**June 14, 1999**


# Voodoo3®

## HIGH-PERFORMANCE

## GRAPHICS ENGINE

## FOR

## 3D GAME ACCELERATION

Programming Guide: Revision 1.4

June 14, 1999



**3Dfx Interactive, Inc.**
4435 Fortran Drive
San Jose CA 95134
Phone: (408) 935-4400
Fax: (408) 935-4424

**Notice:**
3Dfx Interactive, Inc. has made best efforts to ensure that the information contained in this document is accurate and reliable. The information is subject to change without notice. No responsibility is assumed by 3Dfx Interactive, Inc. for the use of this information, nor for infringements of patents or the rights of third parties. This document is the property of 3Dfx Interactive, Inc. and implies no license under patents, copyrights, or trade secrets.

**Trademarks:**
All trademarks are the property of their respective owners.



**Programming Examples:**
Sample programs in this book may be adapted, in part or in whole, for use in any equipment incorporating Voodoo3. Use in any equipment not incorporating Voodoo3 is expressly prohibited.

**Proprietary Information:**
This document contains proprietary information of 3Dfx Interactive, Inc., and its receipt or possession does not convey any rights to reproduce, disclose its contents, or to manufacture, use or sell anything it may describe. Reproduction, disclosure or use without specific written authorization of 3Dfx Interactive, Inc., is strictly forbidden.

# Table of Contents

# List of Tables

# List of Figures

# 1 Introduction

## 1.1 Scope of Document

This is the programming guide for Voodoo3. This document includes a device overview, register descriptions, programming notes, and examples (to be supplied in the fullness of time).

## 1.2 Document History

### Table 1.1 Document History

| Revision | Dated | Description |
|---|---|---|
| 0.3 | Jan 18, 99 | Corrected bit assignments in vidStatusCurrentLine |
| 0.4 | Feb 16, 99 | Corrected pllCtrl2 description, PLL_BYPASS strap |
| 1.0 | Feb 23, 99 | Updated version, date, posted on southpark |
| 1.1 | March 1, 99 | Fixed intrCtrl[4] |
| 1.2 | May 5, 99 | Fixed SRX, SRn Port in Table 3.1 |
| 1.3 | June 8, 99 | CMD FIFO Packet 1, Pallete Load, Texture Memory Access |
| 1.4 | June 14, 99 | Add to description of fbzMode[19, 11] |

## 1.3 Devices Covered

This document covers the production version(s) of Voodoo3.

## 1.4 Audience

This document is tailored to a knowledgeable audience. It is assumed that the reader is familiar with assembly language programming of Pentium® and has a good foundation in computer-generated graphics, especially 3D.
Programmers intending to use Voodoo3 should have experience with AGP graphics devices. Voodoo3 is intended for use with Windows GUI and Glide. Experience with 2D accelerators, 3D accelerators, and video capture applications will be useful. Some interfaces (for example, VMI host interface) are directly programmed with bit significant protocols (once called ‘bit-bopping’).
It is the intent of 3Dfx Interactive, Inc. that this guide eventually contain instructive examples. Programmers are encouraged to study them and adapt them for use in any equipment incorporating Voodoo3.

## 1.5 Conventions

### 1.5.1 Acronyms

The first appearance of each TLA (Three Letter Acronym) is followed immediately by the definition in parentheses.

### 1.5.2 Number Base

Hexadecimal (base 16) numbers use upper case letters ABCDEF. Hexadecimal numbers have a prepended ‘0x’ or an appended ‘h’. The following are examples of hexadecimal numbers: 0x00, 0x3DF, 3DFh, 0x1234, 0x2A.
Decimal (base 10) numbers have no special indicator. The following are examples of decimal numbers: 1234, 2380, 42.
Binary (base 2) have an appended ‘b’. The following are examples of binary numbers: 00b, 01b, 101010b.

Octal (base 8) numbers are not used in this document.
The value zero is often written as 0, without any quotes and without indication as to size or base.

### 1.5.3 Number/Color Formats

The following table shows the conventions used in this book to indicate number formats.

**Table 1.2 Number Formats**

| Format | Description | Example(s) |
|---|---|---|
| i.f | Floating Point Number | 0.19 indicates 19-bit fraction with no integer part |
| | | |
| | | |
| | | |
| | | |

### 1.5.4 Object Grouping

Objects that are grouped together are listed in descending order. A range is indicated with surrounding square brackets and a colon between the highest and the lowest in the range. A[7:0] means A7, A6, A5, A4, A3, A2, A1, A0. This convention is used for bits in a register (for example, CR2[7:0]) and for signal pins (for example, PCI_AD[31:0]).

### 1.5.5 Abbreviations

The following abbreviations are used in this document.

**Table 1.3 Abbreviations**

| Abbreviation | Meaning | Note |
|---|---|---|
| Kbyte | 1024 bytes | |
| Mbyte | 1,048,576 bytes | 1024 Kbytes |
| Gbyte | 1,073,741,824 bytes | 1024 Mbytes |
| Hz | Hertz | frequency |
| kHz | 1000 Hertz | |
| MHz | 1,000,000 Hertz | |
| ms | $10^{-3}$ second | period |
| us | $10^{-6}$ second | |
| ns | $10^{-9}$ second | |
| mA | $10^{-3}$ Ampere | current |
| uA | $10^{-6}$ Ampere | |

**Table 1.3 Abbreviations *(cont.)***

| Abbreviation | Meaning | Note |
|---|---|---|
| uF | $10^{-6}$ Farad | capacitance |
| pF | $10^{-12}$ Farad | |
| tbd, na | To Be Determined, Not Available | used interchangeably |
| Mpixel | 1,000,000 pixels | |
| wrt | with respect to | 3D delta register descriptions |

# 2 Product Overview

## 2.1 Introduction

Voodoo3 incorporates leading-edge 3D graphics and extremely fast 128-bit Windows® GUI/Video Acceleration into a single chip.

### 2.1.1 Voodoo Graphics Compatibility

Since Voodoo3 is upward compatible with Voodoo® 3D, hundreds of 3D titles that have been optimized for acceleration on Voodoo Graphics, Voodoo Rush, Voodoo$^2$, and Voodoo Banshee will run on Voodoo3 without modification. Of course, to take advantage of the Voodoo3 enhanced features, it will be necessary to make changes.

### 2.1.2 3D Performance and Quality

3Dfx Interactive, Inc. is the industry leader in delivering 3D technology for the PC consumer market. Voodoo3 - 333 will continue this heritage, delivering 333 Mtexels/sec and over 6 million triangles per second single-cycle multi-texturing 3D performance[1]. The design philosophy behind all products of 3Dfx Interactive, Inc. is to provide advanced 3D features with the universal requirement of all serious game developers: **no degradation in performance and quality**. As an example, Voodoo3 provides per-pixel level-of-detail MIP mapping and per-pixel atmospheric effects such as fog and haze. Other solutions that provide these features at all do so on a per-polygon basis, yielding an inferior image.

### 2.1.3 Optimized for Pentium® II and AGP-2X Platform

Voodoo3 is the only solution to fully exploit the processing power the Pentium® II, including direct hardware handling of out-of-order writes. From the very beginning, Voodoo3 was designed to maximize the performance of the Pentium Pro and Pentium II I/O architecture. The AGP interface is tuned for optimal 3D performance, and supports sideband addressing for very fast texture downloading and full 2X 133 MHz AGP bus operation.

### 2.1.4 Windows® GUI/Video Acceleration

Voodoo3 - 333 is a 166 MHz (125 for the -250 product) single-cycle GUI accelerator with 128-bit frame buffer interface. Even the VGA core is 128 bits. The design philosophy has been to implement the Microsoft GDI (Graphics Device Interface) in hardware for outstanding windows acceleration. Voodoo3 supports the new features of Windows98 (for example, multi-monitor support) and is PC99 compliant.

### 2.1.5 DVD Acceleration

The video architecture of Voodoo3 is optimized for software DVD acceleration. This optimization includes large FIFOs, YUV 4:2:0 planar to packed pixel conversion with AGP bus-mastering, automatic double-buffering, and alpha blending for sub-picture support.

## 2.2 Feature List

### 2.2.1 General Features

- Two texels per clock
- Fully integrated 128-bit VGA/2D/3D/Video Accelerator
- 2X AGP with sideband addressing
- Fully software-compatible with 3Dfx Voodoo Banshee
- Floating point depth buffer (W buffer)
- Ultimate 3D experience with 333 Mtexels/sec and 6 million triangles/sec for - 333 product,
- 250 Mtexels and 4 million triangles/sec for - 250 product
- Hardware support for Out-of-order writes

1. The -250 product will deliver 250 Mtexel/sec and over 4 million triangles per second.

- Hardware DVD acceleration
- Digital video output for NTSC and PAL TV-out support
- DFP or VESA FPDI LCD support through external ASIC
- Full VMI (includes the host port) for Hardware DVD decoding or digital video capture
- HDTV resolution of 2132 x 1600 at 60 Hz with 350 MHz (-333 product) RAMDAC
- Supports 4-, 8-, 16-Mbyte SGRAM or 16-Mbyte SDRAM frame buffer
- PC99 rev 1.0 compliant
- VESA DDC2B support

### 2.2.2 2D Acceleration

- 333 MHz (250 for -250 product) single-cycle 128-bit Windows GUI acceleration
- Fully-featured 128-bit BitBlt Engine: Windows GDI in hardware
- Acceleration for Bresenham line draw, 2-edge polygon fill, scissor/rectangle clippers, 256 ROPs
- Source and destination chroma-keying for DirectDraw
- SGRAM color expansion support and single-cycle block writes
- Accelerated 8-, 16-, 24 (packed)-, 32-bpp modes

### 2.2.3 3D Acceleration

- Dual texture units: Two texels per-pixel per-clock
- Full hardware setup of triangle parameters
- Support for multi-triangle strips and fans
- 16-bit integer and floating-point Z-buffering with biasing
- Transparency and chroma-key with dedicated color mask
- Alpha blending of source and destination pixels
- Sub-pixel and sub-texel correction to 0.4 x 0.4 resolution
- 24-bit color dithering to native 16-bit RGB
- Per-pixel atmospheric fog with programmable fog zones
- Full-scene polygon-based edge anti-aliasing
- Dynamic environment mapping
- Perspective correct (true divide-per-pixel) 3D texture mapping and Gouraud shading
- Single-cycle bump mapping
- Single-cycle Trilinear Mip-mapping
- Anisotropic filtering
- True per-pixel LOD (level-of-detail) MIP mapping with biasing and clamping
- RGB modulation combines textures and shaded pixels
- Texture compositing for multi-texture special effects
- Support of 14 texture map formats
- 8-bit paletted textures with full bilinear filtering
- Texture compression through narrow-channel YAB format

### 2.2.4 Video Acceleration

- Multiple video window support
- Bilinear horizontal and vertical filtering
- YUV 4:2:2 and YUV 4:2:0 planar support
- De-interlacing using ‘bob’ and ‘weave’
- Automatic page flipping using VBI (Vertical Blanking Interval) for smooth motion video
- Triple 512 x 8 color lookup tables with separate gamma correction for video and graphics
- Separate gamma correction for video and graphics

### 2.2.5 Host Interface

- AGP 2X interface includes optimized support for sideband addressing
- PCI v2.1 bus interface supports 33 MHz and 66 MHz
- FIFO optimized for high speed bursting of geometry and texture data
- Optimized for Pentium II I/O architecture; out-of-order writes handled in hardware

- Bi-endian byte-ordering support

### 2.2.6 Memory System

- Advanced architecture with 3.0 Gbyte/sec (2.7 Gbyte/sec for -250 product) memory bandwidth
- 4 - 16 Mbytes of 166 MHz (125 for -250 product) and faster SGRAM/SDRAM

### 2.2.7 Process and Package Technology

- Custom IC fabricated in 0.25 micron, 5 metal layer CMOS
- 352-lead (plus 100 thermal ball) 35mm PBGA package
- 2.5 volt power with PCI and 5-volt tolerant I/O
- Built-in Iddq, CRC, and Parametric NAND tree for testability

### 2.2.8 Software

- Full BIOS and driver compatibility with Voodoo Banshee for a mature, robust solution
- Windows95, Windows98, WindowNT4.0 device drivers
- Extensive 3D API support including Glide 2.X and 3.X, OpenGL ICD, Microsoft D3D
- Support for TV encoders: Chrontel 7004, Brooktree 868/9
- Software DVD support: Quadrant

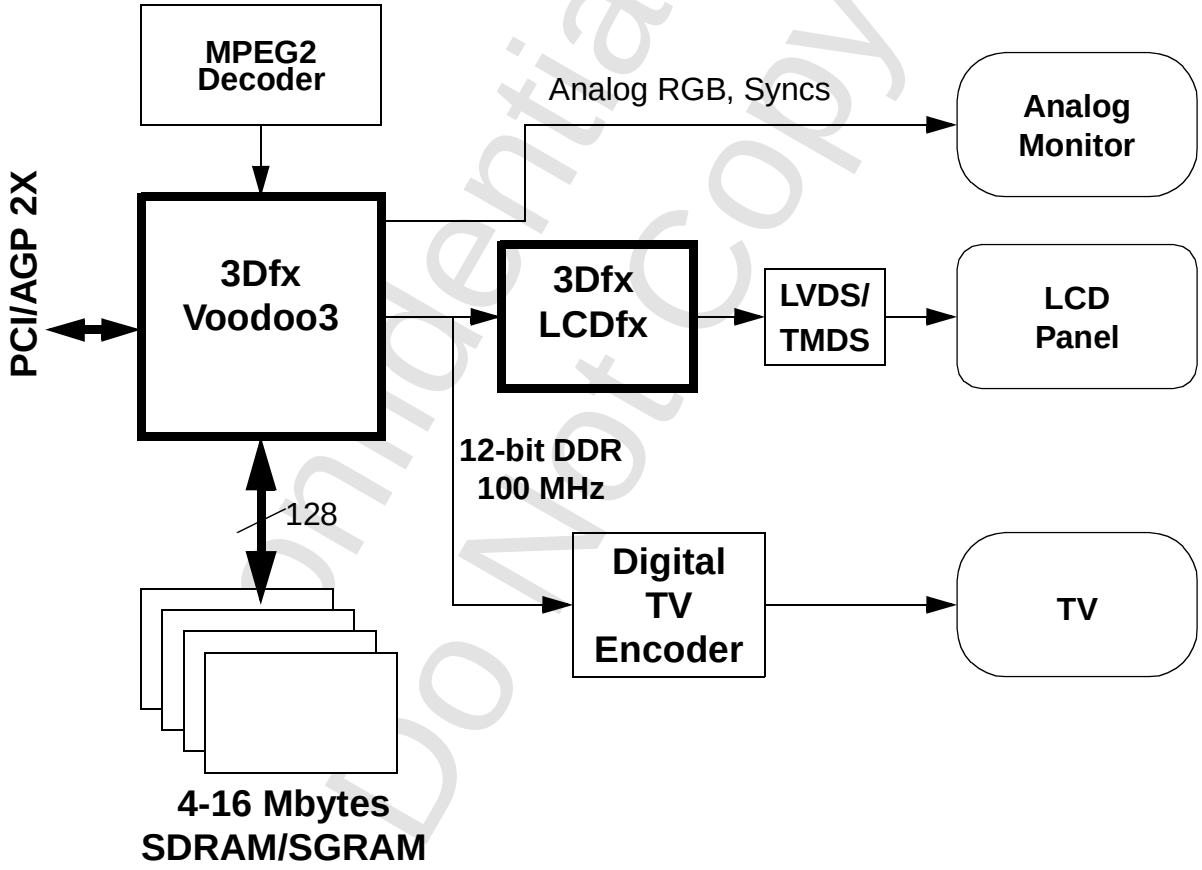



**Figure 2.1 System Block Diagram**

# 3 VGA Core Registers

## 3.1 Overview

Voodoo3 contains an integrated VGA core that is 100% compatible with the IBM PS/2 model 70. While the core is optimized for 128-bit memory transfers, compatibility with legacy software is not compromised. The VGA core supports all PC99 revision 1.0 requirements.
Table 3.1 contains a summary of the VGA registers. The link is a clickable link to the detailed description.
Unless otherwise noted, all non-reserved fields in all registers in the VGA core are read/write.
CR1A through CR1F are extensions to the IBM VGA core.
When the VGA core is configured for monochrome operations, registers at 0x3Dx move to 0x3Bx. See the CRTC address bit in Miscellaneous Output (Section 3.2.1)

### Table 3.1 VGA Core Register Summary

| Register Name | Mnemonic | Port | Index | Link |
|---|---|---|---|---|
| Miscellaneous Output (Write) | MISC | 0x3C2 | - | Section 3.2.1 |
| Miscellaneous Output (Read) | | 0x3CC | - | Section 3.2.1 |
| Feature Control (Write) | FC | 0x3DA | - | Section 3.2.2 |
| Feature Control (Read) | | 0x3CA | - | Section 3.2.2 |
| Input Status Register 0 | FEAT | 0x3C2 | - | Section 3.2.3 |
| Input Status Register 1 | STAT | 0x3DA | - | Section 3.2.4 |
| Motherboard Enable | - | 0x03C3 | - | Section 3.2.5 |
| Adapter Enable | - | 0x46E8 | - | Section 3.2.6 |
| Subsystem Enable | - | 0x0102 | - | Section 3.2.7 |
| CRTC Index | CRX | 0x3D4 | - | Section 3.3.1 |
| CRTC HorTotal | CR0 | 0x3D5 | 0x00 | Section 3.3.2 |
| CRTC HorDisEnEnd | CR1 | 0x3D5 | 0x01 | Section 3.3.3 |
| CRTC HorBlankStart | CR2 | 0x3D5 | 0x02 | Section 3.3.4 |
| CRTC HorBlankEnd | CR3 | 0x3D5 | 0x03 | Section 3.3.5 |
| CRTC HorSyncStart | CR4 | 0x3D5 | 0x04 | Section 3.3.6 |
| CRTC HorSyncEnd | CR5 | 0x3D5 | 0x05 | Section 3.3.7 |
| CRTC VertTotal | CR6 | 0x3D5 | 0x06 | Section 3.3.8 |
| CRTC Overflow | CR7 | 0x3D5 | 0x07 | Section 3.3.9 |
| CRTC PresetRowScan | CR8 | 0x3D5 | 0x08 | Section 3.3.10 |
| CRTC MaxScanLine | CR9 | 0x3D5 | 0x09 | Section 3.3.11 |
| CRTC CursorStart | CRA | 0x3D5 | 0x0A | Section 3.3.12 |

**Table 3.1 VGA Core Register Summary *(cont.)***

| Register Name | Mnemonic | Port | Index | Link |
|---|---|---|---|---|
| CRTC CursorEnd | CRB | 0x3D5 | 0x0B | Section 3.3.13 |
| CRTC ScreenStartHigh | CRC | 0x3D5 | 0x0C | Section 3.3.14 |
| CRTC ScreenStartLow | CRD | 0x3D5 | 0x0D | Section 3.3.15 |
| CRTC CursorLocHigh | CRE | 0x3D5 | 0x0E | Section 3.3.16 |
| CRTC CursorLocLow | CRF | 0x3D5 | 0x0F | Section 3.3.17 |
| CRTC VertSyncStart | CR10 | 0x3D5 | 0x10 | Section 3.3.18 |
| CRTC VertSyncEnd | CR11 | 0x3D5 | 0x11 | Section 3.3.19 |
| CRTC VertDispEnEnd | CR12 | 0x3D5 | 0x12 | Section 3.3.20 |
| CRTC Offset | CR13 | 0x3D5 | 0x13 | Section 3.3.21 |
| CRTC Underline Location | CR14 | 0x3D5 | 0x14 | Section 3.3.22 |
| CRTC VertBlankStart | CR15 | 0x3D5 | 0x15 | Section 3.3.23 |
| CRTC VertBlankEnd | CR16 | 0x3D5 | 0x16 | Section 3.3.24 |
| CRTC Mode Control | CR17 | 0x3D5 | 0x17 | Section 3.3.25 |
| CRTC Line Compare | CR18 | 0x3D5 | 0x18 | Section 3.3.26 |
| CRTC HorExtensions | CR1A | 0x3D5 | 0x1A | Section 3.3.27 |
| CRTC VertExtensions | CR1B | 0x3D5 | 0x1B | Section 3.3.28 |
| CRTC PCIConfig | CR1C | 0x3D5 | 0x1C | Section 3.3.29 |
| CRTC Scratch1D | CR1D | 0x3D5 | 0x1D | Section 3.3.30 |
| CRTC Scratch1E | CR1E | 0x3D5 | 0x1E | Section 3.3.30 |
| CRTC Scratch1F | CR1F | 0x3D5 | 0x1F | Section 3.3.30 |
| CRTC VertCountPreHigh | CR20 | 0x3D5 | 0x20 | Section 3.3.31 |
| CRTC VertCountPreLow | CR21 | 0x3D5 | 0x21 | Section 3.3.32 |
| CRTC Graphics Latch | CR22 | 0x3D5 | 0x22 | Section 3.3.33 |
| CRTC ATTControllerToggle | CR24 | 0x3D5 | 0x24 | Section 3.3.34 |
| CRTC ATTControllerIndex | CR26 | 0x3D5 | 0x26 | Section 3.3.35 |
| Graphics Controller Index | GRX | 0x3CE | - | Section 3.4.1 |
| Graphics Controller SetReset | GR0 | 0x3CF | 0x00 | Section 3.4.2 |
| Graphics Controller SetReset Enable | GR1 | 0x3CF | 0x01 | Section 3.4.3 |

**Table 3.1 VGA Core Register Summary *(cont.)***

| Register Name | Mnemonic | Port | Index | Link |
|---|---|---|---|---|
| Graphics Controller Color Compare | GR2 | 0x3CF | 0x02 | Section 3.4.4 |
| Graphics Controller Rotate | GR3 | 0x3CF | 0x03 | Section 3.4.5 |
| Graphics Controller ReadMap Select | GR4 | 0x3CF | 0x04 | Section 3.4.6 |
| Graphics Controller Mode | GR5 | 0x3CF | 0x05 | Section 3.4.7 |
| Graphics Controller Miscellaneous | GR6 | 0x3CF | 0x06 | Section 3.4.8 |
| Graphics Controller Color Dont Care | GR7 | 0x3CF | 0x07 | Section 3.4.9 |
| Graphics Controller Bit Mask | GR8 | 0x3CF | 0x08 | Section 3.4.10 |
| Attribute Controller Index | ARX | 0x3C0/1 | - | Section 3.5.1 |
| Attribute Controller Palette | AR0-ARF | 0x3C0/1 | 0x00-0x0F | Section 3.5.2 |
| Attribute Controller Mode | AR10 | 0x3C0/1 | 0x10 | Section 3.5.3 |
| Attribute Controller Overscan Color | AR11 | 0x3C0/1 | 0x11 | Section 3.5.4 |
| Attribute Cont. Color Plane Enable | AR12 | 0x3C0/1 | 0x12 | Section 3.5.5 |
| Attribute Controller Pixel Panning | AR13 | 0x3C0/1 | 0x13 | Section 3.5.6 |
| Attribute Controller Color Select | AR14 | 0x3C0/1 | 0x14 | Section 3.5.7 |
| Sequencer Index | SRX | 0x3C4 | - | Section 3.6.1 |
| Sequencer Reset | SR0 | 0x3C5 | 0x00 | Section 3.6.2 |
| Sequencer Clocking Mode | SR1 | 0x3C5 | 0x01 | Section 3.6.3 |
| Sequencer Plane Mask | SR2 | 0x3C5 | 0x02 | Section 3.6.4 |
| Sequencer Character Map Select | SR3 | 0x3C5 | 0x03 | Section 3.6.5 |
| Sequencer Memory Mode | SR4 | 0x3C5 | 0x04 | Section 3.6.6 |
| RAMDAC Pixel Mask | - | 0x3C6 | - | Section 3.7.1 |
| RAMDAC Read Address | - | 0x3C7 | - | Section 3.7.2 |
| RAMDAC Read Status | - | 0x3C7 | - | Section 3.7.3 |
| RAMDAC Write Address | - | 0x3C8 | - | Section 3.7.4 |
| RAMDAC Data | - | -x3C9 | - | Section 3.7.5 |

## 3.2 General Registers

### 3.2.1 Miscellaneous Output

This is one of the heritage VGA registers. This register is written at 0x3C2 and read at 0x3CC.

**Bit** — **Description**

7 — **Vertical Sync Polarity:** When this bit is 0, VSYNC is a normally low signal, going high to indicate the beginning of sync time. When this bit is '1', VSYNC is a normally high signal, going low to indicate the beginning of sync time.

6 — **Horizontal Sync Polarity:** When this bit is 0, HSYNC is a normally low signal, going high to indicate the beginning of sync time. When this bit is '1', HSYNC is a normally high signal, going low to indicate the beginning of sync time. For some heritage monitors, the polarities of HSYNC and VSYNC indicate the number of active scan lines per frame. Fortunately, nearly all of these monitors are in third world countries by now.

| HSYNC | VSYNC | Active Lines |
|---|---|---|
| 0 (Positive) | 0 (Positive) | Reserved |
| 0 (Positive) | 1 (Negative) | 400 |
| 1 (Negative) | 0 (Positive) | 350 |
| 1 (Negative) | 1 (Negative) | 480 |

5 — **Page Select:** This bit is meaningful only in Even/Odd mode. It selects the high 64K bank if '1'.

4 — **Reserved:** Reserved bits must be written as 0 for upward compatibility.

3:2 — **Clock Select:** This field selects the video clock frequency.

| ClockSelect | Frequency |
|---|---|
| 00b | 25.175 MHz |
| 01b | 28.322 MHz |
| 10b | 50.000 MHz |
| 11b | Programmable PLL |

### 3.2.1 Miscellaneous Output *(cont)*

| Bit | Description |
|---|---|
| 1 | **RAM Enable:** If this bit is 0, Voodoo3 does not respond to any access to the standard VGA memory addresses. If this bit is '1', Voodoo3 responds normally. |
| 0 | **CRTC Addresses:** This bit controls the I/O addresses of the CRT controller. |

| CRTCAddress | ISR/FC | CRTC Index | CRTC Data | Mode |
|---|---|---|---|---|
| 0 | 0x3BA | 0x3B4 | 0x3B5 | Mono |
| 1 | 0x3DA | 0x3D4 | 0x3D5 | Color |

### 3.2.2 FC: Feature Control

This is another of the heritage VGA registers.

| Bit | Description |
|---|---|
| 7:4 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 3 | **VSYNC Control:** If this bit is '1', VSYNC is ORed with display enable. |
| 2 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 1:0 | **Feature Control:** These two bits do not actually connect to anything. |

### 3.2.3 FEAT: Input Status Register 0

This register is read-only.

| Bit | Description |
|---|---|
| 7 | **Interrupt Status:** When this bit is '1', an interrupt (specifically a VGA interrupt) is pending. |
| 6:5 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 4 | **DAC Sensing:** This bit reflects the state of the DAC monitor sense logic. |
| 3:0 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |

### 3.2.4 STAT: Input Status Register 1

This is a read-only register. Reading this register has the side effect of forcing the Attribute Controller Toggle to 0 (address).

| Bit | Description |
|---|---|
| 7:6 | **Reserved:** These bits return the value 0. |
| 5:4 | **Display Status:** These two bits return two of eight pixel data outputs of the Attribute Controller. See AR12[5:4] (Section 3.5.5). |
| 3 | **Vertical Sync:** If this bit is '1', vertical sync is active. |
| 2:1 | **Reserved:** This field returns the value 10b. |
| 0 | **Display Enable:** If this bit is '1', vertical or horizontal blanking is active. |

### 3.2.5 Motherboard Enable (0x03C3)

This is one of two enable registers. Bit zero at this address is the same physical flip-flop as 0x46E8[3].

| Bit | Description |
|---|---|
| 7:1 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. These bits will read back as '1's. |
| 0 | **Video Subsystem Enable:** This register was inplemented only on PS/2 motherboard VGAs (and on a number of clone VGA controllers). When this bit is 0, all I/O and memory accesses to the video subsystem are disabled except for accesses to ports 0x0102 and 0x03C3. When both this bit and Subsystem Enable[0] are '1's, the video subsystem is enabled and will respond normally to I/O and memory accesses. |

### 3.2.6 Adapter Enable (0x46E8)

This is the second enable register. Bit three at this address is the same physical flip-flop as 0x03C3[0].

| Bit | Description |
|---|---|
| 7:5 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 4 | **Setup Mode:** When this bit is '1', the adapter is in setup mode and accesses to 0x0102 are allowed. When this bit is 0, accesses to 0x0102 are ignored. |
| 3 | **Video Subsystem Enable:** When this bit is 0, all I/O and memory accesses to the video subsystem are disabled except for accesses to ports 0x0102 and 0x46E8. When both this bit and Subsystem Enable[0] are '1's, the video subsystem is enabled and will respond normally to I/O and memory accesses. |
| 2:0 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |

### 3.2.7 Subsystem Enable (0x0102)

This is the other enable bit. This register is available only when the video subsystem enable is 0.

| Bit | Description |
|---|---|
| 7:1 | Reserved: Reserved bits must be written as 0 for upward compatibility. |
| 0 | Global Subsystem Enable: When this bit is 0, accesses to I/O and memory are ignored regardless of the setting of any other bit. When this bit and the Video System Enable bit are both '1', the accesses to I/O and memory take place normally. This register is visible only when the chip is in setup mode. |

## 3.3 CRT Controller Registers

### 3.3.1 CRX: CRTC Index

This register specifies the register in the CRTC block of the VGA core to be accessed by the next I/O read or I/O write to 0x3D5.

| Bit | Description |
|---|---|
| 7:6 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 5:0 | **CRX:** This field is the address of the register in the CRTC block to be accessed by the next I/O read or I/O write to 0x3D5. |

Table 3.2 shows the register bits for each CRTC parameter. So as to make the table fit in the available space, only the register offsets are shown. Thus, the entry 0[3:0] means CR0[3:0]. CR1A and CR1B are extensions to the VGA core.
Figure 4.1 is the standard CRTC timing diagram.

**Table 3.2 CRTC Register Summary**

| Parameter | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3:0 |
|---|---|---|---|---|---|---|---|---|
| HorTotal | - | - | 1A[0] | 0[7] | 0[6] | 0[5] | 0[4] | 0[3:0] |
| HorDispEnEnd | - | - | 1A[2] | 1[7] | 1[6] | 1[5] | 1[4] | 1[3:0] |
| HorBlankStart | - | - | 1A[4] | 2[7] | 2[6] | 2[5] | 2[4] | 2[3:0] |
| HorBlankEnd | - | - | - | - | 1A[5] | 5[7] | 3[4] | 3[3:0] |
| HorSyncStart | - | - | 1A[6] | 4[7] | 4[6] | 4[5] | 4[4] | 4[3:0] |
| HorSyncEnd | - | - | - | - | - | 1A[7] | 5[4] | 5[3:0] |
| VertTotal | 1B[0] | 7[5] | 7[0] | 6[7] | 6[6] | 6[5] | 6[4] | 6[4:0] |
| VertDispEnEnd | 1B[2] | 7[6] | 7[1] | 12[7] | 12[6] | 12[5] | 12[4] | 12[3:0] |
| VertBlankStart | 1B[4] | 9[5] | 7[3] | 15[7] | 15[6] | 15[5] | 15[4] | 15[3:0] |
| VertBlankEnd | - | - | - | 16[7] | 16[6] | 16[5] | 16[4] | 16[3:0] |
| VertSyncStart | 1B[6] | 7[7] | 7[2] | 10[7] | 10[6] | 10[5] | 10[4] | 10[3:0] |
| VertSyncEnd | - | - | - | - | - | - | - | 11[3:0] |
| LineCompare | - | 9[6] | 7[4] | 18[7] | 18[6] | 18[5] | 18[4] | 18[3:0] |

### 3.3.2 CR0: CRTC Horizontal Total

This register contains the low order eight bits of the total width of the display.

| Bit | Description |
|---|---|
| 7:0 | **HorTotal[7:0]:** This field contains the low order eight bits of the total width of the display, expressed in character clocks, minus five. This includes the retrace time. This field is extended with CR1A[0] to a total of nine bits if vgaInit0[6] is ‘1’. |

### 3.3.3 CR1: CRTC Horizontal Display Enable End

This register contains the low order eight bits of the total number of horizontal characters displayed in each scan line.

| Bit | Description |
|---|---|
| 7:0 | **HorDispEnEnd[7:0]:** This field contains the low order eight bits of the number of visible characters on the screen, minus one. This count does not include the right border, if any. This field is extended with CR1A[2] to a total of nine bits if vgaInit0[6] is '1'. |

### 3.3.4 CR2: CRTC Horizontal Blanking Start

This register contains the low order eight bits of the total number of horizontal characters displayed before horizontal blanking is begun.

| Bit | Description |
|---|---|
| 7:0 | **HorBlankStart[7:0]:** This field contains the low order eight bits of the number of visible characters on the screen, including the right border. This field is extended with CR1A[4] to a total of nine bits if vgaInit0[6] is '1'. |

### 3.3.5 CR3: CRTC Horizontal Blanking End

This register contains the low order five bits of the field that specifies when horizontal blanking ends (and the border for the next scan line is begun). This register also contains the compatibility read bit.

| Bit | Description |
|---|---|
| 7 | **CompatibilityRead:** When this bit is '1', the Vertical Sync Start (CR10) and Vertical Sync End (CR11) registers are readable. When this bit is '0', CR10 and CR11 are not readable (but can still be written). This bit is retained for compatibility with legacy VGA. |
| 6:5 | **DisplayEnableSkew:** This field specifies the number of character clocks that horizontal display enable is delayed with respect to horizontal sync. This field is typically programmed to 01b. If this field is programmed too low, the left-most character is repeated. If this field is programmed too high, one or more characters are not displayed at the left. |
| 4:0 | **HorBlankEnd[4:0]:** This field contains the low order five bits of the HorBlankEnd value. This field is extended with CR5[7] to six bits. This is extended again with CR1A[5] for a total of seven bits if vgaInit0[6] is '1'. This value is compared to the low order six or seven bits of the horizontal counter to determine when blanking is to end. The value for this field can be calculated by adding the desired blanking interval to the HorBlankStart value (and retaining only the low order bits). |

### 3.3.6 CR4: CRTC Horizontal Sync Start

This register contains the low order eight bits of the character count at which horizontal sync is to become active.

| Bit | Description |
|---|---|
| 7:0 | **HorSyncStart[7:0]:** This field contains the low order eight bits of the character count at which horizontal sync is to become active. This field is extended with CR1A[6] to a total of nine bits if vgaInit0[6] is '1'. |

### 3.3.7 CR5: CRTC Horizontal Sync End

This register contains the low order five bits of the field that specifies when horizontal Sync ends. This register also contains one bit of the Horizontal Blank End value.

| Bit | Description |
|---|---|
| 7 | **HorBlankEnd[5]:** This is bit five of the HorBlankEnd field. HorBlankEnd[4:0] are in CR3. Bit six is in CR1A[5] if vgaInit0[6] is '1'. |
| 6:5 | **HorizontalSyncSkew:** This field specifies the number of character clocks that horizontal sync is delayed from the position indicated in the Horizontal Sync Start field. This is necessary in some modes to allow internal timing signals triggered from Horizontal Sync Start to begin prior to Display Enable. |
| 4:0 | **HorSyncEnd[4:0]:** This field contains the low order five bits of the HorBlankEnd value. This field is extended with CR1A[7] for a total of six bits if vgaInit0[6] is '1'. This value is compared to the low order five or six bits of the horizontal counter to determine when sync is to end. The value for this field can be calculated by adding the desired sync interval to the HorSyncStart value (and retaining only the low order bits). Hsync must always end during the horizontal blanking period. |

### 3.3.8 CR6: CRTC Vertical Total

This register contains the low order eight bits of the total number of scan lines in a display frame.

| Bit | Description |
|---|---|
| 7:0 | **VertTotal[7:0]:** This field contains the low order eight bits of the total number of scan lines in each display frame. This field is extended with CR7[0] and CR7[5]. This is additionally extended in CR1B[06] to a total of eleven bits if vgaInit0[6] is '1'. The entire 10- or 11-bit field contains a value two less than the actual number of scan lines. Vertical retrace is included. |

### 3.3.9 CR7: CRTC Overflow

This register contains overflow bits from other CRTC timing registers. This register was included in the IBM VGA core.

| Bit | Description |
|---|---|
| 7 | **VertSyncStart[9]:** This is bit nine of the Vertical Sync Start value. This extends CR10 and CR7[2]. |
| 6 | **VertDispEnEnd[9]:** This is bit nine of Vertical Display Enable End value. This extends CR12 and CR7[1]. This is further extended in CR1B[2] if vgaInit0[6] is ‘1’. |
| 5 | **VertTotal[9]:** This is bit nine of the Vertical Total value. This extends CR6 and CR7[0]. This is further extended in CR1B[0] if vgaInit0[6] is ‘1’. |
| 4 | **LineComp[8]:** This is bit eight of the Line Compare value. This extends CR18 and is further extended in CR9[6]. |
| 3 | **VertBlankStart[8]:** This is bit eight of the Vertical Blank Start value. This extends CR15. and is further extended in CR9[5]. This value is extended to 11 bits in CR1B[4] if vgaInit0[6] is ‘1’. |
| 2 | **VertSyncStart[8]:** This is bit eight of the Vertical Sync Start value. This extends CR10 and is further extended in CR7[7]. |
| 1 | **VertDispEnEnd[8]:** This is bit eight of the Vertical Display Enable End value. This extends CR12 and is further extended in CR7[6]. This is further extended in CR1B[2] if vgaInit0[6] is ‘1’. |
| 0 | **VertTotal[8]:** This is bit eight of the Vertical Total value. This extends CR6 and is further extended in CR7[5]. This is further extended in CR1B[0] if vgaInit0[6] is ‘1’. |

### 3.3.10 CR8: CRTC Preset Row Scan

This register contains a field used for soft scrolling, as well as the byte panning control field.

| Bit | Description |
|---|---|
| 7 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 6:5 | **BytePan[1:0]:** This field controls byte panning. This allows up to three bytes to be panned in modes programmed as multiple shift modes. The table indicates the left shift. |
| 4:0 | **PresetRowScan[4:0]:** This field is used for soft scrolling in character modes. The first character row at the beginning of each frame begins with the scan line specified in this field. Increasing this value by one scrolls the screen (or at least screen 'A') up one scan line. This field must not be programmed to a value larger than CR9[4:0]. |

| CR8[6:5] | Left Shift (Bytes) | Left Shift (Pixels) |
|---|---|---|
| 00 | 0 | 0 |
| 01 | 1 | 8 |
| 10 | 2 | 16 |
| 11 | 3 | 24 |

### 3.3.11 CR9:CRTC Maximum Scan Line

This register specifies the maximum number of scan lines for each character row. There are also some overflow bits.

| Bit | Description |
|---|---|
| 7 | **LineDoubling:** If this bit is '1', each scan line is displayed twice in succession. The parameters based on the vertical counter (character height, cursor start and stop, and underline location) double. This is used to display 200-line modes at 400 scan lines. |
| 6 | **LineComp[9]:** This bit extends the Line Compare value in CR18 and CR7[4] to ten bits. |
| 5 | **VertBlankStart[9]:** This bit extends the Vertical Blank Start value in CR15 and CR7[3] to ten bits. This field is further extended in CR1B[4] if vgaInit0[6] is '1'. |
| 4:0 | **MaxScanLine[4:0]:** This field specifies the maximum number of scanned lines for each row of characters. The value programmed is the maximum number of scan lines per row minus one. This is also the character cell height for every character row except possibly the first. |

### 3.3.12 CRA: CRTC Cursor Start

This register specifies the scan line where the text cursor is to start. In addition, this register contains a bit that enables the text cursor.

| Bit | Description |
|---|---|
| 7:6 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 5 | **DisableTextCursor:** If this bit is '1', the text cursor is disabled. If this bit is 0, the text cursor is enabled. |
| 4:0 | **CursorStart[4:0]:** This field specifies the scan line within the character cell where the text cursor is to start. If this field is greater than the Text Cursor End value, no text cursor is displayed. If this field is equal to the Text Cursor End value, the text cursor occupies a single scan line. |

### 3.3.13 CRB: CRTC Cursor End

This register specifies the scan line where the text cursor is to end.

| Bit | Description |
|---|---|
| 7 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 6:5 | **TextCursorSkew:** This field specifies a delay, in character clocks, from the text cursor location in CRE and CRF to the actual cursor. |
| 4:0 | **CursorEnd:** This field specifies the last row within the character box during which the text cursor is to be active. |

### 3.3.14 CRC: CRTC Screen Start Address High

This register specifies eight bits of the location in display memory where the image to be displayed on the screen begins.

| Bit | Description |
|---|---|
| 7:0 | **ScreenStartAddress[15:8]:** This field specifies bits 15 through 8 of the location in video memory used for screen refresh. The low order eight bits are in CRD. |

### 3.3.15 CRD: CRTC Screen Start Address Low

This register specifies eight bits of the location in display memory where the image to be displayed on the screen begins.

| Bit | Description |
|---|---|
| 7:0 | **ScreenStartAddress[7:0]:** This field specifies bits seven through zero of the location in video memory used for screen refresh. |

### 3.3.16 CRE: CRTC Cursor Location High

This register specifies eight bits of the location in display memory of the character to be displayed with the text cursor.

| Bit | Description |
|---|---|
| 7:0 | **CursorLocation[15:8]:** This field specifies bits 15 through 8 of the location in video memory of the character to be displayed with the text cursor. The low order eight bits are in CRF. |

### 3.3.17 CRF: CRTC Cursor Location Low

This register specifies eight bits of the location in display memory of the character to be displayed with the text cursor.

| Bit | Description |
|---|---|
| 7:0 | **CursorLocation[7:0]:** This field specifies bits seven through zero of the location in video memory of the character to be displayed with the text cursor. |

### 3.3.18 CR10: CRTC Vertical Sync Start

This register contains the low order eight bits of the scan line count at which vertical sync is to become active.

| Bit | Description |
|---|---|
| 7:0 | **VertSyncStart[7:0]:** This field contains the low order eight bits of the Vertical Sync Start value. This value is extended by CR7[2] and CR7[7]. This value is further extended in CR1B[6] if vgaInit0[6] is ‘1’. |

### 3.3.19 CR11: CRTC Vertical Sync End

This register specifies the scan line at which vertical sync becomes inactive. There are also the VGA interrupt control bits.

| Bit | Description |
|---|---|
| 7 | **CRTCRegsWriteProt:** When this bit is 0, registers CR0-CR7 can be written. When this bit is ‘1’, registers CR0-CR7 cannot be written except for CR7[4]. |
| 6 | **DramRefresh:** In the legacy VGA controller, this bit was used to choose the number of DRAM refresh cycles per scan line. In Voodoo3, this bit is not used. |
| 5 | **EnableVertInt:** If this bit is ‘1’, the vertical interrupt is disabled. If this bit is 0, the vertical interrupt is enabled. This bit is used for VGA compatibility only. Voodoo3 has its own interrupt system. |
| 4 | **ClearVertInt:** When this bit is programmed to 0, the vertical interrupt is cleared. When this bit is programmed to ‘1’, the next occurrence of the vertical interrupt is allowed. |
| 3:0 | **VertSyncEnd:** This field contains the VertSyncEnd value. This value is compared with the low order four bits of the vertical counter to determine when sync is to end. The value for this field can be calculated by adding the desired sync interval to the VertSyncStart value (and retaining only the low order four bits). |

### 3.3.20 CR12: CRTC Vertical Display Enable End

This register contains the low order eight bits of the total number of scan lines displayed in each frame.

| Bit | Description |
|---|---|
| 7:0 | **VertDispEnEnd[7:0]:** This field contains the low order eight bits of the number of visible scanlines on the screen, minus one. This field is extended by CR7[1] and CR7[6]. This field is further extended in CR1B[2] if vgaInit0[6] is '1'. |

### 3.3.21 CR13: CRTC Offset

This register specifies the distance in display memory between vertically adjacent character rows or scan lines. Depending on where one went to school, this may be referred to as display pitch or display stride.

| Bit | Description |
|---|---|
| 7:0 | **Offset:** This field specifies the distance in display memory between the beginnings of vertically adjacent character rows or scan lines. At the beginning of each character row or scanline (except the first), the address of the data to be fetched for display is calculated by adding this value to the address used at the beginning of the previous character row or scan line. The offset value is a word address adjusted for word or doubleword display memory access. |

### 3.3.22 CR14: CRTC UnderlineLocation

This register specifies the line within a character cell at which the underline is to be displayed.

| Bit | Description |
|---|---|
| 7 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 6 | **DoubleWordMode:** When this bit is '1', the display memory is addressed for doubleword access. When this bit is 0, the display memory is address for byte or word access. |
| 5 | **CountByFour:** When this bit is 0, the memory address counter is clocked for byte or word access. When this bit is '1', the memory address counter is clocked at the character clock divided by four. This bit must be 0 when CR14[6] is 0. |
| 4:0 | **UnderlineLoc:** This field specifies the row scan counter value within a character cell where the underline is to be displayed. This field is one less than the actual desired scan line number. |

### 3.3.23 CR15: CRTC Vertical Blanking Start

This register contains the low order eight bits of the number of scan lines displayed before vertical blanking is started.

| Bit | Description |
|---|---|
| 7:0 | **VertBlankStart[7:0]:** This field is the low order eight bits of the scan line at which vertical blanking is to start, minus one. This value is extended by CR7[3] and CR9[5]. This is further extended by CR1B[4] to a total of 11 bits if vgaInit0[6] is '1'. |

### 3.3.24 CR16: CRTC Vertical Blanking End

This register specifies the scan line at which vertical blanking is to end and the bottom border (if any) is to begin.

| Bit | Description |
|---|---|
| 7:0 | **VertBlankEnd[7:0]:** This field specifies the scan line on which vertical blanking is to end. This value is compared to the low order eight bits of the vertical counter to determine when blanking is to end. The value for this field can be calculated by adding the desired blanking interval to the VertBlankStart value minus one (and retaining only the low order bits). |

### 3.3.25 CR17: CRTC Mode Control

This register contains control bits for the CRTC.

| Bit | Description |
|---|---|
| 7 | **TimingEnable:** If this bit is '1', the CRTC timing logic operates normally. If this bit is 0, the CRTC timing logic is disabled and the syncs are static. |
| 6 | **ByteWordMode:** If this bit is '1' the contents of the refresh address counter are sent to the frame buffer without being modified. This is for byte addressing. If this bit is 0, the contents of the refresh address counter are rotated left one bit position before being sent to the frame buffer. This is for word addressing. |
| 5 | **AddressWrap:** In byte address mode (CR17[6]=1), this bit is ignored. In word address mode (CR17[6]=0), this bit controls the rotation. If this bit is '1', then 16 bits are rotated; if this bit is 0, then 14 bits are rotated. |
| 4 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 3 | **CountByTwo:** If this bit is 0, the memory address counter is incremented for every character clock. If this bit is '1', the memory address counter is incremented for every second character clock. |
| 2 | **VerticalTimingClockSelect:** If this bit is 0, the vertical counter is clocked with HSYNC. If this bit is '1', the vertical counter is clocked with HSYNC divided by two. All vertical periods are multiples of two scanlines. |
| 1 | **SelectRowScanCounter:** If this bit is 0, row scan counter bit 1 is output at the MA14 address pin. This is for Hercules compatibility. If this bit is '1', CRTC address counter bit 14 is output at the MA14 address pin. |
| 0 | **CompatModeSupport:** If this bit is 0, row scan counter bit 0 is output at the MA13 address pin. This is for CGA compatibility. If this bit is '1', memory address counter bit 13 is output at the MA13 address pin. |

### 3.3.26 CR18: CRTC Line Compare

This register contains the low order eight bits of the line compare value.

| Bit | Description |
|---|---|
| 7:0 | **LineCompare[7:0]:** This field is the low order eight bits of the line compare value. This field is extended by CR7[4] and CR9[6]. This can be used to implement a vertically split character screen. The top portion of the screen is called screen A and can begin anywhere in display memory. Screen A can be panned on scrolled on a pixel basis. The bottom portion of the display is screen B. Screen B always begins at location zero in the frame buffer and cannot be panned or scrolled. |

### 3.3.27 CR1A: CRTC Horizontal Extensions

This register contains extension bits to increase the horizontal resolution available to Voodoo3. This register is active only when vgaInit0[6] is '1'.

| Bit | Description |
|---|---|
| 7 | **HorSyncEnd[5]:** This is bit five of the HorSyncEnd value. This extends CR5[4:0]. |
| 6 | **HorSyncStart[8]:** This is bit eight of the HorSyncStart value. This extends CR4. |
| 5 | **HorBlankEnd[6]:** This is bit six of the HorBlankEnd value. This extends CR3[4:0] and CR5[7]. |
| 4 | **HorBlankStart[8]:** This is bit eight of the HorBlankStart value. This extends CR2. |
| 3 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 2 | **HorDisEnEnd[8]:** This is bit eight of the HorDispEnEnd value. This extends CR1. |
| 1 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 0 | **HorTotal[8]:** This is bit eight of the HorTotal value. This extends CR0. |

### 3.3.28 CR1B: CRTC Vertical Extensions

This register contains extension bits to increase the vertical resolution available to Voodoo3. This register is active only when vgaInit0[6] is '1'.

| Bit | Description |
|---|---|
| 7 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 6 | **VertSyncStart[10]:** This is bit ten of the VertSyncStart value. This extends CR10, CR7[2], and CR7[7]. |
| 5 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 4 | **VertBlankStart[10]:** This is bit ten of the VertBlankStart value. This extends CR15, CR7[3], and CR9[5]. |
| 3 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 2 | **VertDisEnEnd[10]:** This is bit ten of the VertDispEnEnd value. This extends CR12, CR7[1], and CR7[6]. |
| 1 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 0 | **VertTotal[10]:** This is bit ten of the VertTotal value. This extends CR6, CR7[0], and CR7[5]. |

### 3.3.29 CR1C: PCI Configuration Readback

This register allows the application to read the PCI configuration information one byte at a time. It can also be used as a scratch pad register.

| Bit | Description |
|---|---|
| 7:0 | **PCI Configuration:** This field allows the PCI configuration to be read, or can be used as a scratch pad, according to the programming of vgaInit0[7:6]. |

| VGAINIT[7:6] | Description | Note |
|---|---|---|
| 00b | Readback PCI Configuration | Write address, then read data |
| 01 | Scratch Pad | One byte of R/W data |
| 1X | Disabled | - |

### 3.3.30 CR1D, CR1E, CR1F: Scratch Pad

These registers are used as scratch pads. Each can contain one byte. These registers are available only when vgaInit0[6] is '1'.

| Bit | Description |
|---|---|
| 7:0 | **ScratchPad[7:0]:** Each register is one byte of scratch pad. These are reserved for the exclusive use of 3Dfx Interactive, Inc. software. |

### 3.3.31 CR20: CRTC Vertical Counter Preload Low

This register, with CR21, allow the vertical counter to be pre-loaded for factory testing. These registers need never be programmed by any application. This description is included only for the sake of completeness.
The vertical counter is loaded with the contents of these registers on reset, which can be either a hard reset or a soft reset. These registers are active only when vgaInit0[6] is '1'. The value read back from these registers is the value that was loaded, not necessarily the contents of the vertical counter at any given instant.

| Bit | Description |
|---|---|
| 7:0 | **VertCountPrel[7:0]:** This is the low order eight bits of the vertical counter pre-load value. |

### 3.3.32 CR21: CRTC Vertical Counter Preload High

This is the high order three bits of the vertical counter preload value.

| Bit | Description |
|---|---|
| 7:3 | **Reserved:** Reserved bits must be written as '0' for upward compatibility. |
| 2:0 | **VertCountPrel[10:8]:** This is the high order three bits of the vertical counter pre-load value. |

### 3.3.33 CR22: Latches ReadBack

This read-only register can be used to read the contents of the data latches. Cr22, CR24, and CR26 were included in the IBM VGA core, but were never documented by IBM.

| Bit | Description |
|---|---|
| 7:0 | **LatchData:** This read-only field returns the contents of the Graphics Data Controller latch selected by GR4[1:0]. |

### 3.3.34 CR24: Attribute Controller Toggle Readback

This register can be used to read the state of the Attribute Controller Index/Data Toggle. This is a read-only register. Do not write to this register.

| Bit | Description |
|---|---|
| 7 | **Toggle:** When this bit returns 0, the attribute controller toggle is set to Index. When this bit is '1', the attribute controller toggle is set to Data. Reading the STAT register at 0x3DA will always force the toggle to Index. |

### 3.3.35 CR26: Attribute Controller Index Readback

This register can be used to read the contents of the attribute controller index. This is a read-only register. Do not write to this register.

| Bit | Description |
|---|---|
| 7:6 | **Reserved:** Take no action based on the contents of this field. |
| 5 | **DisplayEnable:** This read-only bit follows the Display Enable in the Attribute Controller (this is a read-only copy of ARX[5]). |
| 4:0 | **AttributeIndex:** This read-only field returns a copy of ARX[4:0]. |

## 3.4 Graphics Controller Registers

### 3.4.1 GRX: Graphics Controller Index

This register specifies the register in the graphics controller block of the VGA core to be accessed by the next I/O read or I/O write to 0x3CF.

| Bit | Description |
|---|---|
| 7:4 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 3:0 | **GRX:** This field is the address of the register in the graphics controller block to be accessed by the next I/O read or I/O write to 0x3CF. |

### 3.4.2 GR0: Graphics Controller Set/Reset

This register specifies the values to be written into the respective display memory planes when a Write Mode 0 is executed.

| Bit | Description |
|---|---|
| 7:4 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 3:0 | **SetResetPlane[3:0]:** Each bit specifies the value written into the respective planes when a write mode 0 is executed and the respective bit in GR1 is ‘1’. This field has no effect on reads from the frame buffer or in Write Mode 1 or Write Mode 2. |

### 3.4.3 GR1: Graphics Controller Set/Reset Enable

This register selects the source of data to be written into each display memory plane when a Write Mode 0 is executed.

| Bit | Description |
|---|---|
| 7:4 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 3:0 | **SetResetEnable[3:0]:** When write mode 0 is executed, these bits select the source of data for the respective display memory planes. If a bit is ‘0’, the corresponding value from the data is bus is written into the respective plane. If a bit is ‘1’, the value from the corresponding bit of GR0 is written into the respective plane. This field has no effect on reads from the frame buffer. |

### 3.4.4 GR2: Graphics Controller Color Compare

This register specifies the color compare value for Read Mode 1.

| Bit | Description |
|---|---|
| 7:4 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 3:0 | **ColorComparePlane[3:0]:** When a Read Mode 1 occurs, each of these four bits is compared to the eight bits from the respective display memory plane. A ‘1’ is returned for each bit position for which a match in all four planes takes place. Planes can be masked using GR7. See the description of Read Mode 1 in Section 3.4.7. |

### 3.4.5 GR3: Graphics Controller Data Rotate

This register contains two fields used to control the data written to the frame buffer.

| Bit | Description |
|---|---|
| 7:5 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 4:3 | **FunctionSelect:** This field controls the operation that occurs between the data in the latches and the data from the CPU or the Set/Reset logic. The result of the operation is written into display memory. This field is effective for Write Mode 0 and Write Mode 3. |
| 2:0 | **RotateCount:** This field allows data from the CPU to be rotated to the right by as many as seven bit positions prior to being written. |

| FunctionSelect | Description |
|---|---|
| 00b | None: The data in the latches are ignored. |
| 01b | Logical ‘AND’ |
| 10b | Logical ‘OR’ |
| 11b | Logical ‘XOR’ |

### 3.4.6 GR4: Graphics Controller Read Plane Select

This register specifies the display memory plane for Read Mode 0.

| Bit | Description |
|---|---|
| 7:2 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 1:0 | **PlaneSelect** This field specifies the display memory plane for Read Mode 0. |

### 3.4.7 GR5: Graphics Controller Mode

This register specifies the Read and Write Modes, and selects the configuration of the data shifters.

**Bit** **Description**

7 **Reserved:** Reserved bits must be written as 0 for upward compatibility.

6:5 **ShiftMode:** This field selects the configuration of the data shifters.

| ShiftMode | Configuration |
|---|---|
| 00 | Data is shifted out normally. |
| 01 | Data is shifted out Even/Odd. |
| 1x | 256 Color Mode. |

4 **OddEven:** If this bit is '1', the graphics controller is configured for Odd/Even addressing. This bit should always be the opposite as SR4[2].

3 **ReadMode:** This bit specifies the VGA Read Mode.
When this bit is '0', the CPU reads data directly from the display memory. Each read returns eight adjacent bits of the display memory plane specified in GR4[1:0]. The color-match logic is not used.
When this bit is '1', the CPU reads the results of the color compare logic. This allows eight adjacent pixels to be compared to a specified four-bit color in a single operation. Each of the eight bits returned is the result of a compare between the four bits of GR2[3:0] and the bits from the four display memory planes. If the four bits of GR2 match the four bits from the planes, a '1' is returned in the corresponding bit position. If any bits in GR7[3:0] are '0', the corresponding plane is forced to compare in all eight bit positions. This allows entire planes to be ignored.

2 **Reserved:** Reserved bits must be written as 0 for upward compatibility.

1:0 **WriteMode:** This field specifies the VGA Write Mode.
When this field is '00', Write Mode 0 is selected. Data from the CPU is rotated according to GR3[2:]. If a bit in GR1[3:0] is '1', the corresponding plane is written with the contents of the corresponding bit in GR0[3:0]. If a bit in GR1[3:0] is '0', the corresponding plane is written with the rotated data from the CPU. The contents of the data latches can be combined with the selected bits according to GR3[4:3]. Bit planes are enabled for writing with SR2[3:0] and bit positions are enabled with GR8.
When this field is '01', Write Mode 1 is selected. Each of the four display memory planes is written with the data in the latches. GR8 is ignored.
When this field is '10', Write Mode 2 is selected. Display memory planes 3:0 are written with the values of data bits 3:0, respectively. The four bits are replicated eight times to write up to eight adjacent pixels. Bit planes are enabled with SR2[3:0]; bit positions are enabled with GR8.
When this field is '11', Write Mode 3 is selected. The data for each display memory plane comes from the corresponding bit of GR0. Bit positions are enabled with the logical AND of GR8 and the rotated CPU data.

