P10®

Reference Guide Volume III - Core Registers

DRAFT

3 Dlabs®

P10®

Reference Guide Volume III - Graphics Core and T&L Registers

Issue 1

Proprietary Notice

Email: info@3dlabs.com
Web: http://www.3dlabs.com

3Dlabs Ltd.
Meadlake Place
Thorpe Lea Road, Egham
Surrey, TW20 8HE
United Kingdom
Tel: +44 (0) 1784 470555
Fax: +44 (0) 1784 470699

3Dlabs K.K.
Shiroyama JT Mori Bldg 16F
40301 Toranomon
Minato-ku, Tokyo, 105, Japan
Tel: +81-3-5403-4653
Fax: +91-3-5403-4646

3Dlabs GmbH
Breckenheimer Weg 29
65205 Wiesbaden
Deutschland
Tel: +49 6122 916 778
Fax: +49 6122 919 646

3Dlabs Inc.
480 Potrero Avenue
Sunnyvale, CA 94086,
United States
Tel: +1 (408) 530-4700
Fax: +1 (408) 530-4701

Change History

Document	Issue	Date	Change
174.2.1 01	1	08/06/2001	Creation

User Note

This manual uses hyperlinks in MSWord file distributions to improve ease of access to relevant information for online users. To enable hyperlinks, the complete *Reference Guide* and *Programmer's Guide* file set should be in a single Windows directory or folder.

Table of Contents

1 Functional Overview

This chapter describes, in section 5.1, the static T&L and graphics core registers in region 0, offset group 0x8000-0xFFFF. Within this group the registers are listed alphanumerically.

In P10 some units are fixed-function while others are programmable. Section 5.2 describes the programmable units including interface resources, instruction set and basic programming notes. Programmable registers are described in greater detail in the *P10 Programmer's Guide*. I/O registers were described previously, in chapter 4 volume II of the *P10 Reference Guide*.

Static register details may have the following format information:

Name	The register's name.
Type	Region and function, i.e. core (region 0) command
Tag	The offset of this register from the base address of the region.
Format	Can be bitfield, mask, int or float, for example.
Bit	Bit number and name
Description	What the bitfield is intended to do.
Reserved	Bits that may be used in future devices. To ensure upwards compatibility, any software should always write them as zeros.

Programmable units (Vertex Shading, Texture Coordinate, Shading, Pixel Address and Pixel) are defined by their Instruction Set, ALU characteristics and Sequencer operation. These are all presented in section 5.2.

1.1 Fixed Function Registers

AALineSamples

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x281	fixed	Yes	1 - 4	No

Bits	Name	Description
0…127	PointCoords	Sample points are defined as up to 16 pairs of 4.4 fixed point x, y coordinates, or (setting **RasterMode** *DualAALineSamplePatterns* = true) as up to 8 pairs of coordinates aligned optimally for x-major and y major lines in the lower and upper 64 bits respectively.

Notes:	Holds the sub pixel sample points used when antialiasing lines. Up to 16 sub pixel sample points can be defined using the AA Sample table. They are positioned on a 16x16 grid within a pixel and the coordinates are held as unsigned offsets from the upper left corner of the pixel.. Sample points start at byte 0 in the **AALineSamples** register. The number of sample points to use is held in the **RasterMode** field *AALineSamplePoints*[n] where n = 0 to 15. (Rasterizer)

AATriangleSamples

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x280	fixed	Yes	1 - 4	No

Bits	Name	Description
0…127	PointCoords	Sample points are defined as pairs of 4.4 fixed point x, y coordinates packed 16 to a register.

Notes:	Holds the sub pixel sample points used when antialiasing triangles or points. Up to 16 sub pixel sample points can be defined using the AA Sample table. They are positioned on a 16x16 grid within a pixel and the coordinates are held as unsigned offsets from the upper left corner of the pixel.. Sample points start at byte 0 in the **AATriangleSamples** register. The number of sample points to use is held in the **RasterMode** field *AATriangleSamplePoints[n]* where n=0 to 15. (Rasterizer)

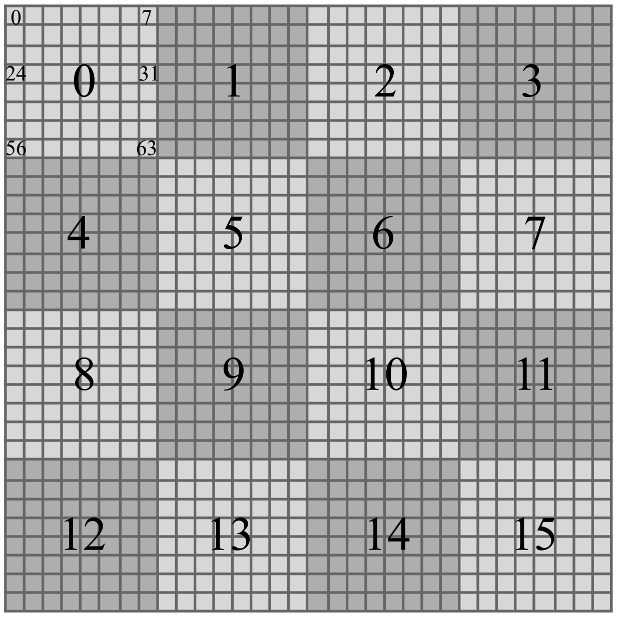
AreaStipple [0-15]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x240-0x24F	mask	Yes	2	No

Bits	Name	Description
0…63	PatternMask	Holds the screen relative area stipple pattern. See below for how these are defined. Loaded 64 bits at a time

Notes:	When the *AreaStippleEnable* bit in the **RasterMode** register is Set every tile is further qualified by the area stipple pattern held in the **AreaStipple[0…15]** registers. The selection between 8x8 and 32x32 stipple pattern is done by **RasterMode**.*AreaStipple8x8*, (8x8 when set, 32x32 when clear). The area stipple can be inverted by **RasterMode**.*Invert*, but not mirrored, byte-swapped or nibble-swapped. The diagram below shows how the area stipple bits are allocated to the **AreaStipple0…15** registers. (Rasterizer)

Figure 1.1 Area Stipple Bitfield Allocation



The gray rectangles represent the 64 bits in a register and the large numbers are the register numbers. The small numbers in rectangle 0 are the bit numbers in the word. The stipple pattern is as it would appear on the screen.

Begin

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core command	0x1B0	Bitfield	No	1	No

Bits	Name	Description
0…3	PrimitiveType	The lower 4 bits sets up the primitive type to process on receiving each new vertex. It has the following values: 0 Null 1 Points 2 Lines 3 LineLoop 4 LineStrip 5 Triangles 6 TriangleStrip 7 TriangleFan 8 Quads 9 QuadStrip 10 Polygon 11 Grid
4…7	GridWidth	Grid width in vertices. The sensible range of widths is 2 to 14 vertices.
8	ProvokingVertex	When set, applies the D3D provoking vertex rules.
9…29	Reserved	
30	Enable	Enables vertex cacheing for indexed vertex arrays. This should only be enabled for Lines, Trangles or Quads
31	Invalidate	Invalidates the current vertex cache

Notes: The following restrictions apply:

- The unit should only be enabled when a single index buffer is used (or emulated). Disable it otherwise.
- The unit should only be enabled when indexed primitives are used.. Disable it otherwise.
- The cache should be invalidated when the unit is first enabled.
- The cache should be invalidated when indexed primitives follow non-indexed primitives.
- The cache should be invalidated when the same indices yield changed vertices (i.e. when the data buffers' addresses or contents change).

(Vertex Machine)

BitMask

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x18F	mask	No	1	No

Bits	Name	Description
0…31	Bitmask	Bitmask data

Notes:	When the **DrawRectangle2D** command is invoked with the operation set to *SyncOnBitMask* a bitmask must be provided for every pixel in the rectangle. If the rasterizer does not receive enough values it aborts the operation. If a bitmask is received at any other time, it is silently discarded. The bitmask is used from the least significant end and any residue at the end of a scanline is (optionally) discarded. *SyncOnBitMask* processes one scanline at a time, in the direction given by the IncreasingX and IncreasingY fields, however it is only useful when X is increasing[1]. (Rasterizer)

CacheControl

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x182	Bitmask	No	1	Yes

Bits	Name	Description
0	Flush LB Cache	
1	Invalidate LB Cache	
2	Flush Pixel Cache	
3	Invalidate Pixel Cache	
4	Invalidate Texture Primary Cache	
5	Invalidate Texture Secondary Cache	

Notes:	Implements cache control operations. (Pixel Address unit, Vertex Shader unit)

[1] For decreasing X the data or bitmask are assigned within each aligned 8 pixel group in the increasing X direction, but between the pixel groups in decreasing X order. In practice this makes this operation very hard for software to use, but all required operations can be achieved by an increasing X order.

ChangeContext

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core command	0x188	tag	No	1	Yes

Bits	Name	Description
0…27	ContextAddress	Indicates the context is changing and any local parameter values should be dumped (in 64-byte tiles) to the context record at the address given.
28…31	Reserved	
32…35	Reserved	Context-id.
36…63	Reserved	

Notes:
- This command causes the current context to be written out to memory and the new context to be loaded for the current active port. The address of the new context is supplied in the data field and is a 28 bit number giving the planar byte tile where the context record starts.
- If this is received on the isochronous port then only the units after the Context Unit will be Context Saved. If it is received on the geom port then the whole chip is Context Saved. (Context unit)
- High frequency transient data such as vertex parameters are not context-switched as any isochronous rendering will set up the plane equations directly rather than via vertex values. (GPIO Command End)

ChangePort

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core command	0x189	tag	No	1	Yes

Bits	Name	Description
0	ChangePort	When the *ChangePort* field is set to 0, the port is changed to the Geometry fifo. When set to 1, it is changed to the Isochronous fifo.

Notes: The **ChangePort** command can be used to force the active port, and hence context, to change. This is intended as a test aid. (Context unit)

CoeffAddr

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x004	bitfield	Yes	1	No

Bits	Name	Description
0…7	Address	

Notes:	Incrementing register load. The address selects the float to write to (not read from) the Coefficient Memory in the Vertex Shader unit. The address is not modified in any way.

CoeffData

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core comand	0x134	data	Yes	4	No

Bits	Name	Description
0…127	coeffdata	128 bits of coeff data

Notes:	The data is written in when the command is received. After doing the write, the coefficient address (not the program address) is incremented.

ColourPlaneDX[0…7]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x100 – 0x107	Fixed point	Yes	4	Yes

Bits	Name	Description
0…21	Gradient	Dx gradient for a color parameter
22…31	Reserved	

Notes:	These hold the four dx gradients for a colour parameter in 2's complement 9.13 fixed point format aligned on 32 bit boundaries. (Shader unit)

ColourPlaneDY[0…7]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x108 – 0x10F	Fixed point	Yes	4	Yes

Bits	Name	Description
0…21	Gradient	dy gradient for a color parameter
22…31	Reserved	

Notes:	These hold the four dy gradients for a colour parameter in 2's complement 9.13 fixed point format aligned on 32 bit boundaries. (Shader unit)

ColourPlaneStart[0…7]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x110 – 0x117	Fixed point	Yes	4	Yes

Bits	Name	Description
0…21	StartValue	Starting value for a color parameter
22…31	Reserved	

Notes:	Hold the four starting values for a colour parameter in 2's complement 9.13 fixed point format aligned on 32 bit boundaries. (Shader unit)

CommandID

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x1C0	bitfield	No	4	No

Bits	Name	Description
0…29	CommandId	Identifier which drives the 30-bit CommandId signal.
30	Reserved	
31	Intr	0 = Command interrupt is not requested. 1 = Command interrupt is requested.

Notes:

CylindricalWrap

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x212	bitfield	Yes	1	No

Bits	Name	Description
0…7	S wrap	Enables S texture coordinate wrapping
8…15	T wrap	Enables T texture coordinate wrapping

Notes:	Defines the cylindrical wrap mode for the 8 sets of texture coordinates. When a bit is Set, the corresponding S or T component of the texture will be wrapped. (Geometry)

DepthMode

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x018	bitfield	Yes	1	Yes

Bits	Name	Description
0	Enable	This bit, when set, enables the depth set up calculations
1	WriteMask	This bit, when set enables the depth value in the local buffer to be updated when doing a read-modify-write operation
2…4	CompareFunction[3]	This field selects the compare function to use. The options are: 0 = Never 1 = Less 2 = Equals 3 = Less Equals 4 = Greater 5 = Not Equal 6 = Greater Equal 7 = Always The compare operation compares the calculated depth value against the source depth value. If the compare function is 'Less' and the result is true then the calculated value is less than the source value
5…6	Width	This field holds the width in bits of the depth field in local buffer. The options are: 0 = 16 bits wide 1 = 24 bits wide 2 = 32 bits wide 3 = 16 bits wide
7,8	Reserved	

9	Format	This bit controls the format of the Z value in the local buffer. The options are: 0 = Integer 1 = Floating Point
10	Complement	This bit, when set, causes the set up calculations to be done with 1.0 - Z value at each vertex rather than Z. This, in conjunction with a floating point Z format allows a non linear Z buffer to be used. This field should not be changed in the middle of a mesh or during rendering as the vertex store and depth buffer will not be consistent.
11	SamplePoint	This field determines where the sample point in a pixel is considered to be. The two options are: 0 = Centre of the pixel (at 0.5, 0.5) 1 = Origin of the pixel (at 0.0, 0.0) OpenGL expects the sample to be at the pixel center while D3D expects it to be at the origin.
12	MultiSampleEnable	When set (=1) maintains multiple depth buffers, one for each bit set in Depth Mode and Multi-sample Mode
13…20	MultiSampleMask	This mask normally has the same number of bits (0…n) set as there are sub-pixel samples set up in the Rasterizer. Setting fewer bits masks out subpixels which would otherwise be tested and updated – providing the basis for implementation of effects such as motion blur and depth of field.
21	UseAllSubsamples	When set, tells the system to write all subpixel buffers, i.e. there will not be any subpixel masking, so motion blur and similar effects will not be used. This sends a completely covered tile, usually without needing any coverage messages (see notes).
22…31	Reserved	

Notes:

- In the *UseAllSubsamples* field, coverage messages are still needed if the Depth test failed some subsamples. Coverage information cannot be derived from the *MultiSampleMask* field because that describes used buffers rather than total buffers (e.g. 0x8 could represent 4 buffers, or 8 buffers of which only the first 4 are used.

DrawIsocRectangle2D

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core command	0x14C	tag	No	1	Yes

Bits	Name	Description
0…12	Width[13]	Specifies the width of the rectangle in pixels. Its range is 0…8191.
13	IncreasingX	This bit, when set, specifies the rasterisation is to be done in increasing X direction.
14	IncreasingY	This bit, when set, specifies the rasterisation is to be done in increasing Y direction.
15	MultiRasteriserEnable	This bit, when set causes super tiles which are not owned by the rasteriser to be skipped.
16…28	Height[13]	Specifies the height of the rectangle in pixels. Its range is 0…8191.
29…31	Reserved	

Notes: The **DrawIsocRectangle2D** command sets up and draws rectangles in two steps. First the origin is established using the **RectanglePosition** command. The **DrawIsocRectangle2D** command provides the width, height and some mode bits and causes the rectangle to be rendered

DrawRectangle

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core command	0x14A	bitfield	No	4	No

Bits	Name	Description
0…17		x in 2's complement 14.4 fixed point
18…35		y in 2's complement 14.4 fixed point
36…53		mx in 2's complement 14.4 fixed point
54…71		px in 2's complement 14.4 fixed point
72…89		my in 2's complement 14.4 fixed point
90…107		py in 2's complement 14.4 fixed point

Notes: Tells the Rasteriser Unit to draw a rectangle. (x + mx, y + my) is the corner nearest the origin, and (x + px, y + py) the corner farthest from the origin. These fields allow an asymmetric rectangle centred on (x, y) to be defined, or a rectangle with a given position and width and height, or opposite corner of a rectangle. The mx, my, etc values are set up to describe a clockwise order when moving from the origin corner to the farthest corner.

DrawRectangle2D

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core command	0x14B	bitfield	No	1	No

Bits	Name	Description
0…12	Width	Specifies the width of the rectangle in pixels (0-8191)
13	Increasing x	When set, specifies that rasterization is to be in the direction of increasing x
14	Increasing y	When set, specifies that rasterization is to be in the direction of increasing y
15	PixelsPerScanline	This field selects the number of pixels per scanline per tile which are processed together. The options are: 4 pixels = 0 8 pixels = 1
16…28	Height	Specifies the height of the rectangle in pixels (0...8191).
29,30	Operation	0 = Normal 1 = SyncOnHostData - When set, a fragment is produced only when one of the following registers have been received from the host: *Depth*, *Stencil*, *Color* or *FBData*, *FBSourceData* 2 = SyncOnBitMask - This bit, when set, causes a number of actions: - The least significant bit or most significant bit (depending on the *MirrorBitMask* bit) in the **BitMask** register is extracted and optionally inverted (controlled by the *InvertBitMask* bit). - If this bit is 0 then any fragments are skipped. After every fragment the **BitMask** register is rotated by one bit. 3 = ScanLineOrder -
31	PackedBitMask	When set, allows a bitmask to continue across scanlines, otherwise a new bitmask is needed per scanline. This can significantly reduce the number of words of bitmask data downloaded for narrow glyphs. *Note: this is intended for glyph download to an aligned address, not for g;yph drawing to an arbitrary pixel boundary.*

Notes:	• The register the Rasteriser Unit to draw a rectangle. (x + mx, y + my) is the corner nearest the origin, and (x + px, y + py) the corner farthest from the origin. These fields allow an asymmetric rectangle centred on (x, y) to be defined, or a rectangle with a given position and width and height, or opposite corner of a rectangle. The mx, my, etc values are set up to describe a clockwise order when moving from the origin corner to the farthest corner. • With *SyncOnBitMask* enabled, if all the bits in the **BitMask** register have been used then rasterisation is suspended until a new BitMaskPattern tag is received. If any other tag is received while the rasterisation is suspended then the rasterisation is aborted. The register which caused the abort is then processed as normal. • The behaviour of *SyncOnBitMask* is slightly different when the *SyncOnHostData* bit is set to prevent a deadlock from occurring. In this case the rasterisation doesn't suspend when all the bits have been used and if new BitMaskPattern tags are not received in a timely manner then the subsequent fragments will just reuse the bit mask.

EdgeFlag

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core command	0x1BD		No	1	No

Bits	Name	Description
0	EdgeFlag	1 = Enable
1…31	Reserved	

Notes:	Sets the current edge flag to the value in the least significant bit (1 = true).

