SST-96 (a.k.a. Voodoo Rush™)

VOODOO GRAPHICS
FOR
WINDOWS

Revision 2.2
November 30, 1999

3Dfx Interactive, Inc.
4435 Fortran Drive
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.]

Proprietary Information

1. Introduction

The SST-96 graphics subsystem combines the high performance graphics of 3Dfx Interactive (3D) with the proven performance of an industry standard windows accelerator (2D). This union creates an extremely cost effective and uncompromising multimedia solution. The 2D engine provides a PCI system interface, VGA, 2D and Video features and the 3D engine adds windowed 3Dfx "**Voodoo Graphics**".

SST-96 connects the 2D and 3D graphics engines through the "Voodoo Rush" (VR) interface. The VR interface consist of a shared frame buffer memory interface and a 3D control interface. The shared frame buffer interface defines a Pseudo Unified Memory Architecture (PUMA) which connects the 2D/3D devices through the pins of the frame buffer memory and the 3D control interface defines signals which coordinate efficient 2D/3D operation.

The SST-96 PUMA interface supports shared access to the frame buffer memory through a request and grant arbitration protocol based on VUMA (VESA Unified Memory Arbitration). PUMA defines the 2D device as the requester and the 3D device as the grantor. As the requester, the 2D device may generate low priority requests for general frame buffer accesses and high priority requests for video/memory refresh accesses.

The SST-96 3D control interface provides for an efficient 2D/3D operation through swap, vsync and status connections. The swap/vsync connections provide for a polling-free 3D-render/2D-refresh buffer swap mechanism and the status connection provides 3D status to the PCI though the 2D chip.

1.1 System Diagram

Error! Not a valid link.

1.2 System Features

Interface

- Low-cost industry-standard 2D/VGA Windows Accelerator.
- High performance “Voodoo Graphics” quality 3Dfx Accelerator.
- Minimum interface overhead:
  3-pin Pseudo Unified Memory Architecture (REQ, GNT & CLK),
  2-Pin buffer management (SWAP & VSYNC),
  1-Pin system status (STATUS).

2D

- PUMA client interface.
- 2 or 4 MB linearly mapped frame buffer memory
- VGA compliance.
- BLTs & ROPs.
- Draw & fill.
- Scrolling, clipping & scaling.
- Video In & Video Out.
- Optional MPEG attachment.
- Video refresh.
- DRAM refresh.

3D

- PUMA core interface.
- 1 to 4 MB of texture memory.
- Triangle raster engine with sub-pixel correction to .4 x .4 resolution.
- Linearly interpolated Gouraud-shaded rendering.
- Perspective-corrected texture-mapped rendering with iterated RGB modulation/addition.
- Texture filtering: point-sampling, bilinear, and tri-linear filtering with mip-mapping.
- 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.
- Transparency with dedicated color mask.
- Source/Destination pixel alpha blending.
- Linearly interpolated 16-bit Z-buffer rendering.
- Perspective-corrected 16-bit floating point W-buffer rendering.
- 24-bit color dithering to native 16-bit RGB buffer using 4x4 or 2x2 ordered dither matrix.
- Non-linear table driven fog.

1.3 System Resolutions

SST-96 supports any resolution which fits in the frame buffer memory, is less than 2K by 2K in X/Y rendering dimension and is supported by the refresh engine of the 2D chip. Full-screen 3D requires the memory space of the front/back 3D buffers, while windowed 3D requires the additional memory space of the 2D desktop. Windowed 3D overlays 3D data onto the desktop from off-screen buffers utilizing either two off-screen buffers (out-of-place 3D) or one on-screen buffer and one off-screen buffer (in-place 3D -- requires equal 2D and 3D color depths).

The following tables detail the memory byte requirements for full-Screen and windowed resolutions. Numbers indicated in light gray require a 4MB frame buffer and numbers indicated in dark gray do not fit within SST-96.

Full Screen 3D

3D Resolution	3D Color Depth	Auxiliary Depth	Color Buffer 1	Color Buffer 2	Auxiliary Buffer	Total of Buffers	CMDFIFO Space
16-bit Full Screen 3D							
640x480	16	0	614400	614400	0	1228800	868352
	16	8	614400	614400	307200	1536000	561152
	16	16	614400	614400	614400	1843200	253952
800x600	16	0	960000	960000	0	1920000	177152
	16	8	960000	960000	480000	2400000	1794304
	16	16	960000	960000	960000	2880000	1314304
1024x768	16	0	1572864	1572864	0	3145728	1048576
	16	8	1572864	1572864	786432	3932160	262144
	16	16	1572864	1572864	1572864	4718592	(524288)
1152x864	16	0	1990656	1990656	0	3981312	212992
	16	8	1990656	1990656	995328	4976640	(782336)
	16	16	1990656	1990656	1990656	5971968	(1777664)
1280x1024	16	0	2621440	2621440	0	5242880	(1048576)
	16	8	2621440	2621440	1310720	6553600	(2359296)
	16	16	2621440	2621440	2621440	7864320	(3670016)
1600x1200	16	0	3840000	3840000	0	7680000	(3485696)
	16	8	3840000	3840000	1920000	9600000	(5405696)
	16	16	3840000	3840000	3840000	11520000	(7325696)
8-bit Full Screen 3D							
640x480	8	0	307200	307200	0	614400	1482752
	8	8	307200	307200	307200	921600	1175552
	8	16	307200	307200	614400	1228800	868352
800x600	8	0	480000	480000	0	960000	1137152
	8	8	480000	480000	480000	1440000	657152
	8	16	480000	480000	960000	1920000	177152
1024x768	8	0	786432	786432	0	1572864	524288
	8	8	786432	786432	786432	2359296	1835008
	8	16	786432	786432	1572864	3145728	1048576
1152x864	8	0	995328	995328	0	1990656	106496
	8	8	995328	995328	995328	2985984	1208320
	8	16	995328	995328	1990656	3981312	212992
1280x1024	8	0	1310720	1310720	0	2621440	1572864
	8	8	1310720	1310720	1310720	3932160	262144
	8	16	1310720	1310720	2621440	5242880	(1048576)
1600x1200	8	0	1920000	1920000	0	3840000	354304
	8	8	1920000	1920000	1920000	5760000	(1565696)
	8	16	1920000	1920000	3840000	7680000	(3485696)

Windowed 3D

2D & 3D Resolution	3D Color Depth	Auxiliary Depth	Color Buffer 1	Color Buffer 2	Auxiliary Buffer	Total of Buffers	CMDFIFO Space
8-bit Desktop and 16-bit 640x480 Out-of-Place 3D (two off-screen 3D buffers)							
1024x768	0	0	0	0	0	786432	na
640x480	16	0	614400	614400	0	2105232	81920
	16	8	614400	614400	307200	2322432	1871872
	16	16	614400	614400	614400	2629632	1564672
1152x864	0	0	0	0	0	995328	na
640x480	16	0	614400	614400	0	2224128	1970176
	16	8	614400	614400	307200	2531328	1662976
	16	16	614400	614400	614400	2838528	1355776
1280x1024	0	0	0	0	0	1310720	na
640x480	16	0	614400	614400	0	2539520	1571168
	16	8	614400	614400	307200	2846720	1263968
	16	16	614400	614400	614400	3153920	956768
1600x1200	0	0	0	0	0	192000	na
640x480	16	0	614400	614400	0	3148800	1045504
	16	8	614400	614400	307200	3456000	736403
	16	16	614400	614400	614400	3763200	431104
16-bit Desktop and 16-bit 640x480 In-Place 3D (one on-screen 3D buffer & one off-screen 3D buffer)							
1024x768	0	0	0	0	0	1572864	na
640x480	16	0	0	614400	0	2187264	2007040
	16	8	0	614400	307200	2492264	1702040
	16	16	0	614400	614400	2799464	1394840
1152x864	0	0	0	0	0	1990656	na
640x480	16	0	0	614400	0	2605056	1896448
	16	8	0	614400	307200	2912256	1282048
	16	16	0	614400	614400	3219456	974848
1280x1024	0	0	0	0	0	2621440	na
640x480	16	0	0	614400	0	3235840	958464
	16	8	0	614400	307200	3543040	651264
	16	16	0	614400	614400	3850240	344064
8-bit Desktop and 16-bit 320x200 Out-of-Place 3D (two off-screen 3D buffers)							
1024x768	0	0	0	0	0	786432	na
320x200	16	0	128000	128000	0	1042432	1054720
	16	8	128000	128000	64000	1106432	990720
	16	16	12800	128000	128000	1170432	926720
1152x864	0	0	0	0	0	995328	na
320x200	16	0	128000	128000	0	1251328	845824
	16	8	128000	128000	64000	1315328	781824
	16	16	128000	128000	128000	1379328	717824
1280x1024	0	0	0	0	0	1310720	na
320x200	16	0	128000	128000	0	1566720	530432
	16	8	128000	128000	64000	1630720	466432
	16	16	128000	128000	128000	1694720	402432
1600x1200	0	0	0	0	0	1920000	na
320x200	16	0	128000	128000	0	2176000	2018304
	16	8	128000	128000	64000	2240000	1954304
	16	16	128000	128000	128000	2304000	1890304

1.4 System Performance

The performance of SST-96 approximates the performance of SST-1. However, modification to the memory architecture resulting from the 2D engine partnership alter the command and rendering interfaces with a slight degradation in performance. Changes come from the additional command traffic across the PUMA frame buffer interface and modifications to the frame buffer memory packing and tiling organizations.

The following table estimates the performance of SST-1 and SST-96 from an architectural simulator. The numbers represent systems rendering depth checked color and no alpha blend at 50MHz system. The simulator assumptions include a never-idle triangle engine (best-case) and full triangle parameter delivery (120 bytes) for each triangle (worst-case) and 15% overhead for memory/display refresh. Additional assumptions include, an efficient command transport from the 2D device (8x64 bits in 11 cycles) and an efficient UMA arbitration overhead (1 dead cycle).

Pixels Per Triangle / CZ Rendering	12 Ktris/Mpixs		24 Ktris/Mpixs		49 Ktris/Mpixs		112 Ktris/Mpixs		200 Ktris/Mpixs		511 Ktris/Mpixs	
SST-1 (direct)	996	12	894	21	533	26	266	29	158	31	65	33
SST-1 (memory fifo)	626	7	501	12	359	17	214	24	138	27	62	31
SST-96 (memory fifo)	511	6	414	10	293	14	179	20	119	23	56	28

Pixels Per Triangle / Rendering Mode	10 Ktri/Mpix		25 Ktri/Mpix		50 Ktri/Mpix		100 Ktri/Mpix		500 Ktri/Mpix		1000 Ktri/Mpix		5000 Ktri/Mpix	
C - color only	787	7	694	17	586	29	409	40	83	41	41	41	8	41
CA - color & blend	580	5	466	11	339	16	232	23	70	35	39	39	8	41
CZ - color & depth	557	5	439	10	307	15	204	20	58	29	31	31	6	34
CAZ - blend & depth	444	4	335	8	335	11	143	14	38	19	21	21	4	22

The SST-96 performance loss at the rendering interface is a result of changes to the packing and tiling organization of pixels within frame buffer memory. SST-96 packs pixel quads of color or depth onto separate mapped memory pages while SST-1 packs pixel pairs of color and depth within a memory word. Additionally, SST-96 tiles pixels of color/depth onto linear strides of memory while SST-1 tiles pixels of color/depth onto rectangular strides of memory. These memory organization differences result in less access efficiency for small polygons in, more page miss penalties for small polygons and more page misses between color/depth accesses.

pixels/triangle	12	24	49	112	200	511
KT Packing Loss	37%	24%	17%	11%	9%	4%
KT Tiling Loss	23%	19%	17%	15%	12%	6%

2. Memory Map

SST-96 requires 4 or 8 Mbytes of PCI address space. The 8MB space consists of a 4 MB region for the frame buffer, a 2 MB region for the 3D control registers and a 2 MB region for the 3D texture memory. Alternatively, a 4MB space consists of a 2 MB region for the frame buffer, a 1 MB region for the 3D control registers and a 1 MB region for the 3D texture memory. The following table defines the ordering of these regions within the space of SST-96.

Address Table

Address	Description
0x000000-0x3fffff	SST-96 frame buffer memory (4 MB)
0x400000-0x5fffff	SST-96 3D control register (2 MB)
0x600000-0x7fffff	SST-96 3D texture memory (2 MB)
<or>	
0x000000-0x1fffff	SST-96 frame buffer memory (2 MB)
0x200000-0x2fffff	SST-96 3D control register (1 MB)
0x300000-0x3fffff	SST-96 3D texture memory (1 MB)

The three regions of the PCI address space define the shared frame buffer memory space of the PUMA interface. Accesses across this interface are performed as EDO DRAM read or write operations. As a result, SST-96 provides page mapped read access of the 3D control registers but does not provide read access of texture memory.

