNTER_LO.

WAIT_ON_DE_COUNTER Packet Description

DW	Field	Description
1	HEADER	Header of the packet.
2	COUNTER_LO	[31:0] Lower 32 bits of the counter.
3	COUNTER_HI	[31:0] Upper 32 bits of the counter.

2.3.7 WAIT_ON_DE_COUNTER_DIFF

Instructs the CE to wait on difference between the ME's copy of the DE counter (DE_COUNT) and the CE's copy (CE_COMPARE_COUNT) to be less than the DIFF.

WAIT_ON_DE_COUNTER_DIFF Packet Description

DW	Field	Description
1	HEADER	Header of the packet.

2	DIFF	Difference between CE_COMPARE_COUNT and DE_COUNT must be less than DIFF to continue.

2.3.8 WRITE_CONST_RAM

Write DWs from the PM4 stream to the CE's constant RAM.

WRITE_CONST_RAM Packet Description

DW	Field	Description
1	HEADER	Header
2	OFFSET	[15:0] Starting DW granularity offset into the constant RAM. Thus, bits[1:0] are zero.
3	DATA	[31:0] DATA – Data to write to the Constant RAM.

2.3.9 WRITE_CONST_RAM_OFFSET

Packet is similar to the WRITE_CONST_RAM packet, except that the DATA ordinal needs to be modified. The data supplied is an offset from the partition base, but the partition base is unknown. The microcode needs to replace the DATA with 'DATA + Partition_Base' before writing it into the CE RAM. The corresponding partition is determined from the Ring to which the packet is submitted (see GDS_*_INDEX as described in SET_BASE packet for more details).

WRITE_CONST_RAM_OFFSET Packet Description

DW	Field	Description
1	HEADER	Header
2	OFFSET	[15:0] OFFSET – unchanged from WRITE_CONST_RAM packet
3	DATA	[31:0] DATA – Data is replaced with DATA + Partition_Base.

2.4 State Management Packets

2.4.1 CONTEXT_CONTROL

For SI, Samplers, Resources, ALU constants are not shadowed since they are written to memory as “buffers” for the SQ to fetch. The CONTEXT_CONTROL packet controls the processing of LOAD packets and shadowing for SET packets. It is valid for each ring, but not all fields apply to rings 1 and 2. Bit zero: of two ordinals is shader-type-less, meaning the shader type bit in the SET_CONFIG_REG packet is ignored, but should be programmed to zero. The Load Control bits enable/disable the CP's processing of each type of LOAD packet. When the Load Control bits are set, the CP will process the LOAD packets indicated. Likewise, if the bits are cleared, the CP will discard the corresponding LOAD packets indicated. There is a bit for enabling/disabling each Load variation.

The Shadow Enable bits are used to turn shadowing on and off for the SET_* packets. When set, the memory-mapped register writes will be shadow to memory and when reset, they will not. There is a bit for enabling/disabling shadowing for each type of SET packet (review the SET packets for more details on shadowing). The Load and Shadow DWs each have a DW Enable bit. When not set, the DW will be discarded so that when needed, the driver may update one without affecting the other.

CONTEXT_CONTROL Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	LOAD_CONTROL	Bit 31 - DW Enable - Load Enables will not be updated unless this bit is set Bit 30:25 - Reserved- must be zero Bit 24 - Enable Load of CS SH Registers (SI) Bit 23:17 - Reserved- must be zero Bit 16 - Enable Load of Gfx SH Registers (SI) Bits 15:2 - Reserved- must be zero Bit 1 - Enable Load Multi-Context Render State Registers Bit 0 - Enable Load Single-Context Configuration Registers, both Gfx and CS
3	SHADOW_ENABLE	Bit 31 - DW Enable - Shadow Enables will not be updated unless this bit is set. Bit 30:25 - Reserved- must be zero Bit 24 - Enable Shadowing of CS SH Registers (SI) Bit 23:17 - Reserved- must be zero Bit 16 - Enable Shadowing of Gfx SH Registers (SI) Bits 15:2 - Reserved- must be zero Bit 1 - Enable Shadowing of Multi-Context Render State Registers Bit 0 - Enable Shadowing of Single-Context Configuration Registers, both Gfx and CS

2.4.2 CLEAR_STATE

The purpose of the Clear_State packet is to reduce command buffer preamble setup time for all driver versions of both DX and OpenGL and to specifically support DX11's Display Lists requirements. The definition of Clear State is essentially everything off, resources all NULL, other values set to a defined default state.

CLEAR_STATE Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet.
2	DUMMY	Dummy Data

2.4.3 LOAD_CONTEXT_REG

This packet provides the ability to have the CP:

- Initialize the CONTEXT_REG_BASE (BASE_ADDR_* fields) internally for later use when a SET_CONTEXT_REG packet is processed and shadowing is enabled (via the CONTEXT_CONTROL packet). For this case, there are 5 DWs in the packet and DWs 4 and 5, REG_OFFSET and NUM_DWords are programmed to zero.
- Fetch eight-context-configuration register data from external memory into the chip that was previously

shadowed. For this case, there are 5 or more DWs and all are meaningful.
The CP computes the DWord-aligned external memory read address as follows:

- Mem_Start_Address[47:2] = CONTEXT_REG_BASE[47:2] + REG_OFFSET

The CP writes the loaded data to consecutive register addresses. The starting address is computed as shown below:

- Reg_Start_Address[17:2] = 0xA000 + REG_OFFSET (Note: Byte Offset 0x28000 = DWord Offset 0xA000)

To preserve coherency between shadowed Set packet writes and Load packet reads from external memory, the CP first waits until all prior SET_* data has been shadowed to memory before issuing the Load"s memory read requests for the register data. The CP will fetch Num_DWords from external memory. If, however, the driver only needs to change the Base_Addr, then Num_DWords can be set to zero and no data will be fetched.

LOAD_CONTEXT_REG Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet.
2	BASE_ADDR_LO	[31:2] - CONTEXT_REG_BASE (31:2) for the block in Memory from where the CP will fetch the state.
3	BASE_ADDR_HI	[15:0] - CONTEXT_REG_BASE (47:32) for the block in Memory from where the CP will fetch the state.
4	REG_OFFSET	[15:0] - REG_OFFSET[15:0] in DWords from the register base address (Fixed at 0xA000 in DWs) and memory base address (CONTEXT _REG_BASE)..
5	NUM_DWords	NUM_DWords[13:0] - Number of DWords that the CP will fetch and write into the chip. A value of zero will cause no constants to be loaded.
N	REG_OFFSET	31:16- Reserved REG_OFFSET[15:0] - Same Definition as Above.
N+1	NUM_DWords	31:12 - Reserved NUM_DWords[13:0] - Same Definition as Above.

2.4.4 SET_CONTEXT_REG

This packet loads the eight-context-renderstate register data, which is embedded in the packet, into the chip. Note: This packet checks if the context needs to be updated and rolls the context as required. The REG_OFFSET field is a DWord-offset from the starting address. All the render state data in the packet is written to consecutive register addresses beginning at the starting address. The starting address for register data is computed as follows:

- Reg_Start_Address[17:2] = 0xA000 + REG_OFFSET (Note: Byte Offset 0x28000; DWord Offset 0xA000)

The CP will write the data to external memory if the corresponding shadow enable is set. This allows the register data to be reloaded into the chip after a context switch with the LOAD_CONTEXT_REG (LCTX) packet. The LCTX packet sets the REG_CONTEXT_BASE and the CONTEXT_CONTROL packet enables/disables write shadowing to external memory (see these packets for more details). The starting external memory address that the render state data is written to is computed as follows:

- Mem_Start_Address[39:2] = CONTEXT_REG_BASE[39:2] + REG_OFFSET

To preserve coherency between shadowed Set packet writes and Load packet reads from external memory, the CP.PFP first waits until all prior SET_* data has been shadowed to memory before issuing the Load's memory read requests for the register data.

SET_CONTEXT_REG Packet Description

DW	Field Name	Description

1	HEADER	Header of the packet. Shader_Type in bit 1 of the Header will correspond to the shader type of the Load, see Type-3 Packet.
2	REG_OFFSET	[15:0] - Offset in DWords from the register base address (0xA000 in DWs) and memory base address (CONTEXT_REG_BASE).
3 to N	REG_DATA	DWord Data for Registers or DW Offset into the Patch Table.