### 3.4.8 GR6: Graphics Controller Miscellaneous

This register contains some miscellaneous fields controlling the graphics controller block.

| Bit | Description |
|---|---|
| 7:4 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 3:2 | **MemoryMap:** This field specifies the beginning address and size of the display memory in the host address space. |

| MemoryMap | Beginning Address | Size | Typical Usage |
|---|---|---|---|
| 00 | 0xA0000 | 128K | (unused) |
| 01 | 0xA0000 | 64K | EGA, VGA, Extended modes |
| 10 | 0xB0000 | 32K | Monochrome text modes |
| 11 | 0xB0000 | 32K | Color text, CGA graphics |

### 3.4.9 GR7: Graphics Controller Color Dont Care

This register specifies the planes to be ignored for a Read Mode 1.

| Bit | Description |
|---|---|
| 7:4 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 3:0 | **ColorComparePlaneDontCare[3:0]:** When a Read Mode 1 occurs, each of these four bits can disable matching on the respective plane. If a bit is ‘1’, the respective plane must match; if a bit is ‘0’, the respective plane is a don’t care for all bit positions. See the description of Read Mode 1 at GR5[3]. |

### 3.4.10 GR8: Graphics Controller Bit Mask

This register controls writes to display memory in Write Modes 0, 2, and 3.

| Bit | Description |
|---|---|
| 7:0 | **BitWiseWriteEnable:** Each bit in this register controls whether the corresponding bit in display memory is written in Write Modes 0, 2, and 3. If a bit in this field is ‘1’, the corresponding bit in display memory can be written. If a bit in this field is ‘0’, the corresponding bit in display memory cannot be written. This write protection is orthogonal to that provided in SR2. |

## 3.5 Attribute Controller Registers

### 3.5.1 ARX: Attribute Controller Index

This register specifies the register in the attribute controller block of the VGA core to be accessed by the next I/O read or I/O write to 0x3C0.
The attribute block uses a flip-flop to determine whether the next write accesses the address or data register (the other blocks use the low order address bit). Each write to 0x3C0 toggles this flip-flop. Reading 0x3DA clears the flip-flop, which selects the address register. Reading a register at 0x3C1 does not toggle the flip-flop. The state of the flip-flop can be determined by reading CR24.

| Bit | Description |
|---|---|
| 7:6 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 5 | **PaletteAddressSouce:** If this bit is ‘0’, the palette outputs are disabled and the screen will display the color specified in the overlay register (AR11). If this bit is ‘1’, the screen will display the normal graphics. |
| 43:0 | **ARX:** This field is the address of the register in the graphics controller block to be accessed by the next I/O read at 0x3C1 or I/O write to 0x3C0. |

### 3.5.2 AR0-ARF: Attribute Controller Palette

This block of registers provides a color palette for planar modes (CGA and EGA modes). The palette contains 16 entries of six bits each and provides a translation from four bits to six bits. The palette output can be combined with, or modified by, the contents of AR14.

| Bit | Description |
|---|---|
| 7:6 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 5:0 | **Palette Entries:** These six-bit fields specify the actual pixel colors. |

### 3.5.3 AR10: Attribute Controller Mode

This register contains miscellaneous control bits for the attribute controller.

| Bit | Description |
|---|---|
| 7 | **VID5/4 Select:** If this bit is ‘0’, video bits AR0-ARF[5:4] are used as the source of VID[5:4]. If this bit is ‘1’, AR14[1:0] are used as the source of VID[5:4]. This allows the rapid selection of four 16-color palette. |
| 6 | **PixelWidth:** If this bit is ‘1’, pixels are clocked on every 2nd VCLK and the palette registers (AR0-ARF) are bypassed. This is used for display mode 13. If this bit is ‘0’, pixels are clocked on every VCLK. |
| 5 | **PixelPanningCompatibility:** If this bit is ‘1’, a line compare match in the CRTC disables the output of the pixel panning register until the next VSYNC. This allows the panning of Screen A without Screen B. If this bit is ‘0’, the two parts of the split screen pan together. |
| 4 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| **3** | **Blink Enable:** If this bit is ‘1’, character blinking is enabled at the vertical refresh frequency divided by 32. |

### 3.5.3 AR10: Attribute Controller Mode *(cont)*

| Bit | Description |
|---|---|
| 2 | **LineGraphicsCharacterCode:** If this bit is '1', the ninth bit of each row of a nine-bit character cell is the same as the corresponding eighth bit. This is for character codes in the range 0xC0 through 0xDF. If this bit is 0, the ninth bit is forced to the background color. |
| 1 | **MonoCharEmulation:** This bit is used only if the VGA core is in alphanumeric mode. If this bit is '1', the attribute byte is taken as MDA-compatible attributes. If this bit is 0, the attribute byte defines the character color. |
| 0 | **GraphicsMode:** If this bit is 0, the attribute controller is in alphanumeric mode. If this bit is '1', the attribute controller is in graphics mode. |

### 3.5.4 AR11: OverScan Color

This register specifies the overscan (border) color. The border is defined as the portion of the raster between blanking and active display, on all four sides. Most display modes are programmed so that there is no border.

| Bit | Description |
|---|---|
| 7:0 | **OverscanColor:** This field specifies the overscan color. This is mechanized as a pointer into the RAMDAC. |

### 3.5.5 AR12: Color Plane Enable

This register contains a field that enables the four color planes. It also contains the field that controls the multiplexor for the diagnostic bits in STAT[5:4].

| Bit | Description |
|---|---|
| 7:6 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 5:4 | **VideoStatusMux:** This field selects two out of eight color outputs that are read in STAT[5:4]. |

| VideoStatusMux | STAT[5] | STAT[4] |
|---|---|---|
| 00 | P[2] | P[0] |
| 01 | P[5] | P[4] |
| 10 | P[3] | P[1] |
| 11 | P[7] | P[6] |

| | |
|---|---|
| 3:0 | **EnableColorPlane[3:0]:** If a bit in this field is '1', the corresponding color plane is enabled. If a bit in this field is 0, the corresponding color plane is disabled and behaves as though it contains all zeroes. |

### 3.5.6 AR13: Pixel Panning

This register specifies the number of pixels the display data is shifted left.

| Bit | Description |
|---|---|
| 7:4 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 3:0 | **PixelPanning:** This field controls the horizontal pixel panning. |

| PixelPanning | 9-bit Text | 256 Color | Other |
|---|---|---|---|
| 000b | 1 | 0 | 0 |
| 0001b | 2 | 1/2 | 1 |
| 0010b | 3 | 1 | 2 |
| 0011b | 4 | 1 1/2 | 3 |
| 0100b | 5 | 2 | 4 |
| 0101b | 6 | 2 1/2 | 5 |
| 0110b | 7 | 3 | 6 |
| 0111b | 8 | 3 1/2 | 7 |
| 1000b - 1111b | 0 | -1[a] | -1 |

a. This is actually a one-pixel right pan.

### 3.5.7 AR14: Attribute Color Select

This register contains two fields that are used in the color selection.

| Bit | Description |
|---|---|
| 7:4 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 3:2 | **ColorValueMSB:** These two bits are concatenated with six bits from the attribute palette registers (AR0-ARF) to form the eight bit color value. These bits are used in all modes with less than eight bits per pixel. |
| 1:0 | **SubColorValueBits:** These bits are substituted for bits five and four from the attribute palette registers when AR10[7] is ‘1’. |

## 3.6 Sequencer Registers

### 3.6.1 SRX: Sequencer Index

This register specifies the register in the sequencer block of the VGA core to be accessed by the next I/O read or I/O write to 03C5.

| Bit | Description |
|---|---|
| 7:0 | **SRX:** This field is the address of the register in the sequencer block to be accessed by the next I/O read or I/O write to 0x3C5. |

### 3.6.2 SR0: Sequencer Reset

This register contains bits that force the sequencer block of the VGA core to be reset.

| Bit | Description |
|---|---|
| 7:2 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 1 | **SynchronousReset:** When this bit is 0, the video timing is cleared and halted. This is used to synchronize changing the clock select field in MISC[3:2]. When this bit is '1', the video timing is operational. |
| 0 | **AsynchronousReset:** When this bit is 0, the sequencer is cleared and halted asynchronously. This bit is used to force the sequencer into a reset state, regardless of the operation it is performing at the moment. |

### 3.6.3 SR1: Sequencer Clocking Mode

This register controls some miscellaneous functions in the sequencer block.

| Bit | Description |
|---|---|
| 7:6 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 5 | **ScreenOff:** When this bit is '1', screen refresh stops. This allows the CPU to use nearly 100% of the display memory bandwidth. HSYNC and VSYNC continue. Display memory refresh continues. BLANK# goes active and stays active. When this bit is 0, screen refresh continues normally. |
| 4 | **Load32:** This bit is used with SR1[2] to control the data shifters according to the table. |
| 3 | **DotClockDiv2:** When this bit is '1', the dot clock is divided by 2. This is for display modes with 320 or 360 pixels per scan line. |
| 2 | **Load16:** See SR1[4]. |

| SR1[4] | SR1[2] | Data Shifters Loaded |
|---|---|---|
| 0 | 0 | Every character clock |
| 0 | 1 | Every 2nd character clock |
| 1 | x | Every 4th character clock |

### 3.6.3 SR1: Sequencer Clocking Mode *(cont)*

| Bit | Description |
|---|---|
| 1 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 0 | **8/9DotClock:** When this bit is '1', the dot clock is divided by eight to generate the character clock. This is used for all graphics modes and alphanumeric modes with eight-bit character cells. When this bit is 0, the dot clock is divided by nine to generate the character clock. This is used for alphanumeric modes with nine-bit character cells (720 x 350 and 720 x 400). |

### 3.6.4 SR2: Sequencer Plane Mask

This register enables writing to the four bit planes of display memory.

| Bit | Description |
|---|---|
| 7:4 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 3:0 | **MapEnable[3:0]:** Each of these four bits controls whether the corresponding bit plane is written with write modes 0, 1, 2, and 3. |

### 3.6.5 SR3: Sequencer Character Map Select

This register selects the primary and secondary character sets. This is used only for alphanumeric modes.

- In alphanumeric modes, the ASCII text character is stored in plane 0, the attribute is stored in plane 1, and the fonts are stored in plane 2.
- Bit 3 of the attribute byte normally controls the foreground color intensity. This bit is redefined to switch between character sets (allowing 512 displayable characters) when SR4[1] is '1' and the map select fields in this register are not identical.

| Bit | Description |
|---|---|
| 7:6 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 5, 3:2 | **PrimaryMapSelect:** These three bits select the primary map, according to the table. |
| 4, 1:0 | **SecondaryMapSelect:** These three bits select the secondary map, according to the table. |

| MapSelect | Map | Table Offset |
|---|---|---|
| 0 00 | 0 | 0K |
| 0 01 | 1 | 16K |
| 0 10 | 2 | 32K |
| 0 11 | 3 | 48K |
| 1 00 | 4 | 8K |
| 1 01 | 5 | 24K |
| 1 10 | 6 | 40K |
| 1 11 | 7 | 56K |

### 3.6.6 SR4: Sequencer Memory Mode

This register controls miscellaneous functions in the sequencer block.

| Bit | Description |
|---|---|
| 7:4 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 3: | **Chain4:** If this bit is '1', the low order two bits of the processor address bus select the memory plane. This has a similar effect as SR4[2] except that two address bits are used. If this bit is 0, the processor sequentially accesses data using the map select register. This bit overrides SR4[2]. |
| 2: | **OddEven:** If SR4[3] is '1', this bit is ignored. If SR[3] is 0, this bit has the following meaning. If this bit is '1', the low order bit of the processor address bus selects the memory plane.<br>This bit must be 0 for text modes. This bit must track GR5[4]; the values are opposite. |
| 1 | **ExtendedMemory:** If this bit is 0, the memory size is restricted to 16/32K. |
| 0 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |

## 3.7 RAMDAC Registers

### 3.7.1 RAMDAC Pixel Mask

The bits in this register mask the VGA data stream.

| Bit | Description |
|---|---|
| 7:0 | **PixelMask:** Each bit in this field is ANDed with the corresponding bit of the VGA data stream before it addresses the RAMDAC. The value in this field has no effect on display modes other than VGA. |

### 3.7.2 RAMDAC Read Address

This write-only register specifies the address at which the processor will read data from the palette. This is a write-only register. The same address is used for the RAMDAC Status.

| Bit | Description |
|---|---|
| 7:0 | **ReadAddress:** When a value is written to this register, it puts the RAMDAC look-up table into 'read' state. The application should follow the write with three consecutive reads at 0x3C9 to recover the red, green, and blue values of the indicated RAMDAC entry. The index will automatically increment following the completion of the third data read (the next entry is read). This is a write-only register, but see Section 3.7.3. Only the first 256 entries of the RAMDAC look-up can be accessed using this port. |

### 3.7.3 RAMDAC Read Status

This read-only register returns the read/write state of the RAMDAC look-up table.

| Bit | Description |
|---|---|
| 7:2 | **Reserved:** Application programs should take no action based on the contents of this field. |
| 1:0 | **RAMDACState:** These two read-only bits will always be the same. If they are 0, a RAMDAC read is in effect (that is, the RAMDAC Read Address was written since the RAMDAC Write Address was written). If they are '1', a RAMDAC write is in effect (that is, the RAMDAC Write Address was written since the RAMDAC Read Address was written). |

### 3.7.4 RAMDAC Write Address

This register specifies the address at which the processor will write data into the palette.

| Bit | Description |
|---|---|
| 7:0 | **WriteAddress:** When a value is written to this register, it puts the RAMDAC look-up table into 'write' state. The application should follow the write with three consecutive writes to 0x3C9 of the red, green, and blue value to be stored at the indicated RAMDAC entry. The index will automatically increment following the completion of the third data write. Only the first 256 entries of the RAMDAC look-up table can be written using this port. This is a read/write register. |

### 3.7.5 RAMDAC Data

This is the color value data register for the RAMDAC look-up table.

| Bit | Description |
|---|---|
| 7:0 | **RAMDACData:** This register is used to write data to and read data from the RAMDAC look-up table. Each table entry requires three accesses to read or write. The colors are always transferred in order red, green, and blue.<br>Data in this register is either 6 bits per color (VGA compatible) or 8 bits per color. This determination is made by vgaInit0[2]. When data is in six-bit format, the two most significant bits are replicated into the two least significant bits to maintain full scale and linearity on the DAC. |

# 4 VGA Programming Notes

## 4.1 Introduction

This chapter consists of programming notes for the VGA core. It does not cover the fundamentals of VGA programming; refer to Ferraro[1] or the equivalent for this.
Voodoo3 incorporates a VGA core that supports all standard VGA modes with full backward compatibility. The incorporation of the VGA core allows the 3D/2D controller to share the frame buffer with the VGA core, eliminating the need for a separate VGA display memory and controller chip.
In addition to the legacy VGA display modes 0-13, Voodoo3 supports VBE (VESA BIOS Extensions).

## 4.2 Accessing VGA Registers Through PCI18

All of the VGA registers (except 0x46E8 and 0x0102) are available through PCI18. The address is calculated by dropping the 0x0300 from the legacy address and adding the result to the I/O base address (the address in PCI18). For example, suppose the application wants to write the Miscellaneous Output register (VGA address 0x03C2). It is available at I/OBase + 0x00C2.

## 4.3 Voodoo3 not Primary VGA

For systems not requiring VGA or for systems in which a VGA device already exists, Voodoo3 allows the use of the VGA registers in extended mode. In this mode, VGA registers are not decoded in legacy VGA space, but in relocatable I/O and memory space.
To use this mode, Voodoo3 must be powered up with VMI_HD4 pulled up. PCI08 will report ‘other multimedia’ rather than ‘VGA’. The system will not try to boot to Voodoo3 and Voodoo3 will not respond to accesses to any legacy VGA I/O or memory space. In order to use the VGA registers, program vgaInit0[8] to ‘1’ to put the wakeup address at 0x03C3. Now program I/OBase plus 0xC3 bit 0 to ‘1’.

## 4.4 Locking VGA Timing for Virtualized Modes

When running VGA applications in a window, it is possible to restrict changes to the VGA timing register set. This is done by setting the respective lock bits in vgaInit1. See Section 6.3.11 for a list of bits.

## 4.5 Two Pixels per Clock Modes

For extended resolutions that run at a pixel clock greater than 135 MHz, the video unit must be programmed so that it supplies two pixels per clock. This implies that the video clock is programmed to one-half the pixel clock (see dacMode in Section 6.4.3). Since the clock is running at one-half the pixel frequency, all horizontal timing registers values must be divided by two. In addition, all horizontal timing will be in increments of 16 pixels.

## 4.6 VGA Address Space

The following table shows the address space used by the VGA core.

### Table 4.1 VGA Address Space

| **Memory Space** |
|---|
| 0x00A0000 - 0x00BFFFF |
| **I/O Addresses (8 / 16 bit addressable)** |
| 0x0102 |

1. Ferraro, Richard F. - Programmer’s Guide to the EGA and VGA Cards Addison-Wesley, third edition, 1995

**Table 4.1 VGA Address Space *(cont.)***

| |
|---|
| 0x03B4, 0x03B5 |
| 0x03BA |
| 0x03C0 - 0x03CA |
| 0x03CE, 0x03CF |
| 0x03DA |
| 0x46E8 |

## 4.7 CRT Timing Diagram

Here is the standard CRTC timing diagram. This was first published (so far as I know) by Cirrus in about 1990.

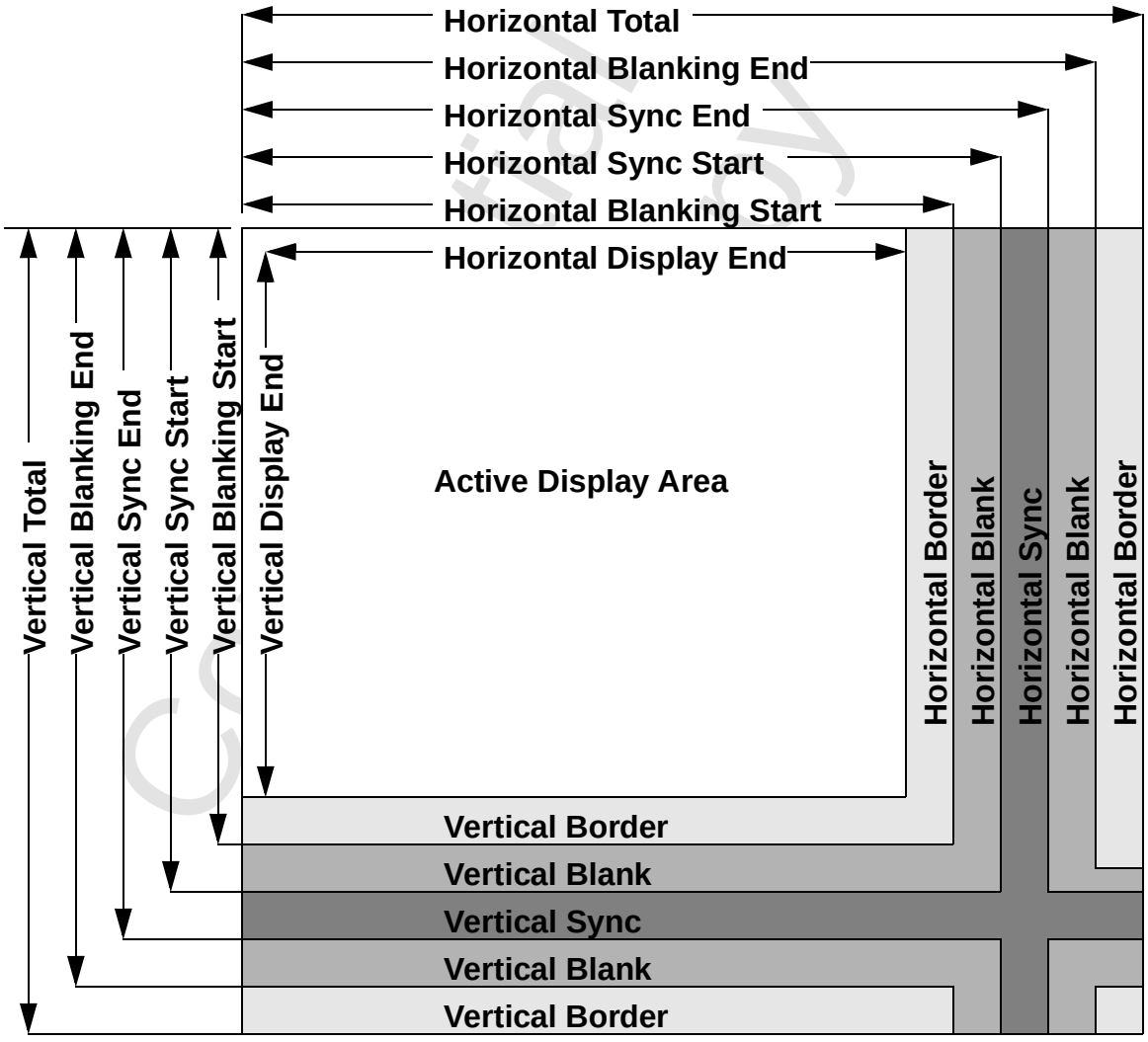



**Figure 4.1 CRTC Timing Diagram**

## 4.8 Accessing Memory in VESA Modes

### 4.8.1 VGA Restriction of 128K

VGA is restricted to only 128K of frame buffer through the aperture beginning at 0x000A 0000. This supports basline VGA graphics modes (mode 0 through 13) satisfactorily. Extended resolutions and color depths in VESA modes require more frame buffer than the 128K allowed by VGA.

### 4.8.2 VESA Extensions

Access to the entire frame buffer is available in VESA modes. vgaInit1 contains two remapping fields, one for writes and one for reads. Each remapping field is ten bits; these are combined with 16 bits from the system address (0x000A 0000 through 0x000A FFFF) to from a 24 bit address that can access any location in a 16 Mbyte frame buffer. Each aperture is 64K byte in size and can be put on any 32K boundary. The remapping field is selected based on whether the access is a read or a write. This allows memory contents to be moved between arbitrary locations without constantly rewriting the remapping field, as would be necessary if there were only a single field.
Figure 4.2 shows the remapping method. Note that system memory space beginning at 0xB0000 is not used in this scheme.

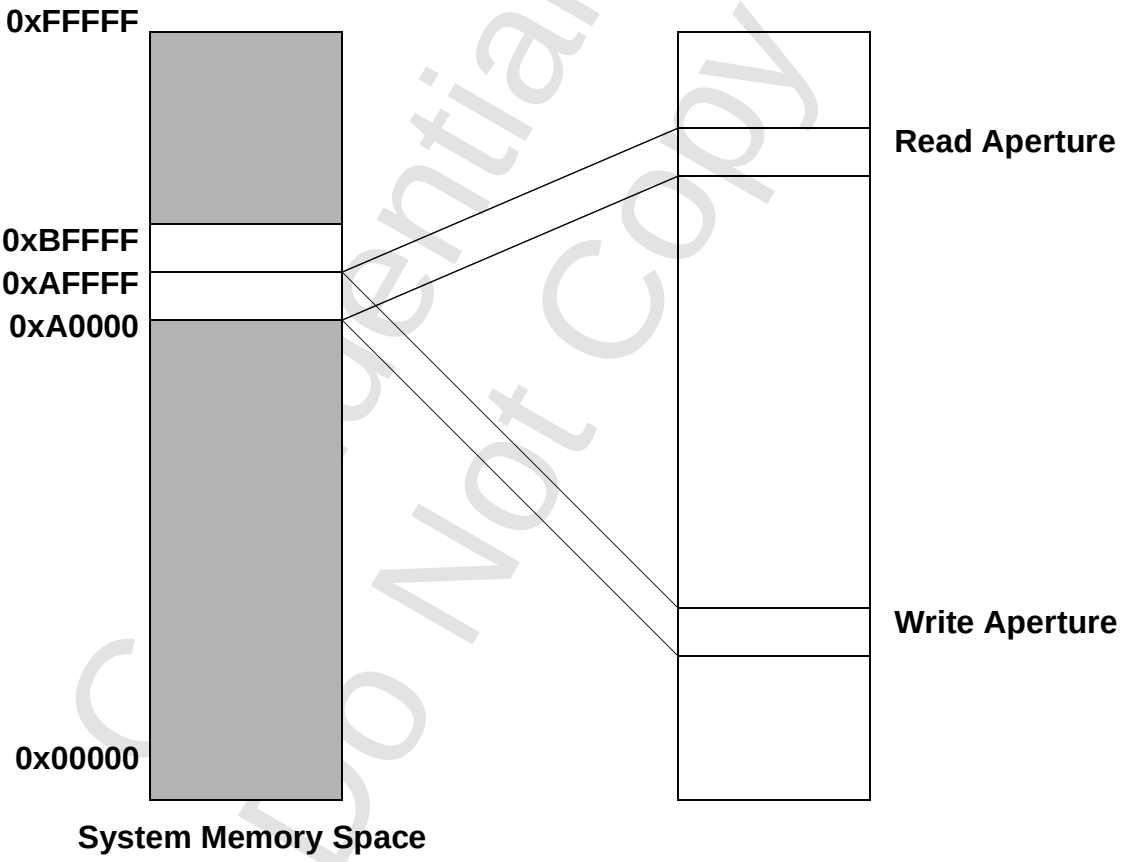



**Figure 4.2 Remapping in VESA Modes**

# 5 PCI Configuration Registers

## 5.1 Overview

Voodoo3 supports the PCI/AGP host bus with the normal configuration registers.
Table 5.1 contains a summary of the PCI registers. The link is a clickable link to the detailed description.
Unless otherwise noted, all non-reserved fields in all registers are read/write.

**Table 5.1 PCI Configuration Register Summary**

| Register Name | PCI Config | Link |
|---|---|---|
| PCI00: Device/Vendor ID | 0x00 | Section 5.2.1 |
| PCI04: Status/Command | 0x04 | Section 5.2.2 |
| PCI08: Class Code | 0x08 | Section 5.2.3 |
| PCI0C: BIST | 0x0C | Section 5.2.4 |
| PCI10: memBaseAddr0 | 0x10 | Section 5.2.5 |
| PCI14:memBaseAddr1 | 0x14 | Section 5.2.6 |
| PCI18: ioBaseAddr | 0x18 | Section 5.2.7 |
| PCI2C: subSystem ID | 0x2C | Section 5.2.8 |
| PCI30: romBaseAddr | 0x30 | Section 5.2.9 |
| PCI34: Capabilities Pointer | 0x34 | Section 5.2.10 |
| PCI3C: Interrupt | 0x3C | Section 5.2.11 |
| PCI40: fabID | 0x40 | Section 5.2.11 |
| PCI4C: cfg Status | 0x4C | Section 5.2.13 |
| PCI50: cfg Scratch | 0x50 | Section 5.2.14 |
| PCI54: AGP Capabilities ID | 0x54 | Section 5.2.15 |
| PCI58: AGP Status Register | 0x58 | Section 5.2.16 |
| PCI5C: AGP Command | 0x5C | Section 5.2.17 |
| PCI60: ACPI Capabilities ID | 0x60 | Section 5.2.18 |
| PCI64: ACPI Control/Status | 0x64 | Section 5.2.19 |

## 5.2 PCI Configuration Register Details

The PCI configuration registers are described as 32-bit registers since this is how they are normally accessed.

### 5.2.1 PCI00: Device/Vendor ID

This read-only register supplies the 16-bit device ID and the 16-bit vendor ID.

**Bit** **Description**

31:16 **Device ID:** This read-only field supplies the device ID assigned to Voodoo3 by 3Dfx Interactive, Inc. This returns the value indicated in the table.

| Value | Description | pllCtrl1[7:2] |
|---|---|---|
| ~~0x0004~~ | ~~Not so fast~~ | ~~Forced to 0x24~~ |
| 0x0005 | Speed Sorted | Programmable |

15:0 **Vendor ID:** This read-only field supplies the vendor ID assigned to 3Dfx Interactive, Inc. by PCISIG. This returns a value of 0x121A.

### 5.2.2 PCI04: Status/Command

The high order 16 bits provide status information for PCI related events. The low order 16 bits provide coarse control over the ability of the Voodoo3 to generate and respond to PCI cycles.

**Bit** **Description**

31 **Detected Parity Error:** This read-only bit is set to '1' when Voodoo3 detects a parity error during a bus transfer. This bit is cleared to 0 by writing to this register.

30 **Signaled System Error:** This read-only bit always returns the value 0.

29 **Received Master Abort:** This read-only bit always returns the value 0.

28 **Received Target Abort:** This read-only bit always returns the value 0.

27 **Signaled Target Abort:** This read-only bit always returns the value 0.

26:25 **DEVSEL Timing:** The value returned in this read-only field depends on configuration bit 0 (VMI_HD0).

| VMI_HD0 | Value Returned | Timing |
|---|---|---|
| Low | 01b | Medium |
| High | 00b | Fast |

24 **Data Parity Error Detected:** This read-only bit always returns the value 0.

23 **Fast Back-to-back Capable:** This read-only bit always returns the value 0.

22 **UDF Supported:** This read-only bit always returns the value 0.

### 5.2.2 PCI04: Status/Command *(cont)*

| Bit | Description |
|---|---|
| 21 | **66 MHz Capable:** This read-only bit returns the value of strapping pin 2 (VMI_HD2). The value ‘0’ indicates 33 MHz; the value ‘1’ indicates 66 MHz. |
| 20 | **New Capabilities:** This read-only bit returns the value of strapping pin 3 (VMI_HD3). The value ‘0’ indicates AGP disabled; the value ‘1’ indicates AGP enabled. |
| 19:16 | **Reserved:** These read-only bits always return the value 0. |
| 15:9 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 8 | **System Error Enable:** This bit is read-only 0. |
| 7 | **Wait Cycle Enable:** |
| 6 | **Parity Error Response:** This bit is read-only 0. |
| 5 | **Palette Snoop Enable:** This bit is read-only 0. |
| 4 | **Memory Write and Invalidate Enable:** This bit is read-only 0. |
| 3 | **Special Cycle Recognition:** This bit is read-only 0. |
| 2 | **Bus Master Enable:** This bit is read-only 0. |
| 1 | **Memory Access Enable:** If this bit is 0, Voodoo3 will not respond to any memory space accesses. If this bit is ‘1’, Voodoo3 will respond to memory space accesses at the appropriate addresses. |
| 0 | **I/O Access Enable:** If this bit is 0, Voodoo3 will not respond to any I/O space accesses. If this bit is ‘1’, Voodoo3 will respond to I/O space accesses at the appropriate addresses. |

### 5.2.3 PCI08: Class Code/Revision ID

This read-only register contains the class code and the revision ID.

**Bit** **Description**

31:8 **Class Code:** The value returned in this read-only field depends on configuration bit 4 (VMI_HD4).

| VMI_HD4 | Class Code | Meaning |
|---|---|---|
| Low | 0x030000 | VGA Compatible Device |
| High | 0x048000 | Other Multimedia Device |

7:0 Revision ID: This read-only returns Voodoo3 Revision Identification. The following IDs have been assigned.

| ID | Meaning |
|---|---|
| 0x00 | (not used) |
| 0x01 | .35 micron |
| 0x02 | .25 micron |

### 5.2.4 PCI0C: BIST

Only a single bit in the Header Type field is assigned.

| Bit | Description |
|---|---|
| 31:24 | **BIST:** This read-only field always returns the value 0. |
| 23:16 | **Header Type:** Bits 6:0 of this field (register bits 22:16) are read-only and return the value '0'. Bit 7 of this field (register bit 23) specifies Voodoo3 as a single function PCI device (read only?). |
| 15:8 | **Latency Timer:** This read-only field always returns the value 0. |
| 7:0 | **Cache Line Size:** This read-only field always returns the value 0. |

### 5.2.5 PCI10: memBaseAddr0

This base register specifies the first base address for the memory space.

| Bit | Description |
|---|---|
| 31:25 | **Base Address:** This read-write field is programmed at POST time to the beginning address of the first memory space. |
| 24:4 | **Reserved:** This read-only field returns the value 0, indicating that this register claims 32 Mbytes. The memory space is allocated as indicated in the table. The link is a clickable link to the description of the resource. |

| Offset | Size | Function | Link |
|---|---|---|---|
| 0x0000 0000 | 512 Kbytes | I/O Register Remap | Chapter 6 |
| 0x0008 0000 | 512 Kbytes | CMD/AGP Transfer/Misc. registers | Chapter 11 |
| 0x0010 0000 | 1 Mbyte | 2D Registers | Chapter 7 |
| 0x0020 0000 | 4 Mbytes | 3D Registers | Chapter 9 |
| 0x0060 0000 | 2 Mbytes | TMU0 Texture Download | |
| 0x0080 0000 | 2 Mbytes | TMU1 Texture Download | |
| 0x00A0 0000 | 2 Mbytes) | FLASH Access to BIOS ROM | - |
| 0x00C0 0000 | 4 Mbytes | YUV Planar space | |
| 0x0100 0000 | 16 Mbytes | 3D Linear Frame Buffer space | |

| | |
|---|---|
| 3 | **Prefetchable:** This bit is read-only and always returns the value 0 to indicate this space is not prefetchable. |
| 2:1 | **Size:** This field is read-only and always returns the value 00b to indicate this base register is 32 bits wide and mapping can be done anywhere in the 32-bit memory space. |
| 0 | **Memory/IO:** This bit is read-only and always returns the value 0 to indicate this base register is claiming a memory space. |

### 5.2.6 PCI14: memBaseAddr1

This base register specifies the second base address for the memory space. The aperture claimed in this register depends on configuration bits [6:5] (VMI_HD[6:5]).

**Bit** **Description**

31:24 **Base Address:** This read-write field is programmed at POST time to the beginning address of the second memory space.

| VMI_HD6 | VMI_HD5 | Frame Buffer Size | LSB of Base Address |
|---|---|---|---|
| Low: 8 Mbit | Low: 4 chips | 4 Mbytes | 22 |
| Low: 8 Mbit | High: 8 chips | 8 Mbytes | 23 |
| High: 16 Mbit | Low: 4 chips | 8 Mbytes | 23 |
| High: 16 Mbit | High: 8 chips | 16 Mbytes | 24 |

23:4 **Reserved:** This read-only field returns the value 0. The size of this field depends on device strapping, as previously indicated. This space is used to access the frame buffer.

3 **Prefetchable:** This bit is read-only and always returns the value 0 to indicate this space is not prefetchable.

2:1 **Size:** This field is read-only and always returns the value 00b to indicate this base register is 32 bits wide and mapping can be done anywhere in the 32-bit memory space.

0 **Memory/IO:** This bit is read-only and always returns the value 0 to indicate this base register is claiming a memory space.

### 5.2.7 PCI18:ioBaseAddr

This base register specifies the base address for PCI I/O accesses. See Table 6.1.

**Bit** **Description**

31:8 **Base Address:** This read-write field is programmed at POST time to the beginning address of the I/O space.

7:2 **Reserved:** this read-only field returns the value 0, indicating that this register claims 256 bytes. This space is used to access the registers in I/O space.

1 **Reserved:** This read-only bit always returns the value 0.

0 **Memory/IO:** This read-only bit always returns the value 1 to indicate this base register is claiming an I/O space.

### 5.2.8 PCI2C: subSystem ID

This register is loaded from the ROM. It contains the subsystem Vendor ID and the Subsystem ID.

| Bit | Description |
|---|---|
| 31:16 | **subSystem ID:** This field is loaded from the ROM during system initialization. |
| 15:0 | **subVendor ID:** This field is loaded from the ROM during system initialization. The table indicates the source of each byte based on whether a 15- or 16-bit address bus is used. Voodoo3 generates addresses with the high order bit set regardless of the ROM size. |

| Byte | 15-bit address | 16-bit address |
|---|---|---|
| 31:24 | 0x7FFB | 0xFFFB |
| 23:16 | 0x7FFA | 0xFFFA |
| 15:8 | 0x7FF9 | 0xFFF9 |
| 7:0 | 0x7FF8 | 0xFFF8 |

### 5.2.9 PCI30: Expansion ROM Base Address

This register contains the address of the expansion ROM.

| Bit | Description |
|---|---|
| 31:16 | **Base Address:** This field is programmed at POST time to the address of the expansion ROM. |
| 15 | **Base Address:** This bit is controlled by strapping pin 1 (VMI_HD1). Note that the strapping option can be overidden by writing miscInit1[25]. |

| VMI_HD1 | ROM Size | Description |
|---|---|---|
| 1 | 64K | This bit is read-only 0. |
| 0 | 32K | This bit is read/write and is programmed to the low order bit of the ROM base address. |

| | |
|---|---|
| 14:1 | **Reserved:** This read-only field always returns the value 0, indicating that this register is claiming at least 32 Kbytes. |
| 0 | **Enable:** If this bit is 1, the expansion ROM address decoder is enabled. |

### 5.2.10 PCI34: Capabilities Pointer

This register contains the offset of the beginning of the capabilities linked list structure.

| Bit | Description |
|---|---|
| 31:24 | **Reserved:** This field returns the value 0. |
| 7:0 | **List Offset:** This read-only field returns the value 0x54. If the Voodoo3 is not strapped for AGP, PCI04[20] is zero and this read-only field returns 0x60. |

### 5.2.11 PCI3C: Interrupt Register

This register contains interrupt and bus mastering configuration.

| Bit | Description |
|---|---|
| 31:24 | **Max Latency:** This read-only field returns the value 0. |
| 23:16 | **Min Grant:** This read-only field returns the value 0. |
| 15:8 | **Interrupt Pin:** This read-only field returns the value 0x01 (indicating INTA). |
| 7:0 | **Interrupt Line:** The contents of this read-write field has no direct effect on the operation of Voodoo3. |

### 5.2.12 PCI40: Fab ID

This register returns the identification of a semiconductor fabrication facility.

| Bit | Description |
|---|---|
| 31:4 | **Scratch Pad:** The value programmed into this read-write field has no direct effect on the operation of Voodoo3. This field is reserved for the exclusive use of 3Dfx Interactive, Inc. software. |
| 3:0 | **Fab ID:** This read-only field returns a value which identifies the semiconductor fabrication facility. No application program should take any action based on the contents of this field. |

### 5.2.13 PCI4C: cfgStatus

This read-only register is a copy of the status register at offset 0 of the I/O mapped register space.

| Bit | Description |
|---|---|
| 31:0 | **status:** This read-only field returns the contents of the status register. See Section 6.2.1. |

### 5.2.14 PCI50: cfgScratch

This is a scratch pad register in the configuration space.

| Bit | Description |
|---|---|
| 31:0 | **Scratch Pad:** The value programmed into this read-write field has no direct effect on the operation of Voodoo3. This field is reserved for the exclusive use of 3Dfx Interactive, Inc. software. |

### 5.2.15 PCI54: AGP Capability ID Register

This register contains the AGP capability identifier.

| Bit | Description |
|---|---|
| 31:24 | **Reserved:** This read-only field returns the value 0. |
| 23:20 | **Major:** This read-only field returns the value xxxxb to indicate the major revision of the AGP specification to which Voodoo3 conforms. |
| 19:16 | **Minor:** This read-only field returns the value xxxxb to indicate the minor revision of the AGP specification to which Voodoo3 conforms. |
| 15:8 | **Next Pointer:** This read-only field returns the value 0x60 to point to the next capability ID. |
| 7:0 | **Capability ID:** This read-only field returns the value 0x02 to indicate AGP. |

### 5.2.16 PCI58: AGP Status

This register indicates the specifics of the capabilities of the AGP interface.

| Bit | Description |
|---|---|
| 31:24 | **RQ_DEPTH:** This read-only field returns the value 0x07 to indicate the Voodoo3 can handle up to seven AGP command requests. |
| 23:10 | **Reserved:** These read-only bits return 0. |
| 9 | **SBA:** This read-only bit returns the value 1 to indicate Voodoo3 supports sideband addressing. |
| 8:6 | **Reserved:** These read-only bits return 0. |
| 5 | **AGP_4G:** This read-only bit returns the value 1 to indicate Voodoo3 AGP supports above 4 Gbytes of memory. |
| 4:3 | **Reserved:** These read-only bits return 0. |
| 2:0 | **Rate:** This read-only field returns the value 011b to indicate Voodoo3 supports 1x and 2x (but not 4x) transfer rates. |

### 5.2.17 PCI5C: AGP Command

This read-write register specifies the AGP capability that the system is prepared to support.

| Bit | Description |
|---|---|
| 31:24 | **RQ_DEPTH:** This read-write field is programmed to the maximum number of pipelined operations Voodoo3 is allowed to enqueue. |
| 23:10 | **Reserved:** These read-only bits return 0. |
| 9 | **SBA:** When this bit is '1', Voodoo3 is allowed to use sideband addressing. The default for this bit is 0. |
| 8 | **AGP_ENABLE:** When this bit is '1', Voodoo3 is allowed to initiate AGP operations. The default for this bit is 0. |
| 7:6 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 5 | **AGP_4G Enable:** When this bit is '1', Voodoo3 is allowed to address above 4 Gbytes of AGP memory. The default for this bit is 0. |
| 4:3 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 2:0 | **Rate:** This field is programmed to specify the desired data transfer rate. Bit zero specifies 1x, bit one specifies 2x, and bit two specifies 4x. Since Voodoo3 does not support 4x, bit two must not be set. |

### 5.2.18 PCI60: ACPI Capability ID Register

This register contains the ACPI capability identifier.

| Bit | Description |
|---|---|
| 31:27 | **PME Support:** This read-only field returns the value 0. |
| 26 | **D2 Support:** This read-only bit returns the value 0. |
| 25 | **D1 Support:** This read-only bit returns the value 0. |
| 24:22 | **Reserved:** This read-only field returns the value 0. |
| 21 | **DS1:** This read-only bit returns the value ‘1’ to indicate that additional software initialization must take place. |
| 20 | **Aux Power Source:** This read-only bit returns the value 0. |
| 19 | **PME Clock:** This read-only bit returns the value 0. |
| 18:16 | **Version:** This read-only field returns the value 001b. |
| 15:8 | **Next Pointer:** This read-only field returns the value 0x00 to indicate no further entries in the capabilities linked list. |
| 7:0 | **Capability ID:** This read-only field returns the value 0x01 to indicate ACPI. |

### 5.2.19 PCI64: ACPI Command/Status/

This register allows transitions between D3 and D0 power state. Voodoo3 does not support the data register or PME#.

| Bit | Description |
|---|---|
| 31:24 | **Data:** This read-only field returns the value 0x00. |
| 23 | **BPCC Enable:** This read-only bit returns the value 0 to indicate Voodoo3 does not support bus power/clock policies defined in the PCI specification. |
| 22 | **B2_B3# Support:** This read-only bit returns the value 0. |
| 21:16 | **Reserved:** This read-only field returns the value 0. |
| 15 | **PME Status:** This read-only bit returns the value 0. |
| 14:13 | **Data Scale:** This read-only field returns the value 0. |
| 12:9 | **Data Select:** This read-only field returns the value 0. |
| 8 | **PME Enable:** This read-only bit returns the value 0. |
| 7:2 | **Reserved:** This read-only field returns the value 0. |
| 1:0 | **Power State:** This field is written to command Voodoo3 into a specific power state and returns the current power state when read. Voodoo3 supports only D0 and $D3_{hot}$. A write of 01b or 10b to this field will complete normally. However, the data will be discarded and no state change occurs. |

# 6 IO-Based Registers

## 6.1 Overview

The following registers are accessible in the address space claimed in ioBaseAddr (PCI18). These registers (with the exception of the VGA core set) are also accessible in memory space at the very beginning of the space claimed in memBaseAddr0 (PCI10).
Table 6.1 contains a summary of the I/O registers. The registers are ordered by I/O offset. The link is a clickable link to the detailed description. Unless otherwise noted, all non-reserved fields in all registers are read/write.

### Table 6.1 I/O Register Summary

| Register Name | I/O Offset | Link |
|---|---|---|
| status register | 0x00 | Section 6.2.1 |
| pciInit0 | 0x04 | Section 6.3.1 |
| sipMonitor | 0x08 | Section 6.3.2 |
| lbfMemoryConfig | 0x0C | Section 6.3.3 |
| miscInit0 | 0x10 | Section 6.3.4 |
| miscInit1 | 0x14 | Section 6.3.5 |
| dramInit0 | 0x18 | Section 6.3.6 |
| dramInit1 | 0x1C | Section 6.3.7 |
| agpInit | 0x20 | Section 6.3.8 |
| tmuGbeInit | 0x24 | Section 6.3.9 |
| vgaInit0 | 0x28 | Section 6.3.10 |
| vgaInit1 | 0x2C | Section 6.3.11 |
| 2DCommand | 0x30 | Section 6.3.12 |
| 2DsrcBaseAddr | 0x34 | Section 6.3.13 |
| strapInfo | 0x38 | - |
| vidTVOutBlankVCount | 0c3C | Section 6.5.1 |
| pllCtrl0 (video clock) | 0x40 | Section 6.4.1 |
| pllCtrl1 (GRX/Memory clock) | 0x44 | Section 6.4.1 |
| pllCtrl2 (AGP clock) | 0x48 | Section 6.4.2 |
| dacMode | 0x4C | Section 6.4.3 |
| dacAddr | 0x50 | Section 6.4.4 |
| dacData | 0x54 | Section 6.4.5 |

**Table 6.1 I/O Register Summary *(cont.)***

| Register Name | I/O Offset | Link |
|---|---|---|
| vidrgbMaxDelta | 0x58 | Section 6.5.2 |
| vidProcCfg | 0x5C | Section 6.5.3 |
| hwCurPatAddr | 0x60 | Section 6.5.4 |
| hwCurLoc | 0x64 | Section 6.5.5 |
| hwCurC0 | 0x68 | Section 6.5.6 |
| hwCurC1 | 0x6C | Section 6.5.6 |
| vidInFormat | 0x70 | Section 6.5.7 |
| vidTVOutblankHCount | 0x74 | Section 6.5.8 |
| vidSerialParallelPort | 0x78 | Section 6.5.9 |
| vidInXDecimDeltas | 0x7C | Section 6.5.10 |
| vidInDecimInitErrs | 0x80 | Section 6.5.11 |
| vidInYdecimDeltas | 0x84 | Section 6.5.12 |
| vidPixelBufThold | 0x88 | Section 6.5.13 |
| vidChromaKeyMin | 0x8C | Section 6.5.14 |
| vidChromaKeyMax | 0x90 | Section 6.5.14 |
| vinInStatusCurrentLine | 0x94 | Section 6.5.15 |
| vidScreenSize | 0x98 | Section 6.5.16 |
| vidOverlayStartCoords | 0x9C | Section 6.5.17 |
| vidOverlayEndScreenCoord | 0xA0 | Section 6.5.18 |
| vidOverlayDudx | 0xA4 | Section 6.5.19 |
| vidOverlayDudxOffsetScrWidth | 0xA8 | Section 6.5.20 |
| vidOverlayDvDy | 0xAC | Section 6.5.21 |
| VGA Registers | 0xB0 - 0xDC | Chapter 3<br>Section 6.6 (ref) |
| vidOverlayDvdyOffset | 0xE0 | Section 6.7.1 |
| vidDesktopStartAddr | 0xE4 | Section 6.7.2 |
| vidDesktopOverlayStride | 0xE8 | Section 6.7.3 |
| vidInAddr0 | 0xEC | Section 6.7.4 |
| vidInAddr1 | 0xF0 | Section 6.7.4 |

**Table 6.1 I/O Register Summary *(cont.)***

| Register Name | I/O Offset | Link |
|---|---|---|
| vidInAddr2 | 0xF4 | Section 6.7.4 |
| vidInStride | 0xF8 | Section 6.7.5 |
| vidCurrOverlayStartAddr | 0xFC | Section 6.7.6 |

## 6.2 Status Register

### 6.2.1 status Register

This is the Voodoo3 status register. This register is also accessible at PCI4C. This register is nominally read-only. Writing to this register address clears any pending PCI interrupts.

| Bit | Description |
|---|---|
| 31 | **PCI Interrupt Generated:** If this bit is '1', it indicates a vertical retrace interrupt is pending. A write of any value clears this bit. |
| 30:28 | **Swap Buffers Pending:** This field maintains a count of the SWAPBUFFER commands that are pending. When a SWAPBUFFER command is received from the host, this field is incremented. When a SWAPBUFFER command completes, this field is decremented. |
| 27:13 | **Reserved:** These bits always return the value 0. |
| 12 | **Cmd FIFO 1 Busy:** When this bit is '1', command FIFO 1 is active. |
| 11 | **Cmd FIFO 0 Busy:** When this bit is '1', command FIFO 0 is active. |
| 10 | **2D Busy:** When this bit is '1', the 2D graphics engine is busy. When this bit is '0', the 2D graphics engine is idle. |
| 9 | **Device Busy:** When this bit is '1', at least one internal unit is active (for example, graphics are being rendered or one or more FIFOs are active). When this bit is '0', the entire Voodoo3 is idle. Read Section 12.2 before writing any code involving this bit. |
| 8 | **TREX Busy:** when this bit is '1', at least one unit in TREX is active. This includes the graphics engine and all internal TREX FIFOs. |
| 7 | **FBI Graphics Engine Busy:** When this bit is '1', the graphics engine in the FBI is active. Note that this bit does not reflect the status of the internal PCI FIFOs. |
| 6 | **Vertical Retrace:** When this bit is '1', vertical retrace is active. When this bit is '0', vertical retrace is not active and the screen is being refreshed. |
| 5 | **PCI FIFO Busy:** When this bit is '1', the PCI FIFO is active. |
| 4:0 | **PCI FIFO Free Entries:** This field indicates the number of entries available in the internal host FIFO. When this field returns the value 0x1F, the FIFO is altogether empty. |

## 6.3 Initialization Registers

### 6.3.1 pciInit0

pciInit0 specifies the details of how the PCI interface is to behave.

| Bit | Description |
|---|---|
| 31:29 | **Reserved:** Reserved fields must be programmed to '0' for upward compatibility. |
| 28:27 | PCI FIFO to LFB Write Timeout: This field specifies the number of MCLK cycles that must elapse before a write timeout is declared in a PCI FIFO to lfb. |

| Value | Timeout |
|---|---|
| 00b | 64 |
| 01b | 128 |
| 10b | 196 |
| 11b | 256 |

| | |
|---|---|
| 26 | **High Priority:** When this bit is '1', it forces PCI/CMD frame buffer accesses to high priority. PCI frame buffer reads are always high priority. |
| 25:20 | **PCI FIFO Read Threshold:** This fields specifies the number of non-modal linear frame buffers accesses that are grouped together before being pushed to memory. This field resets to 1 1000b. |
| 19 | **PCI Interrupt Time-out Enable:** When this bit is '1', the PCI Interrupt Time-out is enabled. This bit resets to '0'. |
| 18 | **PCI Interrupt Enable:** When this bit is '1', the PCI interrupt is enabled. This bit resets to '0'. |
| 17:13 | **Retry Interval:** This field specifies the number of PCI clocks before a retry is attempted. The actual number of clocks is eight greater than specified in this field. This field resets to 0. |
| 12 | **Disable PCI Memory Access Retries:** When this bit is '1', PCI memory access retries will not be attempted. |
| 11 | **Disable PCI IO Access Retries:** When this bit is '1', PCI I/O access retries will not be attempted. |
| 10 | **Reserved:** Reserved bits must be programmed to 0 for upward compatibility. |
| 9 | **Wait States for PCI Write Accesses:** This bit specifies the number of wait cycles that will be inserted for PCI write accesses. Zero specifies no wait states; one specifies one wait state. |
| 8 | **Wait States for PCI Read Accesses:** This bit specifies the number of wait cycles that will be inserted for PCI read accesses. Zero specifies one wait state; one specifies two wait states. |

### 6.3.1 pciInit0 *(cont)*

| Bit | Description |
|---|---|
| 7 | **Reserved:** Reserved bits must be programmed to 0 for upward compatibility. |
| 6:2 | **PCI FIFO Low Water Mark:** This field specifies to PCI FIFO Empty Entries Low Water Mark (the minimum number of ?). The default value for this field is 0x10. This field should never be programmed to zero and should be greater than two for any fast device operations. |
| 1:0 | **Reserved:** Reserved fields must be programmed to 0 for upward compatibility. |

### 6.3.2 sipMonitor Register

This register contains silicon performance counters. No application program should ever write this register or take any action based on the contents of any field in this register. Do not enable the frequency counter (bit 30) without prior written consent of 3Dfx Interactive, Inc.

| Bit | Description |
|---|---|
| 31 | **Reserved:** Reserved bits must be programmed to 0 for upward compatibility. |
| 30 | **Frequency Counter Enable:** When this bit is '0', the frequency counter is disabled. This is the default state of this bit. When this bit is '1', the frequency counter is enabled. Enable this counter at your peril. The oscillator generates lots of heat. |
| 29 | **Oscillator NOR Select:** This bit selects the gate type for the oscillator chain, according to the table. |

| Select | Gate Type | Transistor |
|---|---|---|
| 0 | NAND | P |
| 1 | NOR | N |

| | |
|---|---|
| 28 | **Oscillator Counter Reset:** When this bit is '0', the oscillator counter is cleared to zero. |
| 27:16 | **Oscillator Counter:** This counter is clocked by a ring oscillator. Silicon performance can be measured by comparing the contents of this counter to the contents of the GRX counter. |
| 15:0 | **GRX Counter:** This counter is clocked by the GRX clock. It is used as a reference. |

### 6.3.3 lfbMemoryConfig

This register defines the tile beginning page in SGRAM space. This register is read-write.

| Bit | Description |
|---|---|
| 31:23 | **Reserved:** Reserved fields must be programmed to 0 for upward compatibility. |
| 22:16 | **SGRAM Tiles X:** This field specifies the number of SGRAM tiles in X (or the stride?). This field defaults to 0x0A. |
| 15:13 | **Tile Aperture Stride:** This field specifies the tile stride in bytes and is used for PCI-to-XY translation. |
| 12:0 | **Tile Aperture Begin Page:** This field specifies the first page of tile addressing. |

### 6.3.4 miscInit0

This register contains subsystem resets and some miscellaneous controls.

| Bit | Description |
|---|---|
| 31 | **Word Swizzle non-modal LFB Writes:** When this bit is '1', word swizzling is enabled for incoming non-modal linear frame buffer writes. The default value for this bit is 0. |
| 30 | **Byte Swizzle non-modal LFB Writes:** When this bit is '1', byte swizzling is enabled for incoming non-modal linear frame buffer writes. The default value for this bit is 0. |
| 29:18 | **Y Origin Swap Subtraction Value:** This value is used during address calculation when Y flip is enabled in fbzMode. The reset value for this field is 0. |
| 17 | **Reserved:** Reserved bits must be programmed to '0' for upward compatibility. |
| 16:14 | **DAC Delay Compensation:** This field specifies a programmable delay inserted in the video timing signal (vsync and hsync) paths to match the DAC delay. This delay is in addition to the CLUT Delay Compensation. This delay is specified in terms of pixel clocks and may be programmed to any value in the range of 0 to 7. |
| 13:11 | **CLUT Delay Compensation:** This field specifies a programmable delay inserted in the vsync and hsync paths to match the CLUT delay. This delay is specified in terms of pixel clocks and may be programmed to any value in the range of 0 to 7. |
| 13:11 | **CLUT Delay Compensation:** This field specifies a programmable delay inserted in the blank path to match the CLUT delay. This delay is specified in terms of pixel clocks and may be programmed to any value in the range of 0 to 7. |
| 7 | **VGA Video Timing Reset:** When this bit is '1', the VGA video timing logic is reset. The default state of this bit is 0. |
| 6 | **Memory Timing Reset:** When this bit is '1', the memory timing logic is reset. The default state of this bit is 0. |
| 5 | **2D Graphics Reset:** When this bit is '1', the 2D graphics logic is reset. The default state of this bit is 0. |
| 4 | **Video Timing Reset:** When this bit is '1', the video timing logic is reset. The default state of this bit is 0. |
| 3 | **Word Swizzle Register Writes:** When this bit is '1', word swizzling is enabled for incoming register writes if bit 20 of the PCI address is '1' for 3D registers and bit 19 of the PCI address is '1' for 2D registers. The default state of this bit is 0. |
| 2 | **Byte Swizzle Register Writes:** When this bit is '1', byte swizzling is enabled for incoming register writes if bit 20 of the PCI address is '1' for 3D registers and bit 19 of the PCI address is '1' for 2D registers. The default state of this bit is 0. |
| 7 | **FBI FIFO Reset:** When this bit is '1', the PCI FIFO and PCI data packer is reset. The default state of this bit is 0. |
| 0 | **FBI Graphics Reset:** When this bit is '1', the FBI graphics logic is reset. The default state of this bit is 0. |

### 6.3.5 miscInit1

This register contains a number of additional miscellaneous controls that are usually programmed only when the graphics system is initialized.

| Bit | Description |
|---|---|
| 31:29 | **Reserved:** Reserved bits must always be written as 0 for upward compatibility. |
| 28:24 | **Strapping:** This field returns some of the strapping bits. This field is read-write; changing bits in this field will change corresponding values in the PCI configuration registers. |

| Bits | Strapping Option | Meaning | PCI Configuration Register |
|---|---|---|---|
| 28 | 4 | PCI Device Type | PCI08 (class code) |
| 27 | 3 | AGP Enabled | PCI04 (new capabilities) |
| 26 | 2 | PCI 66 MHz | PCI04 (66 MHz capable) |
| 25[a] | 1 | BIOS Size | PCI30 (aperture size) |
| 24 | 0 | DEVSEL timing | PCI04 (DEVSEL timing) |

a. This bit must cleared to access the 2nd 32K of a 64K ROM. There is apparently a net inversion between the strapping pin and this bit.

| | |
|---|---|
| 23:20 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 19 | **Command Stream Reset:** If this bit is ‘1’, the command stream is reset. Commands which have not be executed will be discarded. This bit should only be set when Voodoo3 is idle. This bit must be cleared after being set. |
| 18 | **Disable 3D Stall on 2D Synchronous Dispatch:** When this bit is ‘1’, 3D will not wait on pending 2D operations to complete before being issued. This bit is for testing and should never be set during normal operation. The default for this bit is 0. |
| 17 | **Disable 2D Stall on 3D Synchronous Dispatch:** When this bit is ‘1’, 2D will not wait on pending 3D operations to complete before being issued. This bit is for testing and should never be set during normal operation. The default for this bit is 0. |
| 16 | **Disable 3D Block Write:** The default for this bit is 0. |
| 15 | **Disable 2D Block Write:** The default for this bit is 0. |
| 14:12 | **Block Write Threshold:** The default value for this field is 0. |
| 10 | **Power Down Graphics/Memory PLL:** When this bit is ‘1’, the PLL that generates the graphics clock and memory clock is powered down. Writing a 0 to this bit will cause the graphics PLL to power back up. The pllCtrl register will have retained its programming, so the pll will operate at the previous frequency. |