The following three tables describe the address maps supported by SST-96. The first map is the preferred map of SST-96, however, the second and third maps may be configured through external control on the 3D device. For each configuration, the address map indicates the PUMA memory signals associated with the PUMA address space.

8MB Address Map

PUMA DRAM Signals		DATA[63:0] CAS[7:0] (byte enables)	
RAS0	WE0 OE0	2MB	**Frame Buffer Dram** Linear read/write access, rendering and refresh.
	WE1 OE1	2MB	
RAS1	WE0 OE0	2MB	**3D Registers** Double word aligned write access. Page aligned read access.
	WE1 OE1	2MB	**3D Texture Memory** Write Only.

3. Command FIFO

A 3D control command FIFO (CMDFIFO) may be established by software within the DRAM of the frame buffer memory. Initialization registers of FBIjr define the CMDFIFO as a circular space of 1 to 512 4KB pages. Once established, software loads commands into the FIFO through linear frame buffer writes and the 3D engine seeks commands out of the FIFO through the PUMA interface.

Software must write the CMDFIFO through consecutive 32-bit LFB addresses filling full 64-bit quad words.

FIFO Management

Hardware manages the CMDFIFO emptiness. The 3D engine maintains a read pointer and an entries count for the CMDFIFO. Writes to the CMDFIFO increment the entries count and reads from the CMDFIFO decrement the entries count while incrementing the read pointer. Then, as the entry count is greater than zero, the 3D engine arbitrates for PUMA access and reads from the CMDFIFO.

Software manages CMDFIFO fullness. To assist, hardware provides read access of the CMDFIFO entries count and a fullness status. From the entries count and knowledge of the CMDFIFO size, software may determine the exact fullness. Alternatively, the status registers within the 2D and 3D chips indicate a fullness measurement based on the comparison to a programmable fullness count. Software is encouraged to manage fullness through read of the 2D serial status as access to the 2D chip does not tax the frame buffer bandwidth.

FIFO Data

The CMDFIFO data controls the 3D engine through non-grouped and grouped write packets. Each packet type contains an even number of 32-bit 8-byte aligned data words which command write operations to PUMA 4-byte word addresses. Non-grouped write packets modify a single address with a single 32-bit data word and grouped-write packets modify up to 32 addresses with as many pieces of 32-bit data words.

The non-grouped write packet consists of an address in the lower word and data in the upper word. Similarly, the grouped write packet begins with an address in the lower double-word but follows the address with a mask control word and up to 32 32-bit data words.

The address of the grouped write format sets the base address of the write and the mask determines the sequence and volume of data to be written. From LSB to MSB of the mask, a “1” enables the write and a “0” disables the write. The sequence of 32-bit data words following the address and mask modify addresses equaling the base address plus N when the mask bit N equals “1” as N goes from 0 to 31.

Non-grouped Write Packet

- 2 64-bit aligned 32-bit words.

63 32	31		0
Data[31:0] (written to Address)	0	rsvd[9:0]	Address[20:0] (PCI address [22:2] of register)

Example: write to triangleCMD register.

63 32	31		0
<triangleCMD_data>	0	0000000000	0x10_0020

Writes PUMA address = 0x10_0020 with data = <triangleCMD_data>

Grouped Write Packet

- 2 to 34 even numbered 32-bit words.

63 32	31 0		
Mask[31:0] (Mask[N]=1 enables write to BaseAddress+N)	1	rsvd[9:0]	BaseAddress[20:0] (PCI address [22:2] of first register)
32-bit data words equaling the count of Mask[N]=1 as N goes from 0 to 31 (0 to 31 32-bit data words)			

Example: set triangle vertex, RGBW start, dX and dY parameters, and triangleCMD registers (19 registers).

63 32	31 0		
0x3803_ffff	1	0000000000	0x10_0002
<vertexAy_data>	<vertexAx_data>		
<vertexBy_data>	<vertexBx_data>		
<vertexCy_data>	<vertexCx_data>		
<dRdX_data>	<startR_data>		
<startG_data>	<dRdY_data>		
<dGdY_data>	<dGdX_data>		
<dBdX_data>	<startB_data>		
<startW_data>	<dBdY_data>		
<dWdY_data>	<dWdX_data>		
0xXXXX_XXXX (filler data must be written)	<triangleCMD_data>		

1. Writes PUMA address = 0x10_0002 with data = <vertexAx_data>
2. Writes PUMA address = 0x10_0003 with data = <vertexAy_data>
3. Writes PUMA address = 0x10_0004 with data = <vertexBx_data>
4. Writes PUMA address = 0x10_0005 with data = <vertexBy_data>
5. Writes PUMA address = 0x10_0006 with data = <vertexCx_data>
6. Writes PUMA address = 0x10_0007 with data = <vertexCy_data>
7. Writes PUMA address = 0x10_0008 with data = <startR_data>
8. Writes PUMA address = 0x10_0009 with data = <dRdX_data>
9. Writes PUMA address = 0x10_000a with data = <dRdY_data>
10. Writes PUMA address = 0x10_000b with data = <startG_data>
11. Writes PUMA address = 0x10_000c with data = <dGdX_data>
12. Writes PUMA address = 0x10_000d with data = <dGdY_data>
13. Writes PUMA address = 0x10_000e with data = <startB_data>
14. Writes PUMA address = 0x10_000f with data = <dBdX_data>
15. Writes PUMA address = 0x10_0010 with data = <dBdY_data>
16. Writes PUMA address = 0x10_001d with data = <startW_data>
17. Writes PUMA address = 0x10_001e with data = <dWdX_data>
18. Writes PUMA address = 0x10_001f with data = <dWdY_data>
19. Writes PUMA address = 0x10_0020 with data = <trangleCMD_data>

4. 3D Registers

SST-96 defines 256 3D control registers mapped onto 1MB of the PUMA address space. Within this space, write addresses pack the registers within the first DRAM page and read addresses spread the registers onto separate DRAM pages. This arrangement allows for efficient bursts when writing registers and DRAM emulation when reading registers. The following table defines the address sub-fields mapping the registers for read and write access.

Type	PCI Address 19:0 (1MB)				
write address	**19:16**	**15:12**	**11:10**	**9:2**	**1:0**
	chip	**wrap**	**00**	**register**	**byte**
read address	**19:12**		**11:2**		**1:0**
	register		**X**		**byte**

The **register** field selects 1 of 256 3D registers defined by the register table of the 3D registers map section.

The **chip** field selects one or more of the SST-96 3D chips (FBI, TREX0, TREX1 or TREX2) to be accessed. Each bit in this field selects a chip for writing. The LSB selects the FBI and the higher order bits select the higher order TREX chips. The value of "0000" also selects all chips. Software controls write data presented to the individual chips through this field. Reads ignore this field as data is always read from FBI. The following table defines the mapping of this field:

Chip Field	SST-96 Chip Selected			
chip[3:0]	TREX2	TREX1	TREX0	FBI
0000	Yes	Yes	Yes	Yes
0001	No	No	No	Yes
0010	No	No	Yes	No
0011	No	No	Yes	Yes
0100	No	Yes	No	No
0101	No	Yes	No	Yes
0110	No	Yes	Yes	No
0111	No	Yes	Yes	Yes
1000	Yes	No	No	No
1001	Yes	No	No	Yes
1010	Yes	No	Yes	No
1011	Yes	No	Yes	Yes
1100	Yes	Yes	No	No
1101	Yes	Yes	No	Yes
1110	Yes	Yes	Yes	No
1111	Yes	Yes	Yes	Yes

The **wrap** field aliases the memory mapped registers to 16 locations. This field allows software to avoid write compression within processors such as the Digital Alpha AXP and the Pentium Pro. These processors contain large write-buffers that collapse multiple writes to the same address.

The **byte** field defines the byte alignment of the register access. This field must be 0x0 for all register accesses. Register accesses are restricted to 4-byte word aligned 32-bit accesses. No byte (8-bit) or half word (16-bit) accesses are allowed. To modify individual bits of a 32-bit register, the entire 32-bit word must be written with valid bits in all positions.

5. Texture Memory

SST-96 defines 2MB of PUMA address space for loading texture memory. Through this space, software may load from 1 to 8 MB of texture memory. Software loads texture memory by setting up the texture write before writing the texture through the PUMA interface. Software sets the texture base address register and various other texture control registers which define the texture packing scheme. Then, software writes the S and T data for each LOD of the texture.

The 21-bit PUMA address of the 2MB PUMA texture space defines the LOD, S and T of the texture through the mapping defined in the following table (S[1] is set to 0 for 8-bit textures).

PCI Address[20:0]	20 17	16 9	8 2	1 0
Texture Fields	LOD[3:0]	T[7:0]	S[7:1]	00

The 32-bit PUMA data word defines the data of the LSBs of S, for 16-bit and 8-bit textures, through the mapping defined in the following table.

PCI Data[31:0]	31 24	23 16	15 8	7 0
16-bit Texture	S[0]=1		S[0]=0	
8-bit Texture	S[1:0]=11	S[1:0]=10	S[1:0]=01	S[1:0]=00

Texture loads of LODs smaller than the extent of S and T ignore the unused MSBs of S and T within the PUMA address and data. The write of a single texel wide 16-bit texture inhibits the write to the upper two bytes of data. The write of a double texel wide 8-bit texel inhibits the write to the upper two bytes of data. Additionally, the write of a single texel wide 8-bit texel inhibits the write to the upper three bytes of data.

The following TREX registers, defining the data packing of the texture, must be set prior to writing a texture.

texBaseAddr - These registers determine the 8 byte aligned base address of the texture within the 1-8MB space of texture memory. Addressing from the base address assumes that the texture uses all LOD levels. LOD0 is stored first and each of the higher numbered LODs are stored contiguously after.

texureMode.tformat - This register field defines the format of the texels. From the standpoint of texture load, this register defines the volume of bytes per texel.

tLOD.lod_split - This register field defines split textures. Split textures load only the relevant even or odd textures as defined by the **tLOD.lod_odd** register field.

tLOD.lod_aspect - This register field defines the 2**N aspect ratio of the texture. Valid values range from 0b00 to 0b11. These values set the following aspect ratios: 00=1to1, 01=2to1, 10=4to1 & 11=8to1.

Efficient packing of textures, with a minimum LOD number greater than 0, requires software mapping of the base address which assumes the low numbered LODs are written. Software must set the base address below the intended write address to account for the space of the unused LODs. The space of an LOD can be determined from the bytes/texel, texels/LOD and LODs/texture (small LODs consume a minimum of 4 bytes (2-3 unused bytes)).

6. Render and Refresh

The SST-96 architecture presents two synchronization issues resulting from the PUMA connection and the shared duties of the 2D and 3D chips. One problem occurs as order-of-evaluation may be lost when the 2D and 3D engines simultaneously render into the memory of the PUMA interface and a second problem occurs as the 2D engine performs display refresh of images rendered by the 2D and 3D engines.

RENDER

Command ordering may become problematic as 3D commands are tossed across the PUMA interface to the 3D device. Within the SST-96 architecture, 2D accesses may over step 3D activities and 3D direct accesses may over step 3D CMD FIFO activities. To ensure proper order of evaluation, software must coordinate these activities through polling for 3D engine idleness and 3D command FIFO emptiness. 3D direct accesses must poll for 3D CMD FIFO emptiness and 2D accesses (BLTs, LFBs & ROPs) must additionally poll for 3D CMD FIFO emptiness.

Windowed Rendering

All rendering, clipping and fast-fill activities of the 3D engine are in window relative coordinates. The hardware translates window relative X and Y drawing coordinates to Linear Frame Buffer (LFB) addresses through the **colBufferSetup** and **AuxBufferSetup** registers.

The color and auxiliary buffer setup registers define separate base address and X stride values for each of the buffers. The base address field defines the LFB base address of the buffer and the X stride field defines the number of pixels between rows of the 3D data. The X strides are restricted to strides which is evenly divisible by 4 pixels and the base addresses must be set with the same quad pixel alignment between the color and auxiliary buffers.

Determination of the LFB address depends on the state of the Y-flip mode (**fbzMode**[17]). For the non-flipped case, X stride is added to the base address for each window Y address and, for the flipped case, X stride is subtracted from the base address for each window Y address. The following diagram illustrates the calculation of LFB address for the non-flipped and flipped window origin rendering modes.

Error! Not a valid link.

REFRESH

Swap coordination within SST-96 requires a swap command, a swap pending increment command and commands to initialize the next buffer. The following command sequence illustrates this process.