End

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core command	0x1B1	tag	No	1	No

Bits	Name	Description
0…31	Reserved	

Notes:	Terminates the series of primitives and performs any tidy up action such as forcing the closing edge of a polygon. The data field is not used. (Vertex Cache…)

FBAddrInfo[0…3]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x07F	Bitfield/fixed	Yes	1	Yes

Bits	Name	Description
0…13	x	x offset
14,15	Reserved	
16…29	y	y offset
30,31	Reserved	

Notes:	Each register holds paired user-supplied data, typically x and y offsets. These can be used within a Pixel Address program as constant data. The format is 2's complement 14 bit number and is typically used to hold counts, offsets, masks, etc. For more information on using **FBAddrInfo** for microcode programs see the *P10 Programers Guide.*

FBBaseAddr[0…3]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x074 – 0x077	data	Yes	1	Yes

Bits	Name	Description
0…27	Address	The base address for the framebuffer region in byte tile units.
28…31	Reserved	

Notes:

- These registers hold the base address in planar byte tile units of the 5 regions in memory where buffers are located. The x, y coordinates of the tile to read and/or write are calculated as part of the program and get applied to the selected buffer.
- For example, in loading fonts the origin of the tiles holding the glyph data is held in **FBBaseAddrGlobal** with **FBBufferGlobal** set up with the width of the glyph, pitch and size set to 4. The *SubFieldStartByte* and *subFieldByteCount* in **FBBuffer** are set up to select the specific byte holding the glyph's bit plane. The height field is set to the glyph's height (in tiles) to enable source read clipping. The alignment of the glyph to the tile (which also takes into account the character position within the tile) is held in FBAddrInfo[6, 7] for the x and y offsets respectively.
- The Primitive Set Up Unit has mechanisms to allow glyph rendering to be set up with the minimum number of host words. (Pixel Address)
- FBBuffer operations can also be performed on Localbuffer memory for purposes such as Depth clears. The FBBuffer would then be set back to pixel memory.

FBBaseAddrGlobal

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x070		Yes	1	No

Bits	Name	Description
0…27	Address	The base address for the framebuffer region in byte tile units.
28…31	Reserved	

Notes: Holds the base address of the framebuffer regions where the glyph data is stored. It is used when setting up a glyph.

FBBuffer[0…3]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x078 – 07B	bitfield	Yes	1	Yes

Bits	Name	Description
0	ReadEnable	When Set, initiates a framebuffer read which may be satisfied by the Pixel cache. If the Pixel Unit's operation doesn't need a read to occur then this bit would be 0, however a read would still be done if a partial tile was being processed as the whole tile is always written (there is no fragment or byte level masking on memory writes). This is only used on Destination addresses. This bit is ORed with the corresponding bit in the **FBReadEnables** register so reads can be enabled from two places.
1	AAReadOnly	This bit, when set, indicates reads should only be done (if *ReadEnable* is set) if the tile has the *aaEnable* bit set so the destination pixel values are needed during blending. If *aaEnable* is not set then the fragments have 100% coverage and the blending does not require the destination pixel data.
2…12	Width[11]	This field holds the width in tiles of the buffer. The range is 0…2047 to allow an 8K pixel wide buffer to be accommodated.
13…16	PixelBytePitch[4]	This field defines the offset between tiles in memory. This is normally the depth of a tile in bytes. The range is 0…15.
17…18	PixelSize[2]	This field defines the number of bytes read from memory (+1) and transferred to the cache. Normally this is the depth of a tile in bytes, but can be less if a subset of the field needs to be read and/or written. The corresponding **FBBaseAddr** register is updated to point to the first byte tile in the subset.
19…20	SubFieldStartByte[2]	This field defines the first byte to transfer from the cache to the Pixel Unit. Normally this is zero (0), but can be non-zero if a subset of the field needs to be read and/or written. This is useful for font alignment where for best cache efficiency the font is stored in 32 bit tiles, but only the byte holding the font bit plane needs to be aligned.
21…22	SubFieldByteCount[2]	Defines the number of bytes to transfer from the cache (+1) to the Pixel Unit. Normally this is the same as the PixelSize, but can be less if a subset of the field needs to be read and/or written.
23	XMask	When set, ANDs the x coordinate with the **xMask** register before the address is calculated. This can be used for pattern replication.

24	YMask	When set, ANDs the y coordinate with the **yMask** register before the address is calculated. This can be used for pattern replication.
25…28	Height[4]	This field holds the height of the buffer region measured in tiles. This is only used for clipping tile reads outside the region as can occur when aligning tiles in a source read. By skipping these tiles it is possible to save the memory read of the tile outside of the legal buffer region and the time taken to align and merge it. This is particularly relevant for font processing. When this field is zero then clipping is disabled.
29…31	Reserved	

Notes:	The buffer address calculation, the amount of data read from memory and the amount of data transferred from the cache to the Pixel Unit are controlled by the five **FBBuffer** registers: These 5 registers hold the key parameters concerning each buffer region. (Pixel-address). FBBuffer operations can also be performed on Localbuffer memory for purposes such as fast depth clears. The FBBuffer would then be set back to pixel memory.

FBBufferEnables

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x072	bitfield	Yes	1	No

Bits	Name	Description
0…3	Enables	One bit per buffer: 0 = FBBuffer0 enabled 1 = FBBuffer1 enabled 2 = FBBuffer2 enabled 3 = FBBuffer3 enabled
4…31	Reserved	

Notes:	Defines which of the four possible buffers the program should be run on. If no buffers are enabled then the FB subsystem is disabled - i.e. no program in the Pixel Address Unit or in the Pixel Unit will be run. (Pixel Address)

FBBufferGlobal

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x071	bitfield	Yes	1	No

Bits	Name	Description
0…3	Enables	One bit per buffer: 0 = FBBuffer0 enabled 1 = FBBuffer1 enabled 2 = FBBuffer2 enabled 3 = FBBuffer3 enabled
4…31	Reserved	

Notes:	Holds the buffer parameters of the framebuffer regions where the glyph data is stored. It is used when setting up a glyph.

FBBufferReadEnables

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x073	Bitfield	Yes	1	No

Bits	Name	Description
0	FBBuffer0	Enable = 1
1	FBBuffer1	Enable = 1
2	FBBuffer2	Enable = 1
3	FBBuffer3	Enable = 1
4	Global buffer	Enable = 1
5…31	Reserved	

Notes:	Defines the read enable status of the five possible buffers. Bit 0 corresponds to buffer 0, bit 1 to buffer 1, etc. and bit 4 to the global buffer. A bit, when set, initiates a framebuffer read which may be satisfied by the Pixel cache. If the Pixel Unit's operation doesn't need a read to occur then this bit would be 0, however a read is still done if a partial tile is being processed as the whole tile is always written (there is no fragment or byte level masking on memory writes). This is only used on Destination addresses. Each bit is ORed with the *ReadEnable* bit in the corresponding **FBBuffer** register so reads can be enabled from two places. (Pixel Address unit)

FBMode

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x01C		Yes	1	Yes

Bits	Name	Description
0	SameTileEnable	Enables 'same tile' caching. If this tile has the same coordinates as the previous tile then the read enable to the cache is forced to be false on destination reads and the *sameTile* bit in the Tile command forwarded on to the Pixel Unit. This forces it to use its own copy of the previous tile results rather than waiting for it from the cache. This improves small primitive performance where successive primitives are likely to be in the same tile. This is normally only set for simple single destination buffer processing
1…5	EntryPoint[5]	This field holds the start address of the program to run when a Tile command is received.
6…31	Reserved	

Notes:	Defines the basic mode of operation for the Pixel Address unit.

FBProg[0…15]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x080 – 0x08F	Bitfield	Yes	1	Yes

Bits	Name	Description
0…14		lower instruction of the pair
15	Reserved	
16…30		upper instruction of the pair
31	Reserved	

Notes:	Each register holds two program instructions For the instruction set see the Pixel Address unit below.

FillDrawRectangle2D

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x2CB		No	1	No

Bits	Name	Description

Notes:	Aliased to **DrawRectangle2D** by GPIO.

FillFBAddrInfo0

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x2C7		No	1	No

Bits	Name	Description

Notes:	Aliased to **FBAddrInfo0** by GPIO.

FillFBBaseAddr[0-1]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x2C5 0x2C6		No	1	No

Bits	Name	Description

Notes:	Aliased to **FBAddrInfo[0-1]** by GPIO.

FillFBBuffer0

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x2C4		No	1	No

Bits	Name	Description

Notes:	Aliased to **FBBuffer0** by GPIO.

FillFBMode[0-1]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x2C1		No	1	No

Bits	Name	Description

Notes:	Aliased to **FBMode** by GPIO.

FillGlyphPosition

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x2CC		No	1	No

Bits	Name	Description

Notes:	Aliased to **GlyphPosition** by GPIO.

FillPixelGlobal[0-1]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x2C8 – 0x2C9		No	1	No

Bits	Name	Description

Notes:	Aliased to **PixelGlobal[0-1]** by GPIO.

FillPixelMode[0-1]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x2C0 - 0x2CF		No	1	No

Bits	Name	Description

Notes:	Aliased to **PixelMode** by GPIO.

FillPrimSetupMode

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x2C3		No	1	No

Bits	Name	Description

Notes:	Aliased to **PrimSetupMode** by GPIO.

FillRasterMode[0-1]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x2C2, 0x2CD		No	1	No

Bits	Name	Description

Notes:	Aliased to **RasterMode** by GPIO.

FillRectanglePosition

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x2CA		No	1	No

Bits	Name	Description

Notes:	Aliased to **RectanglePosition** by GPIO.

FlushContext

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core command	0x18B	tag	No	1	Yes

Bits	Name	Description
0…31		

Notes:	This command causes any outstanding vertices to be processed before it is forwarded. (Vertex Shader unit). It also flushes the context state to memory.

FrustumMax

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x28C	bitfield	Yes	3	No

Bits	Name	Description
0…31	X	Pixels
31…63	Y	Pixels
64…95	Z	Pixels

Notes:	Holds the maximum x, y and z values of the viewing frustum. x and y are measured in pixels (but window relative) and z is normally 1.0. x is in the lower 32 bits, then y and finally z Only 3 significant words. (Clip)

FrustumMin

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x28B	bitfield	Yes	3	No

Bits	Name	Description
0…31	X	Pixels
31…63	Y	Pixels
64…95	Z	Pixels

Notes:	Holds the minimum x, y and z values of the viewing frustum. x and y are measured in pixels (but window relative) and z is normally 0.0. x is in the lower 32 bits, then y and finally z Only 3 significant words. (Clip)

GeometryMode

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x211	bitfield	Yes	1	No

Bits	Name	Description
0, 1	FrontPolyMode[2]	Selects the how a triangle, quad or polygon should be drawn when its orientation is facing forwards. The options are: 0 Point 1: Line 2: Fill
2, 3	BackPolyMode[2]	Selects the how a triangle, quad or polygon should be drawn when its orientation is facing backwards. The options are: 0: Point 1: Line 2: Fill
4	FrontFaceDirection	Selects which direction is the 'front' facing direction. The direction is important as it is used to determine if a triangle, etc. should be culled (if enabled), the material to use during lighting, and the PolyMode to use: 0: Clockwise 1 Counter Clockwise
5	PolygonCull	Enables polygon culling based on the front face direction. It is ignored for points, lines and rectangles.
6, 7	PolygonCullFace[2]	This field determines which direction of face should be culled (if enabled). It has the following values: 0: Front 1: Back 2: Front and Back
8	FlatShading	When set, selects flat shading to be used, otherwise Gouraud shading will be used.
9… 14	UserClipMask[6]	There is one bit per user defined clipping plane. Clipping against a plane is enabled when the corresponding bit is set. The clipping plane is defined in eye space. Bit 0 (i.e. bit 9 in register) corresponds to UserClip0.
15	PolygonOffsetPoint	This bit, if set, causes the polygon offset to be calculated and applied to the points of a polygon when PolyMode is set to Point.
16	PolygonOffsetLine	This bit, if set, causes the polygon offset to be calculated and applied to the lines of a polygon when PolyMode is set to Line.
17	PolygonOffsetFill	This bit, if set, causes the polygon offset to be calculated and applied to the triangles of a polygon when PolyMode is set to Fill.

18	*ClipPoint*	This bit, if set, causes the points to be clipped against the guard band limits and not the view frustum limits. This has the effect of allowing points just outside of the viewing frustum, but whose area extends into the viewing frustum to be drawn. OpenGL requires a point (of any size) is rejected if the vertex is out of view. D3D PointSprites require that the visible part (if any) of the point is still rendered even when the vertex is out of view.
19…21	UploadParameters[3]	This field, when set appropriately, causes the parameter results which would normally be sent to set up a primitive to be made available for upload in different registers. The parameters are provided in order ColourA…ColourH and then TextureA…TextureH, but only for those parameters calculated in the Vertex Shading Unit. No parameters are passed for primitives which have been clipped because the intended use for this is to allow data in on pass of the Vertex Shader to be passed onto the next pass of the Vertex Shader. The options are:: 0 = none. 1 = Use the Upload128 command. 2 = Use the VertexBufferData command. 3 = Use the Pixel Command. 4 = Use the Pixel Command, but pack the least significant byte into the bottom 32 bits.

22	OutputPointSize	This bit, when set, causes the selected parameter to be used as the point size rather than a colour or texture coordinate. The point size will be taken from the bottom 32 bits of the parameter and the other 96 bits are effectively discarded.
23…26	PointSizeParameter[4]	This field identifies which parameter, if any, should be used as the point size.
27	RasterPosEnable	This bit, when set, will cause all points to be treated as rectangles for the purposes of implementing the RasterPos operation in OpenGL.
28…31	Reserved	

Notes: Defines the operation of the Geometry unit. The association of the vertex ordering to a front facing triangle is defined in the **GeometryMode** register. The normal OpenGL way is to calculate the 'area' of the triangle. A zero area is a degenerate triangle (i.e. two or three of the vertices have the same coordinates), otherwise the sign of area indicates if the vertex ordering is clockwise or counter clockwise.

GetCurrent

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core command	0x1BE	tag	No	1	No

Bits	Name	Description
0…31	GetCurrent	Tag only

Notes:

- OpenGL can query at any time what the current values are. Tracking this in software impacts performance as it is the rate at which vertex data is processed which determines overall throughput. Display lists and vertex arrays must also be included in the tracking and it is desirable that the host does not touch any of this vertex data.
- The **GetCurrent** command triggers the current values to be output to the HostOut FIFO using the **Upload128** register where they can be read or DMAed into memory. All 16 parameters are written from this unit and the Vertex Machine Unit appends the current edge flag information.
- The Get buffer holds the 16 parameters in order from *VertexData0* to *VertexData15* and then the current edge flag. The actual meaning of the parameters depends on what conventions have been adopted.

GidMode

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x015	bitfield	Yes	1	Yes

Bits	Name	Description
0	Enable	When set, allows GID testing to be done so the Reference field in this register is compared against the GID information provided by the cache. The pixel ownership is True whenever the two are the same. When this test is disabled then every pixel is owned.
1…8	Reference[8]	This holds the GID value to test each pixel against.
9	Present	This bit, when set, indicates the local buffer pixel format includes the GID field. The GID field is always the least significant byte in the pixel, if it is present.
10	EarlyExitProcessing	This bit, when set, enables early exit processing for the GID, stencil and depth tests.

Notes:	State in these registers is used to control the address generation, reads and number of bytes accessed.

GlyphAddr

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x206	bitfield	Yes	1	No

Bits	Name	Description
0…4	Plane	bit plane in the 32 bit tile to use
5…28	PlaneAddr	address of the planar byte tile where the glyph data starts
29…31	Reserved	

Notes:

GlyphPosition

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x207	Bitfield	No	1	Yes

Bits	Name	Description
0…13	x	
14,15	Reserved	
16…29	y	
30,31	Reserved	

Notes:	Holds the position of the glyph on the screen in 2's complement screen relative coordinates (i.e. the window origin value is ignored). The values are updated as part of the RenderGlyph command.

GuardBandLimits

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x28D	bitfield	Yes	4	No

Bits	Name	Description
0…31	xMin	Floating point value
32…63	yMin	Floating point value
64…95	xMax	Floating point value
96…127	yMax	Floating point value

Notes:	Holds the two corners of the guard band clipping rectangle. The coordinates are entered in windows-relative pixels.

HoldPort

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x18A		NO	1	Yes

Bits	Name	Description
0	Hold	0= Release 1 = Hold
1..31	Reserved	

Notes:	When set (=1) prevents any change to the receiving port (Geometry of Isochronous) until the port is released by resetting bit 1 (=0).

HostOutMode

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x01E	Bitfield	Yes	1	No

Bits	Name	Description
0…1	StatsOperation[2]	This field controls the type of statistics which are gathered on primitives which get rendered. The options are: 0 = None 1 = Picking 2 = Extent
2	OutputSyncTag	This bit, when set, allows Sync tags to be forwarded to the bus interface unit.
3	OutputUploadTag	This bit, when set, allows UploadData tags and data to be written to the output FIFO.
4	OutputStatsTag	This bit, when set, allows statistics related tags and data to be written to the output FIFO.
5	OutputUploadDMA Tags	This bit, when set, allows the UploadDMAControl and UploadDMA tags to be output.

Notes:	Defines HostOut functionality.

InvalidateSecondaryCacheCount

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x0EC	Bitfield	Yes	1	Yes

Bits	Name	Description
0…9	Count	Number of quad-byte tiles to invalidate
10…31	Reserved	

Notes:	Holds a 10-bit Count value used when invalidating secondary texture cache entries based on their tile address. N.B. that the count is in 32-bit tile groups.

InvalidateSecondaryTextureCache

Type	Tag	Format	Context Sw	Datawords	Isochronous
Command	0x183	Address	No	1	Yes

Bits	Name	Description
0…31	Address	

Notes:	Invalidates tiles starting at the address in the 32-bit data field if they are in the secondary texture cache. The address is a byte tile address, but the bottom two bits are ignored – to reach the address of the next tile to invalidate the current address is incremented by 4.

InvViewPortScale

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x283	Float	Yes	3	No

Bits	Name	Description
0…31	X	
32…63	Y	
64…95	Z	

Notes:	Inverse viewpoint scaling factor for the x, y and z directions as floating point numbers. This is read when clipping, to undo the viewport scaling factor already applied to the input window coordinates.

