SST-1(a.k.a. Voodoo Graphics™)

HIGH PERFORMANCE GRAPHICS ENGINE FOR 3D GAME ACCELERATION

Revision 1.61
December 1, 1999

3dfx Interactive, Inc.
4435 Fortran Drive
San Jose, CA 95134
Phone: (408) 935-4400
Fax: (408) 262-8602
www.3dfx.com

Proprietary Information

Copyright Notice:

[English translations from legalese in brackets]

No Warranty

THESE SPECIFICATIONS ARE PROVIDED BY 3DFX "AS IS" WITHOUT ANY REPRESENTATION OR WARRANTY, EXPRESS OR IMPLIED, INCLUDING ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS, OR ARISING FROM THE COURSE OF DEALING BETWEEN THE PARTIES OR USAGE OF TRADE. IN NO EVENT SHALL 3DFX BE LIABLE FOR ANY DAMAGES WHATSOEVER INCLUDING, WITHOUT LIMITATION, DIRECT OR INDIRECT DAMAGES, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SPECIFICATIONS, EVEN IF 3DFX HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

[You're getting it for free. We believe the information provided to be accurate. Beyond that, you're on your own.]

1. General Description

The SST-1 Graphics Engine from 3Dfx Interactive is the first video subsystem that enables personal computers and low cost video game platforms to host true 3D entertainment applications. Optimized for real-time texture-mapped 3D games and educational titles, SST-1 provides acceleration for advanced 3D features including true-perspective texture mapping with trilinear mipmapping and lighting, detail and projected texture mapping, texture anti-aliasing, and high precision sub-pixel correction. In addition, SST-1 supports general purpose 3D pixel processing functions including polygonal-based Gouraud shading, depth-buffering, alpha blending, and dithering.

Features

- Triangle raster engine
- Linearly interpolated Gouraud-shaded rendering
- Perspective-corrected (divide-per-pixel) texture-mapped rendering with iterated RGB modulation/addition
- Detail and Projected Texture mapping
- Linearly interpolated 16-bit Z-buffer rendering
- Perspective-corrected 16-bit floating point W-buffer rendering (patent pending)
- Texture filtering: point-sampling, bilinear, and trilinear filtering with mipmapping
- Texture formats: 8-bit RGB(3-3-2), 8-bit intensity, 8-bit alpha, 8-bit narrow channel YIQ(4-2-2), 8-bit alpha-intensity(4-4), 16-bit RGB(5-6-5), 16-bit ARGB (1-5-5-5), 16-bit ARGB (4-4-4-4), 16-bit ARGB (8-3-3-2), 16-bit narrow channel AYIQ (8-4-2-2), 16-bit alpha-intensity (8-8)
- Texture decompression: 8-bit "narrow channel" YIQ (patent pending)
- Transparency with dedicated color mask
- Source/Destination pixel alpha blending
- Sub-pixel correction to .4 x .4 resolution
- 24-bit color dithering to native 16-bit RGB buffer using 4x4 or 2x2 ordered dither matrix
- Gamma-correction color lookup table
- Advanced memory architecture utilizing 1+ GByte/sec bandwidth
- 2-4 MBytes of EDO DRAM frame buffer memory
- 1-4 MBytes of EDO DRAM texture memory
- Resolution Support:

Frame Buffer Memory	Double Buffered, no Depth-Buffering	Triple Buffered, no Depth-Buffering	Double Buffered, 16-bit Depth-Buffering
2 MBytes	800x600x16	640x480x16	640x480x16
4 MBytes	800x600x16	800x600x16	800x600x16

- Direct memory-mapped access to frame buffer and texture memories via linear address mapping
- PCI Bus 2.1 compliant

Benefits

- Real-time, true-perspective, texture-mapped rendering with lighting support at low cost
- Eases software porting, as SST-1 only accelerates the inner rasterization loop
- Lowest cost solution designed expressly for use in the entertainment markets
- Patent pending techniques result in reduced texture memory requirements

2. Performance

The following tables show the estimated performance of SST-1. Performance is calculated assuming that the PCI Bus master is supplying data at its peak bandwidth. Thus, the performance levels are the maximum sustainable rates of SST-1, not necessarily the system performance. If a particular operation is CPU limited or a particular PCI bus master is not supplying data at its peak rate, then the effective system performance level will decrease. All numbers are represented in 16-bit MPixels/sec unless otherwise specified with the optional memory FIFO is disabled, and are measured assuming 640x480 resolution @ 60 Hz with a 50 MHz graphics clock frequency driving 50-ns Extended-Data-Out (EDO) DRAMs.

Flat shaded triangles (no fogging, alpha-blending, Z-buffering, or sub-pixel correction)	**Ktriangles/sec**
10-pixel, right-angled, horizontally oriented	1911
25-pixel, right-angled, horizontally oriented	1096
50-pixel, right-angled, horizontally oriented	644
1000-pixel, right-angled, horizontally oriented	42

Gouraud shaded, sub-pixel corrected triangles with fogging, alpha-blending and Z-buffering	**Ktriangles/sec**
10-pixel, right-angled, randomly oriented	1231
25-pixel, right-angled, randomly oriented	968
50-pixel, right-angled, randomly oriented	550
1000-pixel, right-angled, randomly oriented	37

Bilinear filtered, Mipmapped, fogged, texture-mapped Gouraud shaded, sub-pixel corrected triangles (no alpha-blending or Z-buffering)	**Ktriangles/sec**
10-pixel, right-angled, randomly oriented	828
25-pixel, right-angled, randomly oriented	823
50-pixel, right-angled, randomly oriented	655
1000-pixel, right-angled, randomly oriented	43

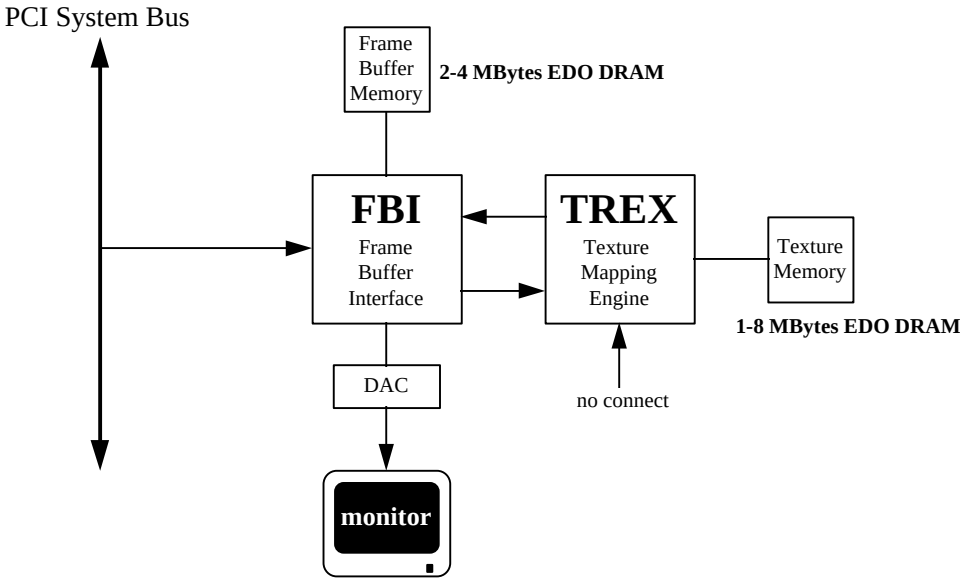
Bilinear filtered, Mipmapped, fogged, texture-mapped Gouraud shaded, sub-pixel corrected triangles with alpha-blending and Z-buffering	**Ktriangles/sec**
10-pixel, right-angled, randomly oriented	826
25-pixel, right-angled, randomly oriented	807
50-pixel, right-angled, randomly oriented	549
1000-pixel, right-angled, randomly oriented	37

Screen Clears	**msec**
RGB Buffer	3.45
Depth Buffer	3.45
RBG and Depth Buffer simultaneously	3.45

3. Architectural and Functional Overview

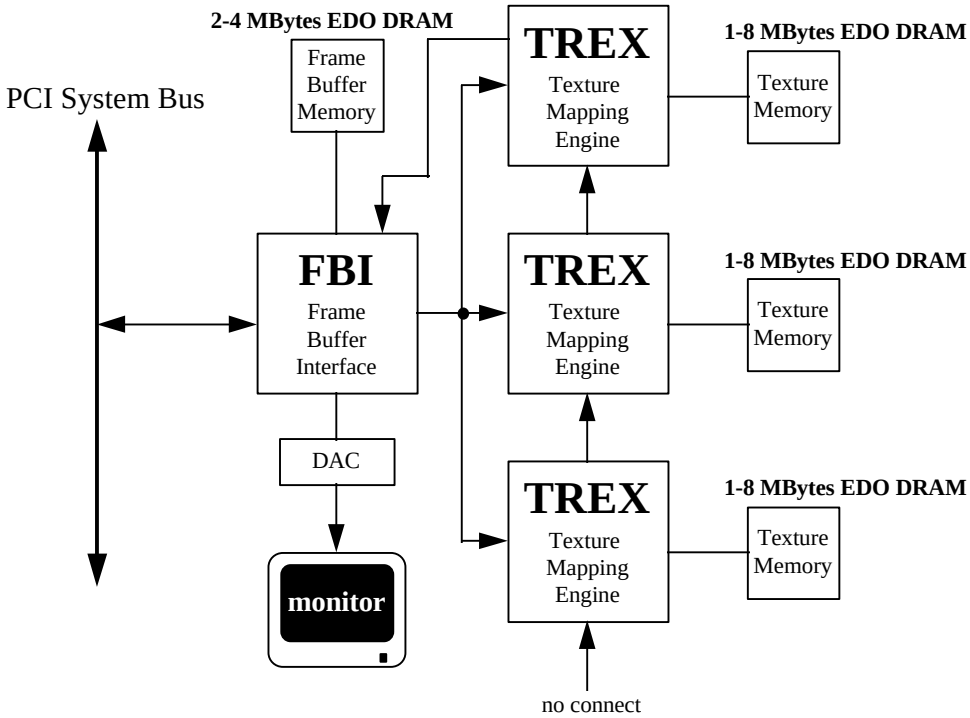
3.1 System Level Diagrams

In its entry level configuration, a SST-1 graphics solution consists of two rendering ASICS: TREX and FBI. FBI ("Frame Buffer Interface") serves as a PCI slave device, and all communication from the host CPU to the SST-1 graphics subsystem is performed through FBI. FBI implements basic 3D primitives including Gouraud shading, alpha blending, depth-buffering, and dithering. FBI also includes logic for the programmable fog table, and incorporates logic to handle all linear frame buffer accesses. Additionally, FBI includes a video display controller which controls output to the display monitor. TREX ("Texture Raster Engine") implements all texture mapping capabilities of the SST-1 graphics subsystem. TREX includes logic to support true-perspective texture mapping (dividing by W every pixel), level-of-detail (LOD) mipmapping, and bilinear filtering. Additionally, TREX implements advanced texture mapping techniques such as detail texture mapping, projected texture mapping, and trilinear texture filtering. Both FBI and TREX support various memory types including standard, Extended-Data-Out (EDO), and Synchronous DRAM to provide a wide range of price/performance options. Note in the single TREX SST-1 solution, the advanced texture mapping techniques of detail texture mapping, projected texture mapping, and trilinear texture filtering are two-pass operations. There is no performance penalty, however, for point-sampled or bilinear filtered texture mapping with mipmapping with the single TREX solution. The diagram below illustrates a base-level SST-1 graphics solution.



TREX includes a dedicated expansion bus which allows multiple TREX ASICs to be chained together. By chaining together multiple TREX ASICs, the performance of advanced texture mapping features such as detailed texture mapping, projected texture mapping, and trilinear filtering can be doubled. A two TREX SST-1 graphics solution allows single pass, full-speed, detail texture mapping, projected texture mapping, or trilinear filtering. The diagram below illustrates a two TREX graphics solution.

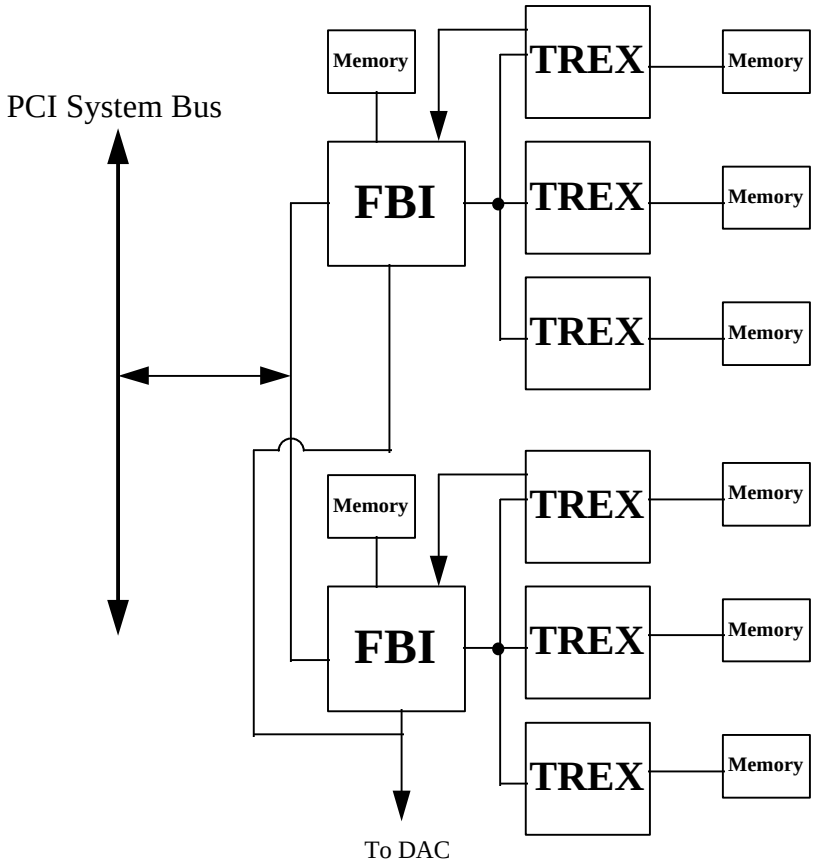
Three TREX ASICs can also be chained together to provide single-pass, full-speed rendering of all supported advanced texture mapping features including projected texture mapping. The diagram below illustrates the three TREX SST-1 graphics architecture:



The chart below provides performance characterization of advanced texture mapping rendering functionaltity for various SST-1 configurations.

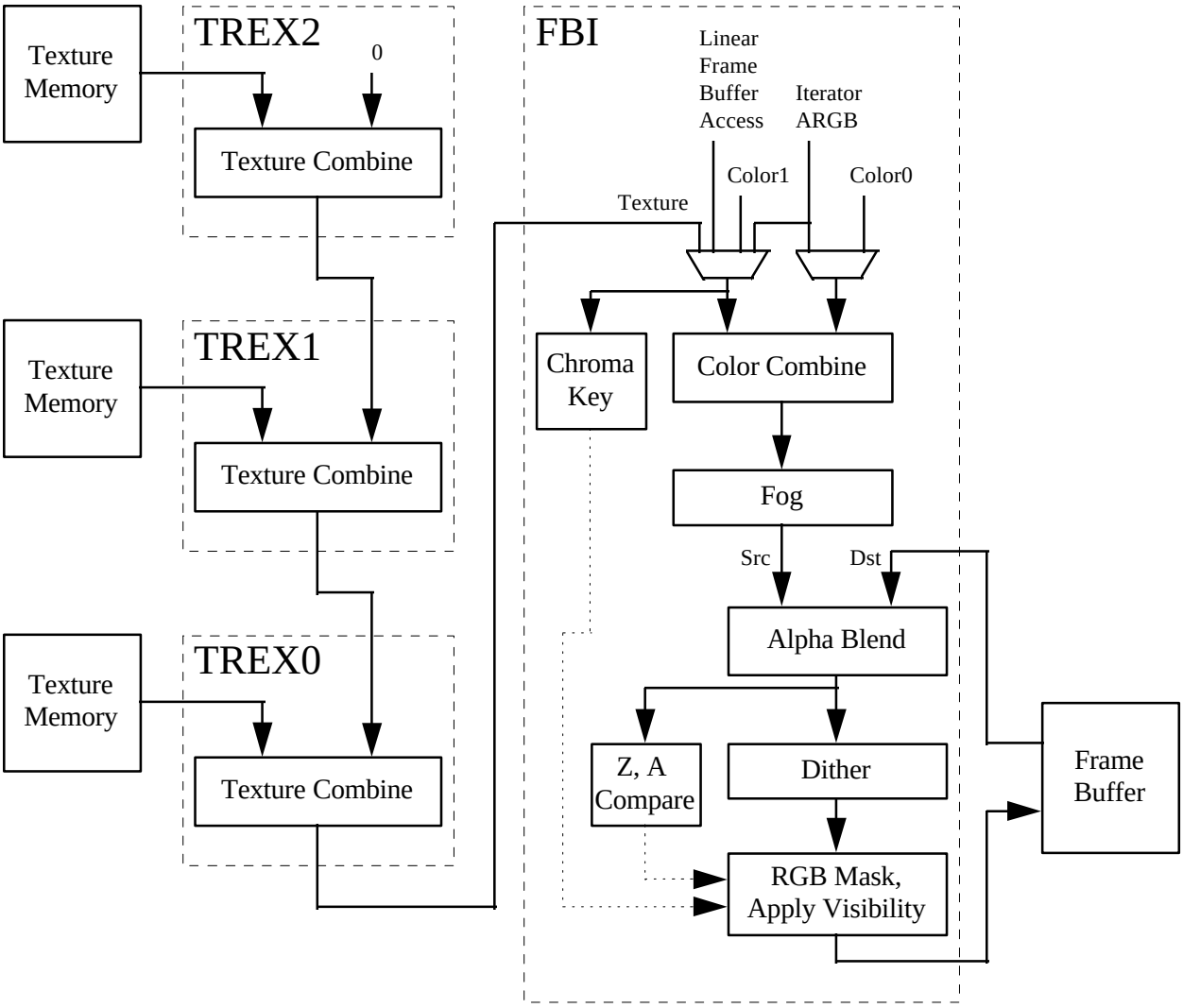
Texture Mapping Functionality	TREX Performance		
	One TREX	Two TREX	Three TREX
Point-sampled with mipmapping	One-Pass	One-Pass	One-Pass
Bilinear filtering with mipmapping	One-Pass	One-Pass	One-Pass
Bilinear filtering with mipmapping and projected textures	Two-Pass	One-Pass	One-Pass
Bilinear filtering with mipmapping and detail textures	Two-Pass	One-Pass	One-Pass
Bilinear filtering with mipmapping, projected and detail textures	Not supported	Two-Pass	One-Pass
Trilinear filtering with mipmapping	Two-Pass	One-Pass	One-Pass
Trilinear filtering with mipmapping and projected textures	Not supported	Two-Pass	One-Pass
Trilinear filtering with mipmapping and detail textures	Not supported	Two-Pass	One-Pass
Trilinear filtering with mipmapping, projected and detail textures	Not supported	Two-Pass	Two-Pass

For the highest possible rendering performance, multiple FBI/TREX subsystems can be chained together utilizing scan-line interleaving to effectvely double the rendering rate of a single FBI/TREX subsystem. The figure below illustrates this high-performance SST-1 architecture:



3.2 Architectural Overview

The diagram below illustrates the abstract rendering engine of the SST-1 graphics subsystem. The rendering engine is structured as a pipeline through which each pixel drawn to the screen must pass. The individual stages of the pixel pipeline modify pixels or make decisions about them.



3.3 Functional Overview

Bus Support: SST-1 implements the PCI bus protocol, and conforms to PCI bus specification 2.1. SST-1 is a slave only device, and supports zero-wait-state and burst transfers.

PCI Bus Write Posting: SST-1 uses an asynchronous FIFO 64 entries deep which allows sufficient write posting capabilities for high performance. The FIFO is asynchronous to the graphics engine, thus allowing the memory interface to operate at maximum frequency regardless of the frequency of the PCI bus. Zero-wait-state writes are supported for maximum bus bandwidth.

Memory FIFO: SST-1 can optionally use off-screen frame bufffer memory to increase the effective depth of the PCI Bus FIFO. The depth of this memory FIFO is programmable, and when used as an addition to the regular 64 entry host FIFO, allows up to 65536 host writes to be queued without stalling the PCI interface.

Memory Architecture: SST-1(FBI) contains a 64-bit wide interleaved datapath to RGB and alpha/depth-buffer memory with support for up to 50 MHz EDO DRAMs. For Gouraud-shaded or textured-mapped polygons with depth buffering enabled, one pixel is written per clock. This results in a 50 Mpixels/sec peak fill rate with an EDO DRAM configuration. For screen/depth-buffer clears, two pixels are written per clock, resulting in a 100 Mpixels/sec fill rate. 2 MBytes of memory is required to support 640x480x16 resolution with 16-bit depth buffering. Additionally, non-depth-buffered modes are supported with the 2 MByte RGB/depth-buffer configuration, including 640x480x16 triple-buffered and 800x600x16 double-buffered. 800x600x16 double-buffered with depth-buffering is supported with 4 MBytes of RGB/depth-buffer memory. The minimum amount of RGB/depth-buffer memory is 2 MBytes, with a maximum of 4 MBytes supported.

For storing texture bitmaps, SST-1(TREX) also contains a separate 64-bit wide datapath to texture memory. TREX provides support for EDO DRAM memory to be used as texture memory. The texture memory datapath is fully interleaved, which allows an individual bank to access data irrespective of the address used to access data in other banks. This interleaved architecture allows SST-1 to perform bilinear texture filtering with no performance penalty relative to point sampling. Another advantage of the interleaved architecture is that it imposes no additional memory cost, since texels are not duplicated in texture memory. The minimum amount of texture memory required is 1 MByte, with a maximum of 8 MBytes of texture memory supported.

Host Bus Addressing Schemes: SST-1 occupies 16 Mbytes of memory mapped address space. SST-1 does not utilize I/O mapped address space. The register space of SST-1 occupies 4 Mbytes of address space, the linear frame buffer access port occupies 4 Mbytes of address space, and the texture memory access port occupies the last 8 Mbytes of address space.

Linear Frame Buffer and Texture Access: SST-1 supports linear frame buffer and texture memory accesses for software ease and regular porting. Multiple color formats are supported for linear frame buffer writes, and all pixels written may optionally be passed through the normal SST-1 pixel pipeline for fogging, lighting, alpha blending, dithering, etc. of linear frame buffer writes. All texture maps are downloaded to local SST-1 texture memory through the texture memory access address space.

Triangle-based Rendering: SST-1 supports a triangle drawing primitive -- spans (both horizontal and vertical) and lines are rendered as special case triangles. Complex primitives such as quadrilaterals must be decomposed into triangles before they can be rendered by SST-1.

Gouraud-shaded Rendering: SST-1 supports Gouraud shading by providing RGBA iterators with rounding and clamping. The host provides starting RGBA and ΔRGBA information, and SST-1 automatically iterates RGBA values across the defined span or trapezoid.

Texture-mapped Rendering: SST-1 supports texture mapping for trapezoids and spans. The host provides starting texture S/W, T/W, 1/W, and their slopes Δ(S/W), Δ(T/W), and Δ(1/W) information. SST-1 automatically performs proper iteration and perspective correction necessary for true-perspective texture mapping. During each iteration of span/trapezoid walking, a division is performed by 1/W to correct for perspective distortion. Texture image dimensions must be powers of 2 and less than or equal to 256. Rectilinear and square texture bitmaps are supported.

Texture-mapped Rendering with Lighting: Texture-mapped rendering can be combined with Gouraud shading to introduce lighting effects during the texture mapping process. The host provides the starting Gouraud shading RGBA and slope ΔRGBA information, as well as the starting texture S/W, T/W, 1/W, and slope Δ(S/W), Δ(T/W), and Δ(1/W) information. SST-1 automatically performs the proper iteration and calculations required to implement the lighting models and texture lookups. A texel is either modulated (multiplied by), added, or blended to the Gouraud shaded color. The selection of color modulation or addition is programmable.

Texture Mapping Anti-aliasing: SST-1 allows for anti-aliasing of texture-mapped rendering with support for texture filtering and mipmapping. SST-1 supports point-sampled, bilinear, and trilinear texture filters. While point-sampled and bilinear are single pass operations, single TREX SST-1 graphics solutions require two-passes for trilinear texture filtering. Multiple TREX SST-1 graphics solutions support trilinear texture filtering as a single-pass operation. Note that regardless of the number of TREX ASICs in a given SST-1 graphics solution, there is no performance difference between point-sampled and bilinear filtered texture-mapped rendering.

In addition to supporting texture filtering, SST-1 also supports texture mipmapping. SST-1 automatically determines the mipmap level based on the mipmap equation, and selects the proper texture image to be accessed. When performing point-sampled or bilinear filtered texture mapping, dithering of the mipmap levels can optionally be used to remove mipmap “banding” during rendering. Using dithered mipmapping with bilinear filtering results in images almost indistinguishable from full trilinear filtered images.

Texture-space Decompression: Texture data compression is accomplished using a patent-pending “narrow channel” YIQ compression scheme. 8-bit YIQ format is supported. The compression is based on an algorithm which compresses 24-bit RGB to a 8-bit YIQ format with little loss in precision. This YIQ compression algorithm is especially suited to texture mapping, as textures typically contain very similar color components. The algorithm is performed by the host CPU, and YIQ compressed textures are passed to SST-1 The advantages of using compressed textures are increased effective texture storage space and lower bandwidth requirements to perform texture filtering.

Depth-Buffered Rendering: SST-1 supports hardware accelerated depth-buffered rendering with no performance penalty when enabled. With 2 MBytes of frame buffer memory, 640x480x16 resolution, double buffered with a 16-bit Z-buffer is supported. To eliminate many of the Z-aliasing problems typically found on 16-bit Z-buffer graphics solutions, SST-1 allows the (1/W) parameter to be used as the depth component for hardware-accelerated depth-buffered rendering. When the (1/W) parameter is used for depth-buffering, a16-bit floating point format is supported. A 16-bit floating point (1/W)-buffer provides much greater precision and dynamic range than a standard 16-bit Z-buffer, and reduces many of the Z-aliasing problems found on 16-bit Z-buffer systems. An additional benefit of using the (1/W) component for depth-buffering is that the host CPU no longer needs to setup the Z component for a given polygon -- the (1/W) component must be setup to perform perspective-corrected texture mapping anyway, and using this parameter for depth-buffering eliminates the need for a separate Z parameter.

Whether to use an integer-Z or floating-point-(1/W) is software programmable. If hardware-accelerated depth-buffering is not required, additional memory may be used to increase screen resolution. With 2 MBytes of frame buffer memory, 800x600x16, double buffered resolutions may be used if hardware-accelerated depth-buffering is not required.

Pixel Blending Operations: SST-1 supports alpha blending functions which allow incoming source pixels to be blended with current destination pixels with no performance penalty when enabled. An alpha channel (i.e. destination alpha) stored in offscreen memory is only supported when depth-buffering is disabled. [TO BE COMPLETED]. The alpha blending function is as follows:

$$D_{new} \Leftarrow (S \cdot \alpha) + (D_{old} \cdot \beta)$$

where

D_{new}	The new destination pixel being written into the frame buffer
S	The new source pixel being generated
D_{old}	The old (current) destination pixel about to be modified
α	The source pixel alpha function.
β	The destination pixel alpha function.

Special Effects: TO BE COMPLETED (e.g. translucent billboards, spotlights, diffuse lighted texture, translucent cellophane [translucent filter]).

FOG: SST-1 supports a 64-entry lookup table to support atmospheric effects such as fog and haze. When enabled, a 6-bit floating point representation of (1/W) is used to index into the 64-entry lookup table. The output of the lookup table is an “alpha” value which represents the level of blending to be performed between the static fog/haze color and the incoming pixel color. Low order bits of the floating point (1/W) are used to blend between multiple entries of the lookup table to reduce fog “banding.” The fog lookup table is loaded by the host CPU, so various fog equations, colors, and effects are supported.