1. Rendering Commands
2. Swap Pending Increment Command
3. Swap Command
4. Set Next Buffer Pointers Registers
5. Fast Fill Buffer Commands
6. Repeat

Tiling or BLTing

SST-96 swaps buffers rendered by the 3D engine and refreshed by the 2D engine through “tiling” or “BL

Tile swapping occurs automatically through hardware. Software sets up a mapping of rectangular regions of the 3D image onto rectangular regions of the display and the hardware synchronizes swaps to VSYNC. Tile data is overlaid onto the display as a back-end process during refresh. No additional frame buffer bandwidth is incurred over regular refresh. Other advantages include bilinear magnification and source-to-destination format conversion (e.g. 16-bit 3D-RGB to 8-bit 2D-RGB).

BLTing occurs through software. Software must poll after every swap command before BLTing data between buffers. The BLT adds two frame buffer accesses per pixel. An additional disadvantage exists as the data formats of the 2D and 3D engines must be equivalent.

Alliance provides 12 tiles. In general, each tile maps data of onto the display during refresh. Multi-buffering requires the assignment of tiles to frames. Double buffering assigns tiles to frames A and B, triple buffering assigns tiles to frames A, B and C and stereo buffering assigns tiles to frames A, B, C and D. Full screen 3D requires ONE tile per frame, 3D in a window requires TWO tiles per frame and 3D in a window with a simple pull down menu requires FOUR tiles per frame (can't be done in stereo quad buffering). More complicated applications require a greater number of tiles per frame and thus limit the number of frames which may be supported. For more tiling information see the Video Window documentation on the 2D chip from Alliance.

Double and Triple Buffering

Double buffered monaural 3D requires a 2D engine which displays the front buffer as the 3D engine renders into the back buffer. As the 3D engine completes rendering the current back buffer, the 3D engine initiates a swap request and suspends execution until the request is fulfilled. Then, with the next VSYNC (exceeding the minimum VSYNC count), the 2D chip swaps (tiles) in the new front buffer allowing the 3D engine to render into the new back buffer.

Triple buffered monaural 3D requires a 2D engine which displays the front buffer as the 3D engine renders into two back buffers. As the 3D engine completes rendering the first back buffer, the 3D engine initiates a swap request and continues rendering into the second back buffer. Meanwhile, with every VSYNC (exceeding the minimum VSYNC count), the 2D chip swaps (tiles) in the new front buffer allowing the 3D engine to render into another back buffer.

Error! Not a valid link.

Stereo Buffering

Double buffered stereo requires a 2D engine which displays the left and right images at the frame rate as the 3D engine renders the opposite images within the frame rate. This configuration precludes 3D rendering which may fall behind the refresh rate of the 2D engine.

Quadruple buffered stereo requires a 2D engine which displays the left and right images at the frame rate as the 3D engine renders the next left and right images. Software renders left and right image pairs with a singular swap command. Each swap command switches in a new set of left and right image pairs at even numbered VSYNC pulses. If the 3D engine gets behind the 3D engine, the 3D engine continues rendering as the 2D engine refreshes the display with the previous left and right images.

Error! Not a valid link.

7. 3D Register Map

The following register map defines the 3D control registers of SST-96. The map describes the **name**, the **address** and the number of **bits** of each of the 3D control registers. The address indicates the 8-bit register number for both read and write addresses. For read accesses, the address of this table reflects bits 9:2 of the PCI address and for write accesses the address of reflects bits 19:12 of the PCI address.

In addition, the register map includes the following **chip**, **RW** and **Sync/Fifo** columns.

- The **chip** column indicates the chips that the registers are stored in (F = FBIjr & T = TREX). A "%" indicates that the registers are unconditionally written to the chips of the chip field regardless of the chip address.
- The **RW** column indicates the read and write capability of individual registers. Reading from a register which is "write only" returns undefined data. Writing a register which is "read only" does nothing.
- The **sync** column identifies registers which may cause the graphics processor to stall. A "yes" indicates that the graphics processor will wait for a flush of the data pipeline before loading the register. Loading these registers results in a small performance degradation when compared to registers which do not need synchronization.
- The **FIFO** column identifies registers whose writes will be pushed into the MEM FIFO. Care must be taken when writing to those registers not pushed into the FIFO. Writes to these registers may complete out-of-order relative to pending writes which have been FIFOed. Also, reads are never FIFOed, therefore reading FIFOed registers will return the current value of the register, irrespective of writes to the register pending in the FIFO.

The register map defines the registers of SST-96 while highlighting modifications from SST-1. Registers which have been modified or added since SST-1 are highlighted in light gray. Additionally, unlike SST-1, the SST-96 map defines only one mapping for the triangle parameter registers. SST-96 defines the triangle parameter registers (start, delta-x and delta-y) grouped by parameter type (R, G, B, A, Z, S and T) only. This grouping was the alternate grouping of SST-1.

Register Table

Name	Address w:[9:2] r:[19:12]	Bits	Chip	RW	Sync? /Fifo?	Description	p#
status	0x000	15:0	F	R	na	Status Register	23
reserved	0x001	na				19	
vertexAx	0x002	15:0	F+T‰	W	N/Y	Vertex A x-coordinate location (12.4 format)	24
vertexAy	0x003	15:0	F+T‰	W	N/Y	Vertex A y-coordinate location (12.4 format)	
vertexBx	0x004	15:0	F+T‰	W	N/Y	Vertex B x-coordinate location (12.4 format)	
vertexBy	0x005	15:0	F+T‰	W	N/Y	Vertex B y-coordinate location (12.4 format)	
vertexCx	0x006	15:0	F+T‰	W	N/Y	Vertex C x-coordinate location (12.4 format)	
vertexCy	0x007	15:0	F+T‰	W	N/Y	Vertex C y-coordinate location (12.4 format)	
startR	0x008	23:0	F	W	N/Y	Starting Red parameter (12.12 format)	24
dRdX	0x009	23:0	F	W	N/Y	Change in Red with respect to X (12.12 format)	
dRdY	0x00a	23:0	F	W	N/Y	Change in Red with respect to Y (12.12 format)	
startG	0x00b	23:0	F	W	N/Y	Starting Green parameter (12.12 format)	
dGdX	0x00c	23:0	F	W	N/Y	Change in Green with respect to X (12.12 format)	
dGdY	0x00d	23:0	F	W	N/Y	Change in Green with respect to Y (12.12 format)	
startB	0x00e	23:0	F	W	N/Y	Starting Blue parameter (12.12 format)	
dBdX	0x00f	23:0	F	W	N/Y	Change in Blue with respect to X (12.12 format)	
dBdY	0x010	23:0	F	W	N/Y	Change in Blue with respect to Y (12.12 format)	
startZ	0x011	31:0	F	W	N/Y	Starting Z parameter (20.12 format)	
dZdX	0x012	31:0	F	W	N/Y	Change in Z with respect to X (20.12 format)	
dZdY	0x013	31:0	F	W	N/Y	Change in Z with respect to Y (20.12 format)	
startA	0x014	23:0	F	W	N/Y	Starting Alpha parameter (12.12 format)	
dAdX	0x015	23:0	F	W	N/Y	Change in Alpha with respect to X (12.12 format)	
dAdY	0x016	23:0	F	W	N/Y	Change in Alpha with respect to Y (12.12 format)	
startS	0x017	31:0	T	W	N/Y	Starting S/W parameter (14.18 format)	
dSdX	0x018	31:0	T	W	N/Y	Change in S/W with respect to X (14.18 format)	
dSdY	0x019	31:0	T	W	N/Y	Change in S/W with respect to Y (14.18 format)	
startT	0x01a	31:0	T	W	N/Y	Starting T/W parameter (14.18 format)	
dTdX	0x01b	31:0	T	W	N/Y	Change in T/W with respect to X (14.18 format)	
dTdY	0x01c	31:0	T	W	N/Y	Change in T/W with respect to Y (14.18 format)	
startW	0x01d	31:0	F+T	W	N/Y	Starting 1/W parameter (2.30 format)	
dWdX	0x01e	31:0	F+T	W	N/Y	Change in 1/W with respect to X (2.30 format)	
dWdY	0x01f	31:0	F+T	W	N/Y	Change in 1/W with respect to Y (2.30 format)	
triangleCMD	0x020	31	F+T‰	W	N/Y	Execute TRIANGLE command (sign bit)	29
reserved	0x021	na					
fvertexAx	0x022	31:0	F+T‰	W	N/Y	Vertex A x-coordinate location (floating point)	24
fvertexAy	0x023	31:0	F+T‰	W	N/Y	Vertex A y-coordinate location (floating point)	
fvertexBx	0x024	31:0	F+T‰	W	N/Y	Vertex B x-coordinate location (floating point)	
fvertexBy	0x025	31:0	F+T‰	W	N/Y	Vertex B y-coordinate location (floating point)	
fvertexCx	0x026	31:0	F+T‰	W	N/Y	Vertex C x-coordinate location (floating point)	
fvertexCy	0x027	31:0	F+T‰	W	N/Y	Vertex C y-coordinate location (floating point)	
fstartR	0x028	31:0	F	W	N/Y	Starting Red parameter (floating point)	25
fdRdX	0x029	31:0	F	W	N/Y	Change in Red with respect to X (floating point)	
fdRdY	0x02a	31:0	F	W	N/Y	Change in Red with respect to Y (floating point)	
fstartG	0x02b	31:0	F	W	N/Y	Starting Green parameter (floating point)	

fdGdX	0x02c	31:0	F	W	N/Y	Change in Green with respect to X (floating point)	
fdGdY	0x02d	31:0	F	W	N/Y	Change in Green with respect to Y (floating point)	
fstartB	0x02e	31:0	F	W	N/Y	Starting Blue parameter (floating point)	
fdBdX	0x02f	31:0	F	W	N/Y	Change in Blue with respect to X (floating point)	
fdBdY	0x030	31:0	F	W	N/Y	Change in Blue with respect to Y (floating point)	
fstartZ	0x031	31:0	F	W	N/Y	Starting Z parameter (floating point)	
fdZdX	0x032	31:0	F	W	N/Y	Change in Z with respect to X (floating point)	
fdZdY	0x033	31:0	F	W	N/Y	Change in Z with respect to Y (floating point)	
fstartA	0x034	31:0	F	W	N/Y	Starting Alpha parameter (floating point)	
fdAdX	0x035	31:0	F	W	N/Y	Change in Alpha with respect to X (floating point)	
fdAdY	0x036	31:0	F	W	N/Y	Change in Alpha with respect to Y (floating point)	
fstartS	0x037	31:0	T	W	N/Y	Starting S/W parameter (floating point)	
fdSdX	0x038	31:0	T	W	N/Y	Change in S/W with respect to X (floating point)	
fdSdY	0x039	31:0	T	W	N/Y	Change in S/W with respect to Y (floating point)	
fstartT	0x03a	31:0	T	W	N/Y	Starting T/W parameter (floating point)	
fdTdX	0x03b	31:0	T	W	N/Y	Change in T/W with respect to X (floating point)	
fdTdY	0x03c	31:0	T	W	N/Y	Change in T/W with respect to Y (floating point)	
fstartW	0x03d	31:0	F+T	W	N/Y	Starting 1/W parameter (floating point)	
fdWdX	0x03e	31:0	F+T	W	N/Y	Change in 1/W with respect to X (floating point)	
fdWdY	0x03f	31:0	F+T	W	N/Y	Change in 1/W with respect to Y (floating point)	
ftriangleCMD	0x040	31	F+T%	W	N/Y	Execute TRIANGLE command (floating point)	29
reserved	0x041	na				Required CMD spacer for SST-96.	
nopCMD	0x042	0	F+T%	W	Y/Y	Execute NOP command	
reserved	0x043	na				Required CMD spacer for SST-96.	
fastfillCMD	0x044	na	F	W	Y/Y	Execute FASTFILL command	
reserved	0x045	na				Required CMD spacer for SST-96.	
swapbufferCMD	0x046	8:0	F	W	Y/Y	Execute SWAPBUFFER command	
reserved	0x047	na				Required CMD spacer for SST-96	
swappendCMD	0x048	na	F	W	N/N	Increments swap pending count	
reserved	0x049	na				Required CMD spacer for SST-96.	
reserved	0x04a to 0x04f	na					
fbzColorPath	0x050	27:0	F+T%	R/W	N/Y	FBI Color Path Control	32
fogMode	0x051	5:0	F	R/W	N/Y	Fog Mode Control	
alphaMode	0x052	31:0	F	R/W	N/Y	Alpha Mode Control	
reserved	0x053	na					
fbzMode	0x054	19:0	F	R/W	Y/Y	Color and Auxiliary Buffer Control	41
stipple	0x055	31:0	F	R/W	Y/Y	Rendering Stipple Value	
color0	0x056	31:0	F	R/W	Y/Y	Constant Color #0	
color1	0x057	31:0	F	R/W	Y/Y	Constant Color #1	
fogColor	0x058	23:0	F	W	Y/Y	Fog Color Value	
zaColor	0x059	31:0	F	W	Y/Y	Constant Alpha/Depth Value	
chromaKey	0x05a	23:0	F	W	Y/Y	Chroma Key Compare Value	
chromaRange	0x05b	28:0	F	W	Y/Y	Chroma Range Compare Values, Modes & Enable	
reserved	0x05c to 0x05f	na					

