Appendix A

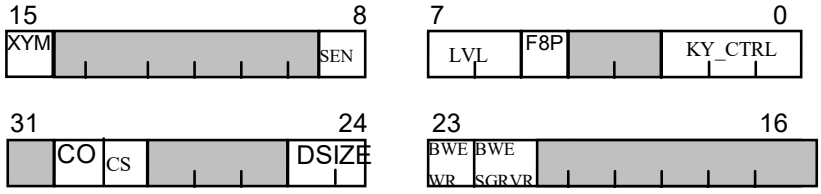
Imagine 128® Theory of Operation

Appendix A: IMAGINE 128™ Theory of Operation - Under Construction

A.1 Buffer Control (Needs to be revised)

Control of IMAGINE 128™'s three memory buffers is achieved through three buffer control registers. Each linear memory window has a buffer control register (MW0_CTRL, MW1_CTRL) and each drawing engine has a buffer control register (BUF_CTRL). The parameters in a specific buffer control register apply only to the resource associated with that register. For example, MW0_CTRL has no affect on how the drawing engine or the other memory window access the three buffers.

The buffer control registers for a linear window and a drawing engine are slightly different, but the drawing engine buffer control register shown below will illustrate all possible buffer control functions.



The **Source Enable** field (SEN) determines whether the internal cache or one of the three buffers is the source of data during a read cycle.

The **CO** bit is useful when consecutive commands within a group of commands are using the cache as the source of data. To use the CO bit, one programs the appropriate drawing engine registers, loads the cache with data, and then triggers the command by writing XY1. The drawing engine will execute the command without interruption. This technique is useful when one wants to partition the cache into halves. The drawing engine can be executing from one half of the cache while new data for the next command is written into the other half of the cache. The CO bit essential indicates to the drawing engine that the cache will always contain valid data. CO is not reset by the drawing engine.

The **CS** bit is used to select cache writing between XY windows and the texel cache/ palette.

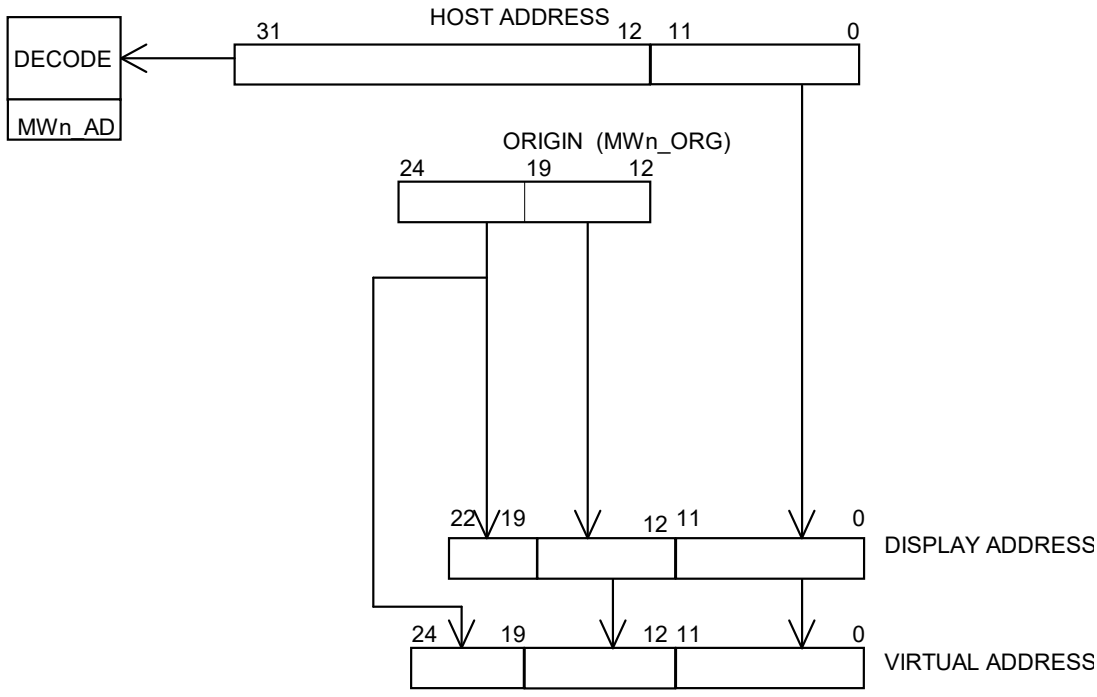
A.2 Linear Memory Windows Operation (Kevin, please review this section)

A Linear Memory Window is defined as a region of system memory that is mapped to IMAGINE 128 local memory space. IMAGINE 128 supports two memory windows. The address in system memory space to be mapped is programmed into MWn_AD, where n=0 or 1 for memory window 0 or memory window 1. The size of the memory window is programmed into MWn_SZ. Memory windows may be programmed from 4 Kbytes to 32 Mbytes. A memory window must begin on an address boundary equal to its size. For example, a 4 Kbyte window can be start at any 4 Kbyte address while a 1 Mbyte window must begin on a 1 megabyte boundary.

The memory window can be mapped to any of IMAGINE 128 's local buffers as described in the previous section. The address in the local buffer to which the memory window is mapped is determined by the MWn_ORG and MWn_PGE registers. The MWn_ORG register determines the starting address of the memory window in local memory space. The MWn_PGE register allows the value in MWn_ORG to be offset from 0 to 31 megabytes. A linear memory window is enabled by setting the appropriate window enable bit (EW0 or EW1) in the CONFIG1 register.

An example of how the local buffer address is calculated for a 4 Kbyte window is shown below. As can be seen from the diagram, host address bits [31:12] are compared with bits [31:12] of the MWn_AD register. If all bits are a match, this access is considered a window hit. The Display buffer address is then generated by adding the DVPGE register to MWn_ORG[24:19] and replacing the lower twelve bits of the result with the lower twelve bits of the host address.

ADDRESS FLOW FOR 4K BYTE MEMORY WINDOW



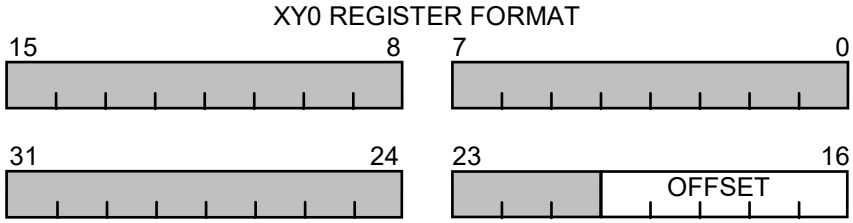
A.3 X-Y Windows Operation (needs to be reviewed)

An X-Y Memory Window maps a region of system memory into an X-Y oriented local memory space. The biggest architectural difference between Linear and X-Y memory windows is that a linear memory window completely bypasses the drawing pipeline while an X-Y window passes thorough the drawing pipeline. Data is transferred through an X-Y window by using the drawing engine's Read Transfer (RXFER) and Write Transfer (WXFER) command.