Color Modes: SST-1 supports 16-bit RGB buffer displays only. The host may transfer 24-bit pixels to SST-1, and color dithering is utilized to convert the input pixels to native 16-bit format with no performance penalty.

Chroma-Key Operation: SST-1 supports a chroma-key operation used for transparent object effects. When enabled, an outgoing pixel is compared with the chroma-key register. If a match is detected, the outgoing pixel is invalidated in the pixel pipeline, and the frame buffer is not updated.

Color Dithering Operations: All operations internal to SST-1 operate in native 24-bit pixel mode. However, color dithering from 24-bit pixels to 16-bit pixels is provided on the back end of the pixel pipeline. Using the color dithering option, the host can pass 24-bit pixels to SST-1, which converts the 24-bit incoming pixels to 16-bit pixels which are then stored in the 16-bit RGB buffer. The 16-bit color dithering allows for the generation of photorealistic images without the additional cost of a true color frame buffer storage area.

Programmable Video Timing: SST-1 uses a programmable video timing controller which allows for very flexible video timing. Any monitor type may be used with SST-1 , with 76+ Hz vertical refresh rates supported at 800x600 resolution, and 100+ Hz vertical refresh rates supported at 640x480 resolution.

Gamma Correction: SST-1 uses a programmable color lookup table to allow for programmable gamma correction. The 16-bit dithered color data from the frame buffer is used an an index into the gamma-correction color table -- the 24-bit output of the gamma-correction color table is then fed to the monitor.

External DAC Support: SST-1 is compatible with industry standard RAMDACs and DACs. The DAC interface is identical to that provided by popular graphics accelerators such as the S3 864 and the Tseng Labs W32p.

4. SST-1 Address Space

SST-1 requires 16 Mbytes of memory mapped address space. SST-1 does not utilize I/O mapped memory. The memory mapped address space is shown below:

Address	Description
0x000000-0x3fffff	SST-1 memory mapped register set (4 MBytes)
0x400000-0x7fffff	SST-1 linear frame buffer access (4 MBytes)
0x800000-0xffffff	SST-1 texture memory access (8 MBytes)

The physical memory address for SST-1 accesses is calculated by adding the SST-1 address offset (0-16 MBytes) to the SST-1 base address register. The SST-1 base address register, **memBaseAddr**, is located in PCI configuration space. **memBaseAddr** is setup by the PCI System BIOS during system poweron and initialization and should not be modified by software. See section 5 for more information on the memory mapped register set, section 6 for more information on the PCI configuration space, section 8 for more information on linear frame buffer access, and section 9 for more information on texture memory access.

5. Memory Mapped Register Set

A 4 Mbyte (22-bit) SST-1 memory mapped register address is divided into the following fields:

Wrap	Chip	Register	Byte
8	4	8	2

The **wrap** field aliases multiple 14-bit register maps. The **wrap** field is useful for processors such as the Digital Alpha AXP which contains large writebuffers which collapse multiple writes to the same address into a single write (an undesirable effect when programming SST-1). By writing to different **wraps**, software can guarantee that writes are not collapsed in the write buffer. Note that SST-1 functionality is identical regardless of which **wrap** is accessed. The **chip** field selects one or more of the SST-1 chips (FBI and/or TREX) to be accessed. Each bit in the **chip** field selects one chip for writing, with FBI controlled by the lsb of the **chip** field, and TREX#2 controlled by the msb of the **chip** field. Note the **chip** field value of 0x0 selects all chips. The following table shows the **chip** field mappings:

Chip Field	SST-1 Chip Accessed
0000	FBI + all TREX chips
0001	FBI
0010	TREX #0
0011	FBI + TREX #0
0100	TREX #1
0101	FBI + TREX #1
0110	TREX #0 + TREX #1
0111	FBI + TREX #0 + TREX #1
1000	TREX #2
1001	FBI + TREX #2
1010	TREX #0 + TREX #2
1011	FBI + TREX #0 + TREX #2
1100	TREX #1 + TREX #2
1101	FBI + TREX #1 + TREX #2
1110	TREX #0 + TREX #1 + TREX #2
1111	FBI + all TREX chips

Note that TREX #0 is always connected to FBI in the system level diagrams of section 3, and TREX #1 is attached to TREX #0, etc. By utilizing the different **chip** fields, software can precisely control the data presented to individual chips which compose the SST-1 graphics subsystem. Note that for reads, the **chip** field is ignored, and read data is always read from FBI. The **register** field selects the register to be accessed from the table below. All accesses to the memory mapped registers must be 32-bit accesses. No byte (8-bit) or halfword (16-bit) accesses are allowed to the memory mapped registers, so the **byte** (2-bit) field of all memory mapped register accesses must be 0x0. As a result, to modify individual bits of a 32-bit register, the entire 32-bit word must be written with valid bits in all positions.

The table below shows the SST-1 register set. The register set shown below is the address map when triangle registers address aliasing (remapping) is disabled(**fbiinit3**(0)=0). When The **chip** column illustrates which registers are stored in which chips. For the registers which are stored in TREX, the % symbol specifies that the register is unconditionally written to TREX regardless of the chip address. Similarly, the * symbol specifies that the register is only written to a given TREX if specified in the chip address. The **R/W** column illustrates the

read/write status of individual registers. Reading from a register which is "write only" returns undefined data. Also, reading from a register that is TREX specific returns undefined data.. Reads from all other memory mapped registers only contain valid data in the bits stored by the registers, and undefined/reserved bits in a given register must be masked by software. The **sync** column indicates whether the graphics processor must wait for the current command to finish before loading a particular register from the FIFO. A "yes" in the **sync** column means the graphics processor will flush the data pipeline before loading the register -- this will result in a small performance degradation when compared to those registers which do not need synchronization. The **FIFO** column indicates whether a write to a particular register will be pushed into the PCI bus FIFO. Care must be taken when writing to those registers not pushed into the FIFO in order to prevent race conditions between FIFOed and non-FIFOed registers. Also note that reads are not pushed into the PCI bus FIFO, and reading FIFOed registers will return the current value of the register, irrespective of pending writes to the register present in the FIFO.

Register Name	Address	Bits	Chip	R/W	Sync? /Fifo?	Description
status	0x000(0)	31:0	FBI	R/W	No / Yes	SST-1 Status
reserved	0x004(4)	n/a	n/a	n/a	n/a	
vertexAx	0x008(8)	15:0	FBI+TREX[%]	W	No / Yes	Vertex A x-coordinate location (12.4 format)
vertexAy	0x00c(12)	15:0	FBI+TREX[%]	W	No / Yes	Vertex A y-coordinate location (12.4 format)
vertexBx	0x010(16)	15:0	FBI+TREX[%]	W	No / Yes	Vertex B x-coordinate location (12.4 format)
vertexBy	0x014(20)	15:0	FBI+TREX[%]	W	No / Yes	Vertex B y-coordinate location (12.4 format)
vertexCx	0x018(24)	15:0	FBI+TREX[%]	W	No / Yes	Vertex C x-coordinate location (12.4 format)
vertexCy	0x01c(28)	15:0	FBI+TREX[%]	W	No / Yes	Vertex C y-coordinate location (12.4 format)
startR	0x020(32)	23:0	FBI	W	No / Yes	Starting Red parameter (12.12 format)
startG	0x024(36)	23:0	FBI	W	No / Yes	Starting Green parameter (12.12 format)
startB	0x028(40)	23:0	FBI	W	No / Yes	Starting Blue parameter (12.12 format)
startZ	0x02c(44)	31:0	FBI	W	No / Yes	Starting Z parameter (20.12 format)
startA	0x030(48)	23:0	FBI	W	No / Yes	Starting Alpha parameter (12.12 format)
startS	0x034(52)	31:0	TREX[*]	W	No / Yes	Starting S/W parameter (14.18 format)
startT	0x038(56)	31:0	TREX[*]	W	No / Yes	Starting T/W parameter (14.18 format)
startW	0x03c(60)	31:0	FBI+TREX[*]	W	No / Yes	Starting 1/W parameter (2.30 format)
dRdX	0x040(64)	23:0	FBI	W	No / Yes	Change in Red with respect to X (12.12 format)
dGdX	0x044(68)	23:0	FBI	W	No / Yes	Change in Green with respect to X (12.12 format)
dBdX	0x048(72)	23:0	FBI	W	No / Yes	Change in Blue with respect to X (12.12 format)
dZdX	0x04c(76)	31:0	FBI	W	No / Yes	Change in Z with respect to X (20.12 format)
dAdX	0x050(80)	23:0	FBI	W	No / Yes	Change in Alpha with respect to X (12.12 format)
dSdX	0x054(84)	31:0	TREX[*]	W	No / Yes	Change in S/W with respect to X (14.18 format)
dTdX	0x058(88)	31:0	TREX[*]	W	No / Yes	Change in T/W with respect to X (14.18 format)
dWdX	0x05c(92)	31:0	FBI+TREX[*]	W	No / Yes	Change in 1/W with respect to X (2.30 format)
dRdY	0x060(96)	23:0	FBI	W	No / Yes	Change in Red with respect to Y (12.12 format)
dGdY	0x064(100)	23:0	FBI	W	No / Yes	Change in Green with respect to Y (12.12 format)
dBdY	0x068(104)	23:0	FBI	W	No / Yes	Change in Blue with respect to Y (12.12 format)
dZdY	0x06c(108)	31:0	FBI	W	No / Yes	Change in Z with respect to Y (20.12 format)
dAdY	0x070(112)	23:0	FBI	W	No / Yes	Change in Alpha with respect to Y (12.12 format)
dSdY	0x074(116)	31:0	TREX[*]	W	No / Yes	Change in S/W with respect to Y (14.18 format)
dTdY	0x078(120)	31:0	TREX[*]	W	No / Yes	Change in T/W with respect to Y (14.18 format)
dWdY	0x07c(124)	31:0	FBI+TREX[*]	W	No / Yes	Change in 1/W with respect to Y (2.30 format)
triangleCMD	0x080(128)	31	FBI+TREX[%]	W	No / Yes	Execute TRIANGLE command (floating point)
reserved	0x084(132)	n/a	n/a	W	n/a	

fvertexAx	0x088(136)	31:0	FBI+TREX[%]	W	No / Yes	Vertex A x-coordinate location (floating point)
fvertexAy	0x08c(140)	31:0	FBI+TREX[%]	W	No / Yes	Vertex A y-coordinate location (floating point)
fvertexBx	0x090(144)	31:0	FBI+TREX[%]	W	No / Yes	Vertex B x-coordinate location (floating point)
fvertexBy	0x094(148)	31:0	FBI+TREX[%]	W	No / Yes	Vertex B y-coordinate location (floating point)
fvertexCx	0x098(152)	31:0	FBI+TREX[%]	W	No / Yes	Vertex C x-coordinate location (floating point)
fvertexCy	0x09c(156)	31:0	FBI+TREX[%]	W	No / Yes	Vertex C y-coordinate location (floating point)
fstartR	0x0a0(160)	31:0	FBI	W	No / Yes	Starting Red parameter (floating point)
fstartG	0x0a4(164)	31:0	FBI	W	No / Yes	Starting Green parameter (floating point)
fstartB	0x0a8(168)	31:0	FBI	W	No / Yes	Starting Blue parameter (floating point)
fstartZ	0x0ac(172)	31:0	FBI	W	No / Yes	Starting Z parameter (floating point)
fstartA	0x0b0(176)	31:0	FBI	W	No / Yes	Starting Alpha parameter (floating point)
fstartS	0x0b4(180)	31:0	TREX[*]	W	No / Yes	Starting S/W parameter (floating point)
fstartT	0x0b8(184)	31:0	TREX[*]	W	No / Yes	Starting T/W parameter (floating point)
fstartW	0x0bc(188)	31:0	FBI+TREX[*]	W	No / Yes	Starting 1/W parameter (floating point)
fdRdX	0x0c0(192)	31:0	FBI	W	No / Yes	Change in Red with respect to X (floating point)
fdGdX	0x0c4(196)	31:0	FBI	W	No / Yes	Change in Green with respect to X (floating point)
fdBdX	0x0c8(200)	31:0	FBI	W	No / Yes	Change in Blue with respect to X (floating point)
fdZdX	0x0cc(204)	31:0	FBI	W	No / Yes	Change in Z with respect to X (floating point)
fdAdX	0x0d0(208)	31:0	FBI	W	No / Yes	Change in Alpha with respect to X (floating point)
fdSdX	0x0d4(212)	31:0	TREX[*]	W	No / Yes	Change in S/W with respect to X (floating point)
fdTdX	0x0d8(216)	31:0	TREX[*]	W	No / Yes	Change in T/W with respect to X (floating point)
fdWdX	0x0dc(220)	31:0	FBI+TREX[*]	W	No / Yes	Change in 1/W with respect to X (floating point)
fdRdY	0x0e0(224)	31:0	FBI	W	No / Yes	Change in Red with respect to Y (floating point)
fdGdY	0x0e4(228)	31:0	FBI	W	No / Yes	Change in Green with respect to Y (floating point)
fdBdY	0x0e8(232)	31:0	FBI	W	No / Yes	Change in Blue with respect to Y (floating point)
fdZdY	0x0ec(236)	31:0	FBI	W	No / Yes	Change in Z with respect to Y (floating point)
fdAdY	0x0f0(240)	31:0	FBI	W	No / Yes	Change in Alpha with respect to Y (floating point)
fdSdY	0x0f4(244)	31:0	TREX[*]	W	No / Yes	Change in S/W with respect to Y (floating point)
fdTdY	0x0f8(248)	31:0	TREX[*]	W	No / Yes	Change in T/W with respect to Y (floating point)
fdWdY	0x0fc(252)	31:0	FBI+TREX[*]	W	No / Yes	Change in 1/W with respect to Y (floating point)
ftriangleCMD	0x100(256)	31	FBI+TREX[%]	W	No / Yes	Execute TRIANGLE command (floating point)
fbzColorPath	0x104(260)	27:0	FBI+TREX[%]	R/W	No / Yes	FBI Color Path Control
fogMode	0x108(264)	5:0	FBI	R/W	No / Yes	Fog Mode Control
alphaMode	0x10c(268)	31:0	FBI	R/W	No / Yes	Alpha Mode Control
fbzMode	0x110(272)	20:0	FBI	R/W	Yes / Yes	RGB Buffer and Depth-Buffer Control
lfbMode	0x114(276)	16:0	FBI	R/W	Yes / Yes	Linear Frame Buffer Mode Control
clipLeftRight	0x118(280)	31:0	FBI	R/W	Yes / Yes	Left and Right of Clipping Register
clipLowYHighY	0x11c(284)	31:0	FBI	R/W	Yes / Yes	Top and Bottom of Clipping Register
nopCMD	0x120(288)	0	FBI+TREX[%]	W	Yes / Yes	Execute NOP command
fastfillCMD	0x124(292)	n/a	FBI	W	Yes / Yes	Execute FASTFILL command
swapbufferCMD	0x128(296)	8:0	FBI	W	Yes / Yes	Execute SWAPBUFFER command
fogColor	0x12c(300)	23:0	FBI	W	Yes / Yes	Fog Color Value
zaColor	0x130(304)	31:0	FBI	W	Yes / Yes	Constant Alpha/Depth Value
chromaKey	0x134(308)	23:0	FBI	W	Yes / Yes	Chroma Key Compare Value

reserved	0x138(312)	n/a	n/a	n/a	n/a	
reserved	0x13c(316)	n/a	n/a	n/a	n/a	
stipple	0x140(320)	31:0	FBI	R/W	Yes / Yes	Rendering Stipple Value
color0	0x144(324)	31:0	FBI	R/W	Yes / Yes	Constant Color #0
color1	0x148(328)	31:0	FBI	R/W	Yes / Yes	Constant Color #1
fbiPixelsIn	0x14c(332)	23:0	FBI	R	n/a	Pixel Counter (Number pixels processed)
fbiChromaFail	0x150(336)	23:0	FBI	R	n/a	Pixel Counter (Number pixels failed Chroma test)
fbiZfuncFail	0x154(340)	23:0	FBI	R	n/a	Pixel Counter (Number pixels failed Z test)
fbiAfuncFail	0x158(344)	23:0	FBI	R	n/a	Pixel Counter (Number pixels failed Alpha test)
fbiPixelsOut	0x15c(348)	23:0	FBI	R	n/a	Pixel Counter (Number pixels drawn)
fogTable	0x160(352) to 0x1dc(476)	31:0	FBI	W	Yes / Yes	Fog Table
reserved	0x1e0(480) to 0x1fc(508)	n/a	n/a	n/a	n/a	
fbiInit4	0x200(512)	12:0	FBI	R/W	(n/a) / No	FBI Hardware Initialization (register 4)
vRetrace	0x204(516)	11:0	FBI	R	(n/a) / No	Vertical Retrace Counter
backPorch	0x208(520)	23:0	FBI	R/W	(n/a) / No	Video Backporch Timing Generator
videoDimensions	0x20c(524)	25:0	FBI	R/W	(n/a) / No	Video Screen Dimensions
fbiInit0	0x210(528)	31:0	FBI	R/W	(n/a) / No	FBI Hardware Initialization (register 0)
fbiInit1	0x214(532)	31:0	FBI	R/W	(n/a) / No	FBI Hardware Initialization (register 1)
fbiInit2	0x218(536)	31:0	FBI	R/W	(n/a) / No	FBI Hardware Initialization (register 2)
fbiInit3	0x21c(540)	31:0	FBI	R/W	(n/a) / No	FBI Hardware Initialization (register 3)
hSync	0x220(544)	25:0	FBI	W	(n/a) / No	Horizontal Sync Timing Generator
vSync	0x224(548)	27:0	FBI	W	(n/a) / No	Vertical Sync Timing Generator
clutData	0x228(552)	29:0	FBI	W	(n/a) / No	Internal Color Lookup Table Initialization
dacData	0x22c(556)	31:0	FBI	W	(n/a) / No	External DAC Initialization
maxRgbDelta	0x230(560)	23:0	FBI	W	(n/a) / No	Max. RGB difference for Video Filtering
reserved	0x234(564) to 0x2fc(764)	n/a	n/a	n/a	n/a	
textureMode	0x300(768)	30:0	TREX*	W	No / Yes	Texture Mode Control
tLOD	0x304(772)	23:0	TREX*	W	No / Yes	Texture LOD Settings
tDetail	0x308(776)	16:0	TREX*	W	No / Yes	Texture LOD Settings
texBaseAddr	0x30c(780)	18:0	TREX*	W	No / Yes	Texture Base Address
texBaseAddr_1	0x310(784)	18:0	TREX*	W	No / Yes	Texture Base Address (supplemental LOD 1)
texBaseAddr_2	0x314(788)	18:0	TREX*	W	No / Yes	Texture Base Address (supplemental LOD 2)
texBaseAddr_3_8	0x318(792)	18:0	TREX*	W	No / Yes	Texture Base Address (supplemental LOD 3-8)
trexInit0	0x31c(796)	31:0	TREX*	W	Yes / Yes	TREX Hardware Initialization (register 0)
trexInit1	0x320(800)	31:0	TREX*	W	Yes / Yes	TREX Hardware Initialization (register 1
nccTable0	0x324(804) to 0x350(848)	31:0 or 26:0	TREX*	W	Yes / Yes	Narrow Channel Compression Table 0 (12 entries)

nccTable1	0x354(852) to 0x380(896)	31:0 or 26:0	TREX*	W	Yes / Yes	Narrow Channel Compression Table 1 (12 entries)
reserved	0x384(900) to 0x3fc(1020)	n/a	n/a	n/a	n/a	

When **fbiinit3**(0)=1, the triangle parameter registers can be aliased to a different address mapping to improve PCI bus throughput. When **fbiinit3**(0)=1 and the upper bit of the **wrap** field in the pci address is 0x1 (**pci_ad**[21]=1), the following table shows the addresses for the triangle parameter registers. Note that enabling triangle parameter remapping (**fbiinit3**(0)=1) has no affect any registers not specified in the table below.

Register Name	Address	Bits	Chip	R/W	Sync? /Fifo?	Description
status	0x000(0)	31:0	FBI	R/W	No / Yes	SST-1 Status
reserved	0x004(4)	n/a	n/a	n/a	n/a	
vertexAx	0x008(8)	15:0	FBI+TREX%	W	No / Yes	Vertex A x-coordinate location (12.4 format)
vertexAy	0x00c(12)	15:0	FBI+TREX%	W	No / Yes	Vertex A y-coordinate location (12.4 format)
vertexBx	0x010(16)	15:0	FBI+TREX%	W	No / Yes	Vertex B x-coordinate location (12.4 format)
vertexBy	0x014(20)	15:0	FBI+TREX%	W	No / Yes	Vertex B y-coordinate location (12.4 format)
vertexCx	0x018(24)	15:0	FBI+TREX%	W	No / Yes	Vertex C x-coordinate location (12.4 format)
vertexCy	0x01c(28)	15:0	FBI+TREX%	W	No / Yes	Vertex C y-coordinate location (12.4 format)
startR	0x020(32)	23:0	FBI	W	No / Yes	Starting Red parameter (12.12 format)
dRdX	0x024(36)	23:0	FBI	W	No / Yes	Change in Red with respect to X (12.12 format)
dRdY	0x028(40)	23:0	FBI	W	No / Yes	Change in Red with respect to Y (12.12 format)
startG	0x02c(44)	23:0	FBI	W	No / Yes	Starting Green parameter (12.12 format)
dGdX	0x030(48)	23:0	FBI	W	No / Yes	Change in Green with respect to X (12.12 format)
dGdY	0x034(52)	23:0	FBI	W	No / Yes	Change in Green with respect to Y (12.12 format)
startB	0x038(56)	23:0	FBI	W	No / Yes	Starting Blue parameter (12.12 format)
dBdX	0x03c(60)	23:0	FBI	W	No / Yes	Change in Blue with respect to X (12.12 format)
dBdY	0x040(64)	23:0	FBI	W	No / Yes	Change in Blue with respect to Y (12.12 format)
startZ	0x044(68)	31:0	FBI	W	No / Yes	Starting Z parameter (20.12 format)
dZdX	0x048(72)	31:0	FBI	W	No / Yes	Change in Z with respect to X (20.12 format)
dZdY	0x04c(76)	31:0	FBI	W	No / Yes	Change in Z with respect to Y (12.12 format)
startA	0x050(80)	23:0	FBI	W	No / Yes	Starting Alpha parameter (12.12 format)
dAdX	0x054(84)	23:0	FBI	W	No / Yes	Change in Alpha with respect to X (12.12 format)
dAdY	0x058(88)	23:0	FBI	W	No / Yes	Change in Alpha with respect to Y (12.12 format)
startS	0x05c(92)	31:0	TREX*	W	No / Yes	Starting S/W parameter (14.18 format)
dSdX	0x060(96)	31:0	TREX*	W	No / Yes	Change in S/W with respect to X (14.18 format)
dSdY	0x064(100)	31:0	TREX*	W	No / Yes	Change in S/W with respect to Y (14.18 format)
startT	0x068(104)	31:0	TREX*	W	No / Yes	Starting T/W parameter (14.18 format)
dTdX	0x06c(108)	31:0	TREX*	W	No / Yes	Change in T/W with respect to X (14.18 format)
dTdY	0x070(112)	31:0	TREX*	W	No / Yes	Change in T/W with respect to Y (14.18 format)
startW	0x074(116)	31:0	FBI+TREX*	W	No / Yes	Starting 1/W parameter (2.30 format)
dWdX	0x078(120)	31:0	FBI+TREX*	W	No / Yes	Change in 1/W with respect to X (2.30 format)
dWdY	0x07c(124)	31:0	FBI+TREX*	W	No / Yes	Change in 1/W with respect to Y (2.30 format)
triangleCMD	0x080(128)	31	FBI+TREX%	W	No / Yes	Execute TRIANGLE command (sign bit)
reserved	0x084(132)	n/a	n/a	W	n/a	
fvertexAx	0x088(136)	31:0	FBI+TREX%	W	No / Yes	Vertex A x-coordinate location (floating point)
fvertexAy	0x08c(140)	31:0	FBI+TREX%	W	No / Yes	Vertex A y-coordinate location (floating point)
fvertexBx	0x090(144)	31:0	FBI+TREX%	W	No / Yes	Vertex B x-coordinate location (floating point)
fvertexBy	0x094(148)	31:0	FBI+TREX%	W	No / Yes	Vertex B y-coordinate location (floating point)
fvertexCx	0x098(152)	31:0	FBI+TREX%	W	No / Yes	Vertex C x-coordinate location (floating point)
fvertexCy	0x09c(156)	31:0	FBI+TREX%	W	No / Yes	Vertex C y-coordinate location (floating point)
fstartR	0x0a0(160)	31:0	FBI	W	No / Yes	Starting Red parameter (floating point)

fdRdX	0x0a4(164)	31:0	FBI	W	No / Yes	Change in Red with respect to X (floating point)
fdRdY	0x0a8(168)	31:0	FBI	W	No / Yes	Change in Red with respect to Y (floating point)
fstartG	0x0ac(172)	31:0	FBI	W	No / Yes	Starting Green parameter (floating point)
fdGdX	0x0b0(176)	31:0	FBI	W	No / Yes	Change in Green with respect to X (floating point)
fdGdY	0x0b4(180)	31:0	FBI	W	No / Yes	Change in Green with respect to Y (floating point)
fstartB	0x0b8(184)	31:0	FBI	W	No / Yes	Starting Blue parameter (floating point)
fdBdX	0x0bc(188)	31:0	FBI	W	No / Yes	Change in Blue with respect to X (floating point)
fdBdY	0x0c0(192)	31:0	FBI	W	No / Yes	Change in Blue with respect to Y (floating point)
fstartZ	0x0c4(196)	31:0	FBI	W	No / Yes	Starting Z parameter (floating point)
fdZdX	0x0c8(200)	31:0	FBI	W	No / Yes	Change in Z with respect to X (floating point)
fdZdY	0x0cc(204)	31:0	FBI	W	No / Yes	Change in Z with respect to Y (floating point)
fstartA	0x0d0(208)	31:0	FBI	W	No / Yes	Starting Alpha parameter (floating point)
fdAdX	0x0d4(212)	31:0	FBI	W	No / Yes	Change in Alpha with respect to X (floating point)
fdAdY	0x0d8(216)	31:0	FBI	W	No / Yes	Change in Alpha with respect to Y (floating point)
fstartS	0x0dc(220)	31:0	TREX*	W	No / Yes	Starting S/W parameter (floating point)
fdSdX	0x0e0(224)	31:0	TREX*	W	No / Yes	Change in S/W with respect to X (floating point)
fdSdY	0x0e4(228)	31:0	TREX*	W	No / Yes	Change in S/W with respect to Y (floating point)
fstartT	0x0e8(232)	31:0	TREX*	W	No / Yes	Starting T/W parameter (floating point)
fdTdX	0x0ec(236)	31:0	TREX*	W	No / Yes	Change in T/W with respect to X (floating point)
fdTdY	0x0f0(240)	31:0	TREX*	W	No / Yes	Change in T/W with respect to Y (floating point)
fstartW	0x0f4(244)	31:0	FBI+TREX*	W	No / Yes	Starting 1/W parameter (floating point)
fdWdX	0x0f8(248)	31:0	FBI+TREX*	W	No / Yes	Change in 1/W with respect to X (floating point)
fdWdY	0x0fc(252)	31:0	FBI+TREX*	W	No / Yes	Change in 1/W with respect to Y (floating point)
ftriangleCMD	0x100(256)	31	FBI+TREX%	W	No / Yes	Execute TRIANGLE command (floating point)

5.1 status Register

The **status** register provides a way for the CPU to interrogate the graphics processor about its current state and FIFO availability. The **status** register is read only, but writing to **status** clears any SST-1 generated PCI interrupts.