colBufferSetup	0x060	31:0	F	R/W	Y/Y	Color Buffer Base Address, Stride and Mode	48
auxBufferSetup	0x061	30:0	F	R/W	Y/Y	Auxiliary Buffer Base Address and Stride	
clipLeftRight0	0x062	31:0	F	R/W	Y/Y	Left and Right of Clipping Register 0	
clipTopBottom0	0x063	31:0	F	R/W	Y/Y	Top and Bottom of Clipping Register 0	
clipLeftRight1	0x064	31:0	F	R/W	Y/Y	Left and Right of Clipping Register 1	
clipTopBottom1	0x065	31:0	F	R/W	Y/Y	Top and Bottom of Clipping Register 1	
reserved	0x066 to 0x06f	na					
fogTable	0x070 to 0x08f	31:0	F	W	Y/Y	Fog Table	51
fbijrInit0	0x090	17:0	F	R/W	N/N	FBIjr Hardware Init 0 - General	52
fbijrInit1	0x091	24:0	F	R/W	N/N	FBIjr Hardware Init 1 - PUMA	
fbijrInit2	0x092	18:0	F	R/W	N/N	FBIjr Hardware Init 2 - MEM FIFO Setup	
fbijrInit3	0x093	30:0	F	R/W	N/N	FBIjr Hardware Init 3 - CMD FIFO Setup	
fbijrInit4	0x094	18:0	F	R/W	N/N	FBIjr Hardware Init 4 - CMD FIFO Entry Count	
fbijrInit5	0x095	18:0	F	R/W	N/N	FBIjr Hardware Init 5 - CMD FIFO Read Pointer	
reserved	0x096 to 0x09f	na					
fbijrVersion	0x0a0	31:0	F	R	na	FBIjr Version (company, device, & board)	56
fbiPixelsIn	0x0a1	23:0	F	R	na	Pixel Counter (Number pixels processed)	
fbiChromaFail	0x0a2	23:0	F	R	na	Pixel Counter (Number pixels failed Chroma test)	
fbiZfuncFail	0x0a3	23:0	F	R	na	Pixel Counter (Number pixels failed Z test)	
fbiAfuncFail	0x0a4	23:0	F	R	na	Pixel Counter (Number pixels failed Alpha test)	
fbiPixelsOut	0x0a5	23:0	F	R	na	Pixel Counter (Number pixels drawn)	
reserved	0x0a6 to 0x0be	na					
texChipSel	0x0bf	1:0	F	R/W	N/Y	Texture Write Chip Select	
textureMode	0x0c0	30:0	T	W	N/Y	Texture Mode Control	58
tLOD	0x0c1	23:0	T	W	N/Y	Texture LOD Settings	
tDetail	0x0c2	16:0	T	W	N/Y	Texture LOD Settings	
texBaseAddr	0x0c3	18:0	T	W	N/Y	Texture Base Address	
texBaseAddr_1	0x0c4	18:0	T	W	N/Y	Texture Base Address (supplemental LOD 1)	
texBaseAddr_2	0x0c5	18:0	T	W	N/Y	Texture Base Address (supplemental LOD 2)	
texBaseAddr_3_8	0x0c6	18:0	T	W	N/Y	Texture Base Address (supplemental LOD 3-8)	
TREXInit0	0x0c7	31:0	T	W	Y/Y	TREX Hardware Initialization (register 0)	
TREXInit1	0x0c8	31:0	T	W	Y/Y	TREX Hardware Initialization (register 1)	
nccTable0	0x0c9 to 0x0d4	31:0 26:0	T	W	Y/Y	Narrow Channel Compression Table 0 (12 entries)	
nccTable1	0x0d5 to 0x0e0	31:0 26:0	T	W	Y/Y	Narrow Channel Compression Table 1 (12 entries)	
reserved	0x0e1 to 0x0ff	na					

8. Register Definitions

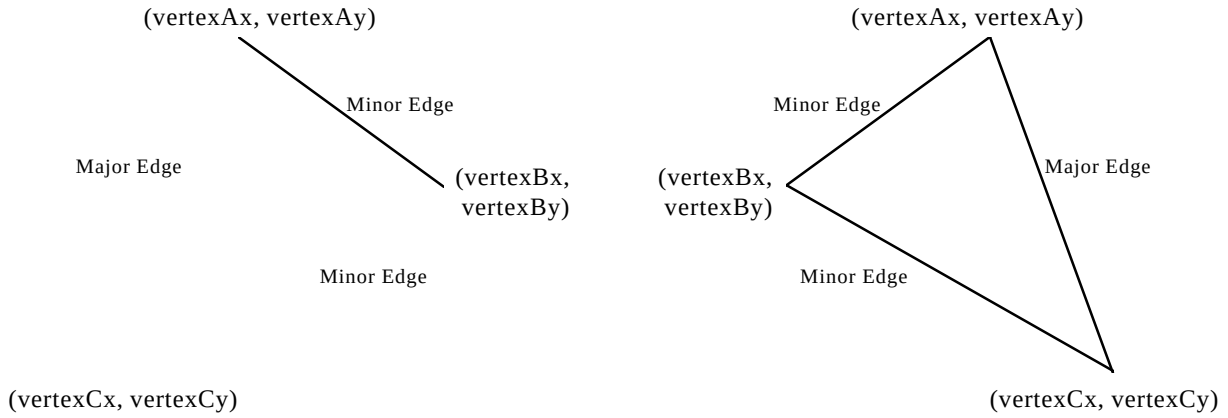
8.1 status Register

The **status** register is a read only register which provides the CPU information on the current state of the graphics processor. The data of the register indicates idleness, swap activity and FIFO fullness. This information is available through both the status register of FBIjr and a status register within the 2D chip. Software is encouraged to acquire the status through the 2D chip as this access does not tax the frame buffer bandwidth. However, software must accept inaccuracies in 2D status resulting from an update latency of up to **TBD** (50+) memory clocks.

Bit	Description
0	**FBIjr Busy** Indicates a busy FBIjr. This includes all units and FIFOs of FBIjr. Default is 0.
1	**TREX Busy** Indicates a busy TREX. This includes all units and FIFOs of the TREX. Default is 0.
2	**Swap Request** Indicates an active swap request from the 3D chip to the 2D chip. Hardware generates swap requests for each swap buffer command (**swapbufferCMD**). Default is 0.
3	**Swap Sync Signal** Indicates the actual VSYNC signal used by the hardware for buffer swap. Software alters the active level and period of this signal through the **fbijrInit0** register. Default is X.
6:4	**Swap Pending Count** Indicates the number of outstanding buffer swaps. Software increments the count with direct writes to the **swappendCMD** Register and hardware decrements the count with the completion of each swap buffer command (**swapbufferCMD**). Valid values range from 0 to 7 swaps. Excessive increments clamp a count of 7 and excessive decrements clamp a count of 0. Default is 0x0.
7	**CMD FIFO Fullness** Indicates a fullness level of the CMDFIFO within PUMA DRAM. An entry-count greater than or equal to the page-threshold of the **fbijrInit2** register sets the fullness status bit and an entry-count less than the threshold resets the status bit. Default is 0.
13:8	**MEM FIFO Free Space** Indicates the free space count of the MEMFIFO of FBIjr. Valid values range from 0 to 63 double-words. Default is 0x0.
14	**CMD FIFO Overflow Error** Indicates a CMDFIFO overflow error. FBIjr sets the error when the CMDFIFO entry count exceeds the CMDFIFO space (top page - bottom page of the **fbijrInit2** register). This error condition is sticky – once set the bit remains set until reset through hard or soft reset. Default is 0x0.
15	**MEM FIFO Overflow Error** Indicates a MEMFIFO overflow error. FBIjr sets the error when data is pushed onto the MEMFIFO as the MEMFIFO free-space count equals zero. This error condition is sticky – once set the bit remains set until reset through hard or soft reset. Default is 0x0.

8.2 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 the 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-96 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)

8.3 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-96 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)

8.4 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-96 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)

8.5 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. The start S and T coordinates must be divided by W. The 3D engine 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-96 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)

8.6 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. The **W** value used for rendering is actually the reciprocal of the 3D-geometry-calculated W value. The 3D engine 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-96 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)

8.7 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 change in color values are added to the color component as the pixel drawn moves from left-to-right, and are subtracted from the color component as the pixel drawn moves from right-to-left. The **fd?dX** registers are floating point equivalents of the **d?dX** registers. SST-96 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)

8.8 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-96 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)

8.9 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. The delta S/W and T/W values used by SST-96 for rendering must be divided by W (i.e. SST-96 uses $\Delta S/W$ and $\Delta T/W$). The **d?dX** registers are floating point equivalents of the **fd?dX** registers. SST-96 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)

8.10 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-96 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)

8.11 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-96 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)

8.12 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-96 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)

8.13 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. The delta S/W and T/W values used by SST-96 for rendering must be divided by W (i.e. SST-96 uses $\Delta S/W$ and $\Delta T/W$). The **d?dY** registers are floating point equivalents of the **fd?dY** registers. SST-96 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)

8.14 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-96 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)

8.15 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 (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. The area determines whether a triangle is clockwise or counter-clockwise geometrically. If bit(31)=0, the triangle is oriented in a counter-clockwise orientation (positive area), otherwise, if bit(31)=1, the triangle is oriented in a clockwise orientation (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-96 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).

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)

8.16 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)

8.17 fastfillCMD Register

Writing to the **fastfillCMD** register executes the FASTFILL command. This command clears a rectangular area of the color and auxiliary buffers. Clearing of the color buffer is enabled through RGB write mask (**fbzMode[9]**) and clearing of the auxiliary buffer is enabled through the depth/alpha write mask (**fbzMode[10]**).

The FASTFILL clearing rectangle is defined by the **clip0** registers. These registers define four corners of the rectangle inclusive of the **clipLeft** and **clipLoY** values and exclusive of the **clipRight** and **clipHiY** values. The state of **clipEnable** and **clipMode** are also ignored as the FASTFILL rectangle is always enabled and inclusive.

The FASTFILL clearing values are defined by the **color1** and **zaColor** registers. The **color1** register defines the color buffer value and the **zaColor** register defines the auxiliary buffer value. The **color1** register defines a 24-bit color which is dithered or truncated to the 16-bit frame buffer format as defined by **fbzMode[8]**.

SST-96 stores color and auxiliary data on separate pages of memory. Operations to both buffers requires a page miss between accesses. As a result, for greater performance, SST-96 software is encouraged to clear the color and auxiliary buffers separately.

8.18 swapbufferCMD Register

The **swapbufferCMD** register executes a SWAPBUFFER command. This command synchronizes swapping of the video refresh and rendering buffers of the 2D and 3D chips. Each swap command causes a swap request from the 3D engine to the video refresh engine of the 2D chip. Then with each swap request, the 2D chip swaps buffers at the next VSYNC period.

The 3D engine issues swap requests to the 2D engine after the graphics pipeline has emptied and the swap buffer interval has been met. Defined within this command, the swap buffer interval specifies the minimum number of VSYNC periods between swaps. This interval forces a maximum frame rate, allowing a greater visual update consistency.

The SWAPBUFFER command stalls the 3D engine to prohibit writing into the front buffer. Double buffering stalls the graphics engine with the first pending swap request and triple buffering stalls the graphics engine with the second pending swap request. While, stereo quad buffering is equivalent to double buffering left and right image pairs.

Software may utilize the swap buffer pending count of the **fbijrStatus** register to determine the number of frames queued in the hardware. This count increments with direct writes to the **swappendCMD** register and decrements with the completion of each SWAPBUFFER command.

Bit	Description
0	Reserved (was SST-1: **Swap Synch Enable**)
8:1	**Swap Buffer Interval** Defines the minimum number of VSYNC periods between swaps. The programmed value specifies N-1 VSYNC intervals, where N equals the minimum number of VSYNCs between swaps. Values range from 0 to 2**8. The default is 0.

SST-96 does not support the SST-1 Swap Sync Enable feature. This feature enabled SST-1 to swap refresh buffers mid-frame. Mid-frame swapping is not possible with the SST-96 Alliance solution. The Alliance solution tiles 3D data onto the desktop with no support for mid-frame tile swapping.