### 6.3.5 miscInit1 *(cont)*

| Bit | Description |
|---|---|
| 9 | **Power Down Video PLL:** When this bit is '1', the PLL that generates the video clock is powered down. Writing a 0 to this bit will cause the video PLL to power back up. The pllCtrl register will have retained its programming, so the pll will operate at the previous frequency. |
| 8 | **Power Down DAC:** When this bit is '1', the DAC is powered down. |
| 7 | **Power Down CLUT:** When this bit is '1', the color look-up table is powered down. It is (not) necessary to re-program the CLUT when it is powered back up. |
| 6 | **Disable Texture Mapping:** When this bit is '1', texture mapping is disabled altogether. When this bit is 0, the texture mapping logic functions normally. |
| 5 | **Enable Alternate Triangle Addressing Map:** When this bit is '1', triangle address aliasing is enabled, allowing better address compaction. See the register tables in Chapter 9. |
| 4 | **Enable ROM Writes:** When this bit is '1', writes to the ROM address range will generate ROM_WE with ROM_CS. Each DWORD write on the PCI will be turned into four byte writes at the ROM interface. |
| 3 | **Enable subSystem ID Writes:** When this bit is '1', PCI2C is read-write. When this bit is 0, PCI2C is read-only. |
| 2:1 | **Tri Mode:** This field yadda yadda. |
| 0 | **Invert CLUT Address:** When this bit is '1', it corrects for the CLUT address being inverted during host accesses. This bit should be '1' for normal operation. |

### 6.3.6 dramInit0

This register specifies timing parameters for the SGRAM interface state machine. The default value of this register is 0x00579D29. See Chapter 20.

| Bit | Description |
|---|---|
| 31:28 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 27 | **Device Size:** This bit is 0 for 8 Mbit SGRAMs and 1 for 16 Mbit SGRAM/SDRAMs. The default value for this bit is configured with VMI_HD6 |
| 26 | **Number of SGRAM ChipSets:** When this bit is 0, it indicates a single SGRAM chipset. MCS 0 is used; MCS1 is a no connect. When this bit is '1', it indicates two SGRAM chipsets. The default value for this bit is configured with VMI_HD5. |
| 25 | **SGRAM Write Per Bit Enable:** When this bit is 0, write-per-bit is disabled. When this bit is '1', write-per-bit is enabled. The default is 0. |
| 24 | **Disable Dead Bus Cycle:** When this bit is 0, one dead cycle between a SGRAM read and SGRAM write will be inserted (to allow for bus turn-around). When this bit is '1', no dead cycle is inserted. The default is 0. |
| 23 | **Disallow WR/BK/WR:** When this bit is 0, the WR/BK/WR sequence is allowed to terminate a read. When this bit is '1', the sequence is not allowed to terminated a read. The default is 0. |
| 22 | **tRL:** This bit specifies the RD to PRE period. A 0 specifies one clock; a '1' specifies two clocks. The default is 1. This bit must be programmed to '1' if the SGRAM does *not* do pipelined precharge, and the CAS latency is three. If the SGRAM does do pipelined precharge (the only known device is from IBM) or if the CAS latency is two, this bit must be programmed to 0. |
| 21:20 | **tBWL:** This field specifies the BKWR to PRE period, according to the table. |

| tBWL | Clocks | Note |
|---|---|---|
| 00 | 1 | |
| 01 | 2 | Default |
| 10 | 3 | |
| 11 | 4 | |

| | |
|---|---|
| 19 | **tWL:** This bit specifies the WR to PRE period. A 0 specifies one clock; a '1' specifies two clocks. The default is 0. |
| 18 | **tBWC:** This bit specifies the block write cycle time. A 0 specifies one clock; a '1' specifies two clocks. The default is 1. |
| 17 | **tDQR:** This bit specifies the RD to DQM assertion delay. A 0 specifies zero clocks; a '1' specifies one clock. The default is 1. |
| 16 | **tMRS:** This bit specifies mode and special mode register cycle time. A 0 specifies one clock; a '1' specifies two clocks. The default is 1. |

### 6.3.6 dramInit0 *(cont)*

| Bit | Description |
|---|---|
| 15:14 | **tCAS Latency:** This field specifies the CAS latency period, according to the table. |

| tCAS | Clocks | Note |
|---|---|---|
| 00 | 1 | |
| 01 | 2 | |
| 10 | 3 | Default |
| 11 | 4 | |

| | |
|---|---|
| 13:10 | **tRC:** This field specifies the minimum row cycle time (1 through 16 clocks). The default for this field is 0111b (eight clocks). |
| 9:6 | **tRAS:** This field specifies the minimum RAS active time (1 through 16 clocks). The default for this field is 0100b (five clocks). |
| 5:4 | **tRP:** This field specifies the row pre-charge time (1 through four clocks). The default for this field is 10b (three clocks). |
| 3:2 | **tRCD:** This field specifies the RAS to CAS delay time (1 through four clocks). The default for this field is 10b (three clocks). |
| 1:0 | **tRRD:** This field specifies the row active to row active time (1 through four clocks). The default for this field is 01b (two clocks). |

### 6.3.7 dramInit1

This register sets the sub-clock delays for the SGRAM clocks. This register also contains some addition controls having to do with the SDRAM interface. See Chapter 20.

| Bit | Description |
|---|---|
| 31 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 30 | **Memory Type:** This bit is programmed to 1 for SDRAMs and 0 for SGRAMs. The default for this bit is configured with VMI_HA2. |
| 29 | **NoVinLocking:** When this bit is 0, video in will have priority over video out for memory bandwidth. When this bit is '1', video out will have priority over video in for memory bandwidth. The default for this bit is 0. |
| 28 | **Tristate Outputs:** When this bit is '1', the data outputs (MD[127:0]) are tristated. The default for this bit is 0. |
| 27 | **Force Pagebreak:** When this bit is '1', it forces a pagebreak for all accesses. The default for this bit is 0. |
| 26 | **Disable Aggressive Row Activation:** When this bit is '1', it disables aggressive row activation. The default for this bit is 0. |
| **25** | **Short Power On:** When this bit is '1', 'power on in 128 cycles'. The default for this bit is configured with VMI_HA1. |
| 24 | **sg_oclk_nodelay:** When this bit is '1', it forces the clock out to SGRAMs to have minimum delay. When this bit is 0, the clock out is delayed. The default for this bit is 0. |
| 23:20 | **sg_oclk_del_adj:** This field specifies the delay for clock out to the SGRAMs. The clock is delayed approximately 0.5 ns per unit of delay. This field is effective only when bit 24 is 0. The default for this field is 0000b. |
| 19:16 | **sg_clk_del_adj:** This field specifies the delay for the SGRAM read data sample clock. The delay is 2+(4*n) NAND gates, where n is the value in this field. The default for this field is 0000b. The delay available is 2, 6, 10...62 NAND gates (approximately 0.5 ns per unit of delay). This field is effective only if bit 13 is 0. |
| 15 | **sg_del_clk_invert:** If this bit is '1', the SGRAM read data sample clock is inverted after being delayed (just before being used). The default for this bit is 0. This bit is effective only if bit 13 is 0. |

### 6.3.7 dramInit1 *(cont)*

| Bit | Description |
|---|---|
| 14 | **sg_use_inv_sample:** If this bit is '1', the registered SGRAM data is resampled with another negative-edge flip-flop before being sampled with MCLK. The default for this bit is 0. |
| 13 | **sg_clk_nodelay:** When this bit is '1', the SGRAM read data sample is not delayed. The default for this bit is '1'. Bit 14 is effective regardless of how this bit is programmed. |
| 12 | **Dither Bypass:** When this bit is '1', the dithering logic is bypassed. When this bit is 0, the dithering logic operates normally. |
| 11 | **Triple Buffer Enable:** When this bit is '1', triple buffering is enabled. When this bit is 0, double buffering is used. |
| 10 | **Video Arbitration Priority:** When this bit is 0, the video arbitration uses normal priority. When this bit is '1', video arbitration uses aggressive priority. Default is 0. |
| 9:1 | **RefreshLoad Value:** This value sets the interval between refresh cycles. This is loaded into the high order nine bits of a 14-bit counter clocked with the memory clock. The default value for this field is 0x100. |
| 0 | **SDRAM Refresh Enable:** When this bit is 0, SDRAM refresh is disabled. When this bit is '1', SDRAM refresh is enabled. |

### 6.3.8 agpInit0

This register controls the details of some AGP interface logic.

| Bit | Description |
|---|---|
| 31:27 | **AGP Requests Outstanding:** This read-only field returns the number of active AGP requests. The value 0x1F indicates no outstanding requests. The value 0x00 indicates 17 outstanding requests. |
| 26:10 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 10:7 | **AGP Read FIFO Full Threshold:** This field specifies when too much data has been returned, and AGP needs to begin stalling. |
| 6:4 | **AGP Request FIFO Full Threshold:** This field specifies when the AGP request FIFO becomes full (requests that have not yet been issued to the AGP target). |
| 3:1 | **Maximum AGP Request Length:** This field specifies the maximum number of octwords in an AGP request (1 through 8 octwords). |
| 0 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |

### 6.3.9 tmuGbeInit

This register contains the FIFO water marks for both the tMU and FBI sections. The default value for this register is 0x00000BFB.

| Bit | Description |
|---|---|
| 31:20 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 19:16 | **tvOut Clk Del Adj:** This field specifies the delay imposed on the tvOut Clock. The default for this field is 0x0. Each increment of value inserts one NAND gate. |
| 15 | **tvOut Clk Invert:** When this bit is '1', the tvOut clock is inverted. The default for this bit is 0. |
| 14 | txc force cam miss: The default for this bit is 0. |
| 13 | txc Use Min Request: When this bit is '1', the minimum number of reads done by the texture cache is three. The default for this bit is 0. |
| 12 | txc Disable Rdrq Max: When this bit is '1', it disables the limit of 16 as the maximum limit of maximum number of reads in a row by the texture cache interface to the memory controller. The default for this bit is 0. |
| 11:8 | **Pixel FIFO High Water Mark:** This field specifies the FIFO fullness level at which the FBI will start a sequence. The default for this field is 1010b. |
| 7:4 | **Texture Read Request High Water Mark:** This field specifies the FIFO fullness level at which the TMU will start a sequence. The default for this field is 1111b. |
| 3:0 | **Texture Read Request Low Water Mark:** This field specifies the FIFO fullness level to which the TMU will empty the read request queue before stopping a sequence. The default for this field is 1010b. |

### 6.3.10 vgaInit0

This register is used for hardware initialization and configuration of the VGA controller.

| Bit | Description |
|---|---|
| 31:23 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 22 | **Disable SGRAM Refresh:** When this bit is '1', SGRAM refresh requests will be disabled on HBLANK. The default is 0. |
| 21:14 | **VGA Base:** This seven-bit field specifies the high-order bits of the VGA base offset. Sixteen low-order zeroes are appended to form a 23-bit address. The default for this field is 0. |
| 13 | **Decode 3C6:** If this bit is '1', the VGA core will decode 0x3C6 (the RAMDAC pixel mask register). If this bit is 0, the VGA core will ignore accesses to 0x3C6. The default is 0. This bit is used for factory testing and need not be set for normal operation. |
| 12 | **Extended Video Shift Out:** If this bit is '1', VGA memory access (from the host?) will be disabled. This is used when the video processor is shifting out data. The default is 0. |
| 11 | **Enable Fast Blink:** When this bit is '1', fast blink is enabled. This bit is intended for factory testing only and should always be programmed to 0. The default is 0. |
| 10 | **Alternate VGA Configuration Readback:** When this bit is '1', the alternate VGA configuration readback is selected. The default is 0. |
| 9 | **Disable VGA Legacy Memory:** When this bit is '1', the access to VGA memory at the legacy address (0xA0000-0xBFFFF) is disabled. The default is 0. |
| 8 | **Wakeup Select:** This bit selects the wake-up register. Zero chooses 0x46E8; '1' chooses 0x3C3. The default is 0. When Voodoo3 is a multimedia device, this bit should be set to '1' and the VGA subsystem should be enabled at IOBase+0xC3. |
| 7 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 6 | **Enable VGA Extensions:** When this bit is '1', the CRTC extensions at CR1A and CR1B are enabled. The default is 0. |
| 5:3 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 2 | **VGA FIFO Depth**: This bit specifies how the VGA DAC control logic views the width of the RAM. For VGA compatibility, this bit should be set to 0 (6-bit DAC). |
| 1 | **External Video Timing:** When this bit is '1', the VGA core uses timing provided on the sync pins. The default is 0. |
| 0 | **VGA Disable:** When this bit is '1', all access to the VGA core is disabled. The default is 0. |

### 6.3.11 vgaInit1

This register specifies the read and write apertures for the VMI host interface. It also contains bits that lock (prevent) writes to various sections of the VGA core.

| Bit | Description |
|---|---|
| 31:29 | **Reserved:** Reserved bits must written as 0 for upward compatibility. |
| 28 | **Lock Character Clock:** When this bit is '1', writes to the character clock select bits in SR1[0] are disabled. The default is 0. |
| 27 | **Lock RAM Enable:** When this bit is '1', writes to the RAM Enable bit in 3C2 bit 1 are disabled. The default is 0. |
| 26 | **Lock Clock Select:** When this bit is '1', writes to the character clock select bits in MISC[3:2] are disabled. The default is 0. |
| 25 | **Lock HSYNC:** When this bit is '1', writes to the hsync polarity select bit in MISC[6] are disabled. The default is 0. |
| 24 | **Lock VSYNC:** When this bit is '1', writes to the vsync polarity select bit in MISC[7] are disabled. The default is 0. |
| 23 | **Lock H2:** When this bit is '1', writes to 3B4/3D4 index 17, bit 2 are disabled. The default is 0. |
| 22 | **Lock Vertical Timing:** When this bit is '1', writes to the CRTC registers that specify vertical timing (3B4/3D4 index 6, 7[ 7, 5, 3, 2, 0], 9, 10, 11[3:0], 15, 16, 1B) are disabled. The default is 0. |
| 21 | **Lock Horizontal Timing:** When this bit is '1', writes to the CRTC registers that specify horizontal timing (3B4/3D4 index 0, 1, 2, 3, 4, 5, 1A) are disabled. The default is 0. |
| 20 | **Enable 0xA0000 Sequential Chain 4:** When this bit is '1', sequential chain mode is enabled. This is a pseudo packed-pixel format. |
| 19:10 | **VBE Read Aperture:** This 10-bit field specifies the high order address bits of the VMI host bus read aperture. Fifteen low-order zeroes are appended to form a 25-bit address. |
| 9:0 | **VBE Write Aperture:** This 10-bit field specifies the high order address bits of the VMI host bus read aperture. Fifteen low-order zeroes are appended to form a 25-bit address. |

### 6.3.12 2D Command Register

A write to offset 0x30 will access the 2D unit's command register (Section 7.2.21). This mapping is intended to provide a way to initialize the SGRAM mode and special mode registers at initialization time.

### 6.3.13 2D srcBaseAddr Register

A write to offset 0x34 will access the 2D unit's scrBaseAddr register (Section 7.2.10). This mapping is intended to provide a way to initialize the SGRAM mode and special mode registers at initialization time.

### 6.3.14 dramData

This register contains the data used during dramSMode writes.

| Bit | Description |
|---|---|
| 31:0 | **Data:** SGRAM data used during dramSMode register writes. Write the data to this register and then specify the color or mask register with a write to dramSMode. |

## 6.4 PLL and DAC Registers

These registers control the frequency synthesizers and the DAC.

### 6.4.1 pllCtrl[1:0]

Each of these two registers control one of two frequency synthesizers. See Chapter 14 for programming information.

**Table 6.2 pllCtrl Registers**

| Offset | Name | Synthesizer |
|---|---|---|
| 0x40 | pllCtrl0 | Video Clock |
| 0x44 | pllCtrl1 | GRX, Memory Clock |

| Bit | Description |
|---|---|
| 31:17 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 16 | **Test:** When this bit is '1', the respective PLL is put into test mode. This bit is for factory testing only and should always be programmed to 0 by any application. This bit is documented here for the sake of completeness. |
| 15:8 | **N:** This eight-bit field is the PLL multiplier. The output frequency of the respective synthesizer can be calculated with the following equation. *ref* is typically 14.31818 MHz. $$fout = \frac{ref \bullet (N+2)}{(M+2) \bullet (2^K)}$$ |
| 7:2 | **M:** This six-bit field is the PLL input divider. If the DeviceID in PCI0[31:16] is 0x0004, this field is forced to the value 0x24 for pllCtrl1 only. This limits the maximum graphics/memory clock to 141 MHz. When the initialization value for pllCtrl1 is selected, this must be taken into account. If the DeviceID is 0x0005, this field is fully programmable (but see the restrictions in Chapter 14). |
| 1:0 | **K:** This two-bit field is the PLL controls the post-scaler value. It sets the size of a divider chain; the post-scaler ratio is $2^K$. |

### 6.4.2 pllCtrl2

This register controls the test modes of the output of the AGP 2X phased-locked loop.

| Bit | Description |
|---|---|
| 31:2 | **Reserved:** Reserved bits must be programmed to 0 for upward compatibility. |
| 1:0 | **pll2 Controls:** This field is used in conjunction with the strapping on VMI_HD2 (66 MHz) and the strapping on VMI_ADDR3 (PLL_BYPASS), according to the table. |

| VMI_HD2 PCI_66Mhz | VMI_ADDR3 PLL_BYPASS | pllCtrl2[1:0] | Clk66 | Clk133 | Note |
|---|---|---|---|---|---|
| 0 | 0 | (don't care) | PCI_CLK | PCI_CLK | PCI |
| 0 | 1 | (don't care) | PCI_CLK | PCI_CLK | PCI |
| 1 | 0 | 0x00 | 1/4 pll output | 1/2 pll output | AGP, PLL on |
| 1 | 0 | 0x01 | 1/4 PCI_CLK | 1/2 PCI_CLK | Test only |
| 1 | 0 | 0x10 | PCI_CLK | 1/2 pll output | Test only |
| 1 | 0 | 0x11 | PCI_CLK | 1/2 PCI_CLK | AGP, bypass 1X only |
| 1 | 1 | (don't care) | 1/4 PCI_CLK | 1/2 PCI_CLK | Test only |

### 6.4.3 dacMode

This register contains some mode bits that control the DAC.

| Bit | Description |
|---|---|
| 31:5 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 4 | **Force HSYNC Value:** This sets the static level on HSYNC when bit 3 is a'1'. |
| 3 | **Enable DPMS on HSYNC:** When this bit is '1', DPMS (Display Power Management Signaling) is enabled on HSYNC. HSYNC will be static, at the level indicated by bit 4. Is any of this right? |
| 2 | **Force VSYNC Value:** This sets the static level on VSYNC when bit 1 is a'1'. |
| 1 | **Enable DPMS on VSYNC:** When this bit is '1', DPMS (Display Power Management Signaling) is enabled on VSYNC. VSYNC will be static, at the level indicated by bit 2. |
| 0 | **DAC Mode:** When this bit is '1', the DAC is in 2:1 mode. When this bit is 0, the DAC is in 1:1 mode. |

### 6.4.4 dacAddr

This register contains the palette address to be accessed.

| Bit | Description |
|---|---|
| 31:9 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 8:0 | **Palette Address:** This nine-bit value specifies the address of the entry in the palette to be accessed for read or write. This value does not automatically increment as does the palette address for VGA. |

### 6.4.5 dacData

Accesses to this register transfer color information to or from the palette. The dacAddr register is incremented each time this register is accessed.

| Bit | Description |
|---|---|
| 31:24 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 23:16 | **Red:** This field specifies the red component of the color. |
| 15:8 | **Green:** This field specifies the green component of the color. |
| 7:0 | **Blue:** This field specifies the blue component of the color. |

## 6.5 Video Registers Part I

### 6.5.1 vidTvOutBlankVCount

This register specifies the timing of vertical blank when TV out Genlock is enabled.

| Bit | Description |
|---|---|
| 31:27 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 26:16 | **Vertical Blank Begins:** This field specifies the number of tv_hsync leading edges after tv_vsync leading edge before vertical blank begins (vertical active region ends). This value must be greater than the value programmed into bits [10:0] (so the vertical active period is positive). |
| 15:11 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 10:0 | **Vertical Blank Ends:** This field specifies the number of tv_hsync leading edges after the tv_vsync leading edge before vertical blank ends (vertical active region begins). |

### 6.5.2 rgbMaxDelta

This register specifies the maximum deltas for video filtering. These fields specify the threshold values allowed for the deviation of a pixel's luminance and chroma components when the pixel is used in the video filter (either the 4 x 1 tap filter or the 2 x 2 box filter.
If the absolute deviation of a pixel component exceeds its threshold, the particular component wil be replaced with that corresponding component of the center pixel before the pixel is used in the filter.

| Bit | Description |
|---|---|
| 31:24 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 23:22 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 21:16 | **Maximum Red Delta:** This six-bit value specifies the maximum red delta for filtering. |
| 15:14 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 13:8 | **Maximum Green Delta:** This six-bit value specifies the maximum green delta for filtering. |
| 7:6 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 5:0 | **Maximum Blue Delta:** This six-bit value specifies the maximum blue delta for filtering. |

### 6.5.3 vidProcCfg

This is the configuration register for the Video Processor. This register is generally written just once during system initialization.

| Bit | Description |
|---|---|
| 31 | **Enable Backend Deinterlacing:** When this bit is '1', 'bob' deinterlacing is employed in the backend pipeline. This method displays either the even or odd frame and interpolates two interlaced lines to obtain the missing field. This is not supported in 2X clocking mode. |
| 30 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. This bit was used in previous product for stride/offset control. |
| 29 | **Disable Memory Optimization for Overlay:** When this bit is '1', optimization for overlay requests is disabled. When this bit is 0, optimization for overlay requests is enabled. The default for this bit is 0. This bit is altogether redefined from its meaning in previous product. |
| 28 | **Disable Memory Optimization for Desktop:** When this bit is '1', optimization for desktop requests is disabled. When this bit is 0, optimization for desktop requests is enabled. The default for this bit is 0. |
| 27 | **HW Cursor Enable:** When this bit is '1', the hardware cursor is enabled. See Chapter 13 registers for controls. |
| 26 | **2X Mode:** When this bit is '1', 2X mode is enabled. This writes two screen pixels for every video clock. When this bit is 0, 1X mode is enabled, which writes one screen pixel for every video clock. |
| 25 | **Overlay Tile Space Enable:** When this bit is '1', the overlay is addressed using tile space. When this bit is 0, the overlay is addressed using linear space. |
| 24 | **Desktop Tile Space Enable:** When this bit is '1', the desktop is addressed using tile space. When this bit is 0, the desktop is addressed using linear space. |
| 23:21 | **Overlay Pixel Format:** This field specifies the pixel format of overlay data according to the table. Values not in the table must not be programmed into this field. |

| Value | Format | Note |
|---|---|---|
| 001b | RGB 5:6:5 undithered | |
| 100b | YUV 4:1:1 | |
| 101b | YUV 4:2:2 | |
| 110b | UYUV 4:2:2 | |
| 111b | RGB 5:6:5 dithered | |

### 6.5.3 vidProcCfg *(cont)*

**Bit** **Description**

20:18 **Desktop Pixel Format:** This field specifies the pixel format of desktop data according to the table. Values not in the table must not be programmed into this field.

| Value | Format | Note |
|---|---|---|
| 000b | 8-bit palettized | |
| 001b | RGB 5:6:5 undithered | |
| 010b | RGB24 packed | |
| 011b | RGB32 | |

17:16 **Overlay Filter Mode:** This field specifies the overlay filter mode, according to the table.

| Value | Mode | Note |
|---|---|---|
| 00b | Point sampling | (no filtering) |
| 01b | 2x2 dither subtract followed by 2x2 box filter | 3D only |
| 10b | 4x4 dither subtract followed by 4x1 tap filter | 3D only |
| 11b | Bilinear scaling | |

15 **Overlay Vertical Scaling Enable:** When this bit is ‘1’, vertical scaling for overlay is enabled. The scale factor is determined by vidOverlayDvdy ().

14 **Overlay Horizontal Scaling Enable:** When this bit is ‘1’, horizontal scaling for overlay is enabled. The scale factor is determined by vidOverlayDudx ().

13 **Overlay CLUT Select:** When this bit is ‘1’, the overlay uses the upper 256 entries of the CLUT. When this bit is 0, the overlay uses the lower 256 entries of the CLUT.

12 **Desktop CLUT Select:** When this bit is ‘1’, the desktop uses the upper 256 entries of the CLUT. When this bit is 0, the desktop uses the lower 256 entries of the CLUT.

11 **Overlay CLUT Bypass:** When this bit is ‘1’, the overlay bypasses the CLUT. When this bit is 0, the overlay uses the CLUT.

10 **Desktop CLUT Bypass:** When this bit is ‘1’, the desktop bypasses the CLUT. When this bit is 0, the desktop uses the CLUT.

9 **Enable Display Video-In as Overlay:** When this bit is ‘1’, the video-in buffer address is used as the overlay start address, directly displaying the video in. When this bit is 0, the video-in is not directly displayed as overlay (the buffers are different).

8 **Overlay Surface Enable:** When this bit is ‘1’, the overlay surface is fetched and displayed.

### 6.5.3 vidProcCfg *(cont)*

| Bit | Description |
|---|---|
| 7 | **Desktop Surface Enable:** When this bit is '1', the desktop surface is fetched and displayed. |
| 6 | **ChromaKeyResultInversion:** When this bit is '1', the desktop is transparent (the overlay is displayed) if the desktop color does not match or fall within the chroma-key range. When this bit is 0, the desktop is transparent if the desktop color does match or fall within the chroma-key range. |
| 5 | **ChromaKeyEnable:** When this bit is '1', chroma-key matching is enabled. |
| 4 | **FetchDesktopStartAddress:** When this bit is '1', a new desktop start address is fetched every VSYNC for non-stereo, interlaced video output. When this bit is 0, a new desktop start address is fetched for every other VSYNC for non-stereo, interlaced video output. |
| 3 | **Interlaced Video Output Mode:** Voodoo3 does not support interlaced video out; this bit must be programmed to 0. |
| 2 | **Overlay Stereo Enable:** When this bit is '1', stereo video is enabled. But we dont have a stereo pin on Voodoo3. |
| 1 | **Cursor Mode:** When this bit is '1', the cursor color select bits are interpreted according to X11 rules. When this bit is '0', the cursor color select bits are interpreted according to Microsoft Windows rules. |
| 0 | **Video/VGA:** When this bit is '1', the video processor is on and VGA mode is off. When this bit is 0, VGA mode is on and the video processor is off. |

### 6.5.4 hwCurPatAddr

This register stores the starting address of the two cursor patterns.

| Bit | Description |
|---|---|
| 31:24 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 23:0 | **Cursor Address:** This is the physical address of the beginning of the cursor patterns in the frame buffer. See Chapter 13 for details of the hardware cursor. |

### 6.5.5 hwCurLoc

This register contains the x-y screen coordinates the lower-right corner of the hardware cursor.

| Bit | Description |
|---|---|
| 31:27 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 26:16 | **HW Cursor Y:** This 11-bit value specifies the Y screen coordinate of the lower right corner of the hardware cursor. An 11-bit field allows the cursor to be placed on any scanline up to 2047. |
| 15:11 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 10:0 | **HW Cursor X:** This 11-bit value specifies the X screen coordinate of the lower right corner of the hardware cursor. An 11-bit field allows the cursor to be placed on any pixel up to 2047. |

### 6.5.6 hwCurC[1:0]

These two registers contain the two cursor colors. Each is a 24-bit true color. hwCurC0 is at 0x68; hwCurC1 is at 0x6C. See Table 13.1 for when these colors are displayed.

| Bit | Description |
|---|---|
| 31:24 | **Reserved:** Reserved fields must be written as 0 for upward compatibility. |
| 23:16 | **Red:** This field specifies the red value of the respective cursor color. |
| 15:8 | **Green:** This field specifies the green value of the respective cursor color. |
| 7:0 | **Blue:** This field specifies the blue value of the respective cursor color. |

### 6.5.7 vidInFormat

This register specifies the format of the video-in. There are also other VMI controls in this register.

| Bit | Description |
|---|---|
| 31:22 | **Reserved:** Reserved fields must be written as 0 for upward compatibility. |
| 21 | **Video-in Vertical Decimation Enable:** When this bit is ‘1’, vertical decimation of video-in is enabled. |
| 20 | **Video-in Horizontal Decimation Enable:** When this bit is ‘1’, horizontal decimation of video-in is enabled. |
| 19 | **TV out Select Display Enable:** When this bit is ‘1’, it selects display_ena instead of vga_blank for driving output video. |
| 18 | **Genlock Source Select:** When this bit is ‘1’, the TV encoder in master mode is selected as the genlock source. When this bit is 0, the VMI inputs (VMI_PCLK, VMI_HSYNC, and VMI_VSYNC) are selected as the genlock source. The default is 0. |

### 6.5.7 vidInFormat *(cont)*

| Bit | Description |
|---|---|
| 17 | **not_use_vga_timing:** When this bit is 0, the timing signals provided by the VGA core are used. This is intended for VMI and TV out slave modes. When this bit is '1', the timing are supplied either by the genlock source (as specified in bit 18) or generated internally by the video controller. If the genlock source is VMI, then VMI_VSYNC and VMI_HSYNC are input from the VMI device (VMI_VACTIVE is always an input). If the genlock source is a TV encoder, TV_HSYNC and TV_VSYNC are inputs from the TV encoder (TV_BLANK is always an output to the TV encoder). |
| 16 | **Genlock Enable:** When this bit is 0, the remaining video logic uses a separate video clock from the on-chip PLL. This case is used for VMI and TV encoder slave mode.<br>When this bit is '1', the remaining video logic uses the genlock source (bit 18) to as its clock. If the genlock source is VMI, Voodoo3 is genlocked to VMI_PCLK. If the genlock source is TV encoder, Voodoo3 is genlocked to Tv_INCLK. This is intended for TV encoder master mode. |
| 15 | **TV out Interface Enable:** When this bit is '1', it enables the TV out interface. |
| 14 | **VMI Interface Enable:** When this bit is '1', it enables the VMI interface. |
| 13 | **TV_HSYNC Polarity:** When this bit is '1', TV_HSYNC is active low (normally high signal going low to indicate the beginning of sync). When this bit is 0, TV_HSYNC is active high (normally low signal going high to indicate the beginning of sync. |
| 12 | **TV_VSYNC Polarity:** When this bit is '1', TV_VSYNC is active low (normally high signal going low to indicate the beginning of sync). When this bit is 0, TV_VSYNC is active high (normally low signal going high to indicate the beginning of sync. |
| 11 | **VideoIn Tile Space Enable:** When this bit is '1', the video-in space is addressed using tile space. When this bit is 0, the video-in space is addressed using linear space. This applies to VMI mode only. |
| 10:9 | **Video-in Buffering Mode Select:** This field specifies the depth of buffering used for video-in, according to the table. Values not in this table must not be programmed into this field. |
| 8 | **TV-out Format:** This bit is programmed to '1' for Brooktree encoder support. This bit is programmed to 0 for Chrontel encoder support. See Table 15.1. |
| 7 | **VMI V_ACTIVE Polarity:** This bit is programmed to '1' for active low V_ACTIVE. This bit is programmed to 0 for active high V_ACTIVE. |

| Value | Buffering | Note |
|---|---|---|
| 00b | Single | |
| 01b | Double | |
| 10b | Triple | |

### 6.5.7 vidInFormat *(cont)*

| Bit | Description |
|---|---|
| 6 | **VMI_HSYNC Polarity:** This bit is programmed to '1' for active low VMI_HSYNC. This bit is programmed to 0 for active high VMI_HSYNC. The default is 0. |
| 5 | **VMI_VSYNC Polarity:** This bit is programmed to '1' for active low VMI_VSYNC. This bit is programmed to 0 for active high VMI_VSYNC. The default is 0. |
| 4 | **Video-In Deinterlacing:** When this bit is '1', weave deinterlacing is done on incoming video data. The video port will merge two consecutive VMI frames into a single frame before signaling frame completion. When this bit is 0, no deinterlacing is done on incoming video data. |
| 3:1 | **Video-in Data Format:** This field specifies video-in data format according to the table. Values not in this table must be programmed into this field. |
| 0 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |

| Value | Format |
|---|---|
| 100b | 8 bit YCbCr 4:1:1 |
| 101b | 8-bit YCbCr 4:2:2 (YUYV) |
| 110b | 8-bit YCbCr 4:2:2 (UYVY) |

### 6.5.8 vidTvOutBlankHCount

This register specifies the timing of horizontal blank when TV out Genlock is enabled.

| Bit | Description |
|---|---|
| 31:27 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 26:16 | **Horizontal Blank Begins:** This field specifies the number of clock cycles after tv_hsync leading edge before horizontal blank begins (horizontal active region ends). This value must be greater than the value programmed into bits [10:0] (so the horizontal active period is positive). |
| 15:11 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 10:0 | **Horizontal Blank Ends:** This field specifies the number of clock cycles after the tv_hsync leading edge before horizontal blank ends (horizontal active region begins). |

### 6.5.9 vidSerialParallelPort

This register controls the I2C, DDC, GPIO, and VMI host interfaces. Since the ROM interface shares pins with the VMI, a given pin can be input or output depending on which interface the chip is configured for at the moment.

| Bit | Description |
|---|---|
| 31 | **TV out Reset:** This bit controls the TV_RESET pin (0 forces a reset). The default value is 0. |
| 30 | **GPIO2 In:** This read-only bit returns the level on GPIO2. GPIO2 is always an input. |
| 29 | **GPIO1 Out:** This bit appears on GPIO1. GPIO1 is always an output. |
| 28 | **VMI Reset_n:** When this bit is 0, the VMI interface is forced to an initial state. The default for this bit is 0. When this bit is 1, the VMI interface functions normally. |
| 27 | **SDA1 In:** This read-only bit returns the level on SDA1 (assigned to the feature connector interface). |
| 26 | **SDC1 In:** This read-only bit returns the level on SDC1 (assigned to the feature connector interface). |
| 25 | **SDA1 Out:** When this bit is 0, SDA1 (assigned to the feature connector interface) is pulled to ground. The default for this bit is '1'. |
| 24 | **SDC1 Out:** When this bit is 0, SDC1 (assigned to the feature connector interface) is pulled to ground. The default for this bit is '1'. |
| 23 | **SDA/SDC1 Enable:** When this bit is 0, the SDA1/SDC1 I2C port is disabled. I guess that means the Voodoo3 cannot pull either pinA low. |
| 22 | **SDA0 In:** This read-only bit returns the level on SD0 (assigned to the monitor interface). |
| 21 | **SDC0 In:** This read-only bit returns the level on SDC0 (assigned to the monitor interface). |
| 20 | **SDA0 Out:** When this bit is 0, SDA0 (assigned to the monitor interface) is pulled to ground. |
| 19 | **SDC0 Out:** When this bit is 0, SDC0 (assigned to the monitor interface) is pulled to ground. |
| 18 | **SDA/SDC0 Enable:** When this bit is 0, the SDA0/SDC0 I2C port is disabled. |
| 17:14 | **VMI Address:** The contents of this field will be driven onto VMI_HA[3:0] when a host interface transaction is executed. |
| 13:6 | **VMI Data:** This field connects to the VMI_HD[7:0] bus for host interface transactions. For a read, the host reads the VMI data from this field; for a write, the host places the data to be transferred into this field. |

### 6.5.9 vidSerialParallelPort *(cont)*

| Bit | Description |
|---|---|
| 5 | **VMI Data Output Enable:** When this bit is 0, the output buffers on VMI_HD[7:0] are enabled. The host should program this bit to 0 during a write transaction on the VMI host interface. The default for this bit is '1'. |
| 4 | **VMI_RDY_N:** This read-only bit returns the level on the VMI_RDY_N (VMI_READY for mode A). |
| 3 | **VMI_RW_N:** This bit drives the VMI_RW_N pin (VMI_WR_N for mode B). |
| 2 | **VMI_DS_N:** This bit drives the VMI_DS_N pin (VMI_RD_N for mode B). |
| 1 | **VMI_CS_N:** this bit drives the VMI_CS_N pin. |
| 0 | **VMI Host Port Enable:** When this bit is 0, the VMI host port control pins are disabled. The default for this bit is 0. |

### 6.5.10 vidInXDecimDeltas

The meaning of this register depends on the Video-in Interface Configuration. When the interface is configured for VMI mode, this register controls horizontal downscaling.

| Bit | Description |
|---|---|
| 31:28 | **Reserved:** Reserved bits must be programmed to 0 for upward compatibility. |
| 27:16 | **Delta Width:** This field is the positive (unsigned) value to be added when the horizontal Bresenham error is greater than 0. This field must be programmed to the difference between the width of the source (VMI input) surface and the destination surface. This field is programmed in terms of pixels. |
| 15:12 | **Reserved:** Reserved bits must be programmed to 0 for upward compatibility. |
| 11:0 | **Destination Width:** This field is positive (unsigned) value to be added when the horizontal Bresenham error is less than 0. This field must be programmed to the width of the destination surface in pixels. |

### 6.5.11 vidInDecimInitErrs

This register contains the initial values for the error terms for video in decimation

| Bit | Description |
|---|---|
| 31:29 | **Reserved:** Reserved bits must be programmed to 0 for upward compatibility. |
| 28:16 | **Inital Vertical Error:** This field specifies the inital value of the error term in the vertical decimation accumulator. This is a signed (2's complement) number. |
| 15:13 | **Reserved:** Reserved bits must be programmed to 0 for upward compatibility. |
| 28:16 | **Inital Horizontal Error:** This field specifies the inital value of the error term in the horizontal decimation accurmulator. This is a signed (2's complement) number. |

### 6.5.12 vidInYDecimDeltas

This register controls horizontal downscaling for video in.

| Bit | Description |
|---|---|
| 31:28 | **Reserved:** Reserved bits must be programmed to 0 for upward compatibility. |
| 27:16 | **Delta Height:** This field is the positive (unsigned) value to be added when the vertical Bresenham error is greater than 0. This field must be programmed to the difference between the height of the source (VMI input) surface and the destination surface. This field is programmed in terms of scan lines. |
| 15:12 | **Reserved:** Reserved bits must be programmed to 0 for upward compatibility. |
| 11:0 | **Destination Height:** This field is positive (unsigned) value to be added when the vertical Bresenham error is less than 0. This field must be programmed to the height of the destination surface in scan lines. |

### 6.5.13 vidPixelBuffThold

This register specifies the number of empty slots in each of the three pixel buffers that will trigger refilling of the respective buffer.

| Bit | Description |
|---|---|
| 31:18 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 17:12 | **Secondary Buffer 1 LowWaterMark:** This field specifies the number of empty slots that will trigger refilling of the respective buffer. The value 0 specifies 1 empty slot; the value 0x3F specifies 63 empty slots. |
| 11:6 | **Secondary Buffer 0 LowWaterMark:** This field specifies the number of empty slots that will trigger refilling of the respective buffer. The value 0 specifies 1 empty slot; the value 0x3F specifies 63 empty slots. |
| 5:0 | **Primary Buffer LowWaterMark:** This field specifies the number of empty slots that will trigger refilling of the respective buffer. The value 0 specifies 1 empty slot; the value 0x3F specifies 63 empty slots. |

### 6.5.14 vidChromaKeyMin/Max

These two registers specify the lower and upper bounds of the chroma key color. The format of the two registers is identical and is shown in the bit description. If the two registers are programmed to the same value, the chroma-key is a single color rather than a range.

**Table 6.3 vidChromaKey Registers**

| Register | Offset | Note |
|---|---|---|
| vidChromaKeyMin | 0x8C | |
| vidChromaKeyMax | 0x90 | |

**Bit** **Description**

31:0 **ChromaKey:** The format depends on the pixel depth, according to the table.

| Pixel Depth | Unused | Index | Red | Green | Blue |
|---|---|---|---|---|---|
| 8 Bits | 31:8 | 7:0 | - | - | - |
| 15 Bits | 31:15 | - | 14:10 | 9:5 | 4:0 |
| 16 Bits | 31:16 | - | 15:11 | 10:5 | 4:0 |
| 24 Bits | 31:24 | - | 23:16 | 15:8 | 7:0 |
| 32 Bits | 31:24 | - | 23:16 | 15:8 | 7:0 |

### 6.5.15 vidStatusCurrentLine

This read-only register returns the current scan line.

| Bit | Description |
|---|---|
| 31:19 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 18 | **Oddity:** This bit indicates whether the frame just captured was an even frame (‘1’) or odd (0) frame. |
| 17:16 | **VideoIn Buffer:** This field specifies the last video-in buffer that the VMI has written into, according to the table. |

| Value | Description |
|---|---|
| 00b | Buffer0 (vidInAddr0) |
| 01b | Buffer1 (vinInAddr1) |
| 10b | Buffer2 (vidInAddr2) |
| 11b | None: First frame is not yet captured. |

| | |
|---|---|
| 15:11 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 10:0 | **Current Video Scan Line:** This read-only field returns the current scan line. |

### 6.5.16 vidScreenSize

This register specifies the size of the video screen. Whenever the screen resolution is changed, the video processor needs to be reset. This is done by writing 0 and then 1 to vidProcCfg[0].

| Bit | Description |
|---|---|
| 31:24 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 23:12 | **vidScreenY:** This field specifies the video screen height in scan lines. |
| 11:0 | **vidScreenX:** This field specifies the video screen width in pixels. If this field is programmed to a value larger than 1280, 2x mode must be enabled. |

### 6.5.17 vidOverlayStartCoord

This register specifies the screen location of the upper left corner of the video overlay.

| Bit | Description |
|---|---|
| 31:28 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 27:26 | **Y-Coord[1:0]:** This field specifies the low order two bits of the Y coordinate of the first pixel of the overlay window with respect to the beginning of the source surface. Since the overlay window may be partially occluded by the dimension of the screen, the first pixel of the window may not necessarily be the first pixel of the source surface. These two bits are used for undithering, and are the same for both linear and tiled address space. |
| 25:24 | **X-Coord[1:0]:** This field specifies the low order two bits of the X coordinate of the first pixel of the overlay window with respect to the beginning of the source surface. Since the overlay window may be partially occluded by the dimension of the screen, the first pixel of the window may not necessarily be the first pixel of the source surface. These two bits are used for undithering, and are the same for both linear and tiled address space. |
| 23:12 | **Y-Coord:** This field specifies the Y coordinate on the screen of the upper left corner of the video overlay. |
| 11:0 | **X-Coord:** This field specifies the X coordinate on the screen of the upper left corner of the video overlay. |

### 6.5.18 vidOverlayEndScreen

This register specifies the screen location of the lower right corner of the video overlay. Since the overlay coordinates are zero-based, each field is programmed to the respective screen size - 1.

| Bit | Description |
|---|---|
| 31:24 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 23:12 | **Y-Coord:** This field specifies the Y coordinate on the screen of the lower right corner of the video overlay. |
| 11:0 | **X-Coord:** This field specifies the X coordinate on the screen of the lower right corner of the video overlay. |

### 6.5.19 vidOverlayDudx

This register specifies the horizontal magnification for video display.

| Bit | Description |
|---|---|
| 31:20 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 19:0 | **Horizontal Step Size:** This field specifies the step size in source per horizontal step in screen space. This is 0.20 fraction (there is a radix point to the left of bit 19). A smaller number in this field yields more horizontal magnification. |

### 6.5.20 vidOverlayDudxOffsetScrWidth

This register specifies the number of bytes of pixels that must be fetched from the frame buffer for each scan line of overlay window.

| Bit | Description |
|---|---|
| 31:19 | **Source Width:** This field specifies the number of byte that must be fetched from the source surface to cover an entire un-occluded scan line for the overlay. This 13-bit field allows up to 8095 bytes to be specified. The following equation may be used to calculate this value.<br>$SourceWidth = PixelDepth \bullet (vidOverlayDudxOffset + (Width \bullet vidOverlayDudx))$<br>For non-scaled overlay with no offset, vidOverlayDudx is 1 and vidOverlayDudxOffset is 0, and the equation simplifies to the following.<br>$SourceWidth = PixelDepth \bullet Width$ |
| 18:0 | **Initial Offset of Dudx:** This field specifies the initial offset of Dudx. This is a 0.19 format fraction. |

### 6.5.21 vidOverlayDvdy

This register specifies the vertical magnification for video display.

| Bit | Description |
|---|---|
| 31:20 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 19:0 | **Vertical Step Size:** This field specifies the step size in source per vertical step in screen space. This is 0.20 fraction. A smaller number in this field yields more vertical magnification. |

## 6.6 VGA Registers

The standard VGA registers are accessible at I/O offsets 0xB0 through 0xDF. They are not accessible in the memory space claimed in PCI10. See Chapter 3 for the descriptions of the VGA core registers. This little note is included for the sake of completeness.

## 6.7 Video Registers Continued

### 6.7.1 vidOverlayDvdyOffset

This register contains the initial offset of Dvdy. This is a 19-bit fraction.

| Bit | Description |
|---|---|
| 31:19 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 18:0 | **Initial Offset of Dvdy:** This is a 19-bit fraction. |

### 6.7.2 vidDesktopStartAddr

This register specifies the physical starting address of the desktop surface.

| Bit | Description |
|---|---|
| 31:24 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 23:0 | **Desktop Starting Address:** This field specifies the physical starting address of the desktop surface in the frame buffer. This a byte-aligned address anywhere in 16 Mbytes. |

### 6.7.3 vidDesktopOverlayStride

This register contains the strides (or offset or pitch, depending on your school) of both the desktop and the overlay surfaces.

| Bit | Description |
|---|---|
| 31 | **Reserved:** Reserved bits must always be written as 0 for upward compatibility. |
| 30:16 | **Overlay Surface Stride:** This field specifies the stride of the overlay surface. If the overlay surface is in linear space, this field is the linear stride in bytes. If interlaced video output mode is enabled, the linear stride is nevertheless programmed to the regular stride of the surface (that is, one scanline) and will be multiplied by two when used.<br>If the overlay is in tile space, this field contains the tile stride of the region. This is specified in the number of tiles across the width of the tile address region, not the width of the overlay surface.<br>For video overlays, the stride must be a multiple of four bytes for YUV 4:2:2 and a multiple of eight bytes for YUV 4:1:1 pixel formats. This guarantees that the right edge of the video source surface will fall on a boundary of two pixels for YUV 4:2:2 and four pixels for YUV 4:1:1.<br>The start address of the overlay region is taken from the FIFO'd leftOverlayBuf and rightOverlayBuf registers. The start address must be aligned on a 32-bit boundary for YUV 4:2:2 and a 64-bit boundary for YUV 4:1:1 pixel formats. The following table summarizes these restrictions. |
| 15 | **Reserved:** Reserved bits must always be written as 0 for upward compatibility. |
| 14:0 | **Desktop Surface Stride:** This field specifies the stride of the desktop surface. If the desktop surface is in linear space, this field is the linear stride in bytes. If interlaced video output mode is enabled, the linear stride is nevertheless programmed to the regular stride of the surface (that is, one scanline) and will be multiplied by two when used.<br>If the desktop is in tile space, this field contains the tile stride of the region. This is specified in the number of tiles across the width of the tile address region, not the width of the desktop surface. |

| Pixel Format | YUV 4:2:2 | YUV 4:1:1 |
|---|---|---|
| Stride Multiple | Four bytes | Eight bytes |
| Right Edge Boundary | Two pixels | Four pixels |
| Start Address Boundary | 32-bit | 64-bit |

### 6.7.4 vidInAddr0/1/2

These three registers contains the addresses of the three video-in buffers.

**Table 6.4 vidInAddr Registers**

| Register | Offser | Note |
|---|---|---|
| vidInAddr0 | 0xEC | |
| vidInAddr1 | 0xF0 | |
| vidInAddr2 | 0xF4 | |

| Bit | Description |
|---|---|
| 31:24 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 23:0 | **vidInAddr:** This field specifies the starting address of the respective video-in buffer. The table indicates which buffers are used, according to the video-in Buffering Mode Select in vidInFormat[10:9]. |

| vidInFormat[10:9] | vidInAddr0 0xEC | vidInAddr1 0xF0 | vidInAddr2 0xF4 |
|---|---|---|---|
| 00b | X | | |
| 01b | X | X | |
| 10b | X | X | X |
| 11B | Reserved, do not use | | |

### 6.7.5 vidInStride

This register contains the stride of the video-in buffer(s).

| Bit | Description |
|---|---|
| 31:15 | **Reserved:** Reserved bits must always be written as 0 for upward compatibility. |
| 14:0 | **Video-In Buffers Stride:** This field specifies the stride of the video-in buffers. If the buffers are in linear space, this field is the linear stride in bytes. If interlaced video input mode is enabled, the linear stride is nevertheless programmed to the regular stride of the surface (that is, one scanline) and will be multiplied by two when used.<br>If the video buffers are in tile space, this field contains the tile stride of the region. This is specified in the number of tiles across the width of the tile address region, not the width of the video-in buffers. |

### 6.7.6 vidCurrOverlayStartAddr

This read-only register returns the start address which the video processor is using the refresh the overlay window for the current frame.

| Bit | Description |
|---|---|
| 31:24 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 23:0 | **Current Overlay Start Address:** This read-only field returns the starting physical address the video processor is using to refresh the overlay window. |

# 7 2D Registers

## 7.1 Overview

The following registers are accessible beginning at offset 0x010 0000 in the space claimed in memBaseAddr0 (PCI10).
Table 7.1 contains a summary of the I/O registers. The registers are ordered by I/O offset. The link is a clickable link to the detailed description. Unless otherwise noted, all non-reserved fields in all registers are read/write. Reading a 2D register will always return the value that will be used if a new operation is begun without writing a new value to that register. This value will either be the last value written to the register, or if an operation has been performed since the value written, the value after all operations have completed.

**Table 7.1 2D Register Summary**

| Register Name | Offset | Link |
|---|---|---|
| status register | 0x010 0000 | Section 6.2.1 |
| intCtrl | 0x010 0004 | Section 9.3.2 |
| clip0Min | 0x010 0008 | Section 7.2.3 |
| clip0Max | 0x010 000C | Section 7.2.3 |
| dstBaseAddr | 0x010 0010 | Section 7.2.4 |
| dstFormat | 0x010 0014 | Section 7.2.5 |
| scrColorKeyMin | 0x010 0018 | Section 7.2.6 |
| scrColorKeyMax | 0x010 001C | Section 7.2.6 |
| dstColorKeyMin | 0x010 0020 | Section 7.2.7 |
| dstColorKeyMax | 0x010 0024 | Section 7.2.7 |
| bresError0 | 0x010 0028 | Section 7.2.8 |
| bresError1 | 0x010 002C | Section 7.2.8 |
| rop | 0x010 0030 | Section 7.2.9 |
| scrBaseAddr | 0x010 0034 | Section 7.2.10 |
| commandExtra | 0x010 0038 | Section 7.2.11 |
| lineStipple | 0x010 003C | Section 7.2.12 |
| lineStyle | 0x010 0040 | Section 7.2.13 |
| pattern0Alias | 0x010 0044 | Section 7.2.14 |
| pattern1Alias | 0x010 0048 | Section 7.2.14 |
| clip1Min | 0x010 004C | Section 7.2.3 |
| clip1Max | 0x010 0050 | Section 7.2.3 |

**Table 7.1 2D Register Summary *(cont.)***

| Register Name | Offset | Link |
|---|---|---|
| srcFormat | 0x010 0054 | Section 7.2.15 |
| srcSize | 0x010 0058 | Section 7.2.16 |
| srcXY | 0x010 005C | Section 7.2.17 |
| colorBack | 0x010 0060 | Section 7.2.18 |
| colorFore | 0x010 0064 | Section 7.2.18 |
| dstSize | 0x010 0068 | Section 7.2.19 |
| dstXy | 0x010 006C | Section 7.2.20 |
| command | 0x010 0070 | Section 7.2.21 |
| Reserved | 0x010 0074 | Section 7.2.22 |
| Reserved | 0x010 0078 | Section 7.2.22 |
| Reserved | 0x010 007C | Section 7.2.22 |
| launchArea | 0x010 0080 - 0x010 00FF | Section 7.2.23 |
| colorPattern | 0x010 0100 - 0x010 01FC | Section 7.2.24 |

## 7.2 2D Register Detailed Descriptions

### 7.2.1 status Register

The Voodoo3 status register is described in Section 6.2.1. This register is accessible at PCI10 + 0x010 0000, PCI18 + 0x00, and PCI4C.

### 7.2.2 intrCtrl

The Voodoo3 intrCtrl registers is described in Section 9.3.2. This register is accessible at PCI10 + 0x010 0004 as well as 0x020 0004.

### 7.2.3 clip0/1Min/Max

The clip registers define the minimum and maximum X and Y coordinates of pixels that will be written into the destination bit map. There are two sets; the active one is selected by command[23]. Clipping is always enabled. To effectively disable clipping, set the min and max to zero and outside the screen dimensions, respectively. See Section 8.2.3.
Clipping is inclusive of the minimum values, and exclusive of the maximum values.

| Register | Offset |
|---|---|
| clip0Min | 0x010 0008 |
| clip0Max | 0x010 000C |
| clip1Min | 0x010 004C |
| clip1Max | 0x010 0050 |

| Bit | Description |
|---|---|
| 31:28 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 27:16 | **Y Clip:** This field specifies the respective minimum or maximum clipping value for clipping register set 0 or 1. |
| 15:12 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 11:0 | **X Clip:** This field specifies the respective minimum or maximum clipping value for clipping register set 0 or 1. |

### 7.2.4 dstBaseAddr

This register contains the physical address in the frame buffer of the pixel at (0, 0) of the destination surface.

| Bit | Description |
|---|---|
| 31 | **Tiled:** If this bit is '1', the destination surface is in tiled memory. If this bit is 0, the destination surface is in linear memory. |
| 30:24 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 23:0 | **Destination Base Address:** This field specifies the base address of the destination surface. This is a byte address. For YUYV 4:2:2 and UYVY 4:2:2 surfaces, this address must be DWORD aligned (the low order two bits must be 0). |

### 7.2.5 dstFormat

This register specifies the format and stride of the destination surface. See Section 8.2.5 for a description of color translations.

| Bit | Description |
|---|---|
| 31:19 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 18:16 | **Destination bpp:** This field specifies the bits per pixel of the destination surface, according to the table. Values not in this table must not be programmed into this field. |

| Value | Destination BPP |
|---|---|
| 001b | 8 bpp |
| 011b | 16 bpp |
| 100b | 24 bpp |
| 101b | 32 bpp |

| | |
|---|---|
| 15:14 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 13:0 | **Destination Stride:** This field specifies the stride of the destination surface. When the destination surface is in linear memory, this is the number of bytes between the starting addresses of adjacent scan lines (bytes, not pixels). When the destination surface is in tiled memory, this is the number of tiles, and only bits [6:0] are used. |

### 7.2.6 srcColorKeyMin/Max

These registers specify the range of colors that will be transparent when source color keying is enabled. Color keying is mechanized by selecting one of four ROPs based on the results of the source and destination color key tests. See Section 7.2.9 for more on ROPs.

A pixel is said to pass the colorkey test if it is within the inclusive range defined by the respective ColorKeyMin and ColorKeyMax registers. If colorkeying is disabled for the source or destination surface, that colorkey test is defined as failing for all pixels.

For pixels with 8 bpp formats, the color indices are compared directly. For pixels with 16-, 24, or 32-bpp formats, each color channel is compared separately and all three must pass for the colorkey test to be passed. In the 32-bpp format, the alpha value is ignored. Source colorkeying cannot be enabled if the source format is 1-bpp.

| Register | Offset |
|---|---|
| srcColorKeyMin | 0x010 0018 |
| scrColorKeyMax | 0x010 001C |

| Bit | Description |
|---|---|
| 31:24 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 23:0 | **ColorKey:** This field contains the respective source color key. See Section 7.2.18 for bit assignments. |

### 7.2.7 dstColorKeyMin/Max

These registers specify the range of colors that will be transparent when destination color keying is enabled.

| Register | Offset |
|---|---|
| dstColorKeyMin | 0x010 0020 |
| dstColorKeyMax | 0x010 001C |

| Bit | Description |
|---|---|
| 31:24 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 23:0 | **ColorKey:** This field contains the respective destination color key. See Section 7.2.18 for bit assignments. |

### 7.2.8 bresError0/1

These registers specify the initial error terms for lines, polygons, and stretch blts.

| Register | Offset | Use |
|---|---|---|
| bresError0 | 0x010 0028 | lines, left edge of polygon, and stretch y-axis |
| bresError1 | 0x010 002C | right edge of polygon and stretch x-axis |

| Bit | Description |
|---|---|
| 31 | **Use Term:** If this bit is '1', the error term in bits 16:0 will be used. If this bit is 0, the error term will be automatically generated by the respective engine. |
| 30:17 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 16:0 | **Error Term:** This field is the respective signed error term. |

### 7.2.9 rop

This register contains three of the four ternary ROPs. The other ROP (ROP0) is specified in the command register. Which of the four ROPs will be used is determined on a per-pixel basis, based on the results of the source and destination colorkey tests. If both source and destination colorkeying are disabled, ROP0 will be used.

**Table 7.2 ROP Selection**

| Source Colorkey Test | Destination Colorkey Test | ROP | Note |
|---|---|---|---|
| Fail | Fail | ROP0 | ROP0 is always used when colorkeying is disabled. |
| Fail | Pass | ROP1 | - |
| Pass | Fail | ROP2 | - |
| Pass | Pass | ROP3 | - |

| Bit | Description |
|---|---|
| 31:24 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 23:16 | **ROP3:** This field specifies ROP3 (pass/pass). |
| 15:8 | **ROP2:** This field specifies ROP2 (pass/fail). |
| 7:0 | **ROP1**: This field specifies ROP1 (fail/pass). |

### 7.2.10 srcBaseAddr

This register contains the physical address in the frame buffer of the pixel at (0, 0) of the source surface. This register is used only for screen-to-screen blts.

| Bit | Description |
|---|---|
| 31 | **Tiled:** If this bit is ‘1’, the source surface is in tiled memory. If this bit is 0, the source surface is in linear memory. |
| 30:24 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 23:0 | **Source Base Address:** This field specifies the base address of the source surface. This is a byte address. For YUYV 4:2:2 and UYVY 4:2:2 surfaces, this address must be DWORD aligned (the low order two bits must be 0). |

### 7.2.11 commandExtra

This register contains a few bits of command specification. This should be treated as an extension to the command register.

| Bit | Description |
|---|---|
| 31:4 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 3 | **Force Pattern Row 0:** When this bit is '1', pattern row zero will be used exclusively for pattern fills. When this bit is 0, all eight pattern rows will be used. |
| 2 | **Wait for VSYNC:** When this bit is '1', the current command, and any following it, will not proceed until the next vertical blanking period. This should not be used when performing non-DMA host-to-screen blts. |
| 1 | **Enable Destination ColorKey:** When this bit is '1', destination colorkeying is enabled. See Section 8.2.7. |
| 0 | **Enable Source ColorKey:** When this bit is '1', source colorkeying is enabled. See Section 8.2.7. |

### 7.2.12 lineStipple

This register specifies the stipple pattern that is used with lineStyle to determine how lines are to be drawn.

| Bit | Description |
|---|---|
| 31:0 | **Line Bit Mask:** This field is the bit mask for stippling. The number of bits actually used is specified in lineStyle[12:8]. |

### 7.2.13 lineStyle

This register specifies how stippled lines will be drawn. See Section 8.3 for a description of how this register is used in conjunction with the lineStipple register to produce stippled lines.

| Bit | Description |
|---|---|
| 31:29 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 28:24 | **Integer Start Position:** This field specifies the integer part of the start position. This selects the first bit in the lineStipple register that is to be used. |
| 23:16 | **Fractional Start Position:** This field specifies the fractional part of the start position. This specifies the number of pixels that are to be taken as already having been drawn using the current bit in the lineStipple register. |
| 15:13 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 12:8 | **Stipple Size:** This field specifies the number of bits in the lineStipple register that will be used. |
| 7:0 | **Repeat Count:** This field specifies the number of pixels that will be drawn with each bit in the lineStipple register. |

### 7.2.14 pattern0/1Alias

These two registers access pattern0 and pattern1.

| Register | Offset | Offset of corresponding register in colorPattern |
|---|---|---|
| pattern0Alias | 0x010 0044 | 0x010 0100 |
| pattern1Alias | 0x010 0048 | 0x010 0104 |

These two addresses provide an alternate way to access the first two registers (first 64 bits) of the colorPattern area).

