Radeon Evergreen/Northern 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 programming guide first as this guide builds on the information in that one. Much of the information in this guide is relevant to previous ASICs as well and is noted where applicable.

2. Scissors

2.1 Overview

There are three 2D coordinate systems relevant for the scissors. All three coordinate systems have the x-axis pointing right and the y-axis pointing down.

1) **Hardware Screen Coordinates** – This coordinate system is defined by the SC number system definition. This coordinate system is the one in which the SC process is performed. The other coordinate systems are "located" within this coordinate system. The screen and window coordinate system can be offset by a programmable amount in the HW Screen coordinate system to allow a maximum amount of guard band for all legitimate window/screen sizes.
2) **Screen Coordinates** – This coordinate system is typically only relevant when rendering a window into a primary surface (i.e. Front/Back buffer). When rendering to an off-screen buffer, typically window and screen are the same coordinate system. When rendering to a primary surface, this coordinate system is typically defined by the primary surface size and has a range of 0 to 16K-1. The origin (0,0) of screen coordinates is located at 0, 0 or a programmable amount in HW Screen coordinates. The window coordinate system typically is located within the screen coordinate system. The offset between the window coordinates and the screen coordinates is defined by a "Window Offset" register controllable by the driver.
3) **Window Coordinates** – This coordinate system is defined by the output of the viewport transform. Typically, 0,0 represents the upper left corner of the visible window and Xmax, Ymax represents the lower right corner of the visible window. Coordinates may range less than 0 and greater than Xmax,Ymax due to the clipping guard band.

2.2 Scissor Rectangles

There are four scissor rectangles supported by evergreen. The first is termed the SCREEN scissor because it is specified in screen coordinates. The second is the WINDOW scissor because it is specified in window coordinates and it can be (conditionally) offset by the window offset and ranges from 0 to 16K for right and bottom and from 0 to 16K-1 for left and top. The offset will be applied if the state register, PA_SC_WINDOW_SCISSOR_TL.WINDOW_OFFSET_DISABLE, is not set. The third scissor is the GENERIC scissor and it is also specified in window coordinates and it can also be (conditionally) offset by the window offset and ranges from 0 to 16K for right and bottom and from 0 to 16K-1 for left and top. The offset will be applied if the state register, PA_SC_GENERIC_SCISSOR_TL.WINDOW_OFFSET_DISABLE, is not set. The fourth scissor is an array of 16 scissors indexed by the viewport array index and it is called the VIEWPORT scissor and ranges from 0 to 16K for right and bottom and from 0 to 16K-1 from left and top. The viewport array index is generally an output of the Geometry Shader. The VIEWPORT scissor can also be offset (conditionally) by the window offset. The offset will be applied if the state register, PA_SC_VPORT_SCISSOR_*_TL.WINDOW_OFFSET_DISABLE. The VIEWPORT scissor can also be enabled/disabled by the state bit PA_SC_MODE_CNTL.VPORT_SCISSOR_ENABLE.

The evergreen scissor rectangles are specified as an upper left x, y and a lower right x, y value in window coordinates. The scissor will be inclusive on LEFT and TOP and exclusive on RIGHT and BOTTOM (i.e. a scissor definition of UL 10,10 and LR 20,20 will draw row and column 10 and will discard row and column 20).

2.3 Clip Rectangles (Auxiliary Scissor)

There are 4 clip rectangles provided by Evergreen. Unlike the Scissor Rectangle, the clip rectangles can be programmed to discard based on included or excluded from the rectangle and may be programmed to form unions and intersections of the 4 clip rectangles. The determination of inside or outside of the rectangle is identical to that of the scissor rectangle (i.e. LEFT/TOP inclusive, RIGHT/BOTTOM exclusive).

3. Compute Shaders

3.1 Overview

Setting up the GPU for compute shaders is very similar to the setup for 3D graphics. When setting compute state, bit 1 in the PM4 packet 3 header needs to be set to 1 to denote a compute shader. To kick-off a compute thread, DISPATCH_* packets are used rather than DRAW_* packets.

For R7xx compute shaders are a special version of the ES shader. Compute inputs are constant buffers, vertex buffers, textures (ES resources), and global memory pointed by SX_MEMORY_EXPORT_BASE. Outputs can go to global memory pointed by SX_MEMORY_EXPORT_BASE or to the ESGS ring or STRMOUT buffers.

For evergreen/cayman compute shaders are a special CS shader type that runs on resources shared with LS shader. The inputs are constant buffers, vertex buffers, textures (LS resources), global memory pointed by SX_MEMORY_EXPORT_BASE (on evergreen) and SX_SCATTER_EXPORT_BASE (on cayman). The outputs can go to global memory or color buffers CB0 to CB11. CB9..11 do not have full color buffer capabilities and can be only used as RATs (Random Access Targets). RATs have capability to "read" the memory from it, in normal read commands or return a result in atomic command (see all the RAT opcodes with _RET). Since there is no return path from CB to shader, each "read" command also sends the return address (offset within corresponding CB_IMMED<n>_BASE surface). When "read" happens, CB does read from the RT and writes to the IMMED surface, and signals end of the operation like it signals confirmation of a normal write. Shader can wait for this write confirmation and then issue read via TC to this location retrieving the "read" value.

SX memory exports use different instructions than CB RAT and can operate only on DWORDs where CB RATs can also work on bytes and shorts.

3.2 State Requirements for Compute

The GPU requires the same state setup for compute as for graphics, with the following differences:

1. **Packet 3 header**
   Set bit 1 in the packet 3 header for compute state
2. **CB**
   See the CB programming section below.
3. **SX**
   Program SX_MEMORY_EXPORT_BASE (R7xx/evergreen)/SX_SCATTER_EXPORT_BASE (cayman) if using global memory exports
4. **DB**
   Program DB_RENDER_CONTROL.COLOR_DISABLE = 1
5. **GDS (Global Data Share)**
   If using GDS, program GDS_ORDERED_WAVE_PER_SE.COUNT = 1, and GDS_ADDR_BASE, and GDS_ADDR_SIZE
6. **SQ**
   Program SQ_THREAD_RESOURCE_MGMT_2.NUM_LS_THREADS and SQ_STACK_RESOURCE_MGMT_3.NUM_LS_STACK_ENTRIES to allocate resources for compute shaders. Also program the LSTMP ring registers: SQ_LSTMP_RING_BASE, SQ_LSTMP_RING_SIZE. On multi-SE (shader engine) asics, the SQ rings are per-SE, so they need to be set separately for each SE. On single SE asics, one only needs to program the SQ rings once.
   Programming multi_SE asics:
   GRBM_GFX_INDEX.INSTANCE_INDEX = 0;
   GRBM_GFX_INDEX.SE_INDEX = 0;
   GRBM_GFX_INDEX.INSTANCE_BROADCAST_WRITES = 1;
   GRBM_GFX_INDEX.SE_BROADCAST_WRITES = 0;

   emit GRBM_GFX_INDEX

emit SQ_LSTMP_RING_BASE, SQ_LSTMP_RING_SIZE for SE0

GRBM_GFX_INDEX.SE_INDEX = 1;

emit GRBM_GFX_INDEX
emit SQ_LSTMP_RING_BASE, SQ_LSTMP_RING_SIZE for SE1

GRBM_GFX_INDEX.SE_INDEX = 0;
GRBM_GFX_INDEX.SE_BROADCAST_WRITES = 1;

emit GRBM_GFX_INDEX

7. **LDS (Local Data Store)**
   Program SQ_LDS_RESOURCE_MGMT.NUM_LS_LDS, SQ_LDS_ALLOC.SIZE and SQ_LDS_ALLOC.HS_NUM_WAVES.

8. **VGT**
   Program the following VGT registers as follows:
   VGT_GS_MODE.MODE = GS_OFF;
   VGT_GS_MODE.COMPUTE_MODE = 1;
   VGT_GS_MODE.PARTIAL_THD_AT_EOI = 1;
   VGT_SHADER_STAGES_EN.LS_EN = CS_STAGE_ON;
   VGT_SHADER_STAGES_EN.HS_EN = HS_STAGE_OFF;
   VGT_SHADER_STAGES_EN.ES_EN = ES_STAGE_OFF;
   VGT_SHADER_STAGES_EN.GS_EN = GS_STAGE_OFF;
   VGT_SHADER_STAGES_EN.VS_EN = VS_STAGE_REAL;

9. **PA**
   Program the following PA registers as follows:
   PA_SU_LINE_CNTL.WIDTH = 0;
   PA_SU_SC_MODE_CNTL.CULL_BACK = 1;
   PA_SU_SC_MODE_CNTL.CULL_FRONT = 1;
   PA_SU_SC_MODE_CNTL.FACE = 1;
   PA_SU_SC_MODE_CNTL.POLY_MODE = 0;
   PA_SU_SC_MODE_CNTL.POLYMODE_FRONT_PTYPE = 2;
   PA_SU_SC_MODE_CNTL.POLYMODE_BACK_PTYPE = 2;
   PA_SU_SC_MODE_CNTL.POLY_OFFSET_BACK_ENABLE = 0;
   PA_SU_SC_MODE_CNTL.POLY_OFFSET_FRONT_ENABLE = 0;
   PA_SU_SC_MODE_CNTL.POLY_OFFSET_BACK_ENABLE = 0;
   PA_SU_SC_MODE_CNTL.POLY_OFFSET_PARA_ENABLE = 0;
   PA_SU_SC_MODE_CNTL.VTX_WINDOW_OFFSET_ENABLE = 0;
   PA_SU_SC_MODE_CNTL.PROVOKING_VTX_LAST = 0;
   PA_SU_SC_MODE_CNTL.PERSP_CORR_DIS = 0;
   PA_SU_SC_MODE_CNTL.MULTI_PRIM_IB_ENA = 0;
   PA_SU_POINT_SIZE = 0;
   PA_SU_POINT_MINMAX = 0;

10. **SPI**
    Program the following SPI registers as follows:
    SPI_COMPUTE_INPUT_CNTL.DISABLE_INDEX_PACK = 0;
    SPI_COMPUTE_INPUT_CNTL.TID_IN_GROUP_ENA = 0;
    SPI_COMPUTE_INPUT_CNTL.bits.TGID_ENA = 0;

11. **Dispatch a Compute Thread**

Instances is the number of thread groups, indices is thread group size. Program the following registers:
VGT_PRIMITIVE_TYPE.PRIM_TYPE = DI_PT_POINTLIST;
VGT_COMPUTE_START_X = 0;
VGT_COMPUTE_START_Y = 0;
VGT_COMPUTE_START_Z = 0;
SPI_COMPUTE_NUM_THREAD_X = indices
SPI_COMPUTE_NUM_THREAD_Y = 1
SPI_COMPUTE_NUM_THREAD_Z = 1
VGT_NUM_INDICES = indices
VGT_COMPUTE_THREAD_GROUP_SIZE = indices;

DISPATCH_DIRECT packet:
DW0:Packet 3 header
DW1: instances
DW2: 1
DW3: 1
DW4: VGT_DISPATCH_INITIATOR.COMPUTE_SHADER_EN = 1
See the PM4 section for more on the DISPATCH* packets.