8.19 swappendCMD Register

Writes to the **swappendCMD** register increment the swap buffer pending count of the FBIjr status register. Writes take effect immediately and are available only through direct access (NO CMDFIFO ACCESS).

Bit	Description

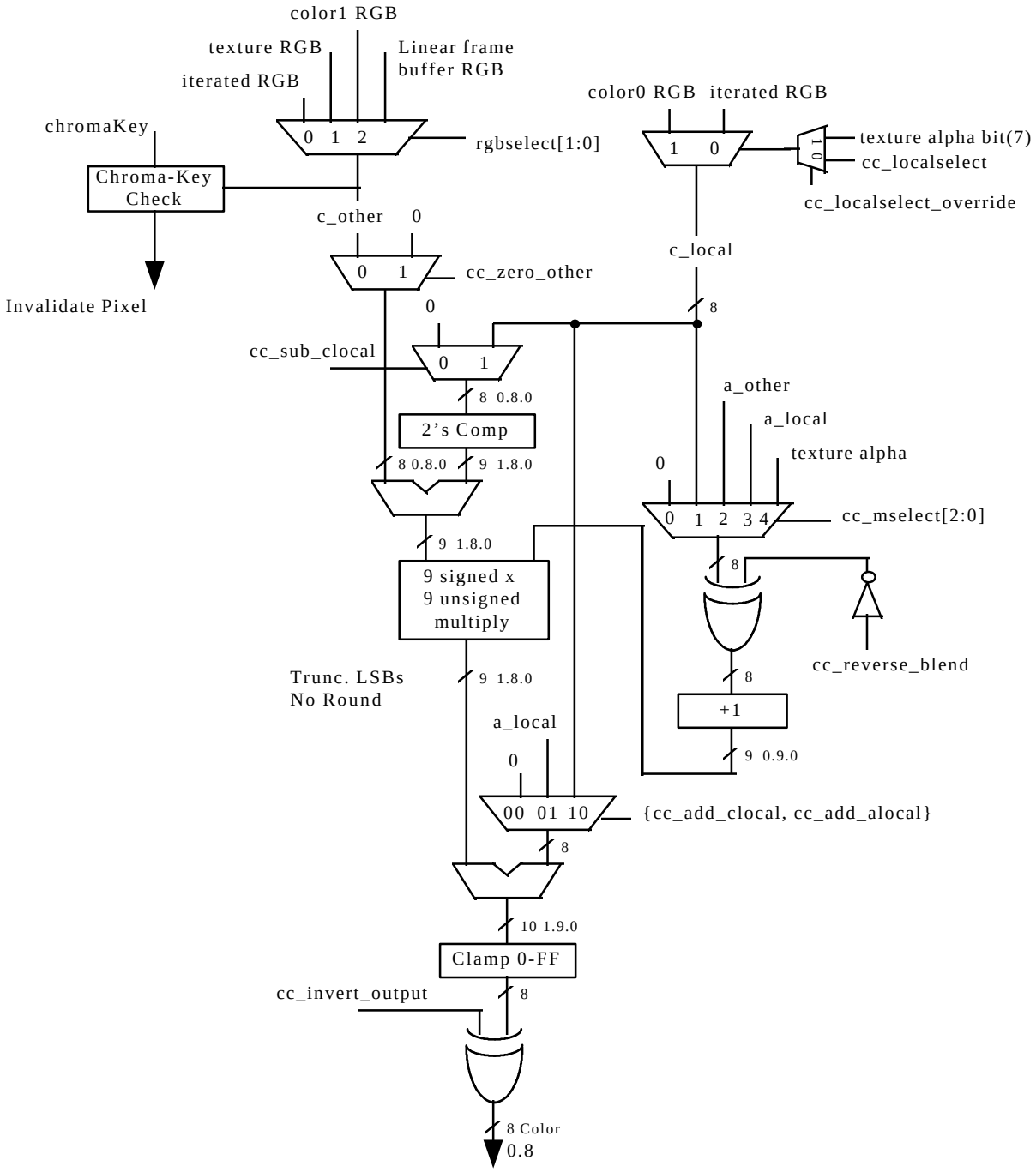
na	**Swap Buffer Pending Count Increment** Writes to this register increment the 3-bit swap buffer pending count of the FBIjr status register. This count increments through direct writes to this register and decrements with the completion of each swap buffer command. Excessive increments clamp a count of 7 and excessive decrements clamp a count of 0. The default is X.

8.20 fbzColorPath Register

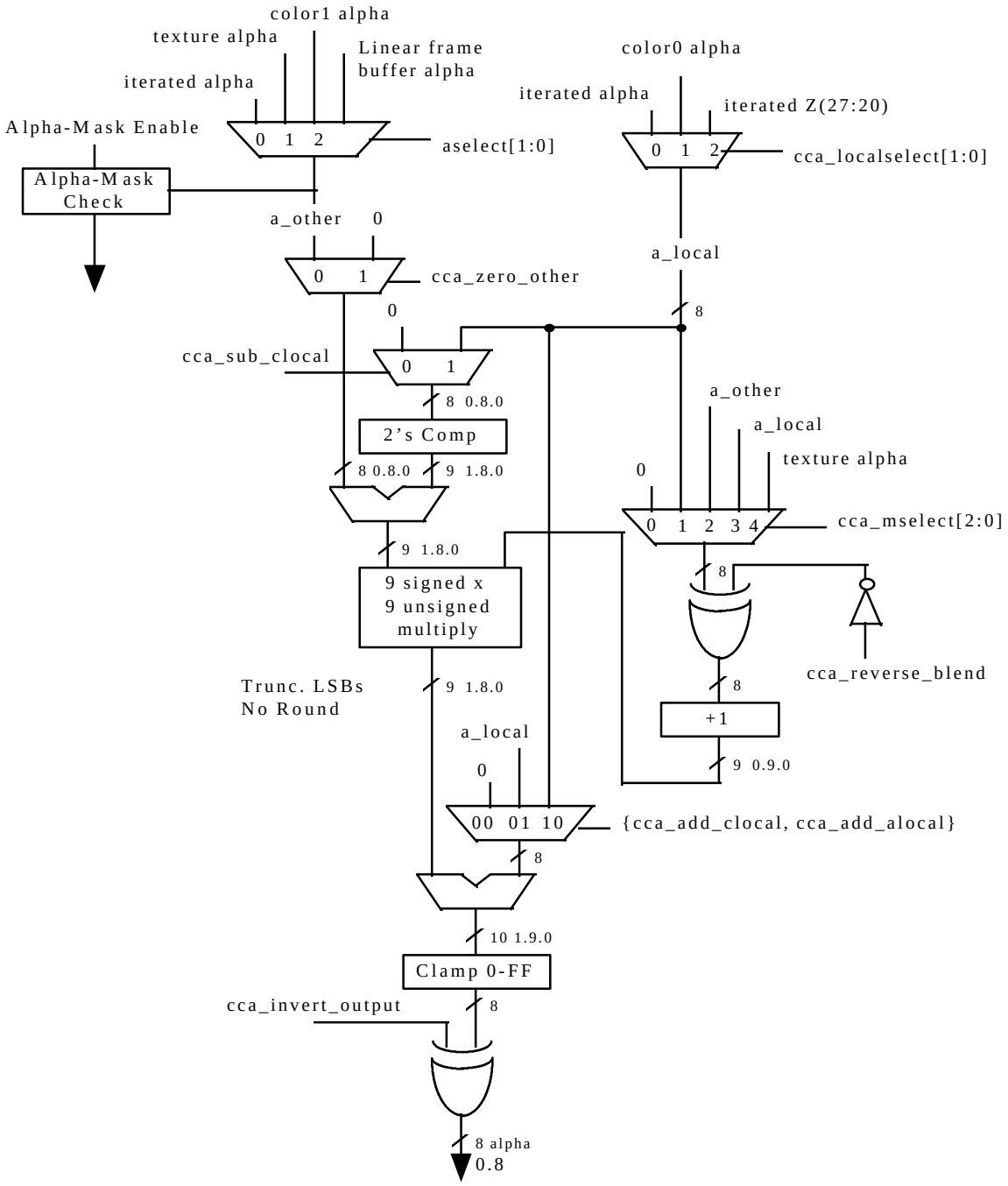
The **fbzColorPath** register controls the color and alpha rendering pixel pipelines. Bits in **fbzColorPath** control color/alpha selection and lighting. Individual bits of **fbzColorPath** are set to enable modulation, addition, etc. for various lighting effects including diffuse and specular highlights.

Bit	Description
1:0	RGB Select (0=Iterated RGB, 1=T Color Output, 2=**Color1** RGB, 3=Reserved)
3:2	Alpha Select (0=Iterated A, 1=T 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:


color1 RGB
texture RGB
Linear frame buffer RGB
iterated RGB
chromaKey
0 1 2
rgbselect[1:0]
Chroma-Key Check
Invalidate Pixel
color0 RGB iterated RGB
1 0
texture alpha bit(7)
cc_localselect
cc_localselect_override
c_other 0
0 1
cc_zero_other
c_local
8
0
cc_sub_clocal
0 1
8 0.8.0
2's Comp
8 0.8.0 9 1.8.0
9 1.8.0
9 signed x 9 unsigned multiply
a_other
a_local
texture alpha
0
0 1 2 3 4
cc_mselect[2:0]
8
cc_reverse_blend
Trunc. LSBs No Round
9 1.8.0
8
+1
9 0.9.0
a_local
0
00 01 10
{cc_add_clocal, cc_add_alocal}
8
10 1.9.0
Clamp 0-FF
cc_invert_output
8
8 Color 0.8

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:



Bit(26) of **fbzColorPath** enables subpixel correction for all parameters. When enabled, SST-96 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].

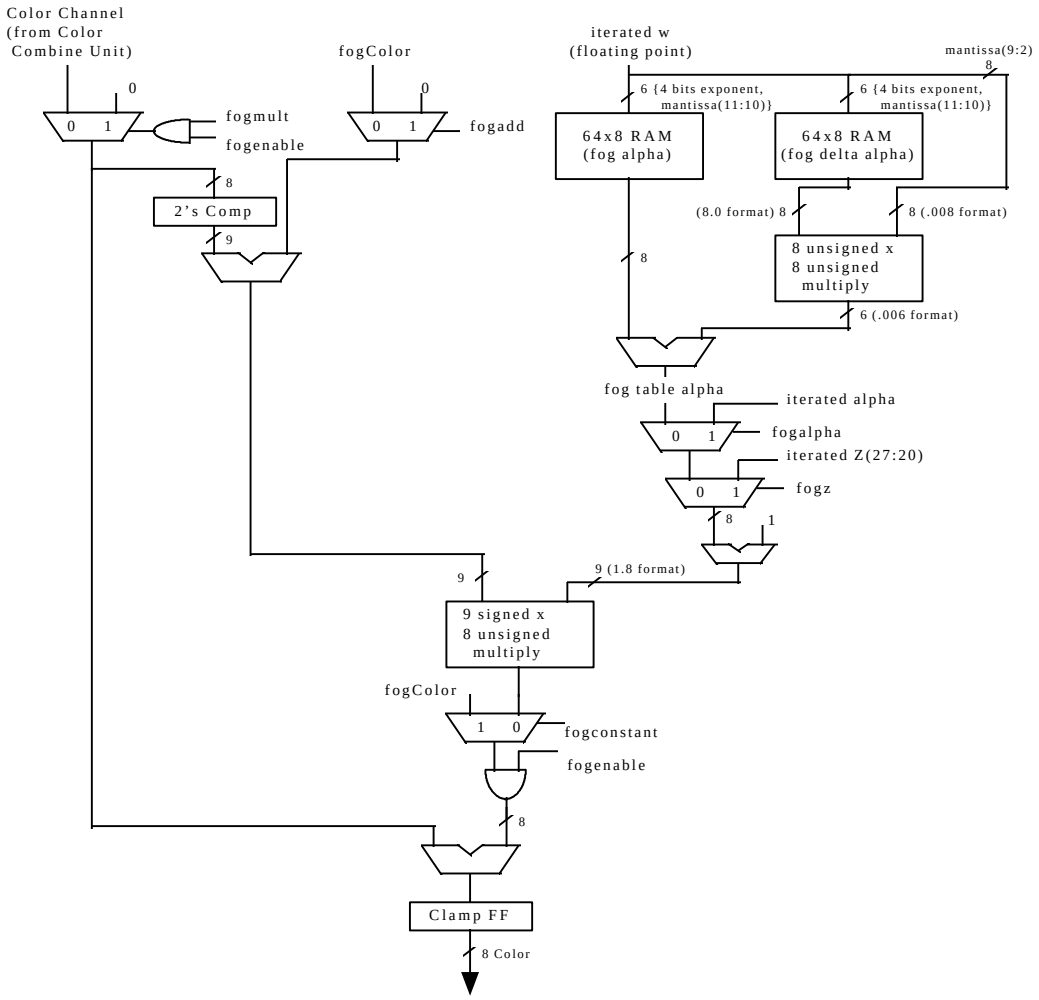
Bit(27) of **fbzColorPath** is used to enable texture mapping. If texture-mapped rendering is desired, then bit(27) of **fbzColorPath** must be set. When bit(27)=1, then data is transfered from T to FBI. If texture mapping is not desired (i.e. Gouraud shading, flat shading, etc.), then bit(27) may be cleared and no data is transfered from T to FBI.