2.4.5 SET_CONTEXT_REG_INDIRECT

This packet loads the eight-context-renderstate register data, which the CP fetches from the Patch Table, starting at offset REG_INDEX. The REG_OFFSET field is a DWord-offset from the starting address. Note: This packet checks if the context needs to be updated and rolls the context as required. The REG_OFFSET field is a DWord-offset from the starting address. The write address for the register data is computed as follows:

- Reg_Start_Address[17:2] = 0xA000 + REG_OFFSET (Note: Byte Offset 0x28000; DWord Offset 0xA000)

The CP will write the data to external memory if the corresponding shadow enable is set. This allows the register data to be reloaded into the chip later with the LOAD_CONTEXT_REG packet. The LOAD_CONTEXT_REG packet sets the REG_CONTEXT_BASE and the CONTEXT_CONTROL packet enables/disables write shadowing to external memory (see these packets for more details). The starting external memory address that the render state data is written to is computed as follows:

- Mem_Start_Address[39:2] = CONTEXT_REG_BASE[39:2] + REG_OFFSET

To preserve coherency between shadowed Set packet writes and Load packet reads from external memory, the CP.PFP first waits until all prior SET_* data has been shadowed to memory before issuing the Load's memory read requests for the register data.

SET_CONSTEXT_REG_INDIRECT Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet. Shader_Type in bit 1 of the Header will correspond to the shader type of the Load, see Type-3 Packet.
2	REG_OFFSET	[15:0] - Offset in DWords from the register base address (0xA000 in DWs) and memory base address (CONTEXT_REG_BASE).
3	REG_INDEX	DW Offset into the Patch Table.

2.4.6 LOAD_SH_REG

This packet provides the ability to have the CP:

- Initialize the per-ring SH_REG_BASE (BASE_ADDR_* fields) internally for later use when a SET_SH_REG packet is processed and shadowing is enabled (via the CONTEXT_CONTROL packet). For this case, there are 5 DWs in the packet and DWs 4 and 5, REG_OFFSET and NUM_DWords are programmed to zero.
- Fetch SH REG data from external memory into the chip that was previously shadowed. For this case, there are 5 or more DWs and all are meaningful.

The CP computes the DWord-aligned external memory read address as follows:

- Mem_Start_Address[47:2] = SH_REG_BASE[47:2] + REG_OFFSET

The CP writes the "loaded" data to consecutive register addresses. The starting address is computed as shown below:

- Reg_Start_Address[17:2] = 0X2C00 + SH_REG_OFFSET

To preserve coherency between shadowed Set packet writes and Load packet reads from external memory, the CP first waits until all prior SET_* data has been shadowed to memory before issuing the Load's memory read requests for the register data. The CP will fetch Num_DWords from external memory. If, however, the driver only needs to change the Base_Addr, then Num_DWords can be set to zero and no data will be fetched.

LOAD_SH_REG Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet. Shader_Type in bit 1 of the Header will correspond to the shader type of the Load, see Type-3 Packet.
2	BASE_ADDR_LO	[31:2] - SH_REG_BASE (31:2) for the block in Memory from where the CP will fetch the constants.
3	BASE_ADDR_HI	[15:0] - SH_REG_BASE (47:32) for the block in Memory from where the CP will fetch the constants.
4	REG_OFFSET	[15:0] - REG_OFFSET[15:0] in DWords from the base SH register address (Fixed at 0x2C00) and memory base address (SH_REG_BASE).
5	NUM_DWords	NUM_DWords[13:0] - Number of DWords that the CP will fetch and write into the chip. A value of zero will cause no constants to be loaded.
N	REG_OFFSET	31:16 - Reserved REG_OFFSET[15:0] - Same Definition as Above.
N+1	NUM_DWords	31:12 - Reserved NUM_DWords[13:0] - Same Definition as Above.

2.4.7 SET_SH_REG

This packet updates the shader persistent register state in the SPI, which is embedded in the packet, into the chip. The REG_OFFSET field is a DWord-offset from the starting address. All the persistent state data in the packet is written to consecutive register addresses beginning at the starting address. The starting address for register data is computed as follows:

- Reg_Start_Address[17:2] = 0x2C00 + REG_OFFSET (Note: Byte Offset 0xB000; DWord Offset 0x2C00)

The CP will write the data to external memory if the corresponding shadow enable is set. This allows the register data to be reloaded into the chip later with the LOAD_SH_REG packet. The LOAD_SH_REG packet sets the SH_REG_BASE and the CONTEXT_CONTROL packet enables/disables write shadowing to external memory (see these packets for more details). The starting external memory address that the render state data is written to is computed as follows:

- Mem_Start_Address[47:2] = SH_REG_BASE[47:2] + REG_OFFSET

To preserve coherency between shadowed Set packet writes and Load packet reads from external memory, the CP.PFP first waits until all prior SET_* data has been shadowed to memory before issuing the Load"s memory read requests for the register data.

SET_SH_REG Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet. Shader_Type in bit 1 of the Header will correspond to the shader type of the Load, see Type-3 Packet.

2	REG_OFFSET	[15:0] - Offset in DWords from the register base address (0x2C00 in DWs) and memory base address (SH_REG_BASE).
3 to N	REG_DATA	DWord Data for Registers.

2.5 Draw Related Packets: Graphics Ring

2.5.1 DRAW_INDEX_2

Draws a set of primitives using fetched indices from a bounded index buffer.

DRAW_INDEX_2 Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	MAX_SIZE	MAX_SIZE [31:0] - VGT DMA maximum number of indices until out of bound index buffer is accessed. Written to the VGT_DMA_MAX_SIZE register (No Context Supplied).
3	INDEX_BASE_LO	Base Address [31:1] of Index Buffer (Word-Aligned). Written to the VGT_DMA_BASE register (No Context Supplied).
4	INDEX_BASE_HI	[15:0] Base Address Hi [47:32] of Index Buffer. Written to the VGT_DMA_BASE_HI register (No Context Supplied).
5	INDEX_COUNT	INDEX_COUNT [31:0] - Number of indices in the Index Buffer. Written to the VGT_DMA_SIZE register (No Context Supplied). Written to the VGT_NUM_INDICES register for the assigned context.
6	DRAW_INITIATOR	Draw Initiator Register. Written to the VGT_DRAW_INITIATOR register for the assigned context.

2.5.2 DRAW_INDEX_AUTO

Draws a set of primitives using indices auto-generated by the VGT.

DRAW_INDEX_AUTO Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	INDEX_COUNT	[31:0] Number of indices to generate. Written to the VGT_NUM_INDICES register for the assigned context.
3	DRAW_INITIATOR	[31:0] Primitive type and other control. Written to the VGT_DRAW_INITIATOR register for the assigned context. DRAW_INDEX_AUTO Packet

2.5.3 DRAW_INDEX_IMMED

Draws a set of primitives using indices in the packet.

DRAW_INDEX_IMMED Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	INDEX_COUNT	INDEX_COUNT [31:0] - Number of indices that will be written to the VGT_IMMED_DATA register. Written to the VGT_NUM_INDICES register

		for the assigned context.
3	DRAW_INITIATOR	Primitive type and other control. Written to the VGT_DRAW_INITIATOR register for the assigned context.
4 to End	[indx16 #1 | indx16 #0] or [indx32 #0	Index Data. Written to the VGT_IMMED_DATA register for the assigned context. See the DRAW_INDEX_TYPE packet for details on how to specify the 16 or 32-bit indices.

2.5.4 DRAW_INDEX_INDIRECT

The information needed for the Draw is embedded in a buffer (rather than in the packet) and therefore the CP must fetch the information from memory before executing the Draw. The data structure in memory has this format:

```
struct DrawIndexedInstancedArgs
{
  UINT IndexCountPerInstance;
  UINT InstanceCount;
  UINT StartIndexLocation;
  UINT BaseVertexLocation;
  UINT StartInstanceLocation;
};
```

Definition of Parameters:

- IndexCountPerInstance: The number of indexes per instance of the index buffer that indexes are read from to draw the primitives.
- InstanceCount: The number of instances of the index buffer that indexes are read from to draw the primitives.
- StartIndexLocation: Index of the first index to use when accessing the vertex buffer; begin at StartIndexLocation to index vertices from the vertex buffer.
- BaseVertexLocation: The number that should be added to each index that is referenced by the various primitives to determine the actual index of the vertex elements in each vertex stream.
- StartInstanceLocation: The first instance of the index buffer that indexes are read from to draw the primitives.

The driver must send the following packets before sending the DRAW_INDEX_INDIRECT packet: SET_BASE packet to specify the start address of BufferForArgs, the INDEX_BASE packet to specify where the index buffer starts and the INDEX_BUFFER_SIZE packet to specify the number of indices.