4. Cayman Shader Changes

4.1 Changes from Evergreen to Cayman

This is a brief summary of the ISA changes from Evergreen to Cayman. See the Cayman and Evergreen ISA documents for further details.

- Vfetch instructions issue through TC.
- Scalar ALU slot (ALU.Trans) removed
- New CF_INST_PUSH_WQM, CF_INST_POP_WQM, CF_INST_ELSE_WQM instructions, and removal of WHOLE_QUAD_MODE bit from certain instructions.
- Changes in stack allocation requirements.
- Restrictions on POP_COUNT settings.
- END_OF_PROGRAM bit replaced with CF_INST_END.
- CF_INST_ALU_BREAK and CF_INST_ALU_CONTINUE are removed and replaced with new EXECUTE_MASK_OP modes.
- Added CF_INST_ALU_VALID_PIXEL_MODE.
- Added CF_INST_REACTIVATE and CF_INST_ALU_REACTIVATE_BEFORE.
- The GROUP_SEQ* ALU synchronization instructions were removed.
- New CF_INST_JUMP_ANY.
- Coalesced scatter reads and the option to write data to LDS.
- Coalesced and structured vfetch and the option to write data to LDS.
- Added MOVA_DST.

4.2 ALU.Trans Changes

These evergreen t-slot only ops are implemented in all vector slots.

- MUL_LIT
- FLT_TO_UINT
- INT_TO_FLT
- UINT_TO_FLT

These evergreen t-slot only opcodes become vector ops, with all four slots expecting the arguments on sources a and b. Result is broadcast to all channels.

- MULLO_INT
- MULHI_INT
- MULLO_UINT
- MULHI_UINT

These evergreen t-slot only opcodes become vector ops in the z, y, and x slots.

- EXP_IEEE
- LOG_IEEE/CLAMPED
- RECIP_IEEE/CLAMPED/FF/INT/UINT/_64/CLAMPED_64
- RECIPSQRT_IEEE/CLAMPED/FF/_64/CLAMPED_64
- SQRT_IEEE/_64

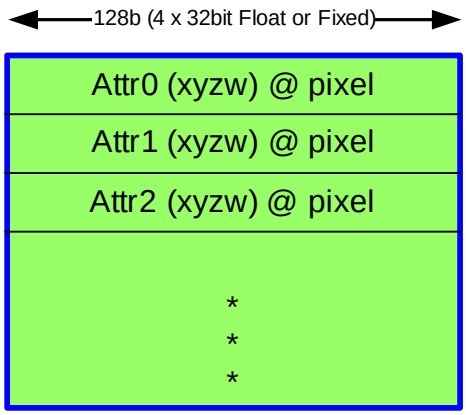
- SIN/COS

The w slot may have an independent co-issued operation, or if the result is required to be in the w slot, the opcode above may be issued in the w slot as well. The compiler must issue the source argument to slots z, y, and x.

5. Unified Interpolation

In R7xx and prior hardware, pixel shader input attribute interpolation (the interpolation of per-vertex attributes to the pixel locations) was done in dedicated interpolation hardware. The controls for this process defined how many attributes were to be interpolated and many unique controls on the method of interpolation for each attribute. The process was initiated once a pixel vector was prepared for the shader core. Prior to any execution of the pixel shader, the interpolation process would compute and write all of the input attributes into the pixel shader general-purpose registers (GPRs).

5.1 R7XX Starting Condition



GPR Pixel N

The R7xx starting condition for the pixel shader is that all of the input attributes are already interpolated to the pixel center (or centroid or a given sample) and are present in the GPRs ready for use. This includes an additional set of data (not from the VS) such as screen-space position, front-face, barycentric parameters, fog terms, and a per-pixel index value.

5.2 Evergreen/Cayman Starting Condition

The Evergreen/Cayman starting condition for the pixel shader is that the GPRs contain the perspective-correct (and/or linear) barycentric coordinates interpolated to the pixel center (and/or centroid and/or sample) along with potentially the same terms still containing w along with 1.0 / w at the pixel center (detail later on how these are used). The pixel shader then has access to the local data store (LDS), which is common storage available to all of the pixels of a given pixel vector. The LDS contains the vertex shader output attribute values as V0, V1-V0, and V2-V0 where V0 is the attribute value at the provoking vertex, V1 is the attribute value at one of the other vertices and V2 is the attribute value at the third vertex. The reason for providing the vertex data in a gradient form (i.e. V0 subtracted from V1 and V2) is that it makes them more applicable to the interpolation math equation (described later).

The terms which are made available to the pixel shader which did not come directly from the vertex shader (such as position in screen space, front-face status, barycentric parameters, per-pixel index value) will still be placed in the GPRs (i.e. NOT in the LDS) at the specified locations, similar to R7xx.

5.2.1 LDS Data

Each pixel vector may use up to 33 attributes, 32 API specified values plus hardware generated ST term (**param gen**). Each pixel vector allocates enough space in the LDS for attribute data ((num_interp + **param_gen**) * prim_count_for_pixel_vector * 12 (the 12 is for 4 components per vector * 3 terms (V0, V1-V0, V2-V0) per gradient), plus any extra LDS space the driver wants to use for the PS (register SQ_LDS_ALLOC_PS). SQ_LDS_ALLOC_PS is in units of 4 dwords to keep the PS params lined up on nice boundaries. Parameter data is written starting at lds_base + SQ_LDS_ALLOC_PS.

Each of the 32 API terms can

- be overridden with **default values** if no semantic match with VS outputs. The SPI will place the appropriate default values in V0 and gradients of 0 in the LDS
- be **flat shaded** if corresponding flat shade bit is set (and global flat shade enable is set). The SPI will place the provoking vertex values in V0 and gradients of 0 in the LDS

The first 20 API terms can

- be overridden as a **point sprite texture**.
- Support **cylindrical wrapping**.

VS FOG can be written to one of the first NUM_INTERP-1 LDS locations. It is enabled by setting PASS_FOG_THROUGH_PS, it uses VS_EXPORTS_FOG and VS_OUT_FOG_VEC_ADDR to know if/where fog is in the param cache, and the result is written to the LDS at FOG_ADDR (range is 0 to NUM_INTERP-1).

The Pixel Shader may reserve some amount of LDS storage for use by the pixel shader (separately from the interpolation attribute storage). This amount is in quantum of 4 DWords to retain the xyzw granularity for the attribute data. The PS storage is in front of (lower address) the attribute data because the number of primitives (which affects the amount of storage required for the attribute data) is variable per pixel shader.

5.2.2 GPR Data

IJ data

- Can write up to 4 GPRs with IJ data.
- Up to 6 sets of IJ for linear/perspective * center/centroid/sample, each IJ takes 2 GPR channels.
- I/W, J/W, 1/W values used for 'true' pull model interpolation, takes up 3 GPR channels.\
- The data will be packed by IJ pairs for the number of IJ pairs enabled in the following priority order
    - Perspective Sample
    - Perspective Center
    - Perspective Centroid
    - Linear Sample
    - Linear Center
    - Linear Centroid

  Followed by an aligned vector with I/W, J/W, 1/W in the x,y,z channels of the subsequent vector.

Front Face, loaded at SPI_PS_IN_CONTROL_1.FRONT_FACE_ADDR.

- X = front face
- Y = prim type
- Z = pixel coverage
- W = gen_index

Pixel Coverage

- SPI stores 32 bit pixel coverage per quad, needs to load 8 bits per pixel to the GPR.
- 8 bit pixel coverage mask loaded into Z channel of Front Face vector.
- Always loaded if Front Face vector is present

Gen Indx Pix

- Loaded into the W channel of Front Face vector.
- Enabled by SPI_PS_IN_CONTROL_1.GEN_INDEX_PIX
- Always loaded if Front Face vector is present

Floating Point Position

- floating point position (pick center/centroid/sample) – X,Y,Z (if PROVIDE_Z), W
- GPR dest address specified by SPI_PS_IN_CONTROL_0.POSITION_ADDR

Fixed Point Position + Misc

- fixed point position (no frac bits). XY = pix XY, Z = rendertarget array index, W = iterated sample num
- GPR dest address is specified by SPI_PS_IN_CONTROL_1.FIXED_PT_POSITION_ADDR.

Line Stipple Texture Coord

- SPI calcs 32 bit line stipple texture coordinate, stores it in the position buffer.
- X = 32b tex coord
- Y = prim type (always loaded)

- GPR dest addr is specified by SPI_PS_IN_CONTROL_2.LINE_STIPPLE_TEX_ADDR.

  The SPI will allocate (MAX(#IJregs-1, FF, POS, FXD, STIP) + 1) GPRs,
  where
  FF = FRONT_FACE_ADDR & FRONT_FACE_ENA,
  POS = POSITION_ADDR & POSITION_ENA,
  FXD = FIXED_PT_POSITION_ADDR & FIXED_PT_POSITION_ENA,
  STIP = LINE_STIPPLE_TEX_ADDR & LINE_STIPPLE_TEX_ENA).

  #IJregs = (((SPI_BARYC_CNTL.PERSP_CENTER_ENA +
  SPI_BARYC_CNTL.PERSP_CENTROID_ENA +
  SPI_BARYC_CNTL.PERSP_SAMPLE_ENA +
  SPI_BARYC_CNTL.LINEAR_CENTER_ENA +
  SPI_BARYC_CNTL. LINEAR_CENTROID_ENA +
  SPI_BARYC_CNTL. LINEAR_SAMPLE_ENA) (add up # IJ pairs)
  + 1) >> 1) (determine #GPRs for
  those pairs)
  + SPI_BARYC_CNTL.PERSP_PULL_MODEL_ENA (add 1 for I/W,J/W,1/W)

Example 1:

All IJ enables
PERSP_PULL_MODEL_ENA (I/W,J/W,1/W)
FRONT_FACE_ENA = 1, FRONT_FACE_ADDR = 4 (PrimType, Coverage Mask, GenIndx are free with presence of FF vector)
POSITION_ENA = 1, POSITION_ADDR = 5, SPI_INPUT_Z = 0 (therefore Z is not valid)
LINE_STIPPLE_TEX_ENA = 1, LINE_STIPPLE_TEX_ADDR = 6 (PrimType is free in STIPPLE vector)

GPR / CHANNEL	X	Y	Z	W
0	I_persp_sample	J_persp_sample	I_persp_center	J_persp_center
1	I_persp_centroid	J_persp_centroid	I_linear_sample	J_linear_sample
2	I_linear_center	J_linear_center	I_linear_centroid	J_linear_centroid
3	I/W	J/W	1/W	
4	FF	PT	MASK	INDX
5	POS.X	POS.Y		POS.W
6	STIP	PT		

Example 2:

IJ persp sample = 1
IJ persp centroid = 1
IJ linear center = 1
PERSP_PULL_MODEL_ENA (I/W,J/W,1/W)
FRONT_FACE_ENA = 1, FRONT_FACE_ADDR = 3 (PrimType, Coverage Mask, GenIndx are free with presence of FF vector)
FIXED_PT_POSITION_ENA = 1, FIXED_PT_POSITION_ADDR = 4

GPR / CHANNEL	X	Y	Z	W
0	I_persp_sample	J_persp_sample	I_persp_centroid	J_persp_centroid
1	I_linear_center	J_linear_center		
2	I/W	J/W	1/W	

3	FF	PT	MASK	INDX
4	FXDPOS.X	FXDPOS.Y	RT ARRAY INDX	ITER SAMPLE

5.3 Mapping from R7xx to Evergreen/Cayman

This is a summary of the changes that will need to happen in order to remap fixed function interpolation from R7xx to the unified interpolation methods of Evergreen. Some of the renderstate controls remain the same as R7xx, a few change meaning slightly, and some are removed and replaced by shader instructions. Since all interpolation (including non-pull-model) now happens in the PS, there must be changes to the driver/compiler as well as to validation test suites and tools to run even the simplest of cases on Evergreen/Cayman.