### 7.2.15 srcFormat

This register specifies the format and stride of the source surface. See Section 8.2.5 for a description of color translations.

**Bit** — **Description**

31:24 — **Reserved:** Reserved bits must be written as 0 for upward compatibility.

23:22 — **Source Packing:** This field specifies the packing of the source surface according to the table:

| Value | Packing | Stride Calculation |
|---|---|---|
| 00b | Use stride register | srcFormat[13:0] |
| 01b | Byte packed | ceil (src_width * src_bpp / 8) |
| 10b | Word packed | ceil (src_width * src_bpp / 16) * 2 |
| 11b | DWORD packed | ceil (src_width * src_bpp / 32) * 4 |

21 — **Host Port Word Swizzle:** If this bit is '1',word swizzling is enabled. The two 16-bit words are swapped.

20 — **Host Port Byte Swizzle:** If this bit is '1', byte swizzling is enabled. Byte three is swapped with byte zero and byte one is swapped with byte two. If both word and byte swizzling are enabled, byte swizzling is done first.

19:16 — **Source Color Format:** This field specifies the format of the source surface, according to the table. Values not in this table must not be programmed into this field.

| Value | Source Format | Note |
|---|---|---|
| 0000b | 1 bpp monochrome | |
| 0001b | 8 bpp palettized | |
| 0011b | 16 bpp RGB | |
| 0100b | 24 bpp RGB | |
| 0101b | 32 bpp RGB | 24 bpp, plus an unused byte |
| 1000b | Packed 4:2:2 YUYV | |
| 1001b | Packed 4:2:2 UYVY | |

15:14 — **Reserved:** Reserved bits must always be written as 0 for upward compatibility.

13:0 — **Source Stride:** This field specifies the stride of the source surface. When the source surface is in linear memory, this is the number of bytes between the starting addresses of adjacent scan lines (bytes, not necessarily pixels). When the source surface is in tiled memory, this is the number of tiles, and only bits [6:0] are used.

### 7.2.16 srcSize

This register specifies the height and width (in pixels) of the source rectangle. This is used only for screen-to-screen stretch blts.

| Bit | Description |
|---|---|
| 31:29 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 28:16 | **Source Height:** This field contains the height of the source rectangle for screen-to-screen stretch blts. |
| 15:13 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 12:0 | **Source Width:** this field contains the width (in pixels) of the source rectangle for screen-to-screen stretch blts. |

### 7.2.17 srcXY

This register contains the X-Y coordinates of the starting pixel of the source. For screen-to-screen blts, this register specifies the starting pixel of the source rectangle as indicated in the description of command[15:14] (it could be any of the four corners). For line and polyline, this register specifies the starting point of the first (or only) line segment. For polygon fill, this register specifies the topmost vertex of the polygon. Reading this register while in polygon mode will always return the last polygon vertex sent for the left side of the polygon. For host-to-screen blts, only the X position field is used (see the description). The values in this register are signed. However, for blts, this register must contain positive values.

| Bit | Description |
|---|---|
| 31:30 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 29 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 28:16 | **Destination Surface Y Position:** This field specifies the signed Y position on the destination surface. |
| 15:14 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 13 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 12:0 | **Destination Surface X Position:** This field specifies the signed X position on the destination surface. For host-to-screen blts, only this entry of srcXY is used. It determines the alignment of the initial pixel in the blt within the first DWORD sent by the host. For monochrome (source) bitmaps, bit [4:0] specify the bit position of the first pixel in the DWORD. For color bitmaps, bits [1:0] specify the position of the first pixel in the DWORD. |

### 7.2.18 colorBack/colorFore

These registers specify the colors used in solid-fill and monochrome bitmap operations, and operations using a monochrome pattern (including stippled lines). These registers contain the respective colors in the destination color format. These formats are also used for the colorKey registers.

| Bit | Description |
|---|---|
| 31:0 | **Color:** The table indicates which bits are used for each color value for each format. The format must match the destination format. |

| Format | Index | Attribute | Red | Green | Blue |
|---|---|---|---|---|---|
| 8 bpp | 7:0 | - | - | - | - |
| 15 bpp | - | 15 | 14:10 | 9:5 | 4:0 |
| 16 bpp | - | - | 15:11 | 10:5 | 4:0 |
| 24 bpp | - | - | 23:16 | 15:8 | 7:0 |
| 32 bpp | - | 31:24 | 23:16 | 15:8 | 7:0 |

### 7.2.19 dstSize

This register specifies the height and width (in pixels) of the destination rectangle. This is used for all blts and rectangle fills.

| Bit | Description |
|---|---|
| 31:29 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 28:16 | **Destination Height:** This field contains the height of the destination rectangle for blts and rectangle fills. |
| 15:13 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 12:0 | **Destination Width:** this field contains the width (in pixels) of the destination rectangle for blts and rectangle fills. |

### 7.2.20 dstXY

This register contains the X-Y coordinates of the starting pixel of the destination. For blts, this register specifies the starting pixel of the destination rectangle as indicated in the description of command[15:14]. For line and polyline, this register specifies the endpoint of the first (or only) line segment. For polygon fills, this register contains the most recent vertex sent for the right side.

| Bit | Description |
|---|---|
| 31:30 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 29 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 28:16 | **Destination Surface Y Position:** This field specifies the signed Y position on the destination surface. |
| 15:14 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 13 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 12:0 | **Destination Surface X Position:** This field specifies the signed X position on the destination surface. |

### 7.2.21 command

This register sets the command mode for the 2D engine and selects a number of options.

| Bit | Description |
|---|---|
| 31:24 | **ROP0:** This field specifies ROP0, used when colorkeying is disabled or when both colorkeys fail. |
| 23 | **Clip Select:** When this bit is 0, clipping values from clip0Min and clip0Max are used. When this bit is 1, clipping values from clip1Min and clip1Max are used. Clipping is always enabled. |
| 22:20 | **Y Pattern Offset:** This field, in conjunction with X pattern offset in bits [19:17], specifies the coordinates within the pattern of the pixel which corresponds to the destination pixel pointed to by the destination base address. This allows the pattern to be positioned arbitrarily on the destination surface. |
| 19:17 | **X Pattern Offset:** This field is used in conjunction with the Y pattern offset. |
| 16 | **Transparent Monochrome:** When this bit is 0, source bitmaps are opaque. A 0 in the source bitmap results in colorBack being written to the corresponding destination pixel. When this bit is 1, source bitmaps are transparent. A 0 in the source bitmap results in the corresponding destination pixel not being changed.<br>This bit also controls whether stippled lines are two colors or a transparent stipple. |
| 15:14 | **X/Y Direction:** These two bits control the directions in which the destination is traversed (the direction of bliting). BLTs are always done by scanline.<br>The corner of the source and destination rectangles will change according to these bits (they will always refer to the first pixel processed).<br>Bit 15 also controls the direction of bliting for host-to-screen blts. This can be used to vertically flip a pixel map. |

| Bit 15 (X) | Bit 14 (Y) | Direction | Note |
|---|---|---|---|
| 0 | 0 | top left to bottom right | ‘normal’: can be used with all BLTs |
| 0 | 1 | top right to bottom left | no color keying, color conversions, stretch blts. |
| 1 | 0 | bottom left to top right | no stretch blts. |
| 1 | 1 | bottom right to top left | no color keying, color conversions, stretch blts. |

## 7.2.21 command *(cont)*

| Bit | Description |
|---|---|
| 13 | **Pattern Format (this really ought to be called colorExpand):** When this bit is 0, the source is not expanded, but rather is written to the destination just as it appears (color conversion will take place if necessary). When this bit is 1, a 64-bit monochrome pattern is expanded to color and repeatedly used to fill the destination.<br>The location of the pixel within the pattern used for the first pixel is specified by X pattern offset and Y pattern offset.<br>The 8-bit x 8-bit pattern is taken from pattern0 and pattern1. Depending on bit 16, the pattern is expanded according to the table. |
| 12 | **Stipple Line Mode:** If this bit is 0, lines will be drawn in a solid color. If this bit is 1, lines are stippled using the line bit-mask in the lineStipple register (see Section 8.3). Bit 16 specifies whether pixels in the line corresponding to 0s in the lineStipple register are drawn using the backColor register, or skipped. |
| 11:10 | **Adjust Dst XY:** These two bits specify whether the respective fields in dstXY are adjusted after each blt or rectangle fill. If bit 11 is 1 dst_X is loaded with dst_X + dst_width. If bit 10 is '1', dst_Y is loaded to dst_Y + dst_height. |
| 9 | **Reversible:** When this bit is '1', lines are drawn so that exactly the same pixels are chosen regardless of whether the line is drawn from X1, Y1 to X2, Y2 or from X2, Y2 to X1, Y1. This is more important and difficult than it sounds. |
| 8 | **Initiate:** When this bit is written as '1', the command will be initiated immediately. When this bit is written as '0', the command will begin with a write to the launch area. |
| 7:4 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |

| Pattern Bit | Opaque (bit 16 = 0) | Transparent (bit 16 = 1) |
|---|---|---|
| 0 | write colorBack | no write |
| 1 | write colorFore | write colorFore |

### 7.2.21 command *(cont)*

| Bit | Description |
|---|---|

3:0 **Command:** This field specifies the command according to the table. Values not in this table must not be written to this field.

| Command | Description | Note |
|---|---|---|
| 0000b | Nop | Wait for idle |
| 0001b | Screen-to-screen blt | |
| 0010b | Screen-to-screen stretch blt | |
| 0011b | Host-to-screen blt | |
| 0100b | Host-to-screen stretch blt | |
| 0101b | Rectangle fill | |
| 0110b | Line Draw | |
| 0111b | Polyline Draw | |
| 1000b | Polygon Fill | |
| 1101b | Write SGRAM Mode Register | |
| 1110b | Write SGRAM Mask Register | |
| 1111b | Write SGRAM Color Register | |

### 7.2.22 Reserved

There are three reserved addresses at 0x010 0074, 0x010 0078, and 0x010 007C. Do not write to these locations. Something very bad might happen.

### 7.2.23 launchArea

Writing to the launch area after a command mode has been set (with command[8] written as 0) initiates the operation. The format and meaning of the data written to the launch area depends on the command mode. The following descriptions are organized by command mode.

#### 7.2.23.1 Screen-to-screen BLT Mode

Writing the launch area while in screen-to-screen BLT mode results in a rectangle being copied from area of the frame buffer to another. The position of the source rectangle is specified in the write to the launch area; the data will be used to load the srcXY register.

| Bit | Description |
|---|---|
| 31:29 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 28:16 | **Source Y:** This field specifies the Y position of the source rectangle. |
| 15:13 | **Reserved:** Reserved bits must be written a 0 for upward compatibility. |
| 12:0 | **Source X:** This field specifies the X position of the source rectangle. |

#### 7.2.23.2 Screen-to-screen Stretch BLT Mode

Writing the launch area while in screen-to-screen stretch BLT mode results in a rectangle being copied from area of the frame buffer to another (which may be a different size). The position of the source rectangle is specified in the write to the launch area; the data will be used to load the srcXY register. The X and Y direction bits do not apply to stretch BLTs. Only top-to-bottom, left-to-right stretch BlTs can be done.

| Bit | Description |
|---|---|
| 31:29 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 28:16 | **Source Y:** This field specifies the Y position of the source rectangle. |
| 15:13 | **Reserved:** Reserved bits must be written a 0 for upward compatibility. |
| 12:0 | **Source X:** This field specifies the X position of the source rectangle. |

#### 7.2.23.3 Host-to-Screen BLT Mode

The data written to the launch area in host-to-screen BLT mode is taken to contain packed pixels used as source data. The application must have programmed the srcFormat register to specify how the source data is packed.

The application must also program the srcXY register to specify the first byte or bit to use from the first DWORD. For monochrome source data, the low order five bits will specify the initial bit; in all other modes, the low order two bits will specify the initial byte. The alignment of the first pixel of each span after the first is determined by adding the source stride (contained in srcFormat) to the alignment of the previous span. If more data is written to the launch area than is required for the host-to-screen blt, the extra data may be discarded or may be used in the following host-to-screen blt if it is requested while the 2D engine is operating on the first blt. If too little data is written to the launch area, the host-to-screen blt will be aborted, and pixels on an incomplete span at the end of the blt may or may not be drawn. It is best to write just as many bytes as are necessary.

#### 7.2.23.4 Host-to-Screen Stretch BLT Mode

Data written to the launch area in host-to-screen stretch BLT mode is taken to contain packed pixels that will be stretched onto the destination rectangle. This works just like host-to-screen BLTs, except when the destination height differs from the source height. In this case, the host must replicate or decimate the source spans to match the number of destination spans required.

#### 7.2.23.5 Rectangle Fill Mode

Data written to the launch area in rectangle fill mode is taken to be the position of the destination rectangle, used to fill the dstXY register.

| Bit | Description |
|---|---|
| 31:29 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 28:16 | **Source Y:** This field specifies the Y position of the destination rectangle. |
| 15:13 | **Reserved:** Reserved bits must be written a 0 for upward compatibility. |
| 12:0 | **Source X:** This field specifies the X position of the destination rectangle. |

#### 7.2.23.6 Line Mode

Data written to the launch area in line mode is taken to be the end point of a line segment. It is used to fill the dstXY register, and a line is drawn from srcXY to dstXY. After the line is drawn, dstXY is copied to srcXY. In line mode, all pixels in the line will be drawn (as specified by the line style register), including both the start and end point.

| Bit | Description |
|---|---|
| 31:29 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 28:16 | **Source Y:** This field specifies the Y position of the end point. |
| 15:13 | **Reserved:** Reserved bits must be written a 0 for upward compatibility. |
| 12:0 | **Source X:** This field specifies the X position of the end point. |

#### 7.2.23.7 Polyline Mode

Data written to the launch area in line mode is taken to be the end point of the line segment. It is used to fill the dstXY register, and a line is drawn from srcXY to dstXY. After the line is drawn, dstXY is copied to srcXY. In polyline mode, the start point will be drawn and the end point will not be drawn. This guarantees that each pixel in a non-overlapping polyline will be written only once.

| Bit | Description |
|---|---|
| 31:29 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 28:16 | **Source Y:** This field specifies the Y position of the end point. |
| 15:13 | **Reserved:** Reserved bits must be written a 0 for upward compatibility. |
| 12:0 | **Source X:** This field specifies the X position of the end point. |

#### 7.2.23.8 Polygon Fill Mode

Data written to the launch area in polygon fill mode is taken to be vertex data. See Section 8.6 for a description of polygon fill.

| Bit | Description |
|---|---|
| 31:29 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 28:16 | **Source Y:** This field specifies the Y position of a polygon vertex. |
| 15:13 | **Reserved:** Reserved bits must be written a 0 for upward compatibility. |
| 12:0 | **Source X:** This field specifies the X position of a polygon vertex. |

### 7.2.24 colorPattern

These registers contain an 8 pixel by 8 scanline pattern. The pattern pixels must be either in the color format of the destination surface, or in 1 bpp (monochrome) format. The colors are packed into as few registers as are necessary to contain the 64 pixels.
Patterns are written into the pattern registers starting with the upper left-hand corner of the pattern, proceeding left to right and then top to bottom. The least-significant bits of pattern always contain pixel(0,0) of a color pattern.

**Table 7.3 Color Pattern Register Usage**

| Format | Total Bits (64 pixels) | Registers | Last Offset |
|---|---|---|---|
| Monochrome | 64 | 2 | 0x010 0107 |
| 8 bpp | 512 | 16 | 0x010 013F |
| 16 bpp | 1024 | 31 | 0x010 017F |
| 24 bpp | ? | ? | ? |
| 32 bpp | 2048 | 64 | 0x010 01FF |

The following table shows the order of pixel storage in the pattern registers for a monochrome pattern. The top four rows of the pattern are in pattern register 0; the bottom four rows of the pattern are in pattern register 1. The array in the table represents the eight by eight pattern as it appears on the screen. The number in each cell is the bit number in the respective register.

**Table 7.4 Monochrome Pattern**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| pattern(0) | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
| pattern(1) | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |

# 8 2D Programming Notes

This chapter covers programming of the 2D engine. This is intended to be used in conjunction with the formal register descriptions in Chapter 7.

## 8.1 Overview

Voodoo3 2D engine programming is straight-forward. The application must program a number of registers and then issue the command to actually blt the bits, draw the line, or fill the rectangle. Most commands can be issued by writing to the launch area; in this case, one parameter is written along with the command itself.
Table 8.1 shows the functions that are available. There are also commands to explicitly control the SGRAMs.

### Table 8.1 2D Functions

| Function | Color Expansion | Pattern | Color Translation |
|---|---|---|---|
| Screen-to-screen BLT | Yes | Yes | Yes |
| Screen-to-screen stretch Blt | No | No | ? |
| Host-to-screen Blt | Yes | Yes | Yes |
| Host-to-screen stretch Blt | No | No | ? |
| Rectangle Fill | Yes | Yes | No |
| Line | Yes | Stipple | No |
| Polyline | Yes | Stipple | No |
| Polygon Fill | Yes | Yes | No |

## 8.2 Common Items

### 8.2.1 Frame Buffer Addressing

An address in the frame buffer is specified to the 2D engine as an XY offset from a base address. The following equation is used to determine the linear byte address of a destination byte in the frame buffer:

$$addr = dstBaseAddr + (dstY \bullet dstStride) + (dstX \bullet PixelSize)$$

There are two base address registers. scrBaseAddr is used only for screen-to-screen blt operations. dstBaseAddr is used to specify the frame buffer address of pixel 0,0 of the destination surface. The high order bit of each register indicates whether the respective surface is tiled.
The XY offsets are specified in srcXy and dstXY. Table 8.2 shows how these offsets are used for each 2D operation.

### Table 8.2 srcXY, dstXY

| Operation | srcXY | dstXY |
|---|---|---|
| Screen-to-screen BLT | offset of first source pixel | offset of first destination pixel |
| Screen-to-screen stretch Blt | offset of first source pixel | offset of first destination pixel |

**Table 8.2 srcXY, dstXY *(cont.)***

| Operation | srcXY | dstXY |
|---|---|---|
| Host-to-screen Blt | alignment of initial pixel | offset of first destination pixel |
| Host-to-screen stretch Blt | alignment of initial pixel | offset of first destination pixel |
| Rectangle Fill | unused | offset of first destination pixel |
| Line | start point of first segment | end point of first segment |
| Polyline | start point of first segment | end point of first segment |
| Polygon Fill | topmost, left side vertex | right side vertex |

### 8.2.2 Launch Area

Operations can be initiated when the command is issued, or can be initiated when a parameter is written to the launch area. If bit 8 is '1' when the command register (see Section 7.2.21) is written, the operation is initiated immediately. All the parameters must have been loaded into the respective registers. This method is perhaps most useful when a single command is to be executed (for example, clearing the entire screen). If bit 8 is 0 when the command register is written, execution will be deferred until the final parameter (for example, polyline end point) is written to the launch area. This method is useful when a number of similar operations are to take place and only a single parameter must be transferred for each. In the sections that follow, some examples will use this method.
The launch area occupies 128 bytes. This means that a movs command that increments the address can be used to send up to 32 parameters (polyline end points, for example).

### 8.2.3 Clipping

There are two sets of registers that define clipping boxes; the set to be used in specified in bit 23 of the command. Clipping is always enabled, as specified in one or the other of the two register sets. Each register set (see Section 7.2.3) contains minimum and maximum x-y coordinates. Clipping is inclusive of the minimum values and exclusive of the maximum values. For example, to confine writes to a 640 x 480 area based at 0, 0 (that is, 0, 0 through 639, 479), one would write 0, 0 to the minimum register and 640, 480 to the maximum register.

### 8.2.4 Color Expansion

The Voodoo3 supports color expansion. Two internal registers (colorFore and colorBack (see Section 7.2.18)) contain a foreground color and a background color. When bit 13 of the command is '1', the source will be interpreted as a monochrome (one bit per pixel) image and will be expanded into the colors specified in the color registers.
The expansion can be either transparent or opaque, based on command[16]. For transparent expansion, only pixels corresponding to '1' bits in the source are written. For opaque expansion, pixels corresponding to '1's and 0s are written. Table 8.3 shows what is written into the frame buffer depending on whether bit 16 of the command is set.

**Table 8.3 Color Expansion**

| Source Bit | Transparent (Command[16] = 1 | Opaque (Command[16] = 0 |
|---|---|---|
| 0 | No write | Write colorBack |
| 1 | Write colorFore | Write colorFore |

### 8.2.5 Formats

The formats of the surface or surfaces are specified in srcFormat (Section 7.2.15) and dstFormat (Section 7.2.5). The number of bits per pixel for each surface is specified. Also, each register specifies the stride of the respective surface.
Data coming through the host port can be byte or word swizzled, according to srcFormat[21:20].
When necessary, the 2D engine will convert source pixels to the destination format. When source pixels in 15- or 16-bpp format are converted to 24- or 32-bpp, color conversion is performed by replicating the MSBs of each channel into the additional LSBs. When source pixels in 24- or 32-bpp format are converted to 15- or 16-bpp, LSBs are truncated. When any format (except 32-bpp) is converted to 32-bpp, the alpha channel is zeroed. Table 8.4 shows the format conversion. To use this table, find the source format across the top and the destination format down the left. The intersection specifies the conversion method.

**Table 8.4 Format Conversions**

| src/dst | 1 bpp | 8-bpp | 15-bpp | 16-bpp | 24-bpp | 32-bpp | YUV |
|---|---|---|---|---|---|---|---|
| 8-bpp | color registers | direct or palette | n/a[a] | n/a | n/a | n/a | n/a |
| 15-bpp | color registers | n/a | none[b] | LSB dropped | LSB dropped | LSB, alpha dropped | YUV to RGB |
| 16-bpp | color registers | n/a | MSB duplication | none | LSB dropped | LSB, alpha dropped | YUV to RGB |
| 24-bpp | color registers | n/a | MSB duplication | MSB duplication | none | alpha dropped | YUV to RGB |
| 32-bpp | color registers | n/a | MSB duplication | MSB duplication | zero alpha | none | YUV to RGB |

a. The translation is not supported, probably because it does not make sense.
b. Source and destination formats are the same. No translation is required.

Destination pixel formats, and RGB source pixel formats, are stored as shown in the description of the colorFore and colorBack (Section 7.2.18). Monochrome and YUV source formats are shown in Table 8.5. For a monochrome source, p0 represents the left-most pixel on the screen and p31 represents the right-most. For YUV formats, ya represents the left pixel and yb represents the right pixel.

**Table 8.5 Monochrome, YUV Source Formats**

| Monochrome | UYVY 4:2:2 | YUYV 4:2:2 |
|---|---|---|
| p24 | yb7 | v7 |
| p25 | yb6 | v6 |
| p26 | yb5 | v5 |
| p27 | yb4 | v4 |
| p28 | yb3 | v3 |
| p29 | yb2 | v2 |

**Table 8.5 Monochrome, YUV Source Formats *(cont.)***

| Monochrome | UYVY 4:2:2 | YUYV 4:2:2 |
|---|---|---|
| p30 | yb1 | v1 |
| p31 | yb0 | v0 |
| p16 | v7 | yb7 |
| p17 | v6 | yb6 |
| p18 | v5 | yb5 |
| p19 | v4 | yb4 |
| p20 | v3 | yb3 |
| p21 | v2 | yb2 |
| p22 | v1 | yb1 |
| p23 | v0 | yb0 |
| p8 | ya7 | u7 |
| p9 | ya6 | u6 |
| p10 | ya5 | u5 |
| p11 | ya4 | u4 |
| p12 | ya3 | u3 |
| p13 | ya2 | u2 |
| p14 | ya1 | u1 |
| p15 | ya0 | u0 |
| p0 | u7 | ya7 |
| p1 | u6 | ya6 |
| p2 | u5 | ya5 |
| p3 | u4 | ya4 |
| p4 | u3 | ya3 |
| p5 | u2 | ya2 |
| p6 | u1 | ya1 |
| p7 | u0 | ya7 |

### 8.2.6 Forcing Error Terms

Bresenham's algorithm is a special case of a DDA (Digital Differential Analyzer), and is used to find pixels along a straight line. The ability to find pixels along a straight line is useful, for example, when drawing a line or finding the edge of a polygon. This deservedly famous contribution to computer science is described in all introductory graphics books and classes.
Registers bresError0 and bresError1 can be used to force the initial error terms for lines, polygons, and stretch blts. These are signed values. Bit 31 of each register specifies whether the error term in the rest of the register is to be used. If the bit is 0, the engine will automatically generate the respective initial error. If the bit is '1', the value in bits 16:0 will be used. After an error term has been specified, it is needful to write 0 to the high order bit of the respective register to prevent the term from being used by subsequent operations.

### 8.2.7 Color Keying

There are four color key registers, enumerated in Table 8.6. Each register pair (MinMax) specifies a range of colors to which each source and destination pixel is compared. The colorkey register format is the same as the colorFore register format.

**Table 8.6 2D Color Key Registers**

| Register | Offset | Note |
|---|---|---|
| srcColorKeyMin | 0x010 0018 | |
| srcColorKeyMax | 0x010 001C | |
| dstColorKeyMin | 0x010 0020 | |
| dstColorKeyMax | 0x010 0024 | |

A pixel is said to pass the colorkey test if it is within the inclusive range defined by the respective Min/Max register pair. If colorkeying is disabled for the source or destination surface, the respective test is defined as having failed. The pass/fail outcomes are used to select alternate ROPs.
For pixels with 8-bpp formats, the color indices are compared directly. For pixels with 16-, 24-, or 32-bpp formats, each color channel is compared separately and all three must pass for the colorkey test to be passed. In the 32-bpp format, the alpha value is ignored. Source colorkeying cannot be enabled if the source format is 1-bpp.
Depending on the outcomes of the comparisons, alternate ROPs can be selected, as indicated in Table 8.7. ROP0 is specified in the command register; the alternate ROPs are specified in the rop register.

**Table 8.7 ROP Selection**

| Source Colorkey Test | Destination Colorkey Test | ROP |
|---|---|---|
| Fail | Fail | ROP0 |
| Fail | Pass | ROP1 |
| Pass | Fail | ROP2 |
| Pass | Pass | ROP3 |

## 8.3 Lines

Voodoo3 draws lines when the command mode (command[3:0] (see Section 7.2.21)) is Line or Polyline. If bit 8 of the command is '1', a single segment will be written for each write to the command register. If bit 8 of the command is 0, a single segment will be written for each write to the launch area.
If bit nine of the command is '1', the line will be reversible. This means that exactly the same pixels are drawn regardless of whether the line is drawn from X1, Y1 to X2, Y2 or from X2, Y2 to X1, Y1.
The (only) difference between line mode and polyline mode is that the endpoint may be written in line mode (depending on the stipple), but is never written in polyline mode. The contents of dstXY will be copied to srcXY for either mode.
When writing to the launch area in line or polyline mode, the data is loaded into dstXY. After the segment is drawn, dstXY is copied to srcXY.

### 8.3.1 Line Stippling

Lines can be drawn as solid or stippled. This is controlled by bits 13 and 12 of the command register, the lineStipple and lineStype registers, and the colorFore and colorBack registers.

**Table 8.8 Line Stippling Overview**

| command[12] | command[13] | stipple = 0 | stipple = 1 |
|---|---|---|---|
| 0 | x | draw solid line using colorFore | |
| 1 | 0 | colorBack | colorFore |
| 1 | 1 | no write | colorFore |

The line pattern is specified in the lineStyle register (see Section 7.2.13) and the lineStipple register (see Section 7.2.12).
The bit mask (the stipple pattern) is contained in the lineStipple register, beginning with bit 0 and extending through the bit whose number is specified in the Stipple Size field of the lineStyle register. For example, to use two bits to represent a dashed line, the Stipple Size field would be set to '1' and lineStipple[1:0] would be set to 01b or 10b. Since the mask can be as large as 32 bits, it is possible to draw a line with a complex pattern.
Each bit from the lineStipple register will be used for between one and 256 pixels. The number is specified in the Repeat Count of the lineStyle register (the actual count is one greater than the value in the Repeat Count field). This allows one's favorite type of line stippling to be easily scaled (but only by integer factors).
The Integer Start Position field of the lineStyle register specifies the first bit in the lineStipple register to be used; the Fractional Start Position specifies the number of pixels that are to be taken as already having been drawn using that bit of the pattern. The number of pixels drawn using the initial bit in the line pattern will equal the Repeat Count (the field entry plus one) minus the Fractional Start Position field. To begin a line at the beginning of the pattern, program both the Integer and Fractional Start Positions to zero.
It is illegal to set the Integer Start Position to greater than the Stipple Size field. It is illegal to set the Fractional Start Position to greater than the Repeat Count.
When the lineStyle register is written, the stipple position (the pointer into the lineStipple register) is loaded with the new value specified.
If the lineStyle register is not written between the execution of two line commands, the stipple position will of the new line will be whatever is left over from the old line. This is useful for continuing a line style around an inflection point.
Reading the lineStyle register while the 2D engine is idle will return the stipple position that which will be used in the next line. Reading the lineStyle register while the 2D engine is busy will return indeterminate data.

### 8.3.2 Line Stippling Examples

The following codes are used in the line stippling examples that follow. Each pixel in the line is represented by a two-character code; the codes are separated in the tables by spaces.

**Table 8.9 Line Stippling Example Codes**

| Code | Color Register | Case |
|---|---|---|
| NB | colorBack (or no write) | First pixel using current bit of lineStipple |
| NF | colorFore | First pixel using current bit of lineStipple |
| SB | colorBack (or no write) | Subsequent pixel using current bit |
| SF | colorFore | Subsequent pixel using current bit |
| FB | colorBack (or no write) | First pixel using zeroth bit of lineStipple |
| FF | colorFore | First pixel using zeroth bit of lineStipple |

#### 8.3.2.1 Line Stippling Example 1

In this example, the Stipple Size field is five and the low order bits of the lineStipple register are 010111b. This will produce a Dash-Dot line with petite spaces. Both the Integer and Fractional Start Positions are zero. According to the Repeat Count, the following lines will be drawn.

**Table 8.10 Line Stippling Example 1**

| Repeat Count (register value) | Pixel Pattern |
|---|---|
| 0 | FF NF NF NB NF NB FF |
| 1 | FF SF NF SF NF SF NB SB NF SF NB SB FF SF |
| 2 | FF SF SF NF SF SF NF SF SF NB SB SB NF SF SF NB SB SB FF SF SF |

#### 8.3.2.2 Line Stippling Example 2

In this example, the Integer Start Position is seven, the Fractional Start Position is two, and the Stipple Size field is nine. The low order ten bits of the lineStipple register are 1010110111b. According to the Repeat Count, the first few pixels of the line will be:

**Table 8.11 Line Stippling Example 2**

| Repeat Count (register value) | Pixel Pattern |
|---|---|
| 1 | This would be illegal (fractional start position > repeat value) |
| 2 | SF NB SB SB NF SF SF FF |
| 3 | SF SF NB SB SB SB NF SF SF SF FF |
| 4 | SF SF SF NB SB SB SB SB NF SF SF SF SF FF |

## 8.4 BitBLTs

### 8.4.1 Overview

There are four flavors of bitBLTs, as enumerated in Table 8.12. Screen-to-screen blts use the frame buffer (not necessarily data that is on the screen) as the source surface. Host-to-screen blts accept data from the host as source information. For a common or garden variety blt, the source and destination surfaces are the same size (in terms of pixels, not necessarily bytes). For a stretch blt, the source and destination surfaces can be (and typically are) different sizes.
Blts can also involve format conversions. If the source and destination surfaces have different formats, conversion will be automatically invoked.

**Table 8.12 BitBLTs**

| | Screen-to-screen | Host-to-screen |
|---|---|---|
| **blt** | command[3:0] = 0001b | command[3:0] = 0011b |
| **stretch** | command[3:0] = 0010b | command[3:0] = 0100b |

### 8.4.2 Screen-to-screen

A screen-to-screen blt copies from scrBaseAddr, offset by scrXY to dstBaseAddr, offset by dstXY. The size of the surfaces (they are the same in terms of pixels and scan lines) is specified in dstSize. The srcSize register is not used.
Colorkeying can be used on both surfaces. Format conversion will be invoked if the source and destination surfaces do not have the same format. Color expansion, transparent or opaque, can be used.
If the launch area is used, the value written is loaded into scrXY. Note that command[11:10] can be programmed to automatically update dstXY at the end of each blt.

### 8.4.3 Host-to-screen

A host-to-screen blt copies from the host port to dstBaseAddr, offset by dstXY. scrBaseAddr is not used at all. srcXY is used to specify the alignment of the initial pixel sent from the host. The size of the destination area is specified in dstSize. The application must send just as many pixels as are needed, no more and no less. The srcSize register is not used.
Colorkeying can be used on the destination surface only. Format conversion will be invoked if the source and destination surfaces do not have the same format. Color expansion, transparent or opaque, can be used.
In host-to-screen blt mode, the launch area should be written with packed pixels to be used as source data. The scrFormat register specifies how the source data is packed.

### 8.4.4 Stretch

For stretch blts, the srcSize register is used to specify the size (in pixels and scan lines) of the source surface. This surface will be resized (interpolation?) to fit the destination.

### 8.4.5 Host blt Example 1

The driver is drawing text to a 1024 x 768 x 16-bpp screen using monochrome bitmaps of various width (not a fixed-width font). The monochrome data is packed, with each row aligned on a byte boundary. The driver begins by setting up register before transmitting the data specific to the first blt.

**Table 8.13 Host blt Example 1**

| Register | Description | Note |
|---|---|---|
| colorBack | background color | only for opaque text |

**Table 8.13 Host blt Example 1 *(cont.)***

| Register | Description | Note |
|---|---|---|
| colorFore | foreground color | |
| dstXY | starting position of first character | |
| dstBaseAddr | base address of destination surface | |
| clip0Min | 0x00000000 | clipping is off |
| clip0Max | 0xFFFFFFFF | |
| command | SRC_COPY, HOST_BLT_MODE | 0xCC000003 |
| dstFormat | 16-bpp, destination stride | 0x00030800 |
| srcFormat | 1-bpp monochrome, byte packed | 0x00400000 |

The command mode is set to host-to-screen blt, with all other features disabled. Since colorkeying is disabled, only ROP0 is required. The srcFormat register sets the host format to unswizzled monochrome, using byte-packing. This means the stride will not have to be set for each blt, but rather will be set to the number of bytes required to store the number of pixels in the source width. Since this is not a stretch blt, the source width equals the destination width, as set later in dstSize.
The clipping registers are set to effectively turn off clipping.
Even though this is a host-to-screen blt, the scrXY register must be set in order to specify the initial alignment of the bitmask. For this example, the source data begins with the lsb of the first dword of host data, so srcXY is set to zero. The driver will set a dstSize value and srcXY value (for alignment) before sending each character to the launch area.

**Table 8.14 Host blt Example 1 Continued**

| Register | Description | Note |
|---|---|---|
| dstSize | 11 x 7 | 0x0007000B |
| srcXY | DWORD alignment | 0x00000000 |
| launch | 0xC0 60 80 20 | First two rows |
| launch | 0xC4 60 C0 60 | Second two rows |
| launch | 0x3B 80 6E C0 | Third two rows |
| launch | 0x00 00 11 00 | Seventh row |

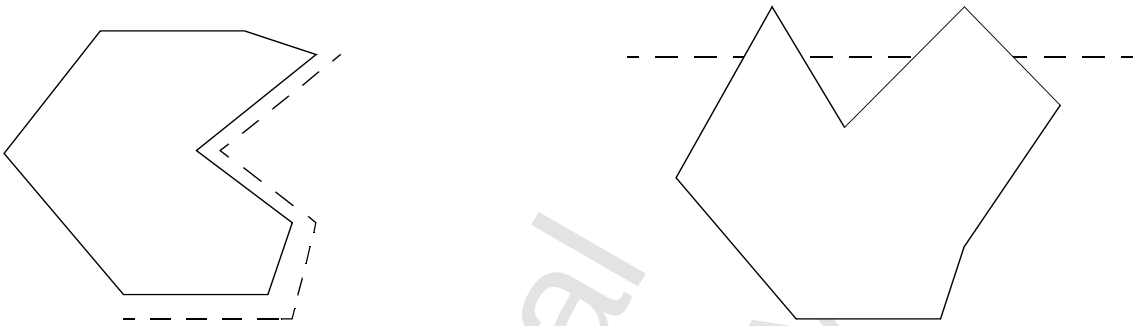
## 8.5 Rectangle Fill

In rectangle fill mode, Voodoo3 writes a single color to a rectangular area. This is typically used to clear a buffer. The rectangle is at dstBaseAddr, offset by dstXY. The size of the rectangle is specified in dstSize. The color is specified in colorFore. The pixel format is specified in dstFormat.
If the launch area is used, the value written is loaded into dstXY.

## 8.6 Polygon Fill

### 8.6.1 Overview

The polygon fill mode can be used to draw simple polygons. A polygon may be drawn using 2D engine if no horizontal span intersects more than two non-horizontal polygon edges. Figure 8.1 below shows two polygons. The polygon on the left can be drawn by the 2D engine; the polygon on the right cannot and must be further decomposed. Note that a polygon can be concave on the sides and still be drawable. It is just the top and bottom that must be convex (or horizontal)/



**Drawable Polygon** **Non-drawable Polygon**

**Figure 8.1 Drawable and non-drawable Polygons**

Once a polygon is determined to be drawable, the application sets up the color(s), ROP, and pattern. The starting vertex is set, the command is written, and then the remaining vertices are written to the launch area in order of increasing y value. See the following example; it covers one polygon in excruciating detail. The ROP used for filling polygons can use the pattern and the destination, but not source data. The colorFore register will be used in the ROP in place of source data. Source colorekying cannot be used, destination colorkeying can.

Pixels that are on the line that forms the left edge of the polygon will be drawn. Pixels that are on the line that forms the right edge of the polygon will not be drawn. For horizontal edges, pixels on a horizontal polygon edge that is on top of the polygon (that is, above the edge is outside the polygon and below the edge is inside the polygon) will be drawn. Pixels on a horizontal polygon edge that is on the bottom of the polygon will not be drawn. In Figure 8.1 above, pixels on the edges next to the dotted line will not be drawn.

### 8.6.2 Example

The polygon shown in Figure 8.2 is to be filled. Traversing the vertex list in counterclockwise order gives the following list of vertices. (4, 1) (2, 4) (3, 6) (1, 6) (2, 8) (5, 11) (8, 8) (13, 8) (11, 6) (11, 3) (10, 1).

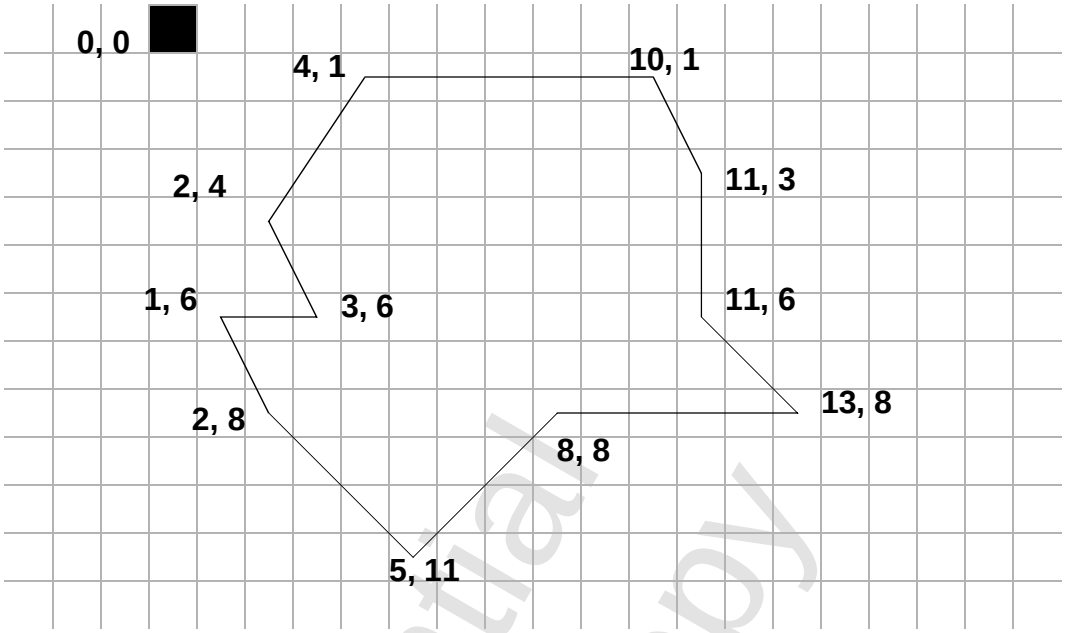



**Figure 8.2 Polygon to be Filled**

Figure 8.3 through Figure 8.14 illustrate the steps in drawing the polygon. Filled circles are vertices of the left polygon edge. Open circles are vertices on the right polygon edge. Pixels that have been drawn at the end of each step are shaded in the respective figures.
The polygon engine keeps track of four vertices at a time. The arrows in the figures indicate when a variable changes between the start of a step and the end of pixel filling for that step.

| Vertex | ID in Figures |
|---|---|
| top vertex of current left edge | L0 |
| bottom vertex of current left edge | L1 |
| top vertex of current right edge | R0 |
| bottom vertex of current right edge | r1 |

First determine the top vertex. This is the vertex with the lowest y coordinate. If multiple vertices share the lowest y coordinate, any such vertex may be used. The coordinates of this vertex should be written to the srcXY register. Now the command register can be written to put the engine into polygon fill mode. Bit 8 is set to '1', so that srcXY will be copied to dstXY.

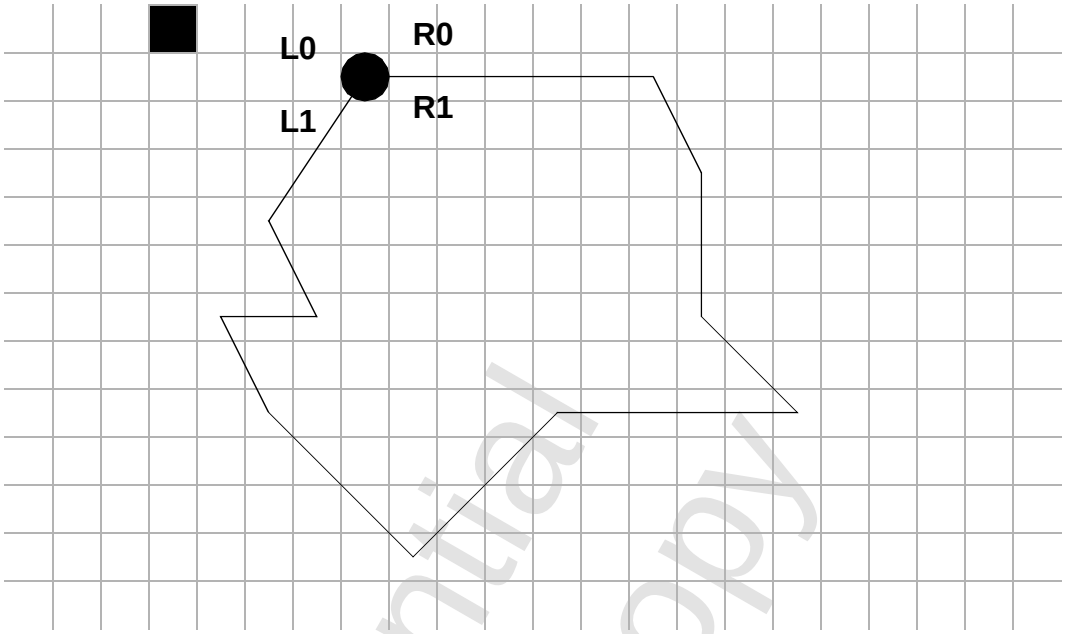



**Figure 8.3 scrXY (4, 1) and command Written**

Now the vertices should be written to the launch area in order of increasing y value. Whenever two vertices share the same y value, the leftmost vertex *must* be written first. The driver must keep track of the last y value sent for the left and right sides (note that this means the application must have sorted the vertices into right side/left side). If the y value for the last vertex sent for the left side is less than or equal the last y value sent for the right side, the next vertex for the left side must be written to the launch area next. Otherwise the next vertex for the right side must be written.

Since R1.y >= L1.y, write the next vertex for the left edge. That is (2, 4). No pixels are drawn yet.

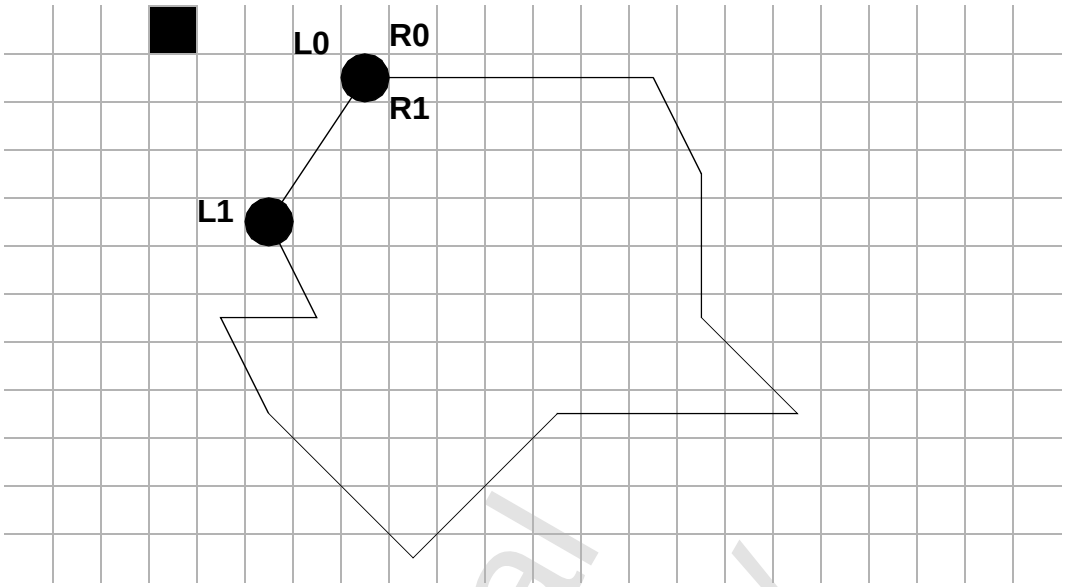



**Figure 8.4 Vertex (2, 4) Written**

R1.y < L1.y, so write the next vertex for the right side (10, 1). The drawing engine now has edges for both the right and left sides. It will draw all spans up to the minimum of R1.y, L1.y. Because R1.y = R0.y, no pixels are drawn, but R0 will be updated to vertex R1.

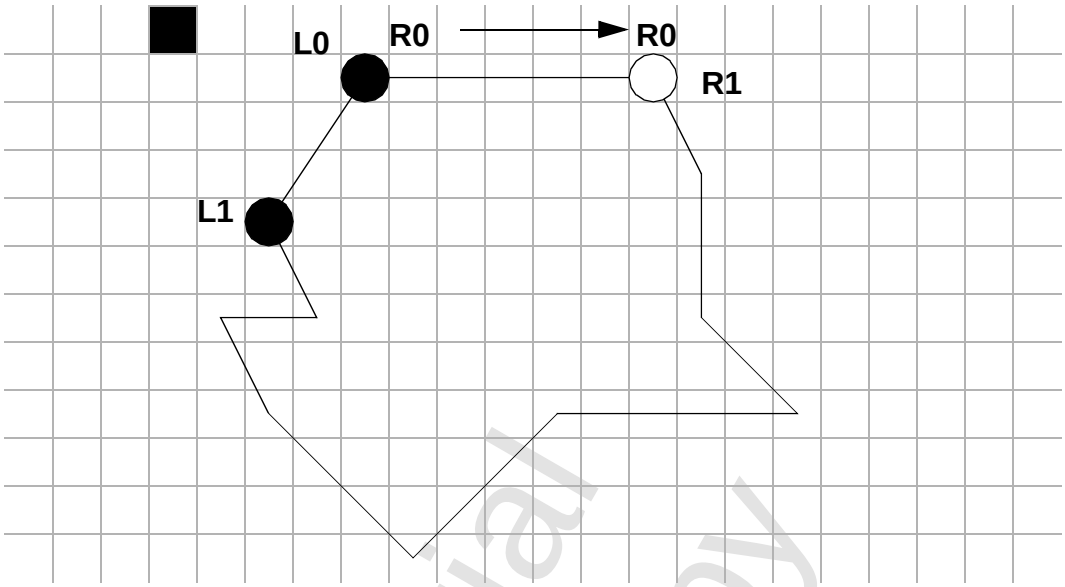



**Figure 8.5 Vertex (10, 1) Written**

R1.y < L1.y so write the next vertex for the right side (11,3). Pixels on all spans from the maximum of (L0.y, R0.y) to the minimum of (L1.y, R1.y) -1 will be drawn. Because R1.y < L1.y, R0 is updated to R1.

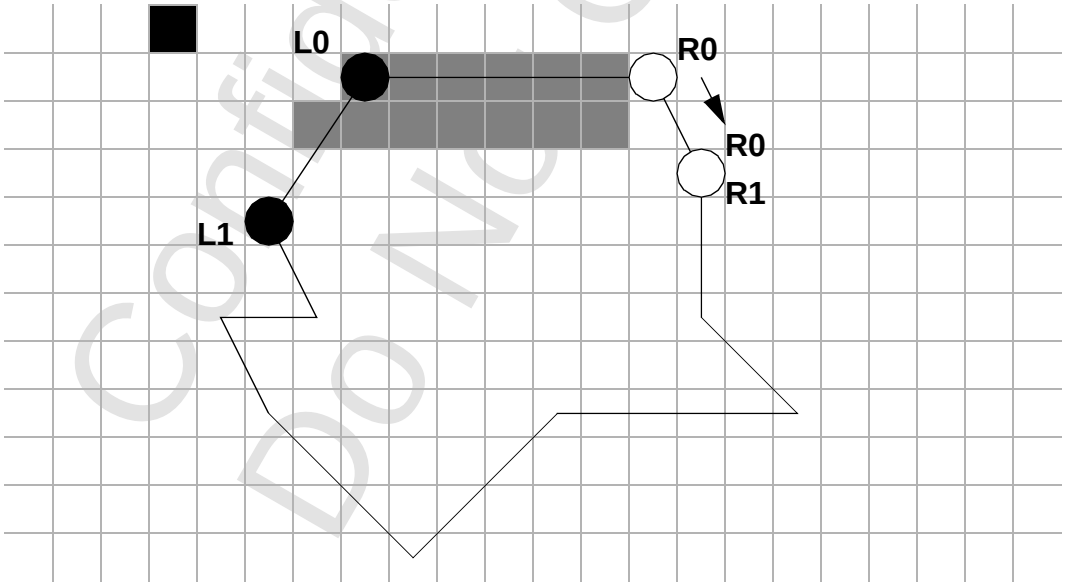



**Figure 8.6 Vertex (11,3) Written**

R1.y < L1.y, so write the next vertex on the right side (11, 6). Pixels are drawn on all spans from max (L0.y, R0.y) to min (L1.y, R1.y) -1. Since R1.y > L1.y, L0 is updated to L1.

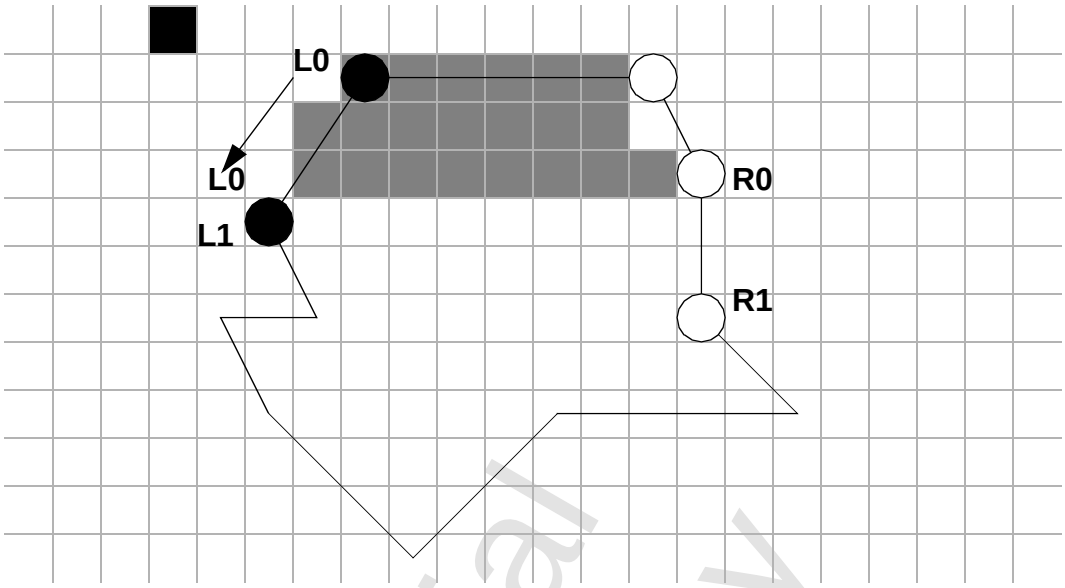



**Figure 8.7 Vertex (11,6) Written**

R1.y >= L1.y, so write the next vertex on the left side (3, 6). Pixels are drawn on all spans from max (L0.y, R0.y) to min (L1.y, R1.y) -1. R0 is updated to R1 and L0 is updated to L1.

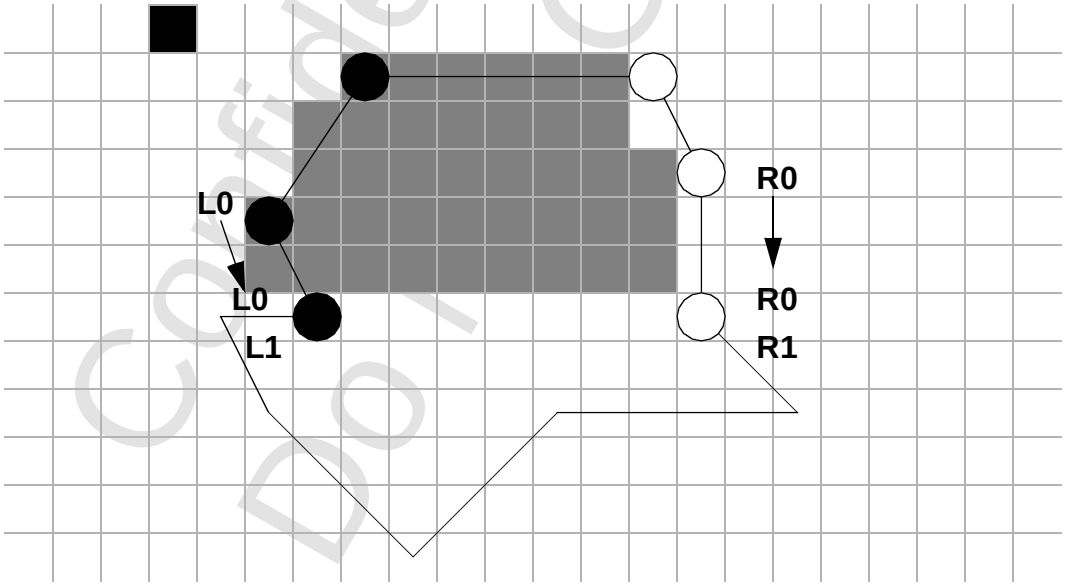



**Figure 8.8 Vertex (3, 6) Written**

R1.y >= L1.y, so write the next vertex on the left side (1, 6). L1.y = R1.y so Ro is updated to R1 (with no effect) and L0 is updated to L1.

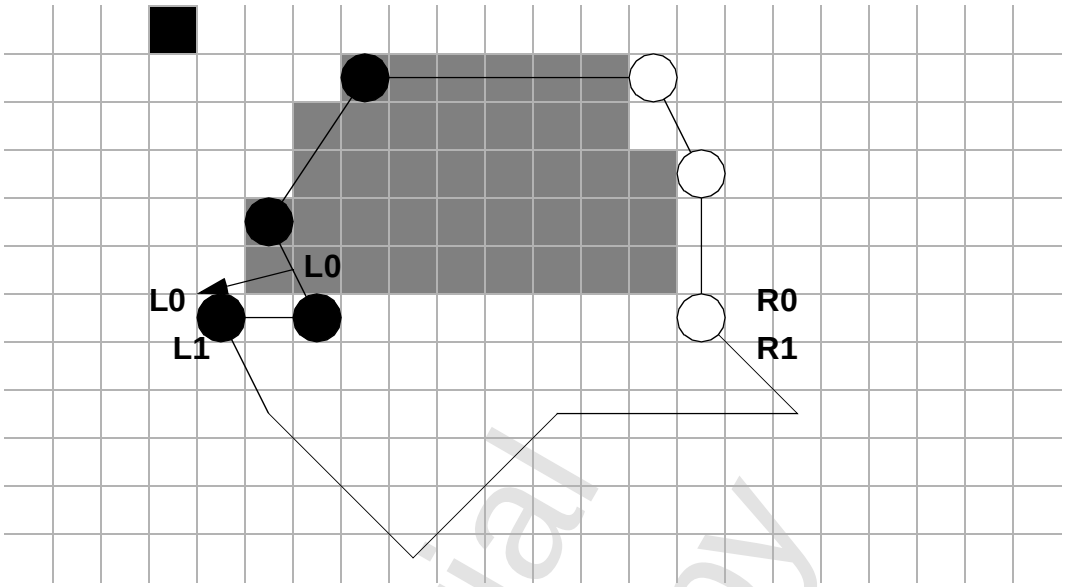



**Figure 8.9 Vertex (1, 6) Written**

R1.y >= L1.y so write the next vertex on the left edge (2,8). L1.y > R1.y, so R0 is updated to R1(with no effect).