LBBaseAddr

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x012	int	Yes	1	Yes

Bits	Name	Description
0…7	BaseAddress	LB region base address
8…31	Reserved	

Notes:	Holds the base address of the local buffer region. The address is in planar byte tile units and is a 28 bit value

LBMode

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x011	Bitfield	Yes	1	Yes

Bits	Name	Description
0	SameTileEnable	This bit, when set enables 'same tile' caching. If this tile has the same coordinates as the previous tile then the read enable to the cache is forced to be false and the sameTile bit in the Tile command forwarded on to the GDS Unit. This forces it to use its own copy of the previous tile results rather than waiting for it from the cache. This improves the small primitive persormance where sucessive primitives are likely to be in the same tile.
1…11	Width[11]	This field holds the width in tiles of the buffer. The range is 0…2047 to allow up to an 8K pixel wide buffer to be accommodated.
12…14	PixelBytePitch[3]	This field defines the offset between tiles in memory. This is normally the depth of a tile in bytes. The range is 1…8.
15…26	OffsetBetweenBuffers	Holds the offset between successive multisample buffers, defined as multiples of 1024-byte tiles. An additional 24 (decimal) byte tile offset is also added between successive multisample buffers to reduce page/bank costs in the memory system when cycling between the multisample buffers and each tile..
27…31	Reserved	

Notes:

- **LBMode** configures Local Buffer setup including **LBAddress**, which calculates the address where the data for the input tile is stored in memory.
- Local buffer data is held in byte planar format - each memory read returns the same byte of data for all fragments within a tile. Multiple reads to successive addresses are needed to build up the full width of the local buffer pixel data. Storing the data in this way results in a consistent format irrespective of the size of a local buffer pixel so keeps the address calculation simple. It also allows non 'power of two' pixel depths without complicated packing of pixels into memory.
- Local buffer pixel depths from 1 to 6 bytes are supported although not all bytes allocated have to be read, if only a subset of the information is needed. An example where this is useful is in just reading the GID field when doing 2D operations.
- The origin of the buffer in memory is always the top left corner and the maximum width is 2047 tiles (to allow an 8K pixel width). Tiles are stored sequentially in memory..

LimitLine

Type	**Tag**	**Format**	**Context Sw**	**Datawords**	**Isochronous**
Core control	0x17B	bitfield	No	2	No

Bits	**Name**	**Description**
0…17	StartLimit	New start limit
18…31	Reserved	
32…49	EndLimit	New end limit
50…63	Reserved	

Notes:	Used to limit the extent of the line in x or y depending on the major axis of the line. (Rasterizer)

LineStart

Type	**Tag**	**Format**	**Context Sw**	**Datawords**	**Isochronous**
Core command	0x204	Fixed point	Yes	1	No

Bits	**Name**	**Description**
0…13	XStart	X start parameter
14,15	Reserved	
16…29	YStart	Y start parameter
30,31	Reserved	

Notes:	Holds the start coordinate of the line drawn by the **RenderLine2D** command. After the line has been drawn the **LineStart** register is updated with the line end point passed in the **RenderLine2D** command. This allows polylines to set up the start vertex with the **LineStart** command, but then just use the **RenderLine2D** command for all subsequent vertices.

LineStipple

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x201	Bitfield	Yes	1	No

Bits	Name	Description
0…7	RepeatFactor[8]	This field holds the positive repeat factor for stippled lines. The repeat factor stored here is one less than the desired repeat factor.
8…23	Pattern[16]	This field holds the stipple pattern to use for lines.
24…31	Reserved	

Notes:	Defines the stipple pattern and repeat factor to use for all lines

LineStipplePosition

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x20E	Bitfield	Yes	1	No

Bits	Name	Description
0…3	BitPosition	Current bit position
4…11	RepeatCount	Repeat counter
12…31	Reserved	

Notes:	holds the current bit position and the repeat counter in the stipple pattern. This is normally only used during context save and restore.

MaxHitRegion

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x186	bitfield	No	1	Yes

Bits	Name	Description
0…12	MaxX	unsigned maximum X
13…25	MaxY	unsigned maximum Y
26…31	Reserved	

Notes:	Writes the current value of the **maxRegion** register to the output FIFO under control of the **HostOutMode** settings. The data field (on input) is not used.(Host Out)

MaxRegion

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x0E1	Bitfield	Yes	1	Yes

Bits	Name	Description
0…12	MaxX	unsigned maximum X
13…25	MaxY	unsigned maximum Y
26…31	Reserved	

Notes:	Starts the maximum region register, which is then updated during Extent checking.

MinHitRegion

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x185	Bitfield	No	1	Yes

Bits	Name	Description
0…12	MinX	unsigned minimum X
13…25	MinY	unsigned minimum Y
26…31	Reserved	

Notes:	Writes the current value of the **MinRegion** register to the output FIFO under control of the **HostOutMode** settings. The data field (on input) is not used. (Host Out)

MinRegion

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x0E0	Bitfield	Yes	1	No

Bits	Name	Description
0…12	MinX	unsigned minimum X
13…25	MinY	unsigned minimum Y
26…31	Reserved	

Notes:	Starts the minimum region register, which is then updated during Extent checking.

Nop

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core command	0x1CC		Yes	-	Yes

Bits	Name	Description
0…31	Reserved	

Notes:	Null command

ParameterSetUpMode

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x01A	Bitfield	Yes	1	Yes

Bits	Name	Description
0…7	UseProvoking[8]	These bits specify which color parameters should be treated as being flat shaded when a primitive is flat shaded. These are not used for points (which are automatically flat shaded) as this is achieved by using the t1…t4 parameters.
8	SamplePoint	Determines where the sample point in a pixel is considered to be. The two options are: 0 = Center of the pixel (at 0.5, 0.5) 1 = Origin of the pixel (at 0.0, 0.0) OpenGL expects the sample to be at the pixel center while D3D expects it to be at the origin.

9	TwoSidedEnable	When set, divides the 8 sets of colours in two: • If the *frontFacing* bit in the **Tile** register is set (=1) it selects set A, C, E, G; • If the *frontFacing* bit in the **Tile** register is reset (=0) it selects set B, D, F, H. • Both sets map to **ColourPlane***[0…3] on output. This allows OpenGL two-sided lighting to be implemented.
10	UnitTexture	When set, forces the texture coordinates to vary from (0, 0, 0, 1) to (1, 1, 0, 1) across the primitive. This is intended to support Sprite Points where, unlike normal points, the texture coordinates vary from 0 at the origin to 1 in the opposite corner. The q value is forced to be one so no perspective is applied across the sprite point.
11…31	Reserved	

Notes:

PickResult

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core command	0x184	tag	No	1	Yes

Bits	Name	Description
		Output = 0 for false or 1 for true.

Notes: This command causes the current value of the pick result flag to be written to the output FIFO under control of the HostOutMode settings. The data field (on input) is not used.

PixelData

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x18E	Data	No	1	No

Bits	Name	Description
0…31	Data	

Notes:	Used in the Pixel unit to pass run length data to the Rasteriser, which holds the packed data for an image download. It only has an effect when the **DrawRectangle2D** command is invoked with the operation set to *SyncOnHostData*. If received at any other time it is silently discarded. When the pixel data is not 32 bits wide the data is unpacked from the least significant end and any residue at the end of a scanline is discarded.

PixelGlobal[0…7]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x068 – 0x06F	Data	Yes	1	Yes

Bits	Name	Description
0…31	Data	Four 8-bit bytes of program data

Notes:	In Primitive Setup, holds the bit plane the glyph data is present in. In the Pixel unit, updates the global registers. The registers are updated 32 bits at a time but are read by a program one byte at a time. Byte 0 (from the program) is the ls byte of PixelGlobal0. Byte 31 (from the program) is the ms byte of PixelGlobal7. Byte31 can be optionally updated by the **RunPixelProg** register.

PixelMask

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x0F0	mask	Yes	2	Yes

Bits	Name	Description
0…63	Mask	

Notes:	Post-context unit 64 bit tags generated by the rasteriser from the bitmask data (when suitably enabled). These can be used by the Pixel Unit to differentiate between foreground and background colour pixels. When this is received it is optionally byte and nibble swapped, mirrored and inverted before being sent on. Note the swapping and mirroring is done separately on the upper and lower 32 bits.

PixelMode

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x01D	Bitmap	Yes	1	Yes

Bits	Name	Description
0…6	TileAddrOnly	This field holds the address of the program to run when a Tile command is received (assuming it is enabled) when the *progID* is 0 and the *SameTile* bit in the **Tile** register is 0.
7…13	TileAddrFirst	This field holds the address of the program to run when a **Tile** command is received (assuming it is enabled) when the *progID* is 1.
14…20	TileAddrMiddle	Holds the address of the program to run when a **Tile** command is received (assuming it is enabled) when the *progID* is 2. It also holds the address of the program to run when the *progID* is 0, but the *sameTile* bit in the **Tile** register is set.
21…27	TileAddrLast	Holds the address of the program to run when a **Tile** command is received (assuming it is enabled) when the *progID* is 3.
28…31	Reserved	

Notes:

PixelProgramAddr

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x002	Tag	Yes	1	Yes

Bits	Name	Description
0…6	Address	
7…31	Reserved	

Notes: Holds the address where subsequent PixelProgramData commands will be loaded. The address is auto incremented after every load. (Pixel unit)

PixelProgramData

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x132	User data	Yes	2	yes

Bits	Name	Description
0…63	Data	Hold tag indicating number of instructions

Notes: Holds the program data to write into the program memory (WCS). After receiving this message and doing the write the program address is incremented.

PointSize

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x20A	Float	No	1	yes

Bits	Name	Description
0…31	PointSize	point size as float

Notes: Holds the point size as a floating point number. The point size is automatically clamped and rounded before use so the range is:

1.0 <= aliased point size < 128 in steps of 1.0

1/16 <= antialiased point size < 128 in steps of 1/16 (Primitive Setup)

PolygonOffsetBias

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x014	Float	Yes	1	Yes

Bits	Name	Description
0…31	Bias	

Notes:	Polygon offset bias as a floatingpoint number

PolygonOffsetFactor

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x013	Float	Yes	1	Yes

Bits	Name	Description
0…31	Offset	

Notes:	Polygon offset factor as a floatingpoint number. (Depth Setup)

PrimSetupMode

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x200	Bitfield	Yes	1	No

Bits	Name	Description
0	AAPointEnable	This bit, when set, causes points to be rendered as antialiased points, otherwise they are drawn as aliased points.
1	AALineEnable	This bit, when set, causes lines to be rendered as antialiased lines, otherwise they are drawn as aliased lines. This bit is ignored by the RenderLine2D command.
2	DiamondExit	This bit, when set, causes aliased line end points to be modified based on the OpenGL diamond exit rule.
3	LineStippleEnable	This bit, when set, enables line stipple processing on both aliased and antialiased lines. The stipple pattern and repeat are held in the LineStipple message. This bit is ignored by the RenderLine2D command.
4	ConstantLineParameters	This bit, when set, avoids any parameter gradient calculations from being done for aliased lines. This is a performance optimisation.
5	AATriangleEnable	This bit, when set, causes triangles to be rendered as antialiased triangles.
6…9	LineWidth[4]	This field holds the width of an aliased line. Legal values are 0…15.
10…17	AALineWidth[8]	This field holds the width of an antialiased line. The width is in unsigned 4.4 fixed point format.
18	NoPlaneEquationsNeeded	This bit, when set, disabled the plane equation calculations and prevents the results from being sent to downstream units. This is primarily done to reduce message traffic and not to increase the calculation speed of this unit.
19	PointGradientEnable	This bit, when set, enables the gradient across a point to be calculated so that a texture map, for example, can be used to modify fragments within the point. The gradient is set up so that (0, 0) is in the top left and (1, 1) is in the bottom right.

20	BiasY	This bit, when set, forces y coordinate value to be incremented by one before it is used during the primitive set up. This allows OpenGL compatible coordinates and arises because the OpenGL coordinate system has its origin at the bottom left and not at the top left like P10 does.
21	D3DpointRules	When set, forces aliased points to conform to D3D point rules about odd and even point size handling and where they are rasterized.
22…31	Reserved	

Notes:	Defines the basic mode of operation for Primitive Setup (PrimSetup).

RasterMode

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x208	bitfield	Yes	1	No

Bits	Name	Description
0	SampleRule	Determines where the sample point in a pixel is considered to be. The two options are: 0 = Centre of the pixel (at 0.5, 0.5) 1 = Origin of the pixel (at 0.0, 0.0) OpenGL expects the sample to be at the pixel center while D3D expects it to be at the origin.
1…4	AATriangleSample Points[4]	Holds the number of entries in the AASamples table to use when antialiasing triangles and points. A value of zero means use one entry, etc. More entries improve antialiasing quality at the expense of performance.
5	IncludeLineEndPoint	When set, includes the line's end point. This is normally reset (=0) for OpenGL.
6	AAType	Controls how antialiasing coverage is accumulated. This can be done as a mask (setting the number of sample points to be greater than 7 discards earlier sample results), or as a count. The coverage mask allows T buffer-like algorithms to be implemented, while a coverage count is more typical for OpenGL style antialiasing. The options are 0 = Mask 1 = Count
7	UserScissorEnable	Clips the rasterised region (the intersection of the primitive against the visible rectangle) against the user supplied scissor region. This is done as the final stage so any host data or bitmask for pixels outside of the user scissor region is consumed as expected.
8	AreaStippleEnable	Enables testing of the rasteriser output against the area stipple pattern. Fragments with a corresponding area stipple bit of zero are discarded.
9	AreaStipple8x8	Reduces the area stipple table from 32x32 in size to 8x8 in size. This avoids the software having to replicate an 8x8 stipple pattern up to 32x32 in size.
10…11	ByteSwap	Controls the byte swapping of the image data , bitmask data or area stipple patterns. If the input data has its bytes labelled ABCD then the options are: 0 = ABCD (i.e. no swap) 1 = BADC 2 = CDAB 3 = DCBA

12	Mirror	Controls mirroring of the image data, bitmask data or area stipple patterns. When set, bits 0 and 31 are swapped, bits 1 and 30 are swapped, etc.
13…15	PixelSize	Holds the pixel size for image data being downloaded through the rasteriser. The options are: 0 = 4 bits 1 = 8 bits 2 = 16 bits 3 = 24 bits 4 = 32 bits
16	NTLines	Forces the edge flags to be set in such a way that NT compliant lines are rasterised.
17	MultiRasterEnable	This bit, when set, forces the rasteriser to skip over super tiles it doesn't own and to reject small primitives which fall entirely in super tiles also not owned. Super tile ownership is established via an external signal.
18…21	AALineSamplePoints[4]	This field holds the number of entries in the AASamples table to use when antialiasing lines. A value of zero means use one entry, etc. More entries improve antialiasing quality at the expense of performance.
22	DualAALineSample Patterns	Setting this bit selects an AA sample pattern from the lower or upper half of the sample point table depending on the orientation of the line (X-major or Y-major). • If the line is x major then the AA sample points will be taken from the lower half of the AA line sample point table, otherwise for y major lines they are taken from the upper half of the table. The maximum number of samples in each set is 8 when in this mode. • The reason for the dual-mode is that you can tailor each of the two sample patterns to suit the line type, and hence get better quality images with fewer samples. This is to benefit performance. • The sample coordinates are set up to go horizontally through the pixel for x-major lines and vertically through the pixel for y-major lines.
23	Invert	Inverts any image data, bitmask data or area stipple patterns before using them or passing them on
24	GeneratePixelMask	This bit passes the bitmask data (after suitable alignment to the current tile) to the Pixel Unit as a PixelMask. When reset (=0) it is ANDed with the tile mask to delete fragments from the tile.
25	AreaStippleRule	Applies OpenGL rules to the area stipple (if enabled). In OpenGL the area stipple is not applied to points and lines.

26	NibbleSwap	Swaps the two nibbles within each byte before the image, bitmask data or area stipple patterns.
27	LimitLine	Limits the extent of a line by using the limits defined in the **LineLimits** register. This should only be used when very fine control is needed over exactly which pixels are generated (as for some NT lines).
28…31	Reserved	

Notes:

- The Rasterizer calculates the fragment visibility of the set of tiles which overlaps the primitive and sends them to the rest of the chip via the **Tile** message. All primitive types are decomposed into 8x8 screen aligned tiles.
- The coordinate system the rasterizer works in is ±8K pixels with 16x16 sub pixels per pixel. The origin is always top left and the rasterizer coordinates are always screen relative. All primitives except **Rectangle2D** define their vertices in sub pixel units.
- The sample point can be in the center of the pixel as OpenGL expects or at the origin of the pixel for D3D. This is controlled by the *SamplePoint* field in e.g. **DepthMode**.
- A primitive is described by one or more vertices and optionally some supplementary width/height/size information. This extra information, if any, is part of the **Draw*** command.
- The types of primitives handled directly are:

Antialiased Point	Line
Triangle	Rectangle
Rectangle2D	

RasterPosRectangle

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x202	fixed	Yes	1	No

Bits	Name	Description
0…12	width	
13…15	Reserved	
16…28	height	
29…31	Reserved	

Notes: Holds the width and height of the rectangle to draw when initiated by the **RenderRectangle** command.

RectanglePosition

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x203	fixed	Yes	1	No

Bits	Name	Description
0…13	x	2's complement integer coordinate
14,15	Reserved	
16…29	y	2's complement integer coordinate
30,31	Reserved	

Notes:	Holds the origin of the rectangle drawn with the DrawRectangle2D command. The coordinates have the window origin added to them as the message flows through this unit. (Rasterizer)

RenderGlyph

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x146	bitfield	No	1	No

Bits	Name	Description
0…6	Width	Specifies the width of the rectangle in pixels. Its range is 0…127.
7…13	Heigth	Specifies the height of the rectangle in pixels. Its range is 0…127.
14…22	AdvanceX	Specifies how much the glyph's position should change in X *before* the glyph is rendered. The offset is measured in pixels is held as a 9 bit 2's complement number.
23…31	AdvanceY	Specifies how much the glyph's position should change in Y *before* the glyph is rendered. The offset is measured in pixels is held as a 9 bit 2's complement number.

Notes:

RenderLine

Type	**Tag**	**Format**	**Context Sw**	**Datawords**	**Isochronous**
Core command	0x142	bitfield	No	1	No

Bits	**Name**	**Description**
0…3	A[4]	Defines the first vertex.
4…7	B[4]	Defines the second vertex.
8…11	Reserved	
12…15	P[4]	Defines the provoking vertex - i.e. the last vertex in the primitive - used by OpenGL to select which colour to use when flat shading.
16	ResetLineStipple	When this bit is set the line stipple pattern will be reset to the start before a line is drawn.
17	UseProvoking	When this bit is set the colour in the provoking vertex should be used.
18	FrontFacing	This bit, when set, indicates this primitive is front facing and if two sided lighting is enabled (in the Parameter Set Up Unit) then the colour is taken from the appropriate place.
19…20	PolygonOffsetMode[2]	This field defines the polgon offset mode. The options are: 0 = Reset 1 = Calculate 2 = CalculateApply 3 = Hold

Notes: This command communicate to the Primitive Set Up Unit what is to be rendered and initiates the rendering described in the data field. The data field encodes information about the line. The data field for the RenderPoint, RenderLine and RenderTriangle messages are similar.

RenderLine2D

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x145	fixed	No	1	No

Bits	Name	Description
0…13	X	2's complement integer end coordinate
14,15	Reserved	
16…29	Y	2's complement integer end coordinate
30,31	Reserved	

Notes: Holds the end coordinate of the line and causes the line to be drawn once the start coordinate has been loaded by the **LineStart** command. After the line has been drawn the **LineStart** register is updated with the end coordinate from this command.