5.3.1 Renderstate Changes

This section will list all of the R7xx interpolation-related renderstate and denote which ones remain, which ones change and which ones are removed for Evergreen.

Existing Registers

The SPI_VS_OUT_ID_0-9 registers are unchanged from R7xx, they are still used to define the semantic "name" of each of the VS outputs for matching in the PS input list.

The SPI_PS_INPUT_CNTL_*(0-31) have some fields that are no longer applicable:

SEMANTIC ID – same as R7xx. The destination in the LDS for Evergreen for the attribute data will correspond to the relative input value (same as GPR locations from R7xx)
DEFAULT_VAL – same as R7xx (affects the gradient values written into the LDS)
FLAT_SHADE – same as R7xx. (See Global Flat Shade Enable below).
SEL_CENTROID – Removed as this is handled by shader instruction by picking appropriate I,J values
SEL_LINEAR - Removed as this is handled by shader instruction by picking appropriate I,J values
CYL_WRAP – same as R7xx (affects the gradient values written into the LDS)
PT_SPRITE_TEX - as R7xx (affects the gradient values written into the LDS)
SEL_SAMPLE – Removed as this is handled by shader instruction by picking appropriate I,J values.

The SPI_VS_OUT_CONFIG registers are unchanged from R7xx, they are still used to define the form, number, and fog specifics of the VS outputs.

The SPI_PS_IN_CONTROL_0 registers are affected as follows:

NUM_INTERP is now the number of interpolants to process into the LDS (instead of the GPRs).
POSITION_ENA/CENTROID/ADDR/SAMPLE are the same as R7xx as this data is placed into the GPRs still (not LDS) because it does not come from the VS, but is generated by the Scan Converter.
PARAM_GEN/ADDR are different in that now

a) the data is placed into the LDS (instead of GPR) and
b) there is only 1 set of data as the persp/linear and center/centroid/sample is controlled by the appropriate i/j selection in the shader.

BARYC_SAMPLE_CNTL/AT_SAMPLE_ENA are removed from here and redefined in SPI_BARYC_CNTL (see below).
PERSP/LINEAR_GRADIENT_ENA are the same as R7xx.

The SPI_PS_IN_CONTROL_1 registers are affected as follows:

GEN_INDEX_PIX is the same as before, (_ADDR) is removed as the value is placed in FRONT_FACE_ADDR.W
FRONT_FACE_ENA is slightly different in that it used to "override" an interpolant in the GPRs, but now the interpolants go into the LDS and the FF still goes to GPRs, so it no longer overrides anything, it simply is placed in the GPRs @ FRONT_FACE_ADDR.X.
FRONT_FACE_ALL_BITS and FRONT_FACE_ADDR are unchanged.
FOG_ADDR now refers to LDS location instead of GPR.
FIXED_PT_POSITION_*/POSITION_ULC are unchanged.

The SPI_INTERP_CONTROL_0 registers are unchanged from R7xx. The affects of these registers are applied to the attribute gradient placed in the LDS. See below for comments on Flat Shading specifics.
The SPI_INPUT_Z register is unchanged from R7xx.

The SPI_FOG_CNTL registers are generally removed as there are no longer any fog computations performed in the SPI. The PASS_FOG_THROUGH_PS field remains with a new meaning (something like "INPUT_VS_FOG") but retains the same name. Fundamentally, the PASS_FOG_THROUGH_PS bit will place the VS output fog (location specified in SPI_VS_OUT_CONFIG) into the LDS at location SPI_PS_IN_CONTROL_1.FOG_ADDR.

The SPI_FOG_FUNC_SCALE and SPI_FOG_FUNC_BIAS are removed for Evergreen as no fog calculations are performed in the SPI for Evergreen.

NEW REGISTERS For Evergreen/Cayman

The SPI_BARYC_CNTL register is new to Evergreen (although it replaces the old SPI_PS_IN_CONTROL_0.BARYC_SAMPLE_CNTL and BARYC_AT_SAMPLE_ENA). This register is used to define which IJ pairs are provided in the GPRs.

Note that the CENTER and CENTROID "enables" are actually more than one bit. The "2" value setting allows the driver to select "center" to be in the "centroid" location and vice versa. This ability is provided for special AA cases (and potentially other uses) to allow the driver to make centroids be at center or centers be at centroid without having to change the underlying shader code (in other words, "center" can be put in the "centroid" ij location and vice versa).

The SPI_PS_IN_CONTROL_2 register is new to Evergreen. This register is used to enable the provision of a line-stipple value into the GPRs (along with the GPR address) .

6. DB Programming

6.1 Compressed Depth/Stencil Textures (DSTs)

When a DST is being bound to the texture unit, a decompress must happen if the surface is compressed. This can be done either by copying the DST to a color buffer, using the DB to copy and decompress the buffer, or it can be done by having the DB do an in-place decompress. On evergreen, all uncompressed Z formats can be read by the texture path, so the DB to CB copy only needs to be done when converting to a linear tiling mode which the DB doesn't support.

6.1.1 In-place DB decompress

There are two methods for the DB to do a decompress differing in performance depending on the circumstances. In both, the htile buffer and depth buffers should remain attached the same as when drawing.

1) Rasterize all tiles and decompress while rasterizing.
    a) Z_ENABLE=0
    b) STENCIL_ENABLE=0
    c) DEPTH_COMPRESS_DISABLE=1 (only if depth is needed in the texture)
    d) STENCIL_COMPRESS_DISABLE=1 (only if stencil is needed)
    e) DB_RENDER_OVERRIDE.NOOP_CULL_DISABLE=1
    f) DB_RENDER_OVERRIDE.DISABLE_PIXEL_RATE_TILES=1
    g) CB_COLOR_CONTROL.MODE=CB_DISABLE
    h) Draw full screen rectangle
2) Rasterize only the tiles that are not already decompressed, and decompress on flush.
    e) DB_RENDER_OVERRIDE.NOOP_CULL_DISABLE=0

6.1.2 DB Copy + Decompress

1) Rasterize all tiles and decompress while rasterizing.
    a) Set the DB_{Z,STENCIL}_READ_BASE registers to the source and the DB_{Z_STENCIL}_WRITE_BASE registers to the destination.
    b) Z_ENABLE=0
    c) STENCIL_ENABLE=0
    d) [DEPTH|STENCIL]_COMPRESS_DISABLE=1 (For either or both)
    e) DB_RENDER_OVERRIDE.NOOP_CULL_DISABLE=1 (same as in-place decomp)
    f) DB_RENDER_OVERRIDE.DISABLE_PIXEL_RATE_TILES=1
    g) CB_COLOR_CONTROL.MODE=CB_DISABLE
    h) FORCE_[Z|STENCIL]_VALID=1 (makes it read tiles that are already decompressed)
    i) FORCE_[Z|STENCIL]_DIRTY=1 (makes it write all tiles even if already decompressed)
    j) PRESERVE_COMPRESSION=1 (preserves the htile buffer for later use with the compressed buffer)
    k) Draw full screen rectangle

6.1.3 Copy Depth/Stencil to a Color Buffer

If the CB path is chosen, it can be done in-place or to a separate buffer. In place is good if the depth buffer is not used again unless cleared first, while the separate buffer is better if not or if it would overflow to memory.

a) Z_ENABLE=0
b) STENCIL_ENABLE=0
c) DEPTH_COMPRESS_DISABLE=unchanged
d) STENCIL_COMPRESS_DISABLE=unchanged
e) DB_RENDER_CONTROL.DEPTH_COPY=1 (only if needed or if in-place and needs to be rendered to again)
f) DB_RENDER_CONTROL.STENCIL_COPY=1 (same)
g) DB_COPY_CENTROID=1

h) DB_COPY_SAMPLE=0
i) CB_TARGET_MASK=1
j) Attach MRT0 to be the same or separate buffer with a format of COLOR_8_24, COLOR_24_8, COLOR_16, COLOR_32_FLOAT, or COLOR_X24_8_32_FLOAT
k) No blending, fog, etc.
l) CB0_COLOR_INFO.SOURCE_FORMAT=EXPORT_4C_32BPC
m) Draw full screen rectangle

6.1.4 Using a DST again after texturing

If rendering is continued on a DST that was attached to the texture pipe, it must be set up to be used by the DB again. If it was not decompressed in place, then nothing needs to be done. If it was decompressed in place via the DB as described above, then recompressing is not possible for depth, and will happen as stencil is reused anyway, so nothing needs to be done. If it is still in a color tiling format, it must be pulled in through a texture and exported to the DB.

a) Attach the DST to a texture.
b) Create a shader that loads a DST sample and exports Z into oDepth.r and stencil in the 8 LSBs into oDepth.g
c) Shader Compiler should then say to set
    a. SQ_PGM_EXPORTS_PS.EXPORT_MODE = 1 (only depth export and no color exports)
    b. DB_SHADER_CONTROL.Z_EXPORT_ENABLE=1
    c. DB_SHADER_CONTROL.STENCIL_REF_EXPORT_ENABLE=1
d) Z_ENABLE=1
e) Z_FUNC=ALWAYS
f) Z_WRITE_ENABLE=1
g) BACKFACE_ENABLE=0 (or draw a front facing rect)
h) STENCIL_ENABLE=1
i) STENCIL_FUNC=REF_ALWAYS
j) STENCIL_WRITE_MASK=0xFF
k) STENCILZPASS= STENCIL_REPLACE
l) CB_COLOR_CONTROL.MODE=CB_DISABLE
m) Draw full screen rect

6.2 Turning off the Shader for Depth Only Rendering

The DB needs some state validation to accelerate Z/Stencil only rendering or null pixel shaders for when no color buffers are being written. R6xx automatically did the state validation to figure out of any Color was going to be written and could shut off the pixel shader if all of color, depth and stencil did not need shader output. R7xx-evergreen needs a hint to know when no Color Buffers can be written. The DB continued to factor in if the pixel shader's output affects the depth surfaces. Cayman adds back in state validation similar to R6xx which can be disabled via DB_RENDER_OVERRIDE2.DISABLE_COLOR_ON_VALIDATION to return to R7xx-evergreen behavior or for explicit control via CB_COLOR_CONTROL.MODE.