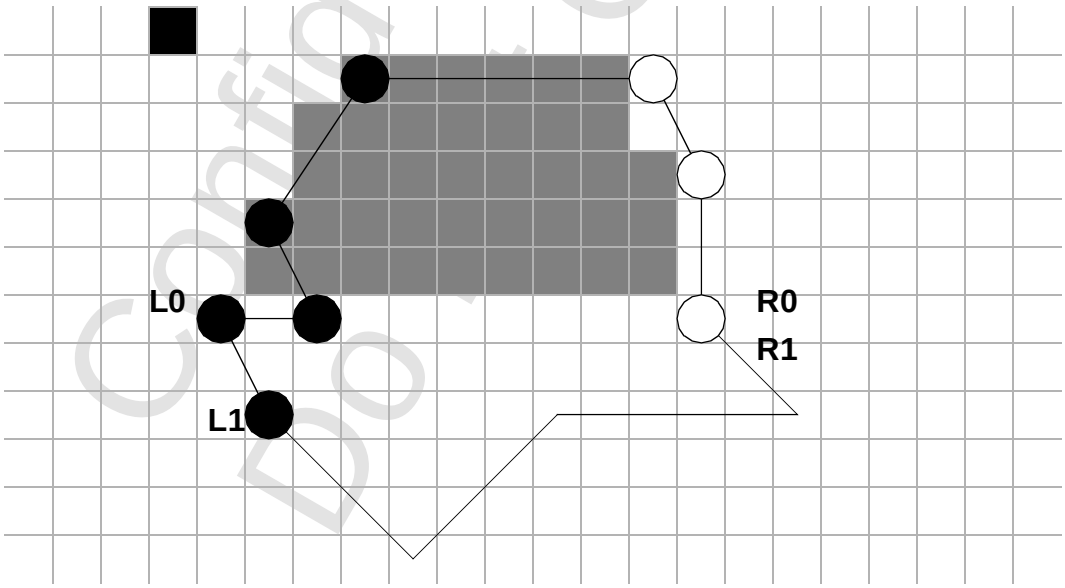



**Figure 8.10 Vertex (2, 8) Written**

R1.y < L1.y, so write the vertex on the right side (13, 8). L1.y = R1.y so R0 is updated to R1, and L0 is updated to L1. Pixels are written.

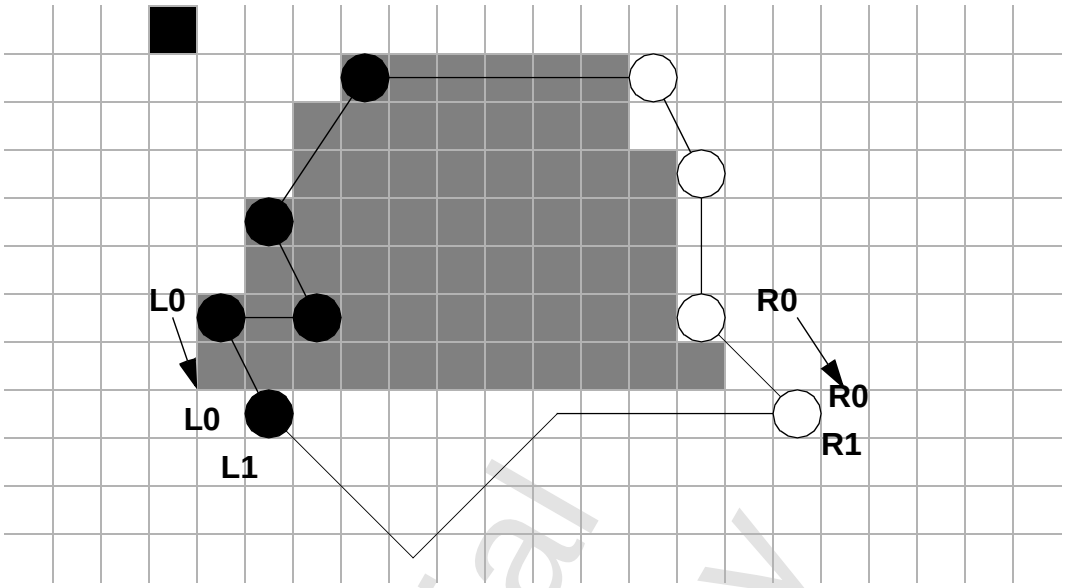



**Figure 8.11 Vertex (13, 8) Written**

R1.y >= L1.y, so write the next vertex on the left side (5, 11). L1.y > R1.y, so R0 is updated to R1 (without effect). No pixels are drawn.L

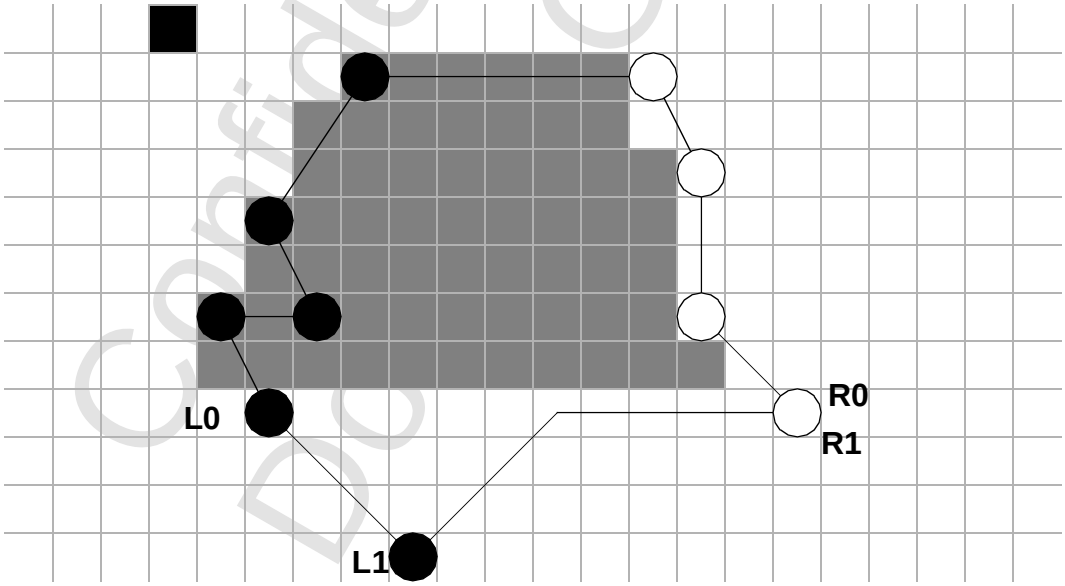



**Figure 8.12 Vertex (5, 11) Written**

R1.y < L1.y, so write the next vertex on the right side (8, 8). L1.y > R1.y, so R0 is updated to R1. No pixels are drawn.

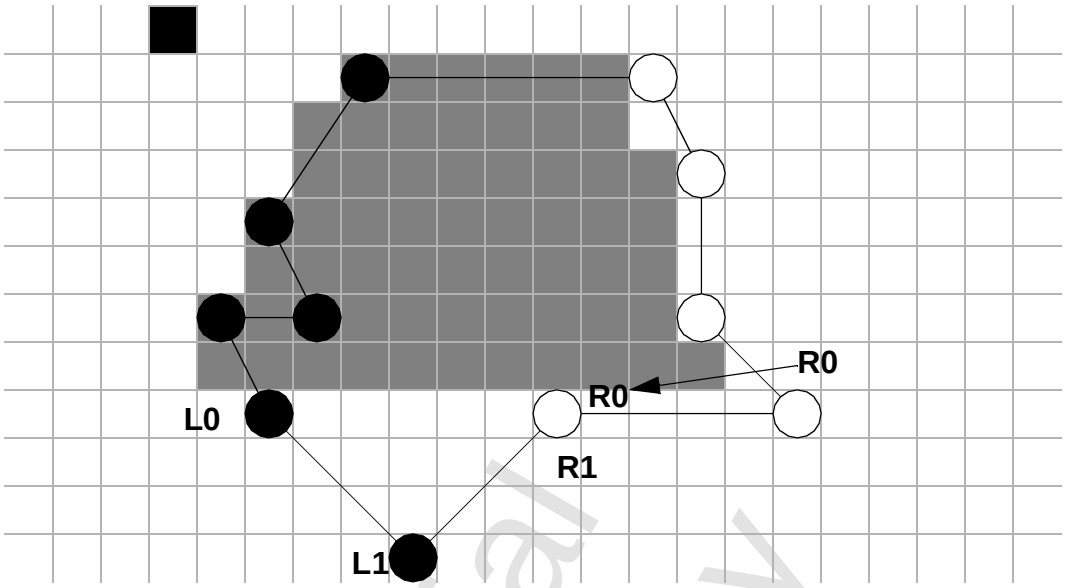



**Figure 8.13 Vertex (8, 8) Written**

R1.y < L1.y, so write the next vertex on the right side (5, 11). This is the final vertex for the polygon. Since the polygon does not have a flat bottom, the last vertex on the right side is the same as the last vertex on the left side. L1.y = R1.y so L0 is updated to L1 and R0 is updated to R1. No pixels on the final span are drawn (this would be true even if L1.y did not equal R1.y). If the launch area is written again before any registers are written, the polygon engine will begin a new polygon starting at (5, 11).

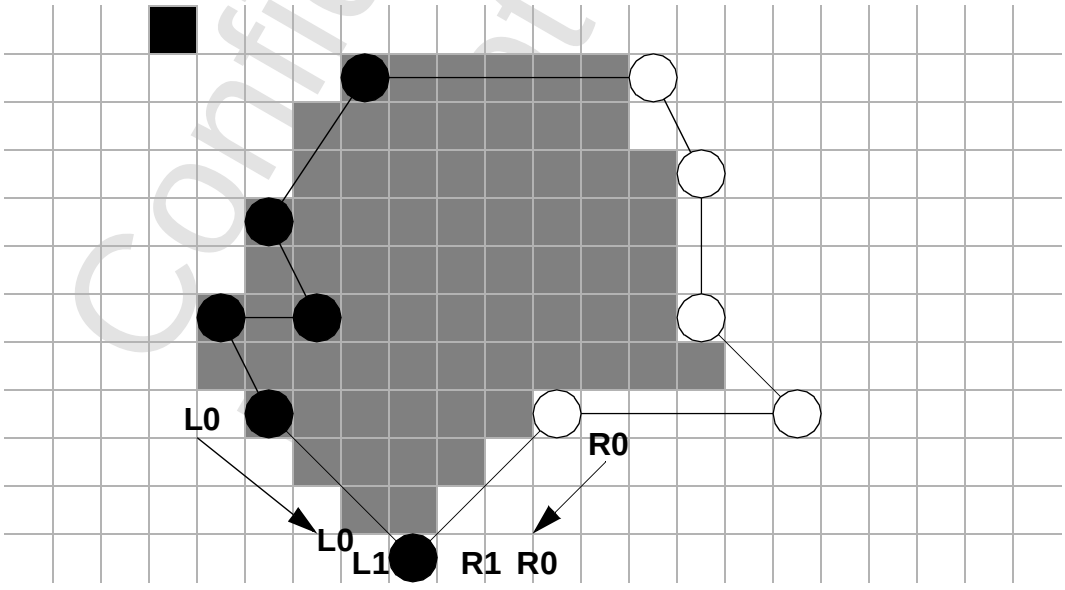



**Figure 8.14 Vertex (5, 11) Written (again)**

## 8.7 SGRAM Control

There are three commands that write the contents of scrBaseAddr to registers in the SGRAMs using special cycles generated by the memory controller. The 32 bits of scrBaseAddr are replicated across the four chips.

### 8.7.1 Write SGRAM Mode Register

Executing this command causes the value in scrBaseAddr[10:0] to be set into the SGRAM mode register.

| Bit | Description |
|---|---|
| 31:11 | **Reserved:** Reserved bits must be written as 0. |
| 10 | **SGRAM-Defined:** See the SGRAM documentation. |
| 9 | **Write Burst Length:** (0 = burst; 1 = single bit). See the SGRAM documentation. |
| 8:7 | **Test Mode:** See the SGRAM documentation. |
| 6:4 | **CAS Latency:** See the SGRAM documentation. |
| 3 | **Burst Type:** See the SGRAM documentation. 0 implies sequential, ‘1’ implies interleaved. |
| 2:0 | **Burst Length:** See the SGRAM documentation. |

### 8.7.2 Write SGRAM Color Register

Executing this command causes the value in srcBaseAddr[31:0] to be set into the SGRAM color register.

### 8.7.3 Write SGRAM Mask Register

Executing the command causes the value in srcBaseAddr[31:0] to be set into the SGRAM mask register.

# 9 3D Registers

## 9.1 Addressing

The 3D registers are accessible beginning at 0x020 0000 in the space claimed in PCI10. The address is used to transfer information besides just a register number; the address field is shown in Table 9.1.

### Table 9.1 3D Register Addressing

| Address Bits | Field Size | Description | Note |
|---|---|---|---|
| 1:0 | Two | Byte Offset | Must be 00b |
| 9:2 | Eight | Register ID | |
| 11:10 | Two | Chip Select | |
| 13:12 | Two | Reserved | Must be 00b |
| 19:14 | Six | Wrap | |
| 20 | One | Swizzle | |
| 21 | One | AltMap | |

The low order two bits must be zeroes; all accesses to memory mapped registers must be 32-bit accesses. No 8-, 16-, or 24-bit accesses are allowed.
Bits [9:2] specify the register within the chip or chips. In the tables and descriptions that follow, registers will be ordered according to this field.
Bits 11:10 specify the chip or chips to be accessed for writes. Each bit selects one chip for writing, as shown in Table 9.2. This is a straight-forward bit-sensitive encoding, except that the value 00b selects both chips, just as the value 11b does. Bits 12 and 13 must be programmed to 00b. Reads always access the FBI chip.

### Table 9.2 3D Register Chip Field Encoding

| Chip | TREX #0 | FBI | Note |
|---|---|---|---|
| 00b | X | X | Future Product |
| 01b | | X | |
| 10b | X | | |
| 11b | X | X | |

If address bit 21 is ‘1’ and if miscInit1[5] is ‘1’, alternate mapping (also called alternate addressing) is selected. In this case, the registers are addressed as shown in Table 9.4. With these addresses, the registers are ordered so that they can be written more efficiently. If address bit 21 is 0, normal addressing is selected. In this case, the registers are addressed as shown in Table 9.3.

## 9.2 3D Register Summary

Table 9.3 is a summary of the Voodoo3 3D memory mapped register set. The addresses are for the case when triangle registers address aliasing (alternate addressing) is not being used. The chip(s) column indicates which registers are stored in which chip(s). TRA indicates all TREX units, regardless of the chip address. TRI indicates only the individual TREX unit(s) specified in the chip address will be written. The access column indicates whether the register is read-only (R/O), read/write (R/W), or write-only (W/O). The sync column indicates whether the processor must wait for the current command to finish before loading the register from the FIFO. The FIFO column indicates whether a write to the registers will be pushed into the PCI bus FIFO. Care must be taken when writing to registers not pushed in order to prevent race conditions between the loading of registers that are pushed and those that are not. See, for example, the intrCtl register.

### Table 9.3 3D Register Summary

| Name | Address | Bits | Chips | Acc | Sync | FIFO | Description | Format | Link |
|---|---|---|---|---|---|---|---|---|---|
| status | 0x020 0000 | 31:0 | F | R/O | No | na | Voodoo3 status | - | Section 9.3.1 |
| intrCtrl | 0x020 0004 | 31:0 | F | R/W | No | No | Interrupt status and control | - | Section 9.3.2 |
| vertexAx | 0x020 0008 | 15:0 | F, TRA | W/O | No | Yes | Vertex A x-coordinate | 12.4 | Section 9.3.3 |
| vertexAy | 0x020 000C | 15:0 | F, TRA | W/O | No | Yes | Vertex A y-coordinate | 12.4 | Section 9.3.3 |
| vertexBx | 0x020 0010 | 15:0 | F, TRA | W/O | No | Yes | Vertex B x-coordinate | 12.4 | Section 9.3.3 |
| vertexBy | 0x020 0014 | 15:0 | F, TRA | W/O | No | Yes | Vertex B y-coordinate | 12.4 | Section 9.3.3 |
| vertexCx | 0x020 0018 | 15:0 | F, TRA | W/O | No | Yes | Vertex C x-coordinate | 12.4 | Section 9.3.3 |
| vertexCy | 0x020 001C | 15:0 | F, TRA | W/O | No | Yes | Vertex C y-coordinate | 12.4 | Section 9.3.3 |
| startR | 0x020 0020 | 23:0 | F | W/O | No | Yes | Starting Red parameter | 12.12 | Section 9.3.5 |
| startG | 0x020 0024 | 23:0 | F | W/O | No | Yes | Starting Green parameter | 12.12 | Section 9.3.5 |
| startB | 0x020 0028 | 23:0 | F | W/O | No | Yes | Starting Blue parameter | 12.12 | Section 9.3.5 |
| startZ | 0x020 002C | 23:0 | F | W/O | No | Yes | Starting Z parameter | 12.12 | Section 9.3.7 |
| startA | 0x020 0030 | 23:0 | F | W/O | No | Yes | Starting Alpha parameter | 12.12 | Section 9.3.5 |
| startS | 0x020 0034 | 31:0 | TRA | W/O | No | Yes | Starting S/W parameter | 14.18 | Section 9.3.9 |
| startT | 0x020 0038 | 31:0 | TRI | W/O | No | Yes | Starting T/W parameter | 14.18 | Section 9.3.11 |

**Table 9.3 3D Register Summary *(cont.)***

| Name | Address | Bits | Chips | Acc | Sync | FIFO | Description | Format | Link |
|---|---|---|---|---|---|---|---|---|---|
| startW | 0x020 003C | 31:0 | F, TRI | W/O | No | Yes | Starting 1/W parameter | 2.30 | Section 9.3.13 |
| dRdX | 0x020 0040 | 23:0 | F | W/O | No | Yes | Change in Red wrt[a] X | 12.12 | Section 9.3.15 |
| dGdX | 0x020 0044 | 23:0 | F | W/O | No | Yes | Change in Green wrt X | 12.12 | Section 9.3.15 |
| dBdX | 0x020 0048 | 23:0 | F | W/O | No | Yes | Change in Blue wrt X | 12.12 | Section 9.3.15 |
| dZdX | 0x020 004C | 31:0 | F | W/O | No | Yes | Change in Z wrt X | 20.12 | Section 9.3.17 |
| dAdX | 0x020 0050 | 23:0 | F | W/O | No | Yes | Change in Alpha wrt X | 12.12 | Section 9.3.15 |
| dSdX | 0x020 0054 | 31:0 | TRI | W/O | No | Yes | Change in S/W wrt X | 14.18 | Section 9.3.19 |
| dTdX | 0x020 0058 | 31:0 | TRI | W/O | No | Yes | Change in T/W wrt X | 14.18 | Section 9.3.19 |
| dWdX | 0x020 005C | 31:0 | F, TRI | W/O | No | Yes | Change in 1/W wrt X | 2.30 | Section 9.3.21 |
| dRdY | 0x020 0060 | 23:0 | F | W/O | No | Yes | Change in Red wrt Y | 12.12 | Section 9.3.15 |
| dGdY | 0x020 0064 | 23:0 | F | W/O | No | Yes | Change in Green wrt Y | 12.12 | Section 9.3.15 |
| dBdY | 0x020 0068 | 23:0 | F | W/O | No | Yes | Change in Blue wrt Y | 12.12 | Section 9.3.15 |
| dZdY | 0x020 006C | 31:0 | F | W/O | No | Yes | Change in Z wrt Y | 20.12 | Section 9.3.17 |
| dAdY | 0x020 0070 | 23:0 | F | W/O | No | Yes | Change in Alpha wrt Y | 12.12 | Section 9.3.15 |
| dSdY | 0x020 0074 | 31:0 | TI | W/O | No | Yes | Change in S/W wrt Y | 14.18 | Section 9.3.19 |
| dTdY | 0x020 0078 | 31:0 | TI | W/O | No | Yes | Change in T/W wrt Y | 14.18 | Section 9.3.19 |
| dWdY | 0x020 007C | 31:0 | F, TRI | W/O | No | Yes | Change in 1/W wrt Y | 2.30 | Section 9.3.21 |
| triangleCMD | 0x020 0080 | 31 | F, TRA | W/O | No | Yes | Render Triangle Command | - | Chapter 9.3.23 |
| reserved | 0x020 0084 | - | - | - | - | - | - | - | |
| fvertexAx | 0x020 0088 | 31:0 | F, TRA | W/O | No | Yes | Vertex A x-coordinate | floating | Section 9.3.4 |

**Table 9.3 3D Register Summary *(cont.)***

| Name | Address | Bits | Chips | Acc | Sync | FIFO | Description | Format | Link |
|---|---|---|---|---|---|---|---|---|---|
| fvertexAy | 0x020 008C | 31:0 | F, TRA | W/O | No | Yes | Vertex A y-coordinate | floating | Section 9.3.4 |
| fvertexBx | 0x020 0090 | 31:0 | F, TRA | W/O | No | Yes | Vertex B x-coordinate | floating | Section 9.3.4 |
| fvertexBy | 0x020 0094 | 31:0 | F, TRA | W/O | No | Yes | Vertex B y-coordinate | floating | Section 9.3.4 |
| fvertexCx | 0x020 0098 | 31:0 | F, TRA | W/O | No | Yes | Vertex C x-coordinate | floating | Section 9.3.4 |
| fvertexCy | 0x020 009C | 31:0 | F, TRA | W/O | No | Yes | Vertex C y-coordinate | floating | Section 9.3.4 |
| fstartR | 0x020 00A0 | 31:0 | F | W/O | No | Yes | Starting Red parameter | floating | Section 9.3.6 |
| fstartG | 0x020 00A4 | 31:0 | F | W/O | No | Yes | Starting Green parameter | floating | Section 9.3.6 |
| fstartB | 0x020 00A8 | 31:0 | F | W/O | No | Yes | Starting Blue parameter | floating | Section 9.3.6 |
| fstartZ | 0x020 00AC | 31:0 | F | W/O | No | Yes | Starting Z parameter | floating | Section 9.3.8 |
| fstartA | 0x020 00B0 | 31:0 | F | W/O | No | Yes | Starting Alpha parameter | floating | Section 9.3.6 |
| fstartS | 0x020 00B4 | 31:0 | TRI | W/O | No | Yes | Starting S/W parameter | floating | Section 9.3.9 |
| fstartT | 0x020 00B8 | 31:0 | TRI | W/O | No | Yes | Starting T/W parameter | floating | Section 9.3.12 |
| fstartW | 0x020 00BC | 31:0 | F, TRI | W/O | No | Yes | Starting 1/W parameter | floating | Section 9.3.14 |
| fdRdX | 0x020 00C0 | 31:0 | F | W/O | No | Yes | Change in Red wrt X | floating | Section 9.3.16 |
| fdGdX | 0x020 00C4 | 31:0 | F | W/O | No | Yes | Change in Green wrt X | floating | Section 9.3.16 |
| fdBdX | 0x020 00C8 | 31:0 | F | W/O | No | Yes | Change in Blue wrt X | floating | Section 9.3.16 |
| fdZdX | 0x020 00CC | 31:0 | F | W/O | No | Yes | Change in Z wrt X | floating | Section 9.3.18 |
| fdAdX | 0x020 00D0 | 31:0 | F | W/O | No | Yes | Change in Alpha wrt X | floating | Section 9.3.16 |
| fdSdX | 0x020 00D4 | 31:0 | TRI | W/O | No | Yes | Change in S/W wrt X | floating | Section 9.3.20 |
| fdTdX | 0x020 00D8 | 31:0 | TRI | W/O | No | Yes | Change in T/W wrt X | floating | Section 9.3.20 |

**Table 9.3 3D Register Summary *(cont.)***

| Name | Address | Bits | Chips | Acc | Sync | FIFO | Description | Format | Link |
|---|---|---|---|---|---|---|---|---|---|
| fdWdX | 0x020 00DC | 31:0 | F, TRI | W/O | No | Yes | Change in 1/W wrt X | floating | Section 9.3.22 |
| fdRdY | 0x020 00E0 | 31:0 | F | W/O | No | Yes | Change in Red wrt Y | floating | Section 9.3.16 |
| fdGdY | 0x020 00E4 | 31:0 | F | W/O | No | Yes | Change in Green wrt Y | floating | Section 9.3.16 |
| fdBdY | 0x020 00E8 | 31:0 | F | W/O | No | Yes | Change in Blue wrt Y | floating | Section 9.3.16 |
| fdZdY | 0x020 00EC | 31:0 | F | W/O | No | Yes | Change in Z wrt Y | floating | Section 9.3.18 |
| fdAdY | 0x020 00F0 | 31:0 | F | W/O | No | Yes | Change in Alpha wrt Y | floating | Section 9.3.16 |
| fdSdY | 0x020 00F4 | 31:0 | TRI | W/O | No | Yes | Change in S/W wrt Y | floating | Section 9.3.20 |
| fdTdY | 0x020 00F8 | 31:0 | TRI | W/O | No | Yes | Change in T/W wrt Y | floating | Section 9.3.20 |
| fdWdY | 0x020 00FC | 31:0 | F, TRI | W/O | No | Yes | Change in 1/W wrt Y | floating | Section 9.3.22 |
| ftriangleCmd | 0x020 0100 | 31 | F, TRA | W/O | No | Yes | Execute triangle command | - | Section 9.3.23 |
| fbzColorPath | 0x020 0104 | 27:0 | F, TRA | R/W | No | Yes | FBI color path control | - | Section 9.3.27 |
| fogMode | 0x020 0108 | 5:0 | F | R/W | No | Yes | Fog mode control | - | Section 9.3.28 |
| alphaMode | 0x020 010C | 31:0 | F | R/W | No | Yes | Alpha mode control | - | Section 9.3.29 |
| fbzMode | 0x020 0110 | 20:0 | F | R/W | Yes | Yes | RGB and Z-buffer control | - | Section 9.3.31 |
| lfbMode | 0x020 0114 | 16:0 | F | R/W | Yes | Yes | Linear frame buffer control | - | Section 9.3.30 |
| clipLeftRight | 0x020 0118 | 31:0 | F | R/W | Yes | Yes | Left/Right clipping register | - | Section 9.3.43 |
| clipLowYHighY | 0x020 011C | 31:0 | F | R/W | Yes | Yes | Top/Bottom clipping register | - | Section 9.3.44 |
| nopCMD | 0x020 0120 | 0 | F, TRA | W/O | Yes | Yes | Execute NOP command | - | Section 9.3.24 |
| fastfillCMD | 0x020 0124 | na | F | W/O | Yes | Yes | Execute FASTFILL cmnd | - | Section 9.3.25 |
| swapbufferCMD | 0x020 0128 | 8:0 | F | W/O | Yes | Yes | Execute SWAPBUFFER | - | Section 9.3.26 |

**Table 9.3 3D Register Summary *(cont.)***

| Name | Address | Bits | Chips | Acc | Sync | FIFO | Description | Format | Link |
|---|---|---|---|---|---|---|---|---|---|
| fogColor | 0x020 012C | 23:0 | F | W/O | Yes | Yes | Fog color value | | Section 9.3.34 |
| zaColor | 0x020 0130 | 31:0 | F | W/O | Yes | Yes | Constant Alpha / Depth | | Section 9.3.35 |
| chromaKey | 0x020 0134 | 23:0 | F | W/O | Yes | Yes | ChromaKey compare value | | Section 9.3.36 |
| chromaRange | 0x020 0138 | 27:0 | F | W/O | Yes | Yes | Chroma range compare | | Section 9.3.37 |
| userIntrcMD | 0x020 013C | 9:0 | F | W/O | Yes | Yes | Execute USERINTERRUPT | | Section 9.3.38 |
| stipple | 0x020 0140 | 31:0 | F | R/W | Yes | Yes | Rendering stipple value | | Section 9.3.32 |
| color0 | 0x020 0144 | 31:0 | F | R/W | Yes | Yes | Constant color 0 | | Section 9.3.33 |
| color1 | 0x020 0148 | 31:0 | F | R/W | Yes | Yes | Constant color 1 | | Section 9.3.33 |
| fbiPixelsIn | 0x020 014C | 23:0 | F | R/O | na | na | Pixels processed | | Section 9.3.46 |
| fbiChromaFail | 0x020 0150 | 23:0 | F | R/O | na | na | Pixels failed Chroma test | | Section 9.3.46 |
| fbiZfuncFail | 0x020 0154 | 23:0 | F | R/O | na | na | Pixels failed Z-test | | Section 9.3.46 |
| fbiAfuncFail | 0x020 0158 | 23:0 | F | R/O | na | na | Pixels failed Alpha test | | Section 9.3.46 |
| fbiPixelsOut | 0x020 015C | 23:0 | F | R/O | na | na | Pixels drawn | | Section 9.3.46 |
| fogTable | 0x020 0160<br>0x020 01DC | 31:0 | F | W/O | Yes | Yes | Fog Table | | Section 9.3.45 |
| reserved | 0x020 01E0 | - | - | - | - | - | - | - | |
| reserved | 0x020 01E4 | - | - | - | - | - | - | - | |
| reserved | 0x020 01E8 | - | - | - | - | - | - | - | |
| colBufferAddr | 0x020 01EC | 23:0 | F | R/W | Yes | Yes | Color buffer base address | | Section 9.3.39 |
| colBufferStride | 0x020 01F0 | 23:0 | F | R/W | Yes | Yes | Color buffer stride, type | | Section 9.3.40 |
| auxBufferAddr | 0x020 01F4 | 23:0 | F | R/W | Yes | Yes | Aux buffer base address | | Section 9.3.41 |

**Table 9.3 3D Register Summary *(cont.)***

| Name | Address | Bits | Chips | Acc | Sync | FIFO | Description | Format | Link |
|---|---|---|---|---|---|---|---|---|---|
| auxBufferStride | 0x020 01F8 | 23:0 | F | R/W | Yes | Yes | Aux buffer stride, type | | Section 9.3.42 |
| reserved | 0x020 01FC | - | - | - | - | - | - | - | |
| clipLeftRight1 | 0x020 0200 | 31:0 | F | R/W | Yes | Yes | Secondary L/R clipping reg | | Section 9.3.47 |
| clipTopBottom1 | 0x020 0204 | 31:0 | F | F/W | Yes | Yes | Secondary T/B clipping reg | | Section 9.3.48 |
| reserved | 0x020 0208<br>0x020 024B | - | - | - | - | - | - | - | |
| swapPending | 0x020 024C | na | F | W/O | No | No | Swap buffer pending | | Section 9.3.49 |
| leftOverlayBuf | 0x020 0250 | 27:0 | F | W/O | No | Yes | Left overlay address | | Section 9.3.50 |
| rightOverlayBuf | 0x020 0254 | 27:0 | F | W/O | No | Yes | Right overlay address | | Section 9.3.51 |
| fbiSwapHistory | 0x020 0258 | 31:0 | F | R/O | na | na | Swap History Buffer | | Section 9.3.52 |
| fbiTrianglesOut | 0x020 025C | 23:0 | F | R/O | na | na | Triangles drawn | | Section 9.3.53 |
| sSetupMode | 0x020 0260 | 19:0 | F | W/O | No | Yes | Triangle setup mode | | Section 9.3.54 |
| sVx | 0x020 0264 | 31:0 | F, TMU | W/O | No | Yes | Triangle setup X | | Section 9.3.55 |
| sVy | 0x020 0268 | 31:0 | F, TMU | W/O | No | Yes | Triangle setup Y | | Section 9.3.55 |
| sARGB | 0x020 026C | 31:0 | F, TMU | W/O | No | Yes | Triangle setup ARGB | | Section 9.3.55 |
| sRed | 0x020 0270 | 31:0 | F | W/O | No | Yes | Triangle setup Red value | | Section 9.3.55 |
| sGreen | 0x020 0274 | 31:0 | F | W/O | No | Yes | Triangle setup Green value | | Section 9.3.55 |
| sBlue | 0x020 0278 | 31:0 | F | W/O | No | Yes | Triangle setup Blue value | | Section 9.3.55 |
| sAlpha | 0x020 027C | 31:0 | F | W/O | No | Yes | Triangle setup Alpha value | | Section 9.3.55 |
| sVz | 0x020 0280 | 31:0 | F | W/O | No | Yes | Triangle setup Z | | Section 9.3.55 |
| sWb | 0x020 0284 | 31:0 | F, TMU | W/O | No | Yes | Triangle setup Global W | | Section 9.3.55 |

**Table 9.3 3D Register Summary *(cont.)***

| Name | Address | Bits | Chips | Acc | Sync | FIFO | Description | Format | Link |
|---|---|---|---|---|---|---|---|---|---|
| sWtmu0 | 0x020 0288 | 31:0 | TMU | W/O | No | Yes | Triangle setup Tmu 1/W | | Section 9.3.55 |
| sS/W0 | 0x020 028C | 31:0 | TMU | W/O | No | Yes | Triangle setup Tmu S/W | | Section 9.3.55 |
| sT/W0 | 0x020 0290 | 31:0 | TMU | W/O | No | Yes | Triangle setup Tmu T/W | | Section 9.3.55 |
| sWtmu1 | 0x020 0294 | 31:0 | TMU1 | W/O | No | Yes | Triangle setup Tmu1 W | | Section 9.3.55 |
| sS/Wtmu1 | 0x020 0298 | 31:0 | F | W/O | No | Yes | Triangle setup Tmu1 S/W | | Section 9.3.55 |
| sT/Wtmu1 | 0x020 029C | 31:0 | F | W/O | No | Yes | Triangle setup Tmu1 T/W | | Section 9.3.55 |
| sDrawTriCMD | 0x020 02A0 | 31:0 | F, TMU | W/O | No | Yes | Triangle setup (Draw) | | Section 9.3.56 |
| sBeginTriDMC | 0x020 02A4 | 31:0 | F | W/O | No | Yes | Triangle setup Start New Tri | | Section 9.3.57 |
| reserved | 0x020 02A8<br>0x020 02FC | - | - | - | - | - | - | - | - |
| textureMode | 0x020 0300 | 30:0 | TRI | W/O | No | Yes | Texture mode control | | Section 9.3.58 |
| tLOD | 0x020 0304 | 23:0 | TRI | W/O | No | Yes | Texture level of detail | | Section 9.3.59 |
| tDetail | 0x020 0308 | 21:0 | TRI | W/O | No | Yes | Texture detail | | Section 9.3.60 |
| texBaseAddr | 0x020 030C | 31:0 | TRI | W/O | No | Yes | Texture base address | | Section 9.3.61 |
| texBaseAddr_1 | 0x020 0310 | 23:0 | TRI | W/O | No | Yes | Texture base addr (LOD1) | | Section 9.3.61 |
| texBaseAddr_2 | 0x020 0314 | 23:0 | TRI | W/O | No | Yes | Texture base addr (LOD2) | | Section 9.3.61 |
| texBaseAdr3_8 | 0x020 0318 | 23:0 | TRI | W/O | No | Yes | Texture base addr (LOD3-8) | | Section 9.3.61 |
| reserved | 0x020 031C | - | - | - | - | - | - | - | - |
| texInit1 | 0x020 0320 | 31:0 | TRI | W/O | Yes | Yes | TREX Hardware Init Reg 1 | | Section 9.3.62 |
| nccTable0 | 0x020 0324<br>0x020 0350 | 31:0<br>26:0 | TRI | W/O | Yes | Yes | Narrow channel comp tab 0 | | Section 9.3.63 |

**Table 9.3 3D Register Summary *(cont.)***

| Name | Address | Bits | Chips | Acc | Sync | FIFO | Description | Format | Link |
|---|---|---|---|---|---|---|---|---|---|
| nccTable1 | 0x020 0354<br>0x020 0380 | 31:0<br>26:0 | TRI | W/O | Yes | Yes | Narrow channel comp tab 1 | | Section 9.3.63 |
| reserved | 0x020 0384<br>0x020 03FC | - | - | - | - | - | - | - | - |

a. wrt: with respect to

When PCI_AD[21] is 1, the triangle parameter registers are aliased to different addresses to improve PCI throughput. The addresses are shown in Table 9.4.

**Table 9.4 3D Register Summary (Alternate Mapping)**

| Name | Address | Bits | Chips | Acc | Sync | FIFO | Description | Format | Link |
|---|---|---|---|---|---|---|---|---|---|
| status | 0x030 0000 | 31:0 | F | R/O | No | na | Voodoo3 status | - | Section 9.3.1 |
| intrCtrl | 0x030 0004 | 31:0 | F | R/W | No | No | Interrupt status and control | - | Section 9.3.2 |
| vertexAx | 0x030 0008 | 15:0 | F, TRA | W/O | No | Yes | Vertex A x-coordinate | 12.4 | Section 9.3.3 |
| vertexAy | 0x030 000C | 15:0 | F, TRA | W/O | No | Yes | Vertex A y-coordinate | 12.4 | Section 9.3.3 |
| vertexBx | 0x030 0010 | 15:0 | F, TRA | W/O | No | Yes | Vertex B x-coordinate | 12.4 | Section 9.3.3 |
| vertexBy | 0x030 0014 | 15:0 | F, TRA | W/O | No | Yes | Vertex B y-coordinate | 12.4 | Section 9.3.3 |
| vertexCx | 0x030 0018 | 15:0 | F, TRA | W/O | No | Yes | Vertex C x-coordinate | 12.4 | Section 9.3.3 |
| vertexCy | 0x030 001C | 15:0 | F, TRA | W/O | No | Yes | Vertex C y-coordinate | 12.4 | Section 9.3.3 |
| startR | 0x030 0020 | 23:0 | F | W/O | No | Yes | Starting Red parameter | 12.12 | Section 9.3.5 |
| dRdX | 0x030 0024 | 23:0 | F | W/O | No | Yes | Change in Red wrt[a] X | 12.12 | Section 9.3.15 |
| dRdY | 0x030 0028 | 23:0 | F | W/O | No | Yes | Change in Red wrt Y | 12.12 | Section 9.3.15 |
| startG | 0x030 002C | 23:0 | F | W/O | No | Yes | Starting Green parameter | 12.12 | Section 9.3.5 |

**Table 9.4 3D Register Summary (Alternate Mapping) *(cont.)***

| Name | Address | Bits | Chips | Acc | Sync | FIFO | Description | Format | Link |
|---|---|---|---|---|---|---|---|---|---|
| dGdX | 0x030 0030 | 23:0 | F | W/O | No | Yes | Change in Green wrt X | 12.12 | Section 9.3.15 |
| dGdY | 0x030 0034 | 23:0 | F | W/O | No | Yes | Change in Green wrt Y | 12.12 | Section 9.3.15 |
| startB | 0x030 0038 | 23:0 | F | W/O | No | Yes | Starting Blue parameter | 12.12 | Section 9.3.5 |
| dBdX | 0x030 003C | 23:0 | F | W/O | No | Yes | Change in Blue wrt X | 12.12 | Section 9.3.15 |
| dBdY | 0x030 0040 | 23:0 | F | W/O | No | Yes | Change in Blue wrt Y | 12.12 | Section 9.3.15 |
| startZ | 0x030 0044 | 23:0 | F | W/O | No | Yes | Starting Z parameter | 12.12 | Section 9.3.7 |
| dZdX | 0x030 0048 | 31:0 | F | W/O | No | Yes | Change in Z wrt X | 20.12 | Section 9.3.17 |
| dZdY | 0x030 004C | 31:0 | F | W/O | No | Yes | Change in Z wrt Y | 20.12 | Section 9.3.17 |
| startA | 0x030 0050 | 23:0 | F | W/O | No | Yes | Starting Alpha parameter | 12.12 | Section 9.3.5 |
| dAdX | 0x030 0054 | 23:0 | F | W/O | No | Yes | Change in Alpha wrt X | 12.12 | Section 9.3.15 |
| dAdY | 0x030 0058 | 23:0 | F | W/O | No | Yes | Change in Alpha wrt Y | 12.12 | Section 9.3.15 |
| startS | 0x030 005C | 31:0 | TRA | W/O | No | Yes | Starting S/W parameter | 14.18 | Section 9.3.9 |
| dSdX | 0x030 0060 | 31:0 | TRI | W/O | No | Yes | Change in S/W wrt X | 14.18 | Section 9.3.19 |
| dSdY | 0x030 0064 | 31:0 | TI | W/O | No | Yes | Change in S/W wrt Y | 14.18 | Section 9.3.19 |
| startT | 0x030 0068 | 31:0 | TRI | W/O | No | Yes | Starting T/W parameter | 14.18 | Section 9.3.9 |
| dTdX | 0x030 006C | 31:0 | TRI | W/O | No | Yes | Change in T/W wrt X | 14.18 | Section 9.3.19 |
| dTdY | 0x030 0070 | 31:0 | TI | W/O | No | Yes | Change in T/W wrt Y | 14.18 | Section 9.3.19 |
| startW | 0x030 0074 | 31:0 | F, TRI | W/O | No | Yes | Starting 1/W parameter | 2.30 | Section 9.3.13 |
| dWdX | 0x030 0078 | 31:0 | F, TRI | W/O | No | Yes | Change in 1/W wrt X | 2.30 | Section 9.3.21 |
| dWdY | 0x030 007C | 31:0 | F, TRI | W/O | No | Yes | Change in 1/W wrt Y | 2.30 | Section 9.3.21 |

**Table 9.4 3D Register Summary (Alternate Mapping) *(cont.)***

| Name | Address | Bits | Chips | Acc | Sync | FIFO | Description | Format | Link |
|---|---|---|---|---|---|---|---|---|---|
| triangleCMD | 0x030 0080 | 31 | F, TRA | W/O | No | Yes | Render Triangle Command | - | Chapter 9.3.23 |
| reserved | 0x030 0084 | - | - | - | - | - | - | - | - |
| fvertexAx | 0x030 0088 | 31:0 | F, TRA | W/O | No | Yes | Vertex A x-coordinate | floating | Section 9.3.4 |
| fvertexAy | 0x030 008C | 31:0 | F, TRA | W/O | No | Yes | Vertex A y-coordinate | floating | Section 9.3.4 |
| fvertexBx | 0x030 0090 | 31:0 | F, TRA | W/O | No | Yes | Vertex B x-coordinate | floating | Section 9.3.4 |
| fvertexBy | 0x030 0094 | 31:0 | F, TRA | W/O | No | Yes | Vertex B y-coordinate | floating | Section 9.3.4 |
| fvertexCx | 0x030 0098 | 31:0 | F, TRA | W/O | No | Yes | Vertex C x-coordinate | floating | Section 9.3.4 |
| fvertexCy | 0x030 009C | 31:0 | F, TRA | W/O | No | Yes | Vertex C y-coordinate | floating | Section 9.3.4 |
| fstartR | 0x030 00A0 | 31:0 | F | W/O | No | Yes | Starting Red parameter | floating | Section 9.3.6 |
| fdRdX | 0x030 00A4 | 31:0 | F | W/O | No | Yes | Change in Red wrt X | floating | Section 9.3.16 |
| fdRdY | 0x030 00A8 | 31:0 | F | W/O | No | Yes | Change in Red wrt Y | floating | Section 9.3.16 |
| fstartG | 0x030 00AC | 31:0 | F | W/O | No | Yes | Starting Green parameter | floating | Section 9.3.6 |
| fdGdX | 0x030 00B0 | 31:0 | F | W/O | No | Yes | Change in Green wrt X | floating | Section 9.3.16 |
| fdGdY | 0x030 00B4 | 31:0 | F | W/O | No | Yes | Change in Green wrt Y | floating | Section 9.3.16 |
| fstartB | 0x030 00B8 | 31:0 | F | W/O | No | Yes | Starting Blue parameter | floating | Section 9.3.6 |
| fdBdX | 0x030 00BC | 31:0 | F | W/O | No | Yes | Change in Blue wrt X | floating | Section 9.3.16 |
| fdBdY | 0x030 00C0 | 31:0 | F | W/O | No | Yes | Change in Blue wrt Y | floating | Section 9.3.16 |
| fstartZ | 0x030 00C4 | 31:0 | F | W/O | No | Yes | Starting Z parameter | floating | Section 9.3.8 |
| fdZdX | 0x030 00C8 | 31:0 | F | W/O | No | Yes | Change in Z wrt X | floating | Section 9.3.18 |
| fdZdY | 0x030 00CC | 31:0 | F | W/O | No | Yes | Change in Z wrt Y | floating | Section 9.3.18 |

**Table 9.4 3D Register Summary (Alternate Mapping) *(cont.)***

| Name | Address | Bits | Chips | Acc | Sync | FIFO | Description | Format | Link |
|---|---|---|---|---|---|---|---|---|---|
| fstartA | 0x030 00D0 | 31:0 | F | W/O | No | Yes | Starting Alpha parameter | floating | Section 9.3.6 |
| fdAdX | 0x030 00D4 | 31:0 | F | W/O | No | Yes | Change in Alpha wrt X | floating | Section 9.3.16 |
| fdAdY | 0x030 00D8 | 31:0 | F | W/O | No | Yes | Change in Alpha wrt Y | floating | Section 9.3.16 |
| fstartS | 0x030 00DC | 31:0 | TRI | W/O | No | Yes | Starting S/W parameter | floating | Section 9.3.10 |
| fdSdX | 0x030 00E0 | 31:0 | TRI | W/O | No | Yes | Change in S/W wrt X | floating | Section 9.3.20 |
| fdSdY | 0x030 00E4 | 31:0 | TRI | W/O | No | Yes | Change in S/W wrt Y | floating | Section 9.3.20 |
| fstartT | 0x030 00E8 | 31:0 | TRI | W/O | No | Yes | Starting T/W parameter | floating | Section 9.3.10 |
| fdTdX | 0x030 00EC | 31:0 | TRI | W/O | No | Yes | Change in T/W wrt X | floating | Section 9.3.20 |
| fdTdY | 0x030 00F0 | 31:0 | TRI | W/O | No | Yes | Change in T/W wrt Y | floating | Section 9.3.20 |
| fstartW | 0x030 00F4 | 31:0 | F, TRI | W/O | No | Yes | Starting 1/W parameter | floating | Section 9.3.14 |
| fdWdX | 0x030 00F8 | 31:0 | F, TRI | W/O | No | Yes | Change in 1/W wrt X | floating | Section 9.3.22 |
| fdWdY | 0x030 00FC | 31:0 | F, TRI | W/O | No | Yes | Change in 1/W wrt Y | floating | Section 9.3.22 |
| ftriangleCmd | 0x030 0100 | 31 | F, TRA | W/O | No | Yes | Render Triangle Command | - | Section 9.3.23 |

a. wrt: with respect to

## 9.3 3D Register Detailed Descriptions

### 9.3.1 status

This register is adequately described in Section 6.2.1

### 9.3.2 intrCtrl

This register controls the interrupt capabilities of Voodoo3. Note that writes to this register are not pushed onto the PCI command FIFO and may be processed out of order. That is, a write to this register may be actually processed earlier than a write to another register that went through the FIFO even though the FIFOed write took place earlier.

| Bit | Description |
|---|---|
| 31 | **PCI_INTA_N Pin Level:** This bit reflects the level on the PCI interrupt request pin. Zero indicates the interrupt request is active. Writing a '1' to this bit removes the interrupt request from the pin. |
| 30 | **VGA Interrupt:** If this bit is '1', it indicates that a VGA interrupt is pending. The interrupt request is cleared by writing 0 to this bit and '1' to bit 31. In the case of the VGA interrupt request, it is also necessary to fiddle bits in CR11 (but you already knew that). |
| 29:24 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 23 | **VMI Interrupt:** If this bit is '1', it indicates that a VMI interrupt is pending. The interrupt request is cleared by writing 0 to this bit and '1' to bit 31. It is also probably necessary to clear the interrupt in the VMI device itself, as well. |
| 22 | **Hole Counting Interrupt:** If this bit is '1', it indicates that a hole counting interrupt is pending. The interrupt request is cleared by writing 0 to this bit and '1' to bit 31. |
| 21 | **VMI Interrupt Enable:** This bit is programmed to '1' to enable VMI interrupts. The default for this bit is 0. |
| 20 | **Hole Counting Interrupt Enable:** This bit is programmed to '1' to enable hole counting interrupts. The default for this bit is 0. |
| 19:12 | **User Interrupt Command Tag:** This read-only field returns the identification associated with the individual USERINTERRUPT that was specified in bits [9:2] of the USERINTERRUPT Command. |
| 11 | **User Interrupt:** If this bit is '1', it indicates that a user interrupt is pending. The interrupt request is cleared by writing 0 to this bit and '1' to bit 31. |
| 10 | **PCI FIFO Full Interrupt:** If this bit is '1', it indicates that a PCI FIFO full interrupt is pending. The interrupt request is cleared by writing 0 to this bit and '1' to bit 31. |
| 9 | **Vertical Sync Falling Edge:** If this bit is '1', it indicates that a vertical sync falling edge interrupt is pending. The interrupt request is cleared by writing 0 to this bit and '1' to bit 31. |
| 8 | **Vertical Sync Rising Edge:** If this bit is '1', it indicates that a vertical sync rising edge interrupt is pending. The interrupt request is cleared by writing 0 to this bit and '1' to bit 31. |
| 7 | **Horizontal Sync Falling Edge:** If this bit is '1', it indicates that a horizontal sync falling edge interrupt is pending. The interrupt request is cleared by writing 0 to this bit and '1' to bit 31. |

### 9.3.2 intrCtrl *(cont)*

| Bit | Description |
|---|---|
| 6 | **Horizontal Sync Rising Edge:** If this bit is '1', it indicates that a horizontal sync rising edge interrupt is pending. The interrupt request is cleared by writing 0 to this bit and '1' to bit 31. |
| 5 | **User Interrupt Enable:** This bit is programmed to '1' to enable the USERINTERRUPT command. The default for this bit is 0. |
| 4 | **PCI FIFO Full Interrupt Enable:** This bit is programmed to '1' to enable interrupts when the PCI FIFO is full. The default for this bit is 0. |
| 3 | **Vertical Sync Falling Edge Interrupt Enable:** This bit is programmed to '1' to enable vertical sync falling edge interrupts. The default for this bit is 0. |
| 2 | **Vertical Sync Rising Edge Interrupt Enable:** This bit is programmed to '1' to enable vertical sync rising edge interrupts. The default for this bit is 0. |
| 1 | **Horizontal Sync Falling Edge Interrupt Enable:** This bit is programmed to '1' to enable horizontal sync falling edge interrupts. The default for this bit is 0. |
| 0 | **Horizontal Sync Rising Edge Interrupt Enable:** This bit is programmed to '1' to enable horizontal sync rising edge interrupts. The default for this bit is 0. |

### 9.3.3 vertexA/B/Cx/y

These registers specify the XY coordinates of a triangle. This set of coordinates is in fixed point format. The table shows the addresses for normal and alternate addressing. For these registers, the normal and alternate addresses are the same.

| Vertex | Normal Addressing | Alternate Addressing |
|---|---|---|
| Ax | 0x020 0008 | 0x020 0008 |
| Av | 0x020 000C | 0x020 000C |
| Bx | 0x020 0010 | 0x020 0010 |
| By | 0x020 0014 | 0x020 0014 |
| Cx | 0x020 0018 | 0x020 0018 |
| Cy | 0x020 001C | 0x020 001C |

| Bit | Description |
|---|---|
| 31:16 | **Reserved:** Reserved bit must be written as 0 for upward compatibility. |
| 15:0 | **Vertex Coordinate:** This field specifies one coordinate of one vertex. This is a fixed point 2's complement number with a 12.4 format (twelve bits of integer and four bits of fraction). This format is converted into an internal fixed point format for rendering. |

### 9.3.4 fvertexA/B/Cx/y

These registers specify the XY coordinates of a triangle. This set of registers is in floating point format. The table shows the addresses for normal and alternate addressing. For these registers, the normal and alternate addresses are the same.

| Vertex | Normal Addressing | Alternate Addressing |
|---|---|---|
| Ax | 0x020 0088 | 0x030 0088 |
| Ay | 0x020 008C | 0x030 008C |
| Bx | 0x020 0090 | 0x030 0090 |
| By | 0x020 0094 | 0x030 0094 |
| Cx | 0x020 0098 | 0x030 0098 |
| Cy | 0x020 009C | 0x30 009C |

| Bit | Description |
|---|---|
| 31:0 | **Vertex Coordinate:** This field specifies one coordinate of one vertex. This is an IEEE 32-bit single-precision floating point number. This format is converted into an internal fixed point format for rendering. |

### 9.3.5 startRGBA

These registers specify the starting color information (red, green, blue, and alpha) of the triangle to be rendered. These colors are associated with vertex A. The table shows the addresses for normal and alternate addressing.

| Color | Normal Addressing | Alternate Addressing |
|---|---|---|
| Red | 0x020 0020 | 0x030 0020 |
| Green | 0x020 0024 | 0x030 002C |
| Blue | 0x020 0028 | 0x030 0038 |
| Alpha | 0x020 0030 | 0x030 0050 |

| Bit | Description |
|---|---|
| 31:24 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 23:0 | **Color Value:** This field specifies the value of the respective color. This is a fixed point 2's complement number with a 12.12 format (twelve bits of integer and twelve bits of fraction). This format is converted into an internal fixed point format for rendering. |

### 9.3.6 fstartRGBA

These registers specify the starting color information (red, green, blue, and alpha) of the triangle to be rendered. These colors are for vertex A. The table shows the addresses for normal and alternate addressing.

| Color | Normal Addressing | Alternate Addressing |
|---|---|---|
| Red | 0x020 00A0 | 0x030 00A0 |
| Green | 0x020 00A4 | 0x030 00AC |
| Blue | 0x020 00A8 | 0x030 00B8 |
| Alpha | 0x020 00B0 | 0x030 00D0 |

| Bit | Description |
|---|---|
| 31:0 | **Color Value:** This field specifies the value of the respective color. This is an IEEE 32-bit single-precision floating point number. This format is converted into an internal fixed point format for rendering. |

### 9.3.7 startZ

This register specifies the start Z information of a triangle to be rendered. This depth is associated with vertex A. The table shows the addresses for normal and alternate addressing.

| | Normal Addressing | Alternate Addressing |
|---|---|---|
| startZ | 0x020 002C | 0x030 0044 |

| Bit | Description |
|---|---|
| 31:0 | **Depth:** This field specifies the initial Z value. This is a fixed point 2's complement number with 20 bits of integer and twelve bits of fraction. This format is converted into an internal fixed point format for rendering. |

### 9.3.8 fstartZ

This register specifies the start Z information of a triangle to be rendered. This depth is associated with vertex A. The table shows the addresses for normal and alternate addressing.

| | Normal Addressing | Alternate Addressing |
|---|---|---|
| fstartZ | 0x020 00AC | 0x030 00C4 |

| Bit | Description |
|---|---|
| 31:0 | **Depth:** This field specifies the initial Z value. This is an IEEE 32-bit single-precision floating point number. This format is converted into an internal fixed point format for rendering. |

### 9.3.9 startS

This register specifies the start S/W information of a triangle to be rendered. This information is associated with vertex A. The S and T coordinates transmitted to Voodoo3 for rendering must be divided by W by the host as Voodoo3 actually 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 table shows the addresses for normal and alternate addressing.

| | Normal Addressing | Alternate Addressing |
|---|---|---|
| startS | 0x020 0034 | 0x030 005C |

| Bit | Description |
|---|---|
| 31:0 | **S:** This field specifies the initial S value. This is a fixed point 2's complement number with 14 bits of integer and 18 bits of fraction. This format is converted into an internal fixed point format for rendering. |

### 9.3.10 fstartS

This register specifies the start S/W information of a triangle to be rendered. This information is associated with vertex A. The S and T coordinates transmitted to Voodoo3 for rendering must be divided by W by the host as Voodoo3 actually 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 table shows the addresses for normal and alternate addressing.

| Depth | Normal Addressing | Alternate Addressing |
|---|---|---|
| fstartS | 0x020 00B4 | 0x030 00DC |

| Bit | Description |
|---|---|
| 31:0 | **S:** This field specifies the initial S value. This is an IEEE 32-bit single-precision floating point number. This format is converted into an internal fixed point format for rendering. |

### 9.3.11 startT

This register specifies the start T/W information of a triangle to be rendered. This information is associated with vertex A. The S and T coordinates transmitted to Voodoo3 for rendering must be divided by W by the host as Voodoo3 actually 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 table shows the addresses for normal and alternate addressing.

| | Normal Addressing | Alternate Addressing |
|---|---|---|
| startT | 0x020 0038 | 0x030 0068 |

| Bit | Description |
|---|---|
| 31:0 | **T:** This field specifies the initial T value. This is a fixed point 2's complement number with 14 bits of integer and 18 bits of fraction. This format is converted into an internal fixed point format for rendering. |

### 9.3.12 fstartT

This register specifies the start T/W information of a triangle to be rendered. This information is associated with vertex A. The S and T coordinates transmitted to Voodoo3 for rendering must be divided by W by the host as Voodoo3 actually 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 table shows the addresses for normal and alternate addressing.

| Depth | Normal Addressing | Alternate Addressing |
|---|---|---|
| fstartT | 0x020 00B8 | 0x030 00E8 |

| Bit | Description |
|---|---|
| 31:0 | **T:** This field specifies the initial T value. This is an IEEE 32-bit single-precision floating point number. This format is converted into an internal fixed point format for rendering. |

### 9.3.13 startW

This register specifies the start 1/W information of a triangle to be rendered. This information is associated with vertex A. The W coordinates transmitted to Voodoo3 for rendering must be the reciprocal of W as Voodoo3 actually iterates 1W prior to perspective correction. During rendering, the iterated W coordinates are (optionally) divided by the iterated W parameter to perform perspective correction. The table shows the addresses for normal and alternate addressing.

| | Normal Addressing | Alternate Addressing |
|---|---|---|
| startW | 0x020 003C | 0x030 0074 |

| Bit | Description |
|---|---|
| 31:0 | **W:** This field specifies the initial W value. This is a fixed point 2's complement number with 14 bits of integer and 18 bits of fraction. This format is converted into an internal fixed point format for rendering. |