DRAW_INDEX_INDIRECT Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	DATA_OFFSET	[31:0] - Byte aligned offset where the required data structure in the DrawIndirect buffer starts. 1:0 must be zero.
3	BASE_VTX_LOC	[15:0] Offset where the CP will write the BaseVertexLocation it fetched from memory. START_INST_LOC will be written to the subsequent register. Two sequential User Data location will be used, likely the user register 0 and 1 in either the VS, LS, or ES depending on where the vertex shader is executing.
4	DRAW_INITIATOR	Draw Initiator Register. Written to the VGT_DRAW_INITIATOR register for the

		assigned context.

2.5.5 DRAW_INDEX_MULTI_AUTO

Combines several individual packets into a single one to allow fast draws of primitives where the number of indices is small.

DRAW_INDEX_MULTI_AUTO Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	PRIMCOUNT	[31:0] – Primitive count
3	DRAW_INITIATOR	Draw Initiator Register. Written to the VGT_DRAW_INITIATOR register for the assigned context.
4	CONTROL	bit[15:0]INDEX_OFFSET – Start index for this primitive. bit[20:16]PRIM_TYPE – Primitive type. bit[31:21]INDEX_COUNT - # of indices for the VGT to generate for this prim.

2.5.6 DRAW_INDEX_OFFSET_2

The purpose of this packet, in conjunction with the INDEX_TYPE Packet and INDEX_BASE packets, draws a set of primitives using fetched indices from a bounded index buffer while minimizing the amount of address patching that the driver must do Vista BDM. The base of the index buffer, supplied in the INDEX_BASE packet, and the index type (16 bit or 32 bit), supplied in the INDEX_TYPE Packet, must have already been sent when this packet arrives at the CP.

DRAW_INDEX_OFFSET_2 Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	MAX_SIZE	MAX_SIZE [31:0] - VGT DMA maximum number of indices until out of bound index buffer is accessed. Written to the VGT_DMA_MAX_SIZE register (No Context Supplied).
3	INDEX_OFFSET	Starting index number in the index buffer. INDEX_OFFSET of zero represents the first index, index one is the second index.
4	INDEX_COUNT	INDEX_COUNT [31:0] - Number of indices in the Index Buffer. Written to the VGT_DMA_SIZE register (No Context Supplied). Written to the VGT_NUM_INDICES register for the assigned context.
5	DRAW_INITIATOR	Draw Initiator Register. Written to the VGT_DRAW_INITIATOR register for the assigned context.

2.5.7 DRAW_INDIRECT

The information needed for the Draw is embedded in a buffer (rather than in the packet) and therefore the CP must fetch the information from memory before executing the Draw. The data structure in memory has this format:

```
struct DrawInstancedArgs
{
  UINT VertexCountPerInstance;
```

```
    UINT InstanceCount;
    UINT StartVertexLocation;
    UINT StartInstanceLocation;
};
```

Note: See the DRAW_INDEX_INDIRECT packet for the definition of Terms. The driver must send the following packet before sending the DRAW_INDEX_INDIRECT packet: SET_BASE packet to specify the start address of BufferForArgs. This packet will write to two SGPRs, so they need to be included in the total number the SPI loads and it must be coordinated with the shader compiler.

DRAW_INDIRECT Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	DATA_OFFSET	[31:0] - Byte aligned offset where the required data structure in the DrawIndirect buffer starts. 1:0 must be zero.
3	BASE_VTX_LOC	[15:0] 16 bit offset where the CP will write the BaseVertexLocation it fetched from memory. This will be a one of the16 SGPR user data registers.
4	START_INST_LOC	[15:0] Offset where the CP will write the StartInstanceLocation it fetched from memory. This should be the SGPR that follows the one specified for the BASE_VTX_LOC.
5	DRAW_INITIATOR	Draw Initiator Register. Written to the VGT_DRAW_INITIATOR register for the assigned context.

2.5.8 INCREMENT_DE_COUNTER

In the ME this packet creates a pipelined event that causes the CP to EOP block to increment it counter. If the packet command specifies to clear the counter, the ME does this at the top of pipe and clears the associated counter. The counter is double buffered.

INCREMENT_DE_COUNTER Packet Description

DW	Field	Description
1	HEADER	Header of the packet.
2	DUMMY	[31:0] Dummy Data.

2.5.9 INDEX_BASE

The CP saves the INDEX_BASE address in this packet, so when the CP processes the DRAW_INDEX_OFFSET packet it can add the base address to the (offset shifted one or two bits depending on the size of the index (16 bits or 32 bits) specified previously in the INDEX_TYPE). This packet is considered part of the draw packet sequence, so the INDEX_BASE is not shadowed. If this packet is not sent before each draw then it will need to be in the preamble of each command buffer to ensure it gets set correctly before the first draw.

INDEX_BASE Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	INDEX_BASE_LO	[30:0] - Base Address [31:1] of Index Buffer (Word-Aligned).

3	INDEX_BASE_HI	[15:0] Base Address Hi [47:32] of Index Buffer.

2.5.10 INDEX_BUFFER_SIZE

The purpose of the INDEX_BUFFER_SIZE packet, in conjunction with the INDEX_TYPE, INDEX_BASE and DRAW_INDEX_INDIRECT packets, is to allow the CP to calculate the value to write to the VGT_DMA_MAX_SIZE register. See the DRAW_INDEX_INDIRECT packet for how it is used for that packet.

INDEX_BUFFER_SIZE Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	INDEX_BUFFER_SIZE	Bits[31:0] - The number of indices in the index buffer.

2.5.11 INDEX_TYPE

This packet is considered part of the draw packet sequence, so the VGT_INDEX_TYPE is not shadowed. If this packet is not sent before each draw then it will need to be in the preamble of each command buffer to ensure it gets set correctly before the first draw.

INDEX_TYPE Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	INDEX_TYPE SWAP_MODE	[0] Index Type - 0 = 16-bits; - 1 = 32-bits. [3:2] Swap Mode[1:0] - Byte swapping control.

2.5.12 INDIRECT_BUFFER

This packet is used for dispatching Indirect Buffers. The KMD will specify the VMID in each Indirect_Buffer packet. The KMD is required to include the VMID in the Indirect_Buffer packet.

INDIRECT_BUFFER Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	IB_BASE_LO	[31:2] - Indirect Buffer Base Address[31:2] - DW-Aligned [1:0] - Swap function used for data write.
3	IB_BASE_HI	[15:0] - Upper bits of Address [47:32]
4	VMID	[31:24] - VMID[7:0], Virtual Memory Domain ID for the command buffer.
	IB_SIZE	[19:0] - Indirect Buffer Size [19:0], size of the Indirect Buffer in DWORDs.

2.5.13 MPEG_INDEX

MPEG_INDEX: Packed register writes for MPEG and Generation of Indices.

MPEG_INDEX Packet Description

DW	Field Name	Description
1	HEADER	Header field of the packet.

2	NUM_INDICES	Number of Indices the VGT will actually fetch + 3 * number of base indices given at end of this packet. Valid values are 0x0003 to 0x3FFF.
3	DRAW_INITIATOR	Written Unconditional to VGT_DRAW_INITIATOR register
4 to 4 + ((NUM_INDICES/3) - 1)	32-Bit INDEX	First Index of Rect. (0x00000000 to 0xFFFFFFFD) For each First Index", CP will generate the other 2 indices and output: FIRST_INDEX FIRST_INDEX+1 FIRST_INDEX+2 All indices are written to the VGT_IMMED_DATA register.

2.5.14 NUM_INSTANCES

NUM_INSTANCES is used to specify the number of instances for the subsequent draw command.

This packet is considered part of the draw packet sequence, so the VGT_NUM_INSTANCES is not shadowed. If this packet is not sent before each draw then it will need to be in the preamble of each command buffer to ensure it gets set correctly before the first draw.

NUM_INSTANCES Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	NUM_INSTANCES	[31:0] Number of Instances. Minimum value is one; if zero is programmed, it will be treated as one. This allows for a max of 2^32 - 1 instances.