8.21 fogMode Register

The **fogMode** register controls the fog functionality of SST-96.

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-96:



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-

96 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

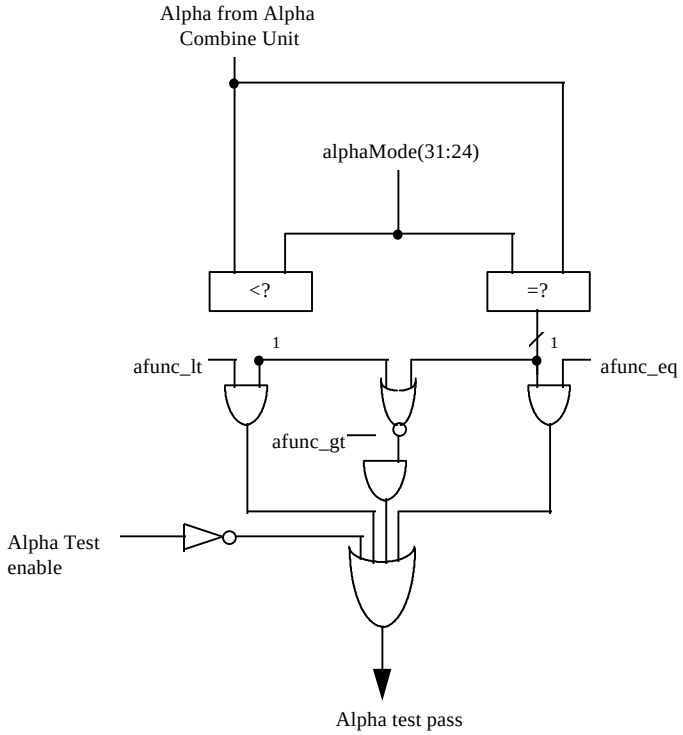
When bit(3) of **fogMode** is set, the integer part of the iterated alpha component is used as the fog alpha instead of the calculated fog alpha value from the fog table. When bit(4) of **fogMode** is set, the upper 8 bits of the iterated Z component are used as the fog alpha instead of the calculated 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.

8.22 alphaMode Register

The **alphaMode** register controls the alpha blending and anti-aliasing functionality of SST-96.

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)
7:8	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-96.

8.22.1 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

8.22.2 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-96 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.

8.23 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 rectangles (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	Reserved (Was SST-1 Draw buffer)
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]. *This mode was not present in FBI revision 1.0.*

Bit(0) of **fbzMode** is used to enable clipping. When set, rendering is clipped to the rectangles defined by the **clipLeftRight[01]** and **clipBottomTop[01]** registers. Each of these registers defines a clip rectangle which may be selectively enabled to include or preclude rendering. Applications rendering outside of the screen resolution must use one of the two clip rectangles, set to screen coordinates, to preclude drawing to off-screen coordinates.

Bit(1) of **fbzMode** is used to enable the chroma-key color compare check. 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.

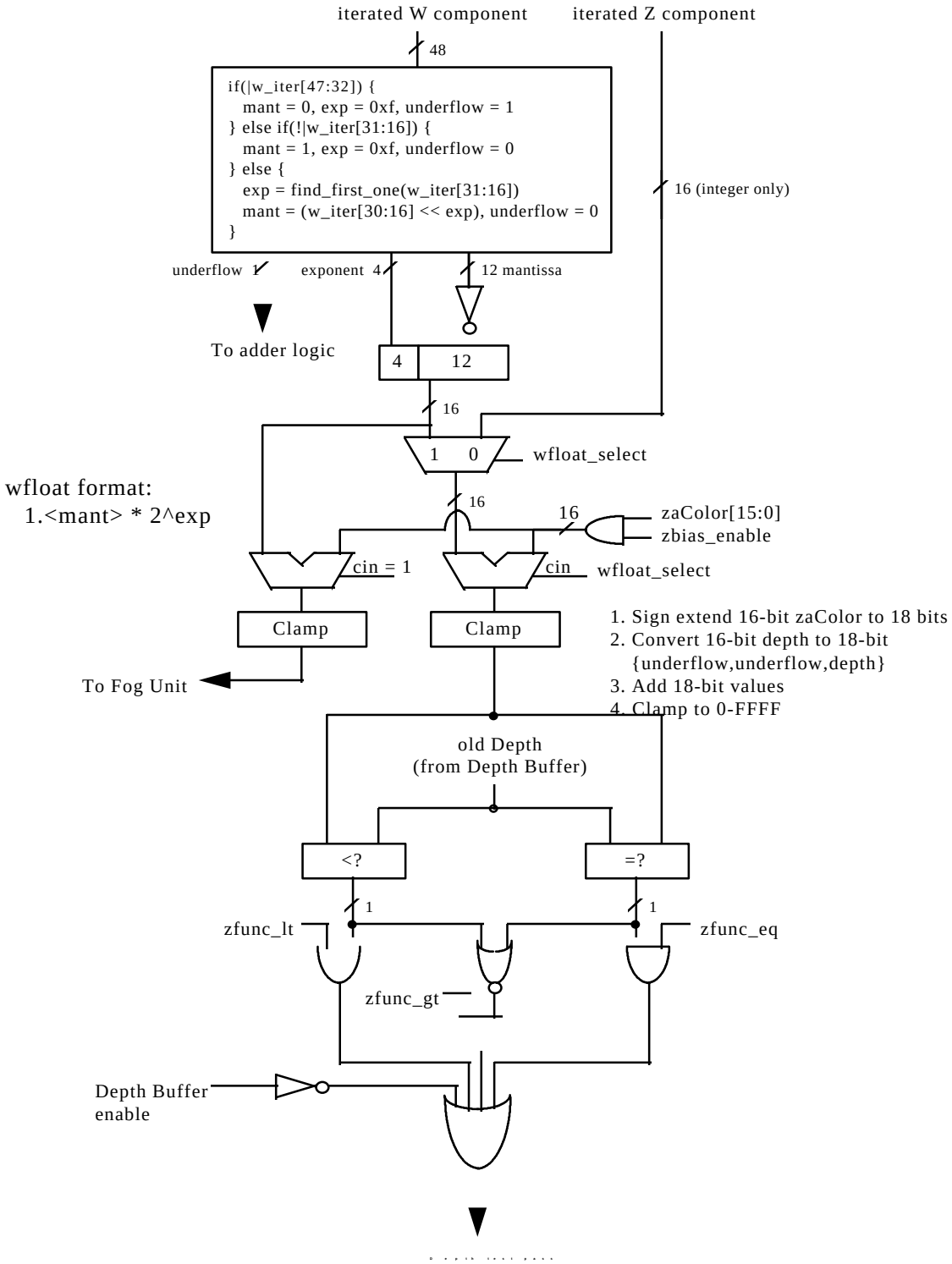
When stipple register masking is enabled and stipple pattern mode is selected, the spatial <x,y> coordinates of a pixel processed in the pixel pipeline are used to lookup a 4x8 monochrone pattern stored in the **stipple** register --

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 8.23.1 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** was used to select the RGB draw buffer for graphics drawing with SST-1. For SST-96 drawing is performed into the buffer selected by the **colorBufferSetup** register.

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). When cleared, the Y origin (Y=0) for all rendering operations when the pixel pipeline is enabled are 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.

8.23.1 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

8.24 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

8.25 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

8.26 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.

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

8.27 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

8.28 zaColor Register

The **zaColor** register is used to specify constant alpha and depth values for FASTFILL commands, and co-planar polygon rendering support. 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

8.29 chromaKey Register

The **chromaKey** register specifies a 24-bit RGB color value which is compared to all pixels to be written to the color buffer. If chroma-keying is enabled (**fbzMode[1]**) and chroma-ranging is disabled (**chromaRange[28]**), an outgoing pixel color equaling the color of the **chromaKey** register blocks the pixel write into the frame buffer.

The chroma comparison is performed immediately after texture lookup, texture blending and Gouraud shading, but before lighting, fog and alpha blending. Chroma-keying ignores the alpha color component of the outgoing pixel. For more information see the **fbzColorPath** block diagram.

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

8.30 chromaRange Register

The **chromaRange** register specifies a color range which is compared to all pixels to be written to the color buffer. If chroma-keying is enabled (**fbzMode[1]**) and chroma-ranging is enabled (**chromaRange[28]**), the outgoing pixel color is compared to a color range formed by the colors of the **chromaKey** and **chromaRange** registers.

The RGB color components of the **chromaKey** and **chromaRange** registers define a color data range for the color components. The color component range includes all color values between and including the lower limit color of the **chromaKey** register and the upper limit color of the **chromaRange** register. Note, software must program the **chromaKey** lower limits less-than or equal to the **chromaRange** upper limits**.**

The RGB color component mode bits of the **chromaRange** register defines the color component range mode as inclusive or exclusive. Inclusive ranges select colors within the range and exclusive ranges select colors outside of the range.

The block mode bit of the **chromaRange** register sets the blocking mode for colors selected within each color component ranges. A blocking mode of "intersection" blocks pixels prohibited by all of the color components and a blocking mode of "union" blocks pixels prohibited by any of the color components.

Bit	Description
7:0	Chroma-Range Blue Upper Limit
15:8	Chroma-Range Green Upper Limit
23:16	Chroma-Range Red Upper Limit
24	Chroma-Range Blue Mode (0=inclusive; 1=exclusive)
25	Chroma-Range Green Mode (0=inclusive; 1=exclusive)
26	Chroma-Range Red Mode (0=inclusive; 1=exclusive)
27	Chroma-Range Block Mode (0=intersection; 1=union)
28	Chroma-Range Enable (0=disable; 1=enable)
31:29	reserved

8.31 colBufferSetup and auxBufferSetup Registers

The **colBufferSetup** and auzBufferSetup registers define the base address and X stride of the color and auxiliary buffers. The color buffer setup register additionally defines the color buffer mode determining the color depth of the buffer as 16-bit RGB or 8-bit color index. The corresponding mode for the auxiliary buffer which defines the buffer as 16-bit depth or 8-bit alpha is established within the **fbzMode** register.

colBufferSetup

Bit	Description
21:0	**Color Buffer Base Address** Sets the color buffer base byte address within the lower 4MB PUMA DRAM address space. Software must set this register with a quad-pixel alignment identical to that of the auxiliary buffer base address. Default is X.
30:22	**Color Buffer X Stride** Defines the X quad-pixel dimension of the color buffer. Full screen 3D rendering sets the dimension of the 3D image and windowed 3D rendering sets the dimension of the desktop. Valid values range from 1 to 511 quad pixels. Default is X.
31	**Color Buffer Mode** sets the color buffer mode to 16-bit RGB or 8-bit color index (0=16-bit RGB; 1=8-bit color index). In either case the buffer consumes a packed linear space. Default is 0.

auxBufferSetup

Bit	Description
21:0	**Auxiliary Buffer Base Address** Sets the auxiliary buffer base byte address within the lower 4MB PUMA DRAM address space. Software must set this register with a quad-pixel alignment identical to that of the color buffer base address. Default is X.
30:22	**Auxiliary Buffer X Stride** Defines the X quad-pixel dimension of the auxiliary buffer. Full screen or windowed 3D rendering sets the dimension of the 3D image. Valid values range from 1 to 511 quad pixels. Default is X.

The following equation describes the calculations of LFB address derived from the X and Y rendering coordinates, the auxiliary buffer base address (BaseAdr) and the auxiliary buffer X stride (xStride) settings.

LFBA (bytes) = BaseAdr + (((y * (xStride * 4)) + x) * N), where N = 1 or 2 bytes/pixel.

Note: X stride is negated in this calculation for the flipped Y origin case.

Note2: drawing operations (triangles or fast fill) to X/Y coordinates which extend beyond the 4MB frame buffer space (after LFB translation) wrap the frame buffer space modifying addresses in the lower frame buffer.