### 9.3.14 fstartW

This register specifies the start 1/W information of a triangle to be rendered. This information is associated with vertex A. The W coordinates transmitted to Voodoo3 for rendering must be the reciprocal of W as Voodoo3 actually iterates 1W prior to perspective correction. During rendering, the iterated W coordinates are (optionally) divided by the iterated W parameter to perform perspective correction. The table shows the addresses for normal and alternate addressing.

| | Normal Addressing | Alternate Addressing |
|---|---|---|
| fstartW | 0x020 00BC | 0x030 00F4 |

| Bit | Description |
|---|---|
| 31:0 | **T:** This field specifies the initial T value. This is an IEEE 32-bit single-precision floating point number. This format is converted into an internal fixed point format for rendering. |

### 9.3.15 dARGBdXY

These registers specify the changes in color information as the triangle is rendered. These are the DDA constants for the color iterators. The d?dX values are added to the respective color component when the pixel being drawn moves from left to right, and are subtracted when the pixel being drawn moves from right to left. The d?dY values are added when the pixel being drawn moves in a positive Y direction and subtracted when the pixel being drawn moves in a negative Y direction. The table shows the addresses for normal and alternate addressing.

| delta | Normal Addressing | Alternate Addressing |
|---|---|---|
| dRdX | 0x020 0040 | 0x030 0024 |
| dRdY | 0x020 0060 | 0x030 0028 |
| dGdX | 0x020 0044 | 0x030 0030 |
| dGdY | 0x020 0064 | 0x030 0034 |
| dBdX | 0x020 0048 | 0x030 003C |
| dBdY | 0x020 0068 | 0x030 0040 |
| dAdX | 0x020 0050 | 0x030 0054 |
| dAdY | 0x020 0070 | 0x030 0058 |

| Bit | Description |
|---|---|
| 31:24 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 23:0 | **dARGBdXY:** This field specifies the change in the respective color component with respect to the respective dimension. This is a fixed point 2's complement number with 12 bits of integer and 12 bits of fraction. This format is converted into an internal fixed point format for rendering. |

### 9.3.16 fdARGBdXY

These registers specify the changes in color information as the triangle is rendered. These are the DDA constants. The d?dX values are added to the respective color component when the pixel being drawn moves from left to right, and are subtracted when the pixel being drawn moves from right to left. The d?dY values are added when the pixel being drawn moves in a positive Y direction and subtracted when the pixel being drawn moves in a negative Y direction. The table shows the addresses for normal and alternate addressing.

| delta | Normal Addressing | Alternate Addressing |
|---|---|---|
| fdRdX | 0x020 00C0 | 0x030 00A4 |
| fdRdY | 0x020 00E0 | 0x030 00A8 |
| fdGdX | 0x020 00C4 | 0x030 00B0 |
| fdGdY | 0x020 00E4 | 0x030 00B4 |
| fdBdX | 0x020 00C8 | 0x030 00BC |
| fdBdY | 0x020 00E8 | 0x030 00C0 |
| fdAdX | 0x020 00D0 | 0x030 00D4 |
| fdAdY | 0x020 00F0 | 0x030 00D8 |

| Bit | Description |
|---|---|
| 31:24 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 23:0 | **dARGBdXY:** This field specifies the change in the respective color component with respect to the respective dimension. This is an IEEE 32-bit single-precision floating point number. This format is converted into an internal fixed point format for rendering. |

### 9.3.17 dZdXY

These registers specify the changes in depth information as the triangle is rendered. These are the DDA constants. The dZdX value is added to the depth component when the pixel being drawn moves from left to right, and are subtracted when the pixel being drawn moves from right to left. The dZdY value is added when the pixel being drawn moves in a positive Y direction and subtracted when the pixel being drawn moves in a negative Y direction.The table shows the addresses for normal and alternate addressing.

| delta | Normal Addressing | Alternate Addressing |
|---|---|---|
| dZdX | 0x020 004C | 0x030 0048 |
| dZdY | 0x020 006C | 0x030 004C |

| Bit | Description |
|---|---|
| 31:0 | **dRGBdXY:** This field specifies the change in the respective depth component with respect to the respective dimension. This is a fixed point 2's complement number with 20 bits of integer and 12 bits of fraction. This format is converted into an internal fixed point format for rendering. |

### 9.3.18 fdZdXY

These registers specify the changes in depth information as the triangle is rendered. These are the DDA constants. The dZdX value is added to the depth component when the pixel being drawn moves from left to right, and are subtracted when the pixel being drawn moves from right to left. The dZdY value is added when the pixel being drawn moves in a positive Y direction and subtracted when the pixel being drawn moves in a negative Y direction.The table shows the addresses for normal and alternate addressing.

| delta | Normal Addressing | Alternate Addressing |
|---|---|---|
| fdZdX | 0x020 00CC | 0x030 00C8 |
| fdZdY | 0x020 00EC | 0x030 00CC |

| Bit | Description |
|---|---|
| 31:0 | f**dZdXY:** This field specifies the change in the respective depth component with respect to the respective dimension. This is an IEEE 32-bit single-precision floating point number. This format is converted into an internal fixed point format for rendering. |

### 9.3.19 dSTdXY

These registers specify the changes in S/W and T/W as the triangle is rendered. These are the DDA constants. The dSTdX value is added to the depth component when the pixel being drawn moves from left to right, and are subtracted when the pixel being drawn moves from right to left. The dSTdY value is added when the pixel being drawn moves in a positive Y direction and subtracted when the pixel being drawn moves in a negative Y direction.The table shows the addresses for normal and alternate addressing.

| delta | Normal Addressing | Alternate Addressing |
|---|---|---|
| dSdX | 0x020 0054 | 0x030 0060 |
| dSdY | 0x020 0074 | 0x030 0064 |
| dTdX | 0x020 0058 | 0x030 006C |
| dTdY | 0x020 0078 | 0x030 0070 |

| Bit | Description |
|---|---|
| 31:0 | **dSTdXY:** This field specifies the change in the respective S/W and t?w component with respect to the respective dimension. This is a fixed point 2's complement number with 14 bits of integer and 18 bits of fraction. This format is converted into an internal fixed point format for rendering. |

### 9.3.20 fdSTdXY

These registers specify the changes in S/W and T/W as the triangle is rendered. These are the DDA constants. The fdSTdX value is added to the depth component when the pixel being drawn moves from left to right, and are subtracted when the pixel being drawn moves from right to left. The fdSTdY value is added when the pixel being drawn moves in a positive Y direction and subtracted when the pixel being drawn moves in a negative Y direction.The table shows the addresses for normal and alternate addressing.

| delta | Normal Addressing | Alternate Addressing |
|---|---|---|
| fdSdX | 0x020 00D4 | 0x030 00E0 |
| fdSdY | 0x020 00F4 | 0x030 00E4 |
| fdTdX | 0x020 00D8 | 0x030 00EC |
| fdTdY | 0x020 00F8 | 0x030 00F0 |

| Bit | Description |
|---|---|
| 31:0 | f**dZdXY:** This field specifies the change in the respective depth component with respect to the respective dimension. This is an IEEE 32-bit single-precision floating point number. This format is converted into an internal fixed point format for rendering. |

### 9.3.21 dWdXY

These registers specify the changes in 1/W information as the triangle is rendered. These are the DDA constants. The dWdX value is added to the depth component when the pixel being drawn moves from left to right, and are subtracted when the pixel being drawn moves from right to left. The dWdY value is added when the pixel being drawn moves in a positive Y direction and subtracted when the pixel being drawn moves in a negative Y direction. The W coordinates transmitted to Voodoo3 for rendering must be the reciprocal of W as Voodoo3 actually iterates 1/W prior to perspective correction. During rendering, the iterated W coordinates are (optionally) divided by the iterated W parameter to perform perspective correction. The table shows the addresses for normal and alternate addressing.

| delta | Normal Addressing | Alternate Addressing |
|---|---|---|
| dWdX | 0x020 005C | 0x030 0078 |
| dWdY | 0x020 007C | 0x030 007C |

| Bit | Description |
|---|---|
| 31:0 | **dWdXY:** This field specifies the change in the respective depth component with respect to the respective dimension. This is a fixed point 2's complement number with 2 bits of integer and 30 bits of fraction. This format is converted into an internal fixed point format for rendering. |

### 9.3.22 fdWdXY

These registers specify the changes in 1/W information as the triangle is rendered. These are the DDA constants. The dWdX value is added to the depth component when the pixel being drawn moves from left to right, and are subtracted when the pixel being drawn moves from right to left. The dWdY value is added when the pixel being drawn moves in a positive Y direction and subtracted when the pixel being drawn moves in a negative Y direction. The W coordinates transmitted to Voodoo3 for rendering must be the reciprocal of W as Voodoo3 actually iterates 1/W prior to perspective correction. During rendering, the iterated W coordinates are (optionally) divided by the iterated W parameter to perform perspective correction. The table shows the addresses for normal and alternate addressing.

| delta | Normal Addressing | Alternate Addressing |
|---|---|---|
| fdWdX | 0x020 00DC | 0x030 00F8 |
| fdWdY | 0x020 00FC | 0x030 00FC |

| Bit | Description |
|---|---|
| 31:0 | **dWdXY:** This field specifies the change in the respective depth component with respect to the respective dimension. This is a fixed point 2's complement number with 2 bits of integer and 30 bits of fraction. This format is converted into an internal fixed point format for rendering. |

### 9.3.23 triangleCMD/ftriangleCMD

Writing to either of these two registers initiates rendering of the triangle defined in the respective of registers. That is, a write to triangleCMD initiates rendering using the parameters in the fixed point registers and a write to ftriangleCMD initiates rendering using the parameters in the floating point registers. The respective register set must have been programmed in order to render anything useful.

| Register | Normal Addressing | Alternate Addressing |
|---|---|---|
| triangleCMD | 0x020 0080 | 0x030 0080 |
| ftriangleCMD | 0x020 0100 | 0x030 0100 |

| Bit | Description |
|---|---|
| 31 | **Sign:** If this bit is 0, the triangle is oriented counter-clockwise (positive area). If this bit is '1', the triangle is oriented clockwise (negative area).<br>To calculate the area of a triangle, the following steps are performed.<br>First, sort the vertices (A, B, C) in order of increasing Y. This results in the following inequality: A.y ≤ B.y ≤ C.y.<br>The area is calculated with the following equation.<br>$$Area = \frac{((dxAB \bullet dyBC) - (dxBC \bullet dyAB))}{2}$$<br>where<br>$$dxAB = Ax - Bx$$<br>$$dyBC = By - Cy$$<br>$$dxBC = Bx - Cx$$<br>$$dyAB = Ay - By$$ |
| 30:0 | **Unused:** This field is unused. The rendering engine requires only the sign of the area. |

### 9.3.24 nopCMD

Writing to this register executes the NOP command, which flushes the graphics pipeline. Bit zero of the data written optionally clears the pixel counter registers.

| Bit | Description |
|---|---|
| 30:1 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 0 | **Clear:** If this bit is '1', the pixel counter registers (fbiPixelsIn, fbiChromaFail, fbiZfuncFail, and fbiPixelsOut) are cleared. |

### 9.3.25 fastfillCMD

Writing to this register executes the FASTFILL command, which is used to clear the RGB and depth buffers quickly. The extent to be cleared is taken from the clipLeftRight and clipLowYHighY registers. This region is inclusive of clipLeft and clipLowY, but exclusive of clipRight and clipHighY.
If fbzMode[9] is '1', the color in color1 is written to the RGB buffer. If fbzMode[9] is 0, the RGB buffer is not written.
If fbzMode[10] is '1', the depth value from the zaColor register is written to the depth buffer. If fbzMode[10] is 0, the depth buffer is not written. It serves no good purpose to execute this command without one or the other of these two bits being set. If the frame buffer is based on SDRAMs, only one of these two bits can be set.
Since color1 is a 24-bit color value, either dithering or bit truncation must be used to translate into the native 16-bit frame buffer. Dithering is enabled by setting fbzMode[8] to '1'. Dithering is never used for SGRAMs since they support blockwriting (even if dithering is enabled both in fbzMode[8] and bit 0 of the data written in this command.

| Bit | Description |
|---|---|
| 30:1 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 0 | **Disable Dithering:** If this bit is '1', dithering will not be used during FASTFILL regardless of the ram type and regardless of the setting of fbzMode[8]. |

### 9.3.26 swapbufferCMD

Writing to this register executes the SWAPBUFFER command. The

| Bit | Description |
|---|---|
| 31:10 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 9 | **Swap Buffer Disable Swap:** When this bit is '1', the outstanding swap count is decremented, but the video pointer is not swapped. |
| 8:1 | **Swap Buffer Interval:** This field is used to specify the number of vertical retraces to wait before swapping buffers. An internal counter is incremented with each vertical retrace, and the color buffers are not swapped until that counter is greater than the value programmed into this field. After a swap occurs, the internal counter is cleared. This is used to maintain a constant frame update rate. This has the effect of slowing the overall frame rate, but guaranteeing that all frames are displayed for the same period. |
| 0 | **Sync Swapping:** When this bit is 1, the swapping of buffers is synchronized with vertical retrace. If this synchronization is not enabled, visible screen tearing is likely. Setting this bit forces the buffer swap to take place during vertical retrace, guaranteeing that any given frame will be displayed from the same buffer, eliminating the tearing. |

### 9.3.27 fbzColorPath

Bits and fields in this register control the color and alpha rendering pixel pipelines. The color channels are controlled separately from the alpha channel. There are two primary color selections units: the CCU (Color Combine Unit) and ACU (Alpha Combine Unit). See Section 10.3.

| Bit | Description |
|---|---|
| 31:30 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 29 | **Enable Anti-Aliasing:** If this bit is '1', anti-aliasing is enabled. |
| 28 | **Enable Parameter Clamping:** When this bit is '1', the W triangle parameter is clamped to the range 0 through 0xFF inclusive for use in the ACU and the fog unit. This bit has no effect on the W triangle during triangle setup and sub-pixel correction. The unclamped output of the W triangle iterator is used when performing floating point W-buffering (fbzMode[21] = 0). When this bit is 0, then the W parameter used as input to the ACU and fog units is allowed to wrap according to the following.<br><br>`if (wIterator[47:32] == 0xFFFF)`<br>`wClamped[7:0] = 0x00;`<br>`else if (zIterator[47:32] == 0x0100`<br>`wClampled[7:0] = 0xFF;`<br>`else`<br>`wClampled[7:0] = wIterator [39:32};` |
| 27 | **Enable Texture Mapping:** When this bit is '1', texture mapping is enabled. If texture mapping is not desired (for example, for Gouraud shading or flat triangles), this bit must be programmed to 0. |
| 26 | **Enable SubPixel Correction:** When this bit is '1', subpixel correction is enabled for all parameters. Voodoo3 will automatically subpixel correct the incoming color, depth, and texture coordinate parameters for triangles not aligned on integer spatial boundaries. Enabling subpixel correction increases the on-chip triangle setup time from 7 clocks to 16 clocks. However, since the triangle setup engine is separately pipelined from the triangle drawing engine, little actual performance penalty will be seen (depending on the average triangle size).<br>When subpixel correction is enabled, the corrections are 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, which have already been corrected for the last rendered triangle. Correcting parameters twice will result in inaccuracies in the starting parameter values. |
| 25 | **cca_invert_output:** When this bit is '1', the alpha value at the output of the ACU is inverted. |
| 24 | **cca_add_alocal:** When this bit is '1', the alocal value is added to the multiplier output in sheet two of the ACU. This bit must be not set simultaneously with bit 23. |
| 23 | **cca_add_clocal:** When this bit is '1', the clocal value is added to the multiplier output in sheet two of the ACU. This bit must not be set simultaneously with bit 24. |

### 9.3.27 fbzColorPath *(cont)*

| Bit | Description |
|---|---|
| 22 | **cca_reverse_blend:** When this bit is ‘1’, the output of the cca_mselect mux is inverted before being used. |
| 21:19 | **cca_mselect[2:0]:** This field controls the cca_mselect mux (right side of ACU, sheet 2) according to the table. Values not in the table must not be programmed into this field. |

| Value | Selection |
|---|---|
| 000b | constant zero |
| 001b | a_local |
| 010b | a_other |
| 011b | a_local |
| 100b | texture alpha |

| | |
|---|---|
| 18 | **cca_sub_clocal:** If this bit is 1, a_local is selected in the sub_alocal mux. If this bit is 0, the constant zero is selected. |
| 17 | **cca_zero_other:** This bit controls the cca_zero_other_mux in the ACU. When this bit is 1, a_other is forced to constant 0. When this bit is 0, a_other is controlled by alphaSelect[1:0]. |
| 16 | **cc_invert_output:** When this bit is ‘1’, the color value at the output of the CCU is inverted. |
| 15 | **cc_add_alocal:** When this bit is ‘1’, the alocal value is added to the multiplier output in sheet two of the CCU. This bit must be not set simultaneously with bit 14. |
| 14 | **cc_add_clocal:** When this bit is ‘1’, the clocal value is added to the multiplier output in sheet two of the CCU. This bit must not be set simultaneously with bit 15. |
| 13 | **cc_reverse_blend:** When this bit is ‘1’, the output of the cc_mselect mux is inverted before being used. |
| 12:10 | **cc_mselect[2:0]:** This field controls the cc_mselect mux (right side of CCU, sheet 2) according to the table. Values not in the table must not be programmed into this field. |

| Value | Selection |
|---|---|
| 000b | constant zero |
| 001b | c_local |
| 010b | a_other |
| 011b | a_local |
| 100b | texture alpha |
| 101b | texture RGB |

### 9.3.27 fbzColorPath *(cont)*

| Bit | Description |
|---|---|
| 9 | **cc_sub_clocal:** If this bit is 1, c_local is selected in the sub_clocal mux. If this bit is 0, the constant zero is selected. |
| 8 | **cc_zero_other:** This bit controls the cc_zero_other_mux. When this bit is 1, c_other is forced to constant 0. When this bit is 0, c_other is controlled by rgbSelect[1:0]. |
| 7 | **cc_localselect_override mux control:** This bit controls the cc_localselect_override mux. When this bit is 0, cc_locationselect controls the cc_localselect mux. When this bit is '1', bit 7? 0? of the texture alpha unit controls the cc_localselect mux. |
| 6:5 | **cca_localselect mux control[1:0]:** These two bits control the a_other mutliplexer at the upper right of the ACU. |

| Value | Selection |
|---|---|
| 00b | iterated alpha |
| 01b | color0 alpha |
| 10b | interated Z[27:20], clamped |
| 11b | iterated W[29:32], clamped |

| | |
|---|---|
| 4 | **cc_localselect mux control:** This bit controls the cc_localselect mux (upper right corner of the CCU). This bit is effective only if bit 7 is 0. |
| 3:2 | **aSelect[1:0]:** These two bits control the a_other mutliplexer at the top left of the ACU. |

| Value | Selection |
|---|---|
| 00b | iterated alpha |
| 01b | texture alpha |
| 10b | color1 alpha |
| 11b | Reserved (frame buffer) |

| | |
|---|---|
| 1:0 | **rgbSelect[1:0]:** These two bits control the c_other multiplexer at the top left of the CCU.. |

| Value | Selection |
|---|---|
| 00b | Iterated RGB |
| 01b | TREX Color Output |
| 10b | color1RGB (constant) |
| 11b | Reserved (frame buffer) |

### 9.3.28 fogMode

This register controls the fog function of Voodoo3. The fog unit is described in Section 10.3.4.

| Bit | Description |
|---|---|
| 31:8 | **Reserved:** Reserved bits must be programmed to 0 for upward compatibility. |
| 7 | **forzones control:** When this bit is '1', signed fog delta is enabled. |
| 6 | **fogdither control:** When this bit is '1', dither the fog blending component. |
| 5 | **fogconstant control:** When this bit is '1', the contents of the fogColor register are used as the fog component. When this bit is 0, the output of the fog multiplier is used. |
| 4 | **fogz control:** This bit is used with bit 3 to control the multiplexor that selects one input to the fog multiplier, according to the table. |

| fogz | fogalpha | Selection | Note |
|---|---|---|---|
| 0 | 0 | fog table alpha | |
| 0 | 1 | iterated alpha | |
| 1 | 0 | iteratedZ[27:20] | clamped |
| 1 | 1 | iteratedW[39:32] | clamped |

| | |
|---|---|
| 3 | **fogalpha control:** This bit is used with bit 4 to control the multiplexor described with bit 4. |
| 2 | **fogmult control:** When this bit is '1', it forces the fog multiplier input to unity. When this bit is 0, the color from the CCU is used. |
| 1 | **fogadd control:** When this bit is '1', the fogColor is not added to the color from the CCU. When this bit is 0, the fogColor is used. |
| 0 | **Enable Fog:** When this bit is '1', the fog unit is enabled. When fog is enabled, the fog color specified in fogColor is blended with the source pixels as a function of the fogTable values and iterated W. |

### 9.3.29 alphaMode

This register controls the alpha blending (Section 10.3.3) functions of Voodoo3, as well as alpha comparison (Section 10.4.1). These two functions are really very different, sharing only the word 'alpha' and a control register. The alpha comparison is controlled with bits [31:24], [3:1], and 0. Bits [23:20], [19:16], [15:12], [11:8], and 4 control alpha blending.

| Bit | Description |
|---|---|
| 31:24 | **Alpha Reference Value:** This field specifies the alpha reference value used in the alpha comparison function. |
| 23:20 | **Destination Alpha-Channel Alpha Blending Factor:** This four-bit field specifies the destination alpha-channel alpha blending factor. See Table 10.4. |
| 19:16 | **Source Alpha-Channel Alpha Blending Factor:** This four-bit field specifies the source alpha-channel alpha blending factor. See Table 10.4. |
| 15:12 | **Destination RGB Alpha Blending Factor:** This four-bit field specifies the destination RGB alpha blending factor. See Table 10.4. |
| 11:8 | **Source RGB Alpha Blending Factor:** This four-bit field specifies the source RGB alpha blending factor. See Table 10.4. |
| 8:7 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 6 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 5 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 4 | **Enable Alpha Blending:** When this bit is '1', alpha blending is enabled. See Section 10.3.3. |
| 3:1 | **Alpha Op:** This field specifies the alpha operation. See Table 10.6. |
| 0 | **Enable Alpha Function:** When this bit is '1'. the alpha function is enabled. See Section 10.4.1. |

### 9.3.30 lfbMode

This register contains bits that control linear frame buffer accesses.

| Bit | Description |
|---|---|
| 31:17 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 16 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 15 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 14 | **Linear Frame Buffer Write Access W Select:** This bit is used to select the W component used for LFB writes processed through the pixel pipeline. If this bit is 0, then the most significant bits of the fractional component of the 48-bit W values passed to the pixel pipeline is the 16-bit Z value associated with the LFB write. Note that the 16-bit Z value is dependent on the LFB format and is either passed down pixel-by-pixel from the CPU, or is set the constant zaColor[15:0].<br>If this bit is '1', then the most significant bits of the fractional component of the 48-bit W value passed to the pixel pipeline for LFB writes is zaColor[15:0].<br>Regardless of the setting of this bit, when LFB writes go through the pixel pipeline, all other bits except the 16 MSBs of the fractional component of the W value are set to 0. This bit is ignored if LFB writes bypass the pixel pipeline. |
| 13 | **LFB Access Y Origin:** When this bit is 0, the Y origin is the top of the screen (smallest Y address). When this bit is '1', the Y origin is the bottom of the screen (largest Y address). Note that this bit does not affect rendering operations (FASTFILL and TRIANGLE commands). |
| 12 | **Byte Swizzle LFB Writes:** When this bit is '1', byte swizzling is enabled. This swizzles the four bytes with a 32-bit word to correct for endian differences between Voodoo3 and the host CPU. This bit should be set for big endian CPUs (for example, PowerPC) and should be cleared for little endian CPUS (for example, x86 processors). When byte swizzling is enabled, bits [31:24] are swapped with bits [7:0] and bits [23:16] are swapped with bits [15:8].<br>The order of swapping and swizzling operations for LFB writes is as follows: Byte swizzling is performed first on all incoming LFB data, if enabled with bit 12 and irrespective of the LFB data format. After byte swizzling, 16-bit word swapping is performed if enabled by bit 11. Sixteen-bit word swapping is never performed on LFB data when data formats 4 or 5 are used. Sixteen-bit word swapping is performed on the LFB data that was previously optionally swapped. Finally, after both swizzling and 16-bit word swapping are performed, the individual color channels are selected as specified in lfbMode[10:9]. The color channels are selected on the LFB data that was previously swizzled and/or swapped. |

### 9.3.30 lfbMode *(cont)*

| Bit | Description |
|---|---|

11 **16-bit Word Swap LFB Writes:** This bit specifies the format of two 16-bit data types passed with a single 32-bit write. For LFB formats 0-2, two 16-bit data transfers can be packing into one 32-bit write. This bit specifies which 16-bit value correspond to which pixel on the screen. The following table shows the pixel packing for packed 32-bit LFB formats 0-2.

| lfbMode[11] | Screen Pixel Packing | |
|---|---|---|
| | Host Data [31:16] | Host Data [15:0] |
| 0 | Right Pixel | Left Pixel |
| 1 | Left Pixel | Right Pixel |

For LFB formats 12-14, this bit specifies the bit locations of the two 16-bit data types passed. The following table shows the data packing for 32-bit LFB formats 12-14.

| lfbMode[11] | Screen Pixel Packing | |
|---|---|---|
| | Host Data [31:16] | Host Data [15:0] |
| 0 | Z Value | RGB Value |
| 1 | RGB Value | Z Value |

For LFB format 15, this bit specifies the bit locations of the 16-bit depth values passed. The following table shows the data packing for 32-bit LFB format 15.

| lfbMode[11] | Screen Pixel Packing | |
|---|---|---|
| | Host Data [31:16] | Host Data [15:0] |
| 0 | Z Right Pixel | Z Left Pixel |
| 1 | Z Left Pixel | Z Right Pixel |

10:9 **LFB RGBA Lanes:** This field specifies the LFB RGBA lanes as shown in the table.

| lfbMode[10:9] | RGB Channel Format |
|---|---|
| 00b | ARGB |
| 01b | ABGR |
| 10b | RGBA |
| 11b | BGRA |

### 9.3.30 lfbMode *(cont)*

| Bit | Description |
|---|---|
| 8 | **Enable Pixel Pipeline Processing for LFB Writes:** When this bit is '1', LFB pixels are processed by the normal Voodoo3 pixel pipeline. Each pixel written must have an associated depth and alpha value, and is also subject to fog mode, alpha function, and so on.<br>When this bit is 0, pixels written using LFB access bypass the normal pixel pipeline and are written to the specified buffer unconditionally and the value written are unconditionally written into the color/depth buffers except for optional color dithering. Depth function, alpha blending, alpha test, and color/depth write masks are all bypassed.<br>If this bit is 0, then only the buffers specified in the particular LFB format are updated. If this bit is 0, the color and Z mask bits in fbzMode[10:9] are ignored for LFB writes. |
| 7:6 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 5:4 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 3:0: | **LFB Write Format:** This field specifies the Voodoo3 linear frame buffer write format. Values not in the table must not be programmed into this field. |

| lfbMode[3:0] | Write Format | Format Size |
|---|---|---|
| 0000b | 16-bit RGB 5-6-5 | 16 bits |
| 0001b | 16-bit RGB x-5-5-5 | 16 bits |
| 0010b | 16-bit ARGB 1-5-5-5 | 16 bits |
| 0100b | 24-bit RGB x-8-8-8 | 32 bits |
| 0101b | 32-bit aRGB 8-8-8-8 | 32 bits |
| 1100b | 16-bit depth, 16-bit RGB 5-6-5 | 32 bits |
| 1101b | 16-bit depth, 16-bit RGB x-5-5-5 | 32 bits |
| 1110b | 16-bit depth, 16-bit ARGB 1-5-5-5 | 32 bits |
| 1111b | 16-bit depth, 16-bit depth | 32 bits |

### 9.3.31 fbzMode

Bits this register specify frame buffer and depth buffer rendering functions of Voodoo3. This includes clipping, chroma-keying, depth-buffering, dithering, and masking.

| Bit | Description |
|---|---|
| 31:22 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 21 | **Depth Float Select:** When this bit is ‘1’, iterated Z is used for floating point depth buffering. When this bit is 0, iterated W is used for floating point depth buffering. Floating point depth buffering is enabled when fbxMode[4] = 1. Either Z or W is converted into a 4.12 floating point format. |
| 20 | **Depth Buffer Source Compare Select:** When this bit is ‘1’, the constant depth value specified in zaColor[15:0] is used as the source depth value for the depth buffer comparison. When this bit is 0, the source depth value for the depth buffer comparison is either iterated Z or iterated W (see fbzMode[3]) and may be biased (see fbzMode[16]). |
| 19 | **Enable Alpha-blending Dither Subtraction:** When this bit is ‘1’, the dither matrix used to convert 24-bit color 16-bit color is subtracted from the destination color before applying the alpha-blending algorithm. This is used to enhance image quality when performing alpha-blending. Bit 11 is used to select which dither matrix to use. |
| 18 | **Enable Alpha Plane:** When this bit is ‘1’, the destination alpha plane is enabled. The auxiliary buffer is used as the destination alpha planes. When this bit is ‘1’, depth buffering cannot be used; fbzMode[4] must be 0. |
| 17 | **Rendering Command Y Origin:** This bit specifies the Y origin for rendering operations (FASTFILL and TRIANGLE) and linear frame buffer writes when the pixel pipeline is bypassed (lfbMode[8] = 0). This bit does not control linear frame buffer writes when the pixel pipeline is bypassed; rather lfbMode[13] is used. |
| 16 | **Enable Depth-biasing:** When this bit is ‘1’, the calculated depth value (regardless of whether Z or 1/W is selected) is added to zaColor[15]. This is used to eliminate aliasing artifacts when rendering co-planar polygons. |
| 15:14 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| **13** | **Enable Alpha-channel Mask:** When this bit is ‘1’, the alpha-channel mask is enabled. Bit zero of the incoming alpha value is used to mask writes to the color and depth buffers. If this bit is ‘1’ and bit zero of the incoming alpha value is 0, the pixel is invalidated in the pixel pipeline. fbiAfuncFail is incremented and no write occurs to the color or depth buffers. If bit zero of the incoming alpha value is 1, then pixel is drawn normally subject to depth functions, alpha blending functions, alpha test and color and depth masking. |
| 12 | **Enable Stipple Pattern Masking:** When fbzMode[2] is ‘1’, this bit selects the stipple mode. When this bit is 0, stipple rotate mode is selected; when this bit is ‘1’, stipple pattern mode is selected. See Section 10.4.4. |
| 11 | **Dither Algorithm Selection:** When fbzMode[8] is ‘1’, this bit selects the dithering algorithm. When this bit is ‘1’, a 2 x 2 ordered dither algorithm is used; when this bit is 0, a 4 x 4 ordered dither algorithm is used. This bit also selects the dither matrix to be used when bit 19 is ‘1’. In some 3Dfx Interactive, Inc. software, this bit is always ‘1’. |

### 9.3.31 fbzMode *(cont)*

| Bit | Description |
|---|---|
| 10 | **Depth/alpha Buffer Write Mask:** When this bit is 0, all writes to the depth buffer are invalidated. This bit must be '1' for normal depth-buffer operation. |
| 9 | **RGB Buffer Write Mask:** When this bit is 0, all writes to the RGB buffers are invalidated. This bit must be '1', for normal drawing into the RGB buffers. |
| 8 | **Enable Dithering:** When this bit is '1', native 24-bit source pixels are dithered into 16-bit RGB color values with no performance penalty. fbzMode[11] specifies the dithering algorithm. When this bit is 0, native 24-bit source pixels are converted into 16-bit RGB color values by bit truncation. |
| 7:5 | **Depth-Buffer Function:** This field specifies the depth-buffering function as shown in Table 10.7. |
| 4 | **Enable Depth Buffering:** When this bit is '1', depth buffering is enabled as defined in fbzMode[3] and fbzMode[7:5]. |
| 3 | **W-Buffer Select:** When fbzMode[4] is '1', this bit selects the interator used for depth buffering comparison. When this bit is 0, the z iterator is used for depth buffer comparison. When this bit is '1', the w iterator is used for depth buffer comparison. In this case, the inverse of the normalized w iterator is used for the depth-buffer comparison. In effect, this implements a floating-point w-buffering scheme utilizing a 4-bit exponent and a 12-bit mantissa (4.12 floating point format). The inverted w interator is used so that the same depth buffer comparisons can be used as with a typical z-buffer. See |
| 2 | **Enable Stipple Register Masking:** When this bit is '1', stipple register masking is enabled. fbzMode[12] specifies the stipple mode. See Section 10.4.4. |
| 1 | **Enable Chroma-keying:** When this bit is '1', color compare check (chroma-keying) is enabled. Any source pixel matching the color specified in chromaKey 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 fogger in the pixel pipeline. |
| 0 | **Enable Clipping Rectangle:** When this bit is '1', the clipping rectangle defined by clipLeftRight and clipBottomTop is enabled. The clipping rectangle must always be less than or equal to the screen resolution in order to clip to screen coordinates. If clipping is enabled, rendering may not occur outside of the screen resolution. |

### 9.3.32 stipple

This register specifies the mask which is used to enable individual pixel writes to the RGB and depth buffers. See Section 10.4.4.

| Bit | Description |
|---|---|
| 31:0 | stippleMask: This field is the stipple mask. |

### 9.3.33 color0/1

These two registers specify constant color values used for certain rendering function. The table shows the addresses for normal addressing.

| color | Normal Addressing | Used for |
|---|---|---|
| color0 | 0x020 0144 | c_local input in CCU and ACU |
| color1 | 0x020 0148 | c_other in CCU, a_other in ACU |

The color1 register is also used by the FASTFILL command as the color for screen clears. Also, for linear frame buffer write format 15 (16-bit depth, 16-bit depth), the color for the pixel pipeline is taken from color1 if the pixel pipeline is enabled for linear frame buffer writes.

| Bit | Description |
|---|---|
| 31:24 | **Constant Color Alpha:** This field specifies the alpha value of the respective color. |
| 23:16 | **Constant Color Red:** This field specifies the red value of the respective color. |
| 15:8 | **Constant Color Green:** This field specifies the green value of the respective color. |
| 7:0 | **Constant Color Blue:** This field specifies the blue value of the respective color. |

### 9.3.34 fogColor

This register specifies the fog color for fogging operations. Fogging is enabled by programming fogMode[0] to ‘1’. See Section 10.3.4.

| Bit | Description |
|---|---|
| 31:24 | **Reserved:** Reserved bits must be programmed to 0 for upward compatibility. |
| 23:16 | **Fog Color Red:** This field specifies the red value of the fog color. |
| 15:8 | **Fog Color Green:** This field specifies the green value of the fog color. |
| 7:0 | **Fog Color Blue:** This field specifies the blue value of the fog color. |

### 9.3.35 zaColor

This register specifies constant alpha and depth values for linear frame buffer writes, FASTFILL, and co-planar rendering support. For certain frame buffer access formats, the alpha and depth values associated with a pixel written are the values specified in this register.

| Bit | Description |
|---|---|
| 31:24 | **Constant Alpha:** This field specifies a constant alpha value. |
| 23:16 | **Reserved:** Reserved bits must be programmed to 0 for upward compatibility. |
| 15:0 | **Constant Depth:** This field specifies a constant depth value. When executing a FASTFILL command, this field specifies the depth value. When depth biasing is enabled, (fbzMode[16] = ‘1’), this field specifies the constant depth value. |

### 9.3.36 chromaKey

This register specifies a color which is optionally compared with all pixels to be written into the RGB buffer. If a match occurs between an outgoing pixel and this value, and chroma-keying is enabled (fbzMode[1] = '1'), then the pixel is not written into the frame buffer. See Section 10.4.3.

| Bit | Description |
|---|---|
| 31:24 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. The alpha color component of an outgoing pixel is ignored in the chroma-key comparison. |
| 23:16 | **Chroma-key Red:** This field specifies the red value of the chroma-key. |
| 15:8 | **Chroma-key Green:** This field specifies the green value of the chroma-key. |
| 7:0 | **Chroma-key Blue:** This field specifies the blue value of the chroma-key. |

### 9.3.37 chromaRange

This register specifies a color which is optionally compared with all pixels to be written into the frame buffer. If chroma-keying is enabled (fbzMode[1] = 1) and chroma-ranging is enabled (chromaRange[28] = 1), the outgoing pixel color is compared to the color range specified by this register and the chromaKey register. See Section 10.4.3.

| Bit | Description |
|---|---|
| 31:29 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 28 | **Chroma-Range Enable:** When this bit is '1', chroma-range is enabled (if chroma-keying is enabled). |
| 27 | **Chroma-Range Block Mode:** When this bit is '1', union chroma-range function is selected. When this bit is 0, intersection chroma-range function is enabled. |
| 26 | **Chroma-Range Red Mode:** When this bit is '1', exclusive chroma-range red function is selected. When this bit is 0, inclusive chroma-range red function is selected. |
| 25 | **Chroma-Range Green Mode:** When this bit is '1', exclusive chroma-range green function is selected. When this bit is 0, inclusive chroma-range green function is selected. |
| 24 | **Chroma-Range Blue Mode:** When this bit is '1', exclusive chroma-range blue function is selected. When this bit is 0, inclusive chroma-range blue function is selected. |
| 23:16 | **Chroma-Range Red Upper Limit:** This field specifies the upper limit of the red value of the chroma-range. |
| 15:8 | **Chroma-Range Green Upper Limit:** This field specifies the upper limit of the green value of the chorma-range. |
| 7:0 | **Chroma-Range Blue Upper Limit:** This field specifies the upper limit of the blue value of the chroma-range. |

### 9.3.38 userIntrCMD

Writing to this register executes the USERINTERRUPT command. Control bits specify whether the graphics engine stalls until the host program clears the interrupt. An eight-bit tag is available to the host program. If intrCtrl[[5] is 0, writes to this register are ignored.

| Bit | Description |
|---|---|
| 31:10 | **Reserved:** Reserved bit must be written as 0 for upward compatibility. |
| 9:2 | **User Interrupt Tag:** When a user interrupt is generated, the value from this field will be available in intrCtrl[19:12]. If multiple USERINTERRUPT commands have been generated before the host software processes them, the tag in intrCtrl[19:12] reflects the tag of the last USERINTERRUPT command processed by the graphics engine. To avoid this behavior, software may force a single USERINTERRUPT command to be executed at a time by setting both bits 1 and 0 and forcing the engine to stall until the interrupt is cleared. |
| 1 | **Wait for Interrupt:** If this bit is '1', the interrupt generated by USERINTERRUPT (visible in intrCtrl[11]) must be cleared before continuing. |
| 0 | **Wait for Interrupt:** If this bit is '1', the USERINTERRUPT must be cleared before continuing. |

### 9.3.39 colBufferAddr

This register specifies the base address of the color buffer. This address must be aligned on a 16-byte boundary; bits 3:0 must be zero.

| Bit | Description |
|---|---|
| 31:24 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 23:4 | **Color Buffer Base Address:** This field specifies the base address of the color buffer. |
| 3:0 | **Zeroes:** This field must be programmed to 0000b to put the base address on a 16-byte boundary. |

### 9.3.40 colBufferStride

This register specified the memory type and stride of the color buffer.

| Bit | Description |
|---|---|
| 31:16 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 15 | **Memory Type:** If this bit is 0, the color buffer is linear. If this bit is '1', the color buffer is linear. |
| 14 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 13:0 | **Stride:** The meaning of this bit depends on bit 15. If bit 15 is zero (linear memory), this field specifies the linear stride of the color buffer in bytes. The stride must be an integer multiple of 16 bytes; bits 3:0 must be programmed to zero. If bit 15 is one (tiled memory) bits [6:0] of this field specifies the tile stride in tiles. Bits [13:7] are reserved in this case and must be programmed to zero for upward compatibility. |

### 9.3.41 auxBufferAddr

This register specifies the base address of the auxiliary buffer. This address must be aligned on a 16-byte boundary; bits 3:0 must be zero. The auxiliary buffer is used for destination alpha planes or depth buffer.

| Bit | Description |
|---|---|
| 31:24 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 23:4 | **Auxiliary Buffer Base Address:** This field specifies the base address of the auxiliary buffer. |
| 3:0 | **Zeroes:** This field must be programmed to 0000b to put the base address on a 16-byte boundary. |

### 9.3.42 auxBufferStride

This register specified the memory type and stride of the auxiliary buffer.

| Bit | Description |
|---|---|
| 31:16 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 15 | **Memory Type:** If this bit is 0, the auxiliary buffer is linear. If this bit is '1', the auxiliary buffer is linear. |
| 14 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 13:0 | **Stride:** The meaning of this bit depends on bit 15. If bit 15 is zero (linear memory), this field specifies the linear stride of the auxiliary buffer in bytes. The stride must be an integer multiple of 16 bytes; bits 3:0 must be programmed to zero. If bit 15 is one (tiled memory) bits [6:0] of this field specifies the tile stride in tiles. Bits [13:7] are reserved in this case and must be programmed to zero for upward compatibility. |

### 9.3.43 clipLeftRight

This register is used with clipLowYHighY to define a rectangle within which all drawing operations are confined. If a pixel is generated outside the clip rectangle, it will not be drawn in the RGB or depth buffers. The values in the clipping registers are specified in pixel units. The drawing rectangle is inclusive of the clipLeft and clipLowY register values, and exclusive of the clipRight and chipHighY values.
The clipping rectangle specified with this set of registers is enabled by programming fbzMode[0] to '1'. When clipping is enabled, the clipping rectangle must always be less than or equal the screen resolution to clip to screen coordinates.
This set of registers is also used to define the rectangular region to be drawn during a FASTFILL command. In this case, the Y origin is defined by fbzMode[17].

| Bit | Description |
|---|---|
| 31:28 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 27:16 | **Left Clipping Edge:** This field is an unsigned integer that specifies the left edge of the clipping rectangle. |
| 15:12 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 11:0 | **Right Clipping Edge:** This field is an unsigned integer that specifies the right edge of the clipping rectangle. |

### 9.3.44 clipLowYHighY

This register is used with clipLeftRight to define a clipping rectangle.

| Bit | Description |
|---|---|
| 31:28 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 27:16 | **Low Y Clipping Edge:** This field is an unsigned integer that specifies the low Y (nearer the origin) edge of the clipping rectangle. For clipping, the origin is always at the top of the screen, regardless of the setting of the two origin bits (fbzMode[17] and lfbMode[13}). If the Y origin is defined to be at the bottom of the screen, the value of clipLowYHighY is usually set as the number of scan lines in the monitor resolution minus the desired Y clipping values. |
| 15:12 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 11:0 | **High Y Clipping Edge:** This field is an unsigned integer that specifies the high Y (further from the origin) edge of the clipping rectangle. |

### 9.3.45 fogTable

The fogTable is used to implement fogging in Voodoo3. This is a 64-entry lookup table with each entry consisting of eight-bit fog blending factors and eight-bit delta fog blending values. A total of 32 32-bit PCI writes are required to load the entire fogTable. When fogging is enabled, this table is addressed with the high order bits of 1/W.

| Bit | Description |
|---|---|
| 31:24 | **FogTable(2n+1) Fog Blending Factor:** This field specifies one eight-bit fog blending factor. This is stored in 8.0 format. |
| 23:16 | **Fog Table[2n+1) Delta Fog Blending Factor:** This field specifies one eight-bit delta fog blending factor. This is stored in 6.2 format. For most applications, the low order two bits will be 00b, with the six most significant bits representing the difference between successive fog blending factors. The 6.2 format means that the difference between successive fog blending factors cannot exceed 63. The sum of each fog blending factor and delta fog blending factor must not exceed 255. |
| 15:8 | **FogTable(2n) Fog Blending Factor:** This field specifies one eight-bit fog blending factor. This is stored in 8.0 format. |
| 7:0 | **Fog Table[2n) Delta Fog Blending Factor:** This field specifies one eight-bit delta fog blending factor. This is stored in 6.2 format. |

### 9.3.46 fbiCounters

These five read-only registers are intended for gathering statistics for performance tuning of drivers. Each register is a 24-bit counter which is incremented as indicated in the table. All five are cleared to 0 on power-up reset, or if '1' is written to nopCMD[0]. The table shows the addresses and when each counter is incremented.

| Counter | Address | Incremented for each pixel that is | Link |
|---|---|---|---|
| fbiPixelsIn | 0x020 014C | processed by the triangle walking engine | - |
| fbiChromaFail | 0x020 0150 | invalidated due to chroma-key color match | Section 10.4.3 |
| fbiZfuncFail | 0x020 0154 | invalidated due to failure in Z test | Section 10.4.2 |
| fbiAfuncFail | 0x020 0158 | invalidated due to failure in alpha test or alpha masking test | Section 10.4.1 |
| fbiPixelsOut | 0x020 015C | written into a color buffer. | - |

| Bit | Description |
|---|---|
| 31:24 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 23:0 | **Counter:** This read-only field returns the respective pixel counter. |

### 9.3.47 clipLeftRight1

This register, in conjunction with clipTopBottom1, defines a second clipping rectangle. The rectangle defined by this set may be inclusive or exclusive. An inclusive rectangle allows drawing within itself and an exclusive rectangle disallows drawing within itself.

| Bit | Description |
|---|---|
| 31 | **Clip Enable:** When this bit is '1', the secondary clipping rectangle is enabled. |
| 30:28 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 27:16 | **Left Clipping Edge:** This field is an unsigned integer that specifies the left edge of the clipping rectangle. |
| 15:12 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 11:0 | **Right Clipping Edge:** This field is an unsigned integer that specifies the right edge of the clipping rectangle. |

### 9.3.48 clipTopBottom1

This register is used with clipLeftRight1 to define the secondary clipping rectangle.

| Bit | Description |
|---|---|
| 31 | **Clip Mode:** If this bit is 0, the secondary clipping rectangle is inclusive. It will allow writing within itself. If this bit is 1, the secondary clipping rectangle is exclusive. It will disallow writing within itself. |
| 30:28 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 27:16 | **Low Y Clipping Edge:** This field is an unsigned integer that specifies the low Y (nearer the origin) edge of the clipping rectangle. For clipping, the origin is always at the top of the screen, regardless of the setting of the two origin bits (fbzMode[17] and lfbMode[13}). If the Y origin is defined to be at the bottom of the screen, the value of clipLowYHighY is usually set as the number of scan lines in the monitor resolution minus the desired Y clipping values. |
| 15:12 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 11:0 | **High Y Clipping Edge:** This field is an unsigned integer that specifies the high Y (further from the origin) edge of the clipping rectangle. |

### 9.3.49 swapBufferPend

Writes to this register increment the swap buffer pending count field in the status register. Writes take effect immediately (they do not flow through the FIFO) and are available only through direct access.

| Bit | Description |
|---|---|
| 31:0 | **Reserved:** The data are ignored; it is the address decode that is important here. |

### 9.3.50 leftOverlayBuf

This register specifies the starting address of the left or Monocular buffer for overlay display (also called overlay surface 0). This register is sampled at the end of vertical retrace.

| Bit | Description |
|---|---|
| 31:28 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 27:0 | **Overlay Surface Buffer 0:** This field specifies the starting address of the left buffer. For video overlays, this address must be aligned on a 32-bit boundary for YUV 4:2:2 pixel format and a 64-bit boundary for YUV 4:1:1 pixel format. If the overlay surface resides in linear space, this is the physical address. |

### 9.3.51 rightOverlayBuf

his register specifies the starting address of the right buffer for overlay display (also called overlay surface 1). This register is sampled at the end of vertical retrace.

| Bit | Description |
|---|---|
| 31:28 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 27:0 | **Overlay Surface Buffer 1:** This field specifies the starting address of the right buffer. For video overlays, this address must be aligned on a 32-bit boundary for YUV 4:2:2 pixel format and a 64-bit boundary for YUV 4:1:1 pixel format. If the overlay surface resides in linear space, this is the physical address. |

### 9.3.52 fbiSwapHistory

This read-only register maintains a count of the number of vertical syncs that occur between executed swap commands. This information is kept for the last eight executed swap commands. Each time a swap command is competed, bits [27:0] of this register are shifted left by four bit positions (into [31:4]). Then bits 3:0 are loaded with the number of vertical syncs that occurred between the previous swap command and the just executed swap command. If more than 15 syncs occurred, the value 15 will be loaded.

| Bit | Description |
|---|---|
| 31:28 | **Vertical Sync SwapBuffer History for Frame n-7:** This read-only field returns the count of vertical syncs for frame n-7. This is the oldest entry and will be lost when the next swap command completes. |
| 27:24 | **Vertical Sync SwapBuffer History for Frame n-6:** |
| 23:20 | **Vertical Sync SwapBuffer History for Frame n-5:** |
| 19:16 | **Vertical Sync SwapBuffer History for Frame n-4:** |
| 15:12 | **Vertical Sync SwapBuffer History for Frame n-3:** |
| 11:8 | **Vertical Sync SwapBuffer History for Frame n-2:** |
| 7:4 | **Vertical Sync SwapBuffer History for Frame n-1:** |
| 3:0 | **Vertical Sync SwapBuffer History for Frame n:** This read-only field returns the count of vertical syncs that occurred between the previous swap command and the just executed swap command, or the value 1111b, whichever is less. |

### 9.3.53 fbiTrianglesOut

This read-only register returns a count of triangles processed by the triangle walking engine. This register is cleared to 0 on power-up reset and whenever ‘1’ is written to nopCMD[1].

| Bit | Description |
|---|---|
| 31:24 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 23:0 | **TrianglesOut:** This read-only field returns a count of triangles processed by the Voodoo3 triangle walking engine. Triangles that are backface culled in the triangle setup unit do not increment this value. This value is reset to 0 by power-up reset and when the value ‘1’ is written nopCMD[1]. |

### 9.3.54 sSetupMode

This register contains bits that specify whether parameters will be calculated in the setup process. A '1' indicates the parameter will be calculated. A 0 indicates the value is not passed down to the triangle and the previous value will be used.

| Bit | Description |
|---|---|
| 31:20 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 19 | **Disable PingPong Sign Correction:** When this bit is '1', it disables the ping pong sign inversion that normally takes place during triangle strips. |
| 18 | **Culling Sign:** When this bit is 0, the culling sign is positive. When this bit is '1', the culling sign is negative. |
| 17 | **Enable Culling:** When this bit is '1', culling is enabled. |
| 16 | **Strip Mode:** When this bit is '1', the triangle strip mode is fan. See for a description of triangle fans and strips. When this bit is 0, the triangle strip mode is strip. |
| 15:8 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 7 | **Setup S1 and T1:** When this bit is 0, the respective parameter(s) will not be passed down to the triangle; the previous value(s) will be used. |
| 6 | **Setup W1:** When this bit is 0, the respective parameter(s) will not be passed down to the triangle; the previous value(s) will be used. |
| 5 | **Setup S0 and T0:** When this bit is 0, the respective parameter(s) will not be passed down to the triangle; the previous value(s) will be used. |
| 4 | **Setup W0:** When this bit is 0, the respective parameter(s) will not be passed down to the triangle; the previous value(s) will be used. |
| 3 | **Setup Wb**: When this bit is 0, the respective parameter(s) will not be passed down to the triangle; the previous value(s) will be used. |
| 2 | **Setup Z:** When this bit is 0, the respective parameter(s) will not be passed down to the triangle; the previous value(s) will be used. |
| 1 | **Setup Alpha:** When this bit is 0, the respective parameter(s) will not be passed down to the triangle; the previous value(s) will be used. |
| 0 | **Setup Red, Green, and Blue:** When this bit is 0, the respective parameter(s) will not be passed down to the triangle; the previous value(s) will be used. |

### 9.3.55 TriangleSetupVertex

This set of registers specify vertices of a triangle strip to be rendered. Triangle strips (and fans) require fewer parameters transfers since vertices are re-used. shows the addresses and descriptions of the setupvertex registers. These are all IEEE 32-bit single-precision floating point numbers except for the sARGB register, which is a packed 32-bit color with alpha in [31:24].