2.5.15 WAIT_ON_AVAIL_BUFFER

This packet is inserted by the driver into the draw command buffer when, and only when, it inserts the SET_CE_DE_COUNTERS packet into the constant command buffer. This indicates that the following indirect buffer should use switch to using the other ping-pong buffer, but it must wait until the buffer is available. The DE is not allowed to get ahead of the CE. The CE forwards both of its flags to the DE. The algorithm for the DE is as follows:

CE 1	CE0	DE1	DE0	Action
0	0	0	0	Reset state, CE can set a flag when it receives SET_CE_DE_COUNTERS
0	0	0	1	Invalid. DE can not have a buffer active if the CE does not. (Invalid #1)
0	0	1	0	(Invalid #1)
0	0	1	1	(Invalid #1)
0	1	0	0	CE set buffer 0, when the DE receives WAIT_AVAIL_BUFFER it will set DE0.
0	1	0	1	CE and DE are both active on buffer 0.
0	1	1	0	(Invalid #1)
0	1	1	1	(Invalid #1)
1	0	0	0	CE set buffer 1, when the DE receives WAIT_AVAIL_BUFFER it will set DE1.
1	0	0	1	(Invalid #1)
1	0	1	0	CE and DE are both active on buffer 1.

1	0	1	1	(Invalid #1)
1	1	0	0	CE is working both buffers, but the DE has not yet even processed the WAIT_ON_AVAIL_BUFFER for the first buffer.
1	1	0	1	CE and DE are both active on buffer 0.CE set buffer 1, when the DE receives WAIT_AVAIL_BUFFER it will set DE1.
1	1	1	0	CE and DE are both active on buffer 1. CE set buffer 0, when the DE receives WAIT_AVAIL_BUFFER it will set DE0.
1	1	1	1	CE and DE are both active on buffer 0. CE and DE are both active on buffer 1.

WAIT_ON_AVAIL_BUFFER Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	Dummy	Dummy DWord is discarded by the CE.

2.5.16 WAIT_ON_CE_COUNTER

Instructs the ME to wait on CE Counter (Write Confirm Constant Set Counts) to be greater than zero.

WAIT_ON_CE_COUNTER Packet Description

DW	Field	Description
1	HEADER	Header of the packet.
2	Dummy	Dummy DW gets discarded.

2.6 Dispatch Packets

2.6.1 DISPATCH_DIRECT

Dispatches a compute job using the parameters embedded in the packet.

DISPATCH_DIRECT Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet. Shader_Type in bit 1 of the Header will be set to 1, since Dispatches are only used for Compute Shaders, see Type-3 Packet.
2	DIM_X	[31:0] + x dimensions of the array of thread groups to be dispatched
3	DIM_Y	[31:0] + y dimensions of the array of thread groups to be dispatched
4	DIM_Z	[31:0] + z dimensions of the array of thread groups to be dispatched
5	DISPATCH_INITIATOR	Dispatch Initiator Register. Written to the VGT_DISPATCH_INITIATOR register for the assigned context.

2.6.2 DISPATCH_INDIRECT

Dispatches a compute job using the parameters fetched from memory.

//At the specified offset, the following data members will be in this order.

```
struct GroupDimensions;
 {
   UINT DIM_X;
   UINT DIM_Y;
   UINT DIM_Z;
 };
```

DISPATCH_INDIRECT Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet. Shader_Type in bit 1 of the Header will be set to 1, since Dispatches are only used for Compute Shaders, see Type-3 Packet.
2	DATA_OFFSET	[31:0] - Byte aligned offset where the required data structure starts.1:0 = “00”.
3	DISPATCH_INITIATOR	Dispatch Initiator Register. Written to the VGT_DISPATCH_INITIATOR register for the assigned context.

2.7 Predication Packets

2.7.1 COND_EXEC

This packet allows the CP to perform a conditional execution of a sequence of packets (type 0, 2, and type 3) based on a Boolean value stored in memory (1 = continue, 0 = COMMAND).

Before sending a COND_EXEC packet the driver allocates a memory location and set it to 0x00000001. It will clear it to zero if it needs the CP to stop and perform the command in the packet the next time the CP encounters a COND_EXEC packet.

Note: Care must be taken to make certain that EXEC_COUNT contains the exact number of DWords for the subsequent packets that are to be predicated if the Boolean value is zero. The CP will start parsing the DWord immediately following EXEC_COUNT DWords. If this is not a packet header, the device will encounter corruption or hang.

COND_EXEC Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	BOOL_ADDR_LO	[31:2] is Boolean Address bits [31:2].
3	COMMAND	Bits[31:28] Command [3:0] 0x0 = DISCARD 0xF to 0x01= Reserved.
	BOOL_ADDR_HI	Bits[15:0] Boolean Address bits [47:32] .
4	EXEC_COUNT	EXEC_COUNT: [13:0] - total number of DWords of the subsequent predicated packets. This count wraps the packets that will be predicated by the device select.
5	Fence	[30:0] Incrementing Fence value.

2.7.2 COND_WRITE

The CP reads either a memory or a register location (indicated by POLL_SPACE) and tests the polled value with the reference value provided in the command packet. The test is qualified by both the specified function and mask. If the test passes, the write occurs to either a register or memory depending on WRITE_SPACE. If the test fails, the CP skips the write. In either case, the CP then continues parsing the command stream.

COND_WRITE Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	WRITE_SPACE | POLL_SPACE | FUNCTION	[31:9] – Reserved [8] - WRITE_SPACE - 0=>Register, - 1=>Memory 7:5 - Reserved [4]: POLL_SPACE - 0=>Register, - 1=>Memory [2:0] - FUNCTION - 000 - Always (Compare Passes). Still does read operation and waits for data to return. - 001 - Less Than (<) the Reference Value. - 010 - Less Than or Equal (<=) to the Reference Value. - 011 - Equal (=) to the Reference Value. - 100 - Not Equal (!=) to the Reference Value. - 101 - Greater Than or Equal (>=) to the Reference Value. - 110 - Greater Than (>) the Reference Value. - 111 - Reserved
3	POLL_ADDRESS_LO	Lower portion of Address to poll If the address is a memory location then bits [31:2] specify the lower bits of the address and [1:0] is the swap code to be used.If the address is a memory-mapped register, then bits [15:0] is the DWord memory-mapped register address that the CP will read.
4	POLL_ADDRESS_HI	Higher portion Address to poll. If the address is a memory location then bits [15:0] specify bits 47:32 of the address. If the address is a memory-mapped register, then this DW is a don"t care.
5	REFERENCE	Reference Value [31:0].
6	MASK	Mask for Comparison [31:0]
7	WRITE_ADDRESS_LO	If WRITE_SPACE + Register: WRITE_ADDRESS[15:0] - DWord memory-mapped register address that the will be written. Else If WRITE_SPACE + Memory: WRITE_ADDRESS[31:2] - DWord-Aligned Address of destination memory location. WRITE_ADDRESS[1:0] - SWAP Used for Memory Write.
8	WRITE_ADDRESS_HI	If WRITE_SPACE + Register: This DWord is a don’t matter. Else if WRITE_SPACE + Memory: 7:0 - WRITE_ADDRESS[47:32]

9	WRITE_DATA	Write Data[31:0] that will be conditionally written to the ADDRESS.

2.7.3 PRED_EXEC

Functionality Perform a predicated execution of a sequence of packets (type 0, 2, and type 3) on select devices.

Notes: The ME_INITIALIZE packet includes a GPU unique Device ID. Care must be taken to make certain that EXEC_COUNT contains the exact number of DWords for the subsequent packets that are to be predicated. The CP.PFP will start parsing the DWord immediately following EXEC_COUNT DWords. If this is not a packet header, the device will encounter corruption or hang.

PRED_EXEC Packet Description

DW	Field Name	Description
1	HEADER	header; Header of the packet
2	CONTROL	[31:24] device_select; To select one or more device IDs upon which the subsequent predicated packets will be executed [13:0] exec_count; Total number of DWords of the subsequent predicated packets. This count wraps the packets that will be predicated by the device select.

2.7.4 SET_PREDICATION

The SET_PREDICTION packet provides a single flexible packet for the driver to specify type of predication check for previous events: ZPASS, PRIMCOUNT, etc.