To set up an X-Y window, the X-Y window address (XYW_AD) must be initialized. XYW_AD defines the region of system memory space that will be decoded as an X-Y window. The SIZE field within XYW_AD determines how much system address space will be mapped to the X-Y window. As with a linear window, the size of the X-Y window determines on what address boundary the window may begin. The X-Y window decode is enabled by setting the X-Y window enable bit (EXA) in the CONFIG1 register.

A.3.1 XY Registers for X-Y Windows

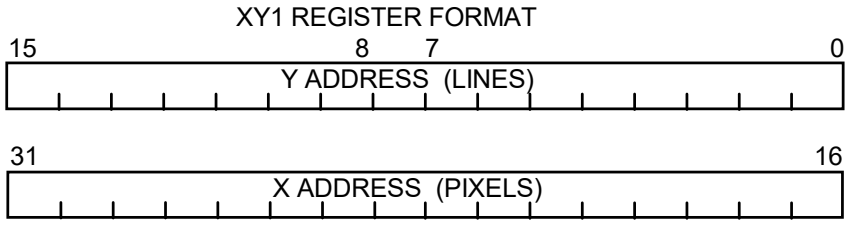
There are three XY registers that are utilized during read or write transfers: XY0, XY1, and XY2. XY0 defines the offset into the first host word of every line that is to be read or written. The XY0 format is shown below:



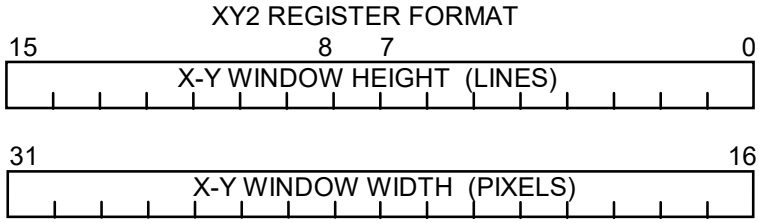
In 8, 16, or 32 bit/pixel mode, the offset specifies where within **the first data dword** of a line the first valid byte of data occurs. Allowable values for offset are 0 to 3., however, the offset should normally be set to the first pixel of data within a dword. This implies that in 32 bit/pixel mode that the offset be set to 0. In 16 bit/pixel mode, the offset should be either 0 or 2. In 8 bit/pixel mode, the offset can be 0 to 3.

If the offset is set to 0, the first data dword contains four bytes of valid data. If the offset is set to 1, the lowest byte of the first data dword is ignored. If the offset were set to 3, the lower three bytes of the first data dword would be ignored. The offset applies to only the first dword of data transfers. Subsequent transfers within a line assume that the entire 32 bit word is valid. In 1 bit/pixel mode (stipple mode), offset specifies where within the first data word the first valid bit of data occurs. Allowable values for offset are from 0 to 31.

The XY1 register specifies the starting address of the X-Y window in local memory space in terms of X-Y coordinates. The X-Y address is combined with the linear origin address in DE_ORG to compute the absolute linear address. XY1 should be programmed after all other XY and drawing parameter registers since writing XY1 will trigger the transfer command. The XY1 format is shown below:



The XY2 register specifies the width and height of the X-Y window in terms of pixels and lines respectively. The XY2 register format is shown below:



A.3.2 Drawing Parameter Registers for X-Y Windows

Since an XY window transfers data through the drawing pipeline, all applicable drawing parameter registers must be programmed appropriately. During a read transfer, the following drawing parameter registers must be programmed:

BUF_CTRL	Buffer Control Register
XYW_AD	XY Window Address
XYW_SZ	XY Window Size
DE_PGE	Page offset into local buffer(s)
DE_SORG	Origin of read data
DE_SPTCH	Source data pitch
CMD_OPC	Opcode for RXFER (0x6)

During a write transfer, the host data is treated as source data and the following drawing parameter registers must be programmed:

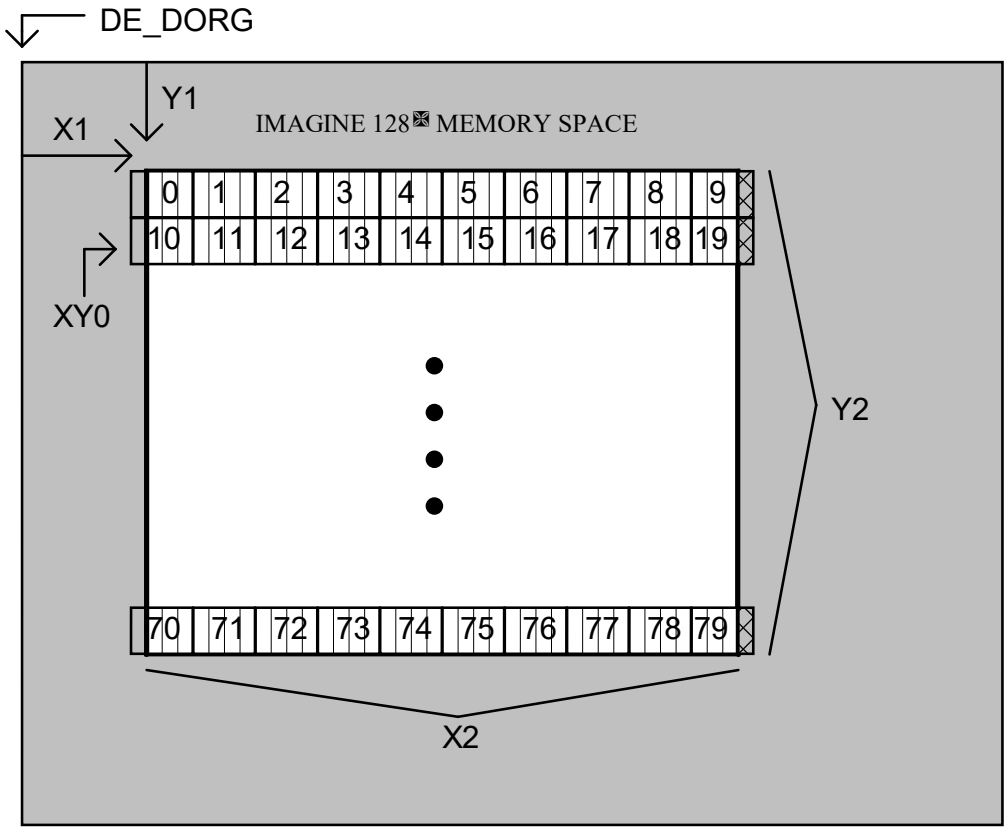
BUF_CTRL	Buffer Control Register
XYW_AD	XY Window Address
XYW_SZ	XY Window Size
DE_PGE	Page offset into local buffer(s)
DE_DORG	Origin of destination data
DE_DPTCH	Destination data pitch
CMD_OPC	Opcode for WXFER (0x7)
CMD_ROP	Raster operation
CMD_STYLE	Command Style
CMD_CLP	Clipping Control
FORE	Foreground Color Register
BACK	Background Color Register
MASK	Plane Mask
RMSK	Raster Mask
CLPTL	Top Left Clip
CLPBR	Bottom Right Clip

A.3.3 XY Window Example

The diagram below illustrates a XY window for a write transfer. The starting address of the window is specified by DE_DORG offset by X1 pixels and Y1 lines. The width of the window is X2 pixels and the height is Y2 lines. In this example, X2 is programmed for 38 pixels and Y2 is programmed for 8 lines. The first word offset, XY0, is programmed to 1 pixel. The pixel depth is assumed to be 8 bits/pixel.