**Table 9.5 Triangle Setup Vertex Registers**

| Register | Description | Address |
|---|---|---|
| sVx | X component of vertex coordinate | 0x020 0264 |
| sVy | Y component of vertex coordinate | 0x020 0268 |
| sARGB | Color of current vertex in 8:8:8:8 format | 0x020 026C |
| sRed | Separated red value | 0x020 0270 |
| sGreen | Separated green value | 0x020 0274 |
| sBlue | Separated blue value | 0x020 0278 |
| sAlpha | Separated alpha value | 0x020 027C |
| sVz | Z value | 0x020 0280 |
| sWb | Global 1/W | 0x020 0284 |
| sWtmu0 | 1/W for all TMU0's, for all tmus | 0x020 0288 |
| sS/W0 | S coordinate divided by W, for all tmus | 0x020 028C |
| sT/W0 | T coordinate divided by W, for all tmus | 0x020 0290 |
| sWtmu1 | TMU1's local 1/W | 0x020 0294 |
| sS/Wtmu1 | TMU1's local S/W | 0x020 0298 |
| sT/Wtmu1 | TMU1's local T/W | 0x020 029C |

### 9.3.56 sDrawTriCMD

A write to this register starts the draw process. See

| Bit | Description |
|---|---|
| 30:1 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 0 | **Draw Triangle:** This bit must be written as '1' to actually initiate the process. |

### 9.3.57 sBeginTricMD

A write to the register begins a new triangle strip starting with the current vertex. No drawing is actually done until a write to sDrawTriCMD is executed.

| Bit | Description |
|---|---|
| 30:1 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 0 | **Begin New Triangle:** This bit must be written as '1' to initiate a new strip. |

### 9.3.58 textureMode

This register controls texture mapping including perspective correction, texture filtering, texture clamping, and multiple texture blending.

| Bit | Description |
|---|---|
| 31 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 30 | **Trilinear:** When this bit is '1', Trilinear texture mapping is enabled. When this bit is 0, point-sampled or bilinear texture mapping is enabled. |
| 29 | **tca_invert_output:** This bit controls the output inverter in the Texture Alpha Combine Unit. A '1' inverts the output. |
| 28 | **tca_add_a_local:** If this bit is '1', a_local is added in the Texture Alpha Combine Unit. This bit must not be '1' simultaneously with bit 27. |
| 27 | **tca_add_c_local:** If this bit is '1', c_local is added in the Texture Alpha Combine Unit. This bit must not be '1' simultaneously with bit 28. |
| 26 | **tca_reverse_blend:** This bit controls the sense of the blending in the Texture Alpha Combine Unit. The value '1', enables reverse blend. |
| 25:23 | **tca_mselect:** This field controls the multiplexor in the Texture Alpha Combine Unit according to the table. Values not in this table must not be programmed into this field. |

| Value | Selection |
|---|---|
| 000b | Constant zero |
| 001b | c_local |
| 010b | a_other |
| 011b | a_local |
| 100b | LOD (Level Of Detail) |
| 101b | LOD_fraction |

| | |
|---|---|
| 22 | **tca_sub_c_local:** If this bit is '1', c_local is subtracted in the Texture Alpha Combine Unit. |
| 21 | **tca_zero_other:** If this bit is '1', constant zero is substituted for c_other in the Texture Alpha Combine Unit. |

### 9.3.58 textureMode *(cont)*

| Bit | Description |
|---|---|
| 20 | **tc_invert_output:** This bit controls the output inverter in the Texture Color Combine Units. A '1' inverts the output. |
| 19 | **tc_add_a_local:** If this bit is '1', a_local is added in the Texture Color Combine Units. This bit must not be '1' simultaneously with bit 18. |
| 18 | **tc_add_c_local:** If this bit is '1', c_local is added in the Texture Color Combine Units. This bit must not be '1' simultaneously with bit 19. |
| 17 | **tc_reverse_blend:** This bit controls the sense of the blending in the Texture Color Combine Units. The value '1', enables reverse blend. |
| 16:14 | **tc_mselect:** This field controls the multiplexor in the Texture Color Combine Units according to the table. Values not in this table must not be programmed into this field. |

| Value | Selection |
|---|---|
| 000b | Constant zero |
| 001b | c_local |
| 010b | a_other |
| 011b | a_local |
| 100b | LOD (Level Of Detail) |
| 101b | LOD_fraction |

| | |
|---|---|
| 13 | **tc_sub_c_local:** If this bit is '1', c_local is subtracted in the Texture Color Combine Units. |
| 12 | **tc_zero_other:** If this bit is '1', constant zero is substituted for c_other in the Texture Color Combine Units. |
| 11:8 | **TFormat:** This field specifies the texture format accessed by the TREX. See Table 10.19. |
| 7 | **tclampt:** When this bit is '1', the T interator is clamped to [0, texture height]. When this bit is 0, T coordinates outside the range [0, texture height] are allowed to wrap into the [0. texture height] range using bit truncation. |
| 6 | **tclamps:** When this bit is '1', the S interator is clamped to [0, texture width]. When this bit is 0, S coordinates outside the range [0, texture width] are allowed to wrap into the [0. texture width] range using bit truncation. |
| 5 | **tnccselect:** This bit specifies the NCC lookup table to be used when decompressing 8-bit NCC textures. 0 selects nccTable0 and '1' selects nccTable1. |
| 4 | **tloddither:** When this bit is 1, LOD (Level-of-Detail) dithering is enabled. This is useful when performing texture mipmapping to remove the LOD bands that can occur with mipmapping without trilinear filtering. This adds an average of 3/8 (0.375) to the LOD value and must be compensated for in the amount of lodbias. |

### 9.3.58 textureMode *(cont)*

| Bit | Description |
|---|---|
| 3 | **tclampw:** When this bit is '1', S and T are each forced to zero when W is negative. This is used when projecting textures to avoid projecting behind the source of the projection. This is usually desirable but not necessary when doing projected textures. |
| 2 | **tmagfilter:** This bit specifies the texture magnification filter. It is used when LOD is less than LODmin. When this bit is 0, point-sampled filtering is selected. When this bit is 1, bilinear filtering is selected. |
| 1 | **tminfilter:** This bit specifies the texture magnification filter. It is used when LOD is equal to or greater than LODmin. When this bit is 0, point-sampled filtering is selected. When this bit is 1, bilinear filtering is selected. |
| 0 | **Enable Perspective Correction:** When this bit is '1', perspective correction is enabled for the S and T interators. There is no performance penalty for performing perspective corrected texture mapping. When this bit is 0, W is forced to 1.0. |

### 9.3.59 tLOD

This register control the texture mapping LOD calculations.

| Bit | Description |
|---|---|
| 31:30 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 29 | **tmirrort:** When this bit is '1', mirror texture in the T dimension. |
| 28 | **tmirrors:** When this bit is '1', mirror texture in the S dimension. |
| 27 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. This bit was used for tdirect_write in Voodoo graphics. |
| 26 | **tdata_swap:** When this bit is '1', incoming texture data is short swapped. The high order 16 bits of each 32-bit texture DWORD are swapped with the low order 16 bits. |
| 25 | **tdata_swizzle:** When this bit is '1', incoming texture data is byte swapped. Bits [31:24] are swapped with bits [7:0] and bits [23:16] are swapped with bits [15:8]. |
| 24 | **tmultibaseaddr:** When this bit is '1' multiple texbaseAddr registers are used. |
| 23 | **lod_zerofrac:** When this bit is '1', the factional part is forced to 0. This is useful for bilinear when the even and odd levels are split across two TREXs. |
| 22:21 | **lod_aspect:** This field specifies the aspect ration, according to the table. |

| lod_aspect | Aspect Ratio |
|---|---|
| 00b | square texture |
| 01b | 2x1 / 1x2 |
| 10b | 4x1 / 1x4 |
| 11b | 8x1 / 1x8 |

### 9.3.59 tLOD *(cont)*

| Bit | Description |
|---|---|
| 20 | **lod_s_is_wider:** This bit specifies which dimension is wider for rectilinear texture maps. When this bit is 1, is specifies that S is wider than T. When this bit is 0, it specifies the T is wider than S. This is a don't care for square texture maps. |
| 19 | **lod_tsplit:** When this bit is '1', it specifies that the texture is split; bit 18 indicates whether is it odd or even levels. When this bit is 0, it specifies the texture contains all LOD levels. |
| 18 | **lod_odd:** This bit is used only when bit 19 is '1'. When this bit is '1', it indicates odd levels. When this bit is 0, it indicates even levels. |
| 17:12 | **lodbias:** This field specifies the LOD Bias. This is a 4.2 signed format. This value is added to the calculated LOD, and the result is clamped to the range [lodmin, min[8.0,lodmax]}. That is, the result cannot go more negative than lodmin, nor can it go more positive than lodmax or the constant 8.0, whichever is smaller. |
| 11:6 | **lodmax:** This field specifies the maximum LOD. This is a 4.2 signed format. |
| 5:0 | **lodmin:** This field specifies the minimum LOD. This is a 4.2 signed format. |

### 9.3.60 tDetail

This register controls the detail texture.

| Bit | Description |
|---|---|
| 31:22 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 21 | **rgb_a_separate_filter:** When this bit is '1', rgb_tminfilter and rgb_tmagfilter define the filter for RGB and a_tminfilter and a_tmagfilter define the filter for alpha. When this bit is 0, tminfilter and tmaxfilter of textureMode define the filter for RGBA. |
| 20 | **a_tmagfilter:** This is the alpha texture magnification filter. '1' specifies bilinear. 0 specifies point-sampled. |
| 19 | **a_tminfilter:** This is the alpha texture minification filter. '1' specifies bilinear. 0 specifies point-sampled. |
| 18 | **rgb_tmagfilter:** This is the RGB texture magnification filter. '1' specifies bilinear. 0 specifies point-sampled. |
| 17 | **rgb_tmagfilter:** This is the RGB texture minification filter. '1' specifies bilinear. 0 specifies point-sampled. |
| 16:14 | **detail_scale:** This field specifies the texture scale shift left. |
| 13:8 | **detail_bias:** This field specifies the texture bias. This is a 6.0 signed format. |
| 7:0 | **detail_max:** This field specifies the texture LOD clamp. This is an 8.0 unsigned format. |

### 9.3.61 texBaseAddr

These registers specify the starting texture memory addresses. There are four registers, selected by tmultibaseaddr and LODBI. The register addresses are shown in the following table.

| Register | Address | Selected when |
|---|---|---|
| texBaseAddr | 0x020 030C | tmultibaseaddr = 0 or LODBI = 0 |
| texBaseAddr1 | 0x020 0310 | tmultibaseaddr = 1 and LODBI = 1 |
| texBaseAddr2 | 0x020 0314 | tmultibaseaddr = 1 and LODBI = 2 |
| texBaseAddr38 | 0x020 0318 | tmultibaseaddr = 1 and LODBI >= 3 |

| Bit | Description |
|---|---|
| 31:25 | **texstride:** This field specifies the tile stride (in the range 0 to 127). This field exists in texBaseAddr only. It is reserved in the other three registers and must be written as 0 for upward compatibility. |
| 24 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 23:4 | **texbaseaddr:** This field specifies the beginning address of the respective texture memory. This address must be aligned on a 16-byte boundary. Bits 3:0 must be written as 0, except for bit zero of texBaseAddr. |
| 3:1 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 0 | **texmemtype:** If this bit is '1', the texture is in tiled memory. If this bit is 0, the texture is in linear memory. |

### 9.3.62 trexInit0/1

trexInit0 does not exist in Voodoo3. trexInit1 is used for hardware initialization of configuration of the TREX portion of Voodoo3.

| Bit | Description |
|---|---|
| 31 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 30 | **always_4texel_needed:** When this bit is '1', it indicates that four texels are required for every pixel. The default value for this bit is 0. |
| 29 | **always_cache_inv:** When this bit is '1', it indicates always cache invalidate each triangle. The default value for this bit is 0. |
| 28 | **nop_per_tri:** When this bit is '1', it indicates insert a nop for every triangle. The default value for this bit is 0. |
| 27 | **a_attr_set_only:** When this bit is '1', it indicates use only the A set of triangle attributes. The default value for this bit is 0. |
| 26 | **use_4bit_st_frac:** When this bit is '1', it indicates use four bits for S and T instead of eight. The default value for this bit is 0. |
| 25:23 | **send_config_sel:** This field specifies which data to transmit to FBI when send_config is '1'. Values not in the table must not programmed into this field. |

| send_config_sel | Description |
|---|---|
| 011b | trexInit1 |
| 100b | texBaseAddr[31:0] (32 bits are retained, non-maskable) |

| | |
|---|---|
| 22:21 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 20 | **reset_graphics:** When this bit is '1', it resets all of the graphics inside the TREX. When this bit is 0, the TREX functions normally. |
| 19 | **reset_FIFOs:** When this bit is '1', it resets all of the FIFOs inside the TREX. When this bit is 0, the FIFOs functions normally. |
| 18 | **send_config:** When this bit is '1', it transmits configuration data (as selected in bits [25:23]) to FBI through the tf_interface. |
| 17 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 16 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 15:12 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 11 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 10:7 | **tt_FIFO_sil:** This field is the TREX-to-TREX interface FIFO stall input level. This specifies the free space level at which the stall signal is sent back to the transmitting chip. |

### 9.3.62 trexInit0/1 *(cont)*

| Bit | Description |
|---|---|
| 6:2 | **ft_FIFO_sil:** This field is the FBI-to-TREX interface FIFO stall input level. This specifies the free space level at which the stall signal is sent back to the transmitting chip. |
| 1 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 0 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |

### 9.3.63 nccTable0/1

These two tables of twelve DWORDs each contain NCC (Narrow Channel Compression) tables used to store lookup values for compressed textures used in YIQ and AYIQ texture formats as specified in the tformat field of textureMode. There are two tables so that they can be swapped on a per-triangle basis when performing multi-pass rendering. The table to be used is selected by textureMode[5]. The table shows the address and content of each table entry.
For offsets 4 through 11 of the table0 space (Addresses 0x020 0334 through 0x020 0350) bit 31 of the data is used to steer the information into the ncc Table or into the 8-bit palette (see Section 10.6.5). If bit 31 is 0, the information goes into the I or Q entry of nccTable0. If bit 31 is '1', the information goes into the 8-bit palette.

| Offset | Table0 Address | Table1 Address | Bits | Contents |
|---|---|---|---|---|
| 0 | 0x020 0324 | 0x020 0354 | 31:0 | {Y3[7:0], Y2[7:0], Y1[7:0], Y0[7:0]} |
| 1 | 0x020 0328 | 0x020 0358 | 31:0 | {Y7[7:0], Y6[7:0], Y5[7:0], Y4[7:0]} |
| 2 | 0x020 032C | 0x020 035C | 31:0 | {Yb[7:0], Ya[7:0], Y9[7:0], Y8[7:0]} |
| 3 | 0x020 0330 | 0x020 0360 | 31:0 | {Yf[7:0], Ye[7:0], Yd[7:0], Yc0[7:0]} |
| 4 | 0x020 0334 | 0x020 0364 | 26:0 | {I0_r[8:0]. I0_g[8:0]. I0_b[8:0] |
| 5 | 0x020 0338 | 0x020 0368 | 26:0 | {I1_r[8:0]. I1_g[8:0]. I1_b[8:0] |
| 6 | 0x020 033C | 0x020 036C | 26:0 | {I2_r[8:0]. I2_g[8:0]. I2_b[8:0] |
| 7 | 0x020 0340 | 0x020 0370 | 26:0 | {I3_r[8:0]. I3_g[8:0]. 3_b[8:0] |
| 8 | 0x020 0344 | 0x020 0374 | 26:0 | {Q0_r[8:0]. Q0_g[8:0]. Q0_b[8:0] |
| 9 | 0x020 0348 | 0x020 0378 | 26:0 | {Q1_r[8:0]. Q1_g[8:0]. Q1_b[8:0] |
| 10 | 0x020 034C | 0x020 037C | 26:0 | {Q2_r[8:0]. Q2_g[8:0]. Q2_b[8:0] |
| 11 | 0x020 0350 | 0x020 0380 | 26:0 | {Q3_r[8:0]. Q3_g[8:0]. Q3_b[8:0] |

# 10 3D Programming Notes

## 10.1 Introduction

We cover 3D programming in this chapter. Ultimately, 3D graphics is all about drawing triangles. It is not the intent in this chapter to cover 3D concepts; the reader should know about depth buffers, alpha blending, chroma keying, and the like. Rather, this chapter will describe the functions available in Voodoo3 3D rendering engine and allow the application to use them as it will.
The formal definitions of the 3D registers are in Chapter 9.

## 10.2 Vertex Definition

There are three vertices in a Voodoo3 triangle[1]. The vertices are unique and are not colinear. The vertices are labeled A, B, and C. Vertex A, by Voodoo3 definition, is at the top of the triangle (the lowest-addressed y coordinate). The AC edge is the major edge; it spans the entire height of the triangle. This means that vertex C has the highest-addressed y coordinate. The other two edges, AB and BC, are called minor edges. Figure 10.1 illustrates the two possible triangle orientations.

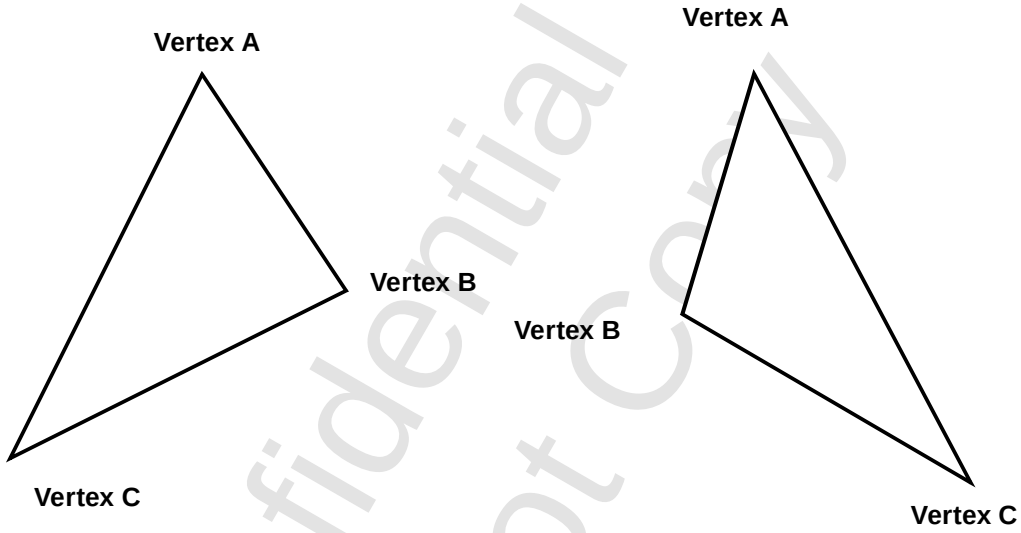



**Figure 10.1 Triangle Orientation**

The beginning vertex (vertex A) has a number of attributes, listed in Table 10.1. There are two sets of registers, one set for fixed point numbers and one set for floating point numbers. The application can use whichever format is more natural to its internal number representation. When rendering is initiated, the application will indicate which set of registers it programmed. This notion of two sets of registers applies not only to the vertex A attributes, but to most of the registers discussed in this chapter.



1. The plane figure formed by connecting three points not in a straight line by straight line segments; a three-sided polygon

### Table 10.1 Vertex A Attributes

| Attribute | Description | Format | |
|---|---|---|---|
| | | Fixed | Float |
| X | horizontal coordinate | 12.4 | IEEE |
| Y | vertical coordinate | 12.4 | IEEE |
| R | start red color | 12.12 | IEEE |
| G | start green color | 12.12 | IEEE |
| B | start blue color | 12.12 | IEEE |
| A | start alpha value | 12.12 | IEEE |
| Z | start Z value (depth) | 20.12 | IEEE |
| S | start S value (texture) | 14.18 | IEEE |
| T | start T value (texture) | 14.18 | IEEE |
| W | start W value (texture) | 2.30 | IEEE |

The other two vertices, B and C, each possess X and Y coordinates, but no color, depth, or texture information.

### Table 10.2 Vertex B/C Coordinates

| Vertex | Coordinate | Fixed | Float |
|---|---|---|---|
| B | X | 12.4 | IEEE |
| B | Y | 12.4 | IEEE |
| C | X | 12.4 | IEEE |
| C | Y | 12.4 | IEEE |

The color, depth, and texture information is calculated by the rendering engine from the starting values provided for vertex A and delta values. Two delta values are provided for each color, etc. One is the X delta and one is the Y delta. These are summarized in Table 10.3.

### Table 10.3 3D Deltas[a]

| Value | Description | Format | |
|---|---|---|---|
| | | Fixed | Float |
| dR | change in red wrt X, Y | 12.12 | IEEE |
| dG | change in green wrt X, Y | 12.12 | IEEE |

**Table 10.3 3D Deltas[a]**

| Value | Description | Format | |
|---|---|---|---|
| | | Fixed | Float |
| dB | change in blue wrt X, Y | 12.12 | IEEE |
| dA | change in alpha wrt X, Y | 12.12 | IEEE |
| dZ | change in depth wrt X, Y | 20.12 | IEEE |
| dS | change in S wrt X, Y | 14.18 | IEEE |
| dT | change in T wrt X, Y | 14.18 | IEEE |
| dW | change in W wrt X, Y | 2.30 | IEEE |

a. wrt: with respect to

When the coordinates, start values, and deltas have been set up, the application initiates rendering by writing to the appropriate triangleCMD register. The register address is chosen based on whether the fixed or floating point register set was loaded.
One additional bit of information is transmitted with the write to the triangleCMD register; this is the sign of the area. The sign indicates whether the triangle is clockwise or counter-clockwise geometrically.
When the triangleCMD register is written, the 3D engine renders the triangle.

## 10.3 Functions Controlling the Color of a Pixel

As the triangle is rendered, the 3D engine must determine the color of each pixel. The blocks that do this are described in the following sections. Texturing Mapping is covered in Section 10.6.

### 10.3.1 Color Combine Unit

Figure 10.2 and Figure 10.3 show the CCU (Color Combine Unit). This is a functional diagram that shows how control bits are applied as a pixel flows through the pipeline. The name and the register[bits] are both given for each control field. For example, rgbSelect[1:0] is in register fbzColorPath[1:0]. These diagrams should be studied with the register descriptions, especially fbzColorPath, in Section 9.3.27.

### 10.3.2 Alpha Combine Unit

Figure 10.4 and Figure 10.5 show the ACU (Alpha Combine Unit). This is a functional diagram that shows how control bits are applied as a pixel flows through the pipeline. The name and the register[bits] are both given for each control field. For example, aSelect[1:0] is in register fbzColorPath[3:2]. These diagrams should be studied with the register descriptions, especially fbzColorPath, in Section 9.3.27.

### 10.3.3 Alpha Blending

When alphaMode[4] is ‘1’, the incoming pixels are blended with destination pixels. The blending can involve both the color values and the alpha values. This should not be confused with the alpha function, covered in Section 10.4.1.
The alpha blending function for each of the three RGB color components is shown in the following equation.

$$Dnew = (S \bullet alpha) + (Dold \bullet beta)$$

where

Dnew is the new destination pixel being written into the frame buffer,
S is the new source pixel being generated,
Dold is the old (current) destination pixel about to be modified,

alpha is the source pixel alpha blending function,
beta is the destination pixel alpha blending function.

The alpha blending function for the alpha component is shown in the following equation.

$$Anew = (AS \bullet Alphad) + (Aold \bullet Betad)$$

where

Anew is the new destination alpha being written into the alpha buffer,
AS is the new source alpha being generated,
Aold is the old (current) destination alpha about to be modified,
alphad is the source alpha alpha-blending function,
betad is the destination alpha alpha-blending function.

Note that the source and destination pixels may have different associated alpha blending functions. Note further 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 alphaMode[15:8]. The alpha blending factors of the alpha component are specified in alphaMode[23:16]. Table 10.4 lists the alpha blending functions supported by Voodoo3.

### Table 10.4 Alpha Blending Functions

| Value | Description | Mnemonic |
|---|---|---|
| 0000b | Zero | AZERO |
| 0001b | Source alpha | ASCR_ALPHA |
| 0010b | Color | A_COLOR |
| 0011b | Destination alpha | ADST_ALPHA |
| 0100b | One | AONE |
| 0101b | 1 - Source alpha | AOMSRC_ALPHA |
| 0110b | 1- Color | AOM_COLOR |
| 0111b | 1 - Destination alpha | AOMDST_ALPHA |
| 1000b - 1110b | Reserved | - |
| 1111b (source alpha blending function) | MIN(Source alpha, 1 - Destination alpha) | ASATURATE |
| 1111b (destination alpha blending function) | Color before Fog Unit | A_COLOR_BEFORE_FOG |

When the value 0010b is selected as the destination alpha blending factor, the source pixel color is used as the destination blending factor. When the value 0010b is selected as the source alpha blending factor, the destination color is used as the source blending factor.
The meaning of alpha blending factor 1111b depends on whether it is being used as a source or destination alpha blending function. When the value 1111b is selected as the destination alpha blending factor, the source color before the fog unit (the ‘unfogged’ color) is used as the destination blending factor. The alpha blending function is useful for multi-pass rendering with atmospheric effects. When the value

1111b is selected as the source alpha blending factor, the alpha-saturate anti-aliasing algorithm is selected. The MIN function performs polygonal anti-aliasing for polygons that are drawn front-to-back.

### 10.3.4 Fog Unit

Figure 10.6 and Figure 10.7 show the Fog Unit. This is a functional diagram that shows how control bits are applied as a pixel flows through the pipeline.

Voodoo3 supports a 64-entry lookup table (Fog Table) to support atmospheric effects such as fog and haze. When enabled, the high order six bits of the depth ($1/W^1$) is used to index into the 64-entry fog table. The output of the lookup table is an 'alpha' value which represents the level of blending to be performed between the static fog-haze color (fogColor) and the incoming pixel color. Eight lower order bits of the floating point 1/W are used to blend between multiple entries of the lookup table to reduce fog 'banding'. Since the lookup table is loaded by the application, various fog equations, colors, and effects can be supported.

Table 10.5 shows the equations for the combinations of fogMode[2:1] and fogMode[0].

**Table 10.5 Fog Equations**

| fogMode[0] fogenable | fogMode[1] fogadd | fogMode[2] fogmult | Equation |
|---|---|---|---|
| 0 | n/a | n/a | 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 |



1. Specifically, a normalized floating point representation of 1/W.

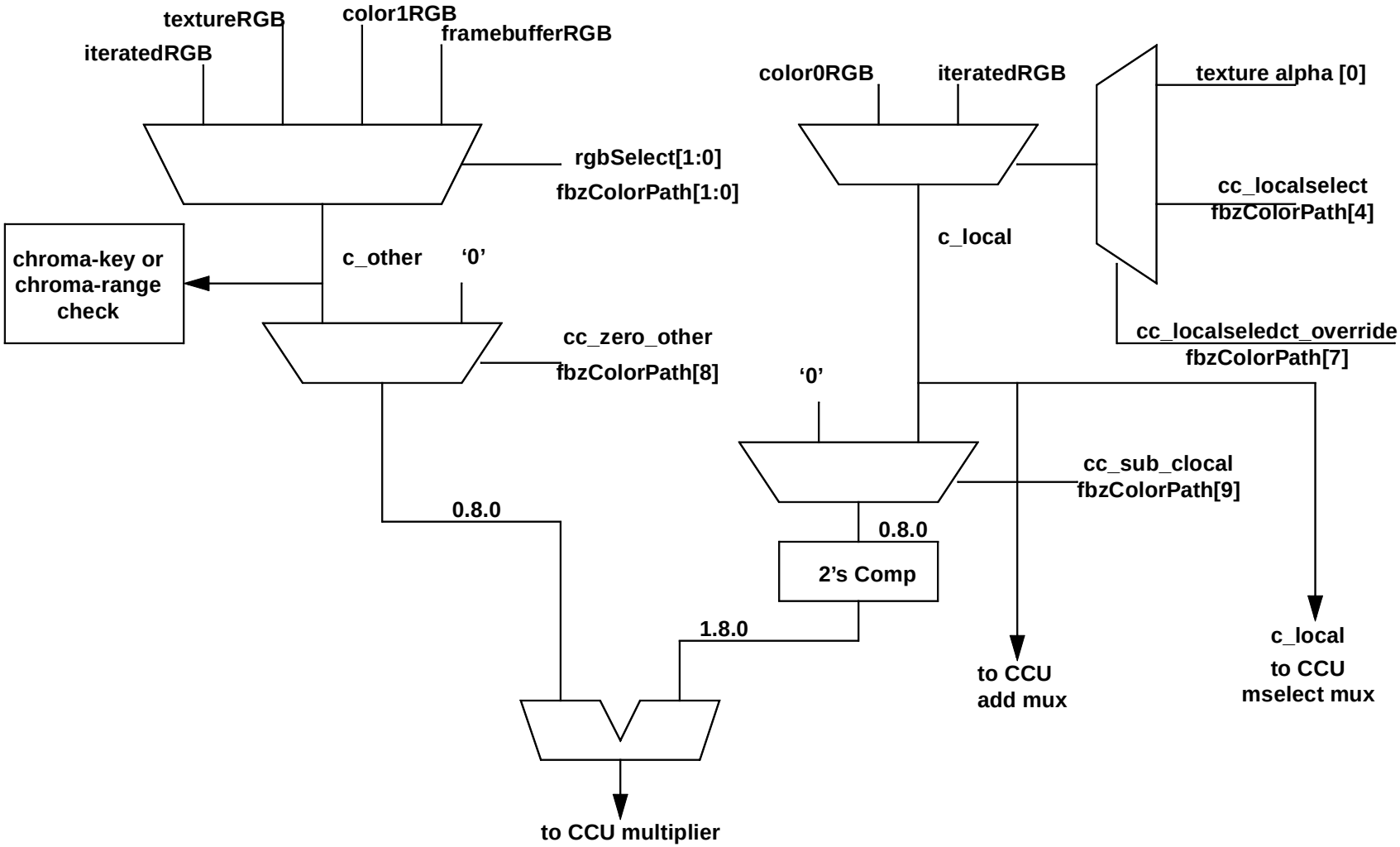



**Figure 10.2 Color Combine Unit: One of Two**

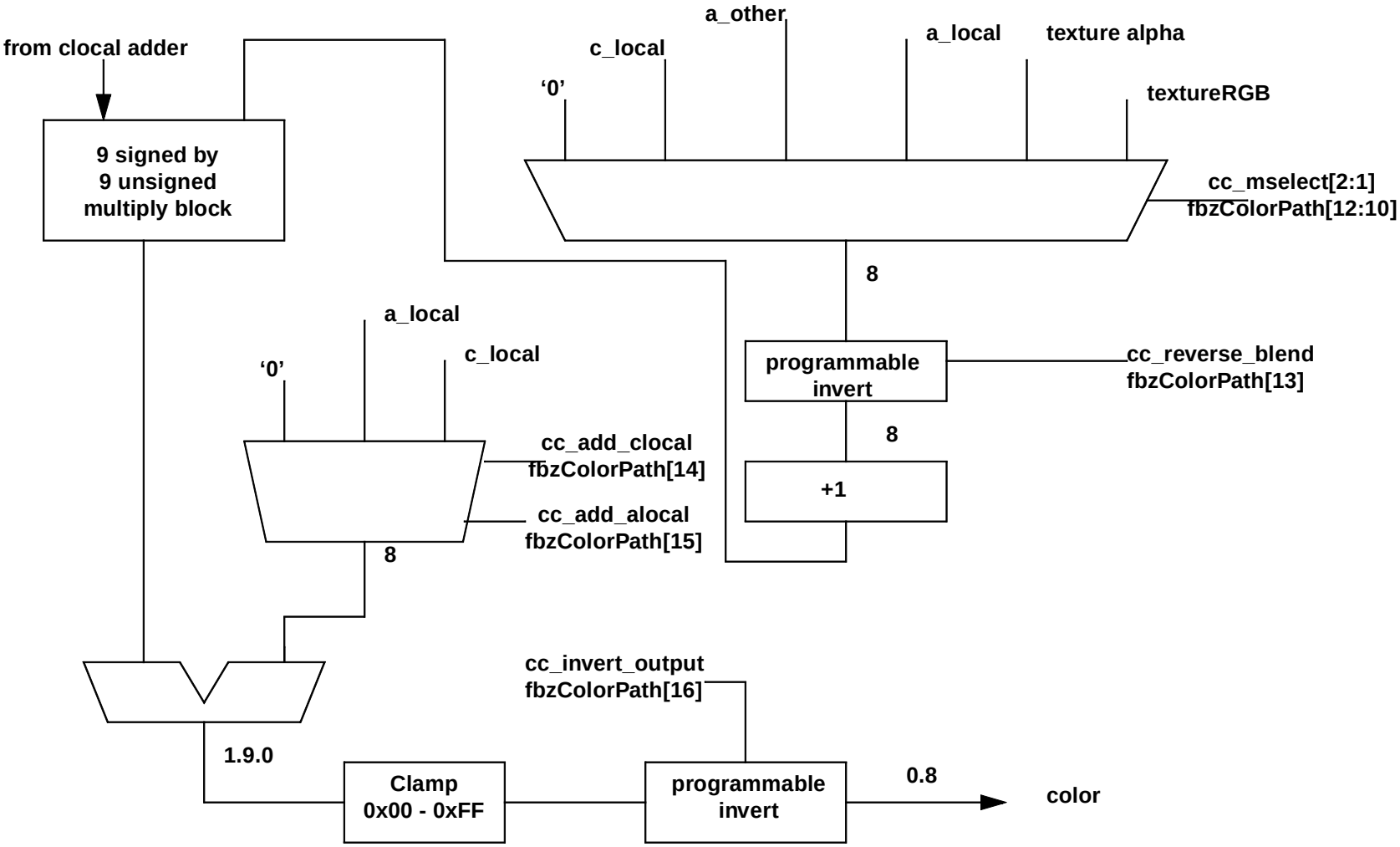



**Figure 10.3 Color Combine Unit: Two of Two**

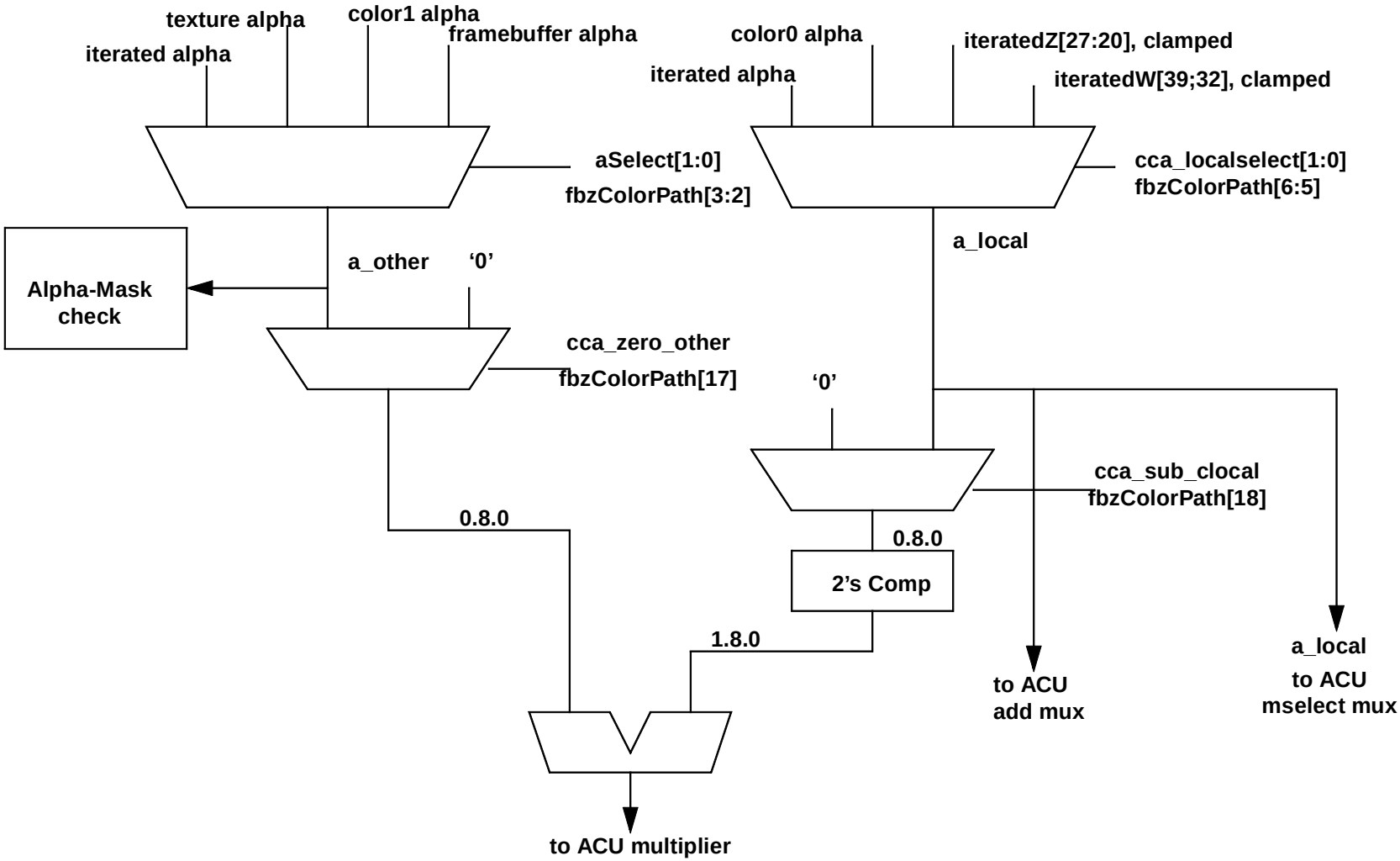



**Figure 10.4 Alpha Combine Unit: One of Two**

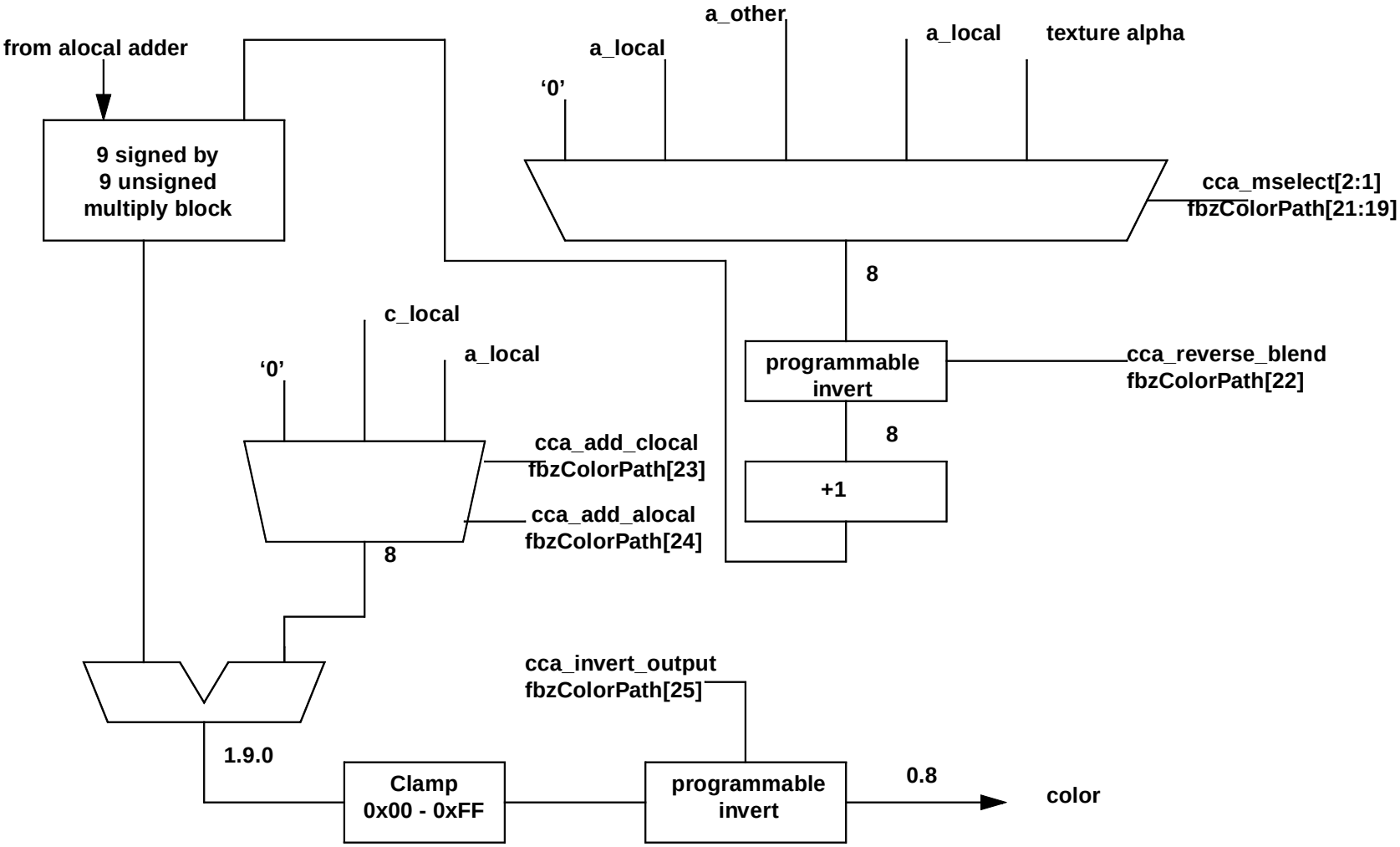



**Figure 10.5 Alpha Combine Unit: Two of Two**

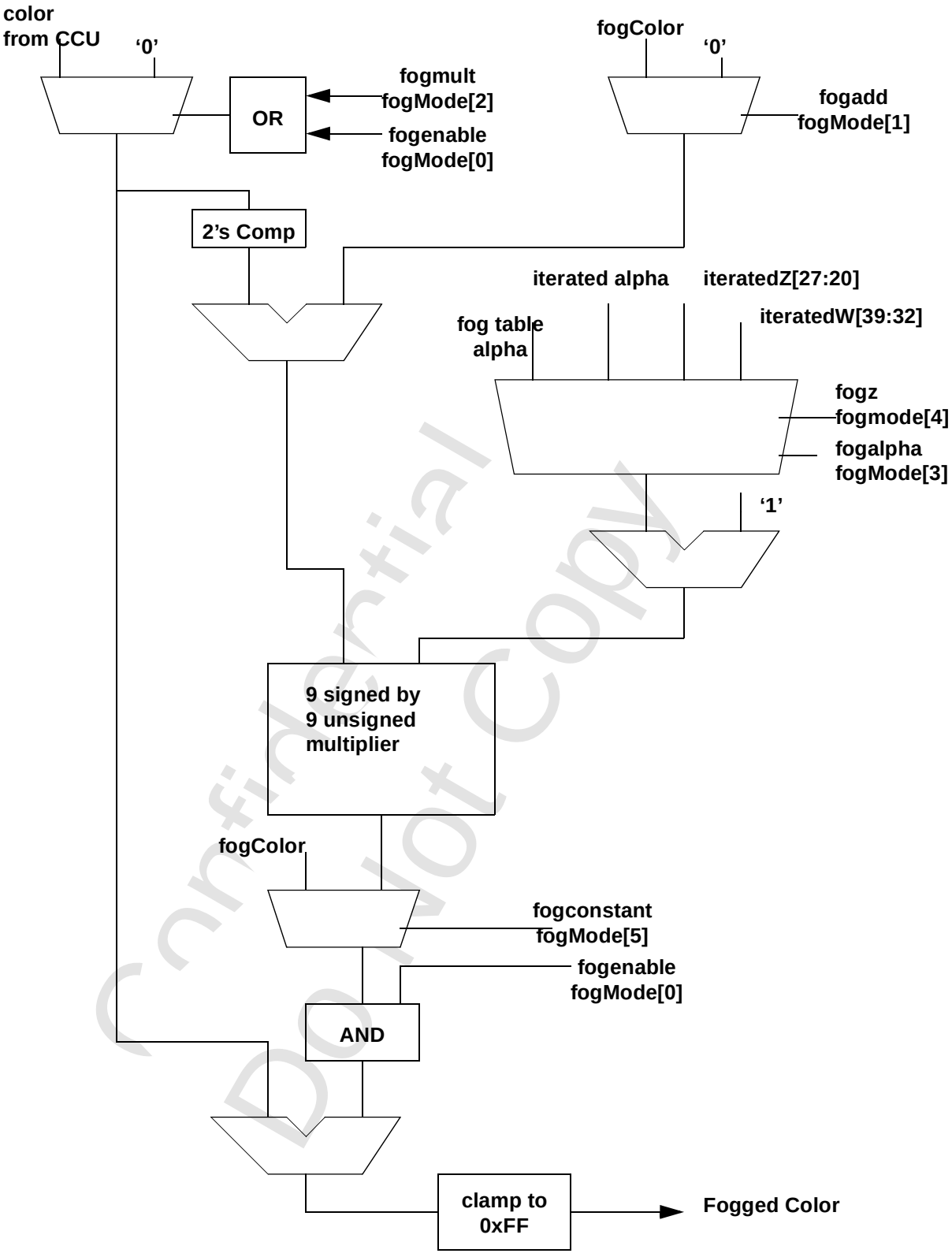



**Figure 10.6 Fog Unit: One of Two**

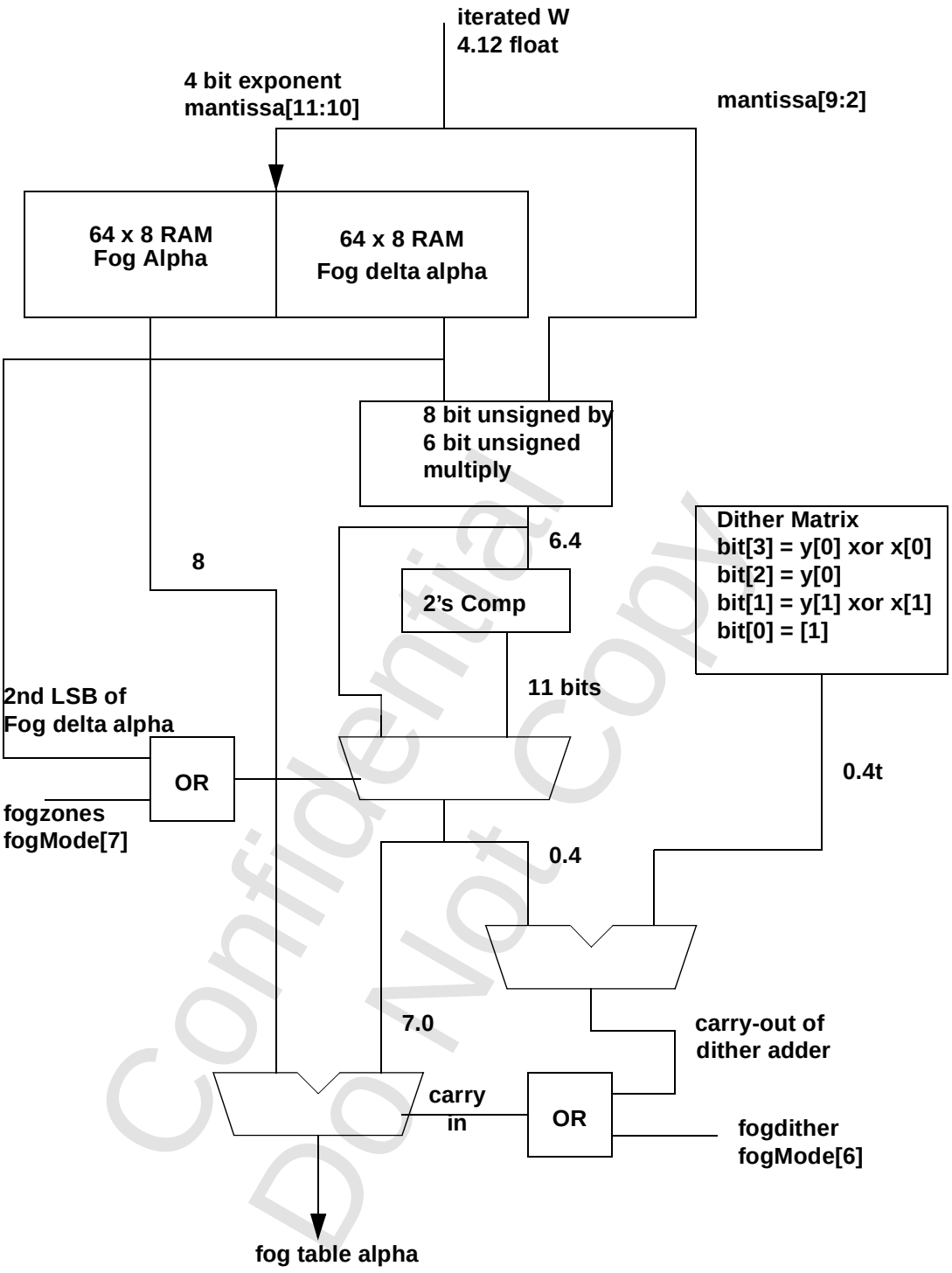



**Figure 10.7 Fog Unit: Two of Two**

## 10.4 Functions Controlling Whether Pixels are Written

### 10.4.1 Alpha Function

The alpha function can be used to selectively suppress pixels in the pipeline depending on the incoming alpha value. This should not be confused with alpha blending, described in Section 10.3.3.
When the alpha function is enabled (alphaMode[0] = '1'), the following alpha comparison is performed:

*AlphaSrc* alphaOP *AlphaRef*

where AlphaSrc is the alpha value of an incoming pixel, alphaOP is specified in alphaMode [3:1], and AlphaRef is the value contained in alphaMode[31:24]. If the alpha function is enabled (alphaMode[0]) and the alpha comparison is false, the fbiAfuncFail register is incremented and the pixel is invalidated in the pixel pipeline. The color and depth buffers are not written, regardless of whether they otherwise would have been. Table 10.6 specifies the supported alpha operations.

**Table 10.6 Alpha Op Functions**

| alphaMode[3:1] | AlphaOp Function |
|---|---|
| 000b | never |
| 001b | strictly less than |
| 010b | equal |
| 011b | less than or equal |
| 100b | strictly greater than |
| 101b | not equal |
| 110b | greater than or equal |
| 111b | always |

The alpha reference (alphaMode[31:24]) is compared with the alpha value associated with the pixel. All three magnitude comparisons (greater than, less than, equal) are done (or implied) simultaneously. If the Alpha Test is not enabled, all pixels will pass the alpha test.

### 10.4.2 Depth-Buffering Function

When the depth-buffering function is enabled (fbzMode[4] = '1'), the following depth comparison is performed:

*DEPTHsrc* depthOP *DEPTHdst*

where DEPTHsrc is the depth value of an incoming pixel, depthOP is specified fbzMode[7:5], and DEPTHdst is the destination depth value. A source pixel is written into an RGB buffer if the depth comparison is true and writing into the RGB buffer is enabled (fbzMode[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[10] = 1). The supported depth comparison functions are shown in Table 10.7.
If the pixel is not written because if fails the Z-buffer test, fbiZfuncFail is incremented.

**Table 10.7 Depth-Buffering Functions**

| fbzMode[7:5] | DepthOP Function |
|---|---|
| 000b | never |
| 001b | strictly less than |

**Table 10.7 Depth-Buffering Functions *(cont.)***

| fbzMode[7:5] | DepthOP Function |
|---|---|
| 010b | equal |
| 011b | less than or equal |
| 100b | strictly greater than |
| 101b | not equal |
| 110b | greater than or equal |
| 111b | always |

### 10.4.3 Chroma Keying

Chroma-keying monitors the color of all pixels to be written into the RGB buffer and conditionally blocks writes of pixels according to their color. Table 10.8 summarizes this function.

**Table 10.8 Chroma Keying**

| fbzMode[1] | chromaRange[28] | Description |
|---|---|---|
| 0 | don't care | chroma-key disabled |
| 1 | 0 | single color in chroma-key register |
| 1 | 1 | color range in chromaKey and chromaRange |

When chroma-range is enabled by programming chromaRange[28] to '1', the outgoing pixel is compared, color by color, with the ranges defined in chromaKey and chromaRange. For each color, the value in chromaKey must be equal to or less than the value in chromaRange (chromaKey specifies the lower value and chromaRange specifies the upper value).
For each color, the application can specify the range as being inclusive or exclusive (chromaRange[26:24]). An inclusive range block colors within the range and an exclusive range blocks color outside tof the range.
Finally, the application can specify whether any color can block a pixel or all colors must match to block the pixel. chromaRange[27] can be set to 0 for intersection mode (all color must block the pixel to prevent writing) or '1' for union mode (any color can block the pixel).
If a pixel is not written because it fails the chromakey test, fbiChromaFail is incremented.
To block colors whose blue value fall in the range 0x64 through 0x69, program the registers as shown in Table 10.9..

**Table 10.9 Chroma Keying Example**

| Field | Register | Value | Note |
|---|---|---|---|
| chroma-key blue | chromaKey 7:0 | 0x64 | lower value of blue |
| chroma-key green | chromaKey 15:8 | 0x00 | allow all shades of green |
| chroma-key red | chromaKey 23:16 | 0x00 | allow all shades of red |
| chroma-range blue | chromaRange 7:0 | 0x69 | upper value of blue to block |

**Table 10.9 Chroma Keying Example**

| Field | Register | Value | Note |
|---|---|---|---|
| chroma-range green | chromaRange 15:8 | 0xFF | allow all shades of green |
| chroma-range red | chromaRange 23:16 | 0xFF | allow all shades of red |
| chromaRange blue mode | chromaRange 24 | 1b | block inclusive |
| chromaRange green mode | chromaRange 25 | 0b | block exclusive (there are no values outside the range 0x00-0xFF |
| chromaRange red mode | chromaRange 26 | 0b | block exclusive |
| chromaRange Block mode | chromaRange 27 | 1b | any color can block |

In the example given above, it would have been just as easy to program chromaRange green mode and chromaRange red mode for block inclusive and then set chromaRange Block mode to union. In that case, green and red would block on every pixel, but since unanimity is required, only pixels blocked by blue would be not written.

### 10.4.4 Stipple Masking

When fbzMode[2] is '1', stipple masking is enabled. fbzMode[12] specifies the stipple mode. When fbzMode[12] is 0, stipple rotate mode is used. When fbzMode[12] is '1', stipple pattern mode is used. When stipple register masking is enabled and stipple rotate mode is specified, stipple[31] 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 is stipple[31] is = and stipple register masking is enabled in stipple rotate mode. After an invalidated pixel is processed in the pipeline, the stipple register is rotated to the left; bit 30 to bit 31, bit 31 to bit 0, and so on. Note that the rotation takes place regardless of whether stipple masking is enabled fbzMode[2] 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 as an index into a 4 x 8 monochrome pattern in the stipple register. The result is used to mask pixel in the pipeline. For all triangle commands and linear frame buffer writes through the pipeline, a stipple bit is selected from the stipple register as indicated in Section 10.10.

**Table 10.10 X-Y Access of stipple Register**

| Y[1:0] | X[2:0] | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 000b | 001b | 010b | 011b | 100b | 101b | 110b | 111b |
| 00b | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 01b | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| 10b | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| 11b | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |

### 10.4.5 Clipping

If a pixel is generated outside the clip rectangle, it will not be drawn into the RGB or depth buffers.
The values in the clipping registers are specified in pixel units. The drawing rectangle is inclusive of the clipLeft and clipLowY register values, and exclusive of the clipRight and chipHighY values.
The clipping rectangle is enabled by programming fbzMode[0] to '1'. When clipping is enabled, the clipping rectangle must always be less than or equal the screen resolution to clip to screen coordinates. clipLowY must be less than clipHighY and clipLeft must be less than clipTight.
The clipLowYHighY register is defined so that y=0 is always at the top of the monitor screen, regardless of the value of the Y origin bits (fbzMode[17] and lfbMode[13]). If the Y origin is defined to be at the bottom of the screen (by setting one of these bits), care must be taken in programming the clipLowYHighY register to obtain proper functionality. In the case where the Y origin is defined to be at the bottom on the screen, the value of clipLowYHighY is usually set to the number of scan lines in the monitor resolution minus the desired Y clipping values.

## 10.5 Linear Frame Buffer Writes

### 10.5.1 LFB Writes: Format 0

When writing to the LFB (Linear Frame Buffer) with 16-bit format 0 (RGB 5-6-5), the RGB channel format specifies the RGB ordering within a 16-bit word. If the Voodoo3 pipeline is enabled for LFB access (lfbMode[8] = '1'), then alpha and depth information for the LFB format 0 is taken from zaColor. Table 10.11 shows the color channels for 16-bit LFB access format 0.

**Table 10.11 Color Channels: 16-bit Access Format 0**

| RGB Channel Format Value | 16-it LFB Access Bits | Channel | | |
|---|---|---|---|---|
| | | Red | Green | Blue |
| 0 | 15:0 | 15:11 | 10:5 | 4:0 |
| 1 | 15:0 | 4:0 | 10:5 | 15:11 |
| 2 | 15:0 | 15:11 | 10:5 | 4:0 |
| 3 | 15:0 | 4:0 | 10:5 | 15:11 |

### 10.5.2 LFB Writes: Format 1

When writing to the LFB with 16-bit format (RGB 5-5-5), the RGB channel format specifies the RGB ordering within a 16-bit word. If the Voodoo3 pixel pipeline is enabled for LFB access (lfbMode[8] = '1'), then alpha and depth information for LFB format 1 is taken from zaColor. Table 10.12 shows the color channels for 16-bit LFB access format 1..

**Table 10.12 Color Channels: 16-bit Access Format 1**

| RGB Channel Format Value | 16-it LFB Access Bits | Channel | | | |
|---|---|---|---|---|---|
| | | (ignored) | Red | Green | Blue |
| 0 | 15:0 | 15 | 14:10 | 9:5 | 4:0 |
| 1 | 15:0 | 15 | 4:0 | 9:5 | 14:10 |
| 2 | 15:0 | 0 | 15:11 | 10:6 | 5:1 |

**Table 10.12 Color Channels: 16-bit Access Format 1**

| RGB Channel Format Value | 16-it LFB Access Bits | Channel | | | |
|---|---|---|---|---|---|
| | | (ignored) | Red | Green | Blue |
| 3 | 15:0 | 0 | 5:1 | 10:6 | 15:11 |

### 10.5.3 LFB Writes: Format 2

When writing to the LFB with 16-bit format 2 (ARGB 1-5-5-5), the RGB channel format specifies the ARGB ordering within a 16-bit word. If the Voodoo3 pixel pipeline is enabled for LFB access (lfbMode[8] = ‘1’), then alpha and depth information for LFB format 1 is taken from zaColor. Note that the one-bit alpha value passed when using LFB format 2 is replicated to yeild the eight-bit alpha used in the pixel pipeline. Table 10.13 shows the color channels for 16-bit LFB access format 2.

**Table 10.13 Color Channels: 16-bit Access Format 2**

| RGB Channel Format Value | 16-it LFB Access Bits | Channel | | | |
|---|---|---|---|---|---|
| | | Alpha | Red | Green | Blue |
| 0 | 15:0 | 15 | 14:10 | 9:5 | 4:0 |
| 1 | 15:0 | 15 | 4:0 | 9:5 | 14:10 |
| 2 | 15:0 | 0 | 15:11 | 10:6 | 5:1 |
| 3 | 15:0 | 0 | 5:1 | 10:6 | 15:11 |

### 10.5.4 LFB Writes: Format 4

When writing to the LFB with 24-bit format 4 (RGB x-8-8-8), the RGB channel format specifies the RGB ordering within a 24-bit word. Note that the alpha/A channel is eignored for 24-bit access format 4. Note also that while only 24 bits of data is transferred for format 4, all data access must be 32-bit aligned. Packed 24-bit writes are not support by Voodoo3. If the Voodoo3 pixel pipeline is enabled for LFB access (lfbMode[8] = ‘1’), then alpha and depth information for LFB format 4 is taken from zaColor. Table 10.14 shows the color channels for 16-bit LFB access format 1..

**Table 10.14 Color Channels: 16-bit Access Format 4**

| RGB Channel Format Value | 16-it LFB Access Bits | Channel | | | |
|---|---|---|---|---|---|
| | | (ignored) | Red | Green | Blue |
| 0 | 31:0 | 31:24 | 23:16 | 15:8 | 7:0 |
| 1 | 31:0 | 31:24 | 7:0 | 15:8 | 23:16 |
| 2 | 31:0 | 7:0 | 31:24 | 23:16 | 15:8 |
| 3 | 31:0 | 7:0 | 15:8 | 23:16 | 31:24 |

### 10.5.5 LFB Writes: Format 5

When writing to the LFB with 32-bit format 5(ARGB 8-8-8-8), the RGB channel format specifies the ARGB ordering within a 32-bit word. If the Voodoo3 pixel pipeline is enabled for LFB access (lfbMode[8] = ‘1’), then alpha and depth information for LFB format 5is taken from zaColor. Table 10.15 shows the color channels for 32-bit LFB access format 5..

**Table 10.15 Color Channels: 16-bit Access Format 5**

| RGB Channel Format Value | 16-it LFB Access Bits | Channel | | | |
|---|---|---|---|---|---|
| | | Alpha | Red | Green | Blue |
| 0 | 31:0 | 31:24 | 23:16 | 15:8 | 7:0 |
| 1 | 31:0 | 31:24 | 7:0 | 15:8 | 23:16 |
| 2 | 31:0 | 7:0 | 31:24 | 23:16 | 15:8 |
| 3 | 31:0 | 7:0 | 15:8 | 23:16 | 31:24 |

### 10.5.6 LFB Writes: Format 12

When writing to the LFB with 32-bit format 12(ZRGB 16-5-6-6), the RGB channel format specifies the RGB ordering within a 32-bit word. If the Voodoo3 pixel pipeline is enabled for LFB access (lfbMode[8] = ‘1’), then alpha information for LFB format 12 is taken from zaColor. Note that the format of the depth value passed when using LFB format 12 must precisely match the format of the type of depth buffer being used (either 16-bit integer Z or 16-bit floating point 1/W). Table 10.16 shows the 16-bit color channels within the 32-bit linear frame buffer access format 12..

**Table 10.16 Color Channels: 16-bit Access Format 12**

| RGB Channel Format Value | 16-it LFB Access Bits | RGB Channel | | |
|---|---|---|---|---|
| | | Red | Green | Blue |
| 0 | 15:0 | 15:11 | 10:5 | 4:0 |
| 1 | 15:0 | 4:0 | 10:5 | 15:11 |
| 2 | 15:0 | 15:11 | 10:5 | 4:0 |
| 3 | 15:0 | 4:0 | 10:5 | 15:11 |

### 10.5.7 LFB Writes: Format 13

When writing to the LFB with 32-bit format 13(ZRGB 16-x-5-5-5), the RGB channel format specifies the RGB ordering within a 16-bit word. If the Voodoo3 pixel pipeline is enabled for LFB access (lfbMode[8] = ‘1’), then alpha information for LFB format 13 is taken from zaColor. Note that the format of the depth value passed when using LFB format 13 must preciesly match the format of the type of dpeth buffering being used (either 16-bit integer Z or 16-bit floating point 1/W). Table 10.17 shows the 16-bit color channels

within the 32-bit LFB access format 13..

**Table 10.17 Color Channels: 16-bit Access Format 13**

| RGB Channel Format Value | 16-it LFB Access Bits | Channel | | | |
|---|---|---|---|---|---|
| | | (ignored) | Red | Green | Blue |
| 0 | 15:0 | 15 | 14:10 | 9:5 | 4:0 |
| 1 | 15:0 | 15 | 4:0 | 9:5 | 14:10 |
| 2 | 15:0 | 0 | 15:11 | 10:6 | 5:1 |
| 3 | 15:0 | 0 | 5:1 | 10:6 | 15:11 |

### 10.5.8 LFB Writes: Format 14

When writing to the LFB with 32-bit format 14(ZARGB 16-1-5-5-5), the RGB channel format specifies the RGB ordering within a 32-bit word. Note that the format of the depth value passed when using LFB format 14 must precisely match the format of the type of depth buffering being used (either 16-bit integer Z or 16-bit floating point 1/W). Note also that the one-bit alpha value passed when using LFB format 14 is replicated to yeild the eight-bit alpha used in the pixel pipeline. Table 10.18 shows the 16-bit color channels within the 32-bit access format 14..

**Table 10.18 Color Channels: 16-bit Access Format 14**

| RGB Channel Format Value | 16-it LFB Access Bits | Channel | | | |
|---|---|---|---|---|---|
| | | Alpha | Red | Green | Blue |
| 0 | 15:0 | 15 | 14:10 | 9:5 | 4:0 |
| 1 | 15:0 | 15 | 4:0 | 9:5 | 14:10 |
| 2 | 15:0 | 0 | 15:11 | 10:6 | 5:1 |
| 3 | 15:0 | 0 | 5:1 | 10:6 | 15:11 |

### 10.5.9 LFB Writes: Format 15

When writing to the LFB with 32-bit format 15(ZZ 16-16), the format of the depth values must preciesly match the format of the type of depth buffering being used (either 16-bit integer Z or 16-bit floating point 1/W). If the Voodoo3 pipeline is enabled for LFB accesses (lfbMode[8] = ‘1’), then rGB color information is taken from color1, and alpha information is is taken from zaColor.

## 10.6 Texture Mapping

The textureMode register (Section 9.3.58) controls texture mapping. The texture mapping capability of Voodoo3 includes perspective correction, texture filtering, texture clamping, and multiple texture blending.