8.32 clipLeftRight0, clipTopBottom0, clipLeftRight1 and clipTopBottom1 Registers

The **clip0** and **clip1** registers specify two rectangular regions which restrict drawing operations. Each rectangle may be defined as inclusive or exclusive through the **clipMode** field of the **clipTopBottom** register. An inclusive rectangle allows drawing within the rectangle and an exclusive rectangle disallows drawing within the rectangle. Drawing within an excluded region of either of the enabled clip rectangles circumvents the write of pixels into both the color and auxiliary buffers.

The clip registers define the four corners of a rectangular region in window relative pixel coordinates (native x/y rendering coordinates). The value of **clipLoY** must be less than **cliipHiY** and the value of **clipLeft** must be less than **clipRight**. This programming results in a rectangular region including the **clipLeft** and **clipLoY** register values, but excluding the **clipRight** and **clipHiY** register values.

The function of each clip rectangle may be selectively enabled and disabled through the **ClipEnable** bit of each of the **ClipLeftRight** registers and a global enable also exists within the **fbzMode** register.

The **clip0** registers also define the rectangular region for the FASTFILL command. For FASTFILL, the clip rectangle is always enabled and always inclusive (i.e. **clipMode** and **clipEnable** are ignored).

ClipLeftRight[01] Register

Bit	Description
10:0	Unsigned integer specifying right clipping rectangle edge
15:11	reserved
26:16	Unsigned integer specifying left clipping rectangle edge
30:27	reserved
31	Clip Enable (0=disable; 1=enable).

clipTopBottom[01] Register

Bit	Description
10:0	Unsigned integer specifying high Y clipping rectangle edge
15:11	reserved
26:16	Unsigned integer specifying low Y clipping rectangle edge
30:27	reserved
31	Clip Mode (0=inclusive; 1=exclusive).

8.33 fogTable Register

The **fogTable** register is used to implement fog functions in SST-96. 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

8.34 fbijrInit0, fbijrInit1, fbijrInit2, fbijrInit3, fbijrInit4 and fbijrInit5 Registers

The **fbijrInit** registers contain the initialization bits of FBIjr. Writes to these registers take effect immediately and are available through direct access only (NO CMDFIFO ACCESS). Software must take care when writing to these registers while the graphics engine or command FIFO pipeline is busy. These registers are read and write assessable.

fbijrInit0 - General

Bit	Description
0	**Software Reset** Resets all of FBIjr minus the PUMA arbiter, the PUMA direct access path, the serial status port, the Init registers and the configuration pins (0=run; 1=reset). Default is 1.
1	**TREX Texture Mapping Disable** Disables texture combine within FBIjr. Default is 0.
2	**Triple Buffering Enable** Enables triple buffering operation. Triple buffering allows the graphics engine to get ahead one swap buffer command before stalling for VSYNC. Default is 0.
3	**VSYNC Level Mode** Sets the VSYNC active edge for the detection of refresh buffer swap (0=active low; 1=active high). Default is 0.
4	**VSYNC Stereo Mode** Inhibits swapping on the odd-numbered VSYNC pulses. The first odd-numbered VSYNC is defined as the first VSYNC pulse after this bit is set. Default is 0.
5	**DRAM Write to Read Slow Timing Enable** Enables a dead cycle between write and read transactions. Default is 0.
6	**DRAM Read Fast RAS Off Timing Enable** Enables the removal of RAS before the end of a read transaction. Default is 0.
7	**DRAM Read to Write Fast Timing Enable** Enables a dead cycle within the DRAM timing between read and write transactions. Default is 0.
11:8	**FBI-To-TREX Bus Clock Delay** Valid values delay the clock 0-15 increments. Default is 0x2.
16:12	**TREX-To-FBI FIFO High Water Mark** Marks a fullness level which stalls data received from the TREX. Valid values range from 0 to 31 entries. Default is 0x0f.
17	**Serial Status Transfer Mode** Sets the active clock edge for the transfer of serial status out of FBIjr (0 = rising edge; 1 = falling edge). Default is 0.

InitState = 0x0000_f201

fbijrInit1 - PUMA

Bit	Description
5:0	**PUMA Low Priority Request Delay Count** Delays the detection of a low priority PUMA request and the beginning of the client grant sequence (provided the core device desires ownership). Valid values range from 0 to 63 cycles. Default is 0.
6	**PUMA Low Priority Grant Limit Enable** Enables the PUMA Low Priority Grant Limit Count. Default is 0.
12:7	**PUMA Low Priority Grant Limit Count** Limits the PUMA grant time of low priority client ownership (provided core device desires ownership). Valid values range from 0-63 cycles. Default is X.
18:13	**PUMA Low Priority Request Urgency Count** Creates a window of time upon which the PUMA core device attempts to gracefully complete the current operation before relinquishing the bus for a low priority request. Valid values range from 0-63 cycles. Default is 0.
22:19	**PUMA High Priority Request Urgency Count** Creates a window of time upon which the PUMA core device attempts to gracefully complete the current operation before relinquishing the bus for a high priority request. Valid values range from 0-15 cycles. Default is 0.
23	**PUMA Texture Base Address Mode** Sets the texture memory base address to 2MB or 6MB within the 8MB PUMA address space (0=2MB and 1=6MB). In the 2MB mode, FBIjr decodes the texture space beginning at the 2MB PUMA address. In the 6MB mode, FBIjr decodes the texture memory space defined in section 5. This bit is ignored while in a 4MB PUMA hardware configuration (section **Error! Reference source not found.**). Default is 1.
24	**PUMA Texture Address Space Mode** Sets the texture memory address space to 1MB or 2MB of the 8MB PUMA address space (0=1MB; 1=2MB). In the 2MB mode, FBIjr decodes the PUMA space defined in section 5. In the 1MB mode, FBIjr decodes a 1MB texture space removing the MSB of LOD from the PCI address. In this mode, the hardware forces LOD[3]=0 and software must load the LOD8 through careful addressing of the texture base address register. This bit is ignored while in a 4MB PUMA hardware configuration (section **Error! Reference source not found.**). Default is 1.

InitState = 0x0180_0000

fbijrInit2 - MEMFIFO Setup

Bit	Description
0	**MEMFIFO Direct Write Stall Disable** Disables the MEMFIFO stall mechanism (PUMA preemption) as direct writes encounter the low space mark of the MEMFIFO. Default is 0.
6:1	**MEMFIFO Direct Write Low Space Mark** Causes the suspension of direct writes into the MEMFIFO. Emptiness equaling this value stalls direct-write accesses through PUMA preemption provided MEMFIFO direct write stall has not been disabled (**fbijrInit2[0]**). The direct-write stall blocks low-priority PUMA request granting until the MEMFIFO empties to meet the high space mark. This register field must be set to sufficiently accommodate the PUMA stall latency of the 2D engine. Valid values range from 0 to 63. Default is 16.
12:7	**MEMFIFO CMDFIFO Read Low Space Mark** Causes the suspension of CMDFIFO reads into the MEMFIFO. Emptiness equaling this value suspends CMDFIFO reads until the MEMFIFO empties to the high space mark. This register must be set to sufficiently accommodate the CMDFIFO pending reads plus subsequent direct writes. Valid values range from 0 to 63. Default is 26.
18:13	**MEMFIFO High Space Mark** Causes the resumption of data acceptance into the MEMFIFO. Emptiness equaling this value enables direct-write accesses or CMDFIFO read operations which have been suspended through the low space marks. Valid values range from 0 to 63. Default is 56.

InitState = 0x0007_0D2D

fbijtInit3 - CMDFIFO Setup

Bit	Description
0	**CMDFIFO Enable** Enables the CMDFIFO within PUMA DRAM. Default is 0.
10:1	**CMDFIFO Bottom Page** Defines the bottom 4KB page of the CMDFIFO positioned within the 4MB DRAM space of PUMA. This register field, compared against the PCI byte address bits [21:12], determines the first page of the CMDFIFO. Valid values range from 1 to 1K pages. Default is X.
20:11	**CMDFIFO Top Page** Defines the top 4KB page of the CMDFIFO positioned within the 4MB DRAM space of PUMA. This register field, compared against the PCI byte address bits [21:12], determines the last page of the CMDFIFO. Valid values range from 1 to 1K pages. Default is X.
30:21	**CMDFIFO Page Threshold** Defines the 4KB page entry-count threshold used in generating the single bit CMDFIFO fullness status of the **fbijrStatus** register. CMDFIFO entry-counts greater than or equal to this threshold set the fullness status and entry-counts lower than this threshold reset the fullness status. Default is X.

InitState = 0x0018_0600

fbijrInit4 - CMDFIFO Entry Count

Bit	Description
18:0	**CMDFIFO Entry Count** Identifies the 64-bit quad-word entry-count of the CMDFIFO. Software initializes the count through direct write access and hardware increments the count as software writes to the FIFO. Then, an enabled CMDFIFO, with an entry-count greater than 0, causes the hardware to read from the CMDFIFO while subtracting from the entry-count. Valid values range from 0 to 512K quad-words. Default is X.

InitState = 0x0000_0000

fbijrInit5 - CMDFIFO Read Pointer

Bit	Description
18:0	**CMDFIFO Read Pointer** Identifies the 64-bit quad-word read-pointer address of the CMDFIFO within the 4MB DRAM space of PUMA. The read-pointer bits reflect bits [21:3] of the PCI byte address. Software initializes the pointer through direct write access and hardware increments the pointer as the 3D engine reads from the FIFO. Hardware reads from the CMDFIFO when the CMDFIFO is enabled and the CMDFIFO entry-count is greater than 0. Valid values range from 0 to 512K quad-words. Default is X.

InitState = 0x0018_0600

8.35 fbijrVersion Register

The **fbijrVersion** register contains four version fields identifying the Board Version, the FBIjr Chip Version, the fbijr Device Version, and the 3Dfx PCI Vender Number. The contents of the 4-bit board version field is set through hardware configuration and all of the other fields are hardwired within FBIjr. This register is read-only.

Bit	Description
3:0	**Board Version ID** Identifies the board version of SST-96. The board version is set through Hardware Configuration straps on the ft_data[3:0] bus bits of FBIjr. FBIjr tri-states this bus during reset to sample the input from external resistive configuration straps. Valid values range from 0 to 16. The initial board version value is 0.
7:4	**FBIjr Version ID = 0x1** Identifies the chip version of FBIjr.
15:8	**FBIjr Device ID = 0x02** Identifies FBIjr as the third chip from 3Dfx.
31:16	**PCI Vendor ID = 0x121a** Identifies 3Dfx through the reserved PCI vendor ID number.

8.36 fbiPixelsIn Register

The **fbiPixelsIn** register is a 24-bit counter which is incremented for each pixel processed by the SST-96 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-96 triangle engine)

8.37 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)

8.38 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)

8.39 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)

8.40 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-96 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)

8.41 texChipSel Register

The **texShipSel** register selects the TREX for writing texture memory. Valid values range from 0 to 2 (0=TREX0; 1=TREX1; 2=TREX2). Default is 0.

Bit	Description
1:0	Texture Chip Select 2 (0=TREX0; 1=TREX1; 2=TREX2). Default is 0.