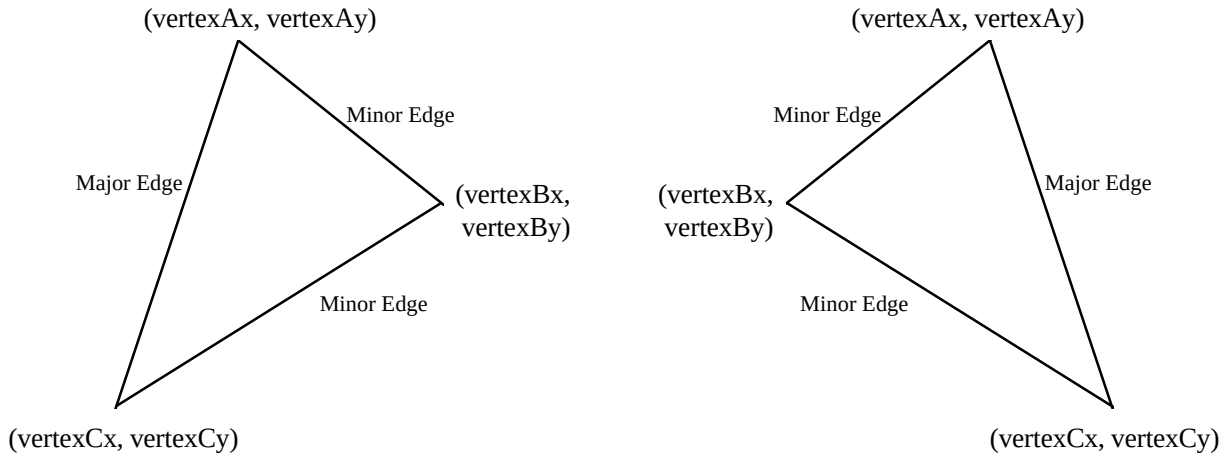
Bit	Description
5:0	PCI FIFO freespace (0x3f=FIFO empty). Default is 0x3f.
6	Vertical retrace (0=Vertical retrace active, 1=Vertical retrace inactive). Default is 1.
7	FBI graphics engine busy (0=engine idle, 1=engine busy). Default is 0.
8	TREX busy (0=engine idle, 1=engine busy). Default is 0.
9	SST-1 busy (0=idle, 1=busy). Default is 0.
11:10	Displayed buffer (0=buffer 0, 1=buffer 1, 2=auxiliary buffer, 3=reserved). Default is 0.
27:12	Memory FIFO freespace (0xffff=FIFO empty). Default is 0xffff.
30:28	Swap Buffers Pending. Default is 0x0.
31	PCI Interrupt Generated. Default is 0x0. (not currently implemented).

Bits(5:0) show the number of entries available in the internal host FIFO. The internal host FIFO is 64 entries deep. The FIFO is empty when bits(5:0)=0x3f. Bit(6) is the state of the monitor vertical retrace signal, and is used to determine when the monitor is being refreshed. Bit(7) of **status** is used to determine if the graphics engine of FBI is active. Note that bit(7) only determines if the graphics engine of FBI is busy -- it does not include information as to the status of the internal PCI FIFOs. Bit(8) of **status** is used to determine if TREX is busy. Note

that bit(8) of **status** is set if any unit in TREX is not idle -- this includes the graphics engine and all internal TREX FIFOs. Bit(9) of **status** determines if all units in the SST-1 system (including graphics engines, FIFOs, etc.) are idle. Bit(9) is set when any internal unit in SST-1 is active (e.g. graphics is being rendered or any FIFO is not empty). Bits(11:10) show which RGB buffer is used for monitor refresh. SST-1 uses the values of bits(11:10) to determine the source of the RGB data that is sent to the monitor. When the Memory FIFO is enabled, bits(27:12) show the number of entries available in the Memory FIFO. Depending upon the amount of frame buffer memory available, a maximum of 65,536 entries may be stored in the Memory FIFO. The Memory FIFO is empty when bits(27:12)=0xffff. Bits (30:28) of **status** track the number of outstanding SWAPBUFFER commands. When a SWAPBUFFER command is received from the host cpu, bits (30:28) are incremented -- when a SWAPBUFFER command completes, bits (30:28) are decremented. Bit(31) of status is used to monitor the status of the PCI interrupt signal. If SST-1 generates a vertical retrace interrupt (as defined in **pciInterrupt**), bit(31) is set and the PCI interrupt signal line is activated to generate a hardware interrupt. An interrupt is cleared by writing to **status** with "dont-care" data. NOTE THAT BIT(31) IS CURRENTLY NOT IMPLEMENTED IN HARDWARE, AND WILL ALWAYS RETURN 0X0.

5.2 vertex and fvertex Registers

The **vertexAx**, **vertexAy**, **vertexBx**, **vertexBy**, **vertexCx**, **vertexCy**, **fvertexAx**, **fvertexAy**, **fvertexBx**, **fvertexBy**, **fvertexCx**, and **fvertexCy** registers specify the x and y coordinates of a triangle to be rendered. There are three vertices in an SST-1 triangle, with the **AB** and **BC** edges defining the minor edge and the **AC** edge defining the major edge. The diagram below illustrates two typical triangles:



The **fvertex** registers are floating point equivalents of the **vertex** registers. SST-1 automatically converts both the **fvertex** and **vertex** registers into an internal fixed point notation used for rendering.

vertexAx, **vertexAy**, **vertexBx**, **vertexBy**, **vertexCx**, **vertexCy**

Bit	Description
15:0	Vertex coordinate information (fixed point two's complement 12.4 format)

fvertexAx, **fvertexAy**, **fvertexBx**, **fvertexBy**, **fvertexCx**, **fvertexCy**

Bit	Description
31:0	Vertex coordinate information (IEEE 32-bit single-precision floating point format)

5.3 startR, startG, startB, startA, fstartR, fstartG, fstartB, and fstartA Registers

The **startR**, **startG**, **startB**, **startA**, **fstartR**, **fstartG**, **fstartB**, and **fstartA** registers specify the starting color information (red, green, blue, and alpha) of a triangle to be rendered. The **start** registers must contain the color values associated with the **A** vertex of the triangle. The **fstart** registers are floating point equivalents of the **start** registers. SST-1 automatically converts both the **start** and **fstart** registers into an internal fixed point notation used for rendering.

startR, startG, startB, startA

Bit	Description
23:0	Starting Vertex-A Color information (fixed point two's complement 12.12 format)

fstartR, fstartG, fstartB, fstartA

Bit	Description
31:0	Starting Vertex-A Color information (IEEE 32-bit single-precision floating point format)

5.4 startZ and fstartZ registers

The **startZ** and **fstartZ** registers specify the starting Z information of a triangle to be rendered. The **startZ** registers must contain the Z values associated with the **A** vertex of the triangle. The **fstartZ** register is a floating point equivalent of the **startZ** registers. SST-1 automatically converts both the **startZ** and **fstartZ** registers into an internal fixed point notation used for rendering.

startZ

Bit	Description
31:0	Starting Vertex-A Z information (fixed point two's complement 20.12 format)

fstartZ

Bit	Description
31:0	Starting Vertex-A Z information (IEEE 32-bit single-precision floating point format)

5.5 startS, startT, fstartS, and fstartT Registers

The **startS**, **startT**, **fstartS**, and **fstartT** registers specify the starting S/W and T/W texture coordinate information of a triangle to be rendered. The **start** registers must contain the texture coordinates associated with the **A** vertex of the triangle. Note that the S and T coordinates used by SST-1 for rendering must be divided by W prior to being sent to SST-1 (i.e. SST-1 iterates S/W and T/W prior to perspective correction). During rendering, the iterated **S** and **T** coordinates are (optionally) divided by the iterated **W** parameter to perform perspective correction. The **fstart** registers are floating point equivalents of the **start** registers. SST-1 automatically converts both the **start** and **fstart** registers into an internal fixed point notation used for rendering.

startS, startT

Bit	Description
31:0	Starting Vertex-A Texture coordinates (fixed point two's complement 14.18 format)

fstartS, fstartT

Bit	Description

31:0	Starting Vertex-A Texture coordinates (IEEE 32-bit single-precision floating point format)

5.6 startW and fstartW registers

The **startW** and **fstartW** registers specify the starting 1/W information of a triangle to be rendered. The **startW** registers must contain the W values associated with the **A** vertex of the triangle. Note that the **W** value used by SST-1 for rendering is actually the reciprocal of the 3D-geometry-calculated W value (i.e. SST-1 iterates 1/W prior to perspective correction). During rendering, the iterated **S** and **T** coordinates are (optionally) divided by the iterated **W** parameter to perform perspective correction. The **fstartW** register is a floating point equivalent of the **startW** registers. SST-1 automatically converts both the **startW** and **fstartW** registers into an internal fixed point notation used for rendering.

startW

Bit	Description
31:0	Starting Vertex-A W information (fixed point two's complement 2.30 format)

fstartW

Bit	Description
31:0	Starting Vertex-A W information (IEEE 32-bit single-precision floating point format)

5.7 dRdX, dGdX, dBdX, dAdX, fdRdX, fdGdX, fdBdX, and fdAdX Registers

The **dRdX**, **dGdX**, **dBdX**, **dAdX**, **fdRdX**, **fdGdX**, **fdBdX**, and **fdAdX** registers specify the change in the color information (red, green, blue, and alpha) with respect to X of a triangle to be rendered. As a triangle is rendered, the **d?dX** registers are added to the the internal color component registers when the pixel drawn moves from left-to-right, and are subtracted from the internal color component registers when the pixel drawn moves from right-to-left. The **fd?dX** registers are floating point equivalents of the **d?dX** registers. SST-1 automatically converts both the **d?dX** and **fd?dX** registers into an internal fixed point notation used for rendering.

dRdX, dGdX, dBdX, dAdX

Bit	Description
23:0	Change in color with respect to X (fixed point two's complement 12.12 format)

fdRdX, fdGdX, fdBdX, fdAdX

Bit	Description
31:0	Change in color with respect to X (IEEE 32-bit single-precision floating point format)

5.8 dZdX and fdZdX Registers

The **dZdX** and **fdZdX** registers specify the change in Z with respect to X of a triangle to be rendered. As a triangle is rendered, the **dZdX** register is added to the the internal Z register when the pixel drawn moves from left-to-right, and is subtracted from the internal Z register when the pixel drawn moves from right-to-left. The **fdZdX** registers are floating point equivalents of the **dZdX** registers. SST-1 automatically converts both the **dZdX** and **fdZdX** registers into an internal fixed point notation used for rendering.

dZdX

Bit	Description
31:0	Change in Z with respect to X (fixed point two's complement 20.12 format)

fdZdX

Bit	Description
31:0	Change in Z with respect to X (IEEE 32-bit single-precision floating point format)

5.9 dSdX, dTdX, fdSdX, and fdTdX Registers

The **dXdX**, **dTdX**, **fdSdX**, and **fdTdX** registers specify the change in the S/W and T/W texture coordinates with respect to X of a triangle to be rendered. As a triangle is rendered, the **d?dX** registers are added to the the internal S and T registers when the pixel drawn moves from left-to-right, and are subtracted from the internal S/W and T/W registers when the pixel drawn moves from right-to-left. Note that the delta S/W and T/W values used by SST-1 for rendering must be divided by W prior to being sent to SST-1 (i.e. SST-1 uses $\Delta S/W$ and $\Delta T/W$). The **d?dX** registers are floating point equivalents of the **fd?dX** registers. SST-1 automatically converts both the **d?dX** and **fd?dX** registers into an internal fixed point notation used for rendering.

dSdX, dTdX

Bit	Description
31:0	Change in S and T with respect to X (fixed point two's complement 14.18 format)

fdSdX, fdTdX

Bit	Description
31:0	Change in Z with respect to X (IEEE 32-bit single-precision floating point format)

5.10 dWdX and fdWdX Registers

The **dWdX** and **fdWdX** registers specify the change in 1/W with respect to X of a triangle to be rendered. As a triangle is rendered, the **dWdX** register is added to the the internal 1/W register when the pixel drawn moves from left-to-right, and is subtracted from the internal 1/W register when the pixel drawn moves from right-to-left. The **fdWdX** registers are floating point equivalents of the **dWdX** registers. SST-1 automatically converts both the **dWdX** and **fdWdX** registers into an internal fixed point notation used for rendering.

dWdX

Bit	Description
31:0	Change in W with respect to X (fixed point two's complement 2.30 format)

fdWdX

Bit	Description
31:0	Change in W with respect to X (IEEE 32-bit single-precision floating point format)

5.11 dRdY, dGdY, dBdY, dAdY, fdRdY, fdGdY, fdBdY, and fdAdY Registers

The **dRdY**, **dGdY**, **dBdY**, **dAdY**, **fdRdY**, **fdGdY**, **fdBdY**, and **fdAdY** registers specify the change in the color information (red, green, blue, and alpha) with respect to Y of a triangle to be rendered. As a triangle is rendered, the **d?dY** registers are added to the the internal color component registers when the pixel drawn in a positive Y direction, and are subtracted from the internal color component registers when the pixel drawn moves in a negative Y direction. The **fd?dY** registers are floating point equivalents of the **d?dY** registers. SST-1 automatically converts both the **d?dY** and **fd?dY** registers into an internal fixed point notation used for rendering.

dRdY, dGdY, dBdY, dAdY

Bit	Description
23:0	Change in color with respect to Y (fixed point two's complement 12.12 format)

fdRdY, fdGdY, fdBdY, fdAdY

Bit	Description
31:0	Change in color with respect to Y (IEEE 32-bit single-precision floating point format)

5.12 dZdY and fdZdY Registers

The **dZdY** and **fdZdY** registers specify the change in Z with respect to Y of a triangle to be rendered. As a triangle is rendered, the **dZdY** register is added to the the internal Z register when the pixel drawn moves in a positive Y direction, and is subtracted from the internal Z register when the pixel drawn moves in a negative Y direction. The **fdZdY** registers are floating point equivalents of the **dZdY** registers. SST-1 automatically converts both the **dZdY** and **fdZdY** registers into an internal fixed point notation used for rendering.

dZdY

Bit	Description
31:0	Change in Z with respect to Y (fixed point two's complement 20.12 format)

fdZdY

Bit	Description
31:0	Change in Z with respect to Y (IEEE 32-bit single-precision floating point format)

5.13 dSdY, dTdY, fdSdY, and fdTdY Registers

The **dYdY**, **dTdY**, **fdSdY**, and **fdTdY** registers specify the change in the S/W and T/W texture coordinates with respect to Y of a triangle to be rendered. As a triangle is rendered, the **d?dY** registers are added to the the internal S/W and T/W registers when the pixel drawn moves in a positive Y direction, and are subtracted from the internal S/W and T/W registers when the pixel drawn moves in a negative Y direction. Note that the delta S/W and T/W values used by SST-1 for rendering must be divided by W prior to being sent to SST-1 (i.e. SST-1 uses $\Delta S/W$ and $\Delta T/W$). The **d?dY** registers are floating point equivalents of the **fd?dY** registers. SST-1 automatically converts both the **d?dY** and **fd?dY** registers into an internal fixed point notation used for rendering.

dSdY, dTdY

Bit	Description
31:0	Change in S and T with respect to Y (fixed point two's complement 14.18 format)

fdSdY, fdTdY

Bit	Description
31:0	Change in Z with respect to Y (IEEE 32-bit single-precision floating point format)

5.14 dWdY and fdWdY Registers

The **dWdY** and **fdWdY** registers specify the change in 1/W with respect to Y of a triangle to be rendered. As a triangle is rendered, the **dWdY** register is added to the the internal 1/W register when the pixel drawn moves in a positive Y direction, and is subtracted from the internal 1/W register when the pixel drawn moves in a negative Y direction. The **fdWdY** registers are floating point equivalents of the **dWdY** registers. SST-1 automatically converts both the **dWdY** and **fdWdY** registers into an internal fixed point notation used for rendering.

dWdY

Bit	Description
31:0	Change in W with respect to Y (fixed point two's complement 2.30 format)

fdWdY

Bit	Description
31:0	Change in W with respect to Y (IEEE 32-bit single-precision floating point format)

5.15 triangleCMD and ftriangleCMD Registers

The **triangleCMD** and **ftriangleCMD** registers execute the triangle drawing command. Writes to **triangleCMD** or **ftriangleCMD** initiate rendering a triangle defined by the **vertex**, **start**, **d?dX**, and **d?dY** registers. Note that the **vertex**, **start**, **d?dX**, and **d?dY** registers must be setup prior to writing to **triangleCMD** or **ftriangleCMD**. The value stored to **triangleCMD** or **ftriangleCMD** is the area of the triangle being rendered -- this value determines whether a triangle is clockwise or counter-clockwise geometrically. If bit(31)=0, then the triangle is oriented in a counter-clockwise orientation (i.e. positive area). If bit(31)=1, then the triangle is oriented in a clockwise orientation (i.e. negative area). To calculate the area of a triangle, the following steps are performed:

1. The vertices (A, B, and C) are sorted by the Y coordinate in order of increasing Y (i.e. A.y <= B.y <= C.y)
2. The area is calculated as follows:

   AREA = ((dxAB * dyBC) - (dxBC * dyAB)) / 2

   where

   dxAB = A.x - B.x
   dyBC = B.y - C.y
   dxBC = B.x - C.x
   dyAB = A.y - B.y

Note that SST-1 only requires the sign bit of the area to be stored in the **triangleCMD** and **ftriangleCMD** registers -- bits(30:0) written to **triangleCMD** and **ftriangleCMD** are ignored.

5.15.1 Caveats

5.15.1.1 Area

If the sign of the value sent t the triangleCMD and ftriangleCMD registers is not the same as the triangle stored in the vertex registers, FBI will go into an infinite rendering loop.

5.15.1.2 Chip Field Transitions

Under certain circumstances, FBI can lose chip field changes in successive writes. This often occurs when sending data to the chip for triangle rendering. The solution is to write one DWORD of data to the lowest texture address in TMU number 3. This is a legal texture address, but since the SST1 architecture only supports up to three TMUs, there can never be a TMU number 3 (4th TMU). \

5.15.1.3 Write Combining

In situations where out-of-order I/O is possible, a fencing operation of some sort must occur both before and after the triangleCMD register is written.

triangleCMD

Bit	Description
31	Sign of the area of the triangle to be rendered

ftriangleCMD

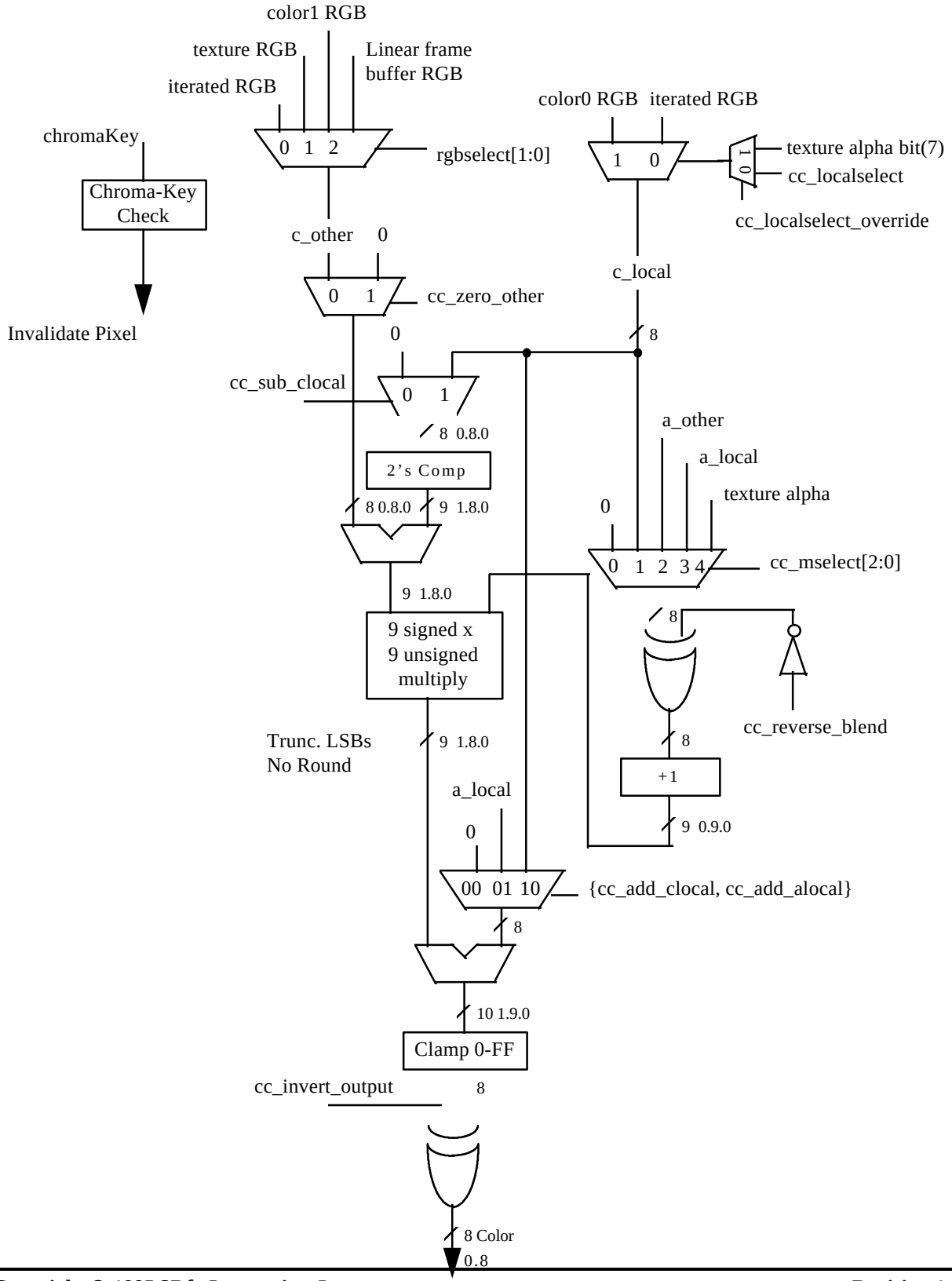
Bit	Description
31	Sign of the area of the triangle to be rendered (IEEE 32-bit single-precision floating point format)

5.16 fbzColorPath Register

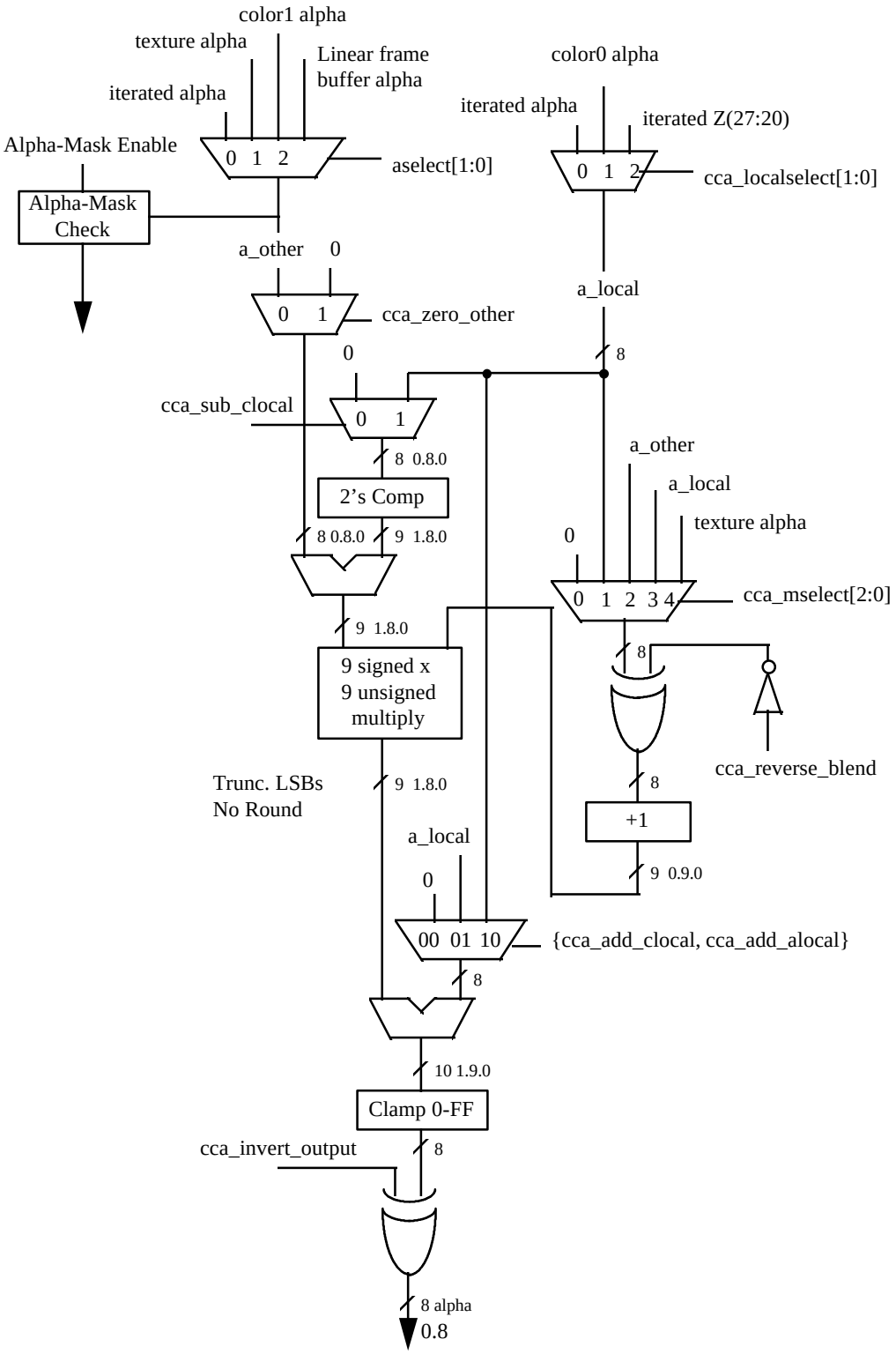
The **fbzColorPath** register controls the color and alpha rendering pixel pipelines. Bits in **fbzColorPath** control color/alpha selection and lighting. Individual bits of **fbzColorPath** are set to enable modulation, addition, etc. for various lighting effects including diffuse and specular highlights.