SET_PREDICATION Packet Description

DW	HEADER	Header of the packet
1	START_ADDR_LO	[31:4] is start address bits [31:4]. Supports a 16 byte aligned address for DB0 count.
2	PRED_PROPERITES	startAddrHi [15:0] Start Address bits [47:32]
		prediationBooleon[8] Predication Boolean (valid for both ops) - 0: Draw if not visible/overflow - 1: Draw if visible/no overflow
		hint[12] (Only valid for Zpass/Occlusion Predicate) -0: CP must wait until final ZPass counts have been written by all DBs. -1: CP should read the results once, if all DBs have not written the results to memory then draw.
		predOp[18:16] predicate operation -000: Clear Predicate -001: Set Zpass Predicate -010: Set PrimCount Predicate -011: Reserved -1xx: Reserved
		continue[31] - Continue set predication (Validfor both ZPASS). This field is used to allow accumulation of ZPASS count data across command buffer boundaries. - 0: This SET_PREDICTION packet is a unique packet or the first of a series of SET_PREDICATION packets. - 1: This SET_PREDICATION packet is a continuation of the previous one

2.8 Synchronization

2.8.1 ATOMIC

Sent only in the DE command buffer to request that the CP does either a single atomic operation or an atomic loop. All supported atomics can perform a single atomic operation, and only CMPSWP atomics support loop CMD. The CP will wait until the preop value is returned and place it into the CP_ATOMIC_PREOP_LO, and for 64 bit atomics the CP_ATOMIC_PREOP_HI register before proceeding to the next packet or next loop. For CMPSWP atomics with CMD = Loop, the CP will take the preop RTN value and compare it to packet compare value (CMP_DATA_*), if equal the packet ends, else it loops again. Subsequent packets may operate on the value returned. For context switching this register needs to be saved and restored. Only opcodes that return preop values are supported for this packet.

ATOMIC Packet Description

DW	Field	Description
1	HEADER	Header of the packet.
2	OP	[6:0] Atomic Operation. This field can only be used with TC_OP_ATOMIC_* ops.
	RSVD	[7] Reserved.
	CMD	[11:8] Command. 0000 Single pass atomic. 0001 Loop until compare is satisfied.
	RSVD	[31:12] Reserved.
3	ADDR_LO	[31:0] 32 byte aligned Address[31:0]. Bits[1:0] are zero for 32 bit, Bits[2:0] are zero for 64 bit.
4	ADDR_HI	[31:0] Address High[63:32].
5	SRC_DATA_LO	[31:0] Lower 32-bits of the atomic source data.
6	SRC_DATA_HI	[31:0] Upper 32-bits of the atomic source data.
7	CMP_DATA_LO	[31:0] Lower 32-bits of the atomic compare data, if op is ATOMIC_CMPSWP_RTN, else the DW is zero.
8	CMP_DATA_HI	[31:0] Higher 32-bits of the atomic compare data, if op is ATOMIC_CMPSWP_RTN_64, else the DW is zero
9	LOOP_INTERVAL	[12:0] When the Command is a "Loop", the CP will wait 64*LOOP_INTERVAL before requesting another ATOMIC_CMPSWP_RTN transaction.

Combinations

OP	CMD	ADDR_LO/HI	SRC_DATA	CMP_DATA	LOOP_INTERVAL
Non-CMPSWP	0	yes	yes	n/a (0x0)	n/a (0x0)
CMPSWP	0	yes	yes	n/a (0x0)	n/a (0x0)
CMPSWP	1	yes	yes	yes	yes

2.8.2 ATOMIC_GDS

The purpose of this packet is to support Atomic operations in the GDS from the CP.

If the Atomic Op returns pre-op source data, the CP will read the data and store it the CP_GDS_ATOMIC* registers. The driver must indicate that the CP needs to do this by setting the "ATOM_READ" and "ATOM_RD_CNTL" control bits in the packet. Reads take a long time to complete, therefore the "ATOM_RD_CNTL" bits allow the driver to optimize for size (32-bits vs. 64-bits) and number of return values (1 or 2). The COPY_DATA packet can be used to "copy" the read return data to various destinations (see COPY_DATA packet for more details).

The GDS supports Compare-Swap Atomic operations. For these ops, the compare data is placed in the ATOM_SRC0* ordinals and the source data is placed in the ATOM_SRC1* ordinals. If the CP should repeat the compare-swap operation until it passes, then the "ATOM_CMP_SWAP" control bit should be set. If the CP does not need to repeat until it passes, then it should not be set. Whenever the "ATOM_CMP_SWAP" control bit is set, the "ATOM_READ" control bit should also be set and the "ATOM_RD_CNTL" bit should be set equal to either '0' or '2'.

If the Atomic Op does not return pre-op source data and the driver wants confirmation that the Atomic Op that has completed, it must set the ATOM_COMPLETE control bit. The ATOM_COMPLETE and ATOM_READ bits should never both be set. Setting neither of the bits is also valid.

The GDS_ATOM_SRC0 triggers the Atomic operation in the GDS and the microcode will therefore write it last.

Any fields not used by the Atomic operation specified can be set to 0.

ATOMIC_GDS Packet Description

DW	Field	Description
1	HEADER	Header of the packet.
2	ATOM_OP	[6:0] ATOM_OP Atomic Operation – See ds_opcodes.docx for complete list of supported opcodes.
	ATOM_CMP_SWAP	[16] ATOM_CMP_SWAP 0 – Indicates the Atomic operation type does not need to repeat. 1 – Indicates the Atomic operation type is a compare-swap and should repeat until it passes.
	ATOM_COMPLETE	[17] ATOM_COMPLETE 0 – Do not wait for GDS Atomic operations to complete. 1 – Wait for all outstanding GDS Atomic operations to complete. Intended for use with Atomic ops that do not return pre-op source data.
	ATOM_READ	[18] ATOM_READ 0 – Do not read the atomic pre-op source0 data. 1 – Read the atomic pre-op source0 data.
	ATOM_RD_CNTL	[20:19] ATOM_RD_CNTL 0 – 32-bits, 1 return value. 1 – 32-bits, 2 return values. 2 – 64-bits, 1 return value. 3 – 64-bits, 2 return values.
3	ATOM_CNTL	[8:0] ATOM_CNTL – See GDS_ATOM_CNTL register for more details. [5:0] Auto-Increment in Bytes.

		[7:6] Reserved. [8] DMODE – controls flushing of denorms.
4	ATOM_BASE	[15:0] ATOM_BASE – See byte granularity GDS_ATOM_BASE register for more details. Base address for Atomic operation relative to the GDS partition base. See the SET_BASE packet for details on setting the GDS partition bases.
5	ATOM_SIZE	[15:0] ATOM_SIZE – See GDS_ATOM_SIZE register for more details. Size in bytes of the DS memory. Determines where clamping begins.
6	ATOM_OFFSET0 ATOM_OFFSET1	[7:0] ATOM_OFFSET0 – See GDS_ATOM_OFFSET0 register for more details. Used to calculate the address of the corresponding source operation. [23:16] ATOM_OFFSET1 – See GDS_ATOM_OFFSET1 register for more details. Used to calculate the address of the corresponding source operation.
7	ATOM_DST	[31:0] ATOM_DST – See GDS_ATOM_DST register for more details. DS Memory address to perform the Atomic operation.
8	ATOM_SRC0	[31:0] ATOM_SRC0 – See GDS_ATOM_SRC0 register for more details. Lower 32-bits of the atomic source0 data for non compare-swap atomic ops. Lower 32-bits of the atomic compare data for compare-swap atomic ops.
9	ATOM_SRC0_U	[31:0] ATOM_SRC0_U – See GDS_ATOM_SRC0_U register for more details. Upper 32-bits of the atomic source0 data for non compare-swap atomic ops. Upper 32-bits of the atomic compare data for compare-swap atomic ops.
10	ATOM_SRC1	[31:0] ATOM_SRC1 – See GDS_ATOM_SRC1 register for more details. Lower 32-bits of the atomic source1 data and source data for compare-swap atomic ops.
11	ATOM_SRC1_U	[31:0] ATOM_SRC1_U – See GDS_ATOM_SRC1_U register for more details. Upper 32-bits of the atomic source1 data and source data for compare-swap atomic ops.

2.8.3 EVENT_WRITE

This packet is used when the driver wants to create a non-TimeStamp/Fence event. See EVENT_WRITE_EOP to send timestamps and fences. The EVENT_WRITE supports two categories of events. Those are:

- 4 DW (DW) event where special handling is required: ZPASS, SAMPLE_PIPELINESTATS, SAMPLE_STREAMOUTSTATS[,1,2,3].
- 2 DW (DW) event where no special handling is required; CP just writes EVENT_TYPE (bits[5:0] of DW 2 from the packet) into VGT_EVENT_INITIATOR register and DWs 3 and 4 do not exist, i.e., the packet is only 2 DWs for these events. These include all other events.

When the EVENT_INDEX is set to '0111' for the CACHE_FLUSH* events, there is also an option to invalidate the TC's L2 cache: INV_L2.

EVENT_WRITE Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	EVENT_CNTL	INV_L2[20] Send WBINVL2 op to the TC L2 cache when EVENT_INDEX = 0111. EVENT_INDEX[11:8] 0000: Any non-Time Stamp/non-Fence/non-Trap EVENT_TYPE not listed.