RenderPoint

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x141	Bitfield	No	1	No

Bits	Name	Description
0…3	A[4]	Defines the first vertex.
4…11	Reserved	
12…15	P[4]	Defines the provoking vertex - i.e. the last vertex in the primitive - used by OpenGL to select which colour to use when flat shading.
16	Reserved	
17	UseProvoking	When this bit is set the colour in the provoking vertex should be used.
18	FrontFacing	This bit, when set, indicates this primitive is front facing and if two sided lighting is enabled (in the Parameter Set Up Unit) then the colour is taken from the appropriate place.
19…20	PolygonOffsetMode[2]	This field defines the polgon offset mode. The options are: 0 = Reset 1 = Calculate 2 = CalculateApply 3 = Hold
21…31	Reserved	

Notes: Defines the point to set up and draw.

RenderRectangle

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x144		No	1	No

Bits	Name	Description
0…3	VertexStore	Vertex coordinates
4…31	Reserved	

Notes:	This command starts rendering of the rectangle described in RasterPosRectangle. Vertex store identified by bits 0…3 is used for the vertex coordinates. (Geometry)

RenderText

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x2D0	Int	No	2	No

Bits	Name	Description
0…31	GlyphAddr	
32…63	RenderGlyph	

Notes:

RenderTriangle

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x143	Bitfield	No	1	No

Bits	Name	Description
0…3	A[4]	Defines the first vertex.
4…7	B[4]	Defines the second vertex.
8…11	C[4]	Defines the third vertex.
12…15	P[4]	Defines the provoking vertex - i.e. the last vertex in the primitive - used by OpenGL to select which colour to use when flat shading.
16	Reserved	
17	UseProvoking	When this bit is set the colour in the provoking vertex should be used.
18	FrontFacing	This bit, when set, indicates this primitive is front facing and if two sided lighting is enabled (in the Parameter Set Up Unit) then the colour is taken from the appropriate place.
19…20	PolygonOffsetMode[2]	This field defines the polgon offset mode. The options are: 0 = Reset 1 = Calculate 2 = CalculateApply 3 = Hold
21…31	Reserved	

Notes:	Defines the point to set up and draw.

RLCount

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x14D	Int	No	1	No

Bits	Name	Description
0…31	Count	

Notes:	This register holds the number of times the 32 bit run length data should be replicated

RLData

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x234	Int	Yes	1	No

Bits	Name	Description
0…31	Data	

Notes:	Holds the run length data to replicate

RouterMode

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x010	bitfield	Yes	1	Yes

Bits	Name	Description
0	Order	0 = TextureDepth 1 = DepthTexture
1…31	Reserved	

Notes:	32 bit post context data tags

RunPixelProg

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core command	0x0EF	Bitfield	Yes	1	Yes

Bits	Name	Description
0…6	run address	
7…14	run data	
15…19	pass number	
20…29	Reserved	
30	EnableData	
31	EnableRun	

Notes:	Starts a program depending on bitfield settings.

RunShadeProg

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core command	0x0EE	Bitfield	Yes	1	Yes

Bits	Name	Description
0…6	run address	
7…14	run data	
15…29	Reserved	
30	EnableData	1 = Set
31	EnableRun	1 = Set

Notes:

- Starts a program running at the address given by the least significant 7 bits of the data field providing bit 31 is set.
- If bit 31 is not set then this message does nothing.
- If bit 30 is set then the value in bits 7…14 is copied into the last global register so it can be used within a program

RunTextureProg

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core command	0xOED	Bitfield	Yes	1	Yes

Bits	Name	Description
0…6	run address	
7…30	Reserved	
31	EnableRun	1 = Set

Notes:

- Starts a program running at the address given by the least significant 7 bits of the data field providing bit 31 is set.
- If bit 31 is not set then this message does nothing.

SetPickResult

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core command	0x0E2	Tag	Yes	1	No

Bits	Name	Description
0	Flag	
1…31	Reserved	

Notes:	Updates the picking result flag with the least significant bit of the data field.

SetUpDerivatives

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x0F8	Bitfield	Yes	4	No

Bits	Name	Description
0…26	reduced float t1	1 bit sign, 6 bits exponent (with a bias of 31) and a 20 bit mantissa (with unspecified leading 1 as in IEEE format).
27…53	reduced float t2	1 bit sign, 6 bits exponent (with a bias of 31) and a 20 bit mantissa (with unspecified leading 1 as in IEEE format).
54…80	reduced float t3	1 bit sign, 6 bits exponent (with a bias of 31) and a 20 bit mantissa (with unspecified leading 1 as in IEEE format).
81…107	reduced float t4	1 bit sign, 6 bits exponent (with a bias of 31) and a 20 bit mantissa (with unspecified leading 1 as in IEEE format).
108…111	a	
112…115	b	
116…119	c	
120…123	P (provoking vertex)	
124	UseProvoking	
125…126	PolygonOffsetMode	
127	FrontFacing	

Notes:	• Defines the parameter gradients and which vertices are to be used in a primitive. It is consumed so that it is not sent to the Pixel Address Unit. Consists of post-context unit 128 bit tags. • The three vertices to use (out of the possible 16 vertex stores[2]) are given in the **SetUpDerivatives** command as is the provoking vertex. For lines or points (where there are less than 3 vertices) the missing vertex is a repeat of a real vertex and the t1…t4 value will be set up to do the correct calculation. (Parmeter_set_up_unit) • The **SetUpDerivatives** command invalidates any previous calculations and the next **Tile** command causes the enabled parameters to be calculated and sent on. This means that parameter set up is only done for primitives which are not scissor clipped out or rejected totally by the GID or depth test (assuming the router is switched appropriately). (Depth, Set Up Primitives..)

ShadeGlobal[0…7]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x060…0x067		Yes	1	yes

Bits	Name	Description

Notes:	These commands update the global registers. The registers are updated 32 bits at a time but are read by a program one byte at a time. Byte 0 (from the program) is the ls byte of PixelGlobal0. Byte 31 (from the program) is the ms byte of PixelGlobal7. Byte31 can be optionally updated by the **RunPixelProg** register.

[2] The message structure has provision for 16 vertex stores so an efficient vertex cache can be implemented local to where the calculations are done. This obviously comes with a gate cost so may be reduced later.

ShadeMode

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x01B	Bitfield	Yes	1	Yes

Bits	Name	Description
0	TileEnable	This bit, when set, enables a Tile message to start a program running.
1…7	TileAddrDefault[7]	This field holds the address of the program to run when a subtile is received (assuming it is enabled) when the prog field is 0. This field also holds the address of the program to run when the Texture Coordinate Unit is disabled.
8…14	TileAddrFirst[7]	This field holds the address of the program to run when a subtile is received (assuming it is enabled) when *progID* = 1
15…21	TileAddrMiddle[7]	This field holds the address of the program to run when a subtile is received (assuming it is enabled) when *progID* = 2
22…28	TileAddrLast[7]	This field holds the address of the program to run when a subtile is received (assuming it is enabled) when *progID* = 3.
29	PlaneOriginAtZero	This bit, when set, forces the plane equation origin to be at zero otherwise the plane origin is the coordinate of the first tile seen by this unit. This bit would normally be set when the Parameter Set Up Unit is not being used to set up the plane equations, such as for 2D operations.
30,31	Reserved	

Notes: **ShadeMode** controls the operation of the Shading unit. The Shading Unit is responsible for calculating fragment color. The color is normally a function of some iterated parameters, some constants and one or more sampled and filtered textures. P10 Shading includes the functionality found in several discrete units in earlier rasterizers incl. Colour DDA, Texture Composition, Texture Application, YUV conversion, Fog and Alpha Test.

ShadeProgramAddr

Type	Tag	Format	Context Sw	Datawords	Isochronous
Command	0x001	Int	Yes	1	Yes

Bits	Name	Description
0…6	ShadeAddress	

Notes:	Holds the address where subsequent ShadeProgramData registers will be loaded. The address is auto-incremented after every load.

ShadeProgramData

Type	Tag	Format	Context Sw	Datawords	Isochronous
Command	0x131	User Data	Yes	2	Yes

Bits	Name	Description
0…63	ProgrammeData	

Notes:	Holds the program data to write into the program memory (WCS). After receiving this data and doing the write the program address is incremented.

StencilData

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x017	Bitfield	Yes	1	Yes

Bits	Name	Description
0…7	Reference[8]	This field holds the value the stencil data from the local buffer is compared with. Before the comparison the reference data is masked by the CompareMask (also held in this register).
8…15	CompareMask[8]	This field holds the mask which is ANDed with the stencil data read from the local buffer and the reference stencil value before the comparison is done. A bit set in the mask allows the corresponding bits in the reference and local buffer stencil values to take part in the comparison operation.
16…23	WriteMask[8]	This field holds the mask used to only allow certain bits in the local buffer stencil field to be updated. A bit set in the mask allows the corresponding stencil bit in the local buffer to be updated.
24…31	Reserved	

Notes: These messages are passed through, but only after the cache has been flushed. This prevents a potential race condition in the GSD Unit from occuring.
State in these messages is used to control the address generation, reads and number of bytes accessed. (GSD)

StencilMode

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x016	Bitfield	Yes	1	Yes

Bits	Name	Description
0	Enable	When set, enables the stencil test and the replacement stencil value to depend on the outcome of the test (and depth test). Otherwise the test always passes and the stencil data in the local buffer is not changed.
1…3	DPpass[3]	These fields control how the stencil field is updated when the depth and stencil tests pass, when the depth test fails and stencil test passes, or when the stencil test fails. The options are: 0 = Keep (i.e. local buffer value not changed) 1 = Zero 2 = Replace with StencilData.Reference 3 = Increment (with saturation) 4 = Decrement (with saturation) 5 = Invert 6 = Increment (with wrapping) 7 = Decrement (with wrapping)
4…6	DPfail[3]	
7…9	Sfail[3]	
10…12	CompareFunction[3]	This field selects the compare function to use. The options are: 0 = Never 1 = Less 2 = Equals 3 = Less Equals 4 = Greater 5 = Not Equal 6 = Greater Equal 7 = Always The compare operation compares the stencil reference value against the source stencil value. If the compare function is 'Less' and the result is true then the reference value is less than the source value.
13	Present	This bit, when set, indicates the local buffer pixel format includes the Stencil field. The Stencil field is always the byte following the GID field or byte 0 if there is no GID field.
14…31	Reserved	

Notes: **StencilMode** controls the address generation, reads and number of bytes accessed.
StencilMode updates are passed on after the GSD cache has been flushed. (GSD)

Sync

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core command	0x180	Bitfield	No	1	Yes

Bits	Name	Description
0…29	SyncId	Identifier which drives the SyncId register.
30	Flush	When 0, a bus master flush is not requested. When 1, a bus master flush is requested and completion awaited.
31	Interrupt	When 0, a Sync interrupt is not requested. When 1, a Sync interrupt is requested.

Notes: Waits for all outstanding vertices to be processed and cache entries to be flushed back to memory (**LBAddress**) before being processed according to **HostOutMode** settings. If bit 31 of the input data is set then an interrupt is generated. (HostOut, LBAddress, Vertex Shader)

SyncWithVTG

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core command	0x174	bitfield	No	1	No

Bits	Name	Description
0	VTG head	
1…31	Reserved	

Notes: Suspends graphics processing until the selected VTG (held in bit 0) indicates it has reached its sync point. The sync point will typically be when is has swapped buffers and this command could be used to delay the clearing of the colour buffer. The suspension is done with a time out so the chip will not hang if the VTG has not been told to generate a sync signal. An interrupt is generated if a timeout occurs. (Vertex Shader)

TextureAddressMode[0…7]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x030 – 0x037	Bitfield	Yes	1	Yes

Bits	Name	Description
0…3	Width[4]	This field holds the width of the texture map in texels as a power of two. For a mip map this corresponds to the width of level 0 i.e. the highest resolution level. The maximum value of this is 13 if no border is present, or 12 if there is a border. Also note that if the map type is 3D then the limits are 10 and 9 respectively. If the texture is compressed then this field should hold the width after compression.
4…7	Height[4]	This field holds the height of the texture map in texels as a power of two. For a mip map this corresponds to the height of level 0 i.e. the highest resolution level. The maximum value of this is 13 if no border is present, or 12 if there is a border. Also note that if the map type is 3D then the limits are 10 and 9 respectively. This field should be set to zero for a 1D map. If the texture is compressed then this field should hold the height after compression.
8…11	Depth[4]	This field holds the depth of the texture map in texels as a power of two. For a mip map this corresponds to the depth of level 0 i.e. the highest resolution level. The maximum value of this is 10 if no border is present, or 9 if there is a border. This field should be set to zero for a 1D or 2D map.
12	Border	This bit, when set, indicates the texture map is surrounded by border texels. The true width, for example, will be $2^{width} + 2$.
13…14	MapType[2]	This field defines the map type. The options are: 0 = 1D map 1 = 2D map 2 = 3D map 3 = Cube map
15…17	Pitch[3]	This field holds the pitch - 1 between tiles of the texture map and is measured in planar byte tiles. Normally this is set to the depth of a texture tile (i.e. 3 for a texture in 8888 format). It allows one colour component to be extracted, for example, from a true colour texture, or the depth value from a buffer with GID and Stencil fields.

18	PowerOfTwoTexture	This bit, when set, indicates the texture is a power of two in size and mip mapping, border, cube processing, etc. should be done, when necessary. When this bit is clear the texture map width is not restricted to be a power of two but can be 0…2047 tiles wide. The 11 bit width is held as the concatenation of the Width, Height and Depth fields.
19	MipMap	This bit, when set, indicates there is a mip map chain (or set of 6 mip map chains for cube maps).
20…24	Format[5]	This field holds the format of the texture map. See below for a description.
25	ConvolutionBorder	This bit, when set, forces the border colour to be taken from the base address given in the next texture map, otherwise it will be taken from the first tile after the selected texture map.
26…31	Reserved	

Notes: Controls the way textures are located and formatted before passing the information on to the Secondary Texture Cache:

- Texture maps used for 3D are generally a power of two in size to allow ease of mip mapping and cube mapping and to allow the repeat, mirror and clamp wrap modes to work. Texture maps used during 2D operations are rarely a power of two in size, but don't need the mip map chains for better minification filtering.
- A 1D texture map occupies (width + 7) / 8 tiles. A 1D texture map can be *folded* into a 2D texture map to conserve memory and cache space.
- A *2D texture* map always has its width and height rounded up to the nearest tile so will occupy (width + 7) / 8 * (height + 7) / 8 tiles. The tiles are stored linearly in memory with one row following the previous.
- A *3D map* is a collection of 2D slices. The layout rules for a 2D slice are the same as for a 2D map. The slices are stored sequentially in memory and if there is a border then there are two extra border slices also stored.
- *Mipmap* levels are stored sequentially in memory with the highest resolution first. Each map level in the mip map chain is treated as a unique map as far as its storage requirements are concerned and will start in the first available byte tile after the previous (higher resolution) map level.
- A *cube map* is a collection of six 2D texture maps indexed by a face number. The face textures are stored sequentially in memory after each other with no gaps. A face texture may be a single texture or a mip map chain and they are always square, with or without a border.
- The convolution border colour is not normally intimately bound to the image data as a texture border colour is to a texture map. It needs to be defined separately and not as part of the texture map. (Texture Address)

Table 1-5 Supported Texture Formats

Format	Name	Width	R	G	B	A
0	A4L4	8	4@0	4@0	4@0	4@4
1	L8	8	8@0	8@0	8@0	255
2	I8	8	8@0	8@0	8@0	8@0
3	A8	8	0	0	0	8@0
4	A8L8	16	8@0	8@0	8@0	8@8
5	555	16	5@0	5@5	5@10	255
6	5551	16	5@0	5@5	5@10	1@15
7	565	16	5@0	6@5	5@11	255
8	4444	16	4@0	4@4	4@8	4@12
9	888	24	8@0	8@8	8@16	255
10	8888	32	8@0	8@8	8@16	8@24
11	YUV422	Compressed formats				
12	DXT1					
13	DXT2					
14	DXT3					
15	DXT4					
16	DXT5					

The DXT1…5 compressed formats are the Microsoft DX texture formats.

TextureBaseAddress[0…7]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x038 – ox03F	Int	Yes	1	Yes

Bits	Name	Description

Notes:

TextureCoordMode

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x019	Bitfield	Yes	1	Yes

Bits	Name	Description
0	FeedbackSource	This bit determines where the feedback data will come from. The options are: 0 = Download Image data 1 = Texture pipe
1	TileEnable	This bit, when set, enables a Tile message to start a program running.
2…8	TileAddr[7]	This field holds the address of the program to run when a Tile message is received (assuming it is enabled).
9	PlaneOriginAtZero	This bit, when set, forces the plane equation origin to be at zero otherwise the plane origin is the coordinate of the first tile seen by this unit. This bit would normally be set when the Parameter Set Up Unit is not being used to set up the plane equations, such as for 2D operations.
10…31	Reserved	

Notes: The Texture Coordinate Unit computes one or more perspectively correct texture coordinates for each fragment and the appropriate level of detail (lod) when mip mapping. In addition the texture coordinates can be perturbed by an earlier texture access (bump mapping) or treated a the index into a cube (cube mapping). Higher qualities of filtering are supported by way of anisotropic mip mapping and high order filters (bicubic for example). Texture coordinates can have 1, 2 or 3 components to support 1D, 2D or 3D texture maps. (Texture Coordinate)
The register is snooped by the Shading unit to find out if any texture data (or at least handshake) is imminent on the texture input port (Shading)

TextureGlobal[0…15]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x040 – 0x04F	User data	Yes	1	Yes

Bits	Name	Description

Notes: Updates the global registers available in program space to hold lod bias values, bump matrices, etc.

TextureGlobal[16…31]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x050 – 0x05F	User data	Yes	1	Yes

Bits	Name	Description
0…31	data	

Notes:	Updates the global registers available in program space to hold lod bias values, bump matrices, etc.

TextureIndexMipControl[0…7]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x008 – ox00F		Yes	1	Yes

Bits	Name	Description
0…11	MinLod[12]	This field holds the minimum level of detail (lod) value. Any input lod less than this will be clamped to this value. Its format is 4.8 unsigned fixed point.
12…23	MaxLod[12]	This field holds the maximum level of detail (lod) value. Any input lod greater than this will be clamped to this value. Its format is 4.8 unsigned fixed point.
24…27	BaseLevel[4]	This field holds the map level which should be treated as level 0. Set to 0
28…31	MaxLevel[4]	This field holds the map level which should be treated as last level in the mip map chain. Set to 14, the theoretical maximum.