Bit	Description
1:0	RGB Select (0=Iterated RGB, 1=TREX Color Output, 2=**Color1** RGB, 3=Reserved)
3:2	Alpha Select (0=Iterated A, 1=TREX Alpha Output, 2=**Color1** Alpha, 3=Reserved)
4	Color Combine Unit control (cc_localselect mux control: 0=iterated RGB, 1=**Color0** RGB)
6:5	Alpha Combine Unit control (cca_localselect mux control: 0=iterated alpha, 1=**Color0** alpha, 2=iterated Z, 3=reserved)
7	Color Combine Unit control (cc_localselect_override mux control: 0=cc_localselect, 1=Texture alpha bit(7))
8	Color Combine Unit control (cc_zero_other mux control: 0=c_other, 1=zero)
9	Color Combine Unit control (cc_sub_clocal mux control: 0=zero, 1=c_local)
12:10	Color Combine Unit control (cc_mselect mux control: 0=zero, 1=c_local, 2=a_other, 3=a_local, 4=texture alpha, 5-7=reserved)
13	Color Combine Unit control (cc_reverse_blend control)
14	Color Combine Unit control (cc_add_clocal control)
15	Color Combine Unit control (cc_add_alocal control)
16	Color Combine Unit control (cc_invert_output control)
17	Alpha Combine Unit control (cca_zero_other mux control: 0=a_other, 1=zero)
18	Alpha Combine Unit control (cca_sub_clocal mux control: 0=zero, 1=a_local)
21:19	Alpha Combine Unit control (cca_mselect mux control: 0=zero, 1=a_local, 2=a_other, 3=a_local, 4=texture alpha, 5-7=reserved)
22	Alpha Combine Unit control (cca_reverse_blend control)
23	Alpha Combine Unit control (cca_add_clocal control)
24	Alpha Combine Unit control (cca_add_alocal control)
25	Alpha Combine Unit control (cca_invert_output control)
26	Parameter Adjust (1=adjust parameters for subpixel correction)
27	Enable Texture Mapping (1=enable)

Note that the color channels are controlled separately from the alpha channel. There are two primary color selection units: the Color Combine Unit(CCU) and the Alpha Combine Unit (ACU). Bits(1:0), bit(4), and bits(16:8) of **fbzColorPath** control the Color Combine Unit. The diagram below illustrates the Color Combine Unit controlled by the **fbzColorPath** register:



Bits(3:2), bits(6:5), and bits(25:17) of **fbzColorPath** control the Alpha Combine Unit. The diagram below illustrates the Alpha Combine Unit controlled by the **fbzColorPath** register:


color1 alpha
texture alpha
Linear frame buffer alpha
iterated alpha
Alpha-Mask Enable
0 1 2
aselect[1:0]
Alpha-Mask Check
color0 alpha
iterated alpha
iterated Z(27:20)
0 1 2
cca_localselect[1:0]
a_other
0
0 1
cca_zero_other
a_local
8
0
cca_sub_clocal
0 1
8 0.8.0
2's Comp
8 0.8.0
9 1.8.0
a_other
a_local
texture alpha
0
0 1 2 3 4
cca_mselect[2:0]
9 1.8.0
8
9 signed x 9 unsigned multiply
cca_reverse_blend
Trunc. LSBs
No Round
9 1.8.0
8
+1
a_local
9 0.9.0
0
00 01 10
{cca_add_clocal, cca_add_alocal}
8
10 1.9.0
Clamp 0-FF
cca_invert_output
8
8 alpha
0.8

Bit(26) of **fbzColorPath** enables subpixel correction for all parameters. When enabled, SST-1 will automatically subpixel correct the incoming color, depth, and texture coordinate parameters for triangles not aligned on integer spatial boundaries. Enabling subpixel correction decreases the on-chip triangle setup performance from 7 clocks to 16 clocks, but as the triangle setup engine is separately pipelined from the triangle rasterization engine, little if any performance penalty is seen when subpixel correction is enabled.

Important Note: When subpixel correction is enabled, the correction is performed on the **start** registers as they are passed into the triangle setup unit from the PCI FIFO. As a result, the host must pass down new starting parameter information for each new triangle -- if new starting parameter information is *not* passed down for a new triangle, the starting parameters will be subpixel corrected starting with the **start** registers already subpixel corrected for the last rendered triangle [in effect the parameters will be subpixel corrected twice, resulting in inaccuracies in the starting parameter values].

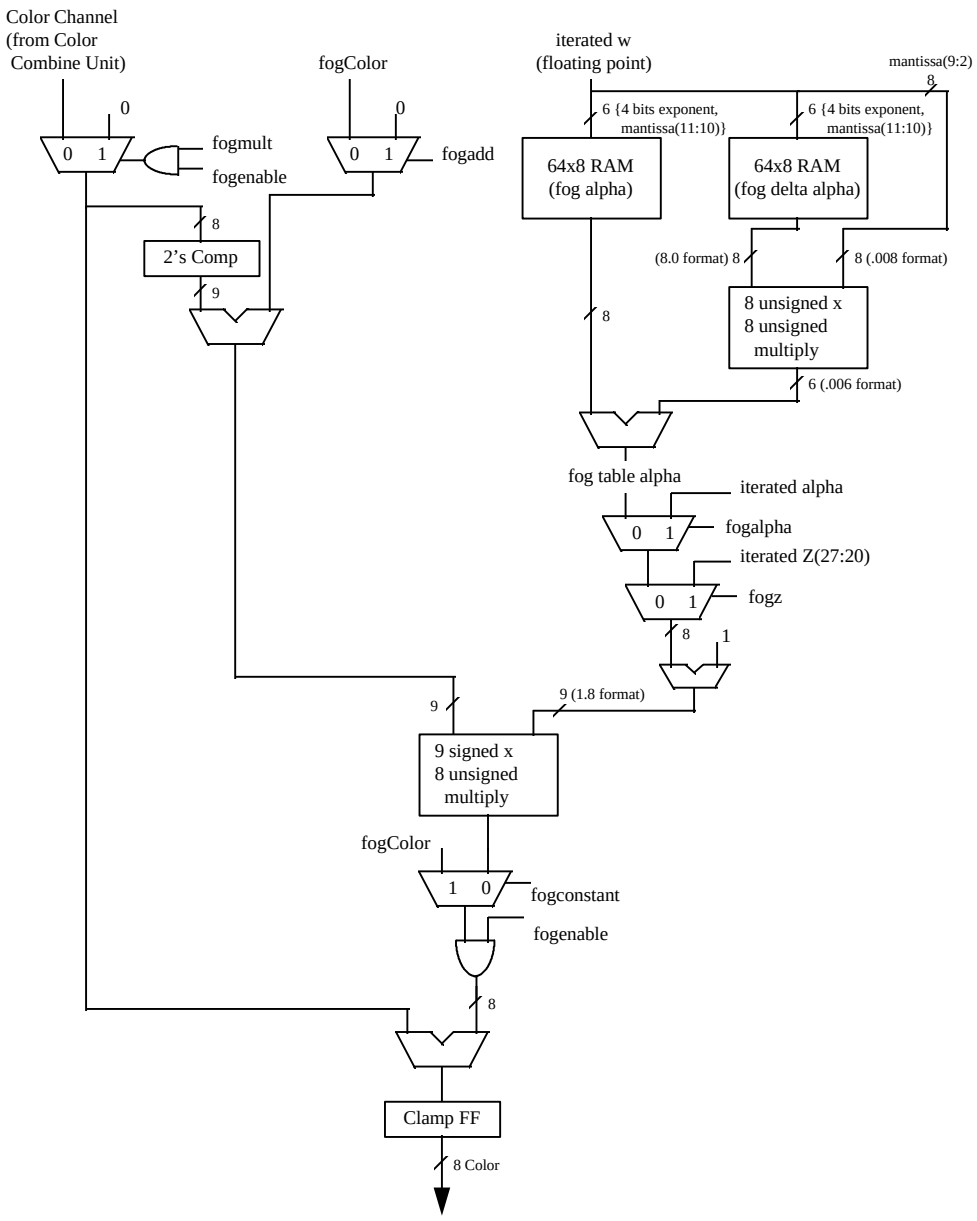
Bit(27) of **fbzColorPath** is used to enable texture mapping. If texture-mapped rendering is desired, then bit(27) of **fbzColorPath** must be set. When bit(27)=1, then data is transfered from TREX to FBI. If texture mapping is not desired (i.e. Gouraud shading, flat shading, etc.), then bit(27) may be cleared and no data is transfered from TREX to FBI.
NOTE: The nopCMD register must be written before any write to fbzColorPath that changes the state of Bit(27).

5.17 fogMode Register

The **fogMode** register controls the fog functionality of SST-1.

Bit	Description
0	Enable fog (1=enable)
1	Fog Unit control (fogadd control: 0=**fogColor**, 1=zero)
2	Fog Unit control (fogmult control: 0=Color Combine Unit RGB, 1=zero)
3	Fog Unit control (fogalpha control: 0=fog table alpha, 1=iterated alpha)
4	Fog Unit control (fogz control: 0=fogalpha mux, 1=iterated z(27:20))
5	Fog Unit control (fogconstant control: 0=fog multiplier output, 1=**fogColor**)

The diagram below shows the fog unit of SST-1:



Bit(0) of **fogMode** is used to enable fog and atmospheric effects. When fog is enabled, the fog color specified in the **fogColor** register is blended with the source pixels as a function of the **fogTable** values and iterated W. SST-1 supports a 64-entry lookup table (**fogTable**) to support atmospheric effects such as fog and haze. When enabled, the MSBs of a normalized floating point representation of (1/W) is used to index into the 64-entry fog table. The ouput of the lookup table is an "alpha" value which represents the level of blending to be performed between the static fog/haze color and the incoming pixel color. 8 lower order bits of the floating point (1/W) are used to blend between multiple entries of the lookup table to reduce fog "banding." The fog lookup table is loaded by the Host CPU, so various fog equations, colors, and effects can be supported.

The following table shows the mathematical equations for the supported values of bits(2:1) of **fogMode** when bits(5:3)=0:

Bit(0) - Enable Fog	Bit(1) - fogadd mux control	Bit(2) - fogmult mux control	Fog Equation
0	ignored	ignored	Cout = Cin
1	0	0	Cout = Afog*Cfog + (1-Afog)*Cin
1	0	1	Cout = Afog*Cfog
1	1	0	Cout = (1-Afog)*Cin
1	1	1	Cout = 0

where:

Cout = Color output from Fog block
Cin = Color input from Color Combine Unit Module
Cfog = **fogColor** register
AFog = alpha value calculated from Fog table

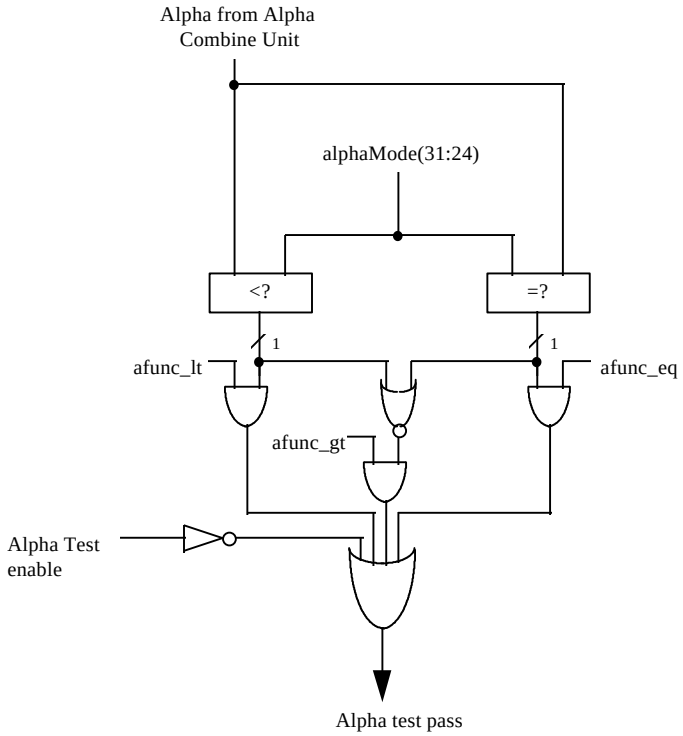
When bit(3) of **fogMode** is set, the integer part of the iterated alpha component is used as the fog alpha instead of the calculated fog alpha value from the fog table. When bit(4) of **fogMode** is set, the upper 8 bits of the iterated Z component are used as the fog alpha instead of the calculated fog alpha value from the fog table. If both bit(3) and bit(4) are set, then bit(4) takes precedence, and the upper 8 bits of the iterated Z component are used for the fog alpha value. Bit(5) of **fogMode** takes precedence over bits(4:3) and enables a constant value(**fogColor**) to be added to incoming source color.

5.18 alphaMode Register

The **alphaMode** register controls the alpha blending and anti-aliasing functionality of SST-1.

Bit	Description
0	Enable alpha function (1=enable)
3:1	Alpha function (see table below)
4	Enable alpha blending (1=enable)
5	Enable anti-aliasing (1=enable)
7:6	reserved
11:8	Source RGB alpha blending factor (see table below)
15:12	Destination RGB alpha blending factor (see table below)
19:16	Source alpha-channel alpha blending factor (see table below)
23:20	Destination alpha-channel alpha blending factor (see table below)
31:24	Alpha reference value

Bits(3:1) specify the alpha function during rendering operations. The alpha function and test pipeline is shown below:



When **alphaMode** bit(0)=1, an alpha comparison is performed between the incoming source alpha and bits(31:24) of **alphaMode**. Section 5.18.1 below further describes the alpha function algorithm.

Bit(4) of **alphaMode** enables alpha blending. When alpha blending is enabled, the blending function is performed to combine the source color with the destination pixel. The blending factors of the source and destinations pixels are individually programmable, as determined by bits(23:8). Note that the RGB and alpha color channels may have different alpha blending factors. Section 5.18.2 below further describes alpha blending.

Bit(5) of **alphaMode** is used to enable anti-aliasing of triangle edges. Anti-aliasing is currently not implemented in SST-1.

5.18.1 Alpha function

When the alpha function is enabled (**alphaMode** bit(0)=1), the following alpha comparison is performed:

AlphaSrc AlphaOP *AlphaRef*

where *AlphaSrc* represents the alpha value of the incoming source pixel, and *AlphaRef* is the value of bits(31:24) of **alphaMode**. A source pixel is written into an RGB buffer if the alpha comparison is true and writing into the RGB buffer is enabled (**fbzMode** bit(9)=1. If the alpha function is enabled and the alpha comparison is false, the **fbiAfuncFail** register is incremented and the pixel is invalidated in the pixel pipeline and no drawing occurs to the color or depth buffers. The supported alpha comparison functions (AlphaOPs) are shown below:

Value	AlphaOP Function
0	never
1	less than
2	equal
3	less than or equal

4	greater than
5	not equal
6	greater than or equal
7	always

5.18.2 Alpha Blending

When alpha blending is enabled (**alphaMode** bit(4)=1), incoming source pixels are blended with destination pixels. The alpha blending function for the RGB color components is as follows:

$$D_{new} \Leftarrow (S \cdot \alpha) + (D_{old} \cdot \beta)$$

where

D_{new}	The new destination pixel being written into the frame buffer
S	The new source pixel being generated
D_{old}	The old (current) destination pixel about to be modified
α	The source pixel alpha blending function.
β	The destination pixel alpha blending function.

The alpha blending function for the alpha components is as follows:

$$A_{new} \Leftarrow (AS \cdot \alpha d) + (A_{old} \cdot \beta d)$$

where

A_{new}	The new destination alpha being written into the alpha buffer
AS	The new source alpha being generated
A_{old}	The old (current) destination alpha about to be modified
αd	The source alpha alpha-blending function.
βd	The destination alpha alpha-blending function.

Note that the source and destination pixels may have different associated alpha blending functions. Also note that RGB color components and the alpha components may have different associated alpha blending functions. The alpha blending factors of the RGB color components are defined in bits(15:8) of **alphaMode**, while the alpha blending factors of the alpha component is specified in bits(23:16) of **alphaMode**. The following table lists the alpha blending functions supported:

Alpha Blending Function	Alpha Blending Function Pneumonic	Alpha Blending Function Description
0x0	AZERO	Zero
0x1	ASRC_ALPHA	Source alpha
0x2	A_COLOR	Color
0x3	ADST_ALPHA	Destination alpha
0x4	AONE	One
0x5	AOMSRC_ALPHA	1 - Source alpha
0x6	AOM_COLOR	1 - Color
0x7	AOMDST_ALPHA	1 - Destination alpha
0x8-0xe		Reserved
0xf (source alpha blending function)	ASATURATE	MIN(Source alpha, 1 - Destination alpha)
0xf (destination alpha blending function)	A_COLORBEFOREFOG	Color before Fog Unit

When the value 0x2 is selected as the destination alpha blending factor, the source pixel color is used as the destination blending factor. When the value 0x2 is selected as the source alpha blending factor, the destination pixel color is used as the source blending factor. Note also that the alpha blending function 0xf is different depending upon whether it is being used as a source or destination alpha blending function. When the value 0xf is selected as the destination alpha blending factor, the source color before the fog unit (“unfogged” color) is used as

the destination blending factor -- this alpha blending function is useful for multi-pass rendering with atmospheric effects. When the value 0xf is selected as the source alpha blending factor, the alpha-saturate anti-aliasing algorithm is selected -- this MIN function performs polygonal anti-aliasing for polygons which are drawn front-to-back.

*** Note that the first silicon spin of SST-1 only supports AZERO and AONE for the alpha blending functions for the alpha channel. All alpha blending functions for the RGB color channels are supported in the first silicon spin.

5.19 fbzMode Register

The **fbzMode** register controls frame buffer and depth buffer rendering functions of the SST-1 processor. Bits in **fbzMode** control clipping, chroma-keying, depth-buffering, dithering, and masking.

Bit	Description
0	Enable clipping rectangle (1=enable)
1	Enable chroma-keying (1=enable)
2	Enable stipple register masking (1=enable)
3	W-Buffer Select (0=Use Z-value for depth buffering, 1=Use W-value for depth buffering)
4	Enable depth-buffering (1=enable)
7:5	Depth-buffer function (see table below)
8	Enable dithering (1=enable)
9	RGB buffer write mask (0=disable writes to RGB buffer)
10	Depth/alpha buffer write mask (0=disable writes to depth/alpha buffer)
11	Dither algorithm (0=4x4 ordered dither, 1=2x2 ordered dither)
12	Enable Stipple pattern masking (1=enable)
13	Enable Alpha-channel mask (1=enable alpha-channel masking)
15:14	Draw buffer (0=Front Buffer, 1=Back Buffer, 2-3=Reserved)
16	Enable depth-biasing (1=enable)
17	Rendering commands Y origin (0=top of screen is origin, 1=bottom of screen is origin)
18	Enable alpha planes (1=enable)
19	Enable alpha-blending dither subtraction (1=enable)
20	Depth buffer source compare select (0=normal operation, 1=**zaColor**[15:0] [**fbzMode** bit(20) is not present in FBI revision 1.0]

Bit(0) of **fbzMode** is used to enable the clipping register. When set, clipping to the rectangle defined by the **clipLeftRight** and **clipBottomTop** registers inclusive is enabled. When clipping is enabled, the bounding clipping rectangle must always be less than or equal to the screen resolution in order to clip to screen coordinates. Also note that if clipping is not enabled, rendering may not occur outside of the screen resolution. Bit(1) of **fbzMode** is used to enable the color compare check (chroma-keying). When enabled, any source pixel matching the color specified in the **chromaKey** register is not written to the RGB buffer. The chroma-key color compare is performed immediately after texture mapping lookup, but before the color combine unit and fog in the pixel datapath.

Bit(2) of **fbzMode** is used to enable stipple register masking. When enabled, bit(12) of **fbzMode** is used to determine the stipple mode -- bit(12)=0 specifies stipple rotate mode, while bit(12)=1 specifies stipple pattern mode.

When stipple register masking is enabled and stipple rotate mode is selected, bit(31) of the **stipple** register is used to mask pixels in the pixel pipeline. For all triangle commands and linear frame buffer writes through the pixel

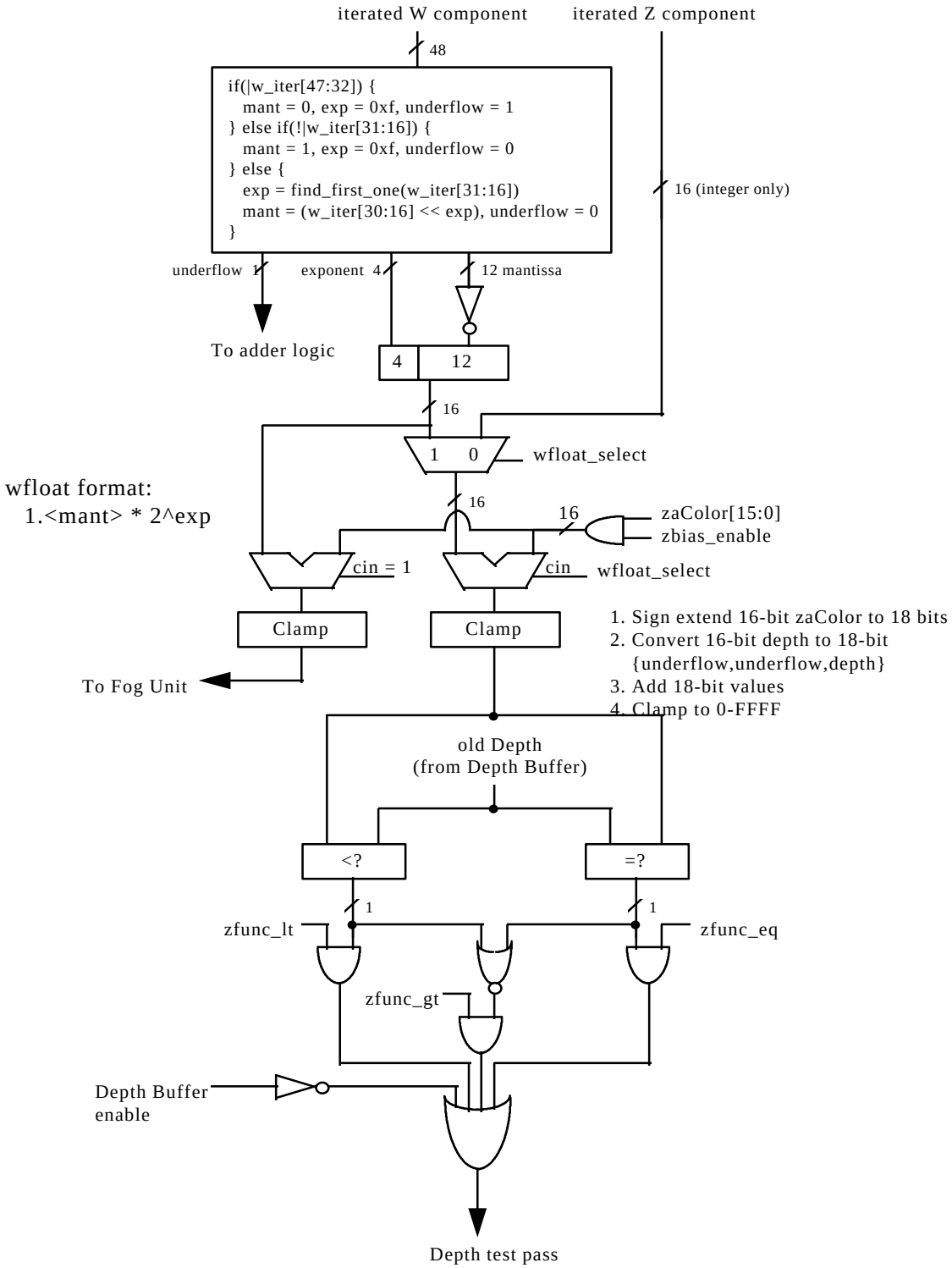
pipeline, pixels are invalidated in the pixel pipeline if **stipple** bit(31)=0 and stipple register masking is enabled in stipple rotate mode. After an individual pixel is processed in the pixel pipeline, the **stipple** register is rotated from right-to-left, with the value of bit(0) filled with the value of bit(31). Note that the **stipple** register is rotated regardless of whether stipple masking is enabled (bit(2) in **fbzMode**) when in stipple rotate mode.

When stipple register masking is enabled and stipple pattern mode is selected, the spatial <x,y> coordinates of a pixel processed in the pixel pipeline are used to lookup a 4x8 monochrone pattern stored in the **stipple** register -- the resultant lookup value is used to mask pixels in the pixel pipeline. For all triangle commands and linear frame buffer writes through the pixel pipeline, a stipple bit is selected from the **stipple** register as follows:

```
switch(pixel_Y[1:0]) {
        case 0: stipple_Y_sel[7:0] = stipple[7:0];
        case 1: stipple_Y_sel[7:0] = stipple[15:8];
        case 2: stipple_Y_sel[7:0] = stipple[23:16];
        case 3: stipple_Y_sel[7:0] = stipple[31:24];
}
switch(pixel_X[2:0] {
        case 0: stipple_mask_bit = stipple_Y_sel[7];
        case 1: stipple_mask_bit = stipple_Y_sel[6];
        case 2: stipple_mask_bit = stipple_Y_sel[5];
        case 3: stipple_mask_bit = stipple_Y_sel[4];
        case 4: stipple_mask_bit = stipple_Y_sel[3];
        case 5: stipple_mask_bit = stipple_Y_sel[2];
        case 6: stipple_mask_bit = stipple_Y_sel[1];
        case 7: stipple_mask_bit = stipple_Y_sel[0];
}
```

If the stipple_mask_bit=0, the pixel is invalidated in the pixel pipeline when stipple register masking is enabled and stipple pattern mode is selected. Note that when stipple pattern mode is selected the **stipple** register is never rotated.

Bits(4:3) specify the depth-buffering function during rendering operations. The depth buffering pipeline is shown below:



Bit(4) of **fbzMode** is used to enable depth-buffering. When depth buffering is enabled, a depth comparison is performed for each source pixel as defined in bits(7:5). When bit(3)=0, the **z** iterator is used for the depth buffer

comparison. When bit(3)=1, the **w** iterator is used for the depth buffer comparison. When bit(3)=1 enabling w-buffering, the inverse of the normalized **w** iterator is used for the depth-buffer comparison. This in effect implements a floating-point w-buffering scheme utilizing a 4-bit exponent and a 12-bit mantissa. The inverted **w** iterator is used so that the same depth buffer comparisons can be used as with a typical z-buffer. Section 5.19.1 below further describes the depth-buffering algorithm.

Bit(8) of **fbzMode** enables 16-bit color dithering. When enabled, native 24-bit source pixels are dithered into 16-bit RGB color values with no performance penalty. When dithering is disabled, native 24-bit source pixels are converted into 16-bit RGB color values by bit truncation. When dithering is enabled, bit(11) of **fbzMode** defines the dithering algorithm -- when bit(11)=0 a 4x4 ordered dither algorithm is used, and when bit(11)=1 a 2x2 ordered dither algorithm is used to convert 24-bit RGB pixels into 16-bit frame buffer colors.

Bit(9) of **fbzMode** enables writes to the RGB buffers. Clearing bit(9) invalidates all writes to the RGB buffers, and thus the RGB buffers remain unmodified for all rendering operations. Bit(9) must be set for normal drawing into the RGB buffers. Similarly, bit(10) enables writes to the depth-buffer. When cleared, writes to the depth-buffer are invalidated, and the depth-buffer state is unmodified for all rendering operations. Bit(10) must be set for normal depth-buffered operation.

Bit(13) of **fbzMode** enables the alpha-channel mask. When enabled, bit(0) of the incoming alpha value is used to mask writes to the color and depth buffers. If alpha channel masking is enabled and bit(0) of the incoming alpha value is 0, then the pixel is invalidated in the pixel pipeline, the **fbiAfuncFail** register is incremented, and no drawing occurs to the color or depth buffers. If alpha channel masking is enabled and bit(0) of the incoming alpha value is 1, then the pixel is drawn normally subject to depth function, alpha blending function, alpha test, and color/depth masking.

Bits(15:14) of **fbzMode** are used to select the RGB draw buffer for graphics drawing. For typical 3D-rendered applications, drawing is only performed into a back buffer. However, some applications may desire to write into the buffer that is being displayed by the monitor (the front buffer). Bit(16) of **fbzMode** is used to enable the Depth Buffer bias. When bit(16)=1, the calculated depth value (irrespective of Z or 1/W type of depth buffering selected) is added to bits(15:0) of **zaColor**. Depth buffer biasing is used to elimate aliasing artifacts when rendering co-planar polygons.

Bit(17) of **fbzMode** is used to define the origin of the Y coordinate for rendering operations (FASTFILL and TRIANGLE commands) and linear frame buffer writes when the pixel pipeline is bypassed for linear frame buffer writes (**lfbMode** bit(8)=0). Note that bit(17) of **fbzMode** does not affect linear frame buffer writes when the pixel pipeline is bypassed for linear frame buffer writes (**lfbMode** bit(8)=0), as in this situation bit(13) of **lfbMode** specifies the Y origin for linear frame buffer writes. Also note that **fbzMode** bit(17) is never used to determine the Y origin for linear frame buffer reads, as **lfbMode** bit(13) always specifies the Y origin for linear frame buffer reads. When cleared, the Y origin (Y=0) for all rendering operations and linear frame buffer writes when the pixel pipeline is enabled is defined to be at the top of the screen. When bit(17) is set, the Y origin is defined to be at the bottom of the screen.

Bit(18) of **fbzMode** is used to enable the destination alpha planes. When set, the auxiliary buffer will be used as destination alpha planes. Note that if bit(18) of **fbzMode** is set that depth buffering cannot be used, and thus bit(4) of **fbzMode** (enable depth buffering) must be set to 0x0.