The first 32 bit word of data that the host transfers through the XY window, word 0, will be written to the first location of the XY window. Since XY0 is set to 1, the first pixel of the first word is discarded. Therefore, only three pixels from the first host word are written. The next eight host data words are written in their entirety to consecutive locations. At this point, a total of 35 pixels have been written: 3 from the first word and 32 from the next eight words. Since the window was programmed to 38 pixels wide, only three pixels from word 9 will be written. It is very important to realize that the last pixel of word 9 will be discarded. It cannot contain the first pixel of data for the next line. It is software's responsibility to make sure that bitmaps that are transferred through an XY window are padded appropriately.

Word 10 of host data will be written to the first location of the second line with the first pixel discarded due to XY0 being set to 1. Host data will continue to be written in the manner described above until the last pixel of the last line is written. When the last pixel is written, the transfer is considered complete and any additional data written to the window will be ignored. Data from the host is always considered to be sequential. Any time data is not sequential, a new XY1 value must be loaded corresponding to the new address of the data.



A.4 Control of the Command Pipeline

Control of the command pipeline is a three step process.

Step 1. Ensure that IMAGINE 128™ is ready to accept new data.

Step 2. Update registers.

Step 3. Trigger New command.

IMAGINE 128™ employs extensive pipelining to smooth the process of loading new commands and parameters. When a command begins execution, all of the required parameters are transferred from the host accessible registers to internal working registers. When the parameters have been transferred to internal registers, IMAGINE 128™ is ready to accept a new command and parameters into its host accessible registers.

The command is queued for execution once the command trigger register (XY1) or the 3D_TRIG register (for 3D lines and triangles) has been written. This register must be the last one written in any command sequence. It is important to note that a command will not be triggered unless the most significant byte of XY1 or 3D_TRIG has been written. As soon as the trigger is written, IMAGINE 128™ will acknowledge by asserting the BUSY signal. A command that is ready for execution must still wait for a previous command to complete execution before it will actually start. During this waiting period, BUSY will remain set.

There are four additional bits that may be monitored to control the command pipeline. PRV (FLOW[3]) has two separate interpretations. During a write transfer, PRV=1 implies that at least half of the data cache is ready to receive data. During all other operations, PRV=1 indicates that the previously triggered command is still executing. It is similar to BUSY, except that PRV will not be de-asserted until both the drawing engine and memory controller have completed all operations associated with the previously triggered command. PRV is useful when operating the drawing engine in "dual cache mode".

CLP (FLOW[2]) will indicate that the command that just completed did so as the result of a clipping condition (i.e. stop on clip boundary). The CLP bit is cleared every time a new command begins execution, so it is important not to pipeline commands if the status of CLP is to be relied on.

Drawing Engine Busy (DEB- FLOW[0]) and Memory Controller Busy (MCB - FLOW[1]) provide additional visibility into the internal state of IMAGINE 128™. The drawing engine is idle when no commands are currently in execution. The memory controller will assert MCB when it is currently running a memory cycle or has a request for a memory cycle in its queue. Note that refresh or display transfer cycles alone will not cause MCB to be asserted, however they will cause MCB to be held if there are normal memory requests in the memory controller queue. The MCB bit is very useful in determining the integrity of data to be read back from the local buffers.

Control of the Command Pipeline (Continued)

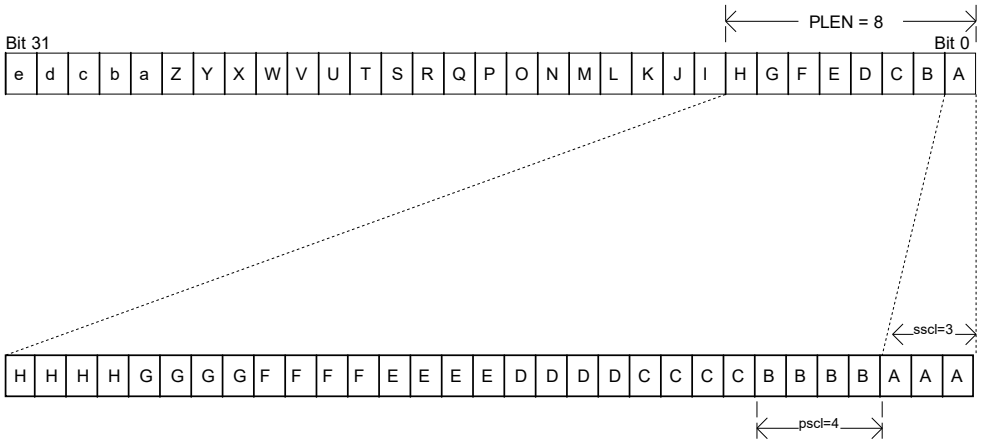
When utilizing X-Y windows, the pipeline must be controlled in a slightly different way. As with any command, the BUSY signal must be monitored before any new parameters are loaded into the host accessible registers. After setting up the X-Y transfer, the DEB bit must be checked to insure that the drawing engine has completed its current command. When the drawing engine is idle, the X-Y transfer may be triggered. When all X-Y data has been transferred from the host, the DEB should again be checked to ensure that the drawing engine is idle before the next command can be triggered. If this is not done, it is possible that there still could be valid data in the cache that could be corrupted if another command is triggered. It is important to note that DEB will remain active until all the data specified by the X-Y parameters has been transferred.

A.5 Draw Style and Patterning (Kinjeo, is this clear?)

Line Style	SOLID	TRNSP	LPAT bit	Result
Line solid	1	X	X	Foreground
Line on off dash	0	1	0	Destination
Line on off dash	0	1	1	Foreground
Line double dash	0	0	0	Background
Line double dash	0	0	1	Foreground

The LPAT bit is the corresponding bit in the pattern register. In addition, the dash members may be controlled by the settings in the pattern register (LPAT) and the pattern control register (PCTRL). The LPAT register is a simple 32 bit stipple pattern that selects foreground, background or transparent depending whether each bit is set or cleared and the state of the TRNP bit. PCTRL allows for scaling and cropping of the LPAT bit pattern.