		0001: ZPASS_DONE 0010: SAMPLE_PIPELINESTAT 0011: SAMPLE_STREAMOUTSTAT[S|S1|S2|S3] 0100: [CS|VS|PS]_PARTIAL_FLUSH 0101: *Reserved for EVENT_WRITE_EOP time stamp/fence event types* 0110: *Reserved for EVENT_WRITE_EOS packet* 0111: CACHE_FLUSH, CACHE_FLUSH_AND_INV_EVENT 1000 - 1111: *Reserved*. EVENT_TYPE[5:0] The CP writes this value to the VGT_EVENT_INITIATOR register for the assigned context.
3	ADDRESS_LO	ADDRESS_LO[31:3] Lower bits of QWORD-Aligned Address. [2:0] - Reserved & must be programmed to zero. Driver should only supply this DW for Sample_PipelineStats, Sample_StreamoutStats, and Zpass (Occlusion).
4	ADDRESS_HI	ADDRESS_HI[15:0] Upper bits of Address [47:32] Driver should only supply this DW for Sample_PipelineStats, Sample_StreamoutStats, and Zpass (Occlusion).

2.8.4 EVENT_WRITE_EOP

The EVENT_WRITE_EOP packet is used when the driver wants to create any end-of-pipe event. TS used below is historical and indicates either fence data, trap or actual timestamp will be written back. Supported Events are:

- Cache Flush TS: provides the driver with a pipelined fence/timestamp indicating that the CBs and DBs have completed flushing their caches.
- Cache Flush And Inval TS: same as above but the CBs and DBs also invalidate their caches before sending the pulse back to the CP.
- Bottom Of Pipe TS: provides the driver with a pipelined timestamp indicating that the CBs and DBs have completed all work before the time stamp. This can be considered a read EOP event in that all reads have occurred but the CBs/DBsz have not written out all the data in their caches.

Use the EVENT_WRITE packet for all others. Supported actions when requested event has completed are:

- Timestamps - 64-bit global GPU clock counter value or CP_PERFCOUNTER_HI/LO, either with optional interrupt .
- Fences - 32 or 64 bit embedded data in the packet with optional interrupt. The privilege vs. unprivileged designation is based on the privilege level of the DMA buffer that included the EVENT_WRITE_EOP packet, not anything to do with the packet itself.
- Traps (interrupt only).

There is also an option to invalidate the TC's L2 cache: INV_L2.

EVENT_WRITE_EOP Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	EVENT_CNTL	INV_L2[20] Send WBINVL2 op to the TC L2 cache. EVENT_INDEX[11:8] 0000 - 0100: *Reserved for EVENT_WRITE packet*.

		0101: EVENT_WRITE_EOP event types 0110: *Reserved for EVENT_WRITE_EOS packet*. 0111: *Reserved for EVENT_WRITE packet*. 1000 - 1111: *Reserved*. EVENT_TYPE[5:0] The CP writes this value to the VGT_EVENT_INITIATOR register for the assigned context.
3	ADDRESS_LO	ADDRESS_LO[31:2] Lower bits of DWord-Aligned Address if DATA_SEL = 001", [31:3] - Lower bits of QWORD-Aligned Address if DATA_SEL = 010" or 011", Else don"t care. [1:0] - Reserved & must be programmed to zero.
4	DATA_CNTL	DATA_SEL[31:29] Selects Source of Data to be written for a End-of-Pipe Done event. 000 - None, i.e., Discard Data. 001 - Send 32-bit Data Low (Discard Data High). 010 - Send 64-bit Data. 011 - Send current value of the 64 bit global GPU clock counter. 100 - Send current value of the CP_PERFCOUNTER_HI/LO. The intent is for the driver to have already selected the “always count” sclks option (0x0) for CP_PERFCOUNTER _SELECT and requested the CP to start counting via the CP_PERFMON_CNTL register. 101 - *Reserved*. 110 - *Reserved*. 111 - *Reserved*. INT_SEL[25:24] Selects interrupt action for End-of-Pipe Done event. 00 - None (Do not send an interrupt). 01 - Send Interrupt Only. Program DATA_SEL 000". 10 - Send Interrupt when Write Confirm is received from the MC. ADDRESS_HI[15:0] Address bits[47:32] . External memory address written for a End-of-Pipe Done event. Read returns last value written to memory.
5	DATA_LO	DATA_LO[31:0] Value that will be written to memory when event occurs. Driver should always supply this DW
6	DATA_HI	DATA_HI[63:32] Value that will be written to memory when event occurs. Driver should always supply this DW

2.8.5 EVENT_WRITE_EOS

The EVENT_WRITE_EOS packet is used when the driver wants to create any end-of-shader event (end of CS or end of PS). Supported Events are CS Done and PS Done.

When the CMD = 001, the CP will copy SIZE dwords starting from the partition base (see GDS_*_INDEX as described in SET_BASE packet for more details) plus GDS_INDEX to the memory address specified. The corresponding partition is determined from the Ring to which the packet is submitted.

EVENT_WRITE_EOS Packet Description

DW	Field	Description
1	HEADER	Header with Shader_Type in bit[1]. Determines event that is sent for Ring 0.
2	EVENT_CNTL	EVENT_INDEX[11:8] 0000 - 0100: Reserved for EVENT_WRITE packet 0101: Reserved for EVENT_WRITE_EOP packet 0110: CS Done, PS Done 0111: Reserved for EVENT_WRITE packet 1000 - 1111: Reserved. EVENT_TYPE[5:0] The CP writes this value to the VGT_EVENT_INITIATOR register for the assigned context.
3	ADDRESS_LO	ADDRESS_LO[31:2] Lower bits of DWord-Aligned Address. [1:0] - Reserved & must be programmed to zero.
4	CMD_INFO	CMD[31:29] 000: Reserved 001: Store GDS Data to memory. (per partition in SI) 010: After all previous GDS data has been write confirmed, store 32-bit “DATA” from this packet to memory as a fence. (evergreen-SI). 011 - 111: Reserved ADDRESS_HI[15:0] Address bits[47:32]. External memory address written for a End-of-Shader Done event. Read returns last value written to memory.
5	DATA_INFO	DATA[31:0] – Used with CMD = 010 EOS fence value that will be written to memory when event occurs. This dword is always supplied, but only valid if "CMD" + Store Packet Data. OR SIZE[31:16] – Used with CMD = 001. Number of DWs to read from the GDS. Currently supports reading of 16KDWs. A value of zero is not supported when CMD = “001”. GDS_INDEX[15:0] – Used with CMD = 001. Indexed offset from the start of the segment within the partition

2.8.6 MEM_SEMAPHORE

The MEM_SEMAPHORE packet supports Signal and Wait Semaphores. Wait Semaphores are executed at the top of pipe (CP) and a Signal Semaphores are executed at the bottom of pipe (after whatever work before it has been completed). If the CP processes a Wait Semaphore there could be a Signal Semaphore still in the Gfx pipe behind draws still being rendered.

MEM_SEMAPHORE Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	ADDRESS_LO	[31:3] Lower bits of QWORD-Aligned Address.
3	SEM_SEL	[31:29] - SEM_SEL - Select either Wait or Signal. - 110: Signal Semaphore.

		- 111: Wait Semaphore.
	CLIENT_CODE	[25:24] - CLIENT_CODE - Client Code - 00: CP - 01: CB - 10: DB - 11: Reserved
	SIGNAL_TYPE	[20] - SIGNAL_TYPE - Signal Type - 0: SEM_SEL + Signal Semaphore and signal type is increment, or the SEM_SEL + Wait Semaphore - 1: SEM_SEL + Signal Semaphore and signal type is write '1'.
	USE_MAILBOX	[16] USE_MAILBOX0 - Signal Semaphore will not wait for mailbox to be written1 - Signal Semaphore will wait for mailbox to be written
	WAIT_ON_SIGNAL	[12] WAIT_ON_SIGNAL - If set the Wait_Semaphore will wait until all outstanding End of Pipe (and therefore Signal_Semaphores) have completed, before being issued. (evergreen only, other asics set to 0) - 0: Don't wait for all Signal Semaphores to complete. - 1: Wait for all Signal Semaphores to complete.
	ADDRESS_HI	[7:0] - ADDRESS_HI - Upper bits (39:32) of Address

2.8.7 OCCLUSION_QUERY

The motivation for this packet is to allow the application to access the accumulated query counts from the shader. Before this packet, the application had to do the query at the driver level.