7. Hierarchical Z (HiZ) and Hierarchical Stencil (HiS)

7.1 Relevant Registers

DB_SHADER_CONTROL
DB_DEPTH_INFO
DB_HTILE_DATA_BASE
DB_HTILE_SURFACE
DB_PREFETCH_LIMIT
DB_PRELOAD_CONTROL
DB_RENDER_CONTROL
DB_RENDER_OVERRIDE
DB_SRESULTS_COMPARE_STATE0
DB_SRESULTS_COMPARE_STATE1
DB_DEPTH_CLEAR
DB_STENCIL_CLEAR

7.2 Driver Hints to the Hardware

DB_SHADER_CONTROL.Z_ORDER is a hint to the hw as to which Z order to use. For most cases, it should be programmed to EARLY_Z_THEN_LATE_Z. It should be determined from the size of the fragment shader. Very short shaders may benefit from LATE_Z, while very long shaders may benefit from RE_Z. EARLY_Z_THEN_LATE_Z and EARLY_Z_THEN_RE_Z will attempt to use EARLY_Z, but if the hw is not able to use EARLY_Z due to the current state, it will use LATE_Z or RE_Z. Note that setting DB_RENDER_OVERRIDE.FORCE_HIZ_ENABLE/FORCE_HIS_ENABLE0/1 to FORCE_OFF disables the overrides and allows HiZ/HiS to be determined by DB_SHADER_CONTROL.

7.3 HTILE buffers

The HTile buffer is a separate surface that holds the meta-data for compression and hierarchical optimizations. An HTile is a 32-bit word that represents the compression and hierarchical information for an 8x8, 4x8, 8x4 or 4x4 region of the screen as specified by HTILE_WIDTH and HTILE_HEIGHT. Each DB has an 8k htile cache (8k htiles, not bytes). 8x8 mode with FULL_CACHE=1 and 4 DBs provides 2 million pixels (8*8 pixels * 4 DBs * 8192 htiles). By contrast 4x4 mode with FULL_CACHE=0 and 4 DBs provides 262144 pixels (4*4 pixels * 4 DBs * 8192 htiles * 1/2 cache). On evergreen/cayman only 8x8 mode is supported.

The following algo is used to determine the htile settings:
tile pipes per DB = number of tile pipes / number of DBs;
max pixels per DB = (DB size in pixels / number of tile pipes) * (tile pipes per DB);
width per DB = (DB width in pixels / number of tile pipes) * (tile pipes per DB);

6xx/7xx:

Max pixels per DB	Width per DB (pixels)	HTILE W x H	FULL_CACHE	LINEAR	PRELOAD	PREFETCH W x H	PRELOAD_WINDOW
<=64k	---	4x4	0	1	1	0x0	0
<=128k	---	4x4	1	1	1	0x0	0
<=256k	---	8x4	1	1	1	0x0	0
<=512k	---	8x8	1	1	1	0x0	0
>512k	<=512	8x8	1	0	1	16x4	1
>512k	<=1024	8x8	1	0	1	16x2	1
>512k	>1024	8x8	1	0	1	16x0	1

Evergreen/Cayman:

Max pixels per DB	Width per DB (pixels)	HTILE W x H	FULL_CACHE	LINEAR	PRELOAD	PREFETCH W x H	PRELOAD_WINDOW
<=256k	---	8x8	1	1	1	0x0	0
<=512k	---	8x8	1	1	1	0x0	0
>512k	<=512	8x8	1	0	1	16x4	1
>512k	<=1024	8x8	1	0	1	16x2	1
>512k	>1024	8x8	1	0	1	16x0	1

7.4 HiZ

HiZ requires an htile buffer and DB_DEPTH_INFO.TILE_SURFACE_ENABLE=1 (DB_Z_INFO.TILE_SURFACE_ENABLE=1 on evergreen+). Unless overridden in DB_RENDER_OVERRIDE, HiZ will be used by the hardware whenever possible.

7.5 HiS

HiS requires an htile buffer and DB_DEPTH_INFO.TILE_SURFACE_ENABLE=1 (DB_Z_INFO.TILE_SURFACE_ENABLE=1 on evergreen/cayman). Unless overridden in DB_RENDER_OVERRIDE, the hardware will perform HiS testing based on the enabled sets of HiS state as determined by DB_SRESULTS_COMPARE_STATE0 and DB_SRESULTS_COMPARE_STATE1. There are two sets so the driver can utilize one while updating the other as the stencil state changes. A full screen blit must be done before changing DB_SRESULTS_COMPARE_STATE* once the stencil buffer is bound.

8. CB Programming

The CB expects the driver to validate state and will expect the driver to catch certain invalid configurations. In many cases, if invalid state is programmed the CB will not hang, but the results are otherwise undefined.

8.1 Normal operation

8.1.1 Register state

Under normal operation (CB_COLOR_CONTROL.MODE = CB_NORMAL, not in dual-source blending or multiwrite mode) the following conditions apply:

- CB_SHADER_MASK must be programmed consistently with the actual shader outputs. If N exports are enabled from the shader, then N fields in CB_SHADER_MASK should have at least one bit set.

- The following additional registers must be programmed:
  - CB_COLOR_CONTROL
  - CB_SHADER_MASK
  - CB_TARGET_MASK.

- If the blend constant is used by any MRT, CB_BLEND_{RED,GREEN,BLUE,ALPHA} must also be programmed. The blend constant is in use if, for any MRT,
  1. The blender for the MRT is enabled and the surface is blendable,
  2. The blend equation specifies CONSTANT for any blend factor.

- If an MRT is not disabled, either with CB_SHADER_MASK or by setting CB_COLOR*_INFO.FORMAT = COLOR_INVALID, then the following registers must be programmed for the MRT:
  - CB_COLOR*_INFO
  - CB_COLOR*_ATTRIB
  - CB_BLEND*_CONTROL
  - CB_COLOR*_BASE
  - CB_COLOR*_SIZE
  - CB_COLOR*_VIEW
  - CB_COLOR*_CMASK (if compression is enabled)
  - CB_COLOR*_FMASK (if compression is enabled)
  - CB_COLOR*_CMASK_SLICE (if compression is enabled).
  - CB_COLOR*_FMASK_SLICE (if compression is enabled).

The following registers must be also be configured:

- GB_ADDR_CONFIG
- PA_SC_AA_CONFIG (evergreen)
- PA_SC_MULTI_CHIP_CNTL
- CP_VMID (cayman)
- CP_RINGID (cayman)

8.1.2 Prohibited combinations

The following combinations of configuration are prohibited in CB:

- Special rop3 modes cannot be used when *any* MRT is using the blender. If any MRT is using the blender, ROP3 must be set to the value 0xCC.
- AA surfaces cannot use 1D tiling modes (resolve target can though).
- MRTs that have COLOR<mrt>_INFO.ARRAY_MODE == ARRAY_LINEAR_GENERAL must use the COLOR<mrt>_INFO.ENDIAN value ENDIAN_NONE. (cayman)

8.1.3 Uncompressed MSAA surfaces

MSAA surfaces may be either compressed or uncompressed. Typical usage is to create a compressed MSAA surface, however some applications may wish to render directly to an uncompressed color surface. To do so, set CB_COLOR*_INFO.COMPRESSION = 0. In this mode, only a color surface needs to be allocated; cmask and fmask surfaces are completely ignored. Uncompressed MSAA surfaces can be rendered to normally, and can even be resolved with CB_RESOLVE; however, an uncompressed surface cannot be decompressed with CB_DECOMPRESS.

8.2 Dual-source mode

Dual-source mode is enabled if SRC1 appears in the blend equation for MRT0 (MRT0 must enable its blender and be a blend-capable surface, the blend equation must specify addition or subtraction as the operator, and either a color or alpha blend factor must refer to SRC1). When in dual-source mode, the following restrictions apply:

1. Define MRT0 settings.
2. CB_SHADER_MASK.OUTPUT0_ENABLE should be programmed based on the components that are exported for src0. CB_SHADER_MASK.OUTPUT1_ENABLE is not checked and should be 0. All other fields in CB_SHADER_MASK should be zero, or bound to a RAT. Note that MRT1 is disabled in this mode (including RATs).
3. COLOR1_INFO.SOURCE_FORMAT must be programmed to match COLOR0_INFO.SOURCE_FORMAT so that the 2 quads come in the same format. (New in Evergreen).
4. Enable blending for MRT0; set CB_BLEND<mrt>_CONTROL.ENABLE = 1.
5. Configure a blend-capable format for MRT0.
6. Configure CB_BLEND0_CONTROL to an equation where SRC1 appears at least once in either the color or alpha blend equation

8.3 Fast Clear

- Fast clear requires a cmask surface.
- Fast clear does not require a fmask surface, but the fmask surface must point to the color surface when a fmask surface does not exist. In particular, make sure to set CB_COLOR*_FMASK.BASE_256B = CB_COLOR*_BASE.BASE_256B and CB_COLOR*_MASK.FMASK_TILE_MAX = CB_COLOR*_SLICE.TILE_MAX, and CB_COLOR*_ATTRIB.FMASK_BANK_HEIGHT = CB_COLOR*_ATTRIB.BANK_HEIGHT.
- Fast clear is supported with point-sampled or compressed multisample surfaces. Fast clear with an uncompressed multisample surface is not supported.
- Fast clear is not supported for linear surfaces (array mode == LINEAR_ALIGNED or LINEAR_GENERAL).

- An eliminate fast clear operation must be done on the surface before another block can read it if some of the pixels in the surface have not been covered by drawing (see below).

To fast clear a surface, do the following:

1. Flush the cmask cache.
2. Clear the cmask surface to all zeroes. The color cache may be used to do this.
3. Wait for idle.

To use a fast cleared surface, do the following:

1. Initialized the cmask and fmask surface registers.
2. Set CB_COLOR<mrt>_INFO.FAST_CLEAR = 1.
3. Set CB_COLOR<mrt_CLEAR_WORD* to the clear color.
4. If CB_COLOR<mrt>_INFO.FORMAT == X24_8_32, the X24 component must be masked in the clear_word registers. This means CB_COLOR<mrt>_CLEAR_WORD1 &= 0x000000FF. (evergreen)

8.4 Eliminate Fast Clear/Decompress/ Fmask Decompress

8.4.1 Eliminate Fast Clear

This eliminates the fast cleared tiles from the color surface and fills them in with the clear color. This allows the surface to be read by other clients like textures.

8.4.2 Decompress

This decompresses the multisample surface so that it may be read without the Cmask or Fmask surfaces. The Cmask and Fmask surfaces will be updated to reflect a decompressed multisample surface, so it is possible to continue rendering with compression enabled after a decompress operation. Fast cleared tiles will be eliminated automatically, so an eliminate fast clear pass before this is unnecessary.

It is illegal to decompress a surface that does not have compression enabled. Decompress must be done with 8x8 pixel tile granularities.

8.4.3 Fmask Decompress

This decompresses the fmask surface so that it may be read by other clients like textures. Fast cleared tiles will be eliminated automatically, so an additional eliminate fast clear pass before the shader reads the compressed AA surface is unnecessary.

Rendering with AA compression can be done even after the fmask has been decompressed.