Bit(19) of **fbzMode** is used to enable dither subtraction on the destination color during alpha blending. When dither subtraction is enabled (**fbzMode** bit(19)=1), the dither matrix used to convert 24-bit color to 16-bit color is subtracted from the destination color before applying the alpha-blending algorithm. Enabling dither subtraction is used to enhance image quality when performing alpha-blending.

Bit(20) of **fbzMode** is used to select the source depth value used for depth buffering. When **fbzMode** bit(20)=0, the source depth value used for the depth buffer comparison is either iterated Z or iterated W (as selected by **fbzMode** bit(3)) and may be biased (as controlled by **fbzMode** bit(16)). When **fbzMode** bit(20)=1, the constant depth value defined by **zaColor**[15:0] is used as the source depth value for the depth buffer comparison. Regardless of the state of **fbzMode** bit(20), the biased iterated Z/W is written into the depth buffer if the depth buffer function passes. Note that **fbzMode** bit(20) is not implemented in FBI revision 1.0.

5.19.1 Depth-buffering function

When the depth-buffering is enabled (**fbzMode** bit(4)=1), the following depth comparison is performed:

DEPTHsrc DepthOP *DEPTHdst*

where *DEPTHsrc* and *DEPTHdst* represent the depth source and destination values respectively. A source pixel is written into an RGB buffer if the depth comparison is true and writing into the RGB buffer is enabled (**fbzMode** bit(9)=1). The source depth value is written into the depth buffer if the depth comparison is true and writing into the depth buffer is enabled (**fbzMode** bit(10)=1). The supported depth comparison functions (DepthOPs) are shown below:

Value	DepthOP Function
0	never
1	less than
2	equal
3	less than or equal
4	greater than
5	not equal
6	greater than or equal
7	always

5.20 lfbMode Register

The **lfbMode** register controls linear frame buffer accesses.

Bit	Description
3:0	Linear frame buffer write format (see table below)
5:4	Linear frame buffer write buffer select (0=front buffer, 1=back buffer, 2-3=reserved).
7:6	Linear frame buffer read buffer select (0=front buffer, 1=back buffer, 2=depth/alpha buffer, 3=reserved).
8	Enable SST-1 pixel pipeline-processed linear frame buffer writes (1=enable)
10:9	Linear frame buffer RGBA lanes (see tables below)
11	16-bit word swap linear frame buffer writes (1=enable)
12	Byte swizzle linear frame buffer writes (1=enable)
13	LFB access Y origin (0=top of screen is origin, 1=bottom of screen is origin)
14	Linear frame buffer write access W select (0=LFB selected, 1=**zacolor**[15:0]).
15	16-bit word Swap linear frame buffer reads (1=enable)
16	Byte swizzle linear frame buffer reads (1=enable)

The following table shows the supported SST-1 linear frame buffer write formats:

Value	Linear Frame Buffer Write Format

	16-bit formats
0	16-bit RGB (5-6-5)
1	16-bit RGB (x-5-5-5)
2	16-bit ARGB (1-5-5-5)
3	Reserved
	32-bit formats
4	24-bit RGB (x-8-8-8)
5	32-bit ARGB (8-8-8-8)
7:6	Reserved
11:8	Reserved
12	16-bit depth, 16-bit RGB (5-6-5)
13	16-bit depth, 16-bit RGB (x-5-5-5)
14	16-bit depth, 16-bit ARGB (1-5-5-5)
15	16-bit depth, 16-bit depth

When accessing the linear frame buffer, the cpu accesses information from the starting linear frame buffer (LFB) address space (see section 4 on SST-1 address space) plus an offset which determines the <x,y> coordinates being accessed. Bits(3:0) of **lfbMode** define the format of linear frame buffer writes. Bits(5:4) of **lfbMode** select which buffer is written when performing linear frame buffer writes (either front or back buffer). Bits(7:6) of **lfbMode** select which buffer is read when performing linear frame buffer reads. Note that for linear frame buffer reads, values from the depth/alpha buffer can be read by setting bits(7:6)=0x2.

When writing to the linear frame buffer, **lfbMode** bit(8)=1 specifies that LFB pixels are processed by the normal SST-1 pixel pipeline -- this implies each pixel written must have an associated depth and alpha value, and is also subject to the fog mode, alpha function, etc. If bit(8)=0, pixels written using LFB access bypass the normal SST-1 pixel pipeline and are written to the specified buffer unconditionally and the values written are unconditionally written into the color/depth buffers except for optional color dithering [depth function, alpha blending, alpha test, and color/depth write masks are all bypassed when bit(8)=0]. If bit(8)=0, then only the buffers that are specified in the particular LFB format are updated. Also note that if **lfbMode** bit(8)=0 that the color and Z mask bits in **fbzMode**(bits 9 and 10) are ignored for LFB writes. For example, if LFB modes 0-2, or 4 are used and bit(8)=0, then only the color buffers are updated for LFB writes (the depth buffer is unaffected by all LFB writes for these modes, regardless of the status of the Z-mask bit **fbzMode** bit 10). However, if LFB modes 12-14 are used and bit(8)=0, then both the color and depth buffers are updated with the LFB write data, irrespective of the color and Z mask bits in **fbzMode**. If LFB mode 15 is used and bit(8)=0, then only the depth buffer is updated for LFB writes (the color buffers are unaffected by all LFB writes in this mode, regardless of the status of the color mask bits in **fbzMode**).

If **lfbMode** bit(8)=0 and a LFB write format is selected which contains an alpha component (formats 2, 5, and 14) and the alpha buffer is enabled, then the alpha component is written into the alpha buffer. Conversely, if the alpha buffer is not enabled, then the alpha component of LFB writes using formats 2, 5, and 14 when bit(8)=0 are ignored. Note that anytime LFB formats 2, 5, and 14 are used when bit(8)=0 that blending and/or chroma-keying using the alpha component is not performed since the pixel-pipeline is bypassed when bit(8)=0.

If **lfbMode** bit(8)=0 and LFB write format 14 is used, the component that is ignored is determined by whether the alpha buffer is enabled -- If the alpha buffer is enabled and LFB write format 14 is used with bit(8)=0, then the depth component is ignored for all LFB writes. Conversely, if the alpha buffer is disabled and LFB write format is used with bit(8)=0, then the alpha component is ignored for all LFB writes.

If **lfbMode** bit(8)=1 and a LFB write access format does not include depth or alpha information (formats 0-5), then the appropriate depth and/or alpha information for each pixel written is taken from the **zaColor** register. Note that if bit(8)=1 that the LFB write pixels are processed by the normal SST-1 pixel pipeline and thus are subject to the per-pixel operations including clipping, dithering, alpha-blending, alpha-testing, depth-testing, chroma-keying, fogging, and color/depth write masking.

Bits(10:9) of **lfbMode** specify the RGB channel format (color lanes) for linear frame buffer writes. The table below shows the SST-1 supported RGB lanes:

Value	RGB Channel Format
0	ARGB
1	ABGR
2	RGBA
3	BGRA

Bit(11) of **lfbMode** defines the format of 2 16-bit data types passed with a single 32-bit writes. For linear frame buffer formats 0-2, two 16-bit data transfers can be packed into one 32-bit write -- bit(11) defines which 16-bit shorts correspond to which pixels on screen. The table below shows the pixel packing for packed 32-bit linear frame buffer formats 0-2:

lfbMode bit(11)	Screen Pixel Packing
0	Right Pixel(host data 31:16), Left Pixel(host data 15:0)
1	Left Pixel(host data 31:16), Right Pixel(host data 15:0)

For linear frame buffer formats 12-14, bit(11) of **lfbMode** defines the bit locations of the 2 16-bit data types passed. The table below shows the data packing for 32-bit linear frame buffer formats 12-14:

lfbMode bit(11)	Screen Pixel Packing
0	Z value(host data 31:16), RGB value(host data 15:0)
1	RGB value(host data 31:16), Z value(host data 15:0)

For linear frame buffer format 15, bit(11) of **lfbMode** defines the bit locations of the 2 16-bit depth values passed. The table below shows the data packing for 32-bit linear frame buffer format 15:

lfbMode bit(11)	Screen Pixel Packing
0	Z Right Pixel(host data 31:16), Z Left Pixel(host data 15:0)
1	Z left Pixel(host data 31:16), Z Right Pixel(host data 15:0)

Note that bit(11) of **lfbMode** is ignored for linear frame buffer writes using formats 4 or 5.

Bit(12) of **lfbMode** is used to enable byte swizzling. When byte swizzling is enabled, the 4-bytes within a 32-bit word are swizzled to correct for endian differences between SST-1 and the host CPU. For little endian CPUs (e.g. Intel x86 processors) byte swizzling should not be enabled, however big endian CPUs (e.g. PowerPC processors) should enable byte swizzling. For linear frame buffer writes, the bytes within a word are swizzled prior to being modified by the other control bits of **lfbMode**. When byte swizzling is enabled, bits(31:24) are swapped with bits(7:0), and bits(23:16) are swapped with bits(15:8). Note the status of bit(12) of **lfbMode** has no affect on linear frame buffer reads.

Very Important Note: The order of swapping and swizzling operations for LFB writes is as follows: byte swizzling is performed first on all incoming LFB data, as defined by **lfbMode** bit(12) and irrespective of the LFB data format. After byte swizzling, 16-bit word swapping is performed as defined by **lfbMode** bit(11). Note that 16-bit word swapping is never performed on LFB data when data formats 4 and 5 are used. Also note that 16-bit word swapping is performed on the LFB data that was previously optionally swapped. Finally, after both swizzling and 16-bit word swapping are performed, the individual color channels are selected as defined in **lfbMode** bits(10:9). Note that the color channels are selected on the LFB data that was previously swizzled and/or swapped

Bit(13) of **lfbMode** is used to define the origin of the Y coordinate for all linear frame buffer reads and linear frame buffer writes when the pixel pipeline is bypassed (**lfbMode** bit(8)=0). Note that bit(13) of **lfbMode** does not affect rendering operations (FASTFILL and TRIANGLE commands) -- bit(17) of **fbzMode** defines the origin of the Y coordinate for rendering operations. Note also that if the pixel pipeline is enabled for linear frame buffer writes (**lfbMode** bit(8)=1), then **fbzMode** bit(17) is used to determine the location of the Y origin. For linear frame buffer reads, however, **lfbMode** bit(13) is always used to determine the Y origin, regardless of the setting of **lfbMode** bit(8). When cleared, the Y origin (Y=0) for all linear frame buffer accesses is defined to be at the top of the screen. When bit(13) is set, the Y origin for all linear frame buffer accesses is defined to be at the bottom of the screen.

Bit(14) of **lfbMode** is used to select the W component used for LFB writes processed through the pixel pipeline. If bit(14)=0, then the MSBs of the fractional component of the 48-bit W value passed to the pixel pipeline for LFB writes through the pixel pipeline is the 16-bit Z value associated with the LFB write. [Note that the 16-bit Z value associated with the LFB write is dependent on the LFB format, and is either passed down pixel-by-pixel from the CPU, or is set to the constant **zaColor**(15:0)]. If bit(14)=1, then the MSBs of the fractional component of the 48-bit W value passed to the pixel pipeline for LFB writes is **zacolor**(15:0). Regardless of the setting of bit(14), when LFB writes go through the pixel pipeline, all other bits except the 16 MSBs of the fractional component of the W value are set to 0x0. Note that bit(14) is ignored if LFB writes bypass the pixel pipeline.

5.20.1 Linear Frame Buffer Writes

Linear frame buffer writes -- format 0:
When writing to the linear frame buffer with 16-bit format 0 (RGB 5-6-5), the RGB channel format specifies the RGB ordering within a 16-bit word. If the SST-1 pixel pipeline is enabled for LFB accesses (**lfbMode** bit(8)=1), then alpha and depth information for LFB format 0 is taken from the **zaColor** register. The following table shows the color channels for 16-bit linear frame buffer access format 0:

RGB Channel Format Value	16-bit Linear frame buffer access bits	RGB Channel
0	15:0	Red (15:11), Green(10:5), Blue(4:0)
1	15:0	Blue (15:11), Green(10:5), Red(4:0)
2	15:0	Red (15:11), Green(10:5), Blue(4:0)
3	15:0	Blue (15:11), Green(10:5), Red(4:0)

Linear frame buffer writes -- format 1:
When writing to the linear frame buffer with 16-bit format 1 (RGB 5-5-5), the RGB channel format specifies the RGB ordering within a 16-bit word. If the SST-1 pixel pipeline is enabled for LFB accesses (**lfbMode** bit(8)=1), then alpha and depth information for LFB format 1 is taken from the **zaColor** register. The following table shows the color channels for 16-bit linear frame buffer access format 1:

RGB Channel Format Value	16-bit Linear frame buffer access bits	RGB Channel
0	15:0	Ignored(15), Red (14:10), Green(9:5), Blue(4:0)
1	15:0	Ignored(15), Blue (14:10), Green(9:5), Red(4:0)
2	15:0	Red (15:11), Green(10:6), Blue(5:1), Ignored(0)
3	15:0	Blue (15:11), Green(10:6), Red(5:1), Ignored(0)

Linear frame buffer writes -- format 2:

When writing to the linear frame buffer with 16-bit format 2 (ARGB 1-5-5-5), the RGB channel format specifies the RGB ordering within a 16-bit word. If the SST-1 pixel pipeline is enabled for LFB accesses (**lfbMode** bit(8)=1), then depth information for LFB format 2 is taken from the **zaColor** register. Note that the 1-bit alpha value passed when using LFB format 2 is bit-replicated to yield the 8-bit alpha used in the pixel pipeline. The following table shows the color channels for 16-bit linear frame buffer access format 2:

RGB Channel Format Value	16-bit Linear frame buffer access bits	RGB Channel
0	15:0	Alpha(15), Red (14:10), Green(9:5), Blue(4:0)
1	15:0	Alpha(15), Blue (14:10), Green(9:5), Red(4:0)
2	15:0	Red (15:11), Green(10:6), Blue(5:1), Alpha(0)
3	15:0	Blue (15:11), Green(10:6), Red(5:1), Alpha(0)

Linear frame buffer writes -- format 3:

Linear frame buffer format 3 is an unsupported format.

Linear frame buffer writes -- format 4:

When writing to the linear frame buffer with 24-bit format 4 (RGB x-8-8-8), the RGB channel format specifies the RGB ordering within a 24-bit word. Note that the alpha/A channel is ignored for 24-bit access format 4. Also note that while only 24-bits of data is transfered for format 4, all data access must be 32-bit aligned -- packed 24-bit writes are not supported by SST-1. If the SST-1 pixel pipeline is enabled for LFB accesses (**lfbMode** bit(8)=1), then alpha and depth information for LFB format 4 is taken from the **zaColor** register. The following table shows the color channels for 24-bit linear frame buffer access format 4:

RGB Channel Format Value	24-bit Linear frame buffer access bits (aligned to 32-bits)	RGB Channel
0	31:0	Ignored(31:24), Red (23:16), Green(15:8), Blue(7:0)
1	31:0	Ignored(31:24), Blue(23:16), Green(15:8), Red(7:0)
2	31:0	Red(31:24), Green(23:16), Blue(15:8), Ignored(7:0)
3	31:0	Blue(31:24), Green(23:16), Red(15:8), Ignored(7:0)

Linear frame buffer writes -- format 5:

When writing to the linear frame buffer with 32-bit format 5 (ARGB 8-8-8-8), the RGB channel format specifies the ARGB ordering within a 32-bit word. If the SST-1 pixel pipeline is enabled for LFB accesses (**lfbMode** bit(8)=1), then depth information for LFB format 5 is taken from the **zaColor** register. The following table shows the color channels for 32-bit linear frame buffer access format 5.

RGB Channel Format Value	24-bit Linear frame buffer access bits (aligned to 32-bits)	RGB Channel
0	31:0	Alpha(31:24), Red (23:16), Green(15:8), Blue(7:0)
1	31:0	Alpha(31:24), Blue(23:16), Green(15:8), Red(7:0)
2	31:0	Red(31:24), Green(23:16), Blue(15:8), Alpha(7:0)
3	31:0	Blue(31:24), Green(23:16), Red(15:8), Alpha(7:0)

Linear frame buffer writes -- formats 6-11:
Linear frame buffer formats 6-11 are unsupported formats.

Linear frame buffer writes -- format 12:
When writing to the linear frame buffer with 32-bit format 12 (Depth 16, RGB 5-6-5), the RGB channel format specifies the RGB ordering within the 32-bit word. If the SST-1 pixel pipeline is enabled for LFB accesses (**lfbMode** bit(8)=1), then alpha information for LFB format 12 is taken from the **zaColor** register. Note that the format of the depth value passed when using LFB format 12 must precisely match the format of the type of depth buffering being used (either 16-bit integer Z or 16-bit floating point 1/W). The following table shows the 16-bit color channels within the 32-bit linear frame buffer access format 12:

RGB Channel Format Value	16-bit Linear frame buffer access bits	RGB Channel
0	15:0	Red (15:11), Green(10:5), Blue(4:0)
1	15:0	Blue (15:11), Green(10:5), Red(4:0)
2	15:0	Red (15:11), Green(10:5), Blue(4:0)
3	15:0	Blue (15:11), Green(10:5), Red(4:0)

Linear frame buffer writes -- format 13:
When writing to the linear frame buffer with 32-bit format 13 (Depth 16, RGB x-5-5-5), the RGB channel format specifies the RGB ordering within the 32-bit word. If the SST-1 pixel pipeline is enabled for LFB accesses (**lfbMode** bit(8)=1), then alpha information for LFB format 13 is taken from the **zaColor** register. Note that the format of the depth value passed when using LFB format 13 must precisely match the format of the type of depth buffering being used (either 16-bit integer Z or 16-bit floating point 1/W). The following table shows the 16-bit color channels within the 32-bit linear frame buffer access format 13:

RGB Channel Format Value	16-bit Linear frame buffer access bits	RGB Channel
0	15:0	Ignored(15), Red (14:10), Green(9:5), Blue(4:0)
1	15:0	Ignored(15), Blue (14:10), Green(9:5), Red(4:0)
2	15:0	Red (15:11), Green(10:6), Blue(5:1), Ignored(0)
3	15:0	Blue (15:11), Green(10:6), Red(5:1), Ignored(0)

Linear frame buffer writes -- format 14:
When writing to the linear frame buffer with 32-bit format 14 (Depth 16, ARGB 1-5-5-5), the RGB channel format specifies the RGB ordering within the 32-bit word. Note that the format of the depth value passed when using LFB format 14 must precisely match the format of the type of depth buffering being used (either 16-bit integer Z or 16-bit floating point 1/W). Also note that the 1-bit alpha value passed when using LFB format 14 is bit-replicated to yield the 8-bit alpha used in the pixel pipeline. The following table shows the 16-bit color channels within the 32-bit linear frame buffer access format 14:

RGB Channel Format Value	16-bit Linear frame buffer access bits	RGB Channel
0	15:0	Alpha(15), Red (14:10), Green(9:5), Blue(4:0)
1	15:0	Alpha(15), Blue (14:10), Green(9:5), Red(4:0)
2	15:0	Red (15:11), Green(10:6), Blue(5:1), Alpha(0)
3	15:0	Blue (15:11), Green(10:6), Red(5:1), Alpha(0)

Linear frame buffer writes -- format 15:

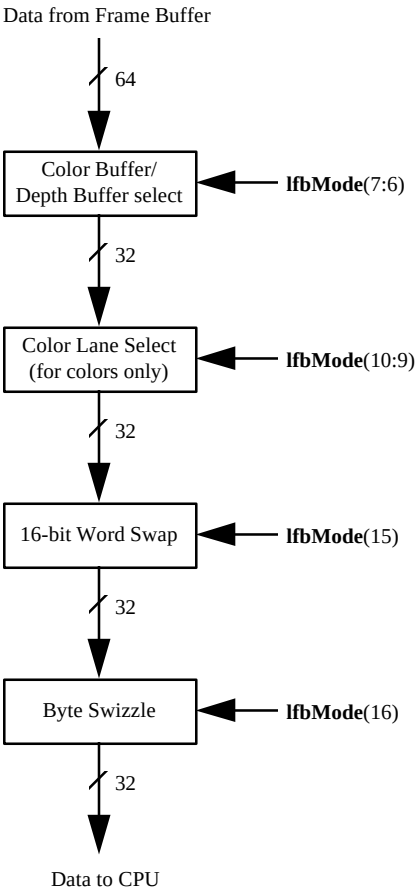
When writing to the linear frame buffer with 32-bit format 15 (Depth 16, Depth 16), the format of the depth values passed must precisely match the format of the type of depth buffering being used (either 16-bit integer Z or 16-bit floating point 1/W). If the SST-1 pixel pipeline is enabled for LFB accesses (**lfbMode** bit(8)=1), then RGB color information is taken from the **color1** register, and alpha information for LFB format 15 is taken from the **zaColor** register.

5.20.2 Linear Frame Buffer Reads

When reading from the linear frame buffer, all data returned is in 16/16 format, with two 16-bit pixels returned for every 32-bit doubleword read. The RGB channel format of the 16-bit pixels read is defined by the rgb channel format field of **lfbMode** bits(12:9). The alpha/depth buffer can also be read by selecting **lfbMode** bits(7:6)=0x2. The mapping of the screen space pixels to the two 16-bit words within a 32-bit read are defined by **lfbMode** bit(15) as shown in the following table:

lfbMode bit(15)	Screen Pixel Packing
0	Right Pixel(host data 31:16), Left Pixel(host data 15:0)
1	Left Pixel(host data 31:16), Right Pixel(host data 15:0)

The value of bit(16) of **lfbMode** also affects the byte positioning of linear frame buffer reads -- if bit(16)=1, then the LFB read data output from the 16-bit word swap logic is byte-swizzled. Note that byte swizzling (if enabled) is performed after 16-bit word swapping (if enabled) for linear frame buffer reads. Also note that byte swizzling and/or word swapping are performed on reads from the depth/alpha buffer (selected when **lfbMode** bits(7:6)=0x2) if either or both are enabled. The value of bit(13) of **lfbMode** selects the position of the Y origin for all linear frame buffer reads. The order of frame buffer read data formatting is illustrated below:



See section 8 for more information on linear frame buffer accesses.

5.21 clipLeftRight and clipLowYHighY Registers

The **clip** registers specify a rectangle within which all drawing operations are confined. If a pixel is to be drawn outside the clip rectangle, it will not be written into the RGB or depth buffers. Note that the specified clipping rectangle defines a valid drawing area in both the RGB and depth/alpha buffers. The values in the clipping registers are given in pixel units, and the valid drawing rectangle is inclusive of the **clipleft** and **clipLowY** register values, but exclusive of the **clipRight** and **clipHighY** register values. **clipLowY** must be less than **clipHighY**, and **clipLeft** must be less than **clipRight**. The **clip** registers can be enabled by setting bit(0) in the **fbzMode** register. When clipping is enabled, the bounding clipping rectangle must always be less than or equal to the screen resolution in order to clip to screen coordinates. Also note that if clipping is not enabled, rendering must not be specified to occur outside of the screen resolution.

Important Note: The **clipLowYHighY** register is defined such that y=0 always resides at the top of the monitor screen. Changing the value of the Y origin bits (**fbzMode** bit(17) or **lfbMode** bit(13)) has no affect on the **clipLowYHighY** register orientation. As a result, if the Y origin is defined to be at the bottom of the screen (by setting one of the Y origin bits), care must be taken in setting the **clipLowYHighY** register to ensure proper functionality. In the case where the Y origin is defined to be at the bottom of the screen, the value of **clipLowYHighY** is usually set as the number of scan lines in the monitor resolution minus the desired Y clipping values.

The **clip** registers are also used to define a rectangular region to be drawn during a FASTFILL command. Note that when **clipLowYHighY** is used to specify a rectangular region for the FASTFILL command, the orientation of the Y origin (top or bottom of the screen) is defined by the status of **fbzMode** bit(17). See section 7 and the **fastfillCMD** register description for more information on the FASTFILL command.

clipLeftRight Register

Bit	Description
9:0	Unsigned integer specifying right clipping rectangle edge
15:10	reserved
25:16	Unsigned integer specifying left clipping rectangle edge
31:26	reserved

clipLowYHighY Register

Bit	Description
9:0	Unsigned integer specifying high Y clipping rectangle edge
15:10	reserved
25:16	Unsigned integer specifying low Y clipping rectangle edge
31:26	reserved

5.22 nopCMD Register

Writing any data to the **nopCMD** register executes the NOP command. Executing a NOP command flushes the graphics pipeline. The lsb of the data value written to **nopCMD** is used to optionally clear the **fbiPixelsIn**, **fbiChromaFail**, **fbiZfuncFail**, **fbiAfuncFail**, and **fbiPixelsOut** registers. Writing a '1' to the lsb of **nopCMD** will clear the aforementioned registers. Writing a '0' to the lsb of nopCMD will not modify the values of the aforementioned registers.

Bit	Description
0	Clear **fbiPixelsIn**, **fbiChromaFail**, **fbiZfuncFail**, **fbiAfuncFail**, and **fbiPixelsOut** registers (1=clear registers)

5.23 fastfillCMD Register

Writing any data to the **fastfill** register executes the FASTFILL command. The FASTFILL command is used to clear the RGB and depth buffers as quickly as possible. Prior to executing the FASTFILL command, the **clipLeftRight** and **clipLowYHighY** are loaded with a rectangular area which is the desired area to be cleared. Note that **clip** registers define a rectangular area which is inclusive of the **clipLeft** and **clipLowY** register values, but exclusive of the **clipRight** and **clipHighY** register values. The **fastfillCMD** register is then written to initiate the FASTFILL command after the **clip** registers have been loaded. FASTFILL will optionally clear the color buffers with the RGB color specified in the **color1** register, and also optionally clears the depth buffer with the depth value taken from the **zaColor** register. Note that since **color1** is a 24-bit value, either dithering or bit truncation must be used to translate the 24-bit value into the native 16-bit frame buffer -- dithering may be employed optionally as defined by bit(8) of **fbzMode**. Disabling clearing of the color or depth buffers is accomplished by modifying the rgb/depth mask bits(10:9) in **fbzMode**. This allows individual or combined clearing of the RGB and depth buffers.

5.24 swapbufferCMD Register

Writing to the **swapbufferCMD** register executes the SWAPBUFFER command. If the data written to **swapbufferCMD** bit(0)=0, then the frame buffer swapping is not synchronized with vertical retrace. If frame buffer swapping is not synchronized with vertical retrace, then visible frame "tearing" may occur. If **swapbufferCMD** bit(0)=1 then the frame buffer swapping is synchronized with vertical retrace. Synchronizing frame buffer swapping with vertical retrace eliminates the aforementioned frame "tearing." When a **swapbufferCMD** is received in the front-end PCI host FIFO, the swap buffers pending field in the status register is incremented. Conversely, when an actual frame buffer swapping occurs, the swap buffers pending field in the **status** register (bits(30:28)) is decremented. The swap buffers pending field allows software to determine how many SWAPBUFFER commands are present in the SST-1 FIFOs. Bits(8:1) of **swapbufferCMD** are used to specify the number of vertical retraces to wait before swapping the color buffers. An internal counter is incremented whenever a vertical retrace occurs, and the color buffers are not swapped until the internal vertical retrace counter is greater than the value of **swapbufferCMD** bits(8:1) -- After a swap occurs, the internal vertical retrace counter is cleared. Specifying values other than zero for bits(8:1) are used to maintain constant frame rate. Note that if vertical retrace synchronization is disabled for swapping buffers (**swapbufferCMD**(0)=0), then the swap buffer interval field is ignored. TO BE COMPLETED: SWAPBUFFER command and triple buffering. Note that syncing to vertical retrace must be enabled and the swapbuffer interval must be 0x0 when using triple buffering.

Bit	Description
0	Synchronize frame buffer swapping to vertical retrace (1=enable)
8:1	Swap buffer interval

5.25 fogColor Register

The **fogColor** register is used to specify the fog color for fogging operations. Fog is enabled by setting bit(0) in **fogMode**. See the **fogMode** and **fogTable** register descriptions for more information fog.