Notes:

TextureIndexMode[0…7]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x028 – 0x02F	Bitfield	Yes	1	Yes

Bits	Name	Description
0…3	Width[4]	This field holds the width of the texture map as a power of two. The legal range of values for this field is 0 (map width = 1) to 13 (map width = 8192). If a border is present then the maximum value is 12, and for a 3D map it is 10 without a border, or 9 with a border. These later limits are not enforced by hardware.
4…7	Height[4]	This field holds the height of the texture map as a power of two. The legal range of values for this field is 0 (map width = 1) to 13 (map width = 8192). If a border is present then the maximum value is 12, and for a 3D map it is 10 without a border, or 9 with a border. These later limits are not enforced by hardware. This field should be set to 0 for a 1D map.
8…11	Depth[4]	This field holds the depth (i.e. number of slices) of the texture map as a power of two. The legal range of values for this field is 0 (map width = 1) to 10 (map width = 1024). If a border is present then the maximum value is 9. These later limits are not enforced by hardware. This field should be set to 0 for a 1D or 2D map.
12	Border	This bit, when set indicates there is a one texel border surrounding the texture map.
13…14	MapType[2]	This field selects the type of map and how many axis it has. The options are: 0 = 1D texture map 1 = 2D texture map 2 = 3D texture map 3 = Cube map
15…17 18…20 21…23	WrapU[3] WrapV[3] WrapW[3]	These fields selects how the u, v and w coordinate are wrapped to fit on the texture map. The options are: 0 = Clamp 1 = Repeat 2 = Mirror 3 = ClampEdge 4 = ClampBorder
24	MagnificationFilter	This field selects the magnification filter to use. The options are 0 = Nearest 1 = Linear

25…27	MinificationFilter	This field selects the minification filter to use. The options are 0 = Nearest 1 = Linear 2 = NearestMipNearest 3 = NearestMipLinear 4 = LinearMipNearest 5 = LinearMipLinear
28	FilterBank	When the filter mode is Nearest or Linear then this field will specify which filter bank to use in the Primary Texture Cache. By using alternating banks there will be less thrashing between the texture maps in the cache.
29…31	Reserved	

Notes:

TexturePlaneDX[0…7]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x118 – 0x11F	Float	Yes	4	Yes

Bits	Name	Description
0…31	Gradient	

Notes: Hold the four dx gradients for a texture coordinates in floating point format. (Texture Coordinate)

TexturePlaneDY[0…7]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x120 – 0x127	Float	Yes	4	Yes

Bits	Name	Description
0…31	Gradient	

Notes: Hold the four dy gradients for a texture coordinates in floating point format. (Texture Coordinate)

TexturePlaneScale[0…7]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x020 – 0x027	Bitfield	Yes	1	Yes

Bits	Name	Description
0…3	S scale	
4…7	T scale	
8…11	R scale	
12…15	Q scale	
16…31	Reserved	

Notes:	These messages hold with length of the axis of the texture map each component refers to. These are just used in lod calculation and are held as a power of two

TexturePlaneStart[0…7]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x128 – 0x12F	Float	Yes	4	Yes

Bits	Name	Description
0…127	data	4 texture coordinate start values as floating point numbers

Notes:	Holds the four starting values for texture coordinate parameter in floating point format. (Texture Coordinate)

TextureProgramAddr

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x000	bitfield	Yes	1	Yes

Bits	Name	Description
0…6	ProgramAddr	
7…31	Reserved	

Notes:	Holds the address where subsequent **TextureProgramData** registers will be loaded, and is auto incremented after every load.

TextureProgramData

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x130	Data	Yes	2	Yes

Bits	Name	Description
0…63	UserData	

Notes:	Holds the program data to write into the program memory (WCS). After receiving this message and doing the write the program address is incremented.

Tile

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x140	Bitfield	No	4	Yes

Bits	Name	Description
0…2	Reserved	
3…12	X	unsigned X coordinate
13…18	Reserved	
19…28	Y	unsigned Y coordinate
29…31	Reserved	
32…95	tile mask	
96	subsystemEnable	*SubsystemEnable* is the master enable for the whole LB subsystem and is derived from one place to ensure consistent operation
97	PixBufSelect OR destBuffer	*pixBufSelect* identifies which of the P double buffers to use when processing the pixel (Pixel unit) *destBuffer* (Depth Setup, LBAddress) identifies which of the double buffers to use when processing the pixel
98	newPrimitive	
99	sameTile	
100	aaEnable	
101…108	passNumber	
109…110	progID	
111	destCacheEnable	*destCacheEnable* prevents a destination cache line from being released or updated if one has not been allocated because only source reads have been done (Pixel unit)
112	endOfTile	*endOfTile* controls when the fragment buffer is swapped as it must be retained across multiple buffers. On output the tile mask field may be modified. (Pixel unit)
101…132	*Dzdx (DepthSetup only)*	*Float (output)*
133…164	*Dzdy (DepthSetup only)*	*Float (output)*
165…196	*zStart (DepthSetup only)*	*Float (output)*
197…228	*zRef (DepthSetup only)*	*Float (output)*

Notes:	Holds the detail on the current tile to process.

TimeStamp

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core command	0x187	Bitfield	No	1	Yes

Bits	Name	Description
0…14	StartScanline	
15…29	EndScanline	
30	Head	
31	Reserved	

Notes:	Holds the range of scanlines within which Isochronous stream commands are to be carried out. This register is ignored by Geometry functions.

Upload128

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x170	User data	No	4	No

Bits	Name	Description
0…127	Data	

Notes:	Transient data or commands - a generic message any unit can use to pass wide data back to the host (such as the current vertex state). (Geometry)

Upload32

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x171	User data	No	1	No

Bits	Name	Description

Notes:	a generic message any unit can use to pass 32 bit wide data back to the host (such as the line stipple state). (Host Out)

UploadDMA

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x1C9	Bitfield	No	2	No

Bits	Name	Description
0,1	ByteSwap	Byte swap: 0 = ABCD (no swap) 1 = BADC 2 = CDAB 3 = DCBA
2,3	PixelSize	Pixel size: 0 = 8-bit pixel 1 = 16-bit pixel 2 = 24-bit pixel 3 = 32-bit pixel
4	Protocol	Bus Protocol: 0 = PCI 1 = AGP
5	Enable	Unit enable: 0 = Input messages forwarded to next unit 1 = Input messages forwarded to bus master
6…19	Count	The upload count, in pixels – 1.
20…31	-	Reserved.
32…63	Addr	The upload address, in bus address space. If the address is not aligned on a pixel boundary then upload performance will be degraded.

Notes: Controls the upload DMA controller (GPIO Upload DMA)

UploadDMAControl

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x1C8		No	3	No

Bits	Name	Description

Notes:

UploadPixelData

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x172	Bitfield	No	3	No

Bits	Name	Description
0…63	Scan	8 byte planes on the current scanline
64…71	Mask	byte mask
72…73	PlaneNo	byte plane number.

Notes:	holds pixel data to upload. (Host Out)

UserClip[0…5]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x285 – 0x28A		Yes	4	No

Bits	Name	Description

Notes:	Hold the user clip plane (x, y, z, w) components for the 6 possible clipping planes.

UserFragData[0…15]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x090 – 0x09F	Int	Yes	1	Yes

Bits	Name	Description

Notes:	Hold download image data which will be loaded into the appropriate fragment's feedback register when enabled by the mode register.

UserFragData[16…31]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x0A0 – 0x0AF	Int	Yes	1	Yes

Bits	Name	Description

Notes:	Hold download image data which will be loaded into the appropriate fragment's feedback register when enabled by the mode register.

UserFragData[32…47]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x0B0 – 0x0BF	Int	Yes	1	Yes

Bits	Name	Description

Notes:	Hold download image data which will be loaded into the appropriate fragment's feedback register when enabled by the mode register.

UserFragData[48…63]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x0C0 – 0x0CF	Int	Yes	1	Yes

Bits	Name	Description
0…31	UserData	Download image data

Notes:	Hold download image data which will be loaded into the appropriate fragment's feedback register when enabled by the mode register.

UserScissor

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x251	Bitfield	Yes	1	No

Bits	Name	Description
0…13	x	
14,15	Reserved	
16…29	y	
30,31	Reserved	

Notes:	Holds the x and y coordinate of the user scissor rectangle region. The rasteriser processes tiles outside this region (so unwanted image or bitmask data will be clipped), but does not render pixels outside this area. A pixel is 'in' if min <= (x and y) < max This is enabled by a mode bit.

VertexBufferAddr

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x005	Int	Yes	1	No

Bits	Name	Description
0..7	Address	

Notes:	Holds the auto-incrementing address used to load Vertex Program data into the WCS

VertexCacheMode

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x1E3	bitfield	Yes	1	No

Bits	Name	Description
0…3	Type	Primitive Type: 0 = Null 1 = Points 2 = Lines 3 = LineLoop 4 = LineStrip 5 = Triangles 6 = TriangleStrip 7 = TriangleFan 8 = Quads 9 = QuadStrip 10 = Polygon 11 = Grid
4…7	GridWidth	Gris width in vertices
8	Provoking Vertex	Select OpenGL or D3D provoking vertex rules: 0 = OpenGL rule (last vertex in a primitive) 1 = D3D rule (first vertex in a primitive)
9…29	Reserved	
30	Enable	Unit enable: 0 = Disabled 1 = Enabled
31	Invalidate	Cache invalidated when set to 1, unmodified otherwise.

Notes:	(Vertex Cache Unit)

VertexData[0…15]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x190 – ox19F	Float	No	4	No

Bits	Name	Description

Notes:	Hold the parameter values as 4 floating point values. On input the tag size field is used to indicate a short form of the parameter and the missing components are set to their default values.

VertexDataBuffer[0..15]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x260-26f	Bitfield	Yes	2	No

Bits	Name	Description
0…31	Addr	Data buffer address, in 32-bit words.
32,33	ByteSwap	Byte swap: 0 = ABCD (no swap) 1 = BADC 2 = CDAB 3 = DCBA
34…39	DataSize	Data size, in 32-bit words – 1.
40…53	DataStride	Data stride, in 32-bit words – 1.
54…63	-	Reserved.

Notes: Vertex parameters can be grouped into vertex elements and read from multiple data buffers. The maximum number of buffers supported for this implementation is 16. The address of each buffer is defined in the **VertexDataBuffer** registers. (GPIO Vertex Data)

VertexDataBufferEnable

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x1E1	Bitfield	Yes	1	No

Bits	Name	Description
0…15	Enable	Data buffer enable mask. This indicates which data buffers are used (if index buffers are not used), or which data buffers are used by the lowest numbered index buffer (otherwise).
16…18	CacheMode	Tile cache mode: 0 = 1×16 1 = 2×8 2 = 4×4 3 = 8×2 4 = 16×1 5 = 16×1 6 = 16×1 7 = 16×1
19…31	-	Reserved.

Notes:	To help save memory bandwidth when reading single elements from memory, an internal 16-entry tile cache exploits the locality of elements within groups of tiles. The tile cache is configured and invalidated by the **VertexDataBufferEnable** command. The *CacheMode* field specifies how the 16-entry tile cache is to be shared among the 16 data buffers as shown in the table below. Each buffer is individually enabled according to the 16-bit mask supplied by the **VertexIndexBuffer**[0..15] messages (in Index Lookup Mode) or **VertexDataBufferEnable** message (in Data Lookup Mode). (GPIO Vertex Data)

Data Lookup Mode can read element arrays from memory ("fast path") if:

- only one buffer is enabled, and
- the element size is equal to the stride.

Otherwise Data Lookup Mode can only read single elements from memory ("slow path").

CacheMode	Sharing	Use With …
0 (1×16)	Buffers 0-15 share entries 0-15	1 indexed buffer
1 (2×8)	Buffers 0,2,4,6,8,10,12,14 share entries 0-7 Buffers 1,3,5,7,9,11,13,15 share entries 8-15	2 indexed buffers
2 (4×4)	Buffers 0,4,8,12 share entries 0-3 Buffers 1,5,9,13 share entries 4-7 Buffers 2,6,10,14 share entries 8-11 Buffers 3,7,11,15 share entries 12-15	3-4 indexed buffers
3 (8×2)	Buffers 0 & 8 share entries 0-1 Buffers 1 & 9 share entries 2-3 Buffers 2 & 10 share entries 4-5 Buffers 3 & 11 share entries 6-7 Buffers 4 & 12 share entries 8-9 Buffers 5 & 13 share entries 10-11 Buffers 6 & 14 share entries 12-13 Buffers 7 & 15 share entries 14-15	5-8 indexed buffers
4 (16×1)	Buffer *n* uses entry *n* (no sharing)	9-16 indexed buffers
5 (16×1)	Buffer *n* uses entry *n* (no sharing)	Reserved
6 (16×1)	Buffer *n* uses entry *n* (no sharing)	Reserved
7 (16×1)	Buffer *n* uses entry *n* (no sharing)	Non-indexed buffers

VertexDataBufferLookup

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x1C4	Bitfield	No	2	No

Bits	Name	Description
0…31	First	The first element to read in each buffer.
32…61	Count	The number of elements to read in each buffer.
61…63	Reserved	

Notes:	In Data Lookup Mode, the offset of the first element to read and the number of elements to read are given by the **VertexDataBufferLookup** message. This message then triggers the DMA. (GPIO Vertex Data)

VertexDataBufferLookupPacked

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x1C5	Bitfield	No	1	No

Bits	Name	Description
0…15	First	The first element to read in each buffer.
16…31	Count	The number of elements to read in each buffer.

Notes:	In Data Lookup Mode, the offset of the first element to read and the number of elements to read are given by the VertexDataBufferLookup message. This message then triggers the DMA. (GPIO Vertex Data)

VertexDataByte[0…15]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x1A0 – 0x1AF	Float	No	1	No

Bits	Name	Description
0…31		

Notes:	Hold the parameter values as 4 unsigned byte values.

VertexGridLookup

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x1C3	bitfield	No	1	No

Bits	Name	Description
0…15	First	The first column to read in each row.
16…19	Count	The number of columns to read in each row. This must be in the range 2–15.
20…31	Reserved	

Notes:

VertexGridSize

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x1E5	bitfield	No	1	No

Bits	Name	Description
0…15	Width	Grid width – 1
16…31	Height	Grid height – 1

Notes:

VertexIndex

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x1C2		No	1	No

Bits	Name	Description
0…31	Index	The element to read in each buffer.

Notes: In Index Lookup Mode, the association of indices to index buffers is given by the 16-bit mask supplied by the **VertexIndexBufferEnable** message, and the index of each element to read is given by the **VertexIndex** message. This message then triggers the DMA. (GPIO Vertex Data) (GPIO Vertex Cache)

VertexIndexBounds

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x253		Yes	2	No

Bits	Name	Description
0…31	Base	Index base
32…63	Count	Index count

Notes:	Index values supplied by VertexIndex messages are checked against base and count values given by the **VertexIndexBounds** message. If the check fails then an Index error signal is asserted and the remaining **VertexIndex** messages in the sequence are discarded. (GPIO Vertex Data)

VertexIndexBuffer[0…15]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x270 – 0x27F	Bitfield	Yes	2	No

Bits	Name	Description
0..31	Addr	Index buffer address, in 32-bit words.
32,33	ByteSwap	Byte swap: 0 = ABCD (no swap) 1 = BADC 2 = CDAB 3 = DCBA
34,35	IndexSize	Index size: 0 = 8-bit index 1 = 16-bit index 2 = 32-bit index 3 = Reserved (32-bit index)
36…47	-	Reserved.
48…63	Enable	Data buffer enable mask. This indicates which data buffers are used by this index buffer.

Notes:	Each buffer is individually enabled according to the 16-bit mask supplied by the **VertexIndexBuffer**[0..15] messages (in Index Lookup Mode) or **VertexDataBufferEnable** message (in Data Lookup Mode).. (GPIO Vertex Data)

VertexIndexBufferEnable

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x1E2		Yes	1	No

Bits	Name	Description
0…15	Enable	Index buffer enable mask.
16…31	Reserved	

Notes:	In Index Lookup Mode, the association of indices to index buffers is given by the 16-bit mask supplied by the **VertexIndexBufferEnable** message, and the index of each element to read is given by the **VertexIndex** message. This message then triggers the DMA. (GPIO Vertex Data)

VertexIndexBufferLookup

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x1C1	bitfield	No	2	No

Bits	Name	Description
0…31	First	The first index to read in each buffer. This is scaled by the index size to give a byte offset, which is then added to the buffer address.
31…63	Count	The number of indices to read in each buffer. This is scaled by the index size to give a byte count

Notes:

VertexParameterEnable

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x1E0		Yes	1	No

Bits	Name	Description
0…15	Enable	Parameter enable mask.
16…31	Reserved	

Notes:	Each vertex parameter is individually enabled by setting the corresponding bit. Pre-context 32 bit tags. (GPIO Vertex Data)

VertexParameterMsg[0…15]

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x1F0 – 0x1FF	Bitfield	Yes	1	No

Bits	Name	Description
0..9	Tag	Parameter message tag.
10,11	Reserved	Reserved for future tag expansion.
12,13	Size	Parameter size, in 32-bit data words – 1.
14	Reserved	
15	Send	When 0, the parameter is skipped. When 1, the parameter is sent.
16…31	Reserved	.

Notes:

- Each API vertex is characterized by a number of parameters. For OpenGL, parameters include the vertex coordinates, RGBA colour, surface normal, texture coordinates, and polygon edge flag. For DX7, parameters include the position, normal, diffuse colour, specular colour, and texture coordinates.
- Each parameter contains 1 to 4 32-bit words. These will usually be IEEE floating-point values. However, apart from the size, this unit places no interpretat.
- Each parameter is individually enabled according to the 16-bit mask supplied by the VertexParameterEnable message.

(GPIO Vertex Data)

VertexProgramAddr

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x003		Yes	1	No

Bits	Name	Description
0…7	ProgramAddr	
8..31	Reserved	

Notes: Holds the address where subsequent **VertexProgramData** registers will be loaded. The address is auto-incremented after every load.

VertexProgramData

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x133	User data	Yes	3	No

Bits	Name	Description
0…95	UserData	

Notes: Holds the program data to write into program memory (WCS). After receiving the data and writing it, the program address is incremented.

VertexShadingMode

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x210	bitfield	Yes	1	No

Bits	Name	Description
0…3	TriggerParameter[4]	This field holds the parameter which should be used as the trigger parameter. This will typically be the vertex position when in Begin/End paradigm, or the last parameter in the vertex array (per vertex).
4…11	ProgramAddr[8]	This field holds the address of the vertex shading program to run to transform, light, etc. the input vertices.
12	EyeVertexPresent	This bit, when set, will cause the parameter selected by the EyeVertexParameter field to be used as the eyeVertex for user plane clipping in the Geometry Unit.
13	UserOutcodePresent	This bit, when set, will cause the parameter selected by the SpecialParameter field to be used as the user clipping planes outcode value for user plane clip testing in the Cull Unit.
14…17	EyeVertexParameter[4]	This field identifies the Vec4 output parameter register to be used as holding the eyeVertex.
18…21	SpecialParameter[4]	This field identifies the Vec4 output parameter register to be used as holding UserClipOutcode (in the x component).
22…31	Reserved	

Notes: Defines the trigger parameter for this unit (Vertex Shading, Current Parameter))

ViewPortOffset

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x284	Float	Yes	3	No

Bits	Name	Description
0…31	OffsetX	
32…63	OffsetY	
64…95	OffsetZ	

Notes:	Viewport offset factor for the x, y and z directions as floating point numbers. This is used during viewport mapping after clipping has taken place. Only 3 significant words. (Geometry)

.