8.4.4 Common programming procedure

1. Flush the color cache if this is decompress. Flush the fmask cache if this is fmask decompress.
2. Fully enable MRT0 in CB_SHADER_MASK (OUTPUT0_ENABLE is 0xF). All other MRTs must be disabled.
3. Set MRT0 to be the multi-sampled surface.
4. Disable blending.
5. Disable ROP3.
6. Set CB_TARGET_MASK.TARGET0_ENABLE = 0xF.
7. Set CB_COLOR_CONTROL.MODE = {CB_DECOMPRESS, CB_ELIMINATE_FAST_CLEAR, CB_FMASK_DECOMPRESS}.
8. Draw over the regions that should be modified. Usually, this will just be a large rectangle the size of the surface. Do not draw the same pixel twice.
9. Flush the color cache for all the modes except when this is a fmask decompress without fast clear. Also, flush the fmask cache if this is fmask decompress.

Note that the rtindex clamping feature is not allowed.

8.5 Resolve

A single multi-sampled surface may be resolved into a point-sampled surface. The point-sampled surface must not be fast cleared. The multi-sampled surface is bound as MRT0, and the point-sampled surface is bound as MRT1. MRT1 must have the same format, number type, component swap and endianness as MRT0; the format must be a blend-capable format. The surfaces may have different tiling modes, but neither surface can use ARRAY_LINEAR_GENERAL or ARRAY_LINEAR_ALIGNED tiling. Fast cleared tiles will be eliminated automatically, so an eliminate fast clear pass before this is unnecessary. To resolve the multi-sampled surface, do the following:

1. Flush and invalidate color cache.
2. Fully enable MRT0 in CB_SHADER_MASK (OUTPUT0_ENABLE is 0xF). All other MRTs must be disabled.
3. Set MRT0 to be the multi-sampled surface. MRT0 must be in a blend-capable format.
4. Set the base address of MRT1 to the resolve destination, but do not enable it in CB_SHADER_MASK. Configure other address-related registers for MRT1, including the tiling in CB_COLOR1_INFO. The following settings for MRT1 should agree with MRT0:
    - CB_COLOR1_INFO.ENDIAN
    - CB_COLOR1_INFO.FORMAT
    - CB_COLOR1_INFO.NUMBER_TYPE
    - CB_COLOR1_INFO.COMP_SWAP
    - Any field that is deterministic based on the surface format.
5. Disable Blending.
6. Disable ROP3.
7. Set CB_TARGET_MASK.TARGET0_ENABLE = 0xF.
8. Set CB_COLOR_CONTROL.MODE = CB_RESOLVE.
9. Draw over the regions that should be resolved. Usually, this will just be a large rectangle the size of the buffer. Do not draw the same pixel twice.
10. Flush and invalidate color cache.

Note that the rtindex clamping feature is not allowed in resolve mode.

8.6 Compute Shader

Compute shaders can perform atomic writes (“device reduction operations”) to memory via the CB. The order of execution of the operations is not guaranteed, only that they are atomic. These writes can include simple operations (min, max, add, and, or, exchange, compare-exchange) and can optionally return a value (pre-op) back to the shader.

The CF_export adds two new opcodes for RAT exports: EXPORT_RAT and EXPORT_RAT_CACHELESS.

If CB_COLOR<mrt>_INFO.RAT is programmed, the surface is treated as a Random Access Target and can only be drawn by Compute Shader operations. A set of MRTs can be configured for RATs and normal rendering. The only stipulation is that all RAT MRTs must be assigned to higher number MRTs than normal rendering MRTs.

These are the major additional rules of the state of a RAT:

1. CB_COLOR<mrt>_ATTRIB must be programmed in accordance with the surface tile format.
2. CB_COLOR<mrt>_INFO.SOURCE_FORMAT must be 4c_32bpc (value of 0).
3. If CB_COLOR<mrt>_INFO.RESOURCE_TYPE is 1D or BUFFER, (or STRUCTUREDBUFFER on cayman)
    a. CB_COLOR<mrt>_ATTRIB.NON_DISP_TILING_ORDER must be 1

   b. CB_COLOR<mrt>_INFO.ARRAY_MODE must be ARRAY_LINEAR_ALIGNED
4. CB_COLOR<mrt>._INFO.FAST_CLEAR is not permitted for RATs.
5. Set CB_IMMED<mrt>_BASE
6. Normally, CB_COLOR<mrt>_INFO.FORMAT must be COLOR_32. Also, the NUMBER_TYPE must be NUMBER_UINT. For all other formats and number types, the only ops supported by Compute Shader quads are STORE and NOP.
7. For immediate ops, the offset specified in the green channel of each scatter export in the quad must be unique.

The driver allocates a region of video memory where atomic operations return data. This acts as a mailbox. The driver programs CB_IMMED<mrt>_BASE with the base address of the return-value memory. The shader export instructions then include the return address offset (per pixel) as part of the address export. The CB performs the atomic operation and also writes back the pre-op value to the return address specified. The shader must use write-with-acknowledge with these operations to know when the return data has been written to the return buffer. The driver should set up a vertex buffer constant to point to this return-value memory for reads. CB_IMMED<mrt>_BASE must be programmed uniquely for each shader engine on mulit-SE asics.

9. PM4

9.1 Introduction

When programming in the PM4 mode, the driver does not write directly to the GPU registers to carry out drawing operations on the screen. Instead, it prepares data in t he 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.

9.1.1 Type-0 Packet

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. This packet supports a register memory map up to 64K DWords (256K Bytes).

Type-0 Packet Description

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.

9.1.2 Type-1 Packet (not-supported)

Type-1 packets are not supported by any R6xx+ families.

9.1.3 Type-2 Packet

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. 29:00 Reserved; This field is undefined, and is set to zero by default. 31:30 TYPE; Packet identifier. It should be 2.
2-N	Body	bit [31:0] Body; The information body "IT_BODY" will be described extensively in the following sections

9.1.4 Type-3 Packet

Carry out the operation indicated by field IT_OPCODE.

Type-3 Packet Description

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

9.2 Initialization Packets

9.2.1 ME_INITIALIZE - R7xx/Evergreen/Cayman

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 (i.e., Cayman), 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	bit [0] - Default Reset Control; Resets specific areas of the Scratch memory to known values and Resets the Current and Last contexts
3	Reserved	bit [31:0] Reserved; Program to zero.
4	MAX_CONTEXT	bit [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	bit [31:24] - Reserved bit [23:16] - Device-ID bit [15:2] - Reserved bit [1:0] - Swap Code Used for the following transactions: Load_* , Set_* , PM4 headers
6	Header_Dump_Base Header_Dump_Swap	bit [31:4] - Header_Dump_Base : a 4 Kbyte aligned address, i.e. base memory address [39:12] of the external memory location where CP will

		dump PM4 Headers. bit [3:2] - Reserved: should be set to zero. bit [1:0] - Header_Dump_Swap: the 2 bit Swap Code used when writing headers to memory.
7	Header_Dump_Enable Header_Dump_Size	bit [31] - Header_Dump_Enable: Enable Writing PM4 Headers to Memory for Debug bit [30] - Reserved. bit [29:0] - Header_Dump_Size: Size in DWords for the Header Dump Ring in External Memory.

9.2.2 PREAMBLE_CNTL - R7xx/Evergreen/Cayman

This packet has two purposes: (1) indicate the packets that belong to each preamble in a command buffer, so can record the location of the last preamble. If there is a context switch, the CP could then fetch the last completed preamble to reload the GPU state when the process was preempted. The CP will then skip to the location in the command buffer where it left off when the process was switched out. There can be multiple preambles in an IB to initialize the current state, so the CP must update its internal values whenever the packets are received. Another purpose is to (2) indicate to the CP the packets that program the Clear State, versus other Set_* packets that go to the next available state.
The Driver will send the PREAMBLE_CNTL packets before and after the Clear State programming. Initializing the Clear State is only supported immediately after the ME_INITIALIZE packet. The Begin marker must always come first and the End marker must be of the same type as the previous "Begin". The driver initializes the internal GPU context with the help of the Preamble_Cntrl packet by sending the "clear state" between two instances of this packet, one representing the begin marker and the second the end marker. Any register is allowed to be programmed within the begin and end markers, however, only multi-state GFXDEC registers will be programmed automatically by the CLEAR_STATE packet. Config and Constant updates must be handled by the driver. See CLEAR_STATE packet for special provision for very fast clearing of all constants.

PREAMBLE_CONTROL Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet.
2	CMD	Command bit [19:0] Reserved; Reserved for internal use by the CP bit [27:20] Reserved; bit [31:28] Command; - 0000 : Begin Preamble, domain records the current IB offset for later use - 0001 : End, domain records the current IB offset for later use - 0010 : Begin of Clear State initialization [Evergreen+] - 0011 : End of Clear State initialization [Evergreen+] - 1xxx, x1xx : Reserved

9.3 Command Buffer Packets

9.3.1 INDIRECT_BUFFER - R7xx/Evergreen/Cayman

This packet is used for dispatching Indirect Buffers.

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	[7:0] - Upper bits of Address [39:32]
4	VMID	[31:24] - VMID[7:0], Virtual Memory Domain ID for the command buffer. This field is valid starting with cayman
	IB_SIZE	[19:0] - Indirect Buffer Size [19:0], size of the Indirect Buffer in DWORDs.

9.3.2 DRAW_INDEX - R7xx/Evergreen/Cayman

DRAW_INDEX draws a set of primitives using fetched indices. The SOURCE_SELECT field in the DRAW_INITIATOR indicates that the VGT will DMA the indices.

DRAW_INDEX 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). Written to the VGT_DMA_BASE register (No Context Supplied).
3	INDEX_BASE_HI	[7:0] - Base Address Hi [39:32] of Index Buffer. Written to the VGT_DMA_BASE_HI register (No Context Supplied).
4	INDEX_COUNT	[31:0] - 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.

9.3.3 DRAW_INDEX_2 - R7xx/Evergreen/Cayman

Draws a set of primitives using fetched indices from a bounded index buffer. The SOURCE_SELECT field in the DRAW_INITIATOR indicates that the VGT will DMA the indices.

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	[30:0] - Base Address [31:1] of Index Buffer (Word-Aligned). Written to the

		VGT_DMA_BASE register (No Context Supplied).
4	INDEX_BASE_HI	[7:0] - Base Address Hi [39:32] 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.

9.3.4 DRAW_INDEX_AUTO - R7xx/Evergreen/Cayman

Draws a set of primitives using indices auto-generated by the VGT. The SOURCE_SELECT field in the DRAW_INITIATOR indicates that the VGT need to auto-generate the indices.

DRAW_INDEX_AUTO Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	INDEX_COUNT	INDEX_COUNT [31:0] - Number of indices to generate. 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.

9.3.5 DRAW_INDEX_IMMED - R7xx/Evergreen/Cayman

Draws a set of primitives using indices in the packet. The SOURCE_SELECT field in the DRAW_INITIATOR indicates that the VGT will use immediate data for the indices. This packet is generally used for draw packets with a small number" of indices. It is faster for the driver to just put the indices into the command buffer instead of copying them to a separate index buffer.

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 INDEX_TYPE packet for details on how to specify the 16 or 32-bit indices.

9.3.6 DRAW_INDEX_OFFSET - R7xx/Evergreen/Cayman

The purpose of this feature is to reduce the amount of addresses the driver must patch in the IB to only the first index buffer call instead of every draw that uses that index buffer. 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.