The scaling factor is specified in the PSCL register and will cause each bit in the pattern register to be scaled from one to eight times. The SSCL register specifies the scale to be applied to the first bit in the pattern this value must be less than PSCL. The PLEN register specifies the pattern length, starting with LPAT[0]. Note that PLEN specifies pattern bits and not pixels drawn. For example, if PLEN is 10 (decimal) and PSCL is 5 (decimal) the total number of pixels drawn would be 50 (decimal) with each of the 10 pattern bits replicated 5 times.



The PATRN register provides additional information as to how to start and end. The NLST (no last) bit will cause the line algorithm not to draw the last pixel in the line, and not increment the line pattern for that pixel. This should be used in the drawing of poly lines where the end point of one line segment is common with the start point of the next line segment.

The PRST (pattern reset) bit will cause IMAGINE 128 to initialize the pattern at the start of each new LINE command. If PRST is not set, the pattern will be continuous from line to line.

For BITBLT and TRIAN, the APAT bit in the PATRN register will cause the source to be a 32x32 tile of pixels or stipple pattern if stipple is set. This pattern is locked to the screen, that is to say two abutting triangles or rectangles will have matching patterns at their edges.

A.6 Fill Style and Patterning

Fill Style	SOLID	TRNSP	STPL	stipple bit	Result
Fill Solid	1	X	X	n/a	fore op dst
Fill Tiled	0	0	0	n/a	src op dst
Fill Stippled	0	1	1	0	destination
Fill Stippled	0	1	1	1	fore op dst
Fill Opaque Stippled	0	0	1	0	back op dst
Fill Opaque Stippled	0	0	1	1	fore op dst

The transparency function is only meaningful when a stipple operation is being executed. Transparency should be set to zero at all other times.

A.7 Display List Processor (Kinjeo, is this clear?)

The Imagine 128 contains a Display List Processor (DLP) which can read drawing engine commands from memory and execute them. The DLP can accept four distinct formats, XY format (format 1), REG3 format (format 0), DMA format, or text mode. The format is selected when writing the end address, however a given format must be held constant though an entire list. Therefore list formats cannot change until after writing a new start address, or the list reaches an explicit stop point, and a new end address is written with the new format. It is very important that software maintain coherency by abiding by the above rules or the display list can have unpredictable results.

The XY Format is suitable for applications where only XY0 through 3 need to be accessed. The REG3 format provides the most flexibility where individual registers need to be accessed. The DMA format can start multiple DMA requests over AGP. Finally, the text mode provides a very efficient means of cacheing glyphs and drawing them rapidly to the screen.

Selecting format 0 specifies the Register/DMA mode. Setting bits 25:24 to 0 specifies the Non-DMA register format. Three registers can be specified by writing the lower 8 bits of the drawing engine registers into the first 3 bytes of the command. The upper 8th bit (or bank select) for each of the registers is located in bits 28-30 respectively. Bit 31 specifies waiting for vertical blank before executing the list. Bits 32 through 27 specify the register values to be written in the three registers, and bits 26-27 select the number of registers to be written (1-3). Registers are always written from A-C, no gaps allowed.

Setting bits 25:24 to a 01 selects the DMA mode. This mode may be used to request Multiple AGP DMA transfers. If bit 31 is set, then the DLP will wait after the current AGP request until it is complete.

Setting bit 25 to a 1 selects Text mode. In text mode, up to two glyphs can be specified, plus their destination addresses. The glyphs each have their own text table entry which provides information about the glyph to enahance rapid accessing. Software must set up all registers not accessed by the DMA, but needed for writing text, prior to executing this command. REG3 mode can be used to do this.

Any version of format 0 are allowed to be mixed within a single list.

After the data is ready for display list execution, the application must set the start and end addresses. The display list start address is loaded into DL_ADR. The display list end address is loaded into DL_CNTRL and the stop bit is set to 0. This triggers execution of the list. Three other bits are used for display list control. DL_SEN selects the primary or virtual buffer, DL_FMT selects the format, and DL_SVD selects DMA mode. These three buffers must follow the following rules to maintain coherency:

1) They can never be changed while a list is executing.
2) They can only be changed after a start address is written (for the new list), or only after a list has become idle and a new end address is written.

The DLP maintains two independent lists, an active list and pending list. The active list is the currently running list which is triggered by a start address and end address pair. It is possible to append to the list by writing to just the end address. This allows dynamic list building.

Whenever a list is running, a new start address can be written which will change the internal pointer to the pending list. The old list will continue executing, but all subsequent end addresses will now be associated with the pending lists start address. The pending list can be treated just like the primary list.

If a third start address is written, retries will be issued until the pending list becomes the active list and there is room for a new pending list.

** NOTE: The stop bit will stop all lists, including any pending lists and cause the lists to be inaccessible.

A.8 Texel Cache

The Imagine 128 is equipped with an advanced 8KB texel cache (TC) which greatly improves texture mapping and video applications. The cache can handle several palette and non-palette texture formats with sizes up to 512x512. Mipmapping is supported up to 10 levels of detail. The TC is capable of generating 4 texels/clock for 1 clock cycle bi-linear interpolation. It also employs an advanced look ahead algorithm for minimizing memory reads.

A.8.1 TC Sizes

The texture cache is capable of handling textures (or video frames) of up to 512 x 512 with the following restrictions. Textures must be a power of 2 in both X and Y directions. X and Y sizes are independent, i.e. 2 x 64, 128 x 32, etc.. are allowed.

The cache is optimized for an efficient use of space, however the following formats (or smaller) work best as they will fit directly into the cache:

Texture Format (bits per textured pixel (bpt))	Texture Size (XxY)
32 bpt	32x64
16 bpt	64x64
8 bpt (palettized or direct)	128x64
4 bpt (palette)	256x64
2 bpt (palette)	512x64
1 bpt (palette)	512x64

Using textures of the above size or less will result in the maximum number of hits, minimizing misses and greatly improving cache performance.

A.8.2 Texture Formats

Please see the register definition section for TEX_CTRL

A.8.3 Special Commands

There are two commands associated with the texture cache, INV_TEX and LD_PAL. The INV_TEX command instructs the texel cache that the next triangle coming through is using a different texture than the previous ones.

The LD_PAL command must be used prior to a palettized texture map if the palette hasn't been loaded. This command will load a palette for use.

A.9 Alpha Blending

The Imagine 128 implements full OpenGL compatible alpha blending. The blending function can be applied to any Drawing Engine command. The control of blending is performed in the ACNTRL register.

To properly set up a blending command, the following must be done. First, select a source and destination blending function. These are defined in the lower byte of the ACNTRL register. Second, if you are in a mode which doesn't support alpha or are going to use the source or destination alpha registers, you must load them into the lower two bytes of the ALPHA register. If you are in a mode which supports alpha and you want to use the constant registers, you must set the SRE and DRE bits to override the default settings. Finally, the blending enable bit must be set to activate blending.