ViewPortScale

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x282	Float	Yes	3	No

Bits	Name	Description
0…31	ScaleX	
32…63	ScaleY	
64…95	ScaleZ	

Notes:	Viewport scaling factor for the x, y and z directions as floating point numbers. This is used during viewport mapping after clipping has taken place.. Only 3 significant words. (Geometry)

VisRect

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x250	Bitfield	Yes	2	No

Bits	Name	Description
0…13	min x	
14,15	Reserved	
16…29	min y	
30,31	Reserved	
32…45	max x	
46,47	Reserved	
48…61	max y	

Notes: Holds the x and y coordinate of the visible rectangle region the rasteriser will stay within. This would typically be the overlap of the screen rectangle and the window rectangle (or viewport). The coordinates are held as unsigned integers: A pixel is 'in' if min <= (x and y) < max

VTGCommand

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x173		No	1	No

Bits	Name	Description

Notes: Holds the serial command stream to pass on to the VTG. No interpretation of the data is done and it is serialised into an 8 bit wide FIFO. (Host_Out)

WaitForCompletion

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core command	0x181	bitfield	Yes	1	Yes

Bits	Name	Description
0,1	UnitName	0 = Rasteriser 1 = Rectangle Rasteriser 2 = GPIO 3 = none
2…31	Reserved	

Notes:	This message causes the rasterizer (if selected) to suspend all processing until a Completion signal has been received from the Host Out Unit. The **WaitForCompletion** command is forwarded on immediately by the rasteriser, but is delayed by any unit which can write to memory until all outstanding writes have completed. This allows gross synchronisation to be done between different parts of the core such as making sure an edit to a texture map (via the Pixel Unit) is in memory before the texture map is referenced by the Texture subsystem. (Rasterizer) When the tag reaches the Host Out Unit it releases the rasterizer, which will have stalled after sending the command into the message stream.

WindowOrigin

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x205	Fixed	Yes	1	No

Bits	Name	Description
0…13	x	
14,15	Reserved	
16…29	y	
30,31	reserved	

Notes:	Holds the window origin coordinate added to all primitives (except **RenderLine2D**) to convert a window relative coordinate to a screen relative 2's complement coordinate.

WriteCurrent

Type	Tag	Format	Context Sw	Datawords	Isochronous
Core control	0x1BF		No	1	No

Bits	Name	Description
0…3	Parameter	
4…31	Reserved	

Notes:	Writes the selected parameter (in the lower 4 bits) to the Coefficient Memory in the Vertex Shading Unit.

1.2 Programmable Registers

The Vertex Shading T&L unit is programmable, as are the Texture Coordinate, Shading (or Primitive Coloring), Pixel Address and Pixel Units in the graphics core. Of these, each functional group contains both programmable and fixed-function registers.

P10 supports hardware context switching which maintains the current chip state in memory. Switches occur automatically on GPIO circular buffer events (15us) and Isochronous evnts (3us) triggered by timestamps based on VTG# and scanline range.

Each programmable unit has its own assembler, disassembler, instruction set and interfaces. The Assemblers produce "C" array files with unsigned integers for inclusion in a compilation. It is also possible to generate these "on the fly" using supplied library function files. Each programmable unit has full assembler/disassembler user documentation:

The assembled instructions behave as if they complete in one (1) cycle but for floating point operations this is unusual. However the hardware automatically stalls to give the correct behavior. Understanding stall behavior and recovery is important to efficient use of the P10 chip.

All memory accesses are virtual and unified (command and vertex buffers, depth, colors) whether on- or off-card). On-card bandwidth is app. 16Gb/s (250MHz memory), while host bandwidth is constrained by the AGP rate – typically 1Gb/s for AGP 4x.

Most programs are generated algorithmically with some hand polishing for frequently-used routines. Programs run when a **Tile** command or **Run*prog** command is received by the appropriate unit.

This section does not provide programming examples or suggestions. For more detailed information on the use of programmable registers see the *Miranda P10 Programmers Guide*.

1.2.1 Vertex Shading Unit (T&L)

The DX8 vertex shading language has only one type of data – a four component floating point vector ("Vec4"). The instruction set is very much geared to vector operations. Scalar operations are supported by promoting one component to a vector and then completing the vector operation as normal. Operations like dot products between two vectors are also supported as well some scalar-only operations (e.g. reciprocals and inverse square roots). Each vector operation should take only one cycle.

Each vertex is processed independently and no connectivity knowledge is available. Effectively, a vertex is simply a unit of work. So for multi-pass processes it may be necessary to use **WaitForCompletion** to ensure pass 1 finishes before pass 2 starts.

The vertex shader program can be up to 128 instructions long.

Vertex Shading allows T&L programs for OpenGL Basic Transforms, Directional Lights, Material, Projection and Viewport Mapping and Tessellation, Transform Coding (e.g. for IDCT MPEG decoding) and color space conversion.

1.2.1.1 Resources

Constant registers ("coefficient memory") hold long-term data such as matrices, lighting and material parameters.. The programs can only read this memory which has a minimum size of 96 Vec4 words.

The ***vertex registers*** hold vertex parameter data likely to be updated frequently. It has a minimum size of 16 Vec4 words and is read only from the program.

The ***scratch registers*** hold the temporary working variables. They have three read ports[3] and one write port and support masked writes to address individual components. The registers hold 16 Vec4 entries and the read and write addresses are encoded in the instruction. There are scratch registers for 64 floating point numbers and program storage for 256 instructions.

The ***ALU*** is basically a three input Vec4 multiplier/adder with instructions like add, subtract, multiply, multiply add, dot product, etc. Scalar instructions include reciprocal, inverse square root, log2 and antilog2. The input components can be swizzled to reorder or replicate components to allow scalar operations.

Updates to the scratch registers and the output registers can be ***masked*** so any combination of the 4 components in the vector can be written.

Starting any program triggers a ***watchdog timer***. The watchdog will time out after 4096 cycles - if the program hasn't finished normally by then it is terminated (by an interrupt). This mechanism prevents a faulty program from hanging the unit (and hence chip).

[3] There is one instruction (Multiply add) which could use the three ports but most of the example programs only need two vectors to come from the scratch memory, so it would be possible to limit this to a two read port device and catch any three port read access in a user program (during translation to our microcode format) and use two instruction with some temporary storage to implement this operation.

1.2.1.2 Vertex Shader Instruction Set

Bits	Name	Description
0…4	OpCode	This field holds the ALU operation. See later for a description.
5…6	VectorCount	This field holds the number of components in the vector. The options are: 0 = one component vector (i.e. a scalar) 1 = two component vector 2 = three component vector 3 = four component vector
7…16	CoeffAddr	This field selects the float to read from the coefficient memory. The address is modified by the CoeffAddrBase and CoeffDataType fields.
17…22	InVertexAddr	This field selects the float to read from the input vertex registers. The address is modified by the InVertexAddrBase and InVertexDataType fields.
23…28	ScrAddrA	This field selects the float to read from the scratch register file. This value is srcA data. The address is modified by the ScrAddrBaseA and ScrDataTypeA fields.
29…34	ScrAddrB	This field selects the float to read from the scratch register file. This value is srcB data. The address is modified by the ScrAddrBaseB and ScrDataTypeB fields.
35…36	ArgA	This field selects the argA input to the ALU. The options are: 0 = coeff data 1 = input vertex data 2 = srcA from the scratch register file 3 = srcB from the scratch register file
37…38	ArgB	This field selects the argB input to the ALU. The options are: 0 = coeff data 1 = input vertex data 2 = srcA from the scratch register file 3 = srcB from the scratch register file
39…40	ArgC	This field selects the argC input to the ALU. The options are: 0 = coeff data 1 = input vertex data 2 = srcA from the scratch register file 3 = srcB from the scratch register file

Bits	Name	Description
41…42	ModA	This field defines how argA is modified before going into the ALU. The options are: 0 = pass 1 = negate 2 = absolute 3 = clamp to zero if negative
43…44	ModB	This field defines how argB is modified before going into the ALU. The options are: 0 = pass 1 = negate 2 = absolute 3 = clamp to zero if negative
45…46	CoeffAddrBase	This field defines how the coefficient address is generated. The options are:: 0 = relative (i.e. base + CoeffAddr) 1 = absolute (i.e. CoeffAddr) 2 = indirect (i.e. addressReg + CoeffAddr) 3 = circular addr = coeffBase + coeffAddr if (addr > coeffEnd) addr = coeffOrigin + addr - coeffEnd
47	CoeffDataType	0 = Scalar1 = Vector
48	InVertexAddrBase	0 = Relative 1 = Absolute
49	InVertexDataType	0 = Scalar1 = Vector
50	SrcAddrBaseA	0 = Relative 1 = Absolute
51	ScrDataTypeA	0 = Scalar1 = Vector
52	SrcAddrBaseB	This field defines how the scratch regiseter B address is generated. The options are: 0 = relative (i.e. base + ScrAddrB) 1 = absolute (i.e. SrcAddrB)
53	SrcDataTypeB	This field defines the data type. The options are: 0 = scalar 1 = vector

Bits	**Name**	**Description**
54…61	DestAddr	This field holds the address to update with the results of an ALU operation. The address (after modification by the *DestAddrBase*) is decoded into the following ranges: 0…63 = scratch register 64…95 = *ColourA[r, g, b, a]*…*ColourH[r, g, b, a]* 96…127 = *TextureCoordH*[s, t, r, q]… 128…130 = window coordinate 131 = homogenous W 132 = address register 133…256 = no write Note the *ColourA*…*ColourH* parameters are automatically clamped when used downstream. The interpretation of the ColourA…ColourH and TextureCoordA…TextureCoordH values is down to the programs running in the Texture Coordinate Unit and the Shading Unit.
62	DestAddrBase	This field defines how the destination address is generated. The options are: 0 = relative (i.e. base + DestAddr) 1 = absolute (i.e. DestAddr)
63	DestDataType	This field defines the data type. The options are: 0 = scalar 1 = vector
64…67	Sequencer	This field holds the sequencer operation. See later for a description.
68…76	SeqData	This field holds data mainly for sequencer related operations such as jump or subroutine addresses, loop counter values. It can also supply a value to be loaded or added to the base registers. Instruction addresses can be absolute (0) or relative (1) and this is controlled by the most significant bit.

Notes: Typical instructions are:

```
Reg[0+] = Add3 ( [coeff3+], in[8] );
        //Add scalar held in input vertex register 8 to
        //Vec3 starting at address3 in coefficient memory, then store result in
        //scratch register starting at 0.
 Reg[0] = Madd4 (coeff[4+], reg[8+], reg[0];
        //4-component dot product of the Vec4 in coeff memory at address 4 and
        //the Vec4 in Scratch register 8, add the result to Scratch register 0.
        //If Madd is changed to Dot then reg[0] is cleared first..
```

1.2.1.3 Vertex Shader ALU

The Arithmetic Logic Unit supports common operators such as Move, Add, Mul, MAdd, Min, Max, RSqrt, Fract etc. Some special purpose opcodes are also supported:

- ShiftSign – used to build up outcode for user clip planes
- Mantissa and Exponent – For DX Log instruction

The ALU instructions are shown below (d is destination, s0, s1 and s2 are the three sources).

Value.	Name	Stall	Description
0	Move	0	d = s0
1	Add	1	d = s0 + s1
2	MAdd	3	d = s0 * s1 + s2
3	Mul	1	d = s0 * s1
4	Min	1	d = Min (s0, s1)
5	Max	1	d = Max (s0, s1)
6	SLT	1	if (s0 < s1) d = 1.0 else d = 0.0
7	SGE	1	if (s0 >= s1) d = 1.0 else d = 0.0
8	Fract	1	d = fractional part of s0
9	Trunc	0	d = integer part of s0 (as a floating point number)
10	Dot	3	d = s0 * s1 for first component, else d = s0 * s1 + s2
11	ShiftSign	0	d = s0 << 1 | s1.sign allows user clip outcode to be build up
12	Recip	8[4]	d = 1.0 / s0, returns maximum positive number if s0 = 0.0
13	Div	8	d = s1 / s0, returns maximum positive or negative number if s0 = 0.0
14	RSqrt	0	d = 1.0 / sqrt (s0) (10 bits precision)
15	ALog	1	d = 2^{s0} (10 bits prescision)
16	Log	2	d = log2 (s0) (10 bit precision)
17	Exponent	2	d = IntToFloat (s0.e - 127)
18	Mantissa	2	d = IntToFloat (1.0 + s0.m)
19	IntToFloat	2	d = IntToFloat (s0)
20	FloatToInt	2	d = FloatToInt (s0)
21	HRecip	8	d = 1.0 / s0, returns 1.0 if |s0| < epsilon. epsilon = 2^{-120}

The ALU is pipelined and has a throughput of one operation per cycle with an anticipated latency of 3 cycles for the result.

1.2.1.4 Vertex Shader Sequencer Instructions

All sequencer operations are free. Miranda P10 includes flow control for subroutines:

[4] The Recip, Div and HRecip instructions will stall by the same amount if the next instruction also uses the multiplier.

Value	Name	Description
0	Inc	The next sequencer address is the current sequencer address + 1.
1	Jump	The next sequencer address is the address in the seqData field. The address in the seqData field is an absolute address if the most significant bit is clear, or a relative address if it is set.
2 3	Loop0 Loop1	The loop counter 0 or 1 is loaded with the contents of the seqData field. The maximum loop count is 127 and is primarily intended for looping around lights. The next sequencer address is the current sequencer address + 1.
4 5	DJNZ0 DJNZ1	The loop counter 0 or 1 is decremented and if the result is zero the next sequencer address is the current sequencer address + 1, otherwise the address in the seqData field is used as the next sequencer address. The seqData address can be absolute or relative.
6	Call	The current address + 1 is pushed on to the return stack and the next sequencer address is the address in the seqData field. The stack is only four deep and there is no protection against overflow. The seqData address can be absolute or relative.
7	Return	The next sequencer address is taken from the return stack and the stack is popped. The stack is only four deep and there is no protection against underflow.
8	Stop	This terminates the program and implements the necessary handshaking to accept more vertex data and pass any results into the pipeline for culling and clipping.
9	IncCoeffBaseReg	The coeff address register used for relative addressing has the sequencer data field added to it. The sequencer data is sign extented before the addition. The next sequencer address is the current sequencer address + 1.
10	LoadCoeffBaseReg	The coeff address register used for relative addressing has the sequencer data field loaded into it. The sequencer data is first multiplied by 2 before loading. The next sequencer address is the current sequencer address + 1.
11	LoadCoeffOriginReg	The coeff origin register used for circular addressing has the sequencer data field loaded into it. The sequencer data is first multiplied by 2 before loading. The next sequencer address is the current sequencer address + 1.
12	LoadCoeffEndReg	The coeff end register used for circular addressing has the sequencer data field loaded into it. The sequencer data is first multiplied by 2 before loading. The next sequencer address is the current sequencer address + 1.

1.2.2 Shader (Primitive Color) Unit

The Shading Unit is responsible for calculating the color of a fragment. The color is normally a function of some iterated parameters, some constants and one or more sampled and filtered textures. This unit replaces the functions previously carried out by the following units in earlier rasterizer chips:

- Color DDA Unit
- Texture Composite Unit
- Texture Application Unit
- YUV Unit
- Fog Unit
- Alpha Test Unit

P10 supports primitive color programming in two stages: Texture co-ordinate calculation and color calculation. Texture coordinate calculation is handled by the Texture Coordinate unit, described later. Color calculation handled by the Shader unit programme(s) combine texel data with interpolated values and constants. Calculations are done in fixed point signed 4.8 integers. The distinction between units is in the type of plane equation supported – textures and colors themselves are simply names, so for example a color can be perspective-corrected.

1.2.2.1 Resources

The unit supports 8 simultaneous textures with any combination of 1d, 2d, 3d and cube maps, however by using cube maps to hold mipmap chains, or 3d maps to hold bilinear 2d arrays, many more are possible. The limit is probably the number of possible floating point plane equations (32 each for texture and color).

1.2.2.2 Shader (Primitive Color) Instruction Set

Bits	Name	Width	Description
0…7	Aaddr	8	This field selects a byte or 12 bits to input into the A port of the ALU. The field is decoded into the following ranges: 0…31 Local register (1 off 32) 32…63 Texture register (1 off 32) 64…95 Plane equation (1 off 32) 96…127 Global register (1 off 32) 128…255 Constant field (only one) The common address range allows one parameterised subroutine to be used for all input sources. The address can be modified according to the AAddrMode field.
8…9	AAddrMode	2	This field defines how the address given in the AAddr field is to be modified. The options are: 0 Absolute (i.e. use value as given) 1 Absolute Component. Replace bottom two bits by component number. 2 ArgRelative (to value pushed on subroutine call). 3 ArgRelative Component. As for relative but with bottom two bits replaced by component number. The ArgRelative mode uses the top two bits of the AAddr field to select which of the four arguments should be used as the base address. The remaining AAddr bits are zero extended before the relative calculation is done.
10…12	AFormat	3	This field selects how the selected A data is converted from a byte to the 12 bit signed format. Data which is already in 12 bit format (i.e. from the local register file) is passed on unchanged. The options are: 0 MapToOne (if x == 255, y = 1.0 else y = 0.x) 1 Zero extend 2 Bias1 (if x == 255, y = 0.5 else y = x - 0.5) 3 Bias2 (if x == 255, y = 1.0 else y = (x - 0.5) * 2) 4 Bias8 (if x == 255, y = 4.0 else y = (x - 0.5) * 8) 5 Invert (y = ~x, zero extended) 6 Half (y = 0.5)

13…20	BAddr	8	This field selects a byte or 12 bits to input into the B port of the ALU. The field is decoded into the following ranges: 0…31 Local register (1 off 32) 32…63 Texture register (1 off 32) 64…95 Plane equation (1 off 32) 96…127 Global register (1 off 32) 128…255 Constant field (only one) The common address range allows one parameterised subroutine to be used for all input sources. When a resource conflict occurs between the *AAddr* and *BAddr* the *AAddr* always wins and the value referenced by it will be used as the B value. The plane, global and constant fields use the same resources so are mutually exclusive across both addresses. The texture registers can only be used once in an instruction. The address can be modified according to the *BAddrMode* field. The *ArgRelative* mode uses the top two bits of the *BAddr* field to select which of the four arguments should be used as the base address. The remaining *BAddr* bits are zero extended before the relative calculation is done.
21…22	BAddrMode	2	This field defines how the address given in the *BAddr* field is to be modified. The options are: 0 Absolute (i.e. use value as given) 1 Absolute Component. Replace bottom two bits by component number. 2 *ArgRelative* (to value pushed on subroutine call). 3 *ArgRelative* Component. As for relative but with bottom two bits replaced by component number.
23…25	BFormat	3	This field selects how the selected B data is converted from a byte to the 12 bit signed format. Data which is already in 12 bit format (i.e. from the local register file) is passed on unchanged. The options are: 0 MapToOne (if x == 255, y = 1.0 else y = 0.x) 1 Zero extend 2 Bias1 (if x == 255, y = 0.5 else y = x - 0.5) 3 Bias2 (if x == 255, y = 1.0 else y = (x - 0.5) * 2) 4 Bias8 (if x == 255, y = 4.0 else y = (x - 0.5) * 8) 5 Invert (y = ~x, zero extended) 6 Half (y = 0.5)

26…31	Waddr	6	This field selects a register to write to. The address range is split up: 0…31 Local register (1 off 32) 32…63 C FIFO (1 off 8, but replicated 4 times) The address is modified by the *WAddrMode* field and the write action is qualified by the *WEMode* field. Writes to the C FIFO are automatically scaled and clamped to be in the range 0…255. The *WAddr* field also supplies the test condition used when the ALU operation is Sub* as follows: 0 = Never 1 = Less 2 = Equal 3 = Less Equal 4 = Greater 5 = Not Equal 6 = Greater Equal 7 = Always
32…33	WAddrMode	2	This field defines how the address given in the *WAddr* field is to be modified. The options are: 0 Absolute (i.e. use value as given) 1 Absolute Component. Replace bottom two bits by component number. 2 *ArgRelative* (to value pushed on subroutine call). 3 *ArgRelative* Component. As for relative but with bottom two bits replaced by component number. The *ArgRelative* mode uses the top two bits of the *WAddr* field to select which of the four arguments should be used as the base address. The remaining *WAddr* bits are zero extended before the relative calculation is done.
34…35	WEMode	2	This field defines the write action in the local register file. The options are: 0 = No write 1 = Unconditional write 2 = Write if flag bit is 0 3 = Write if flag bit is 1 Also forms bits 0…1 of arg D on a subroutine call when Op is Arg.
36…39	Op	4	See table below
40	Div2	1	This bit when set will divide the ALU output by 2, making use of the extra bit of internal precision on add, sub and mulS(which is not available easily if this is done as a separate instruction). For Saturate it selects between the ranges 0…1 (when set) and -1…1 (when clear). Also forms bits 2 of arg D on a subroutine call when Op is Arg.