Bit	Description
7:0	Fog Color Blue
15:8	Fog Color Green
23:16	Fog Color Red
31:24	reserved

5.26 zaColor Register

The **zaColor** register is used to specify constant alpha and depth values for linear frame buffer writes, FASTFILL commands, and co-planar polygon rendering support. For certain linear frame buffer access formats, the alpha and depth values associated with a pixel written are the values specified in **zaColor**. See the **lfbMode** register description for more information. When executing the FASTFILL command, the constant 16-bit depth value written into the depth buffer is taken from bits(15:0) of **zaColor**. When **fbzMode** bit(16)=1 enabling depth-biasing, the constant depth value required is taken from **zaColor** bits(15:0).

Bit	Description
15:0	Constant Depth
23:16	reserved
31:24	Constant Alpha

5.27 chromaKey Register

The **chromaKey** register specifies a color which is compared with all pixels to be written into the RGB buffer. If a color match is detected between an outgoing pixel and the **chromaKey** register, and chroma-keying is enabled (bit(1)=1 in the **fbzMode** register), then the pixel is not written into the frame buffer. An outgoing pixel will still be written into the RGB buffer if chroma-keying is disabled or the **chromaKey** color does not equal the outgoing pixel color. Note that the alpha color component of an outgoing pixel is ignored in the chroma-key color match circuitry. The chroma-key comparison is performed immediately after texture lookup, but before lighting, fog, or alpha blending. See the description of the **fbzColorPath** register for further information on the location of the chroma-key comparison circuitry. The format of **chromaKey** is a 24-bit RGB color.

Bit	Description
7:0	Chroma-key Blue
15:8	Chroma-key Green
23:16	Chroma-key Red
31:24	reserved

5.28 stipple Register

The **stipple** register specifies a mask which is used to enable individual pixel writes to the RGB and depth buffers. See the stipple functionality description in the **fbzMode** register description for more information.

Bit	Description
31:0	stipple value

5.29 color0 Register

The **color0** register specifies constant color values which are used for certain rendering functions. In particular, bits(23:0) of **color0** are optionally used as the c_local input in the color combine unit. In addition, bits(31:24) of **color0** are optionally used as the c_local input in the alpha combine unit. See the **fbzColorPath** register description for more information.

Bit	Description
7:0	Constant Color Blue
15:8	Constant Color Green
23:16	Constant Color Red
31:24	Constant Color Alpha

5.30 color1 Register

The **color1** register specifies constant color values which are used for certain rendering functions. In particular, bits(23:0) of **color1** are optionally used as the c_other input in the color combine unit selected by bits(1:0) of **fbzColorPath**. The alpha component of **color1**(bits(31:24)) are optionally used as the a_other input in the alpha combine unit selected by bits(3:2) of **fbzColorPath**. The **color1** register bits(23:0) are also used by the FASTFILL command as the constant color for screen clears. Also, for linear frame buffer write format 15(16-bit depth, 16-bit depth), the color for the pixel pipeline is taken from **color1** if the pixel pipeline is enabled for linear frame buffer writes (**lfbMode** bit(8)=1).

Bit	Description
7:0	Constant Color Blue

15:8	Constant Color Green
23:16	Constant Color Red
31:24	Constant Color Alpha

5.31 fbiPixelsIn Register

The **fbiPixelsIn** register is a 24-bit counter which is incremented for each pixel processed by the SST-1 triangle walking engine. **fbiPixelsIn** is incremented irrespective if the triangle pixel is actually drawn or not as a result of the depth test, alpha test, etc. **fbiPixelsIn** is used primarily for statistical information, and in essence allows software to count the number of pixels in a screen-space triangle. **fbiPixelsIn** is reset to 0x0 on power-up reset, and is reset when a '1' if written to the lsb of **nopCMD**.

Bit	Description
23:0	Pixel Counter (number of pixels processed by SST-1 triangle engine)

5.32 fbiChromaFail Register

The **fbiChromaFail** register is a 24-bit counter which is incremented each time an incoming source pixel (either from the triangle engine or linear frame buffer writes through the pixel pipeline) is invalidated in the pixel pipeline because of the chroma-key color match test. If an incoming source pixel color matches the **chomaKey** register, **fbiChromaFail** is incremented. **fbiChromaFail** is reset to 0x0 on power-up reset, and is reset when a '1' if written to the lsb of **nopCMD**.

Bit	Description
23:0	Pixel Counter (number of pixels failed chroma-key test)

5.33 fbiZfuncFail Register

The **fbiZfuncFail** register is a 24-bit counter which is incremented each time an incoming source pixel (either from the triangle engine or linear frame buffer writes through the pixel pipeline) is invalidated in the pixel pipeline because of a failure in the Z test. The Z test is defined and enabled in the **fbzMode** register. **fbiZfuncFail** is reset to 0x0 on power-up reset, and is reset when a '1' if written to the lsb of **nopCMD**.

Bit	Description
23:0	Pixel Counter (number of pixels failed Z test)

5.34 fbiAfuncFail Register

The **fbiAfuncFail** register is a 24-bit counter which is incremented each time an incoming source pixel (either from the triangle engine or linear frame buffer writes through the pixel pipeline) is invalidated in the pixel pipeline because of a failure in the alpha test. The alpha test is defined and enabled in the **alphaMode** register. The **fbiAfuncFail** register is also incremented if an incoming source pixel is invalidated in the pixel pipeline as a result of the alpha masking test (bit(13) in **fbzMode**). **fbiAfuncFail** is reset to 0x0 on power-up reset, and is reset when a '1' if written to the lsb of **nopCMD**.

Bit	Description
23:0	Pixel Counter (number of pixels failed Alpha test)

5.35 fbiPixelsOut Register

The **fbiPixelsOut** register is a 24-bit counter which is incremented each time a pixel is written into a color buffer during rendering operations (rendering operations include triangle commands, linear frame buffer writes, and the FASTFILL command). Pixels tracked by **fbiPixelsOut** are therefore subject to the chroma-test, Z test, Alpha test, etc. that are part of the regular SST-1 pixel pipeline. **fbiPixelsOut** is used to count the number of pixels actually drawn (as opposed to the number of pixels processed counted by **fbiPixelsIn**). Note that the RGB mask (**fbzMode** bit(9) is ignored when determining **fbiPixelsOut**. **fbiPixelsOut** is reset to 0x0 on power-up reset, and is reset when a '1' if written to the lsb of **nopCMD**.

Bit	Description
23:0	Pixel Counter (number of pixels drawn to color buffer)

5.36 fogTable Register

The **fogTable** register is used to implement fog functions in SST-1. The **fogTable** register is a 64-entry lookup table consisting of 8-bit fog blending factors and 8-bit Δfog blending values. The Δfog blending values are the difference between successive fog blending factors in **fogTable** and are used to blend between **fogTable** entries. Note that the Δfog blending factors are stored in 6.2 format, while the fog blending factors are stored in 8.0 format. For most applications, the 6.2 format Δfog blending factors will have the two LSBs set to 0x0, with the six MSBs representing the difference between successive fog blending factors. Also note that as a result of the 6.2 format for the Δfog blending factors, the difference between successive fog blending factors cannot exceed 63. When storing the fog blending factors, the sum of each fog blending factor and Δfog blending factor pair must not exceed 255. When loading **fogTable**, two fog table entries must be written concurrently in a 32-bit word. A total of 32 32-bit PCI writes are required to load the entire **fogTable** register.

fogTable[n] ($0 \le n \le 31$)

Bit	Description
7:0	FogTable[2n] ΔFog blending factor
15:8	FogTable[2n] Fog blending factor
23:16	FogTable[2n+1] ΔFog blending factor
31:24	FogTable[2n+1] Fog blending factor

5.37 vRetrace Register

The **vRetrace** register is used to determine the position of the monitor vertical refresh beam. The **vRetrace** register allows software to read the status of the internal **vSync_off** counter used for vertical video timing. The **vRetrace** register allows an application to determine the amount of time before the next vertical sync. Note that **vRetrace** is read only. See section 11 for more information on video timing.

Bit	Description
11:0	internal **vSync_off** counter value

5.38 hSync Register

The **hSync** register specifies the timing values used to generate the horizontal sync (hsync) signal. See section 11 for more information on video timing.

Bit	Description

7:0	Horizontal sync on (internal **hSync_on** register)
15:8	reserved
25:16	Horizontal sync off (internal **hSync_off** register)

5.39 vSync Register

The **vSync** register specifies the timing values used to generate the vertical sync (vsync) signal. See section 11 for more information on video timing.

Bit	Description
11:0	Vertical sync on (internal **vSync_on** register)
15:12	reserved
27:16	Vertical sync off (internal **vSync_off** register)

5.40 backPorch Register

The **backPorch** register specifies the timing values used to define the video backporch area. See section 11 for more information on video timing.

Bit	Description
7:0	Horizontal backporch (internal **hBackPorch** register)
15:8	reserved
23:16	Vertical backporch (internal **vBackPorch** register)

5.41 videoDimensions Register

The **videoDimensions** register specifies the dimensions used to generate video timing values. See section 11 for more information on video timing.

Bit	Description
9:0	X (width) dimension (internal **xWidth** register)
15:10	reserved
25:16	Y (height) dimension (internal **yHeight** register)

5.42 fbiInit0 Register

The **fbiInit0** register is used for hardware initialization and configuration of the FBI chip. Writes to **fbiInit0** are ignored unless PCI configuration register **initWrEnable** bit(0)=1. Writes to **fbiInit0** are not put into the PCI bus FIFO and are written immediately, so care must be taken when writing to **fbiInit0** if data is in the PCI bus FIFO or the graphics engine is busy.

Bit	Description
	Miscellaneous Control
0	VGA passthrough (controls external pins **vga_pass** and **vga_pass_n**). Default is defined by strapping pin **fb_addr**[4].
1	FBI Graphics Reset (0=run, 1=reset). Default is 0.
2	FBI FIFO Reset (0=run, 1=reset). Default is 0. [resets PCI FIFO and the PCI data packer]
3	Byte swizzle incoming register writes (1=enable). [Register byte data is swizzled if **fbiInit0**[3]==1 and pci_address[21]==1]. Default is 0.

	FIFO Control
4	Stall PCI enable for High Water Mark (0=disable, 1=enable). Default is 1.
5	reserved
10:6	PCI FIFO Empty Entries Low Water Mark. Valid values are 0-31. Default is 0x10.
11	Linear frame buffer accesses stored in memory FIFO (1=enable). Default is 0.
12	Texture memory accesses stored in memory FIFO (1=enable). Default is 0.
13	Memory FIFO enable (0=disable, 1=enable). Default is 0.
24:14	Memory FIFO High Water Mark (bits [15:5]). Default is 0x0.
30:25	Memory FIFO Write Burst High Water Mark (Range 0-63 -- must be greater than **fbiinit4**[7:2]). Default is 0x0.
31	reserved

5.43 fbiInit1 Register

The **fbiInit1** register is used for hardware initialization and configuration of the FBI chip. Writes to **fbiInit1** are ignored unless PCI configuration register **initWrEnable** bit(0)=1. Writes to **fbiInit1** are not put into the PCI bus FIFO and are written immediately, so care must be taken when writing to **fbiInit1** if data is in the PCI bus FIFO or the graphics engine is busy.

Bit	Description
	PCI Bus Controller Configuration
0	PCI Device Function Number, specifed with power-on latched values (0=pass-thru SST-1 only, 1=combo board with VGA dev #0 and SST-1 dev#1). Read only.
1	Wait state cycles for PCI write accesses (0=no ws, 1=one ws). Default is 1.
2	Multi-SST configuration detect (0=one SST-1 configuration, 1=two SST-1 configuration). Read only.
3	Enable linear frame buffer reads (1=enable). Default is 0. This bit is included so that SST-1 potentially won't hang the system during random reads during powerup.
	Video Controller Configuration (1)
7:4	Number of 64x16 video tiles in X dimension divided by 2 (3.1 format). Default is 0.
8	Video Timing Reset (0=run, 1=reset). Default is 1.
9	Software override of HSYNC/VSYNC (0=normal operation, 1=software override). Default is 0.
10	Software override HSYNC value. Default is 0.
11	Software override VSYNC value. Default is 0.
12	Software blanking enable (0=normal operation, 1=Always blank monitor). Default is 1.
13	Drive video timing data outputs (0=tristate, 1=drive outputs). Default is 0.
14	Drive video timing blank output (0=tristate, 1=drive output). Default is 0.
15	Drive video timing hsync/vsync outputs (0=tristate, 1=drive outputs). Default is 0.
16	Drive video timing dclk output (0=tristate, 1=drive output). Default is 0.
17	Video timing vclk input select (0=vid_clk_2x, 1=vid_clk_slave). Default is 0.
19:18	Vid_clk_2x delay select (0=no delay, 1=4 ns, 2=6 ns, 3=8 ns). Default is 0.
21:20	Video timing vclk source select (0=vid_clk_slave, 1=vid_clk_2x [divided by 2], 2,3=vid_clk_2x_sel). Default is 2.
22	Enable 24 Bits-per-pixel video output (1=enable). Default is 0.
23	Enable scan-line interleaving (1=enable). Default is 0.

24	reserved
25	Enable video edge detection filtering (1=enable). Default is 0.
26	Invert vid_clk_2x (0=pass-thru vid_clk_2x, 1=invert vid_clk_2x). Default is 0.
28:27	Vid_clk_2x_sel delay select (0=no delay, 1=4 ns, 2=6 ns, 3=8 ns). Default is 0.
30:29	Vid_clk delay select (0=no delay, 1=4 ns, 2=6 ns, 3=8 ns). Default is 0.
31	Disable fast Read-Ahead-Write to Read-Ahead-Read turnaround (1=disable). Default is 0.

5.44 fbiInit2 Register

The **fbiInit2** register is used for hardware initialization and configuration of the FBI chip. Writes to **fbiInit2** are ignored unless PCI configuration register **initWrEnable** bit(0)=1. Writes to **fbiInit2** are not put into the PCI bus FIFO and are written immediately, so care must be taken when writing to **fbiInit2** if data is in the PCI bus FIFO or the graphics engine is busy. TO BE COMPLETED.

Bit	Description
	DRAM Memory Controller Configuration
0	Enable video dither subtraction (1=enable). Default is 0x0.
1	DRAM banking configuration (0=128Kx16 banking, 1=256Kx16 banking)
3:2	reserved
4	Triple Buffering Enable (1=enable). Default is 0x0.
5	Enable fast RAS read cycles [bring RAS high early on reads] (1=enable). Default is 0x0.
6	Enable generated dram OE signal (1=enable). Default is 0x1.
7	Enable fast Read-Ahead -Write turnaround [bit(6) must be set]. (1=enable). Default is 0x0.
8	Enable pass-through dither mode [For 8 BPP apps only] (1=enable). Default is 0x0.
10:9	Swap buffer algorithm (0=based on **dac_vsync**, 1=based on **dac_data**(0), 2=based on pci_**fifo_stall**, 3=reserved). Default is 0x0.
	Video/DRAM Controller Configuration (2)
19:11	Video Buffer Offset (=150 for 640x480, =297 for 832x608). Default is 0x0.
20	Enable DRAM banking (1=enable). Default is 0x0.
21	Enable DRAM Read Ahead FIFO (1=enable). Default is 0x0.
	DRAM Refresh Control
22	Refresh Enable (0=disable, 1=enable). Default is 0.
31:23	Refresh_Load Value. (Internal 14-bit counter 5 LSBs are 0x0). Default is 0x100.

5.45 fbiInit3 Register

The **fbiInit3** register is used for hardware initialization and configuration of the FBI chip. Writes to **fbiInit3** are ignored unless PCI configuration register **initWrEnable** bit(0)=1. Writes to **fbiInit3** are not put into the PCI bus FIFO and are written immediately, so care must be taken when writing to **fbiInit3** if data is in the PCI bus FIFO or the graphics engine is busy. TO BE COMPLETED.

Bit	Description
	Miscellaneous Control

0	Triangle register address remapping (0=use normal register mapping, 1=use aliased register mapping). Default is 0x0.
5:1	Video Fifo threshold. Default is 0x0.
6	Disable Texture Mapping (0=normal, 1=disable Trex-to-FBI Interface). Default is 0x0.
7	reserved
	FBI power-on configuration bits (read only)
10:8	FBI memory type (000=EDO DRAM, 001=Synch. DRAM)
11	VGA_PASS reset value
12	Hardcode PCI base address 0x1f000000 (1=enable, 0=normal operation).
	TREX interface configuration bits
16:13	fbi-to-trex bus clock delay selections (0-15). Default is 0x2.
21:17	trex-to-fbi bus fifo full threshold (0-31). Default is 0xf.
	Y Origin Definition bits
31:22	Y Origin Swap subtraction value (10 bits). Default is 0x0.

5.46 fbiInit4 Register

The **fbiInit4** register is used for hardware initialization and configuration of the FBI chip. Writes to **fbiInit4** are ignored unless PCI configuration register **initWrEnable** bit(0)=1. Writes to **fbiInit4** are not put into the PCI bus FIFO and are written immediately, so care must be taken when writing to **fbiInit4** if data is in the PCI bus FIFO or the graphics engine is busy. TO BE COMPLETED.

Bit	Description
	Miscellaneous Control
0	Wait state cycles for PCI read accesses (0=1 ws, 1=2 ws). Default is 1.
1	Enable Read-ahead logic for linear frame buffer reads (1=enable). Default is 0.
7:2	Memory FIFO low water mark for PCI Fifo. [Dump PCI fifo contents to memory if PCI fifo freespace falls below this level]. Default is 0.
17:8	Memory FIFO row start (base row address for beginning of memory FIFO). Default is 0.
27:18	Memory FIFO row rollover (row value when fifo counters rollover). Default is 0.
31:28	reserved

5.47 clutData Register

The **clutData** register is used the load values into the internal Color Lookup table. The FBI internal Color Lookup table is used for gamma correction of 16-bit RGB values during video refresh. The 16-bit RGB values read from the frame buffer are used to index into the internal Color Lookup table. The output of the Color Lookup table is then fed to an external DAC. The Color Lookup Table is stored internally as a 33x24 RAM. As RGB values are input from memory, the 5 MSBs of a particular color channel are used to index into the Color Lookup Table. The 3 LSBs of a particular color channel are then used to linearly interpolate between multiple Color Lookup Table entries. As a result of the linear interpolation performed, smooth transitions from one Color Lookup Table index to surrounding indices results. To modify an entry in the Color Lookup Table, writes are performed to the **clutData** register. Notice that the index of the Color Lookup Table entry to be modified is stored in the data passed to the **clutData** register.

Bit	Description
7:0	Blue color component to be written to Color Lookup Table
15:8	Green color component to be written to Color Lookup Table
23:16	Red color component to be written to Color Lookup Table
29:24	Index of Color Lookup Table to be written (Range 0-32 only).

5.48 dacData Register

The **dacData** register is write to the external DAC. Writes to the external DAC are only allowed when the memory bus is idle, as the external DAC register bus is time-multiplexed with the DRAM data lines. Thus, software must guarantee that there are no conflicts between the DRAM memory controller and exteranal DAC accesses. This can be accomplished either by holding the video control unit in reset (**fbiinit1** bit(8)=1) and flushing the pixel pipeline with a NOP command, or by waiting for VSYNC to be active and also flushing the pixel pipeline. Writes to the external DAC are performed by writing the **dacData** register, with bits(7:0) specifying the register data, bits (10:8) specifying the register address, and bit(11) cleared to 0. Bit(11) of **dacData** must be cleared when performing external DAC writes. Reads from the External DAC are performed by *writing* to the **dacData** register with the register address specified in bits (10:8) and bit(11) set to 1. Bit(11) of **dacData** must be set when performing external DAC reads. The data read from the External DAC is stored in an internal register of FBI, and is read by setting bit(2) in the PCI Configuration register **initEnable** and reading from the fbiinit2 register. When **fbiinit2**/**fbiinit3** address remapping is enabled (PCI Configuration register **initEnable** bit(2)=1), reading from **fbiinit2** bits (7:0) returns the last value read from the external DAC (**fbiinit2** bits(31:8) are undefined when address remapping is enabled). Note that reading from the external DAC is a two-step process: first the read is initiated by writing to the **dacData** register with bit(11) set to 1; the read data is then read by the CPU by reading from **fbinit2** bits(7:0) with **fbiinit2**/**fbinit3** address remapping is enabled.

Bit	Description
7:0	External DAC register write data
10:8	External DAC register address
11	External DAC read command (1=read external DAC, 0=write external DAC)

5.49 maxRgbDelta Register

TO BE COMPLETED.

Bit	Description
7:0	Maximum blue delta for video filtering
15:8	Maximum green delta for video filtering
23:16	Maximum red delta for video filtering

5.50 textureMode Register

The **textureMode** register controls texture mapping functionality including perspective correction, texture filtering, texture clamping, and multiple texture blending.

Bit	Name	Description
0	*tpersp_st*	Enable perspective correction for S and T iterators (0=linear interploation of S,T, force W to 1.0, 1=perspective correct, S/W, T/W)
1	*tminfilter*	Texture minification filter (0=point-sampled, 1=bilinear)
2	*tmagfilter*	Texture magnification filter (0=point-sampled, 1=bilinear)
3	*tclampw*	Clamp when W is negative (0=disabled, 1=force S=0, T=0 when W is negative)

4	*tloddither*	Enable Level-of-Detail dithering (0=no dither, 1=dither)
5	*tnccselect*	Narrow Channel Compressed (NCC) Table Select (0=table 0, 1=table 1)
6	*tclamps*	Clamp S Iterator (0=wrap, 1=clamp)
7	*tclampt*	Clamp T Iterator (0=wrap, 1=clamp)
11:8	*tformat*	Texture format (see table below)
		Texture Color Combine Unit control (RGB):
12	*tc_zero_other*	Zero Other (0=c_other, 1=zero)
13	*tc_sub_clocal*	Subtract Color Local (0=zero, 1=c_local)
16:14	*tc_mselect*	Mux Select (0=zero, 1=c_local, 2=a_other, 3=a_local, 4=LOD, 5=LOD_frac, 6-7=reserved)
17	*tc_reverse_blend*	Reverse Blend (0=normal blend, 1=reverse blend)
18	*tc_add_clocal*	Add Color Local
19	*tc_add_alocal*	Add Alpha Local
20	*tc_invert_output*	Invert Output
		Texture Alpha Combine Unit control (A):
21	*tca_zero_other*	Zero Other (0=c_other, 1=zero)
22	*tca_sub_clocal*	Subtract Color Local (0=zero, 1=c_local)
25:23	*tca_mselect*	Mux Select (0=zero, 1=c_local, 2=a_other, 3=a_local, 4=LOD, 5=LOD_frac, 6-7=reserved)
26	*tca_reverse_blend*	Reverse Blend (0=normal blend, 1=reverse blend)
27	*tca_add_clocal*	Add Color Local
28	*tca_add_alocal*	Add Alpha Local
29	*tca_invert_output*	Invert Output
30	*trilinear*	Enable trilinear texture mapping (0=point-sampled/bilinear, 1=trilinear)
31	*seq_8_downld*	Sequential 8-bit download (0=even 32-bit word addresses, 1=sequential addresses) (Must be set to 0 for revision 0 TMU)

tpersp_st bit of **textureMode** enables perspective correction for S and T iterators. Note that there is no performance penalty for performing perspective corrected texture mapping.

tminfilter, tmagfilter bits of **textureMode** specify the filtering operation to be performed. When point sampled filtering is selected, the texel specified by <s,t> is read from texture memory. When bilinear filtering is selected, the four closet texels to a given <s,t> are read from memory and blended together as a function of the fractional components of <s,t>. *tminfilter* is referenced when LOD>=LODmin, otherwise *tmagfilter* is referenced.

tclampw bit of **textureMode** is used when projecting textures to avoid projecting behind the source of the projection. If this bit is set, S, T are each forced to zero when W is negative. Though usually desireable, it is not necessary to set this bit when doing projected textures.

tloddither bit of **textureMode** enables Level-of-Detail (LOD) dither. Dithering the LOD calculation is useful when performing texture mipmapping to remove the LOD bands which can occur from with mipmapping without trilinear filtering. This adds an average of 3/8 (.375) to the LOD value and needs to compensated in the amount of *lodbias*.

tnccselect bit of **textureMode** selects the NCC lookup table to be used when decompressing 8-bit NCC textures.

tclamps, tclampt bits of **textureMode** enable clamping of the S and T texture iterators. When clamping is enabled, the S iterator is clamped to [0, texture width) and the T iterator is clamped to [0, texture height). When clamping is disabled, S coordinates outside of [0, texture width) are allowed to wrap into the [0, texture width)

range using bit truncation. Similarly when clamping is disabled, T coordinates outside of [0, texture height) are allowed to wrap into the [0, texture height) range using bit truncation.