A.10 Programming Imagine 128[4], 3D Operations

The Imagine 128™ is a vertex-based 3D polygon accelerator. It performs the setup for 3D triangles, lines and points in hardware.

There are 2 3D Rendering Commands:

1) TRIAN_3D
2) LINE_3D

NOTE: LINE_3D can be used to render 3D points. Texture Mapped 3D Lines are not supported.

There are up to three steps to take to program a 3D operation:

Step 1: Control and Mode Setup
Step 2: Parameter Load
Step 3: Trigger

A.10.1 Control and Mode Setup

a) BUF_CTRL <25:24> Destination Pixel Size and Format
b) DE_SORG: Source Origin
c) DE_DORG: Destination Origin
d) DE_TPTCH: Texture Pitch of LOD0
e) DE_ZPTCH: Z Pitch
f) DE_SPTCH: Source Pitch
g) DE_DPTCH: Destination Pitch
h) CMD: Command
i) FORE: Foreground Color
j) BACK: Background Color
k) MASK: 0xFFFFFFFF Pixel Plane Mask
l) DE_KEY: Color Key
m) LPAT: Line Pattern (For 3D Lines)
n) PCTRL: Line Pattern Control
o) DE_ZORG: Z-Buffer Origin
p) LOD0_ORG: Texture Mipmap Level 0 Origin
q) LOD1_ORG: Texture Mipmap Level 1 Origin
r) LOD2_ORG: Texture Mipmap Level 2 Origin
s) LOD3_ORG: Texture Mipmap Level 3 Origin
t) LOD4_ORG: Texture Mipmap Level 4 Origin
u) LOD5_ORG: Texture Mipmap Level 5 Origin
v) LOD6_ORG: Texture Mipmap Level 6 Origin
w) LOD7_ORG: Texture Mipmap Level 7 Origin
x) LOD8_ORG: Texture Mipmap Level 8 Origin
y) LOD9_ORG: Texture Mipmap Level 9 Origin
z) HITH: Hither Value (For Z-Buffer Enabled)
aa) YON: YON Value (For Z-Buffer Enabled)
bb) FOG_COL: Fog Color (For Fog Mode)
cc) ALPHA: <23:16> Alpha Test Value (For Alpha Compare)
dd) TEX_BORDER: Texture Border Color
ee) KEY_3D_LOW: 3D Key Color Low Range
ff) KEY_3D_HI: 3D Key Color High Range
gg) A_CNTRL: Alpha Control Register
hh) 3D_CNTRL: 3D Control Register
ii) TEX_CNTRL: Texture Map Control Register

A.10.2 Parameter Load

a) CP0: Pattern Pointer
b) CP1: Vertex 0 X
c) CP2: Vertex 0 Y
d) CP3: Vertex 0 Z
e) CP4: Vertex 0 W
f) CP5: Vertex 0 Color {A,R,G,B}
g) CP6: Vertex 0 Specular {F,Rs,Gs,Bs}
h) CP7: Vertex 0 U
i) CP8: Vertex 0 V
j) CP9: Vertex 1 X
k) CP10: Vertex 1 Y
l) CP11: Vertex 1 Z
m) CP12: Vertex 1 W
n) CP13: Vertex 1 Color {A,R,G,B}
o) CP14: Vertex 1 Specular {F,Rs,Gs,Bs}
p) CP15: Vertex 1 U
q) CP16: Vertex 1 V
r) CP17: Vertex 2 X
s) CP18: Vertex 2 Y
t) CP19: Vertex 2 Z
u) CP20: Vertex 2 W
v) CP21: Vertex 2 Color {A,R,G,B}
w) CP22: Vertex 2 Specular {F,Rs,Gs,Bs}
x) CP23: Vertex 2 U
y) CP24: Vertex 2 V

Note: **The three sets of parameters (CP1-CP8), (CP9-CP16) and (CP17-CP24) represent the parameters for the three vertices of a 3D triangle. For the 3D Line or Point, (CP9-CP16) and (CP17-CP24) represent the parameters for the two endpoints. Since texture mapped lines are not supported, the (w,u,v) parameters for 3D Line or Point commands are meaningless.**

A.10.3: Trigger:

To trigger the 3D rendering command, write into the 3D_TRIGGER register. This register needs only to be written. No data needs to be supplied.

A.11 3D Operational Modes and Control

For 3D triangle commands, the minimal set of CP parameters you need to set are the x,y coordinate pairs for the three vertices (CP1, CP2), (CP9, CP10), and (CP17, CP18). For 3D line commands, the minimal set of CP parameters you need to set are the x,y coordinate pairs for the two endpoints (CP9, CP10), and (CP17, CP18).

A.11.1 Gouraud Shading Mode

- To enable Gouraud Shading, set ***3D_CNTRL<24>*** = 1.
- Must disable SOLID mode by setting ***CMD<16>*** = 0.
- Set the color values for the three vertices of the triangle (CP5, CP13, CP21).

-OR- set the color values for the two endpoints of the line (CP13, CP21).

A.11.2 Specular Highlighting Mode

- To enable Specular Highlighting, set ***3D_CNTRL<25>*** = 1.
- Set the Specular color values for the three vertices of the triangle (CP6, CP14, CP22).
  -OR- set the Specular color values for the two endpoints of the line (CP14, CP22).

A.11.3 Z Buffer Mode

- To enable Z-Buffering, set ***3D_CNTRL<0>*** = 1. If disabled, no z-buffer hidden surface removal or z-buffer updates will occur.
- Z Scale Mode: Silver Hammer supports 16bpp and 24 bpp Z. It accepts either unscaled or scaled Z (normalized to the range of 0 to 1). If the Z Scale Mode bit (***3D_CNTRL<30> = 1***) is set, then the normalized Z value will be scaled to either 2^16 in 16bpp destination pixel mode or 2^24 in 32bpp destination pixel mode.
- If Z compares and Z-Buffer updating are desired, disable Read_Only_Z (***3D_CNTRL<1>*** = 0). If Read_Only_Z is enabled, then Z compares would occur, but the Z-Buffer would not be updated.

Dest Pixel Size	Clamping	Setup	Z Size	Dest Zsize
16bpp	*16bpp*	Yes	16bpp	16bpp
32bpp	*24bpp*	Yes	24bpp	24bpp

- Z Compare Operators: (***3D_CNTRL<7:5>***) See 5.8.35 for the table of Z compare operators used in Z-Buffer hidden surface removal operation.
- **Note:** If Z Compare Operator is set to ALWAYS (***3D_CNTRL<7:5>*** = 0x1), performance will be enhanced by 40%. An example of an application that would utilize this performance enhancing feature would be Z Fills.
- Yon Compare Operators: (***3D_CNTRL<10:8>***) See 5.8.35 for the table of Yon compare operators used in Yon Z Clipping surface removal operation.
- Hither Compare Operators: (***3D_CNTRL<13:11>***) See 5.8.35 for the table of Hither compare operators used in Hither Z Clipping surface removal operation.
- Set the z values for the three vertices of the triangle (CP3, CP11, CP19)

-OR- set the z values for the two endpoints of the line (CP11, CP19).