Draws a set of primitives using fetched indices with no patching required. The CP will shift the

INDEX_OFFSET by one or two bits depending on the value in INDEX_TYPE and then add that offset to the Base Address previously supplied in the INDEX_BASE packet.

The functionality is implemented using one current packet, INDEX_TYPE, and two new packets Draw/Dispatch Packets DRAW_INDEX_OFFSET and INDEX_BASE. The driver sends the INDEX_TYPE and INDEX_BASE packets before the DRAW_INDEX_OFFSET packet.

DRAW_INDEX_OFFSET Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	INDEX_OFFSET	Starting index number in the index buffer. INDEX_OFFSET of zero represents the first index, index one is the second index.
3	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.
4	DRAW_INITIATOR	Draw Initiator Register. Written to the VGT_DRAW_INITIATOR register for the assigned context.

9.3.7 DRAW_INDEX_OFFSET_2 - R7xx/Evergreen/Cayman

ASICs Supported:

Family	Supported Members
6xx	All, except R600, which does not have the VGT_DMA_MAX_SIZE register.
7xx	All
Evergreen+	All

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.

The CP will shift the INDEX_OFFSET by one or two bits depending on the value in INDEX_TYPE and then add that offset to the Base Address previously supplied in the INDEX_BASE packet.

The functionality is implemented using one current packet, INDEX_TYPE, and two new packets Draw/Dispatch Packets DRAW_INDEX_OFFSET and INDEX_BASE. The driver sends the INDEX_TYPE and INDEX_BASE packets before the DRAW_INDEX_OFFSET packet.

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.

9.3.8 INDEX_BASE - R7xx/Evergreen/Cayman

The purpose of the INDEX_BASE packet, in conjunction with the INDEX_TYPE Packet and DRAW_INDEX_OFFSET packets, is to minimize the amount of address patching that the driver must do. The driver only needs to send the INDEX_BASE (and therefore patch the indirect buffer) once when the index buffer call is made. Subsequence calls to draw with an index buffer offset (and resulting DRAW_INDEX_OFFSET packet) need no patching.

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	Bits [7:0] Base Address Hi [39:32] of Index Buffer.

9.3.9 INDEX_TYPE - R7xx/Evergreen/Cayman

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.

9.3.10 NUM_INSTANCES - R7xx/Evergreen/Cayman

NUM_INSTANCES is used to specify the number of instances for the subsequence draw command. This packet is considered part of the draw packet sequence. 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.

9.3.11 MPEG_INDEX - R7xx/Evergreen/Cayman

MPEG_INDEX: Packed register writes for MPEG and Generation of Indices. The VGT_PRIMITIVE_TYPE:PRIM_TYPE should be a DI_PT_RECTLIST.

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.

9.3.12 DISPATCH_DIRECT - Evergreen/Cayman

Used for dispatching a compute thread with the array dimensions 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	Bits [31:0] + x dimensions of the array of thread groups to be dispatched
3	DIM_Y	Bits [31:0] + y dimensions of the array of thread groups to be dispatched
4	DIM_Z	Bits [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.

9.3.13 DISPATCH_INDIRECT - Evergreen/Cayman

Used for dispatching a compute thread with the array dimensions 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	Bits [31:0] + Byte aligned offset where the required data structure starts.Bits 1:0 = 00.
3	DISPATCH_INITIATOR	Dispatch Initiator Register. Written to the VGT_DISPATCH_INITIATOR register for the assigned context.

9.4 State Management Packets

9.4.1 CLEAR_STATE - Evergreen/Cayman

The purpose of the Clear_State packet is to reduce command buffer preamble setup time for all driver versions. The definition of Clear State is essentially everything off, resources all NULL, other value set to the API defined default state, nulling of constant buffers and constants programmed via the GRBM (Tex_Resource, Tex_Samplers, Boolean, Loop, Ctl). Clear State includes all multi-copy state (Gfx Decode) and Constants nulling registers. It does not include single copy configuration registers. The Constants are cleared via five new context registers: SQ_TEX_SAMPLER_CLEAR, SQ_TEX_RESOURCE_CLEAR, SQ_LOOP_BOOL_CLEAR, SQ_VTX_BASE_VTX_LOC, SQ_VTX_START_INST_LOC. ALU constant buffers are managed in the GFXDEC (8-state) space so they are cleared by the normal clear by clearing the SQ_PGM_* , and SQ_ALU_CONST_BUFFER_SIZE_* registers.

CLEAR_STATE 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	DUMMY	Dummy Data

9.4.2 DEALLOC_STATE - Cayman

The purpose of the DEALLOC_STATE packet is to free an allocated compute shader state back to the state-set pool. Shader type should always be '1', i.e., compute shader.

- It will be placed in the Ring Buffer by the kernel driver.
- The driver will guarantee that no computer shaders are active in the GPU when this packet is received.
- There may or may not have been any Dispatch packets between the CLEAR_STATE packet that allocated the CS state and this packet that deallocates (frees it).
- Though there will be a write to the GFX_COPY_STATE register sometime after the context done event, a significant number of other packets could be processed between them.

DEALLOC_STATE Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet. Shader_Type in bit 1 of the Header will be '1', i.e., Computer Shader, see the Type-3 Packet section.
2	DUMMY	Dummy Data

9.4.3 MODE_CONTROL - Evergreen/Cayman

This generic packet is used to reset the Graphics chip out of DX9 ALU Constant Emulation mode back to DX10 Constant Buffer mode.

PREAMBLE_CONTROL Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet.
2	CMD	Command [31:3] - Reserved [2:0] - CMD

		000: Reserved 001: Reset DX9 Constant Emulation Mode. That is, switch to DX10 style constant buffer mode.

9.4.4 CONTEXT_CONTROL - R7xx/Evergreen/Cayman

The CONTEXT_CONTROL packet controls the processing of LOAD packets and shadowing for SET packets. 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.

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 Bits 30:13 - Not used Bit 12 - Enable Load CS Samplers Bit 11 - Enable Load CS Resources Bit 10 - Enable Load CS Loop Constants Bit 9 - Enable Load CS Boolean Constants Bit 8 - Enable Load CS Multi-Context Render State Registers Bit 7 - Enable Load Control Constants Bit 6 - Enable Load Samplers Bit 5 - Enable Load Resources Bit 4 - Enable Load Loop Constants Bit 3 - Enable Load Boolean Constants Bit 2 - Enable Load ALU Constants (R7xx) - Reserved (Evergreen+) Bit 1 - Enable Load Multi-Context Render State Registers Bit 0 - Enable Load Single-Context Configuration Registers
3	SHADOW_ENABLE	Bit 31 - DW Enable - Shadow Enables will not be updated unless this bit is set. Bits 30:13 - Not used Bit 12 - Enable Shadowing of CS Samplers Bit 11 - Enable Shadowing of CS Resources Bit 10 - Enable Shadowing of CS Loop Constants Bit 9 - Enable Shadowing of CS Boolean Constants Bit 8 - Enable Shadowing of CS Multi-Context Render State Registers Bit 7 - Enable Shadowing of Control Constants Bit 6 - Enable Shadowing of Samplers Bit 5 - Enable Shadowing of Resources Bit 4 - Enable Shadowing of Loop Constants Bit 3 - Enable Shadowing of Boolean Constants Bit 2 - Enable Shadowing of ALU Constants (R7xx) - Reserved (Evergreen+) Bit 1 - Enable Shadowing of Multi-Context Render State Registers Bit 0 - Enable Shadowing of Single-Context Configuration Registers

9.4.5 LOAD_LOOP_CONST - R7xx/Evergreen/Cayman

This packet provides the ability to have the CP:

- Initialize the LOOP_CONST_BASE (BASE_ADDR_* fields) internally for later use when a SET_LOOP_CONST 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, CONST_OFFSET and NUM_DWordS are programmed to zero.
- Fetch ALU constant 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[39:2] = LOOP_CONST_BASE[39:2] + CONST_OFFSET

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

- Reg_Start_Address[17:2] = SQ_LOOP_COUNT_CONST_0[17:2] + CONST_OFFSET

LOAD_LOOP_CONST 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] - LOOP_CONST_BASE (31:2) for the block in Memory from where the CP will fetch the constants.
3	BASE_ADDR_HI	[7:0] - LOOP_CONST_BASE (39:32) for the block in Memory from where the CP will fetch the constants.
4	CONST_OFFSET	[31:16] - Reserved [15:0] - CONST_OFFSET[15:0] in DWords from the base alu constant address (SQ_ LOOP _CONSTANT0_0) and base alu constant memory address (LOOP _CONST_BASE).
5	NUM_DWords	[31:14] - Reserved 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	CONST_OFFSET	[31:16] - Reserved CONST_OFFSET[15:0] - Same Definition as Above.
N+1	NUM_DWord	[31:14] - Reserved NUM_DWords[13:0] - Same Definition as Above.

9.4.6 LOAD_ALU_CONST - R7xx/Evergreen/Cayman

The LOAD_ALU_CONST packet is used by the driver to specify the ALU constant buffer base address for the CP to use as the start address of the 8 Constant buffers it manages to emulate DX9 ALU data on HW that only supports constant buffers.

LOAD_ALU_CONST Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	ALU_CONST_BASE	[31:9] - Address [39:17]. 128 K Byte boundary. [8:1] - Reserved. [0] - COMPLETE_UPDATE

		0: CP manages incremental ALU CONST Updates 1: Driver includes a complete set of ALU CONST updates.

9.4.7 LOAD_BOOL_CONST - R7xx/Evergreen/Cayman

This packet provides the ability to have the CP initialize the BOOL_CONST_BASE (BASE_ADDR_* fields) internally for later use when a SET_BOOL_CONST 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, CONST_OFFSET and NUM_DWordS are programmed to zero.

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

- Mem_Start_Address[39:2] = BOOL_CONST_BASE[39:2] + CONST_OFFSET

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

- Reg_Start_Address[17:2] = SQ_BOOL_CONST_0[17:2] + CONST_OFFSET

LOAD_BOOL_CONST 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] - BOOL_CONST_BASE (31:2) for the block in Memory from where the CP will fetch the constants.
3	BASE_ADDR_HI	[7:0] - BOOL_CONST_BASE (39:32) for the block in Memory from where the CP will fetch the constants.
4	CONST_OFFSET	[15:0] - CONST_OFFSET[15:0] in DWords from the base alu constant address (SQ_BOOL_CONSTANT0_0) and base alu constant memory address (BOOL_CONST_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	CONST_OFFSET	[31:16] - Reserved CONST_OFFSET[15:0] - Same Definition as Above.
N+1	NUM_DWordS	[31:14] - Reserved NUM_DWordS[13:0] - Same Definition as Above.

9.4.8 LOAD_CONFIG_REG - R7xx/Evergreen/Cayman

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[39:2] = CONFIG_REG_BASE[39:2] + 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)

LOAD_CONFIG_REG Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet. Shader_Type in bit 1 of the Type-Header will always be zero since these are configuration registers.
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	[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".
	BASE_ADDR_HI	[7:0] - CONFIG_REG_BASE (39:32) 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	[31:14] - Reserved NUM_DWordS[13:0] - Same Definition as Above.