41…42	FlagMode	2	This field determines how the status value generated by the ALU is combined with the value in the flag register. The options are: 0 = Hold 1 = Replace 2 = Replace with status AND flag 3 = Replace with status OR flag This field is also used to hold the value to load into the component register on a subroutine call if the CC field is also set. Also forms bits 3…4 of arg D on a subroutine call when Op is Arg.
43…45	Sequencer	3	This field controls the sequencer operations. The options are: 0 = Increment 1 = Jump 2 = JumpTrue 3 = JumpFalse 4 = Call 5 = Return 6 = Done 7 = DoneAnd
46	CC	1	This field selects which condition code the sequencer should test. The options are: 0 = AND of all flag bits from fragment array 1 = OR of all flag bits from fragment array If this bit is set on a Call then the FlagMode field is used to load the Component register used in the address modification process.
47…54	Constant	8	Holds a constant or jump address (absolute or relative).

1.2.2.3 Shader (Primitive Color) ALU

The ALU supports ADD and SUB with and without Carry or Saturation. Subroutines are supported with Args.

Number	Operation	Q	Notes
0	Add	Q = A + B	
1	AddC	Q = A + B + carry	Add with carry
2	AddS	Q = Min (A + B,0x7.ff)	Add with saturate
3	AddSC	Q = Min (A + B + carry, 0x7.ff)	
4	Sub	Q = A - B	
5	SubC	Q = A - B – carry	
6	SubS	Q = Max (A - B, -0x8.00)	
7	SubSC	Q = Max (A - B – carry, -0x8.00)	

8	MultU	Q = (A * B) >> 12	upper 12 bits
9	MultL	Q = (A * B)	lower 12 bits

10	MultS	Q = A * B
		Q = Min (Q, 0x7.ff)
		Q = Max (Q, -0x8.00)	

11	PassA	Q = A	
12	SelectA	Q = A if flag is true else Q = B	
13	SelectB	Q = B if flag is true else Q = A	

14	Saturate	if (Div2)
		Q = Max (A, 0)
		Q = Min (Q, 0x1.00)
		else
		Q = Max (A, -0x1.00)
		Q = Min (Q, 0x1.00)
			if div2 field is set
			Clamp to 0…+1 range
			else
			Clamp to -1…+1 range

15	Arg	Nop	Sets arg D from the WEMode, FlagMode and Div2 fields. No writes are done.

1.2.2.4 Shader (Primitive Color) - Sequencer Instructions

Name	Description
Increment	This causes the next instruction address to be current instruction address + 1.
Jump	This causes the next instruction address to be taken from the constant field in the instruction. The most significant bit of the constant field determines if the address is an absolute address (0) or a relative address (1). If it is a relative address then the value in the constant field is added to the current address.
JumpTrue	This causes the next instruction address to be taken from the constant field in the instruction if the selected condition (masked by the subtile mask) is true, otherwise the next instruction address is the current instruction address + 1. The address can be absolute or relative.
JumpFalse	This causes the next instruction address to be taken from the constant field in the instruction if the selected condition (masked by the subtile mask) is false, otherwise the next instruction address is the current instruction address + 1. The address can be absolute or relative.

Name	Description
Call	This causes the next instruction address to be taken from the constant field in the instruction and the current instruction address + 1 written to the return address stack. The address can be absolute or relative. The A, B and W addresses in effect at the time of the call are also pushed onto the stack. Subsequent addresses can be made relative to these pushed addresses to allow limited input parameters to subroutines without having to copy data into fixed places.
Return	This causes the next instruction address to be taken from the return stack. Calls and Returns do not need to be balanced as the stack is reset at the start of a program. The address stack is also popped.
Done	This causes the sequencer to halt and any handshaking with the double buffered texel registers and output colour FIFO to be done. The handshaking is only done in the case of a program initiated by the Tile message.
DoneAnd	This causes the sequencer to halt and any handshaking with the double buffered texel registers and output colour FIFO to be done. The handshaking is only done in the case of a program initiated by the Tile message. The fragment flags are anded with the subtile mask before the subtile is passed on.

1.2.3 Texture Coordinate Unit

The Texture Coordinate Unit computes one or more perspectively correct texture coordinates for each fragment and the appropriate level of detail (lod) when mip mapping. In addition the texture coordinates can be perturbed by an earlier texture access (bump mapping) or treated as the index into a cube (cube mapping). Higher qualities of filtering are supported by way of anisotropic mip mapping and high order filters (bicubic for example). Texture coordinates can have 1, 2 or 3 components to support 1D, 2D or 3D texture maps.

1.2.3.1 Resources

The Texture Coordinate ALU supports 32 floating point plane equations, 32 global registers and 16 scratch registers. The output register is 64 bits wide and normally holds 32bit RGBA plus optional depth, 16-bit color components etc. Program storage handles up to 128 instructions. There is also a flag register for conditional execution.

Bump Mapping, Displacement Mapping[5] and High Order or Multi-tap filters are supported using cubic functions to hold additional texel descriptors.

1.2.3.2 Texture Coordinate Instruction Set

The Texture Coordinate unit can run a First program, a Middle program and a Last program in the Shader unit (below), revisiting the same Shader data repeatedly. Each program can be run up to *n* times for *n* bits of source data. (For example, First = Zero Accumulator; Second = Add to the Accumulator; Third = Scale and Output the Accumulator.)

Bits	**Name**	**Width**	**Description**
0…2	SourceA[6] PlaneBase[0…2]	3	This field selects what data is placed on the inputs to the A input of the fragment array. The options are: 0 = the constant 0.0. 1 = the constant 1.0 2 = dpdx plane equation parameter. 3 = dpdy plane equation parameter. 4 = start plane equation parameter. The plane equation to use is held in the constant field. It also holds 3 of the 5 bits of the base address of the plane registers when this is enabled for loading.

6 Displacement mapping is a technique where a surface is tessellated and the tessellation vertices are displaced along the normal by an amount looked up from a displacement map. The displacement map is really a height field stored in a texture map. The displaced surface will naturally also perturb the normal from the base surface so the surface lighting will match the new geometry. The advantage displacement mapping has over bump mapping 1 is that the visibility along the silhouette edge follows the cues given by the lighting, but this comes at a very high cost as the tessellation triangles need to be very small - of the order of a few pixels in size.

Bits	Name	Width	Description
3…4	SourceB PlaneBase[3…4]	2	This field selects what data is placed on the inputs to the B input of the fragment array. The options are: 0 = the constant 0.0. 1 = the constant 1.0 2 = lower word of the global registers. 3 = upper word of the global registers. The global register to use is held in the constant field. It also holds 2 of the 5 bits of the base address of the plane registers when this is enabled for loading.
5…7	SourceC GRBase[0…2]	3	This field selects what data is placed on the inputs to the C input of the fragment array. The options are: 0 = the constant 0.0. 1 = the constant 1.0 2 = dpdx plane equation parameter. 3 = dpdy plane equation parameter. 4 = saved dpdx plane equation parameter. 5 = saved dpdy plane equation parameter. The plane equation to use is held in the constant field. It also holds 3 of the 4 bits of the base address of the global registers when this is enabled for loading (note these are addressed in pairs).
8…9	SourceD GRBase[3]	2	This field selects what data is placed on the inputs to the D input of the fragment array. The options are: 0 = the constant 0.0. 1 = the constant 1.0 2 = lower word of the global registers. 3 = upper word of the global registers. The global register to use is held in the constant field. It also holds 1 of the 4 bits of the base address of the global registers when this is enabled for loading (note these are addressed in pairs).

[6] We could combine the Source* and corresponding Arg* fields into a single field and save 5 bits on the instruction width. This two level decode at present separates the decode and muxing into a set which is outside of the fragment processors and a set which is inside the fragment processors. This split can naturally be done from a single combined field, but is less obvious.

Bits	Name	Width	Description
10…11	SourceScale NegateScale	2	This field selects what data is placed on the inputs to the Scale input of the fragment array. The options are: 0 = zero 1 = the plane equation scale field. 2 = the constant field (bottom 5 bits) The plane equation to use is held in the constant field. This may be overridden by the ArgScale field, and when it does it the least significant bit is then used to control the negation of the scale value.
12	SaveParameterGradients	1	This bit, when set, will copy the currently addressed plane equation dx and dy gradients into separate registers so they can be used by SourceC. This avoids needing two read ports on the plane equation storage during partial derivative calculations.
13…14	ArgA	2	This field selects what data is placed on the inputs to the A port of the ALU. The options are: 0 = register file A output port. 1 = SourceA input. 2 = divide result. 3 = feedback value (see Feedback* fields)
15…16	ArgB	2	This field selects what data is placed on the inputs to the B port of the ALU. The options are: 0 = register file A output port. 1 = register file B output port. 2 = SourceB input. 3 = X coordinate for current fragment.
17…18	ArgC	2	This field selects what data is placed on the inputs to the C port of the ALU. The options are: 0 = register file A output port. 1 = register file B output port. 2 = SourceC input.
19…20	ArgD	2	This field selects what data is placed on the inputs to the D port of the ALU. The options are: 0 = register file B output port. 1 = SourceD input. 2 = X coordinate for current fragment. 3 = current lod value

Bits	**Name**	**Width**	**Description**
21	ArgScale	1	This field selects what data is placed on the inputs to the Scale port of the ALU. The options are: 0 = SourceScale input. 1 = Scale register (loaded as a Special Function Operation).
22…25	AddrA TexID GRegDestRegTexID	4	This field provides the address for the register file A port. It also provides the 3 bit textureID (in the least significant bits) when a command is being sent. When *GRegDestRegTexID* (in the most significant bit) is set the *TexID* and *DestReg* will come from the global register selected by an earlier field rather than from the corresponding integer fields. The TexID is loaded from bits 0…2 of an even global register and the DestReg is loaded from bits 3…5 of the same even global register.
26…29	AddrB DestReg LoadShading	4	This field provides the address for the register file B port. It also provides the 3 bit destReg (in the least significant bits) when a command is being sent. The most significant bit is loadShading bit.
30…33	AddrW	4	This field provides the address for the register file W port.
34	Indirect	1	This bit, when set, causes the *AddrA* and *AddrB* fields to be treated as indirect offsets which are mapped to actual addresses via mapping information set up from cube sorting. It will also cause the values read from these two ports to be optionally negated and this is controlled by the cube sorting information.
35…38	ALUOp	4	See table below.
39…40	FlagMode	2	This field determines how the status value generated by the ALU is combined with the value in the flag register. The options are: 0 = Hold 1 = Replace 2 = Replace with status AND flag 3 = Replace with status OR flag
41…42	WEMode	2	This field defines the write action in the local register file. The options are: 0 = No write 1 = Unconditional write 2 = Write if flag bit is 0 3 = Write if flag bit is 1

Bits	Name	Width	Description
43…45	Output	3	This field controls writing to the output FIFO. The options are: 0 = No write. 1 = Write the output of the ALU (texelCoord) to addr 0. 2 = Write the output of the ALU (texelCoord) to addr 1. 3 = Write the output of the ALU (texelCoord) to addr 2. 4 = Write the output of the ALU (texelCoord) to addr 3. 5 = Write the lod and face number to addr 3. 6 = Write the Command to the output FIFO. 7 = Special Function Operation The command data is taken from Command, TexID, DestReg, LoadShading, FeedbackEnable and Prog fields.
46…47	FeedbackSize Command SpecialFunction[0…1]	2	When accessing the feedback register this field holds the size of the item to be read. The options are: 0 = 8 bits 1 = 16 bits 2 = 24 bits 3 = 32 bits When a command is being sent this field holds the command. The options are: 0 = Nop 1 = PassThrough2 2 = FilterTexture 3 = PassThrough4 When Special Function Operation this field (in conjunction with the next field) holds the command it should execute: 0 = Load ScaleReg from ALU output 1 = Load PlaneBaseReg (from PlaneBase fields) 2 = Load GRegBaseReg (from GRBase fields) 3 = Load PlaneBaseReg and GRegBase 4 = LoadQ2 from ALU output 5 = LoadMax from ALU output 6 = MergeMax from ALU output 7 = LoadMag from ALU output 8 = MergeMag from ALU output

Bits	Name	Width	Description
48…49	FeedbackPosition Prog SpecialFunction[2…3]	2	When accessing the feedback register this field holds the position of the data in the 32 bit word to extract. The options are: 0 = starts at bit 0 1 = starts at bit 8 2 = starts at bit 16 3 = starts at bit 24. When a command is being sent this field holds the program in the Shading Unit to run. The options are: 0 = default program (none if not end of subtile) 1 = start program 2 = middle program 3 = last program
50	FeedbackSignExtend EnableFeedback	1	When accessing the feedback register this bit causes the data (with width and position given by the previous two fields) to be sign extended (1) or zero extended (0) to 32 bits before being used. When a command is being sent this bit, when set, enabled the filtered texture data to be fed back to the Texture Coordinate Unit.
51…54	Sequencer	4	This field controls the sequencer operations. The options are: 0 = Increment 1 = Jump 2 = JumpTrue 3 = JumpFalse 4 = Call 5 = Return 6 = Done 7 = DoneAnd 8 = LoadCounter 9 = DJNZ 10 = WaitForFeedbackData 11 = FinishedWithFeedbackData 12 = KillFragment
55	CC LoopID Status	1	This field selects which condition code the sequencer should test. The options are: 0 = AND of all flag bits from fragment array 1 = OR of all flag bits from fragment array For the loop related operations this filed holds which counter to use. It also selects what status is generated from the ALU from the zero and negative flags. The options are: 0 = Zero 1 = Positive

Bits	Name	Width	Description
56…63	Constant	8	This field is used for several different purposes: For sequencer jump address the type of address is encoded in the most significant bit. Two types of addresses are supported: absolute address (0) or relative addresses (1). The bottom 7 bits hold the address or offset from the current address. For plane equation addresses the type of address is encoded in bit 5 (of the field). Two types of addresses are supported: • absolute address (0) or • relative addresses (1). The bottom 5 bits hold the address or offset from the PlaneBaseReg (loaded as a Special Function Operation). For global register addresses the type of address is encoded in bit 6 (of the field). Two types of addresses are supported: absolute address (0) or relative addresses (1). The bottom 5 bits hold the address or offset from the GRegBaseReg (loaded as a Special Function Operation).

1.2.3.3 Texture Coordinate ALU

The Texture Coordinate ALU includes special logic for LOD and cube mapping.

Number	Operation	Result	Notes
0	MAdd	r = a * b + c * d	Add, Mult and Pass are done by setting input values to 0.0 or 1.0 as necessary.
1	MSub	r = a * b - c * d	Sub is done by setting input values to 0.0 or 1.0 as necessary.
2	IntToFloat	r = Float (a)	a is treated as a signed integer.
3	FloatToInt	r = Integer (a)	
4	Fract	r = Fraction of (a)	
5	Min	if (a > b) r = b else r = a	
6	Max	if (a > b) r = a else r = b	
7	AMax	if (|a| > |b|) r = |a| else r = |b|	Can also be used for Abs

Number	Operation	Result	Notes
8	Wrap	f = Fract (a * b) set flags if: (a * b) > 1.0 (a * b) is odd (a * b) is < 0.0	The fixed point fract value and the two flags are combined into a 24 bit texture coordinate value intended to be passed to the Texture Index Unit.
9	Select	if (flag) r = a else r = b	Flag is taken from the flag register.

10	Div	divResult = a / b
		r = 0.0
			Asynchronous divide operation, result accurate to 24 bits, i.e. three levels of refinement. This will return a result after 7 cycles.

11	AnisoRatio	if (a / b <= 4.0) r = 2.0
		if (a / b < 8.0) r = 4.0
		if (a / b >= 8.0) r = 8.0
			The divide is just done by subtracting the exponents.

12	LoadDiv Result	divResult = a
		r = 0.0
			Used during cube sort to provide a third argument (S in this case).

13	CubeSort	Sort (a, b, c) and set up face number and indirect addressing. r = 0.0	

14	DivLP	divResult = a / b
		r = 0
			Asynchronous divide operation, result accurate to 14 bits, i.e. two levels of refinement. This will return a result after 5 cycles.