Available Controls

1. Specify the 4-byte aligned 40-bit MC address where the current 64-bit accumulation value is stored.
2. Specify the 16-byte aligned 40-bit MC starting address where a set of eight DB Zpass count pairs are stored.

Hard coded behavior

1. The Driver must initialize Begin and End occlusion data for DBs that do exist to 0x00000000 and for those that don’t exist to 0x80000000.
2. The Driver must initialize AccumCnt to zero before the OCCLUSIONQUERY packet is sent to the GPU.
3. The CP will keep reading the DB ZPASS occlusion data until they are all valid.
4. The CP will write-confirm the final accumulated value before proceeding to the next packet.

OCCLUSION_QUERY Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet.
2	START_ADDR_LO	[31:4] - DB0 Start Address bits [31:4]. Supports a 16 byte aligned address for DB0 count. [3:0] - Reserved, program to zero.
3	START_ADDR_HI	[15:0] - DB0 Start Address bits [47:32] .
4	QUERY_ADDR_LO	[31:2] - DW aligned Accumulated Query address bits [31:2]. [1:0] - Reserved, program to zero.

5	QUERY_ADDR_HI	[31:8] - Reserved, program to zero. [15:0] - Accumlated Query address bits [47:32]

2.8.8 PFP_SYNC_ME

This packet is inserted by the driver when it needs the PFP to stall or wait until the ME is at the synced up to the PFP.

PFP_SYNC_ME Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	DUMMY	Dummy Data

2.8.9 STRMOUT_BUFFER_UPDATE

The STRMOUT_BUFFER_UPDATE packet is expected to be used in a variety of streamout scenarios. When a streamout operation spreads across two command buffers, the driver needs to ensure BufferFilledSize is captured for each streamout buffer at the end of the first command buffer and restart streamout buffers with the captured values in the next command buffer.

STRMOUT_BUFFER_UPDATE Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	CONTROL	[31:10] - Reserved [9:8] - Buffer Select, indicates the stream out being updated - 00: Stream out buffer 0 - 01: Stream out buffer 1 - 10: Stream out buffer 2 - 11: Stream out buffer 3 [7:3] - Reserved [2:1] - Source_Select: to write into VGT_STRMOUT_BUFFER_OFFSET - 00: Use BUFFER_OFFSET in this packet - 01: Read VGT_STRMOUT_BUFFER_FILLED_SIZE - 10: from SRC_ADDRESS - 11: None [0] - Update_Memory: Store BufferFilledSize to memory - 0: Don"t update memory; DST_ADDRESS_LO/HI are don"t care, but must be provided - 1: Update memory at DST_ADDRESS with VGT_STRMOUT_BUFFER_FILLED_SIZE
3	DST_ADDRESS_LO	[31:2] - Lower bits of DWord-Aligned Destination Address [31:2]. Valid only if Update_Memory is "1". [1:0] - Swap [1:0] function used for data write.
4	DST_ADDRESS_HI	[15:0] - Upper bits of Destination Address [47:32]. DW valid only if Store BufferFilledSize is 01".
5	BUFFER_OFFSET or	If Source Select = "00", bits[31:0] has the BUFFER_OFFSET[31:0] in DWs to

		write to VGT_STRMOUT_BUFFER_OFFSET. If Source Select = "01", this ordinal is don"t care
	SRC_ADDRESS_LO	If Source Select = "10",- bits [31:2] is "SRC_ADDRESS_LO" (the lower bits of DWord-Aligned Source Address [31:2]).- bits [1:0] - Swap [1:0] function used for data read. DW valid only if Source _Select is "10".
6	SRC_ADDRESS_HI	bits [15:0] - Upper bits of Source Address [47:32] . DW valid only if Source_Select is "10".

2.8.10 SURFACE_SYNC

The SURFACE_SYNC packet will allow the driver to place the surface sync commands as one atomic packet.

SURFACE_SYNC Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	ENGINE COHER_CNTL	[31] - ENGINE: 0=PFP, 1=ME. Perform Surface Synchronization at PFP (so index DMA requests are not sent to VGT until the surface is coherent) or at the ME as done in previous ASICs. [28:0] - COHER_CNTL: See the CP_COHER_CNTL register for the definition.
3	COHER_SIZE	Coherency Surface Size has a granularity of 256 Bytes.
4	COHER_BASE	CP_COHER_BASE[31:0] + virtual memory address [47:8]. This value times 256 is the byte address of the start of the surface to be synchronized (to create the high 32-bits of a 40-bit virtual device address).
5	POLL_INTERVAL	[31:16] - Reserved [15:0] - Poll_Interval[15:0]: Interval to wait between the time an unsuccessful polling result is returned and a new poll is issued. Time between these is 16*Poll_Interval clocks. The minimum value is 0x04. A value less than 0x04 will be forced to 0x04.

2.8.11 WAIT_REG_MEM

The WAIT_REG_MEM packet can be processed by either the CP.PFP or the CP.ME, as indicated by the ENGINE field. Zero was chose for the ME for backward compatibility. The CP.PFP is limited to polling a memory location, where the ME can be programmed to poll either a memory location or a register (indicated by MEM_SPACE). The polled value is then tested against the reference value given in the command packet. The test is qualified by both the specified function and mask. If the test passes, the parsing continues. If it fails, the CP waits for the Wait_Interval * 16 Clocks, then tests the Poll Address again.

Note: The driver should always insert a packet that re-programs the CP_WAIT_REG_MEM_TIMEOUT register to the expected value before submitting the WAIT_REG_MEM packet.

WAIT_REG_MEM Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	ENGINE	[8] - ENGINE:

		- 0=ME , - 1=PFP [7:5] - Reserved
	MEM_SPACE	[4] - MEM_SPACE: - 0=Register, - 1=Memory. If ENGINE == PFP, only Memory is valid.
		[3] - Reserved
	FUNCTION	[2:0] - FUNCTION -000 - Always (Compare Passes). Still does read operation and waits for read results to come back. - 001 - Less Than (<) the Reference Value. - 010 - Less Than or Equal (<=) to the Reference Value. - 011 - Equal (==) to the Reference Value. - 100 - Not Equal (!=) to the Reference Value. - 101 - Greater Than or Equal (>=) to the Reference Value. - 110 - Greater Than (>) the Reference Value. - 111 - Reserved. If ENGINE==PFP, only 101/Greater Than or Equal is valid.
3	POLL_ADDRESS_LO	Lower portion of Address to poll If the address is a memory location then bits [31:2] specify the lower bits of the address and [1:0] specify SWAP used for memory read. If the address is a memory-mapped register, then bits [15:0] is the DWord memory-mapped register address that the CP will read.
4	POLL_ADDRESS_HI	Higher portion Address to poll If the address is a memory location then bits [15:0] specify bits 47:32 of the address. If the address is a memory-mapped register, then this DW is a don"t care.
5	REFERENCE	[31:0] - Reference Value.
6	MASK	[31:0] - Mask for Comparison.
7	POLL_INTERVAL	[15:0] - Poll_Interval: Interval to wait between the time an unsuccessful polling result is returned and a new poll is issued. Time between these is 16*Poll_Interval clocks. The minimum value is 0x04. A value less than 0x04 will be forced to 0x04.

2.9 Misc/Data Transfer Packets

2.9.1 ALLOC_GDS

The packet will allocate a new segment within its corresponding GDS partition. The corresponding partition is determined from the Ring to which the packet is submitted. The microcode will first wait until the active partition count equals zero before continuing. This guarantees that the entire contents of the previous allocated segment have been dumped to memory before allocating the new segment within the current partition. It will also check if the segment size is less than partition size and interrupt if the current segment does not fit into its specified partition.

ALLOC_GDS Packet Description

DW	Field	Description
1	HEADER	Header

2	SEGMENT_SIZE	[15:0] SEGMENT_SIZE – Size of segment to allocate within corresponding partition.

2.9.2 COPY_DATA

The purpose of this packet is to provide a generic and flexible way for the CP to copy data by reading it from any source and writing it to any destination to which it has access. When applicable, it can copy either 32-bits or 64-bits of data. All read and write addresses auto-increment for 64-bit operations. The write to destination phase can stall the CP until the write has completed by setting the write confirm bit. The CP can efficiently copy multiple DWs to and from any combination of memory, registers or GDS.