A.11.4 Fog Mode

- To enable Fog, set ***3D_CNTRL<27>*** = 1.

Result_Color = Color * Fog_factor_percentage + Fog_Color * (1 - Fog_factor_percentage).

The Fog_factor_percentage (or just fog_factor) can either come from the vertices on the polygon or from a user-defined fog table. The user-defined table would be indexed by either the Z-value or the 1/w value of the drawn pixel.

A.11.4.1 Vertex Fog Mode (3D_CNTRL[4] = 0)

- Set the vertex-based fog_factor value for the three vertices of the triangle (CP6, CP14, CP22) -OR- set the fog_factor value for the two endpoints of the line (CP14, CP22).

A.11.4.2 Table Fog Mode(3D_CNTRL[4] = 1)

- The fog_factor value for the Fog tables are loaded via the host bus by setting the CS bit in BUF_CNTRL before writing to the Silverhammer cache. There are 65 entries in the table. Fog Tables can be indexed by the per-pixel 1/w or z value. The index will range from 0 to 2. Index of 0 will index the first entry and index of 2.0 will index the 65th entry. Fog_Index_Select (3D_CNTRL[3]) selects between using 1/w (FIS=0) or using z (FIS=1).

Fog Enable	Fog Selector	Fog Index Select	Fog
0	*x*	x	No Fog
1	0	x	Vertex Fog
1	1	0	1/w Table Fog
1	1	1	z Table Fog

A.11.5 Alpha Modes and Control

A.11.5.1 Alpha Comparison

- To enable Pixel Alpha Compare, set ***A_CNTRL<19>*** = 1.
- Set the Alpha Test Value in ***ALPHA_REG<23:16>***.
- The Pixel Alpha Compare Operators are set in ***A_CNTRL<18:16>***. See 5.8.35 for the table of alpha compare operators used in Alpha Clipping.
- If the alpha value to be compared is originating from the vertices of the triangle or line, set the alpha values for the three vertices of the triangle (CP5, CP13, CP21)

-OR- set the alpha values for the two endpoints of the line (CP13, CP21).

A.11.5.2 Alpha Select, Alpha Modulation and Alpha Blend Select

- Alpha Select (***A_CNTRL<24>***) will select alpha value from current texel if A_CNTRL<24> is set to 0 and texture mode (***TEX_CNTRL<0>*** = 1) is enabled. Otherwise, the alpha value will be selected from either the interpolated pixel value, foreground or background value, i.e. non_texel_alpha depending upon which pixel color mode is set.
- Enable Alpha Modulation (***A_CNTRL<25>*** = 1) will, with texture mode enabled, modulate the texel alpha value with non_texel_alpha.

  Resultant_Texel_Alpha = Pixel_Alpha * Texel_Alpha

- Enable Alpha Blend Select (OpenGL mode) (***3D_CNTRL<17>*** = 1) will, with texture mode enabled, alpha blend the alpha value:

  Resultant_Alpha = Pixel_Alpha * (1 - Texel_Alpha) + OpenGL_Blend_Color_Alpha * Texel_Alpha

Alpha Select	Texture Mode	Alpha Modulation	Alpha_Blend_Sel	Alpha Value
1	x	x	x	*Gourard Shaded, Flat Shaded, or Bg/Fg (Line) Alpha*
0	1	0	0	*Texel Alpha*
0	1	0	1	*Alpha Blended Alpha*
0	1	1	0	*Alpha Modulated Alpha*
0	1	1	1	*Texel Alpha*

A.11.5.3 Decal Alpha Mode

- Enabling Decal Alpha Mode (***A_CNTRL<26>*** = 1) would perform the following alpha blending function:

RGB = (Texel_RGB)*(Texel_Alpha) + (Non_Texel_RGB)*(1-Texel_Alpha).

A.11.6 Rectangle Mode

- Enabling Rectangle Mode (***3D_CNTRL<28>***) will render a rectangle with the triangle command, but the triangle must be specified as follows:
  1) It is a flat top triangle.
  2) Its spatial (x,y) vertex coordinates must be specified as follows:

  V0 V1 OR V1 V0

  V2 (V3) (V3) V2

  where VO(x,y) corresponds to (CP1,CP2), V1(x,y) corresponds to (CP9,CP10) and V2(x,y) corresponds to (CP17,CP18). The command will drop a vertical edge from the specified V1 point to (V3) to complete the rectangle, thus achieving a rendered rectangle on a triangle command.

A.11.7 Dither Mode

- Dithering is used, among other things, to remove banding artifacts when displaying an image at a lower pixel depth, i.e. from 24bpp to 16bpp. To enable dithering, set (***3D_CNTRL<16> = 1***).

A.11.8 Backface Cull Mode

- To enable backface cull mode, set (***3D_CNTRL<23> = 1***). With this mode turned on, SilverHammer will not render triangles that are facing the back. This is accomplished by performing a calculation on an ordered set of vertex spatial (x,y) coordinates. The order of the coordinates may be either clockwise or counter-clockwise and must to set by setting the CW/CCW Selector Bit for Culling (***3D_CNTRL<22>*****).** Setting it to zero would denote clockwise direction and a setting of one would denote counter-clockwise.

A.11.9 Texture Map Modes and Control

Note: Before using a new texture, the texel cache must be invalidated. After the TEX_CNTRL register is set up, and before the 3D Texture command is triggered, you must invalidate the texel cache as follows:

1) Set command opcode to 0xA.
2) Trigger this command by writing into XY1 register. Any value is okay because it is only used to trigger this command.