9.4.9 LOAD_CONTEXT_REG - R7xx/Evergreen/Cayman

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[39:2] = CONTEXT_REG_BASE[39:2] + 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)

LOAD_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	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]	[7:0] - CONTEXT_REG_BASE (39:32) 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	[31:14] - Reserved NUM_DWordS[13:0] - Same Definition as Above.

9.4.10 LOAD_CTL_CONST - R7xx/Evergreen/Cayman

This packet provides the ability to have the CP:

- Initialize the CTL_CONST_BASE (BASE_ADDR_* fields) internally for later use when a SET_CTL_CONST 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, CONST_OFFSET and NUM_DWordS are programmed to zero.
- Fetch ALU constant 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[39:2] = CTL_CONST_BASE[39:2] + CONST_OFFSET

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

- Reg_Start_Address[17:2] = mmSQ_VTX_BASE_VTX_LOC[17:2] + CONST_OFFSET

LOAD_CTL_CONST Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet. Shader_Type in bit 1 of the Header will always be zero, since no CS constants are set with this packet, see Type-3 Packet.
2	BASE_ADDR_LO	[31:2] - Lower Base address bits (31:2) for the block in Memory from where the CP will fetch the constants.
3	BASE_ADDR_HI	[7:0] - Upper Base address bits (39:32) for the block in Memory from where the CP will fetch the constants.
4	CONST_OFFSET	CONST_OFFSET[15:0] - Offset in DWords from the base address.
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	CONST_OFFSET	[31:16] - Reserved CONST_OFFSET[15:0] - Same Definition as Above.
N+1	NUM_DWords	[31:14] - Reserved NUM_DWordS[13:0] - Same Definition as Above.

9.4.11 LOAD_RESOURCE - R7xx/Evergreen/Cayman

This packet provides the ability to have the CP:

- Initialize the RESOURCE_CONST_BASE (BASE_ADDR_* fields) internally for later use when a SET_RESOURCE 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, CONST_OFFSET and NUM_DWordS are programmed to zero.
- Fetch ALU constant 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[39:2] = RESOURCE_CONST_BASE[39:2] + CONST_OFFSET

The CP writes the 'loaded' data to consecutive register addresses. The starting address is computed as shown below:
- Reg_Start_Address[17:2] = SQ_TEX_RESOURCE_WORD0_0[17:2] + CONST_OFFSET

LOAD_RESOURCE 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] - RESOURCE_CONST_BASE (31:2) for the block in Memory from where the CP will fetch the constants.
3	BASE_ADDR_HI	[7:0] - RESOURCE_CONST_BASE (39:32) for the block in Memory from where the CP will fetch the constants.
4	CONST_OFFSET	[15:0] - CONST_OFFSET[15:0] in DWords from the base alu constant address (SQ_TEX_RESOURCE_WORD0_0) and base alu constant memory address (RESOURCE_CONST_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	CONST_OFFSET	[31:16] - Reserved CONST_OFFSET[15:0] - Same Definition as Above.
N+1	NUM_DWords	[31:14] - Reserved NUM_DWordS[13:0] - Same Definition as Above.

9.4.12 LOAD_SAMPLER - R7xx/Evergreen/Cayman

This packet provides the ability to have the CP initialize the SAMPLER_CONST_BASE (BASE_ADDR_* fields) internally for later use when a SET_SAMPLER 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, CONST_OFFSET and NUM_DWordS are programmed to zero. The CP also uses this packet to fetch ALU constant 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[39:2] = SAMPLER_CONST_BASE[39:2] + CONST_OFFSET

The CP writes the 'loaded' data to consecutive register addresses. The starting address is computed as shown below:
- Reg_Start_Address[17:2] = SQ_TEX_SAMPLER_WORD0_0[17:2] + CONST_OFFSET

LOAD_SAMPLER 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] - SAMPLER_CONST_BASE (31:2) for the block in Memory from where the CP will fetch the constants.
3	BASE_ADDR_HI	[7:0] - SAMPLER _CONST_BASE (39:32) for the block in Memory from where the

		CP will fetch the constants.
4	CONST_OFFSET	[15:0] - CONST_OFFSET[15:0] in DWords from the base alu constant address (SQ_TEX_SAMPLER_WORD0_0) and base alu constant memory address (SAMPLER_CONST_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	CONST_OFFSET	[31:16] - Reserved CONST_OFFSET[15:0] - Same Definition as Above.
N+1	NUM_DWords	[31:14] - Reserved NUM_DWordS[13:0] - Same Definition as Above.

9.4.13 SET_BOOL_CONST - R7xx/Evergreen/Cayman

The SET_BOOL_CONST packet loads the constant Boolean data, which is embedded in the packet, into the chip. The CONST_OFFSET field is a DWord-offset from the starting address. All the constant data in the packet is written to consecutive register addresses beginning at the starting address. The starting address for

Boolean constants is computed as follows:

- Reg_Start_Address[17:2] = SQ_BOOL_CONST_0[17:2] + CONST_OFFSET

The CP will write the data to external memory if the corresponding shadow enable is set. This allows the Boolean constants to be reloaded into the chip after a context switch with the LOAD_BOOL_CONST (LBC) packet. The LBC packet sets the BOOL_CONST_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 constant data is written to is computed as follows:

- Mem_Start_Address[39:2] = BOOL_CONST_BASE[39:2] + CONST_OFFSET

SET_BOOL_CONST 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	CONST_OFFSET	[31:16] - Must be programmed to zero for legacy support [15:0] - Offset in DWords from the BOOL const base address (DW address of SQ_BOOL_CONSTANT0_0) and memory base address (BOOL_CONST_BASE).
3 to N	CONST_DATA	DWord Data for Constants.

9.4.14 SET_CONFIG_REG - R7xx/Evergreen/Cayman

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 after a context switch with the LOAD_CONFIG_REG (LCFG) packet. The LCFG 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[39:2] = CONFIG_REG_ BASE[39:2] + REG_OFFSET

SET_CONFIG_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	[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.

9.4.15 SET_CONTEXT_REG - R7xx/Evergreen/Cayman

This packet loads the eight-context-renderstate 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 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

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.

9.4.16 SET_CTL_CONST - R7xx/Evergreen/Cayman

This packet loads the constant Control data, which is embedded in the packet, into the chip. The CONST_OFFSET field is a DWord-offset from the starting address. All the constant data in the packet is written to consecutive register addresses beginning at the starting address. The starting address for Control constants is computed as follows:

- Reg_Start_Address[17:2] = mmSQ_VTX_BASE_VTX_LOC[17:2] + CONST_OFFSET

The CP will write the data to external memory if the corresponding shadow enable is set. This allows the Control constants to be reloaded into the chip after a context switch with the LOAD_CTL_CONST (LCC) packet. The LCC packet sets the CONTROL_CONST_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 constant data is written to is computed as follows:

- Mem_Start_Address[39:2] = CONTROL_CONST_BASE[39:2] + CONST_OFFSET

SET_CTL_CONST Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet. Shader_Type in bit 1 of the Header will always be zero, since no CS constants are set with this packet, see Type-3 Packet.
2	CONST_OFFSET	[31:16] - Must be programmed to zero for legacy support [15:0] - Offset in DWords from the CTL const base address (DW address of mmSQ_VTX_BASE_VTX_LOC) and memory base address (CONTROL_CONST_BASE).
3 to N	CONST_DATA	DWord Data for Constants.

9.4.17 SET_LOOP_CONST - R7xx/Evergreen/Cayman

This packet loads the constant Loop data, which is embedded in the packet, into the chip. The CONST_OFFSET field is a DWord-offset from the starting address. All the constant data in the packet is written to consecutive register addresses beginning at the starting address. The starting address for Loop constants is computed as follows:

- Reg_Start_Address[17:2] = SQ_LOOP_COUNT_CONST_0[17:2] + CONST_OFFSET

The CP will write the data to external memory if the corresponding shadow enable is set. This allows the Loop constants to be reloaded into the chip after a context switch with the LOAD_LOOP_CONST (LLC) packet. The LLC packet sets the LOOP_CONST_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 constant data is written to is computed as follows:

- Mem_Start_Address[39:2] = LOOP_CONST_BASE[39:2] + CONST_OFFSET

SET_LOOP_CONST 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	CONST_OFFSET	[31:16] - Must be programmed to zero for legacy support [15:0] - Offset in DWords from the LOOP const base address (DW address of SQ_LOOP_COUNT_CONSTANT0_0) and memory base address (LOOP_CONST_BASE).
3 to N	CONST_DATA	DWord Data for Constants.

9.4.18 SET_RESOURCE - R7xx/Evergreen/Cayman

This packet loads the Resource data, which is embedded in the packet, into the chip. The CONST_OFFSET field is a DWord-offset from the starting address. All the Resource data in the packet is written to consecutive register addresses beginning at the starting address. The starting address for Resources is computed as follows:

- Reg_Start_Address[17:2] = SQ_TEX_RESOURCE_WORD0_0[17:2] + CONST_OFFSET

The CP will write the data to external memory if the corresponding shadow enable is set. This allows the resource data to be reloaded into the chip after a context switch with the LOAD_RESOURCE (LRS) packet. The LRS packet sets the RESOURCE_CONST_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 Resource data is written to is computed as follows:

- Mem_Start_Address[39:2] = RESOURCE_CONST_BASE[39:2] + CONST_OFFSET

SET_RESOURCE 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	CONST_OFFSET	[15:0] - CONST_OFFSET in DWords from the RESOURCE const base address (DW address of SQ_ TEX_RESOURCE_WORD0_0) and memory base address (RESOURCE_CONST_BASE).
3 to N	CONST_DATA	DWord Data for Constants.

9.4.19 SET_SAMPLER - R7xx/Evergreen/Cayman

This packet loads the Sampler data, which is embedded in the packet, into the chip. The CONST_OFFSET field is a DWord-offset from the starting address. All the Sampler data in the packet is written to consecutive register addresses beginning at the starting address. The starting address for Samplers is computed as follows:

- Reg_Start_Address[17:2] = SQ_TEX_SAMPLER_WORD0_0[17:2] + CONST_OFFSET

The CP will write the data to external memory if the corresponding shadow enable is set. This allows the sampler data to be reloaded into the chip after a context switch with the LOAD_SAMPLER (LSP) packet. The LSP packet sets the SAMPLER_CONST_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 sampler data is written to is computed as follows:

- Mem_Start_Address[39:2] = SAMPLER_CONST_BASE[39:2] + CONST_OFFSET

SET_SAMPLER 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	CONST_OFFSET	[15:0] - Offset in DWords from the SAMPLER const base address (DW address of SQ_ TEX_SAMPLER_WORD0_0) and memory base address (SAMPER_CONST_BASE).
3 to N	CONST_DATA	DWord Data for Constants.

9.5 Command Predication Packets

9.5.1 COND_EXEC - R7xx/Evergreen/Cayman

Perform a conditional execution of a sequence of packets (type 0, 2, and type 3) based on a Boolean value stored in memory.

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.PFP will start parsing the DWord immediately following EXEC_COUNT DWords.