1.2.3.4 Texture Coordinate Sequencer Instructions

Name	Description
Increment	This causes the next instruction address to be current instruction address + 1.
Jump	This causes the next instruction address to be taken from the constant field in the instruction. The most significant bit of the constant field determines if the address is an absolute address (0) or a relative address (1). If it is a relative address then the value in the constant field is added to the current address.
JumpTrue	This causes the next instruction address to be taken from the constant field in the instruction if the selected condition is true, otherwise the next instruction address is the current instruction address + 1. The true address can be absolute or relative. The condition to test is either the AND of all the fragment flags (masked by the tile mask) or the OR of all the fragment flags.
JumpFalse	This causes the next instruction address to be taken from the constant field in the instruction if the selected condition is false, otherwise the next instruction address is the current instruction address + 1. The false address can be absolute or relative. The condition to test is either the AND of all the fragment flags (masked by the tile mask) or the OR of all the fragment flags.

Name	Description
Call	This causes the next instruction address to be taken from the constant field in the instruction and the current instruction address + 1 written to the return address stack. The call address can be absolute or relative.
Return	This causes the next instruction address to be taken from the return stack. Calls and Returns do not need to be balanced as the stack is reset at the start of a program.
Done	This causes the sequencer to halt and any handshaking done. The handshaking is only done in the case of a program initiated by the Tile message.
DoneAnd	This causes the sequencer to halt and any handshaking done. The handshaking is only done in the case of a program initiated by the Tile message. The fragment flags are anded with the tile mask before the tile mask is passed on.
KillFragment	The fragment flags are anded with the tile mask before the tile mask is passed on. This is the same as the DoneAnd command but program execution continues. This allows an early test to delete fragments from subsequent texture accesses, whereas the DoneAnd would only do it for the last texture access.
LoadCounter	This loads one of the two 8 bit counters from the constant field in the instruction.
DJNZ	This Decrements the counter and Jumps if the counter is Not Zero to the address in the constant field of the instruction, otherwise the next instruction address is the current instruction address + 1. The jump address can be absolute or relative. One of the two counters is used.
WaitForFeedback Data	This instruction stalls the program execution until all the feedback data has been received. If no feedback data has been requested then this instruction is ignored.
FinishedWith FeedbackData	This instruction is used to indicate the data in the Feedback registers has been finished with and any pending feedback data can be loaded. A count of the number of outstanding feedback requests is kept to try and prevent a lock up occurring.

1.2.4 Pixel Address

The Pixel Address Unit calculates the address where the data for the input tile(s) are stored in memory. This is more complicated than the address calculation for the LB pixel data because multiple addresses are needed and source reads may not be aligned to tile boundaries. The Pixel Addressing unit works with the Pixel Unit (below). The Pixel Unit only 'knows' about data from the Shader and memory. The Pixel Address unit controls access to additional memory data by the Pixel Unit.

The range of operations for which addresses are calculated can include:

- Simple: aligned destination reads and writes for regular 2D or 3D operations.
- Blits where the source tile is typically non-aligned and the destination tile may need to be read (e.g. because only a subset of the bits (in a pixel) are being blitted, or only a partial destination tile is being updated).
- Multibuffer updates.
- Accumulation buffer processing. This involves mixed 32 bit/64 bit buffer reads and writes. To accumulate the color buffer, for example, we store 8 successive planar byte tiles per accumulation tile. The actual accumulation and any scaling are actually done in the Pixel Unit, which expects to find the destination data in register0.
- Font processing: The font bitmask needs to be read and aligned to the destination rectangle.
- Convolution: This involves multiple (9 for a 3x3 kernel) non aligned tile reads and a single aligned destination tile read and/or write.
- Mipmapping
- Multi-sample antialiasing: The subpixel mask information is used to control the update of *n* subpixel color buffers, which are then averaged for display. The averaging can be done at video level, using a blit before display, or on the fly.[7]

1.2.4.1 Pixel Address Programming

Address generation is controlled by a user-defined program instead of a long and changing list of mode bits for various APIs and extensions. The program runs once per enabled buffer, typically Front/Back, Left/Right. There is also one global buffer. Usually the program reads 64 bits of Read data (Source and Destination) and writes 32 bits (Destination Write).

The general mode of operation is that an input Tile starts the address generation program running. It calculates all the addresses needed and issues them to the Pixel Cache Unit. If there are two or less tiles (up to 32 bpp) to read it forwards the Tile to the Pixel Unit where it will be paired up with the data (if any). If there are more than two tiles to read then multiple tiles are sent to the Pixel Unit to be matched up with sets of tile data from the cache. Different programs in the Pixel Unit can be run on the first set, middle sets and last set of tile data and a pass number is also provided.

Programming in this unit typically provides blitting, pattern fills and multi-sample antialiasing.

[7] Although the Pixel Unit could also perform this task it is far simpler for the Pixel Address Unit to use the coverage information to generate the tile masks for each buffer.

1.2.4.2 Resources

Program store: 32 instructions of 15 bits, loaded 2 per 32 bit word.

FBAddrInfo registers.

FBBaseAddr registers

FBBuffer **registers**

1.2.4.3 Pixel Address Instruction Set

The instruction format for **Copy, Add, Dec, LoadXYMask, LoadXYFromTile, LoadXY** and **SetTileMaskfromCoverage** is:

Bits	Name	Width	Description
0…3	opCode	4	This field selects the basic operation. The options are: 0 = Copy 1 = Add 2 = Dec 3 = LoadXYMask 4 = LoadXYFromTile 5 = LoadXY 6 = SendDestAddr 7 = SendSourceAddr 8 = SendTile 9 = JumpNotZero 10 = SendDestAddrAndTile 11 = SendSourceAddrAndTile 12 = SetTileMaskfromCoverage
4	argA	1	This field selects the source for argument A. The options are: 0 = working register given by the ra field 1 = addrInfo register given by the ra field
5…7	ra	3	This field selects the register in the working set or addrInfo to load the alu a argument from.
8	argB	1	This field selects the source for argument B. The options are: 0 = working register given by the rb field 1 = tileX or tileY register given by the rb field
9…11	rb	3	This field selects the register in the working set or tileX (0) or tileY (1) to load the alu b argument from.
12…14	rc	3	This field selects the register in the working set to update if required by the opCode.

The instruction format for **SendDestAddr, SendSourceAddr, SendTile, SendDestAddrAndTile** and **SendSourceAddrAndTile** is:

Bits	**Name**	**Width**	**Description**
0…3	opCode	4	This field selects the basic operation. The options are: 0 = Copy 1 = Add 2 = Dec 3 = LoadXYMask 4 = LoadXYFromTile 5 = LoadXY 6 = SendDestAddr 7 = SendSourceAddr 8 = SendTile 9 = JumpNotZero 10 = SendDestAddrAndTile 11 = SendSourceAddrAndTile 12 = SetTileMaskfromCoverage
4…6	buffer	3	This field selects which of the 5 buffers to use with SendDestAddr or SendSourceAddr opcodes. Buffers 0…3 are relative and are offset by the buffer number the program is being run on. Buffers 4…7 all map to buffer 4 and this is an absolute buffer so can be used to select global data to apply to each buffer such as font data.
7	puReg	1	This field selects the target register in the Pixel Unit register to update on a SendDestAddr or SendSourceAddr opcode.
8…9	progID	2	This field holds the tile program the Pixel Unit should run once all the data has been delivered from the cache. The options are: 0 = Only 1 = First 2 = Middle 3 = Last This field is only used by the SendTile opcode.
10…14	not used	5	

The instruction format for **JumpNotZero** is:

Bits	Name	Width	Description
0…3	opCode	4	This field selects the basic operation. The options are: 0 = Copy 1 = Add 2 = Dec 3 = LoadXYMask 4 = LoadXYFromTile 5 = LoadXY 6 = SendDestAddr 7 = SendSourceAddr 8 = SendTile 9 = JumpNotZero 10 = SendDestAddrAndTile 11 = SendSourceAddrAndTile
4	argA	1	This field selects the source for argument A. The options are: 0 = working register given by the ra field 1 = addrInfo register given by the ra field
5…7	ra	3	This field selects the register in the working set or addrInfo to test for zero.
8…12	jumpAddr	5	This field holds the address to jump to if the result of the test is true.
13…14	not used	2	

The Opcodes have the following effects:

Opcode	Syntax	Description
Copy	Copy (rc, ra)	rc = ra
Add	Add (rc, ra, rb)	rc = ra + rb
Dec	Dec (rc, ra)	rc = ra – 1
LoadXYMask	LoadXYMask (ra, rb)	xMask = ra, yMask = rb; the xMask and yMask registers are used in source address calculations to limit the range of x and y coordinates (if enabled by the buffer state).
LoadXYFromTile	LoadXYFromTile ()	x = tileX, y = tileY; x and y are registers used in the address computation
LoadXY	LoadXY (ra, rb)	x = ra, y = rb; x and y are registers used in the address computation
SendDestAddr	SendDestAddr (buffer, puReg)	Read, if necessary, an aligned tile and transfer to the Pixel Unit. Instruction fields provide which memory region (buffer) to read and/or write, and the Pixel Unit register to write to. The planar byte tile address is automatically calculated using the values in the x and y registers and the selected buffer parameters (base address, width, etc).

SendSourceAddr	SendSourceAddr (buffer, puReg)	Read a tile (maybe non aligned) and transfer to the Pixel Unit. Instruction fields provide which memory region (buffer) to read, and the Pixel Unit register to write to. The planar byte tile address is automatically calculated using the values in the x and y registers and the selected buffer parameters (base address, width, etc). Multiple reads may be initiated depending on the degree of missalignment and the tiles are automatically merged together.
SendTile	SendTile (progID)	The Tile message which caused the program to run is forwarded on and the pass number and double buffering information automatically appended. The tile program to run in the Pixel Unit is provided as part of the instruction. The options are: 0 = Only 1 = First 2 = Middle 3 = Last A SendTime (Only) or SendTile (Last) instruction will terminate the program.
JumpNotZero	JumpNonZero (ra, jumpAddr)	This tests ra against zero and if it is not zero then the program control is passed to the address held in the jump instruction.
SendDestAddrAndTile	Send…Tile (buffer, puReg, progID)	This instruction combines the SendDestAddr and SendTile actions and is only provided as an optimisation to allow a shorter program and less commands being sent to the cache.
SendSourceAddrAndTile	Send…Tile (buffer, puReg, progID)	This instruction combines the **SendSourceAddr** and **SendTile** actions and is only provided as an ptimisation to allow a shorter program and fewer commands being sent to the cache.
SetTileMaskfromCoverage	Set…Coverage()	This instruction replaces the tile mask used in all subsequent instructions with one extracted from the coverage information for this tile. The fragment position in the coverage mask is give by the pass number.

1.2.5 Pixel Unit

1.2.5.1 Resources

The Pixel Unit uses a programming syntax similar to "C" in several respects:

- It assumes predefined variables and arrays corresponding to the registers introduced earlier, e.g. A[], B[] and W[] are entries in the local register file for reading, reading and writing respectively.
- Conditional writes are shown within the [], e.g. W[2, flag==false] updates local register 2 only if the flag is false.
- ALU operations are treated as functions with the input arguments as parameters.
- A single instruction may run across several lines for clarity, and is closed with a semicolon;
- Labels are shown as a symbol name and semicolon
- The default sequencer operation is Increment. It is not normally specified.

Particularly in conjunction with the Pixel Address unit the Pixel Unit supports multi-pass programs capable of both conventional and exotic effects, including pattern fill, BLITs with XOR, dither, scaling during color buffer accumulation, convolutions, Radial gradient fill and Photoshop filters. It would, for example, be theoretically possible to implement the Game of Life in hardware. For details of programming implementation see the *Miranda P10 Programmers Guide*.

1.2.5.2 Pixel Programming Instruction Set

Bits	Name	Width	Description
0…2	Paddr	3	This field selects a byte of the eight P[0…7] register files to read. An additional address bit is supplied by the double buffer logic and the least significant bit is ignored when the ArgA field selects the R, G or B component of a 16 bit colour.
3…4	Faddr	2	This field selects a byte of the four F[0…7] register files to read. An additional address bit is supplied by the double buffer logic if double buffering, or by the FAddrExt bit if single buffering..
5…8	Aaddr	4	This field selects a byte from the local register file to read on port A.
9…12	Baddr	4	This field selects a byte from the local register file to read on port B.

Bits	Name	Width	Description
13…16	Waddr	4	This field selects a byte to write to from port W in the local register file. The write action is qualified by the WEMode field. The WAddr field also supplies the test condition used when the ALU operation is Sub* as follows: 0 = Never 1 = Less 2 = Equal 3 = Less Equal 4 = Greater 5 = Not Equal 6 = Greater Equal 7 = Always
17…18	WEMode	2	This field defines the write action in the local register file. The options are: 0 = No write 1 = Unconditional write 2 = Write if flag bit is 0 3 = Write if flag bit is 1
19…20	CAddr	2	This field selects which byte in the cache line to write to. The cache line is automatically provided. The write is qualified by the CWEMode and CWEMask fields.
21…23	CWEMode	3	This field defines the write action in the local register file. The options are: 0 = No write 1 = Unconditional write 2 = Unconditional write R 3 = Unconditional write G 4 = Unconditional write B 5 = ForwardToHostOut
24	CWEMask	1	This field determines how the cache writes should be masked. The options are: 0 = By the Tile Mask 1 = By the TileMask & flag register

Bits	Name	Width	Description
25…27	ArgA	3	This bit selects where the argument for the ALU port A comes from. The options are: 0 = Local register file Read A 1 = Local register file Read A 2 = External Data 3 = External Data 4 = Red Pixel Data (Pr) 5 = Green Pixel Data (Pg) 6 = Blue Pixel Data (Pb) 7 = Pixel data (P)
28	InvA	1	This bit, when set, inverts the A input to the ALU. If the CC field is zero then all the bits are inverted, otherwise just the ms bit is inverted. This can be used to convert a biased (by 128) number into a negative number.
29…30	ArgB	2	This bit selects where the argument for the ALU port B comes from. The options are: 0 = Local register file Read B 1 = Local register file Read B 2 = External Data 3 = Fragment data (F)
31	InvB	1	This bit, when set, inverts the B input to the ALU. If the CC field is zero then all the bits are inverted, otherwise just the ms bit is inverted. This can be used to convert a biased (by 128) number into a negative number.
32	ArgI	1	This bit selects where the argument for the ALU port I comes from. The options are: 0 = Local register file Read A 1 = Local register file Read B
33…37	Op	5	See table below.
38…40	FlagMode	3	This field determines how the status value generated by the ALU is combined with the value in the flag register. The options are: 0 = Hold 1 = Replace 2 = Replace with status AND flag 3 = Replace with status OR flag 4 = Replace with corresponding bit from Pixel Mask

Bits	Name	Width	Description
41…42	ExternalType	2	This field selects where the external data to the fragment array comes from. The options are: 0 = Global registers 1 = Constant (from the jump field) 2 = PerFragmentData 3 = DecodeFromExternalSource
43…47	ExternalSource	5	This field selects from one of 32 sets of global registers or one of 4 sets of per fragment data depending on the ExternalType field. When the ExternalType field is DecodeFromExternalSource this field is decoded as follows: 0 = GlobalIndex8 (GReg[passNumber]) 1 = GlobalIndex16U (GReg[passNumber*2+1]) 2 = GlobalIndex16L (GReg[passNumber*2]) 3 = Upper byte of fragment X coordinate (XU) 4 = Lower byte of fragment X coordinate (XL) 5 = Upper byte of fragment Y coordinate (YU) 6 = Lower byte of fragment Y coordinate (YL) 7 = Coverage
48…50	Sequencer	3	This field controls the sequencer operations. The options are: 0 = Increment 1 = Jump 2 = JumpTrue 3 = JumpFalse 4 = Call 5 = Return 6 = Done 7 = DoneAnd
51…52	CC	2	This field selects which condition code the sequencer should test. The options are: 0 = AND of all flag bits from fragment array 1 = OR of all flag bits from fragment array 2 = aaEnable bit from tile message.
53…60	Constant	8	Holds a constant or jump address.

Bits	Name	Width	Description
61	InvI	1	This bit, when set, inverts the I input to the ALU.
62	InvQ	1	This bit, when set, inverts the Q output of the ALU.
63	FAddrExt	1	This bit is used to extend the FAddr field by one bit when accessing 64 bit fragment data from the Shading Unit.

1.2.5.3 Pixel Programming ALU

Number	Operation	Q	Notes
0	Add	Q = A + B	
1	AddC	Q = A + B + carry	Add with carry
2	AddS	Q = Min (A + B, 255)	Add with saturate
3	AddSC	Q = Min (A + B + carry, 255)	
4	Sub	Q = A - B	
5	SubC	Q = A - B – carry	
6	SubS	Q = Max (A - B, 0)	
7	SubSC	Q = Max (A - B – carry, 0)	
8	MultU	Q = (A * B) >> 8	upper byte
9	MultL	Q = (A * B)	lower byte

10	Modulate	Q = B if A == 255, else
		Q = A if B == 255, else
		Q = (A * B) >> 8	

11	Lerp	Q = B if I == 255 else
		Q = A + (B – A) * I	

12	And	Q = A & B	
13	Or	Q = A | B	
14	Xor	Q = A ^ B	
15	Bit	flag = bit B of A	Uses ls 3 bits of B
16	PassA	Q = A	
17	SelectA	Q = A if flag is true else Q = B	
18	CarryExtend	Q = carry in all bit positions	
19	PassB	Q = B	
20	SelectB	Q = B if flag is true else Q = A	
21	SMultU	Q = (A * B) >> 8	upper byte, A and B are signed
22	SMultL	Q = (A * B)	lower byte, A and B are signed

23	LerpR	Q = B if I == 255 else
		Q = A + (B – A) * I + 128
			0.5 added to round result

24	MAddU	Q = (B * I + A) >> 8	upper byte
25	MAddL	Q = B * I + A	lower byte

1.2.5.4 Pixel Sequencer

Name	Description
Increment	This causes the next instruction address to be current instruction address + 1.
Jump	This causes the next instruction address to be taken from the constant field in the instruction. The most significant bit of the constant field determines if the address is an absolute address (0) or a relative address (1). If it is a relative address then the value in the constant field is added to the current address.
JumpTrue	This causes the next instruction address to be taken from the constant field in the instruction if the selected condition (masked by the tile mask) is true, otherwise the next instruction address is the current instruction address + 1. The true address can be absolute or relative.
JumpFalse	This causes the next instruction address to be taken from the constant field in the instruction if the selected condition (masked by the tile mask)is false, otherwise the next instruction address is the current instruction address + 1. The true address can be absolute or relative.
Call	This causes the next instruction address to be taken from the constant field in the instruction and the current instruction address + 1 written to the return address register. The true address can be absolute or relative.
Return	This causes the next instruction address to be taken from the return register. Calls and Returns do not need to be balanced as the stack is reset at the start of a program.
Done	This causes the sequencer to halt and any handshaking with the double buffered fragment and pixel registers to be done. The handshaking is only done in the case of a program initiated by the Tile message.
DoneAnd	This causes the sequencer to halt and any handshaking with the double buffered fragment and pixel registers to be done. The handshaking is only done in the case of a program initiated by the Tile message. The fragment flags are anded with the tile mask before the Tile message is passed on.

INDEX