COPY_DATA Packet Description

DW	Field	Description
1	HEADER	Header.
2	CONTROL	[31:30] ENGINE_SEL 00 = ME 01 = PFP (not supported) 10 = CE 11 = Reserved. [20] WR_CONFIRM 0 = Do not wait for write confirmation. 1 = Wait for confirmation that the write has completed. [16] COUNT_SEL 0 = 32-bits (1 DW) 1 = 64-bits (2 DW) [11:8] DEST_SEL: 0 = Mem-mapped Register 1 = Memory (sync) 2 = TC/L2 3 = GDS 4 = Reserved 5 = Memory (async) [3:0] SRC_SEL: 0 = Mem-mapped Register 1 = Memory 2 = TC/L2 3 = GDS 4 = Reserved 5 = Immediate Data 6 = Atomic Return Data 7 = GDS Atomic Return Data0 8 = GDS Atomic Return Data1
3	SRC_ADDR_LO	[31:0] SRC_ADDR_LO [15:0] Mem-mapped Register [31:2] 32-bits Memory, TC [31:3] 64-bits Memory, TC [15:0] GDS offset from partition base

		[31:0] Immediate Data
4	SRC_ADDR_HI	[31:0] SRC_ADDR_HI [63:32] Memory, TC [63:32] Immediate Data
5	DST_ADDR_LO	[31:0] DST_ADDR_LO [15:0] Mem-mapped Register [31:2] 32-bits Memory, TC [31:3] 64-bits Memory, TC [15:0] GDS offset from partition base
6	DST_ADDR_HI	[31:0] DST_ADDR_HI [63:32] Memory, TC

Support Tables

Source Address Support

SRC_SEL	SRC_ADDR_HI	SRC_ADDR_LO
Engine: ME		
0 = Mem-mapped Register	N/A	Reg address
1 = Memory	Low addr bits	Low addr bits
2 = TC/L2	Low addr bits	Low addr bits
3 = GDS	N/A	GDS offset
4 = Reserved for future use	N/A	N/A
5 = Immediate Data	Data[63:32] if COUNT_SEL=1, else ‘0’.	Data[31:0]
6 = Atomic Return Data	N/A	N/A
7 = GDS Atomic Return Data0	N/A	N/A
8 = GDS Atomic Return Data1	N/A	N/A
Engine: PFP		
0 = Mem-mapped Register	N/A	Reg address
1 = Memory	Low addr bits	Low addr bits
2 = TC/L2	Low addr bits	Low addr bits
3 = GDS	N/A	GDS offset
4 = Reserved for future use	N/A	N/A
5 = Immediate Data	Data[63:32] if COUNT_SEL=1, else ‘0’.	Data[31:0]
6 = Atomic Return Data	N/A	N/A
7 = GDS Atomic Return Data0	N/A	N/A
8 = GDS Atomic Return Data1	N/A	N/A
Engine: CE		
0 = Mem-mapped Register	N/A	Reg address
1 = Memory	Low addr bits	Low addr bits
2 = TC/L2	Low addr bits	Low addr bits
3 = GDS	N/A	GDS offset
4 = Reserved for future use	N/A	N/A
5 = Immediate Data	Data[63:32] if COUNT_SEL=1, else ‘0’.	Data[31:0]
6 = Atomic Return Data	N/A	N/A
7 = GDS Atomic Return Data0	N/A	N/A
8 = GDS Atomic Return Data1	N/A	N/A

Destination Write Confirm and Address Support

DST_SEL	WR_CONFIRM	DST_ADDR_HI	DST_ADDR_LO
Engine: ME			
0 = Mem-mapped Register 1 = Mem (sync -GRBM) 2 = TC/L2 3 = GDS 4 = Reserved for future use 5 = Mem (async - direct)	Reg: do read Mem: wait for wc TC/L2: wait for wc GDS: do read N/A Mem: wait for wc	N/A Low/Hi addr bits Low/Hi addr bits N/A N/A Low addr bits	Reg address Low/Hi addr bits Low/Hi addr bits GDS offset N/A Low addr bits
Engine: PFP			
0 = Mem-mapped Register 1 = Mem (sync -GRBM) 2 = TC/L2 3 = GDS 4 = Reserved for future use 5 = Mem (async - direct)	Reg: do read N/A N/A (EOP Uses this interface) N/A N/A Mem: wait for wc	N/A Low/Hi addr bits Low/Hi addr bits N/A N/A Low addr bits	Reg address Low/Hi addr bits Low/Hi addr bits GDS offset N/A Low addr bits
Engine: CE			
0 = Mem-mapped Register 1 = Mem (sync -GRBM) 2 = TC/L2 3 = GDS 4 = Reserved for future use 5 = Mem (async - direct)	Reg: do read N/A Mem: wait for wc N/A N/A N/A	N/A Low/Hi addr bits Low/Hi addr bits N/A N/A Low addr bits	Reg address Low/Hi addr bits Low/Hi addr bits GDS offset N/A Low addr bits

2.9.3 WRITE_GDS_RAM

This packet writes the embedded immediate data into the GDS starting at the indexed offset from the partition base (see GDS_*_INDEX as described in SET_BASE packet for more details). The corresponding partition is determined from the Shader-_Type bit in the header along with the Ring to which the packet is submitted.

WRITE_GDS_RAM Packet Description

DW	Field	Description
1	WRITE_GDS_RAM	Header with Shader-_Type in bit[1]
2	GDS_INDEX	[15:0] GDS_INDEX - Indexed offset from the start of the segment within the partition to where the immediate data will be written.
3-N	DATA	[31:0] DATA

2.9.4 WRITE_DATA

The purpose of this packet is to provide a generic and flexible way for the CP to write N Dwords of data to any destination to which it has access. As applicable, the writes can be sent from the CE, PFP, ME or DE (Dispatch Engine). The CE (and PFP) are limited to the GRBM and MC as destinations. Optionally, the writes can be "confirmed" before continuing.

WRITE_DATA Packet Description

DW	Field	Description
1	HEADER	Header.
2	CONTROL	[31:30] ENGINE_SEL 0 = ME (Micro Engine) 1 = PFP (Prefetch Parser) 2 = CE (Constant Engine) 3 = DE (Dispatch Engine) [20] WR_CONFIRM 0 = Do not wait for write confirmation. 1 = Wait for confirmation that the write has completed. [16] WR_ONE_ADDR 0 = Increment address. 1 = Do not increment the address. [11:8] DST_SEL 0 = Mem-mapped Register 1 = Memory (sync) 2 = TC/L2 3 = GDS 4 = Reserved 5 = Memory (async)
3	DST_ADDR_LO	[31:0] DST_ADDR_LO [15:0] Mem-mapped Register address, DEST_SEL = '0' [31:2] 32-bits Memory or TC L2, DEST_SEL = '1'or '5' [31:3] 64-bits Memory or TCL2, DEST_SEL = '0' [15:0] GDS offset from partition base, DEST_SEL = '0'
4	DST_ADDR_HI	[31:0] DST_ADDR_HI [63:32] Memory, TC
5-to-N	DATA	[31:0] Data

Engine	WR_CONFIRM	WR_ONE_ADDR	DST_SEL	DST_ADDR_LO	DST_ADDR_HI
ME	Reg: do read	0 or 1	Reg,	Reg address	-
	Mem: wait for wc	0 or 1	Mem sync,	Low addr bits	Hi addr bits
	TC/L2: wait for wc	0 or 1	TC/L2,	Low addr bits	Hi addr bits
	GDS: do read	0 or 1	GDS,	GDS offset	-
	Mem: wait for wc	0 or 1	Mem async	Low addr bits	Hi addr bits
PFP	Reg: do read	0 or 1	Reg,	Reg address	-
	Mem: wait for wc	0 or 1	Mem sync,	Low addr bits	Hi addr bits
CE	Reg: do read	0 or 1	Reg,	Reg address	-

	Mem: wait for wc Mem: wait for wc	0 or 1 0 or 1	Mem sync, Mem async	Low addr bits Low addr bits	Hi addr bits Hi addr bits
DE	Reg: do read Mem: wait for wc TC/L2: wait for wc GDS: do read Mem: wait for wc	0 or 1 0 or 1 0 or 1 0 or 1 0 or 1	Reg, Mem sync, TC/L2, GDS, Mem async	Reg address Low addr bits Low addr bits GDS offset Low addr bits	- Hi addr bits Hi addr bits - Hi addr bits

Notes: "wc": Write Confirm, Ack, or acknowledge from the destination.

2.9.5 NOP

Skip a number of DWords to get to the next packet.

NOP Packet Description

DW	Field	Description
1	HEADER	Header of the packet.
2..n	{DATA_BLOCK}	DATA_BLOCK - This field may consist of a number of DWords, and the content may be anything.