COND_EXEC Packet Description

DW	Field Name	Description

1	HEADER	Header of the packet
2	BOOL_ADDR_LO	Bits [31:2] is Boolean Address bits [31:2].
3	BOOL_ADDR_HI	Bits[7:0] Boolean Address bits [39:32].
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.

9.5.2 COND_WRITE - R7xx/Evergreen/Cayman

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	31:9 - Reserved 8 - WRITE_SPACE 0=Register, 1=Memory
	POLL_SPACE	7:5 - Reserved 4: POLL_SPACE 0=Register, 1=Memory
	FUNCTION	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 pollIf the address is a memory location then bits [7:0] specify bits 39:32 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. ElseIf 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 care. If WRITE_SPACE + Memory: Bits 7:0 - WRITE_ADDRESS[39:32]
9	WRITE_DATA	Write Data[31:0] that will be conditionally written to the ADDRESS.

9.5.3 SET_PREDICATION - R7xx/Evergreen/Cayman

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

SET_PREDICATION Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	START_ADDR_LO	Bits [31:4] is start address bits [31:4]. Supports a 16 byte aligned address for DB0 count.
3	CONTINUE	Bit [31] - Continue set predication (Valid only for use with 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.
	PRED_OP	Bit [18:16] Pred_Op -000: Clear Predicate -001: Set Zpass Predicate -010: Set PrimCount Predicate -011: Reserved -1xx: Reserved .
	HINT	Bit [12] Hint (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.
	PREDICATION_BOOLEAN	Bit [8] Predication Boolean (valid for both ops) - 0: Draw if not visible/overflow - 1: Draw if visible/no overflow
	START_ADDR_HI	Bits[7:0] Start Address bits [39:32]

9.5.4 PRED_EXEC - R7xx/Evergreen/Cayman

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 will start parsing the DWord immediately following EXEC_COUNT DWords.

PRED_EXEC Packet Description

DW	Field Name	Description
1	HEADER	header; Header of the packet
2	CONTROL	bit [31:24] device_select; To select one or more device IDs upon which the subsequent predicated packets will be executed bit [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.

9.6 Synchronization Packets

9.6.1 EVENT_WRITE - R7xx/Evergreen/Cayman

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.

EVENT_WRITE Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	EVENT_INDEX EVENT_TYPE	EVENT_INDEX[11:8] - Event Index - 0000: Any non-Time Stamp/non-Fence/non-Trap EVENT_TYPE not listed. - 0001: ZPASS_DONE - 0010: SAMPLE_PIPELINESTAT - 0011: SAMPLE_STREAMOUTSTATS, SAMPLE_STREAMOUTSTATS1, SAMPLE_STREAMOUTSTATS2, SAMPLE_STREAMOUTSTATS3 - 0100: CS_PARTIAL_FLUSH, VS_PARTIAL_FLUSH, PS_PARTIAL_FLUSH - 0101: Reserved for EVENT_WRITE_EOP time stamp/fence event types - 0111 - 1111: Reserved for future use. EVENT_TYPE[5:0] - The CP writes this value to the VGT_EVENT_INITIATOR register for the assigned context.
3	ADDRESS_LO	Bits [31:3] - Lower bits of QWORD-Aligned Address. Bits [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	Bits [7:0] - Upper bits of Address [39:32] Driver should only supply this DW for Sample_PipelineStats, Sample_StreamoutStats, and Zpass (Occlusion).

9.6.2 EVENT_WRITE_EOP - R7xx/Evergreen/Cayman

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 time stamp will be written back.

Supported Events are:

- Cache Flush TS: provides the driver with a pipelined fence/time stamp indicating that the CBs, DBs, and SX have

completed flushing their caches.
- Cache Flush And Inval TS: same as above but the CBs, DBs, and SX, also invalidate their caches before sending the pulse back to the CP.
- Bottom Of Pipe TS: provides the driver with a pipelined time stamp indicating that the CBs, DBs, and SX 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/DBs/SX 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.

EVENT_WRITE_EOP Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	EVENT_INDEX EVENT_TYPE	EVENT_INDEX[11:8] - Event Index - 0000: Any non-Time Stamp/non-Fence/non-Trap EVENT_TYPE not listed. - 0001: ZPASS_DONE - 0010: SAMPLE_PIPELINESTAT - 0011: SAMPLE_STREAMOUTSTATS, SAMPLE_STREAMOUTSTATS1, SAMPLE_STREAMOUTSTATS2, SAMPLE_STREAMOUTSTATS3 0100: CS_PARTIAL_FLUSH, VS_PARTIAL_FLUSH, PS_PARTIAL_FLUSH - 0101: Reserved for EVENT_WRITE_EOP time stamp/fence event types - 0111- 1111: Reserved for future use. EVENT_TYPE[5:0] - The CP writes this value to the VGT_EVENT_INITIATOR register for the assigned context.
3	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. Bits [1:0] - Reserved & must be programmed to zero.
4	DATA_SEL	[31:29] - DATA_SEL - Selects Source of Data to be written for a End-of-Pipe Done event. - 000 - None, i.e., Discard Data. Used when only an interrupt is needed. Program INT_SEL to 01". - 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. - 101-111 Reserved for future use.
	INT_SEL	[25:24] - INT_SEL 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	[7:0] - ADDR_HI, address bits[39:32]. External memory address written for a End-of-Pipe Done event. Read returns last value written to memory.
5	DATA_LO	Data [31:0] value that will be written to memory when event occurs. Driver should always supply this DW

6	DATA_HI	Data [63:32] value that will be written to memory when event occurs. Driver should always supply this DW

9.6.3 EVENT_WRITE_EOS - Evergreen/Cayman

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. The CP will generate the end-of-shader event given in the packet by writing to the VGT_EVENT_INITIATOR register.

EVENT_WRITE_EOS Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	EVENT_INDEX EVENT_TYPE	EVENT_INDEX[11:8] - Event Index - 0000 - 0100: Reserved for EVENT_WRITE event types - 0101: Reserved for EVENT_WRITE_EOP event types - 0110: CS Done, PS Done - Others up to 1111: Reserved for future use. EVENT_TYPE[5:0] - The CP writes this value to the VGT_EVENT_INITIATOR register for the assigned context.
3	ADDRESS_LO	[31:2] - Lower bits of DWord-Aligned Address. [1:0] - Reserved & must be programmed to zero.
4	CMD	[31:29] - CMD - 000: Store Append Count to memory - 001: Store GDS Data to memory - 010: Store 32-bit "DATA" from this packet to memory - Others: Reserved
	ADDRESS_HI	[7:0] - ADDR_HI, address bits[39:32].External memory address written for a End-of-Shader Done event. Read returns last value written to memory.
5	SIZE REG_ADDR OR DATA	[30:16] - SIZE: Number of DWs to read from the GDS. Currently supports reading of 16KDWs. A value of zero is not supported when CMD = 001. [15:0] - REG_ADDR: Register address of the register to read, not an offset into a specific register type sub-space. OR [31:0] - DATA: EOS fence value that will be written to memory when event occurs. This DW is always supplied, but only valid if "CMD" + Store Packet Data.

9.6.4 MEM_SEMAPHORE - R7xx/Evergreen/Cayman

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).

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. This is a multi-bit field to be DW compatible with EVENT_WRITE_EOP. - 110: Signal Semaphore. - 111: Wait Semaphore.
	CLIENT_CODE	[25:24] - CLIENT_CODE - Client Code - 00: CP - 01: CB - 10: DB - 11: SX
	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 - This field should be set in evergreen, but in cayman it is reserved and should be set to zero. If set the Wait_Semaphore will wait until all outstanding End of Pipe (and therefore Signal_Semaphores) have completed, before being issued. - 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

9.6.5 PFP_SYNC_ME - R7xx/Evergreen/Cayman

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

9.6.6 STRMOUT_BUFFER_UPDATE - R7xx/Evergreen/Cayman

STMOUT_BUFFER_UPDATE Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	CONTROL	Bits [31:10] - Reserved Bits [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 Bits [7:3] - Reserved

		Bits [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: Read data from SRC_ADDRESS - 11: None Bit [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	Bits [31:2] - Lower bits of DWord-Aligned Destination Address [31:2]. Valid only if Update_Memory is "1". Bits [1:0] - Swap [1:0] function used for data write.
4	DST_ADDRESS_HI	Bits [7:0] - Upper bits of Destination Address [39:32]. DW valid only if Store BufferFilledSize is 01".
5	[BUFFER_OFFSET] - or - SRC_ADDRESS_LO	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 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 [7:0] - Upper bits of Source Address [39:32]. DW valid only if Source_Select is "10".

9.6.7 SURFACE_SYNC - R7xx/Evergreen/Cayman

The SURFACE_SYNC packet will allow the driver to place the surface sync commands as one atomic packet and to allow the driver to send the same COHER_CNTL value regardless of the ASIC.

SURFACE_SYNC Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	ENGINE	[31] - ENGINE: 0=PFP, 1=ME. Perform Surface Synhronization at CP.PFP (so index DMA requests are not sent to VGT until the surface is coherent) or at the CP.ME as done in previous ASICs.
	COHER_CNTL	[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 [39:8]. 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	VMID	[31:24] - VMID[7:0]: Virtual Memory ID to be synchronized. (cayman)
	POLL_INTERVAL	[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.

9.6.8 WAIT_REG_MEM - R7xx/Evergreen/Cayman

The WAIT_REG_MEM packet can be processed by either the CP.PFP or the CP.ME, as indicated by the ENGINE field.

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.
		Bits [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 theReference Value. - 101 - Greater Than or Equal (>=) to the Reference Value. - 110 - Greater Than (>) the Reference Value. - 111 - Reserved. If 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 Bits [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 [7:0] specify bits 39:32 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.

9.7 Misc Packets

9.7.1 MEM_WRITE - R7xx/Evergreen/Cayman

The MEM_WRITE packet provides the opportunity to write two DWords to a QWORD-aligned memory location at the top of the graphics pipe.

MEM_WRITE Packet Description

DW	Field Name	Description
1	HEADER	Header of the packet
2	ADDRESS_LO SWAP	[31:3] Lower bits of QWORD-Aligned Address. Bits [1:0] - Swap function used for data write.
3	DATA32 WR_CONFIRM CNTR_SEL CNTR64_SEL ADDRESS_HI	[18] - Data32 writes on the lower 32-bits to memory when set to '1'. [17] - Write Confirm is requested for this write when set. [16] - Selects sending - 0: embedded packet Data - 1: a copy of the 64 bit counter indicated by the CPCNTR_SEL field [14] - Selects the 64 bit counter to send if bit 16 (CNTR_SEL) is programmed to "1". - 0: Send the current value of CP's copy of the 64 -bit GPU counter - 1: Send current value of the CP_PERFCOUNTER_HI/LO. The intent is for the driver to have already selected the always count option for CP_PERFCOUNTER_SELECT and requested the CP to start counting via the CP_PERFMON_CNTL register. [7:0] - Upper bits (39:32) of Address
4	DATA_LO	[31:0] Data
5	DATA_HI	[31:0] Data

9.7.2 NOP - R7xx/Evergreen/Cayman

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

MEM_WRITE Packet Description

1	HEADER	Header of the packet.
2	{DATA_BLOCK}	DATA_BLOCK

This field may consist of a number of DWords, and the content may be anything.