A.11.9.1 Non-Mipmap Texture Map Mode Set Up

- Enable Texture Map Mode by setting ***TEX_CNTRL<0>*** = 1.
- To enable Perspective Correction, set ***TEX_CNTRL<6>*** = 1. Otherwise, it would be orthogonal mode and the w values at the vertices are automatically set to 1.
- Set the perspective w and u,v texel coordinate values for the three vertices of the triangle V0(CP4, CP7, CP8), V1(CP12, CP15, CP16), and V2(CP20, CP23, CP24).
- The texture image size is specified by the MMSIZEX (***TEX_CNTRL<19:16>*****)** and MMSIZEY (***TEX_CNTRL<23:20>*****)** fields of the TEX_CNTRL register. The texture size must be a power of 2 and not necessarily the same power of 2, i.e. SilverHammer support power of 2 rectangular texture images.
- The origin of a non-mipmapped (see Mipmap Mode) texture is to be specified in the ***LOD0_ORG*** register. The LOD origin must be 128 bits or 16-byte aligned. The ***TPTCH*** register contains the pitch of LOD0_ORG.
- ***Important: Each vertex w must be written before its respective vertex u,v coordinate pairs.***
- **Disabling texture map mode disables all texture map modes.**

A.11.9.2 MipMap Texture Map Mode Set Up

- Enable Texture Map Mode by setting ***TEX_CNTRL<0>*** = 1.
- To enable Per-Pixel MipMap Mode, set ***TEX_CNTRL<1>*** = 1.
- To enable Perspective Correction, set ***TEX_CNTRL<6>*** = 1. Otherwise, it would be orthogonal mode and the w values at the vertices are automatically set to 1.
- Set the perspective w and u,v texel coordinate values for the three vertices of the triangle V0(CP4, CP7, CP8), V1(CP12, CP15, CP16), and V2(CP20, CP23, CP24).
- Ten levels of detail (LOD) Mipmapping is supported in SilverHammer and is on a per pixel basis. The origin, pitch and Mipmap XSIZE and YSIZE of the largest LOD is to be specified in LOD0_ORG, TPTCH, ***TEX_CNTRL<19:16>*** and ***TEX_CNTRL<23:20>***, respectively. Subsequent LOD origins are to be specified in LOD1_ORG through LOD9_ORG and the number of mipmap levels are to be specified in ***TEX_CNTRL<15:12>***. Only the origins of the specified number of mipmap levels are required to be specified in LOD0_ORG through LOD9_ORG.

- SilverHammer will support texture images up to a maximum size of 512x512 to a minimum size of 1x1. The width and height must be a power of 2 and they need not have to be equal, i.e. SilverHammer supports square and rectangular texture images. Each subsequent LOD mipmap must be half the size in both x and y dimensions until it reaches the size of 1x1. In the case of rectangular texture images, each subsequent mipmap x and y size will be halved until one of the dimension reaches 1. At that level, then the other dimension of each subsequent LOD mipmap gets halved until a 1x1 size is achieved. E.g.

LOD0	64 x 16
LOD1	32 x 8
LOD2	16 x 4
LOD3	8 x 2
LOD4	4 x 1
LOD5	2 x 1
LOD6	1 x 1

A.11.9.3 Texture Filtering, Lighting and Coordinate Set Up

- SilverHammer supports magnification and minification filtering, (***TEX_CNTRL<2> and TEX_CNTRL<4>)***, respectively. For Magnification Filtering, setting ***TEX_CNTRL<2> = 0,*** will enable bilinear filtering. Otherwise, it will perform nearest texel sampling. Similarly, setting ***TEX_CNTRL<4> = 0,*** will enable bilinear filtering and disabling this bit will give you nearest texel sampling for minification.
- Texture Format is specified in ***TEX_CNTRL<29:24>***. Please refer to Chapter 5 for formats supported. Note: Texel formats other than 8888 will be expanded to 8888 before texture processing is performed. For example, 1555 texel format will be expanded as follows:

alpha[0]	->	8{alpha[0]}
red[4:0]	->	{red[4:0], red[4:2]}
green[4:0]	->	{green[4:0], green[4:2]}
blue[4:0]	->	{blue[4:0], blue[4:2]}

- The vertex u,v coordinates may either be specified as a scaled number, normalized to be between 0 to 1 inclusive, or unscaled. To be in UV scale mode, set ***TEX_CNTRL<31> = 1.*** The uv values will be scaled to the Mipmap XSIZE and YSIZE, respectively.
- To enable RGB Modulation Mode, set ***TEX_CNTRL<5>*** = 1. This mode modulates the texture image with an interpolated rgb surface generated by the Gouraud shader. So, Gouraud Shading needs to be enabled by setting ***3D_CNTRL<24>*** = 1 and the three color parameters (or two, depending upon the rendering command) need to be set (see Gouraud shading)**.**

  For Example:

  Modulated_Texel_Red = Pixel_Red * Texel_Red
  Modulated_Texel_Green = Pixel_Green * Texel_Green
  Modulated_Texel_Blue = Pixel_Blue * Texel_Blue

- To enable Texture Wrap Mode, disable the U and V Texture Clamp Bits, ***TEX_CNTRL<8> = 0 and TEX_CNTRL<9> = 0***. To clamp in U or V coordinate space or both coordinate space, just enable each bit (or both) accordingly. In Texture Clamping, it could either be clamped to a border color specified in TBORD_COL register or to the color at the edge of the textured image. Setting the clamp selector to 1, ***TEX_CNTRL<7> = 1***, will select the Texture Border Color. Otherwise, it would clamp to the texel value at the edge of the texture image.

- SilverHammer supports Mipmap Linear Filtering. This mode will interpolate between the two nearest LOD mipmap for better texture image quality. It requires two passes to perform this feature.

  Setup: Refer to section A.11.9.2 MipMap Texture Map Mode Set Up to set up a mipmap texture triangle command.

  Pass 1:
  1) Enable Mipmap Linear Mode by setting ***TEX_CNTRL<3> = 1.***
  2) Set Mipmap Linear Pass 2 bit ***TEX_CNTRL<10> = 0.***
  3) Set the fog color register (**FOG_COL**) to 0.
  4) Enable Texture Composite Mode, ***3D_CNTRL<27> = 1***.
  5) If Z buffer is enabled (***3D_CNTRL<0> = 1)***,
     enable Z Read Only Bit ***3D_CNTRL<1> = 1.***
  6) Disable Alpha Blending Mode, ***A_CNTRL<10> = 0.***
  7) Trigger the 3D command, i.e. write into 3D_TRIG register.

  Pass 2:
  1) Enable Mipmap Linear Mode by setting ***TEX_CNTRL<3> = 1.***
  2) Set Mipmap Linear Pass 2 bit ***TEX_CNTRL<10> = 1.***
  3) Set the fog color register (**FOG_COL**) to 0.
  4) Enable Texture Composite Mode, ***3D_CNTRL<27> = 1***.
  5) If Z buffer is enabled (***3D_CNTRL<0> = 1)***,
     disable Z Read Only Bit ***3D_CNTRL<1> = 0*** unless Z Read Only Bit ***3D_CNTRL<1>*** was already set to 1 before pass 1.

     Note: Another way of interpreting how to set Z Read Only Bit is as follows:

     Z_Read_Only = Z_Read_Only_Original | NOT Mipmap_Linear_Pass_2;

  6) Enable Alpha Blending Mode, ***A_CNTRL<10> = 1.*** Set Blending Source and Destination Function to SRC_ONE and DST_ONE, respectively, i.e. set ***A_CNTRL<3:0> = 0x1 and A_CNTRL<7:4> = 0x1.***
  7) Trigger the 3D command, i.e. write into 3D_TRIG register.