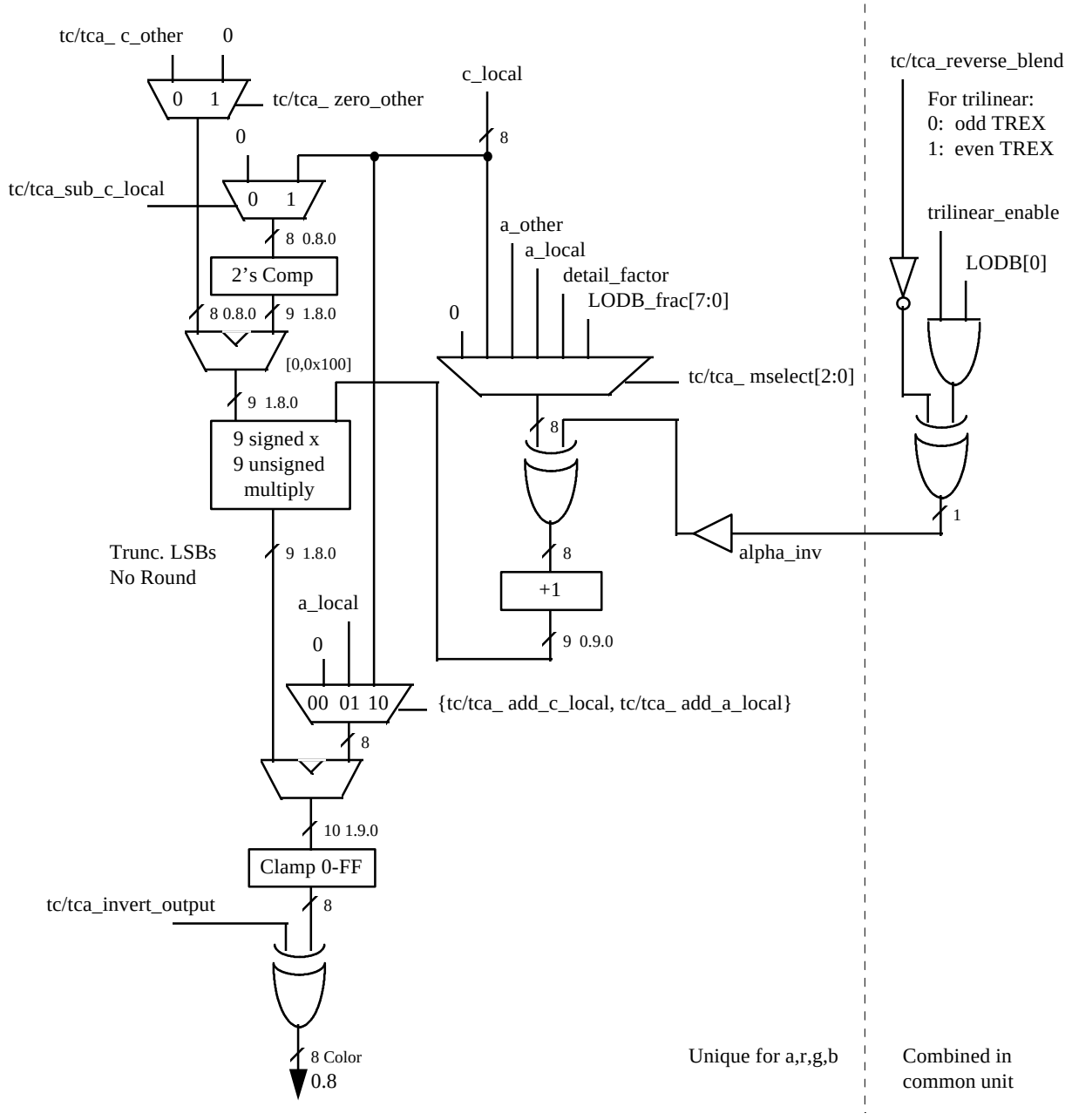
tformat field of **textureMode** specifies the texture format accessed by TREX. Note that the texture format field is used for both reading and writing of texture memory. The following table shows the texture formats and how the texture data is expanded into 32-bit ARGB color:

***tforma t* Value**	**Texture format**	**8-bit Alpha**	**8-bit Red**	**8-bit Green**	**8-bit Blue**
0	8-bit RGB (3-3-2)	0xff	{r[2:0],r[2:0],r[2:1]}	{g[2:0],g[2:0],g[2:1]}	{b[1:0],b[1:0],b[1:0],b[1:0]}
1	8-bit YIQ (4-2-2)	0xff	ncc _red[7:0]	ncc _green[7:0]	ncc _blue[7:0]
2	8-bit Alpha	a[7:0]	a[7:0]	a[7:0]	a[7:0]
3	8-bit Intensity	0xff	i [7:0]	i[7:0]	i[7:0]
4	8-bit Alpha, Intensity (4-4)	{a[3:0],a[3:0]}	{i[3:0],i[3:0]}	{i[3:0],i[3:0]}	{i[3:0],i[3:0]}
5	8-bit Palette (not revisoin 0 TMU)	0xff	palette r[7:0]	palette g[7:0]	palette b[7:0]
6-7	Reserved				
8	16-bit ARGB (8-3-3-2)	a[7:0]	{r[2:0],r[2:0],r[2:1]}	{g[2:0],g[2:0],g[2:1]}	{b[1:0],b[1:0],b[1:0],b[1:0]}
9	16-bit AYIQ (8-4-2-2)	a[7:0]	ncc _red[7:0]	ncc _green[7:0]	ncc _blue[7:0]
10	16-bit RGB (5-6-5)	0xff	{r[4:0],r[4:2]}	{g[5:0],r[5:4]}	{b[4:0],b[4:2]}
11	16-bit ARGB (1-5-5-5)	{a[0],a[0],a[0],a[0], a[0],a[0],a[0],a[0]}	{r[4:0],r[4:2]}	{g[4:0],g[4:2]}	{b[4:0],b[4:2]}
12	16-bit ARGB (4-4-4-4)	{a[3:0},a[3:0]}	{r[3:0},r[3:0]}	{g[3:0},g[3:0]}	{b[3:0},b[3:0]}
13	16-bit Alpha, Intensity (8-8)	a[7:0]	i[7:0]	i[7:0]	i[7:0]
14	16-bit Alpha, Palette (8-8) (not revision 0 TMU)	a[7:0]	palette r[7:0]	palette g[7:0]	palette b[7:0]
15	Reserved				

where a, r, g, b, and i(intensity) represent the actual values read from texture memory. YIQ texture and palette formats are detailed later in the nccTable description and palette description.

There are three Texture Color Combine Units (RGB) and one Texture Alpha Combine Unit(A), all four are identical, except for the bit fields that control them. The *tc_** fields of **textureMode** control the Texture Color Combine Units; the *tca_** fields control the Texture Alpha Combine Units. The diagram below illustrates the Texture Color Combine Unit/Texture Alpha Combine Unit:

Blend with Incoming Color



tc_ prefix applies to R,G and B channels. tca_ prefix applies to A channel.

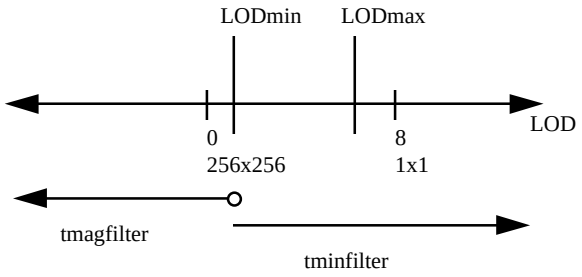
5.51 tLOD Register

The **tLOD** register controls the texture mapping LOD calculations.

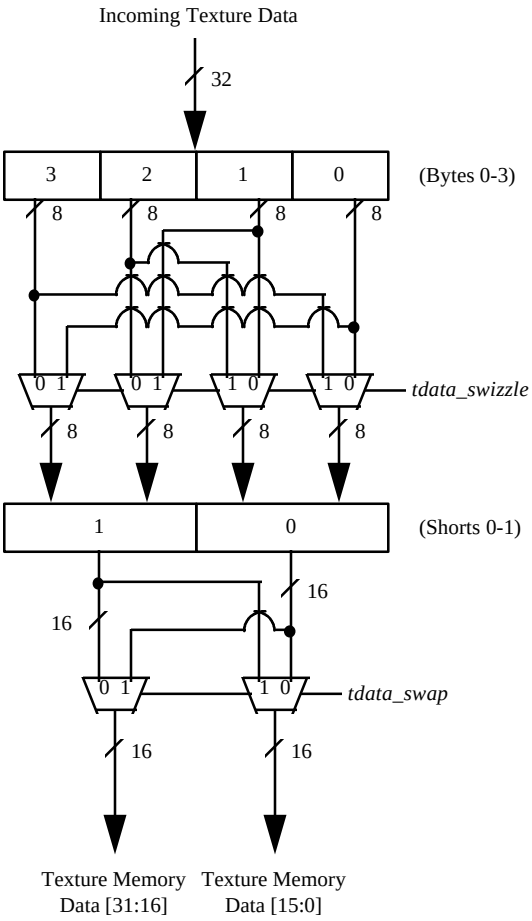
Bit	Name	Description
5:0	*lodmin*	Minimum LOD. (4.2 unsigned)
11:6	*lodmax*	Maximum LOD. (4.2 unsigned)
17:12	*lodbias*	LOD Bias. (4.2 signed)
18	*lod_odd*	LOD odd (0=even, 1=odd)
19	*lod_tsplit*	Texture is Split. (0=texture contains all LOD levels, 1=odd or even levels only, as controlled by *lod_odd*)
20	*lod_s_is_wider*	S dimension is wider, for rectilinear texture maps. This is a *don't care* for square textures. (1=S is wider than T).
22:21	*lod_aspect*	Aspect ratio. Equal to 2^n. (00 is square texture, others are rectilinear: 01 is 2x1/1x2, 10 is 4x1/1x4, 10 is 8x1/1x8)
23	*lod_zerofrac*	LOD zero frac, useful for bilinear when even and odd levels are split across two TREXs (0=normal LOD frac, 1=force fraction to 0)
24	*tmultibaseaddr*	Use multiple texbaseAddr registers
25	*tdata_swizzle*	Byte swap incoming texture data (bytes 0<->3, 1<->2).
26	*tdata_swap*	Short swap incoming texture data (shorts 0<->1).
27	*tdirect_write*	Enable raw direct texture memory writes (1=enable). *seq_8_downld* must equal 0.

lodbias is added to the calculated LOD value, then it is clamped to the range [*lodmin*, min(8.0, *lodmax*)]. Note that whether the LOD is clamped to *lodmin* is used to determine whether to use the minification or magnification filter, selected by the *tminfilter* and *tmagfilter* bits of **textureMode**:

LOD bias, clamp



The *tdata_swizzle* and *tdata_swap* bits in **tLOD** are used to modify incoming texture data for endian dependencies. The *tdata_swizzle* bit causes incoming texture data bytes to be byte order reversed, such that bits(31:24) are swapped with bits(7:0), and bits(23:16) are swapped with bits(15:8). Short-word swapping is performed after byte order swizzling, and is selected by the *tdata_swap* bit in **tLOD**. When enabled, short-word swapping causes the post-swizzled 16-bit shorts to be order reversed, such that bits(31:16) are swapped with bits(15:0). The following diagram shows the data manipulation functions perfomed by the *tdata_swizzle* and *tdata_swap* bits:



5.52 tDetail Register

The **tDetail** register controls the detail texture.

Bit	Name	Description
7:0	*detail _max*	Detail texture LOD clamp (8.0 unsigned)
13:8	*detail_bias*	Detail texture bias (6.0 signed)
16:14	*detail_scale*	Detail texture scale shift left

detail_factor is used in the Texture Combine Unit to blend between the main texture and the detail texture.
detail_factor (0.8 unsigned) = max(*detail_max*, ((*detail_bias* - LOD) << *detail_scale*))

5.53 texBaseAddr, texBaseAddr1, texBaseAddr2, and texBaseAddr38 Registers

The **texBaseAddr** register specifies the starting texture memory address for accessing a texture, at a granularity of 8 bytes. It is used for both texture writes and rendering. Calculation of the texbaseaddr is described in the **Texture Memory Access** section. Selection of the base address is a function of *tmultibaseaddr* and LODBI.

Bit	Name	Description
18:0	*texbaseaddr*	Texture Memory Base Address, *tmultibaseaddr*==0 or LODBI==0

18:0	*texbaseaddr1*	Texture Memory Base Address, *tmultibaseaddr*==1 and LODBI==1
18:0	*texbaseaddr2*	Texture Memory Base Address, *tmultibaseaddr*==1 and LODBI==2
18:0	*texbaseaddr38*	Texture Memory Base Address, *tmultibaseaddr*==1 and LODBI>=3

.

5.54 trexInit0 Register

The **trexInit0** register is used for hardware initialization and configuration of the TREX chip(s). TO BE COMPLETED. See TREX spec.

5.55 trexInit1 Register

The **trexInit1** register is used for hardware initialization and configuration of the TREX chip(s). TO BE COMPLETED. See TREX spec.

5.56 nccTable0 and nccTable1/Palette Registers

The **nccTable0** and **nccTable1** registers contain two Narrow Channel Compression (NCC) tables used to store lookup values for compressed textures (used in YIQ and AYIQ texture formats as specified in *tformat* of **textureMode**). These registers are also used to write the palette.

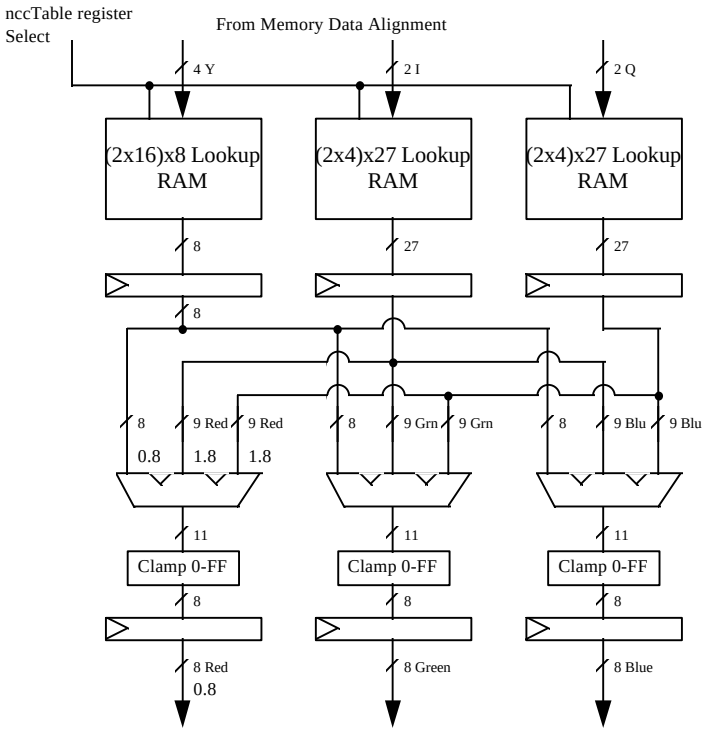
5.56.1 NCC Table

Two tables are stored so that they can be swapped on a per-triangle basis when performing multi-pass rendering, thus avoiding a new download of the table. Use of either **nccTable0** or **nccTable1** is selected by the Narrow Channel Compressed (NCC) Table Select bit of **textureMode**. **nccTable0** and **nccTable1** are stored in the format of the table below, and are write only.

nccTable Address	Bits	Contents
0	31:0	{Y3[7:0], Y2[7:0], Y1[7:0], Y0[7:0]}
1	31:0	{Y7[7:0], Y6[7:0], Y5[7:0], Y4[7:0]}
2	31:0	{Yb[7:0], Ya[7:0], Y9[7:0], Y8[7:0]}
3	31:0	{Yf[7:0], Ye[7:0], Yd[7:0], Yc[7:0]}
4	26:0	{I0_r[8:0], I0_g[8:0], I0_b[8:0]}
5	26:0	{I1_r[8:0], I1_g[8:0], I1_b[8:0]}
6	26:0	{I2_r[8:0], I2_g[8:0], I2_b[8:0]}
7	26:0	{I3_r[8:0], I3_g[8:0], I3_b[8:0]}
8	26:0	{Q0_r[8:0], Q0_g[8:0], Q0_b[8:0]}
9	26:0	{Q1_r[8:0], Q1_g[8:0], Q1_b[8:0]}
10	26:0	{Q2_r[8:0], Q2_g[8:0], Q2_b[8:0]}
11	26:0	{Q3_r[8:0], Q3_g[8:0], Q3_b[8:0]}

Undefined MSB's must be written as 0's, or the writes may be interpreted as palette writes.

The following figure illustrates how compressed textures are decompressed using the NCC tables:



5.56.2 8-Bit Palette (not revision 0 TMU)

The 8-bit palette is used for 8-bit P and 16-bit AP modes. The palette is loaded with register writes. During rendering, four texels are looked up simultaneously, each an independent 8-bit address.

Palette Write

The palette is written through the NCC table 0 I and Q register space when the MSB of the register write data is set. The NCC table write is inhibited.

Palette Load Mechanism

Register Address	LSB of P	Register Write Data 31				0
nccTable0 I0	P[0]=0	1	P[7:1]	R[7:0]	G[7:0]	B[7:0]
nccTable0 I1	P[0]=1	1	P[7:1]	R[7:0]	G[7:0]	B[7:0]
nccTable0 I2	P[0]=0	1	P[7:1]	R[7:0]	G[7:0]	B[7:0]
nccTable0 I3	P[0]=1	1	P[7:1]	R[7:0]	G[7:0]	B[7:0]
nccTable0 Q0	P[0]=0	1	P[7:1]	R[7:0]	G[7:0]	B[7:0]
nccTable0 Q1	P[0]=1	1	P[7:1]	R[7:0]	G[7:0]	B[7:0]
nccTable0 Q2	P[0]=0	1	P[7:1]	R[7:0]	G[7:0]	B[7:0]
nccTable0 Q3	P[0]=1	1	P[7:1]	R[7:0]	G[7:0]	B[7:0]

Note that the even addresses alias to the same location, as well as the odd ones. It is recommended that the table be written as 32 sets of 8 so that PCI bursts can be 8 transfers long.

6. PCI Configuration Register Set

Register Name	Addr	Bits	Description
Vendor_ID	0 (0x0)	15:0	3Dfx Interactive Vendor Identification
Device_ID	2 (0x2)	15:0	Device Identification
Command	4 (0x4)	15:0	PCI bus configuration
Status	6 (0x6)	15:0	PCI device status
Revision_ID	8 (0x8)	7:0	Revision Identification
Class_code	9 (0x9)	23:0	Generic functional description of PCI device
Cache_line_size	12 (0xc)	7:0	Bus Master Cache Line Size
Latency_timer	13 (0xd)	7:0	Bus Master Latency Timer
Header_type	14 (0xe)	7:0	PCI Header Type
BIST	15 (0xf)	7:0	Build In Self-Test Configuration
memBaseAddr	16 (0x10)	31:0	Memory Base Address
Reserved	20-59 (0x14-0x3b)		Reserved
Interrupt_line	60 (0x3c)	7:0	Interrupt Mapping
Interrupt_pin	61 (0x3d)	7:0	External Interrupt Connections
Min_gnt	62 (0x3e)	7:0	Bus Master Minimum Grant Time
Max_lat	63 (0x3f)	7:0	Bus Master Maximum Latency Time
initEnable	64 (0x40)	31:0	Allow writes to hardware initialization registers
busSnoop0	68 (0x44)	31:0	FBI bus snooping address 1 (write only)
busSnoop1	72 (0x48)	31:0	FBI bus snooping address 0 (write only)
cfgStatus	76 (0x4c)	31:0	Aliased memory-mapped **status** register
Reserved	80-255 (0x50-0xff)	n/a	Reserved

6.1 Vendor_ID Register

The **Vendor_ID** register is used to identify the manufacturer of the PCI device. This value is assigned by a central authority that will control issuance of the values. This register is read only.

Bit	Description
7:0	3Dfx Interactive Vendor Identification. Default is 0x121a.

6.2 Device_ID Register

The **Device_ID** register is used to identify the particular device for a given manufacturer. This register is read only.

Bit	Description
15:0	SST-1 Device Identification. Default is 0x1.

6.3 Command Register

The **Command** register is used to control basic PCI bus accesses. See the PCI specification for more information. Bit 1 is R/W, and bits 0, 15:2 are read only.

Bit	Description

0	I/O Access Enable. Default is 0.
1	Memory Access Enable (0=no response to memory cycles). Default is 0.
2	Master Enable. Default is 0.
3	Special Cycle Recognition. Default is 0.
4	Memory Write and Invalidate Enable. Default is 0.
5	Palette Snoop Enable. Default is 0.
6	Parity Error Respond Enable. Default is 0.
7	Wait Cycle Enable. Default is 0.
8	System Error Enable. Default is 0.
15:9	reserved. Default is 0x0.

6.4 Status Register

The **Status** register is used to monitor the status of PCI bus-related events. This register is read only and is hardwired to the value 0x0.

Bit	Description
7:0	Reserved. Default is 0x0.
8	Data Parity Reported. Default is 0.
10:9	Device Select Timing. Default is 0x0.
11	Signalled Target Abort. Default is 0.
12	Received Target Abort. Default is 0.
13	Received Master Abort. Default is 0.
14	Signalled System Error. Default is 0.
15	Detected Parity Error. Default is 0.

6.5 Revision_ID Register

The **Revision_ID** register is used to identify the revision number of the PCI device. This register is read only.

Bit	Description
7:0	SST-1 Revision Identification. Value represents the current revision number.

6.6 Class_code Register

The **Class_code** register is used to identify the generic functionality of the PCI device. See the PCI specification for more information. This register is read only.

Bit	Description
23:0	Class Code. Default is 0x0.

6.7 Cache_line_size Register

The **Cache_line_size** register specifies the system cache line size in doubleword increments. It must be implemented by devices capable of bus mastering. This register is read only and is hardwired to 0x0.

Bit	Description
7:0	Cache Line Size. Default is 0x0.

6.8 Latency_timer Register

The **Latency_timer** register specifies the latency of bus master timeouts. It must be implemented by devices capable of bus mastering. This register is read only and is hardwired to 0x0.

Bit	Description
7:0	Latency Timer. Default is 0x0.

6.9 Header_type Register

The **Header_type** register defines the format of the PCI base address registers (**memBaseAddr** in SST-1). Bits 0:6 are read only and hardwired to 0x0. Bit 7 of **Header_type** specifies SST-1 as a single function PCI device.

Bit	Description
6:0	Header Type. Default is 0x0.
7	Multiple-Function PCI device (0=single function, 1=multiple function). Default is 0x0.

6.10 BIST Register

The **BIST** register is implemented by those PCI devices that are capable of built-in self-test. SST-1 does not provide this capability. This register is read only and is hardwired to 0x0.

Bit	Description
7:0	BIST field and configuration. Default is 0x0.

6.11 memBaseAddr Register

The **memBaseAddr** register determines the base address for all PCI memory mapped accesses to SST-1. Writing 0xffffffff to this register will reset it to its default state. Once **memBaseAddr** has been reset, it can be probed by software to determine the amount of memory space required for SST-1. A subsequent write to **memBaseAddr** will set the memory base address for all PCI memory accesses. See the PCI specification for more details on memory base address programming. SST-1 requires 16 MBytes of address space for memory mapped accesses. For memory mapped accesses on the 32-bit PCI bus, the contents of **memBaseAddr** are compared with the **pci_ad** bits 31..24 (upper 8 bits) to determine if SST-1 is being accessed. This register is R/W.

Bit	Description
31:0	Memory Base Address. Default is 0xff000000.

6.12 Interrupt_line Register

The **Interrupt_line** register is used to map PCI interrupts to system interrupts. In a PC environment, for example, the values of 0 to 15 in this register correspond to IRQ0-IRQ15 on the system board. The value 0xff indicates no connection. This register is R/W.

Bit	Description
0:7	Interrupt Line. Default is 0x5 (IRQ5)

6.13 Interrupt_pin Register

The **Interrupt_pin** register defines which of the four PCI interrupt request lines, INTA* - INTRD*, the PCI device is connected to. This register is read only and is hardwired to 0x1.

Bit	Description
0:7	Interrupt Pin. Default is 0x1 (INTA*)

6.14 Min_gnt Register

The **Min_gnt** register specifies the burst period a PCI bus master requires. It must be implemented by devices capable of bus mastering. This register is read only and is hardwired to 0x0 since SST-1 does not support bus mastering.

Bit	Description
7:0	Minimum Grant. Default is 0x0.

6.15 Max_lat Register

The **Max_lat** register specifies the maximum request frequency a PCI bus master requires. It must be implemented by devices capable of bus mastering. This register is read only and is hardwired to 0x0 since SST-1 does not support bus mastering.

Bit	Description
7:0	Maximum Latency. Default is 0x0.

6.16 initEnable Register

The **initEnable** register controls write access to the **fbiinit** registers and also controls the FBI PCI bus snooping functionality. Bit(0) of **initEnable** enables writes to the FBI hardware initialization registers **fbiInit0**, **fbiInit1**, **fbiInit2**, and **fbiInit3**. By default writes to the hardware initialization registers are not allowed. Writes to the hardware initialization registers when **initEnable** bit(0)=0 are ignored. Bit(1) of **initEnable** enables writes to the PCI FIFO. Bit(1) of **initEnable** must be set for normal SST-1 operation. Bits (9:4) of **initEnable** control the FBI PCI bus snooping functionality. See the **busSnoop** register description for more information on FBI bus snooping. Bit(10) of **initEnable** determines which scanline interleave device (master or slave) drives the PCI bus during scan line interleaving. When scanline interleaving is enabled (**fbiInit1**(23)=1), then **initEnable**(11) determines if FBI is the master or slave for scanline interleaving. If **initEnable**(11) and **initEnable**(10) are set to the same value, then the programmed FBI will drive the PCI bus during scanline interleaving.

Bit	Description
0	Enable writes to hardware initialization registers. (1=enable writes to the hardware initialization registers). Default is 0.
1	Enable writes to PCI FIFO (1=enable writes to PCI FIFO). Default is 0.
2	Remap {**fbiinit2**, **fbiinit3**} to {**dacRead**, **videoChecksum**} (1=enable). Default is 0.
3	reserved.
4	FBI snooping register 0 enable (1=enable). Default is 0.
5	FBI snooping register 0 memory matching type (0=memory access, 1=IO access). Default is 0.
6	FBI snooping register 0 read/write matching type (0=write access, 1=read access). Default is 0.
7	FBI snooping register 1 enable (1=enable). Default is 0.
8	FBI snooping register 1 memory matching type (0=memory access, 1=IO access). Default is 0.

9	FBI snooping register 1 read/write matching type (0=write access, 1=read access). Default is 0.
10	Scan-line interleaving PCI bus ownership. (0=SLI master owns PCI bus, 1=SLI slave owns PCI bus). Default is 0.
11	Scan-line interleaving master/slave determination (0=master/even scan lines, 1=slave/odd scan lines). Default is 0.
31:12	reserved

6.17 busSnoop0 and busSnoop1 Registers

The **busSnoop0** and **busSnoop1** registers control the FBI PCI bus "snooping" functionality. When bus snooping is enabled, a PCI cycle with characteristics (i.e. write/read type, io/mem type, etc). and address matching those characteristics and address specified in the **initEnable** and **busSnoop** registers will set the **vga_pass** FBI external pin. Note that the snooping functionality does not affect the PCI data transfer, as FBI does not own the address space specified in the snooping registers. **busSnoop** bits(1:0) are only used for IO PCI access types, as bits(1:0) of the PCI address are used to uniquely map IO space for PCI devices -- bits(1:0) of the **busSnoop** registers are ignored for PCI memory access types. The FBI snooping functionality is useful for making sure VGA passthrough capability does not drive the video monitor upon soft and hard resets. Note that the **busSnoop0** and **busSnoop1** registers are write-only, and will return 0x0 when read.

Bit	Description
1:0	PCI Snooping address register bits 1:0. (ignored for memory access types).
31:2	PCI Snooping address registers bits 31:2. Used for all PCI access types.

6.18 cfgStatus Register

The **cfgStatus** register is an alias to the normal memory-mapped **status** register. See section 5.1 for a description of the **status** register. Reading the configuration-space **cfgStatus** register will return the same data as if reading from the memory-mapped **status** register.

6.19 NOP Command

The NOP command performs no operation other than to flush the graphics pipeline. This command is used primarily for debugging and verification purposes.

6.20 Triangle Command

TO BE COMPLETED.

6.21 FASTFILL Command

The FASTFILL command is used for screen clears. When the FASTFILL command is executed, the depth-buffer comparison, alpha test, alpha blending, and all other special effects are bypassed and disabled. The FASTFILL command uses the status of the RGB write mask (bit(9) of **fbzMode**) and the depth-buffer write mask (bit(10) of **fbzMode**) to access the RGB/depth-buffer memory. The FASTFILL command also uses bits (15:14) of **fbzMode** to determine which RGB buffer (front or back) is written. Prior to executing the FASTFILL command, the **clipLeftRight** and **clipLowYHighY** registers must be loaded with a rectanglar area which is desired to be cleared -- -- the **fastfillCMD** register is then written to initiate the FASTFILL command. Note that **clip** registers define a rectangular area which is inclusive of the **clipLeft** and **clipLowY** register values, but exclusive of the **clipRight** and **clipHighY** register values. Note also that the relative position of the Y origin (either top of bottom of the

screen) is defined by **fbzMode** bit(17). The 24-bit color specified in the **Color1** register is written to the RGB buffer (with optional dithering as specified by bit(8) of **fbzMode**), and the depth value specified in bits(15:0) of the **zaColor** register is written to the depth buffer.

6.22 SWAPBUFF Command

The SWAPBUFF command is used to swap the drawing buffers for smooth animation. When the SWAPBUFF command is executed, **swapbufferCMD** bit(0) determines whether the drawing buffer swapping is synchronized with vertical retrace. Typically, it is desired that buffer swapping be synchronized with vertical retrace to eliminate frame “tearing” typically found on single buffered displays. If vertical retrace synchronization is enabled for double buffered applications, the graphics command processor will block on a SWAPBUFF command until the monitor vertical retrace signal is active. If the number of vertical retraces seen exceeds the value stored in bits(8:1) of **swapbufferCMD**, then the pointer used by the monitor refresh control logic is changed to point to another drawing buffer. If vertical retrace synchronization is enabled for triple buffered applications, the graphics processor does NOT block on a SWAPBUFF command. Instead, a flag is set in the monitor refresh control logic that automatically causes the data pointer to be modified in the monitor refresh control logic during the next active vertical retrace period. Using triple buffering allows rendering operations to occur without waiting for the vertical retrace active period.

When a **swapbufferCMD** is received in the front-end PCI host FIFO, the swap buffers pending field in the **status** register is incremented. Conversely, when an actual frame buffer swapping occurs, the swap buffers pending field in the **status** register (bits(30:28)) is decremented. The swap buffers pending field allows software to determine how many SWAPBUFFER commands are present in the SST-1 FIFOs. Note that for triple buffered applications, if a new SWAPBUFFER command is received and the swap buffers pending field is greater than zero, then the graphics processor must block until vertical retrace is active in order to ensure rendering does not occur over the frontbuffer.