8.42 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)

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 T. 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:

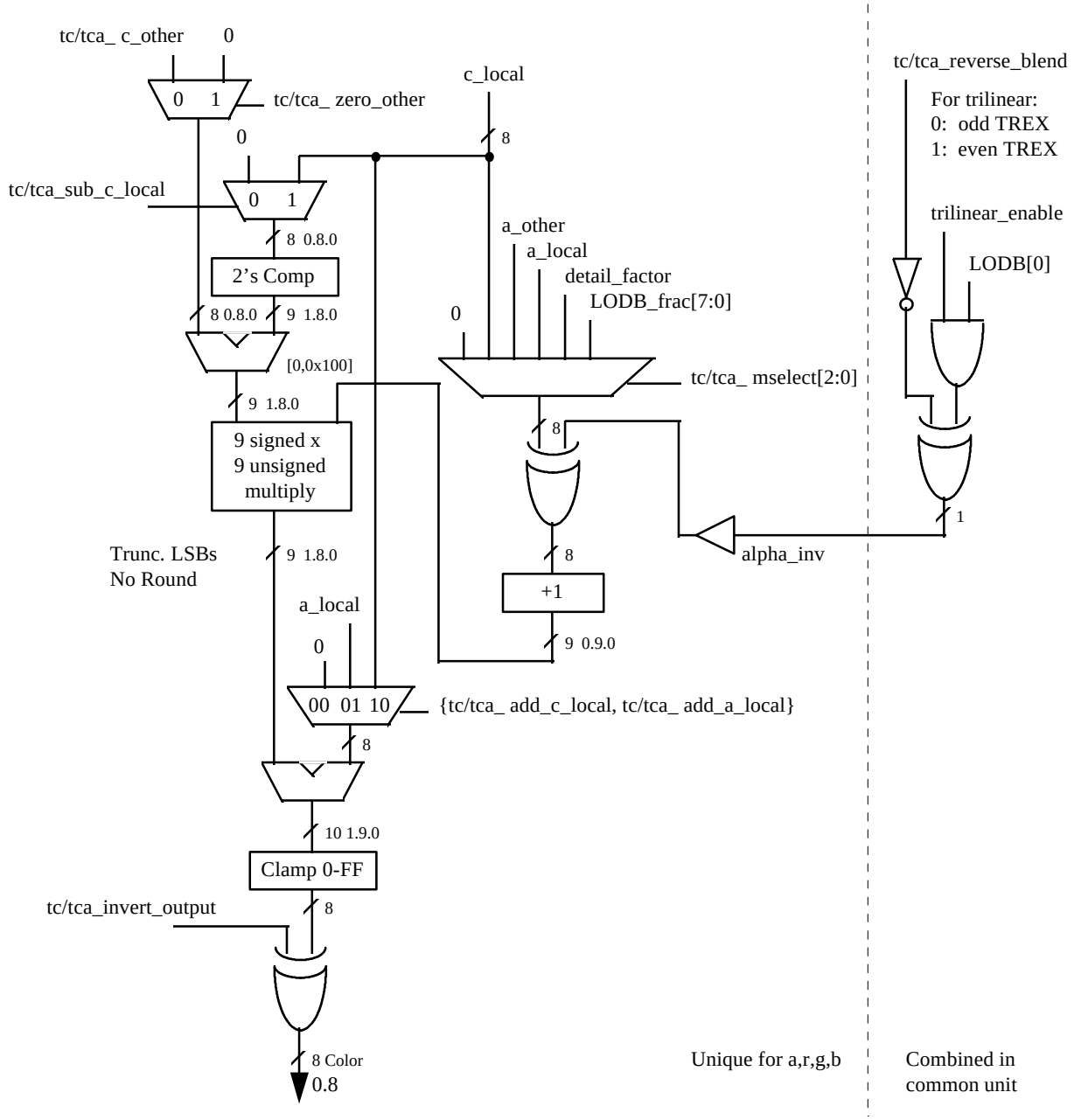
***tforma t* Value**	**Texture format**	**8-bit Alpha**	**8-bit Red**	**8-bit Green**	**8-bit Blue**
0	8-bit RGB (3-3-2)	0xff	{r[2:0],r[2:0],r[2:1]}	{g[2:0],g[2:0],g[2:1]}	{b[1:0],b[1:0],b[1:0],b[1:0]}
1	8-bit YIQ (4-2-2)	See below			
2	8-bit Alpha	a[7:0]	a[7:0]	a[7:0]	a[7:0]
3	8-bit Intensity	0xff	i [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]}
5-7	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)	See below			
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]
15-14	Reserved				

where a, r, g, b, and i(intensity) represent the actual values read from texture memory. The following table shows how 32-bit RGBA texture information is derived from the YIQ texture formats. This is detailed later in the nccTable description.

Texture format	**8-bit Alpha**	**8-bit Red**	**8-bit Green**	**8-bit Blue**
8-bit YIQ (4-2-2)	0xff	ncc _red[7:0]	ncc _green[7:0]	ncc _blue[7:0]
16-bit AYIQ (8-4-2-2)	a[7:0]	ncc _red[7:0]	ncc _green[7:0]	ncc _blue[7:0]

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.

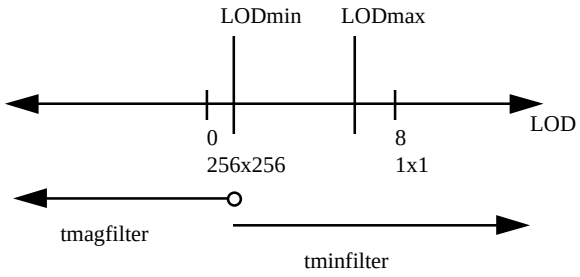
8.43 tLOD Register

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

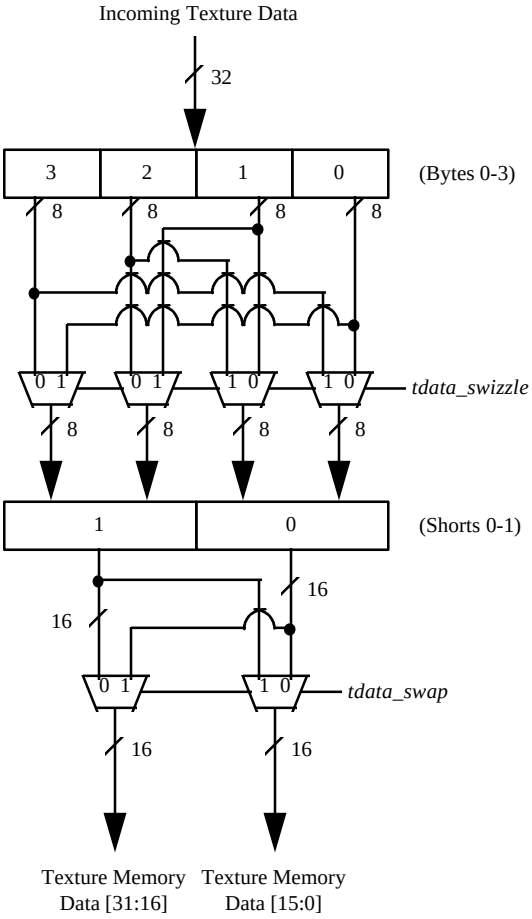
Bit	Name	Description
5:0	*lodmin*	Minimum LOD. (4.2 unsigned)
11:6	*lodmax*	Maximum LOD. (4.2 unsigned)
17:12	*lodbias*	LOD Bias. (4.2 signed)
18	*lod_odd*	LOD odd (0=even, 1=odd)
19	*lod_tsplit*	Texture is Split. (0=texture contains all LOD levels, 1=odd or even levels only, as controlled by *lod_odd*)
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 Ts (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).

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:



8.44 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*))

8.45 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

8.46 TREXInit0 Register

This register is used for hardware initialization and configuration of the TREX chip(s). See TREX spec.

8.47 TREXInit1 Register

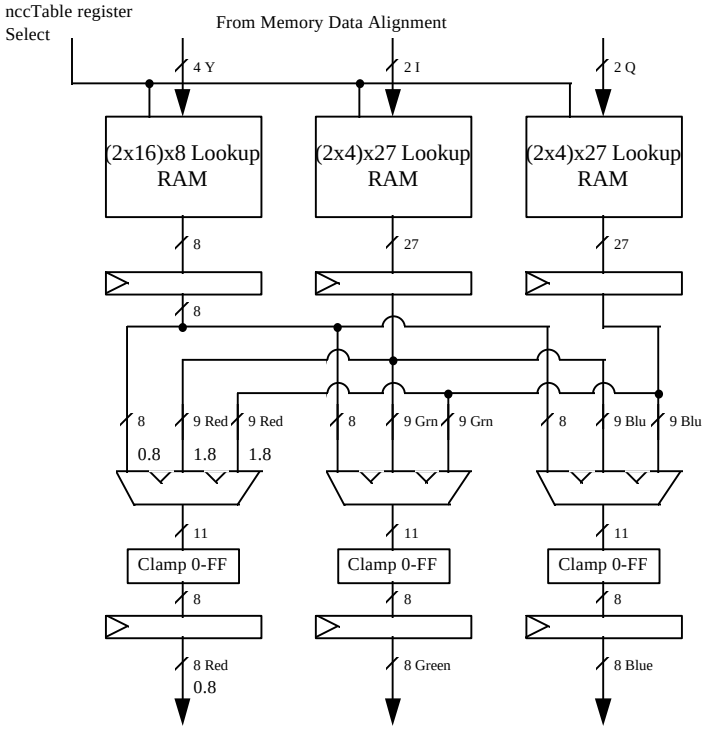
This register is used for hardware initialization and configuration of the TREX chip(s). See TREX spec.

8.48 nccTable0 and nccTable1 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**). 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]}

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


nccTable register Select
From Memory Data Alignment
4 Y
2 I
2 Q
(2x16)x8 Lookup RAM
(2x4)x27 Lookup RAM
(2x4)x27 Lookup RAM
8
27
27
8
8
9 Red
9 Red
8
9 Grn
9 Grn
8
9 Blu
9 Blu
0.8
1.8
1.8
11
11
11
Clamp 0-FF
Clamp 0-FF
Clamp 0-FF
8
8
8
8 Red
0.8
8 Green
8 Blue

9. Changes from SST-1

- **Modifications**

1. Register Address Map

Changed slightly.

2. Initialization Registers

100% modified.

3. CMDFIFO

Software loaded and managed for fullness.

4. Linear Frame Buffer Access

Access performed through 2D engine.

5. Color and Auxiliary Buffers

Data is stored in separate linearly packed buffers. Software manage the regions in both the 2D and 3D engines.

6. Swap Buffer Mechanism

Command required to increment swap pending count. No swap without VSYNC.

7. Video and DRAM Refresh

Performed on 2D chip.

8. Multi-TMU

Requires a register write to select the TMU for texture memory access.

- **Additions**

1. Clipping

Second clip window and inclusive/exclusive option.

2. Window Relative Rendering

Separate base address and X stride programming for both color and auxiliary buffers.

3. Chroma Range

Inclusive/exclusive & union/intersection ranges.

4. 2D BLT/ROP funtions

All functions of the 2D engine with operation ordering controlled by software.

5. Video in a window

Potential video textures if video data is in proper format and data is BLT to texture memory.

6. Stereo Video

Support for Quad buffered stereo.

10. Revision History

1. Version 0.1; Date: 2/9/96
- Original FBIjr version.
2. Version 0.2; Date: 2/13/96
- Modified the mode register addresses for easier group decode.
- Added render/refresh section.
3. Version 0.3; Date: 2/28/96
- Modified Init3&4 to become CMD FIFO specific registers.
- Added Version register - read only with pci vendor number, device number and device version. Additionally moved the block of read only non-status registers down to a 16 double word aligned address.
- Reset fbzMode register back to SST-1 (bit(1) enables clipping globally, bit(10) enables writes to the AUX buffer, bits(16:15) do nothing for SST-96, bit(18) sets AUX buffer to alpha).
- Removed AUX type bit from auxBufferSetup register.
- Added clip enable bits to both clip0 and clip1 registers.
- Modified CMDFIFO text to reflect write pointer which points to last place written and read pointer which points to the next place to read from. As a result, emptiness is determined by write_pointer == read_pointer+1.
- Modified fbijr status register fbijr Idle bit to include all internal activities.
- Added revision history.
4. Version 0.4; Date: 2/28/96
- Fixed parameter register mapping error. The previous ordering was from an out of date SST-1 spec. The current ordering is exactly equal to STT-1 alternate mapping.
5. Version 0.5; Date: 3/7/96
- Added CMDFIFO examples.
- Improved swap buffer tiling description.
- Added Hardware Configuration Section **Error! Reference source not found.**.
- Modified FBIjr Version register to include 4 bit Board Version which is loaded from the Hardware configuration straps.
- Added Texture Memory Load description.
- Improved windowOffset and xStride descriptions.
6. Version 0.6; Date: 3/14/96
- Improved Swap Buffer Command description. Removed swap on sync enable.
- Modified CMDFIFO initialization from read/write pointers to entries-count/read-pointer protocol.
- Added chromaRange register.
- Changed register address map to 1MB for 4MP puma mode.
- Added multiple TREX support. Changed register address map chip field to 4 bits. Added textureChipSel register (00=TREX0, 01=TREX1, 10=TREX2).
7. Version 0.7; Date: 3/18/96
- Added fbzMode[20] to select the source depth value used for depth buffering.
- Added SST-1 changes section to 1.0.
- Added windowed 320x200 3D resolutions.
8. Version 0.8; Date: 4/4/96
- Moved MEMFIFO fields to separate initialization register. Now there are 6 initialization registers.
- Changed MEMFIFO low/high free space registers to 6 bits each.
- Changed CMDFIFO base/size register change top/bottom registers.
- Added CMDFIFO entries threshold register.
- Changed fbijrStatus register CMDFIFO and MEMFIFO fullness fields.
- Changed Color and Auxiliary buffer setup registers. Added X stride fields.

- Eliminated the Offset and Stride registers. These features are now in the buffer setup registers.
- Added CMDFIFO and MEMFIFO overflow error bits to the status register.

9. Version 0.9; Date: 8/22/96

- Removed performance tiling/packing information.
- General cleaning.