  Note: *Linear* Mipmap Linear Mode, also known as Trilinear Mipmapping, is Mipmap Linear with the texture filtering set to bilinear interpolation. If texture filtering is set to Nearest Texel Sampling, then we would be in *Nearest* Mipmap Linear Mode.

- Texture Cache supports two cache configurations: Linear or Tile Mode. Setting the Texture Cache Tile Mode, ***TEX_CNTRL<30> = 1,*** will configure the cache lines to be a 1 page by 8 configuration. This mode would be better for non-horizontal texture mapping applications. In Texture Cache Linear Mode, ***TEX_CNTRL<30> = 0,*** cache will be configured as 8 pages by 1. This mode would be better for horizontal texture mapping applications, e.g. rectangular video clip acceleration.

- To enable 3D Texture Color Range Keying, set (***3D_CNTRL<15> = 1)***. To set the 3D High Color Key value, set KEY_3D_HI. Similarly, the low value would be set in KEY_3D_LOW. SilverHammer supports inclusive as well as exclusive color range keying. For inclusive color range keying, set the 3D Key Polarity bit (***3D_CNTRL<14>)*** to 0. Otherwise, setting it to one, would enable exclusive 3D color range keying. The Color keying is performed on all texels requested and before any filtering or processing has occurred. In other words, it is performed on the unmodified texels.

- SilverHammer supports integer- and 0.5-centered pixel and texel coordinates. This gives SilverHammer flexibility to support various different 3D API's specifications, i.e. Direct3D vs OpenGL. For integer-centered pixels, set (***3D_CNTRL<21>)*** to 0. For 0.5-centered pixels, set (***3D_CNTRL<21>)*** to 1. For integer-centered texels, set (***3D_CNTRL<26>)*** to 0. For 0.5-centered texels, set (***3D_CNTRL<26>)*** to 1.

- For backward compatibility to Imagine 128^4 (T2R IV), SilverHammer supports a special 8bpt index mode. By setting (***3D_CNTRL<29>)*** to 1, and setting ***(TEX_CNTRL<29:24> = 0x0D, 8-bit index mode)***, the triangle will be texture-mapped (Nearest mode must also be set) with the 8-bit index texture image. This mode will then use 8-bit value to index into the DAC palette to deliver the palettized result.

- To select which RGB value will be displayed, there is an RGB_SELECT bit (***3D_CNTRL<19>)*** which would select the RGB value either from the texture image or the vertices. Setting it to 0 would select the RGB from the texture image. Setting it to 1 will select RGB from the vertices.

A.11.10 Pixel Color Table

Non_Texel_Color Priority
1) Foreground Color if SOLID (***CMD<16>*** = 1) is enabled.
If RGB_Select is set to 1:
2) Background Color if internal pattern bit is set to zero.
3) Foreground Color if internal pattern bit is set to one and Gouraud Shading is not set.
4) Interpolated Gouraud Shaded Color if internal pattern bit is set to one and Gouraud Shading is set.
Else (RGB_Select is set to 0)
Texel_Color Priority (When texture mode (***TEX_CNTRL<0>*** = 1) is enabled.)
1) Modulated Texel Color corresponds to a modulated, blended or filtered Texel Color.
2) Texel Color.

A.11.11 OpenGL Texture Environment and Texture Functions

Please refer to the OpenGL Specification 1.1, Chapter 3.8.5 and 3.8.6, Table 3.10 and 3.11 for more details.

The following registers are used in the table to generate all of the OpenGL functions.

Control	Definition	Register Location
ABS	Alpha Blend Select	3D_CTRL[17]
TBS	Texture Blend Select	3D_CTRL[18]
RSEL	RGB Select	3D_CTRL[19]
RM	RGB Modulation	TEX_CNTRL[5]
ASL	Alpha Select	ACNTRL[24]
AMD	Alpha Modulation	ACNTRL[25]
DA	Decal Alpha Blend	ACNTRL[26]

There can only be at most one RGB texture mode enabled and at most one Alpha texture mode enabled. Otherwise, the results are unpredictable.

Note: Still Under Construction….

The following table shows the OpenGL blend functions and how they can be obtained using the SilverHammer graphics processor.

Texture FMT	Fragment	Replace	Modulate	Decal	Blend
Alpha	RGB	RSEL = 1 RM = x DA = x TBS = x	RSEL = 1 RM = x DA = x TBS = x	Undefined	RSEL = 1 RM = x DA = x TBS = x
	Alpha	ASL = 0 AMD = 0 ABS = 0	ASL = 0 AMD = 1 ABS = 0	Undefined	ASL = 0 AMD = 1 ABS = 0
:Luminance	RGB	RSEL = 0 RM = 0 DA = 0 TBS = 0	RSEL = 0 RM = 1 DA = 0 TBS = 0	Undefined	RSEL = 0 RM = 0 DA = 0 TBS = 1
	Alpha	ASL = 1 AMD = x ABS = x	ASL = 1 AMD = x ABS = x	Undefined	ASL = 1 AMD = x ABS = x
Luminance Alpha	RGB	RSEL = 0 RM = 0 DA = 0 TBS = 0	RSEL = 0 RM = 1 DA = 0 TBS = 0	Undefined	RSEL = 0 RM = 0 DA = 0 TBS = 1
	Alpha	ASL = 0 AMD = 0 ABS = 0	ASL = 0 AMD = 1 ABS = 0	Undefined	ASL = 0 AMD = 1 ABS = 0
Intensity	RGB	RSEL = 0 RM = 0 DA = 0 TBS = 0	RSEL = 0 RM = 1 DA = 0 TBS = 0	Undefined	RSEL = 0 RM = 0 DA = 0 TBS = 1
	Alpha	ASL = 0 AMD = 0 ABS = 0	ASL = 0 AMD = 1 ABS = 0	Undefined	ASL = 0 AMD = 0 ABS = 1
RGB	RGB	RSEL = 0 RM = 0 DA = 0 TBS = 0	RSEL = 0 RM = 1 DA = 0 TBS = 0	RSEL = 0 RM = 0 DA = 0 TBS = 0	RSEL = 0 RM = 0 DA = 0 TBS = 1
	Alpha	ASL = 1 AMD = x ABS = x	ASL = 1 AMD = x ABS = x	ASL = 1 AMD = x ABS = x	ASL = 1 AMD = x ABS = x
RGBA	RGB	RSEL = 0 RM = 0 DA = 0 TBS = 0	RSEL = 0 RM = 1 DA = 0 TBS = 0	RSEL = 0 RM = 0 DA = 1 TBS = 0	RSEL = 0 RM = 0 DA = 0 TBS = 1
	Alpha	ASL = 0 AMD = 0 ABS = 0	ASL = 0 AMD = 1 ABS = 0	ASL = 1 AMD = x ABS = x	ASL = 0 AMD = 1 ABS = 0