7. Linear Frame Buffer Access

The SST-1 linear frame buffer base address is located at a 8 Mbyte offset from the **memBaseAddr** PCI configuration register and occupies 4 Mbytes of SST-1 address space (see section 4 for an SST-1 address map). Regardless of actual frame buffer resolution, all linear frame buffer accesses assume a 1024-pixel logical scan line width. The number of bytes per scan line depends on the format of linear frame buffer access format selected in the **lfbMode** register. Note for all accesses to the linear frame buffer, the status of bit(16) of **fbzMode** is used to determine the Y origin of data accesses. When bit(16)=0, offset 0x0 into the linear frame buffer address space is assumed to point to the upper-left corner of the screen. When bit(16)=1, offset 0x0 into the linear frame buffer address space is assumed to point to the bottom-left corner of the screen. Regardless of the status of **fbzMode** bit(16), linear frame buffer addresses increment as accesses are performed going from left-to-right across the screen. Also note that clipping is not automatically performed on linear frame buffer writes if scissor clipping is not explicitly enabled (**fbzMode** bit(0)=1). Linear frame buffer writes to areas outside of the monitor resolution when clipping is disabled will result in undefined behavior.

7.1 Linear frame buffer Writes

The following table shows the supported linear frame buffer write formats as specified in bits(3:0) of **lfbMode**:

Value	Linear Frame Buffer Access Format
	16-bit formats
0	16-bit RGB (5-6-5)
1	16-bit RGB (x-5-5-5)
2	16-bit ARGB (1-5-5-5)
3	Reserved
	32-bit formats
4	24-bit RGB (8-8-8)
5	32-bit ARGB (8-8-8-8)
7:6	Reserved
11:8	Reserved
12	16-bit depth, 16-bit RGB (5-6-5)
13	16-bit depth, 16-bit RGB (x-5-5-5)
14	16-bit depth, 16-bit ARGB (1-5-5-5)
15	16-bit depth, 16-bit depth

When writing to the linear frame buffer with a 16-bit access format (formats 0-3 and format 15 in **lfbMode**), each pixel written is 16-bits, so there are 2048 bytes per logical scan line. Remember when utilizing 16-bit access formats, two 16-bit values can be packed in a single 32-bit linear frame buffer write -- the location of each 16-bit component in screen space is defined by bit(11) of **lfbMode**. When using 16-bit linear frame buffer write formats 0-3, the depth components associated with each pixel is taken from the **zaColor** register. When using 16-bit format 3, the alpha component associated with each pixel is taken from the 16-bit data transfered, but when using 16-bit formats 0-2 the alpha component associated with each pixel is taken from the **zaColor** register. The format of the individual color channels within a 16-bit pixel is defined by the RGB channel format field in **lfbMode** bits(12:9). See the **lfbMode** description in section 5 for a detailed description of the rgb channel format field.

When writing to the linear frame buffer with 32-bit access formats 4 or 5, each pixel is 32-bits, so there are 4096 bytes per logical scan line. Note that when utilizing 32-bit access formats, only a single pixel may be written per 32-bit linear frame buffer write. Also note that linear frame buffer writes using format 4 (24-bit RGB (8-8-8)),

while 24-bit pixels, must be aligned to a 32-bit (doubleword) boundary -- packed 24-bit linear frame buffer writes are not supported by SST-1. When using 32-bit linear frame buffer write formats 4-5, the depth components associated with each pixel is taken from the **zaColor** register. When using format 4, the alpha component associated with each pixel is taken from the **zaColor** register, but when using format 5 the alpha component associated with each pixel is taken from the 32-bit data transfered. The format of the individual color channels within a 24/32-bit pixel is defined by the rgb channel format field in **lfbMode** bits(12:9).

When writing to the linear frame buffer with a 32-bit access formats 12-14, each pixel is 32-bits, so there are 4096 bytes per logical scan line. Note that when utilizing 32-bit access formats, only a single pixel may be written per 32-bit linear frame buffer write. If depth or alpha information is not transfered with the pixel, then the depth/alpha information is taken from the **zaColor** register. The format of the individual color channels within a 24/32-bit pixel is defined by the rgb channel format field in **lfbMode** bits(12:9). The location of each 16-bit component of formats 12-15 in screen space is defined by bit(11) of **lfbMode**. See the **lfbMode** description in section 5 for more information about linear frame buffer writes.

7.2 Linear frame buffer Reads

When reading from the linear frame buffer, all data returned is in 16-bit format, so there are 2048 bytes per logical scan line. Note that when reading from the linear frame buffer, data is returned in 16/16 format, with two 16-bit pixels returned for every 32-bit doubleword read -- the location of each pixel read packed into the 32-bit host read is defined by bit(11) of **lfbMode**. The RGB channel format of the 16-bit pixels read is defined by the rgb channel format field of **lfbMode** bits(12:9).

It is important to note that reads from the linear frame buffer bypass the PCI host FIFO (as well as the memory FIFO if enabled) but are blocking. If the host FIFO has numerous commands held, then the read will take potentially a very long time before data is returned, as data is not read from the frame buffer until the PCI host FIFO is empty and the graphics pixel pipeline has been flushed. One way to minimize linear frame buffer read latency is to guarantee that the SST-1 graphics engine is idle and the host FIFOs are empty (in the **status** register) before attempting to read from the linear frame buffer.

8. Texture Memory Access

The SST-1 texture memory base address is located at an 8 Mbyte offset from the **memBaseAddr** PCI configuration register and occupies 8 Mbytes of SST-1 address space (see section 4 for an SST-1 address map). Note that the texture memory is write only -- reading from the texture memory address space returns undefined data.

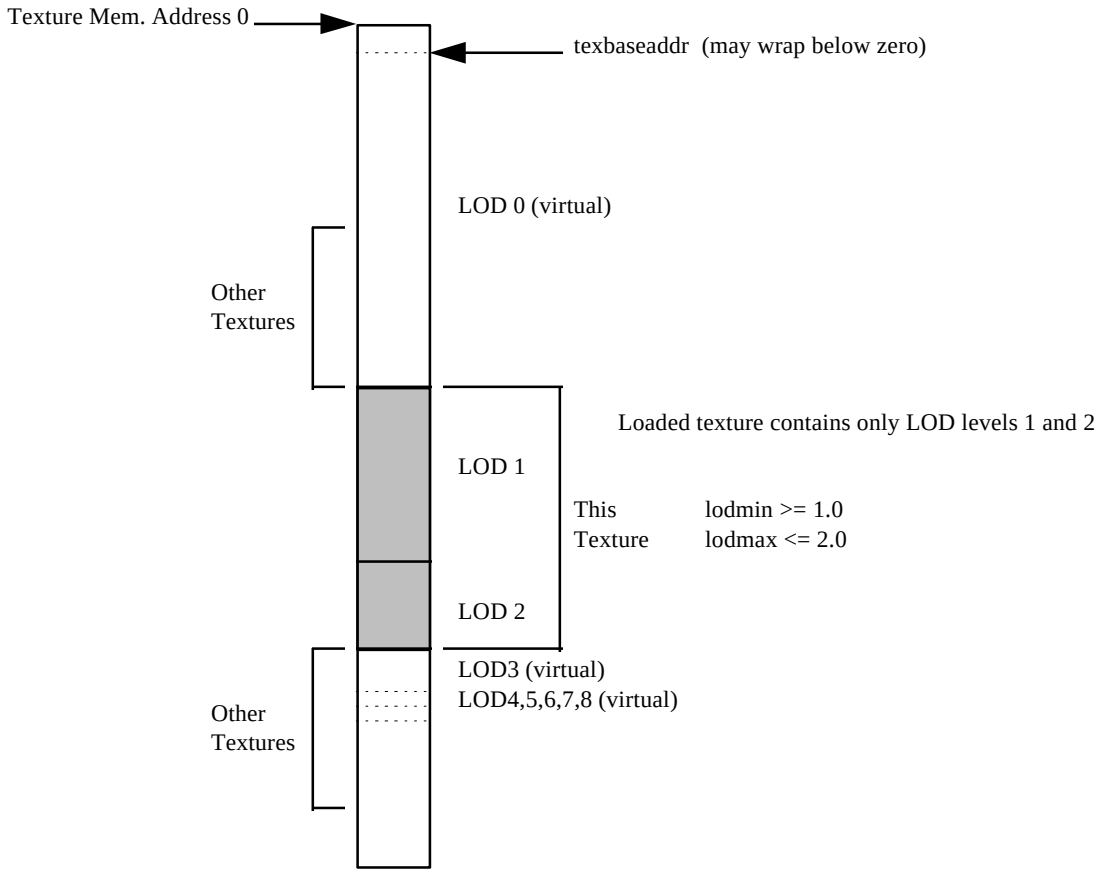
The following section is copied in from the TREX specification. Modifications should be made there and copied over this (initially, trex may sometimes be more current).

Textures are write only. Actual order of write doesn't matter. The texel data can be indirectly read by rendering a texture into the FBI frame buffer, though color dithering alters the values.

Textures are stored as if mipmapped, even for textures containing only one level of detail. The largest texel map (LOD=0) is stored first, and the others are packed contiguously after. *texbaseaddr* points to where the texture would start if it contained LOD level 0 (256x* dimension), in a granularity of 8 bytes. When only some or one of the LOD levels are used, *lodmin* and *lodmax* are used to restrict texture lookup to the levels that were loaded.

texbaseaddr can be set below zero, such that the offset to the texture wraps to a positive number. When two memory banks are used (8 DRAMs), a texture can not span both banks because each bank has one RAS.

Texture Base Address Example



Addresses are generated by adding *texbaseaddr* and an offset that is a function of LOD, S, T, *tclamps*, *tclampt*, *tformat*, *lod_odd*, *lod_tsplit*, *lod_aspect*, *lod_s_is_wider*, *trexinit0*, *trexinit1*. Except for *tclamps* and *tclampt*, all of these values must be valid for texture load.

The size of each level must be known to calculate the *texbaseaddr* and the amount of memory used by the texture. The size can be looked up from a table.

Texture map sizes for 16-bit texel modes, in units of 8 bytes:

		lod_aspect			
LOD	**Size**	**00 1:1**	**01 2:1**	**10 4:1**	**11 8:1**
0	256x*	2^14	2^13	2^12	2^11
1	128x*	2^12	2^11	2^10	2^9
2	64x*	2^10	2^9	2^8	2^7
3	32x*	2^8	2^7	2^6	2^5
4	16x*	2^6	2^5	2^4	2^3
5	8x*	2^4	2^3	2^2	2^2
6	4x*	2^2	2^1	2^1	2^1
7	2x*	1	1	1	1
8	1x*	1	1	1	1

For 8-bit textures, the sizes are half as much as 16-bit. In cases where a half location is used for a level, subsequent levels use the next free half, but a remaining half can not be used as part of the subsequent texture.

In the following examples, sizes and addresses are shown in units of 8 bytes, which is the granularity *texbaseaddr*.

Example 1

16-bit *tformat*, aspect ratio is 1:1, *lod_tsplit* = 0, only LOD levels 1 and 2 are used, start address is 0x00010.

size of level 0 = 2^14 = 0x04000

texbaseaddr = 0x00010 - 0x04000 = 0xfc010

Note that the base wrapped below zero, but lodmin restricts addresses to >= 0x00010.

texture size = size of level 1,2 = 2^12 + 2^10 = 0x01400

next available start address = 0x00010 + 0x01400 = 0x01410

Example 2

8-bit *tformat*, aspect ratio is 8:1, *lod_tsplit* = 0, S is wider, LOD levels 4-8 are used, start address is 0x10000.

size of levels 0,1,2,3 = (2^11 + 2^9 + 2^7 + 2^5) / 2 = 0x00550

texbaseaddr = 0x10000 - 0x00550 = 0x0fab0

texture size = size of levels 4,5,6,7,8 = (2^3 + 2^1 + 1 + 1 + 1) / 2 = 0x00006 + 1/2 -> 0x00007

next available start address = 0x10000 + 0x00007 = 0x10007

Example 3

8-bit *tformat*, aspect ratio is 8:1, *lod_tsplit* = 1, *lod_odd* = 0, S is wider, LOD levels 4-8 are used, start address is 0x10000.

size of levels 0, 2 = (2^11 + 2^7) / 2 = 0x00440

texbaseaddr = 0x10000 - 0x00440 = 0x0fbc0

texture size = size of levels 4,6,8 = (2^3 + 1 + 1) / 2 = 0x00005 + 0/2 -> 0x00005

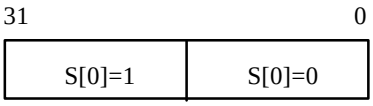
next available start address = 0x10000 + 0x00005 = 0x10005

Texture Load

Two 16-bit or four 8-bit texels are written at a time. For maps that are less than 4 texels wide in the S dimension, the upper texels are inhibited from being written. Only 32-bit accesses are valid, at byte addresses that are a multiple of 4 (2 LSBs are 0).

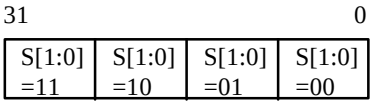
Texture Load Format

16-bit Texture Write Data:



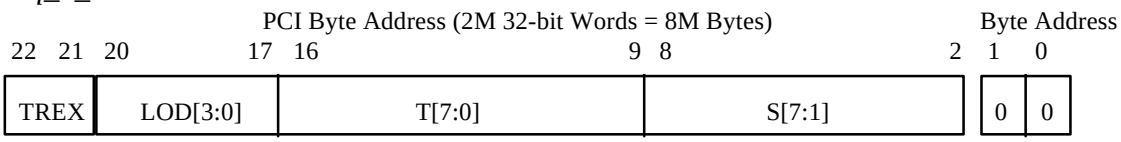
For 1xN textures, write of the upper 2 bytes is inhibited.

8-bit Texture Write Data:



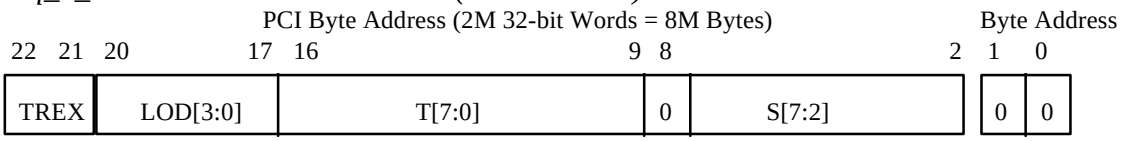
For 2xN textures, write of the upper 2 bytes is inhibited.
For 1xN textures, write of the upper 3 bytes is inhibited.

seq_8_downld==0 or 16-bit texture:



For 8-bit textures, s[1] is set to 0.
For textures smaller than 256x256, S is right aligned to bit 2 and T is right aligned to bit 9. Alignment is the same for 8- and 16-bit textures.

seq_8_downld==1 and 8-bit texture (not revision 0):



For textures smaller than 256x256, S is right aligned to bit 2 and T is right aligned to bit 9.

9. Programming Caveats

The following is a list of programming guidelines which are detailed elsewhere but may have been overlooked or misunderstood:

9.1 I/O Accesses

SST-1 does not support I/O accesses. All I/O accesses to SST-1 are ignored.

9.2 Memory Accesses

All Memory accesses to SST-1 registers must be 32-bit word accesses only. Linear frame buffer accesses may be 32-bit or 16-bit accesses, depending upon the linear frame buffer access format specified in **lfbMode**. Texture memory accesses must be 32-bit word accesses. Byte(8-bit) accesses are not allowed to SST-1 register, linear frame buffer, or texture memory space.

9.3 Determining SST Idle Condition

After certain SST operations, and specifically after linear frame buffer acceses, there exists a potential deadlock condition between internal SST state machines which is manifest when determining if the SST subsystem is idle. To avoid this problem, always issue a NOP command before reading the **status** register when polling on the SST busy bit. Also, to avoid asynchronous boundary conditions when determing the idle status, always read SST inactive in **status** three times. A sample code segment for determining SST idle status is as follows:

```
/**********************************************************
 * SST_IDLE:
 *    returns 0 if SST is not idle
 *    returns 1 if SST is idle
 **********************************************************/
SST_IDLE()
{
  ulong j, i;

  // Make sure SST state machines are idle
  PCI_MEM_WR(NOPCMD, 0x0);
  i = 0;
  while(1) {
    j = PCI_MEM_RD(STATUS);
    if(j & SST_BUSY)
      return(0);
    else
      i++;
    if(i > 3)
      return(1);
  }
}
```

9.4 Triangle Subpixel Correction

Triangle subpixel correction is performed in the on-chip triangle setup unit of SST-1. When subpixel correction is enabled (**fbzColorPath**(26)=1), the incoming starting color, depth, and texture coordinate parameters are all

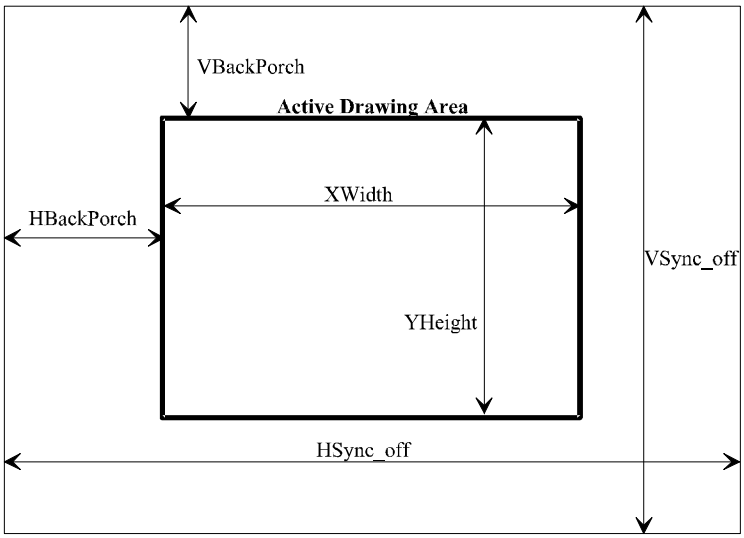
corrected for non-integer aligned starting triangle <x,y> coordinates. The subpixel correction in the triangle setup unit is performed as the starting color, depth, and texture coordinate parameters are read from the PCI FIFO. As a result, the exact data sent from the host CPU is changed to account for subpixel alignments. If a triangle is rendered with subpixel correction enabled, all subsequent triangles must resend starting color, depth, and texture coordinate parameters, otherwise the last triangle's subpixel corrected starting parameters will be subpixel corrected (again!), and inaccuracies will result.

9.5 Loading the internal Color Lookup Table

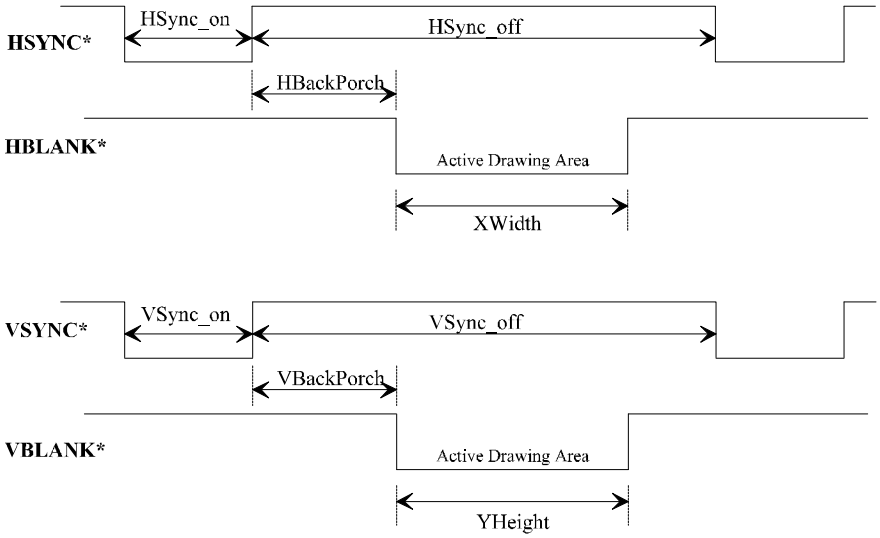
When loading the color lookup table by writing data to **clutData**, the software video reset bit must be disabled (**fbiinit1**(8)=0). If the software video reset bit is enabled (**fbiinit1**(8)=1), the data written to **clutData** will be ignored.

10. Video Timing

SST-1 video timing is defined by the **hSync**, **vSync**, **backPorch**, and **videoDimensions** registers. The following diagram illustrates the video timing parameters of SST-1:



Timing Constraints



The screen resolution is defined in the **videoDimensions** register. Note that regardless of whether SST-1 is in MODE_640 or MODE_800 (see **fbiInit0** register description), smaller monitor resolutions may be programmed in **videoDimension**.

The **hSync** register is used to control the horizontal sync period. The values of **hSync** are specified in VCLK units, which is the video dot clock.

hSync_on = (Number VCLKs of active horizontal Sync) - 1
hSync_off = (Number VCLKs of inactive horizontal Sync) - 1

The **vSync** register is used to control the vertical sync period. The values of **vSync** are specified in horizontal scan line units. The width of a horizontal scan line is defined by the **hSync** register.

vSync_on = (Number horizontal scan lines of active vertical Sync)
vSync_off = (Number horizontal scan lines of inactive vertical Sync)

The area between the left hand side of the monitor and the active video region, known as the horizontal back porch, is defined by the **hBackPorch** field in the **backPorch** register. The register value is specified in VCLK units.

hBackPorch = (Number VCLKs of active horizontal back porch Blank) - 2

The area between the right hand side of the monitor and the active video region, known as the horizontal front porch, is inferred from the horizontal Sync and the horizontal display resolution information. The area between the top of the monitor and the active video region, known as the vertical back porch, is defined by the **vBackPorch** field in the **backPorch** register. The register value is specified in horizontal scan line units.

vBackPorch = (Number Horizontal Scan Lines of active vertical back porch Blank)

The area between the bottom of the monitor and the active video region, known as the vertical front porch, is inferred from the vertical Sync and the vertical display resolution information.

When generating PCI interrupts, the status of the internal **vSync_off** counter is compared to bits(27:16) of the **pciInterrupt** register. Note that the value of the internal **vSync_off** counter may be probed in software by reading the **vRetrace** register.

11. Scanline Interleaving

This section to be completed.
Functions and support which change when scanline interleaving is enabled:

- Polling Status (use **status** and **cfgStatus**)
- Y-Origin bit (must use even yorigin_swapval value if want to swap y-origin)
- Linear frame buffer reads (must change who controls PCI bus)
- Setting up Scanline interleaving
- Flushing PCI Packer (use read from **status** register)

12. SST-1 Revision 2.0 Changes

The following describes the silicon differences between the silicon revision 1.0 chipset and revision 2.0:

FBI (revision 2.0, second silicon):

- Updated PCI configuration register field ***revision_id*** to 0x2
- Split meaning of reset strapping pin **fb_addr_a**[5]. **fb_addr_a**[5] is now used to hardcode the PCI address to 0x1f000000 (1=enable), and **fb_addr_a**[4] is used to specify the reset/default **vga_pass** output value
- **fb_addr_a**[8] configures PCI configuration **status** register to specify fast **pci_devsel** timing and specifies capability of handling fast back-to-back PCI requests (1=enable fast PCI timing)
- Added fix for flow control problem (FBI drops texels) when texture mapping
- Added proper decoding of PCI configuration cycles
- Added proper driving of **pci_fifo_stall** signal
- Added proper swap interval determination
- Added capability to read **lfbMode**[16:15]
- Added better busy detection for pixel_pack module
- Changed **dac_hsync** and **dac_vsync** output drivers to 12 mA
- Changed **tf_stall** output driver to 8 mA
- Changed **fb_cas**[3:0] output drivers to 8 mA
- TREX busy bit in **status** is cleared if **fbiinit3**[6]=1 (disables texture mapping functionality)
- Added capability to always guarantee writes to TREX occur even if **ft_stall** is asserted
- Added pseudo-stencil capability for cockpits (**fbzMode**[20])

TREX (revision 1.0, second silicon):

- Changed revision number to 0x1
- Added proper software reset functionality (trexInit1: *reset_FIFOs, reset_graphics*)
- Changed tex_ras[1:0], tex_we, and tex_cas[3:0] outputs from 4mA to 8mA drive
- Fixed performance problem when texturing (generated page miss every 13-14 cycles)
- Added support for 8-bit paletted textures, AP and P tformat's (write timing adjust trexInit1: *palette_del*)
- Added fast download capability for 8-bit textures (textureMode: *seq_8_downld*)
- Added two-memory support (trexInit0: *mem2*)
- Added more config. sense modes, backward compatible with rev. 0 (see trex spec., trexInit1)

13. Revision History

0.7 Updated to version 1.1a of sst.h

0.8 Added proper SST-1 address space
Added Spatial and parameter iterators for new edge walking algorithm
Updated to latest (4/2/95) version of sst.h

0.9 Added further descriptions of TREX-specific registers
Added SST-1 1st Silicon Functionality section
Modified system level diagrams to show 1-8 Mbytes of TREX memory support

0.95 Numerous miscellaneous typos and changes
Added byte swizzling bit(12) in **lfbMode**
Added triangle area formula in **triangleCMD** register description
Writing '1' to lsb of **nopCMD** now clears the **fbiPixelsIn**, **fbiChromaFail**, **fbiZfuncFail**, **fbiAfuncFail**, and **fbiPixelsOut** registers
Added **clutData**, **dacData**, **fbiInit2**, and **fbiInit3** registers
Renamed **clipTopBottom** to **clipLowYHighY**
Changed encoding values for **chip** field of SST-1 address space

1.00 PCI write wait states now only 0 or 1 (specified in **fbiinit1**)

1.10 Moved enabled alpha-planes bit to **fbzMode**
Miscellaneous changes to the **fbiinit** registers
Updated Depth-buffering diagram

1.20 Updated **tDetail** with bias size 6.0 signed
Changed PCI configuration register name **initWrEnable** to **initEnable**
Added PCI configuration registers **busSnoop0** and **busSnoop1**, and added PCI snooping bits in **initEnable**
Removed **pciInterrupt** register
Added **fbiInit4** register
Added description of **dacData** register
Revised depth generation diagram
Added description of idiosyncrasies of subpixel correction in the triangle setup unit
Added stipple pattern mode
Removed 1st silicon spin functionality section

1.30 Added cc_localselect_override bit in **fbzColorPath**
Added **cfgStatus** configuration register
Fixed typo in definition of **dacData** register for specifying read/write command
Removed definition in **trexInit** registers. Temporarily in TREX spec only.
Updated ACU diagram to show alpha-mask test.
Fbi-to-trex bus clock delay default now 0x2.
Added dither subtraction for alpha-blending (**fbzMode**(19))
Moved PCI bus owner for scanline interleaving to Configuration **initEnable**(10)
Added PCI read wait states and linear frame buffer readahead in **fbiInit4**
Added separate control bits in **lfbMode** for word swapping and byte swizzling for lfb reads
Removed TREX initialization bits from **fbiInit3**
Added section on scanline interleaving support
Added triangle parameter address remapping support (**fbiinit3**(0)=1)

1.40 Changed triangle parameter address remapping to RGBZASTW order
Added Vendor ID 0x121a in PCI configuration register space
Added maxRgbDelta register
Moved scan_interleave_slv to Pci configuration register **initEnable**

Added disable texture mapping bit in **fbiinit3**(6)
Added memory FIFO configuration bits in **fbiinit4**
Changed performance charts

1.50 Fixed typos in PCI register address map and added hex addresses
Added performance estimates
Added more SST-1 revision 2.0 changes

1.60 Added *seq_8_downld* functionality for TREX revision 1.0 in **textureMode**
Added 8-bit palette format descriptions in **textureMode** description
Added 8-bit palette download description in **NCC** description

?? Added details to trex rev. 1 changes
seq_8_downld must equal 0 for direct writes.