### 10.6.1 Texture Color/Alpha Combine Units

Figure 10.8 shows the Texture Color Combine / Texture Alpha Combine Units. There are four essentially identical units, one for each color and one for alpha. The only difference is in the control bits. In most case, there are two sets of bit numbers in the register bit labels in Figure 10.8. The first bit number is the control bit for the Texture Color Combine Units; the second is the control bit for the Texture Alpha Combine Unit.

For example, tc/tca_zero_other is textureMode[12] for the Texture Color Combine Units and textureMode[21] for the Texture Alpha Combine Unit.

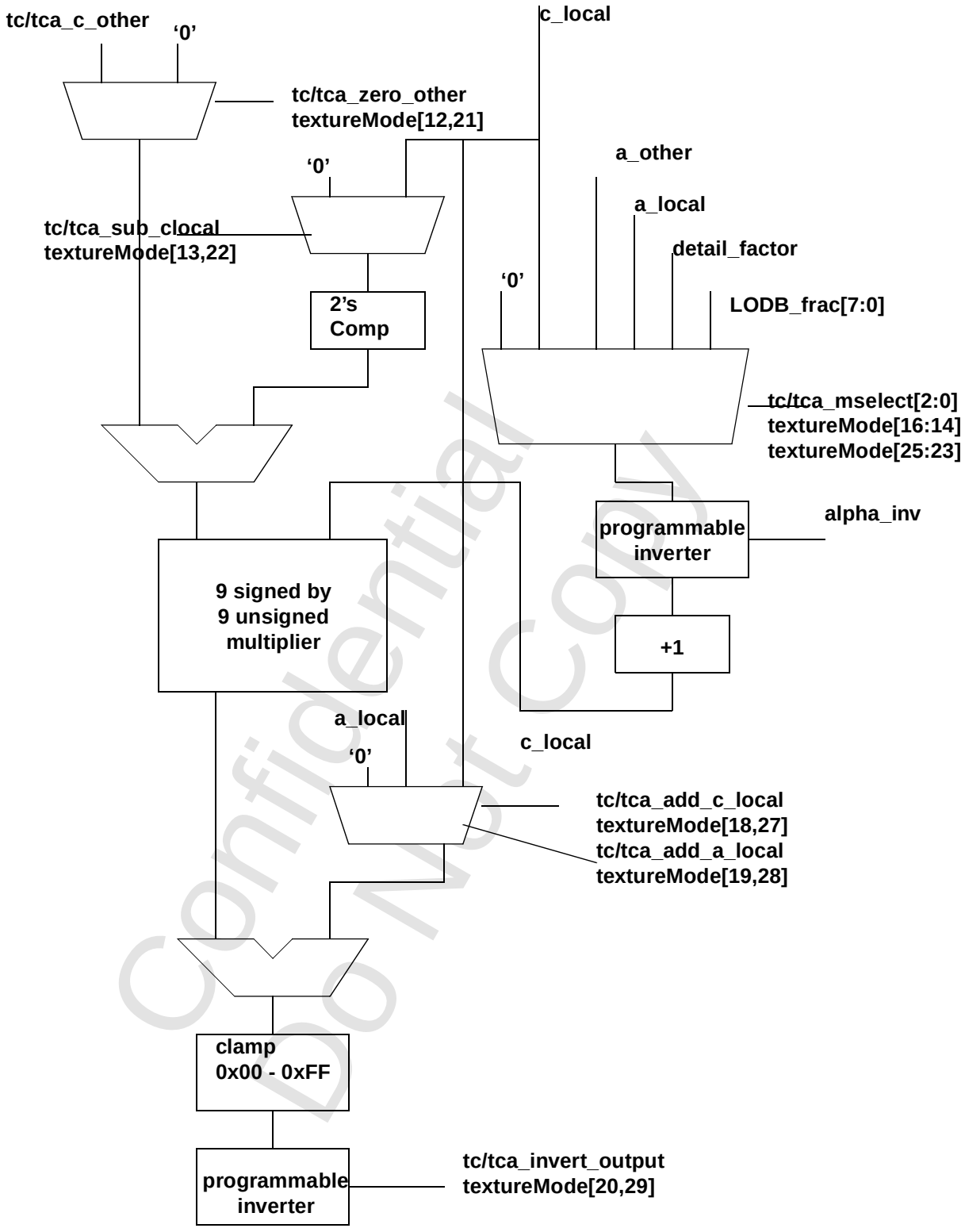



**Figure 10.8 Texture Color/Alpha Combine Unit**

### 10.6.2 Texture Format

textureMode[11:8] specifies the format of the texture accessed by TREX. The texture format field is used for both reading and writing the texture memory. Table 10.19 shows the texture formats and shows how the texture data is expanded into 32-b9t ARGB.

**Table 10.19 Texture Formats (tformat)**

| tformat Value | Format | Expansion to 8-bit | | | |
|---|---|---|---|---|---|
| | | Alpha | Red | Green | Blue |
| 0000b | 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]} |
| 0001b | 8-bit YIQ (4-2-2) | See Table 10.20 | | | |
| 0010b | 8-bit Alpha | a[7:0] | a[7:0] | a[7:0] | a[7:0] |
| 0011b | 8-bit Intensity | 0xFF | i[7:0] | i[7:0] | i[7:0] |
| 0100b | 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]} |
| 0101b | 8-bit Palette to RGB | 0xFF | palette r[7:0] | palette g[7:0] | palette b[7:0] |
| 0110b | 8-bit Palette to RGBA | {palette_r[7:2], palette_r[7:6]} | {palette_r[1:0], palette_g[7:4], palette_r[1:0]} | {palette_g[3:0], palette_b[7:6], palette_g[3:2]} | {palette_b[5:0], palette_b[5:4]} |
| 0111b | Reserved | | | | |
| 1000b | 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]} |
| 1001b | 16-bit AYIQ (8-4-2-2) | See Table 10.20 | | | |
| 1010b | 16-bit RGB (5-6-5) | 0xFF | {r[4:0], r[4:2]} | {g[5:0], g[5:4]} | {b[4:0], b[4:2]} |
| 1011b | 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]} |
| 1100b | 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]} |
| 1101b | 16-bit Alpha, Intensity (8-8) | a[7:0] | i[7:0] | i[7:0] | i[7:0] |

**Table 10.19 Texture Formats (tformat) *(cont.)***

| tformat Value | Format | Expansion to 8-bit | | | |
|---|---|---|---|---|---|
| | | Alpha | Red | Green | Blue |
| 1110b | 16-bit Alpha, Palette (8-8) | a[7:0] | palette r[7:0] | palette g[7:0] | palette b[7:-] |
| 1111b | Reserved | | | | |

.

**Table 10.20 Texture Formats (YIQ Expansion)**

| tformat Value | Format | Expansion to 8-bit | | | |
|---|---|---|---|---|---|
| | | Alpha | Red | Green | Blue |
| 0001b | 8-bit YIQ (4-2-2) | 0xFF | ncc_red[7:0] | ncc_green[7:0] | ncc_blue[7:0] |
| 1001b | 16-bit AYIQ (8-4-2-2) | a[7:0] | ncc_red[7:0] | ncc_green[7:0] | ncc_blue[7:0] |

### 10.6.3 LOD (Level of Detail)

The tLOD register (Section 9.3.59) controls the texture mapping LOD calculations.
lodbias (tLOD[17:12]) is added to the calculated LOD, then it is clamped to the range [lodmin, min[8.0, lodmax]]. That is, the result of the clamping will not be less than lodmin, nor greater than the constant 8.0 or loadbias, whichever is less.
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 to be byte order reversed ([31:24] are swapped with [7:0] and [23:16] are swapped with [15:8]). The tdata_swap bit causes incoming texture data to be word swappe
d ([31:16] are swapped with [15:0]}. If both are enabled, the word swap takes place after the byte swizzling.

See Figure 10.9.

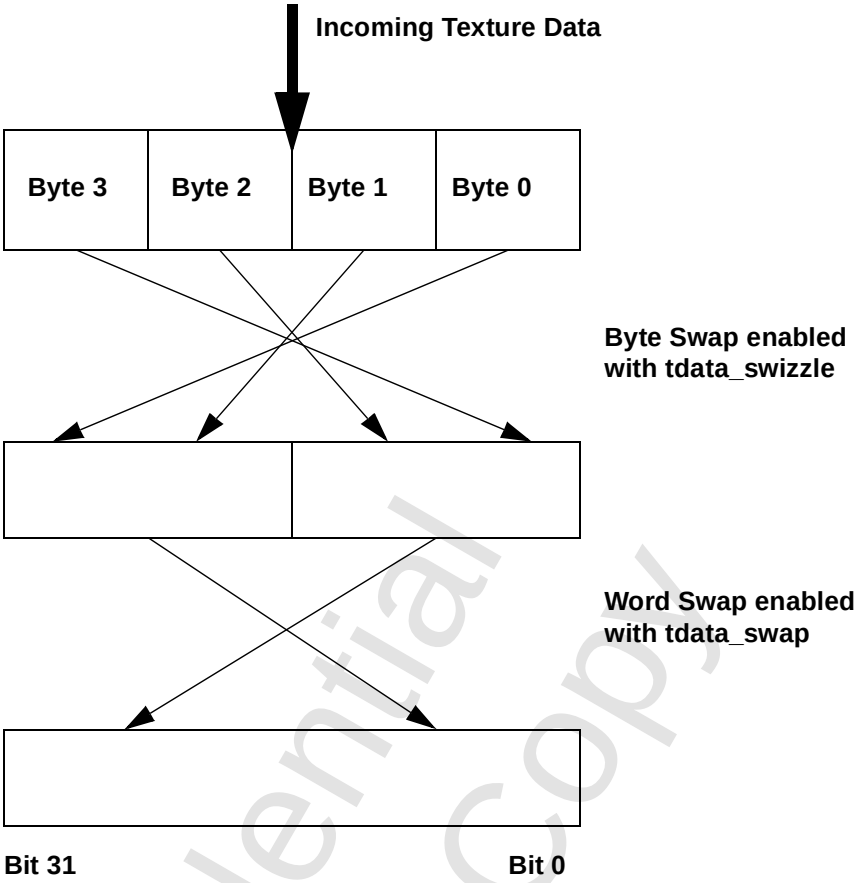



**Figure 10.9 tdata_swizzle, tdata_swap**

Table 10.21 shows the results of swizzling, swapping and both.

**Table 10.21 Swizzling and Swapping**

| Case | Byte 3 | Byte 2 | Byte 1 | Byte 0 |
|---|---|---|---|---|
| Swizzle | Byte 0 | Byte 1 | Byte 2 | Byte 3 |
| Swap | Byte 1 | Byte 0 | byte 3 | Byte 2 |
| Both | Byte 2 | Byte 3 | Byte 0 | Byte 1 |

### 10.6.4 NCC (Narrow Channel Compression)

The two registers nccTable0 and nccTable1 (Section 9.3.63) contain two NCC tables used to store lookup values for the compressed textures. These tables are used when tformat specifies YIQ or AYIQ texture formats.

Two tables are available so that they can be swapped on a per-triangle basis when performing multi-pass rendering. This avoids constantly downloading new tables. The table to be used is chosen in textureMode[5].

Figure 10.10 shows how compressed textures are decompressed using the NCC tables.

.

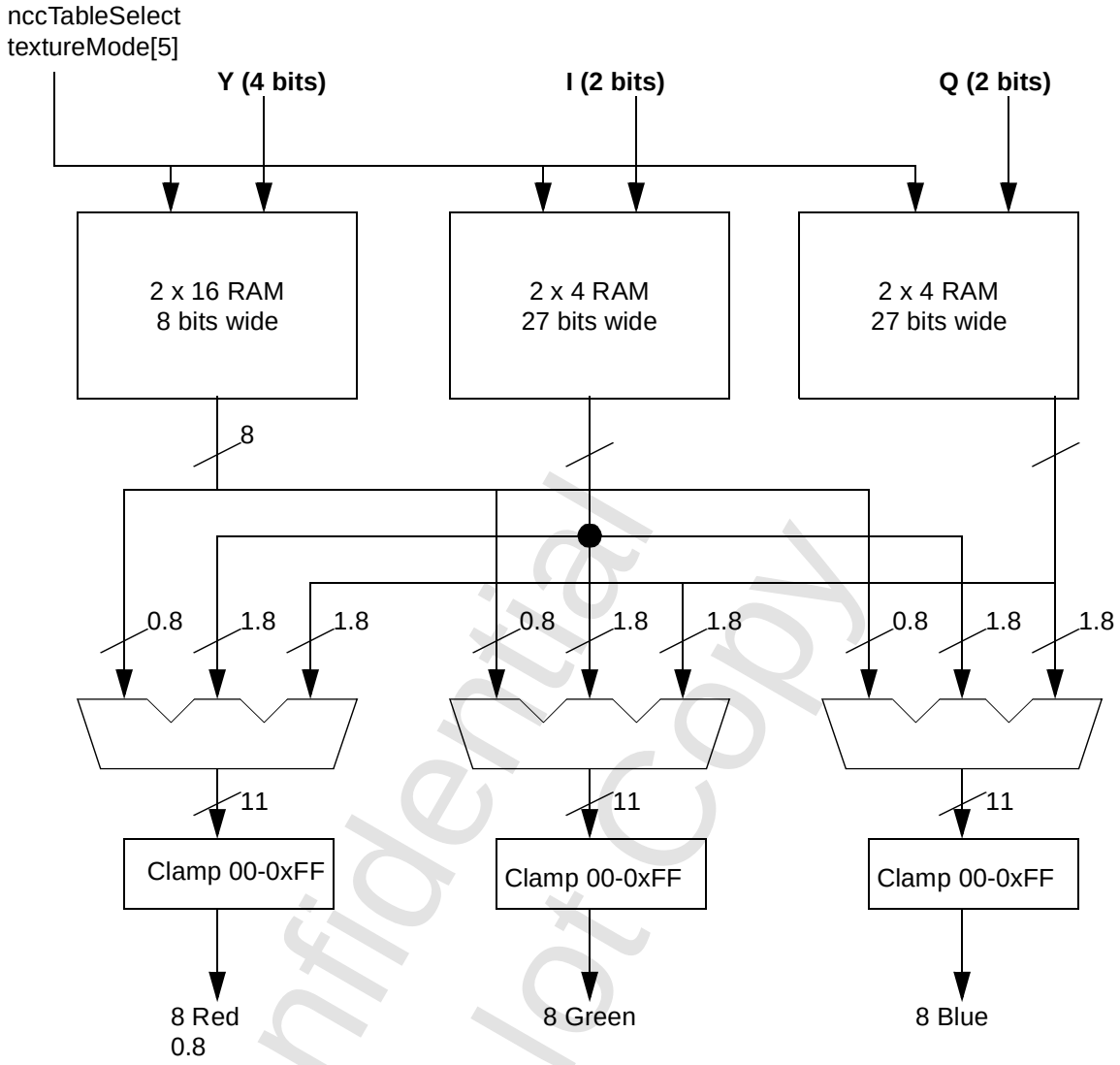



**Figure 10.10 NCC Decompression**

### 10.6.5 Eight-bit Palette

The 8-bit palette is used for 8-bit and 16-bit AP modes. The palette is loaded with register writes. During rendering, four texels are looked up simultaneously, each using an independent 8-bit address.
The palette is written using the same addresses as nccTable0 I and Q space (0x020 0334 through 0x020 0350). If the high order bit of the data is ‘1’, bits 30:0 are loaded into the 8-bit palette (see ). If the high

order bit of the data is ‘0’, bits [26:0] are loaded into the I or Q space of nccTable0

**Table 10.22 Palette Load Mechanism**

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

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

### 10.6.6 Texture Memory Access

There are two fundamental methods; a single base address for all LODs within a texture, multiple base addresses.

### 10.6.7 Single Base Address

When tLOD[24] is 0, a single texBaseAddr register is used. Textures are stored as if mipmapped, even for textures containing a single LOD. The largest texel map (LOD level 0) is stored first, and the other are stored contiguously after.
For linear space, texBaseAddr points to where the texture would start if it contained LOD level 0. In this case, all LODs are stored contiguously after the first. See Figure 10.11.

.

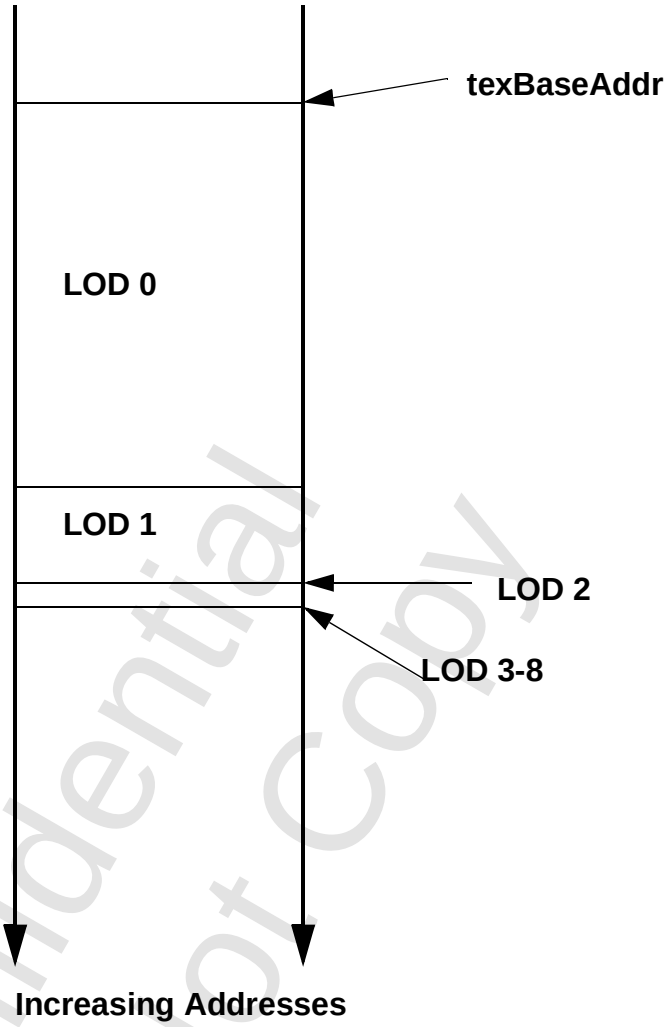



**Figure 10.11 Linear Texture Space**

For tiled space, successive LODs after 0 are stored in a manner that groups the LODs into a more-or-less rectangular space. See Figure 10.12. When only one or some (less than all) of the LOD levels are used, lodmax and lodmin (in TLOD, Section 9.5.2) are used to restrict texture lookup to the levels that are

actually loaded

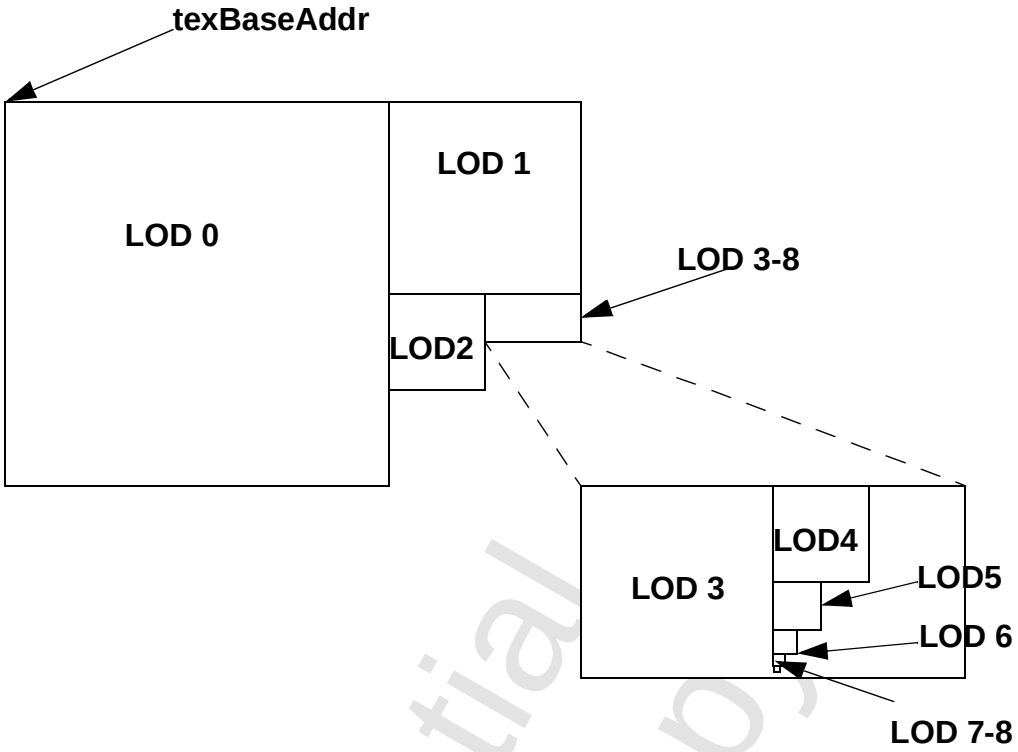



**Figure 10.12 Tiled Texture Space**

### 10.6.8 Multiple Base Addresses

When tLOD[24] is ‘1’, up to four texBaseAddr registers are used. This mode provides more granularity to texture storage and can help with texture memory allocation. There is only one base address for maps 3-8 (or 3-11), so these are stored contiguously.

**Table 10.23 Multiple Texture Base Addresses**

| Register | Address | Level of Detail |
|---|---|---|
| texBaseAddr | 0x020 030C | 0 |
| texBaseAddr1 | 0x020 0310 | 1 |
| texBaseAddr2 | 0x020 0314 | 2 |
| texBaseAddr38 | 0x020 0318 | 3 - 8 |

### 10.6.9 Writing to Texture Space

Voodoo3 provides a dedicated texture download port that is synchronized with normal rendering. Texture downloads done through this port are guaranteed to be processed in-order with 3D rendering, making it unnecessary to idle the chip before downloading (*does this mean the cache coherency problem goes away?*). This port allows writing to frame buffer memory within a four-byte aligned region.
The texture download port occupies 16M of PCI address space (PCI10 + 0x0060 0000). Note that only writes are supported through this port. Texture downloads through this port are always texBaseAddr-relative, even if tLOD[24] is ‘1’. The other three texBaseAddr registers are unused for downloads.
The texture space can also be accessed (not synchronized) through the LFB port (PCI14). This port

provides both read and write access.

### 10.6.10Calculating Texel Addresses

When downloading through the texture download port, address translation depends on whether the texture memory space is tiled or linear.

#### 10.6.10.1 Linear Space

In linear texture space, texels are packed linearly starting from the base address. The offset of a texel from texBaseAddr can be calculated as the sum of:

**Table 10.24 Texel Address: Linear Space**

| Element | Description | Note |
|---|---|---|
| lod_offset | offset of the mipmap from the base | sum of sizes of proceeding mipmaps |
| t_offset | offset within the map to t | t times lod_width |
| s_offset | offset within the line | s times texel size |

For example, the offset to texel 30, 45 in LOD 2, with aspect of 1 x 1, and 16-bpp texels is:

**Table 10.25 Texel Address Example: Linear Space**

| Element | Arithmetic | Note |
|---|---|---|
| lod_offset[a] | (256*256*2) + (128*128*2) = 163,328 | size_lod_0 + size_lod_1 |
| t_offset | 45*64*2 = 5760 | skip over first 45 ts |
| s_offset | 30 * 2 = 60 | skip over first 30 ss |

a. Bit textures are not enabled: that would change the constants in the second column.

The offset from the base is 169,148 = 0x96BC.

#### 10.6.10.2 Tiled Space: not big textures

If tbig (tLOD[30]) = 0, the texture address bit definitions are the same as older product:

**Table 10.26 Tiled Textures**

| Texture Size | Zeroes | LOD[3:0] | t[7:0] | s[7:n] |
|---|---|---|---|---|
| 32-bit textures | - | 21:18 | 17:10 | s[7:0] = 9:2 |
| 16-bit textures | 21 | 20:17 | 16:9 | s[7:1] = 8:2 |
| 8-bit textures | 21:20 | 19:16 | 15:8 | s[7:2] = 7:2 |

# 11 AGP/CMD Transfer/Misc Registers

## 11.1 Overview

This chapter covers three groups of registers that are addressable in memory space beginning at PCI10 plus 0x008 0000. Table 11.1 summarizes these registers. The link is a clickable link to the detailed description. Unless otherwise noted, all non-reserved fields in all registers are read/write.

**Table 11.1 AGP/CMD Transfer/ Misc Register Summary**

| Register Name | Address | Link |
|---|---|---|
| agpReqSize | 0x008 0000 | Section 11.2.1 |
| agpHostAddressLow | 0x008 0004 | Section 11.2.2 |
| agpHostAddressHigh | 0x008 0008 | Section 11.2.3 |
| agpGraphicsAddress | 0x008 000C | Section 11.2.4 |
| agpGraphicsStride | 0x008 0010 | Section 11.2.5 |
| agpMoveCMD | 0x008 0014 | Section 11.2.6 |
| cmdBaseAddr0 | 0x008 0020 | Section 11.3.1 |
| cmdBaseSize0 | 0x008 0024 | Section 11.3.2 |
| cmdBump0 | 0x008 0028 | Section 11.3.3 |
| cmdRdPtrL0 | 0x008 002C | Section 11.3.4 |
| cmdRdPtrH0 | 0x008 0030 | Section 11.3.5 |
| cmdAMin0 | 0x008 0034 | Section 11.3.6 |
| cmdAMax0 | 0x008 003C | Section 11.3.7 |
| cmdStatus0 | 0x008 0040 | |
| cmdFifoDepth0 | 0x008 0044 | Section 11.3.9 |
| cmdHoleCnt0 | 0x008 0048 | Section 11.3.10 |
| cmdBaseAddr1 | 0x008 0050 | Section 11.3.1 |
| cmdBaseSize1 | 0x008 0054 | Section 11.3.2 |
| cmdBump1 | 0x008 0058 | Section 11.3.3 |
| cmdRdPtrL1 | 0x008 005C | Section 11.3.4 |
| cmdRdPtrH1 | 0x008 0060 | Section 11.3.5 |
| cmdAMin1 | 0x008 0064 | Section 11.3.6 |
| cmdAMax1 | 0x008 006C | Section 11.3.7 |

**Table 11.1 AGP/CMD Transfer/ Misc Register Summary *(cont.)***

| Register Name | Address | Link |
|---|---|---|
| cmdFifoDepth1 | 0x008 0074 | Section 11.3.9 |
| cmdHoleCnt1 | 0x008 0078 | Section 11.3.10 |
| cmdFifoThresh | 0x008 0080 | Section 11.3.11 |
| cmdHoleInt | 0x008 0084 | Section 11.3.12 |
| yuvBaseAddress | 0x008 0100 | Section 11.4.1 |
| yuvStride | 0x008 0104 | Section 11.4.2 |

## 11.2 AGP Register Details

This block of registers controls AGP transfers (initiated by Voodoo3). Each register in this group is read/ write except agpMoveCMD.

### 11.2.1 agpReqSize

This register specifies the AGP packet size. The maximum size is 4 Mbytes.

| Bit | Description |
|---|---|
| 31:20 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 19:0 | **Transfer Size:** This field specifies the transfer size. This would have to be a DWORD count to be able to get 4 Mbytes out of 20 bits. |

### 11.2.2 agpHostAddressLow

This register specifies the 32 low order bits of the address in AGP memory. This is a byte address. The high order address bits are in agpHostAddressHigh.

| Bit | Description |
|---|---|
| 31:0 | **HostAddressLow:** This field contains the 32 low order bits of the address in AGP (host) memory. This is a byte address. |

### 11.2.3 agpHostAddressHigh

This register specifies the four high order bits of the address in AGP memory. It also contains the stride and width of AGP memory.

| Bit | Description |
|---|---|
| 31:28 | **HostAddressHigh:** This field contains the four high order bits of the address in AGP (host) memory. This is concatenated with the 32 bits from agpHostAddressLow to form a 36-bit byte address. |
| 27:14 | **AGP Stride:** This field specifies the stride in AGP memory. This is expressed in quadwords (64 bits). Presumably, this is added to the current address each time AGPWidth quadwords have been transferred. |
| 13:0 | **AGP Width:** This field specifies the width in AGP memory. This is expressed in quadwords. |

### 11.2.4 agpGraphicsAddress

This register specifies the destination frame buffer address for an AGP transfer.

| Bit | Description |
|---|---|
| 31:26 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 25:0 | **Frame Buffer Address:** This field specifies the address in the frame buffer to be used as the destination of an AGP transfer. This must be a byte address. |

### 11.2.5 agpGraphicsStride

This register specifies the stride in frame buffer memory for an AGP transfer.

| Bit | Description |
|---|---|
| 31:15 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 14:0 | **Frame Buffer Stride:** This field specifies the stride in frame buffer memory for an AGP transfer. This is a byte count. This value is added to the current frame buffer address after each AGP Width quadwords have been moved. |

### 11.2.6 agpMoveCMD

When this register is written, the AGP transfer begins. This register is write only.

| Bit | Description |
|---|---|
| 31:6 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 5 | **Command Stream ID:** This bit defines which command FIFO to use when using a host initiated AGP data move. The default for this bit is 0. |
| 4:3 | **Destination Memory Type:** This field specifies the type of destination memory, according to the table. |

| Value | Memory Type | Note |
|---|---|---|
| 00b | Linear Frame Buffer | |
| 01b | Planar YUV | |
| 10b | 3D Linear Frame Buffer | |
| 11b | Texture Port | |

| | |
|---|---|
| 2:0 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |

## 11.3 Command List Registers

There are two completely independent sets of command list registers; each set can control a command list. Command lists are covered in Chapter 19. This section covers the registers themselves.

### 11.3.1 cmdBaseAddr0/1

This register specifies the beginning of the command list. This is expressed in terms of pages of 4K bytes each (there are 12 implied trailing zeroes).

| Bit | Description |
|---|---|
| 31:24 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 23:0 | **Base Address:** This field contains the base address of the respective command list. There are twelve implied trailing zeroes for a total of 36 bits of address (enough for AGP). |

### 11.3.2 cmdBaseSize0/1

This register specifies the size of the area set aside for the command list.

| Bit | Description |
|---|---|
| 31:11 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 10 | **Disable Hole Counter:** When this bit is '1', the respective hole counter logic is disabled. I suppose that means that software list management must be used. When this bit is 0, the hole counter logic is enabled. The default for this bit is 0. |
| 9 | **List Location:** When this bit is '1', the respective command list is in AGP memory. When this bit is 0, the command list is in frame buffer memory. The default for this bit is 0. |
| 8 | **List Enable:** When this bit is '1', the respective command list is enabled and will execute packets. When this bit is 0, the respective command list is disabled. |
| 7:0 | **Size:** This field specifies the size of the command list. This is expressed in terms of 4K pages. There are twelve implied trailing zeroes. This value is zero based. The value zero implies 4K bytes; the value one implies 8K bytes, and so on. |

### 11.3.3 cmdBump0/1

The value written to this field increments the respective cmdFifoDepth value. This is used with software list management.

| Bit | Description |
|---|---|
| 31:16 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 15:0 | **Count:** This value is added to the respective cmdFIFODepth value. This is used with software list management. This register is written after the application has added to the list being executed and after the application has flushed the write buffers in the CPU and core logic. |

### 11.3.4 cmdRdPtrL0/1

This register specifies the low order 32 bits of the respective command list read pointer.

| Bit | Description |
|---|---|
| 31:0 | **cmdRdPtrL:** This field is the low order 32 bits of the respective command list read pointer. This is read/write. When the application is setting up the command list, it writes the command list beginning address (cmdBaseAddr with twelve trailing zeroes) to this register. |

### 11.3.5 cmdRdPtrH0/1

This register specifies the high order four bits of the respective command list read pointer.

| Bit | Description |
|---|---|
| 31:4 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 3:0 | **cmdRdPtrH:** This is the high four bits of the respective command list read pointer. This is an extension of cmdRdPtrL. |

### 11.3.6 cmdAMin0/1

This register contains the minimum address pointer. This is used with hardware list management.

| Bit | Description |
|---|---|
| 31:25 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 24:0 | **cmdAMin:** This field contains the minimum address pointer. This is used with hardware list management. When Voodoo3 determines that no holes exist between this value and the respective cmdAMax, this value is replaced with the contents of cmdAMax. When the application is setting up the command list, it writes the address implied in cmdBaseAddr minus four. |

### 11.3.7 cmdAMax0/1

This register contains the maximum address pointer. This is used with hardware list management.

| Bit | Description |
|---|---|
| 31:25 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 24:0 | **cmdAMax:** This field contains the maximum address pointer. This is used with hardware list management. This register is automatically updated whenever a write to the respective command list takes place and the address is greater than the current contents of this register. This register tracks the highest addressed write. When the application is setting up the command list, it writes the address implied in cmdBaseAddr minus four. |

### 11.3.8 cmdStatus0/1

This read-only register allows visibility of the command fifo hardware.

| Bit | Description |
|---|---|
| 31 | **AGP Transfer:** If this bit is ‘1’, it indicates an AGP (packet 6) data transfer is in progress. |
| 30 | **Unpacker Busy:** If this bit is ‘1’, it indicates the unpacker is busy. |
| 29 | **Packet 6 Decompress Busy:** If this bit is ‘1’, it indicates the packet 6 decompression logic is busy. |
| 28 | **Host Data Transfers Complete:** If this bit is ‘1’, it indicates the host data transfers are complete. |
| 27 | **JSR Active:** If this bit is ‘1’, it indicates that a JSR (fetch control) is active. |
| 26 | **Executed Depth Zero:** If this bit is ‘1’, it indicates the exectured depth is zero. |
| 25 | **Prefetched Depth Zero:** If this bit is ‘1’, it indicates the prefetched depth is zero. |
| 24 | **On-Chip FIFO Empty:** If this bit is ‘1’, it indicates the on-chip FIFO is empty. |
| 23:17 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 16 | **JSR (Unpacker) Empty:** |
| 15 | **Jump Tag:** |
| 14:12 | **Jump Command:** |
| 11 | **Header Valid:** |
| 10:6 | **Local State: Remaining entries in packet.** |
| 5:3 | **Extended Packet Command:** |
| 2:0 | **Packet Type:** This field returns the current packet type. |

### 11.3.9 cmdFifoDepth0/1

This register contains the count of unexecuted words in the command list. This is used with software list management.

| Bit | Description |
|---|---|
| 31:20 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 19:0 | **cmdFifoDepth:** This register contains the count of unexecuted words in the respective command list. This is used with software list management. This value is decremented for each word of command list executed. When this value is decremented to zero, the respective command list stalls. When a write is done to the cmdBump address, the data is added to the contents of the respective cmdFifoDepth. In this way, the application notifies Voodoo3 that additional list is available |

### 11.3.10 cmdHoleCnt0/1

This register keeps the count of unwritten locations between cmdAMin and cmdAMax. This is used with hardware list management.

| Bit | Description |
|---|---|
| 31:16 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 15:0 | **cmdHoleCnt:** This field contains the number of unwritten locations between the respective cmdAMin and cmdAMax. When this field goes to zero, the respective cmdAMin is replaced with the respective cmdAMax. |

### 11.3.11 cmdFifoThresh

This register contains the fifo thresholds for both command lists.

| Bit | Description |
|---|---|
| 31:22 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 21:9 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 8:5 | **High Water Mark:** When fifo freespace is above this value, then fill requests will be generated. The default value for this field is seven. |
| 4:0 | **Low Water Mark:** When the fifo freespace is below this value, no new requests are made. The default value for this field is 0. |

### 11.3.12 cmdHoleInt

This register specifies how long (in terms of MCLK cycles) a hole can remain in a command list.

| Bit | Description |
|---|---|
| 31:23 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 22 | **HoleInt Enable:** When this bit is ‘1’, the hole time-out interrupt counter and interrupt are enabled. When this bit is 0, the counter and interrupt are disabled. The default for this field is 0. |
| 21:0 | **Time-out Value:** This field contains the number of MCLK cycle a hole counter can have hole before generating an interrupt. The counter is enabled only when bit 22 is ‘1’. |

## 11.4 Miscellaneous Registers

Here are a couple of registers that control the yuv aperture.

### 11.4.1 yuvBaseAddress

This register specifies the starting frame buffer address of the yuv aperture.

| Bit | Description |
|---|---|
| 31:25 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 24:0 | **YUV Base Address:** This field specifies the starting frame buffer address of the yuv aperture. |

### 11.4.2 yuvStride

This register specifies the destination stride values of the U and V planes.

| Bit | Description |
|---|---|
| 31 | **Tiled:** If this bit is '1', the YUV destination is tiled. If this bit is 0, the destination is linear. The default for this bit is 0. |
| 30:14 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 13:0 | **Stride:** This field specifies the stride for Y, U, and V. |

# 12 Caveats

There are a few programming caveats listed in the internal documentation. They are repeated pretty much verbatim here.

## 12.1 Memory Access Size

All memory accesses to Voodoo3 registers, except VGA registers, must be 32-bit DWORD accesses. Linear Frame Buffer accesses may be 32-bit or 16-bit access, depending on the linear frame buffer access format specified in lbfMode. Byte (8-bit) access are only allowed to Voodoo3 linear frame buffer.

## 12.2 Determining Voodoo3 Idle Condition

After certain Voodoo3 operations (including linear frame buffer accesses), the application must recite a specific incantation to determine whether the chip is really, truly, most sincerely, idle. The following pseudo-code fragment illustrates the method.

```
/**************************************************
***SST_IDLE
* returns 0 if SST is not idle
* returns 1 if SST is idle
**************************************************/
SST_IDLE()
{
        ulong j, i;
        //Make sure the state machines are idle
        PCI_MEM_WR(NOPCMD, 0x0);                //write to the nopcmd register
        i = 0;
        while (1)
        {
        j = PCI_MEM_RD (STATUS);                //read the status register
        if (j & SST_BUSY)
                return (0);                     //definitely not idle
        else
                i++;                            //keep the counter
        if (i>3)
                return (1);                     //three times it was idle
        }
}
```

## 12.3 Triangle Subpixel Correction

Triangle subpixel correction is performed in the on-chip triangle setup unit of Voodoo3. When subpixel correction is enabled (fbzColorPath[26] = ‘1’), the incoming starting color, depth, and texture coordinate parameters are all corrected for non-integer aligned starting triangle XY coordinates.
The subpixel correction in the triangle setup unit is performed as the starting color, depth, and texture coordinates parameters are read from the PCI FIFO. As a result, the exact data sent from the host CPU is changed to account for subpixel correction. If a triangle is rendered with subpixel correction enabled, all subsequent triangle must resend starting color, depth, and texture coordinate parameters. Otherwise the last triangle’s subpixel corrected starting parameters are corrected (again!) and incorrect results are generated.

# 13 Hardware Cursor

Voodoo3 supports a 64 x 64 hardware cursor.

The hardware cursor pattern is stored as two monochrome 64 x 64 bit maps. This allows for a hardware cursor of 64 x 64 pixels; each pixel is controlled by two bits (one from each of the two maps).

At each horizontal retrace, the video processor determines whether any part of the current scanline is 'covered' by the cursor. If so, it fetches the cursor pattern bits for the current scanline (actually, enough cursor pattern for eight consecutive scanlines is fetched, reducing the number of memory accesses from 64 to eight per frame).
For each of the 4096 pixels 'under' the cursor, the processor interprets the two bits (one from each map) according to the Cursor Mode in vidProcCfg[1] (see Section 6.5.3). Table 13.1 shows what is displayed for each case. Unused portions of the 64 x 64 pattern should be programmed to display the current screen color. For example, if the application requires a 32 x 32 cursor, three-quarters of the pattern would be current screen color.

### Table 13.1 Hardware Cursor Pixels

| Pattern 0 | Pattern 1 | vidProcCfg[1] = 0 Windows Mode | vidProcCfg[1] = 1 X11 Mode |
|---|---|---|---|
| 0 | 0 | Color0 | Current Screen Color |
| 0 | 1 | Color1 | Current Screen Color |
| 1 | 0 | Current Screen Color | Color0 |
| 1 | 1 | Invert Screen Color | Color1 |

The patterns are stored in the frame buffer at the location specified in hwPatAddr (Section 6.5.4). The patterns are stored in alternate 64-bit scanlines. Pattern 0 is stored in the 64-bit block following the corresponding 64-bit block of pattern 1.
The pattern is displayed at the location specified in hwCurLoc (Section 6.5.5). This register contains x and y coordinates of the lower right corner (unlike some system that use the upper left corner). Using the lower right corner allows the cursor to be displayed anywhere on the screen (including partially off the left or top) without resorting to signed numbers or some offset mechanism.
The two cursor colors are stored in hwCurC0 and hwCurC1. The cursor is enabled by programming vidProcCfg[27] to '1'.

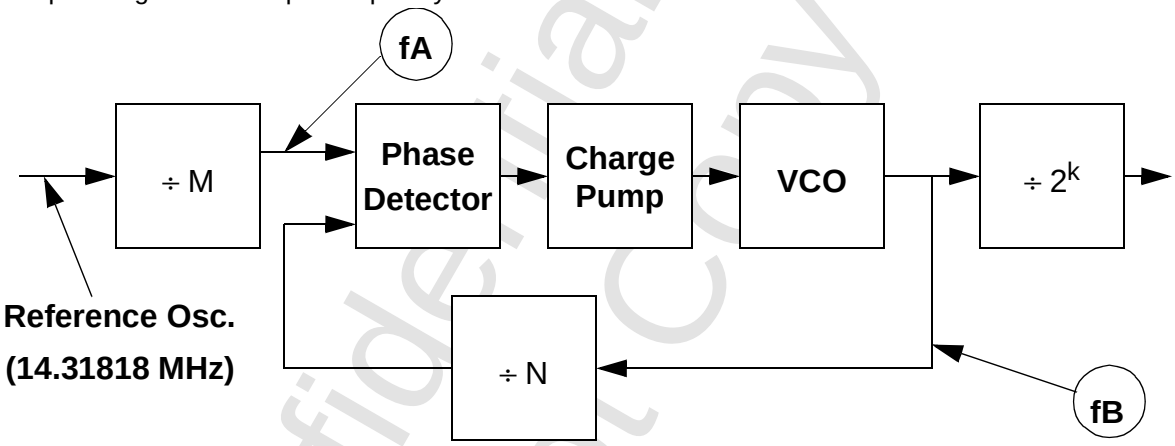
# 14 Frequency Synthesizers

## 14.1 Introduction

Voodoo3 incorporates three on-chip PLL (Phase-Locked Loop) frequency synthesizers. Two are used to generate the (combined) graphics engine/memory controller clock and the video clock. They are functionally the same and are programmed similarly. They use a common reference oscillator, typically at 14.31818MHz.
The third synthesizer is used to generate the AGP clock. It is programmed very differently from the other two and will be covered in Section 14.6.

## 14.2 Block Diagram

Figure 14.1 is a functional block diagram of a typical PLL synthesizer. The reference frequency (typically 14.31818 MHz) is divided by M in the input pre-scaler. This produces one input (labeled fA in the diagram) to the phase detector. This input is compared to the divided output of the VCO. The charge pump attempts to adjust the frequency of the VCO so that the two inputs to the phase detector are the same.
The output of the VCO is labeled fB in the diagram. It is divided by N to form the second input to the phase detector. The VCO also goes to the post-scaler, when it is divided by $2^k$ to become the synthesizer output. The equation gives the output frequency.




$$fout = \frac{ref \bullet (N+2)}{(M+2) \bullet (2^K)}$$

**Figure 14.1 Frequency Synthesizer Functional Block Diagram**

## 14.3 Programming

Each synthesizer is programmed with a single register, as indicated in Table 14.1.

**Table 14.1 pllCtrl Registers**

| Offset | Name | Synthesizer | Note |
|---|---|---|---|
| 0x40 | pllCtrl0 | Video Clock | DAC and CRTC |
| 0x44 | pllCtrl1 | GRX/Memory Clock | 2D and 3D Logic, Memory Interface |

Each pllCtrl register contains fields to program the three dividers. N and M must each be programmed to two less than the actual calculated divider. Observe that K sets the size of a divider chain, rather than being used directly.
If the DeviceID in PCI0[31:16] is 0x0004, the 'M' field is forced to the value 0x24 for pllCtrl1 only. This limits the maximum graphics/memory clock to 141 MHz. When the initialization value for pllCtrl1 is selected, this must be taken into account. If the DeviceID is 0x0005, this field is fully programmable (but see the programming restrictions in Section 14.4).

### Table 14.2 pllCtrl Fields

| Bits | Field | Size | Offset |
|---|---|---|---|
| 15:8 | N | Eight bits | 2 |
| 7:2 | M | Six bits | 2 |
| 1:0 | K | Two bits | 0 |

## 14.4 Programming Restrictions

Table 14.3 shows the restrictions that apply to the programming of the synthesizers. They may be difference for each synthesizer.

### Table 14.3 Programming Restrictions

| Parameter | pllCtrl0: Video Clock | pllCtrl1: GRX/Memory Clock |
|---|---|---|
| Output Freq | | |
| Value of N | | |
| Value of M | | |
| Value of K | | |
| fA | | |
| fB | | |

## 14.5 Programming Notes

Generally, there will be a number of combinations of N, M, and K that will produce a given frequency. It is best to program a synthesizer so that the frequencies at the input of the phase detector are as high as possible (subject to the restrictions noted in Table 14.3). This is because a higher frequency corresponds to a smaller period at the phase detector, reducing the jitter. To keep the frequency high, program the pre-scaler and post-scaler to small values.

## 14.6 AGP Phase-locked Loop

The AGP PLL, controlled by pllCtrl2, is quite different. Only two bits in the register (pllCtrl2) are defined. These two bits are used in conjunction with the PLL_BYPASS pin and the strapping of VMI_HD2 to generate the two clocks used in the AGP interface.

**Table 14.4 AGP PLL Summary**

| VMI_HD2 PCI66 | VMI_ADDR3 PLL_BYPASS | pllCtrl2[1:0] | Clk66 | Clk133 | Note |
|---|---|---|---|---|---|
| 0 | x | (don't care) | PCI_CLK | PCI_CLK | PCI, bypass pll |
| 1 | 0 | 0x00 | 1/4 pll output | 1/2 pll output | AGP, PLL on |
| 1 | 0 | 0x01 | 1/4 PCI_CLK | 1/2 PCI_CLK | Test only |
| 1 | 0 | 0x10 | PCI_CLK | 1/2 pll output | Test only |
| 1 | 0 | 0x11 | PCI_CLK | 1/2 PCI_CLK | AGP bypass pll |
| 1 | 1 | (don't care) | 1/4 PCI_CLK | 1/2 PCI_CLK | Test only |

# 15 Digital RGB Outputs

## 15.1 Digital Data Formats

Digital RGB is clocked on both edges of the clock. Which bit is clocked on each edge is controlled by vidInFormat[8] (Section 6.5.7).

**Table 15.1 Tv out Digital Data Formats**

| Pin | vidInFormat[8] = 0 Chrontel Encoder | | vidInFormat[8] = 1 Brooktree Encoder | |
|---|---|---|---|---|
| | Rising Edge | Falling Edge | Rising Edge | Falling Edge |
| VMI_HA[3] | G0[4] | R0[7] | R0[7] | G0[4] |
| VMI_HA[2] | G0[3] | R0[6] | R0[6] | G0[3] |
| VMI_HA[1] | G0[2] | R0[5] | R0[5] | G0[2] |
| VMI_HA[0] | B0[7] | R0[4] | R0[4] | B0[7] |
| VMI_HD[7] | B0[6] | R0[3] | R0[3] | B0[6] |
| VMI_HD[6] | B0[5] | G0[7] | G0[7] | B0[5] |
| VMI_HD[5] | B0[4] | G0[6] | G0[6] | B0[4] |
| VMI_HD[4] | B0[3] | G0[5] | G0[5] | B0[3] |
| VMI_HD[3] | G0[0] | R0[2] | R0[2] | G0[0] |
| VMI_HD[2] | B0[2] | R0[1] | R0[1] | B0[2] |
| VMI_HD[1] | B0[1] | R0[0] | R0[0] | B0[1] |
| VMI_HD[0] | B0[0] | G0[1] | G0[1] | B0[0] |

# 16 External VMI/TV Support

Table 16.1 shows the configurations of external VMI/TV supported by Voodoo3.

### Table 16.1 VMI/TV Support

| Configuration | Gnelock Enable VidInFormat[16] | Genlock Source VidInformat[18] | not_use_vga_timing VidInFormat[17] |
|---|---|---|---|
| TV encoder Master | 1 | 1 (TV encoder) | 1 |
| TV encoder Slave | 0 | (don't care) | 0 |
| VMI genlock | 1 | 0 (VMI) | 0 |
| VMI slave | 0 | (don't care) | 0 |
| TV encoder master plus VMI slave | 1 | 1 (TV encoder) | 1 |
| TV encoder slave plus VMI genlock | 1 | 0 (VMI) | 0 |
| Special case[a] | 1 | 0 (VMI) | 1 |

a. While this case is possible to configure, it probably doesn't make sense, because one pixel of input data from the VMI device requires two clocks while one pixel of output data to the monitor or TV encoder requires only a single clock. Because of this, the timing of the input and output devices cannot be aligned.

# 17 Video In

## 17.1 Introduction

When video data arrives through the VMI, optional decimation and filtering takes place. The data are packed into words of 128 bits in a FIFO before being written into the frame buffer. Since the writes are always aligned on a 128-bit boundary, the first and last words may not be complete.
Supported pixel formats for the video-in data are YUV 4:2:2 and YUV 4:1:1. Both pixel formats are stored in a form of 16 bits per pixel; this means that 4 bits per pixel are unused in the case of YUV 4:1:1.

## 17.2 VMI Data

Eight-bit YCbCr interface is used. The data format is CCIR-656 YCbCr 4:2:2. Pixels arrive in the style of (Cb0[7:0] or U0[7:0]) → Y0[7:0] → (Cr0[7:0] or V0[7:0]) → Y1[7:0]. Video data may be interlaced.

## 17.3 VMI Timing

Timing signals include VREF (VMI_VSYNC), HREF (VMI_HSYNC), VACTIVE (VMI_BLANK), and PIXCLOCK (VMI_PCLK). VREF and HREF polarity are programmable in vidInFormat[5 and 6]. If HREF is active during the trailing edge of VREF, the field is even. If HREF is not active during the trailing edge of VREF, the field is odd.

## 17.4 Buffering

Video data are stored into the frame buffer at locations whose starting addresses are specified in vidInAddr0, vidInAddr1, and vidInAddr2. The specific buffers used are summarized in Table 17.1.

### Table 17.1 Video In Buffer Summary

| Buffering | vidInFormat[10:9] | vidInAddr0 | vidInAddr1 | vidInAddr2 |
|---|---|---|---|---|
| Single | 00b | Used | Unused | Unused |
| Double | 01b | Used | Used | Unused |
| Triple | 10b | Used | Used | Used |

Double- and triple-buffering is used to avoid video tearing. However, since video is coming at a different rate from the screen refresh (screen refresh is typically at a higher frequency than video capture), switching of the video-in capture buffer is not synchronous with screen refresh.
At the end of each VMI frame, the vmi_int input signal will be asserted. The video processor will then switch to the next video-in capture buffer (assuming multiple buffering is enabled). At the same time, the video processor updates vidInStatusCurrentLine[18:17] to indicate the buffer it just completed writing as well as vidInStatusCurrentLine[16] to indicate whether the frame (more precisely, this is called a field) just captured was even or odd. An interrupt signal will notify the host for display buffer flipping for the video-in data. Alternately, if the Video_in data displayed as overlay enable (vidProcCfg[9]) bit is set, the video processor will handle the display buffer flipping automatically.

## 17.5 Scaling

The following pseudo-code fragment shows the Bresenham scaler for scaling down a video window in the horizontal direction.

```
error = vidInXDecimInitErr;
repeat until the source pixels of a video window scanline are exhausted
        {
        if (error < 0)
```

```
        {
        move to next source pixel;
        error = error + vidInXDecimDelta1;
        }
    else
        {
        select the current pixel as the destination pixel;
        move to next source pixel;
        error = error - vidInXDecimDelta2;
        }
}
```

The following pseudo-code fragment shows the Bresenham scaler for scaling down a video window in the vertical direction.

```
error = vidInYDecimInitErr;
at each VideoIn Hsync
    {
    if (error < 0)
        {
        skip the entire line of video-in data;
        error = error + vidInYDecimDelta1;
        }
    else
        {
        select the current line of video-in data;
        error = error - vidInYDecimDelta2;
        }
    }
```

### 17.5.1 Scaling Example

Suppose the source size is 640 x 240 and it is desired to have it magnified to 1024 x 768 on the screen.

**Table 17.2 Video-In Scaling Example**

| Item | Register | Value | How calculated |
|---|---|---|---|
| Source Width | vidOverlayDudx-OffsetSrcWidth[31:19] | 0x500 | 640 times 2 bytes = 1280 (16 bpp source assumed) |
| Hor Step Size | vidOverlayDudx[19:0] | 0xA0000 | 640/1024 = 0.625 = 10/16 |
| Vert Step Size | vidOverlayDvdy[19:0] | 0x50000 | 240/768 = 0.3125 = 5/16 |
| Hor Offset | vidOverlayDudx-OffsetSrcWidth[18:0] | 00000h | If no offset is needed. If the upper-left overlay pixel must be the center of the first pixel of the overlay surface, both offsets must be set to 0.5 (0x40000). |
| Vert Offset | vidOverlayDvdyOffset[18:0] | 00000h | |

## 17.6 Weave De-interlacing

If Weave deinterlaced mode is enabled, the video processor can determine even/odd fields. If odd, the specifed vidInAddr will be used as the starting address of the video-in capture buffer. If even, vidInStride

will be used as the starting address offset, and added to the specified vidInAddr. Video-in buffer will be switched at every other VSYNC. vidInStride must be programmed to a value which equals 1X the regular line stride regardless of whether the video-in data is interlaced or not.

## 17.7 Bob De-interlacing

When the video processor display the even field, it adds the constant 0.5 to the initial vertical offset (dvdy offset) used by the backend bilinear scaler. Since de-interlacing in the backend uses the bilinear scaler unit to interpolate between two interlaced lines, the application must enabled bilinear filtering, overlay vertical scaling, overlay horizontal scaling, and set up the initial dvdy offset, dvdy, initial dudx offset and dudx according to the desired magnification factor between the source video and display video. The suggested settings fro the parameters for backend de-interlacing without horizontal magnification are shown in Table 17.3.

**Table 17.3 Backend De-interlacing w/o Horizontal Magnification**

| Parameter | Register | Value | Note |
|---|---|---|---|
| bilinear filter enable | vidProcCfg[17:16)? | 11b | |
| overlay vertical scaling enable | vidProcCfg[15] | 1 | |
| overlay horizontal scaling enable | vidProcCfg[14] | 0 | |
| initial dvdy offset | vidOverlayDvdyOffset | 0.25 | |
| dvdy | vidOverlay Dudx | 0.5 | |
| initial dudx offset | (vidOverlayDvdyOffset) | (don't care) | |
| dudx | (vidOverlayDvdy) | (don't care) | |

### 17.7.1 2x Mode

Backend De-interlacing is not support for 2x mode (2-pixel per video clock mode) since bilinear filtering is not available in 2x mode.

## 17.8 Video Limitations

- In 1x mode, 3 streams of pixel fetching will consume more memory bandwidth than is available for 32-bit (pixel size) desktop. This means chroma-keying and bilinear filtering cannot be turned on simultaneously for 32-bit desktop.
- In 2x mode (used for any display larger than 1280 x 1024), bilinear filterning is not supported. All backend zoom (magnification) is done by point sampling (replication).
- 1x to 10x backend zoom (magnification) is supported with increments of 0.1x. Larger magnification is supported, but with larger increments.
- 1x to 1/16x video-in decimation (minimization) is supported with increments of 0.015x.
- Retain the 3-bit tap filter for RGB 5:6:5 dithered as an alternative to the 2x2 box filte.
- Interlaced video output is not implemented.
- Hardware Cursor is two-color only.
- YUV 4:1:1 pixel format is stored as unpacked in the frame buffer. This means each pixel will occupy 16 bits. This makes it easier to extract pixels from the frame buffer.
- Video with YUV 4:2:2 format must be stored on a four-byte memory boundary while YUV 4:1:1 must be stored on an eight-byte boundary. This is necessary since U and V information are shared between two pixels in 4:2:2 mode and four pixels in 4:1:1 mode.

# 18 VMI Host Interface

Transactions on the VMI host interface are generated by programming each control bit explicitly. This is unlike some graphics chips incorporating state machines that translate writes or reads to an address range into VMI host interface writes or reads.

## 18.1 Register Bit Assignments

The bits that control the VMI host are in vidSerialParallelPort. The VideoIn Interface Configuration field (vidInFormat[15:14]) must be programmed to 01b. Table 18.1 shows the bit assignments.

### Table 18.1 VMI Host Interface Control Bits

| Field | Description | Note |
|---|---|---|
| 17:14 | VMI Address | This field is driven onto VMI_HA[3:0] |
| 13:6 | VMI Data | Bidirectional data on VMI_HD[7:0] |
| 5 | VMI Data Output Enable | Active low enable for VMI_HD[7:0] |
| 4 | VMI_RDY_N | DTACK_N for mode A, READY for mode B |
| 3 | VMI_RW_N | R/W# for mode A, WR# for mode B |
| 2 | VMI_DS_N | DS# for mode A, RD# for mode B |
| 1 | VMI_CS_N | Active low chip select |
| 0 | VMI Host Port Enable | Active low enables for control pins |

## 18.2 Mode B Write Example

The following values would be written to vidSerialParallelPort to execute a mode B write. Remember that vidSerialParallelPort[31:18] are used for the serial ports (and GPIO) and ought not be changed. It is assumed that bit 0 was set to '1' when the VMI was put into VMI host interface mode.

### Table 18.2 VMI Mode B Write Example

| 17:14 | 13:6 | 5 | 4 | 3 | 2 | 1 | 0 | Note |
|---|---|---|---|---|---|---|---|---|
| Adrs | Data | 0 | X | 1 | 1 | 1 | 1 | Drive address and data |
| Adrs | Data | 0 | X | 1 | 1 | 0 | 1 | Drive CS# |
| Adrs | Data | 0 | X | 0 | 1 | 0 | 1 | Drive WR# (must follow CS# by at least 10 ns) |
| | | | ? | | | | | Read DTACK_N (must wait at least 28 ns after WR#) |
| Adrs | Data | 0 | X | 1 | 1 | 0 | 1 | Remove WR# (minimum command width is 40 ns) |
| Adrs | Data | 1 | X | 1 | 1 | 1 | 1 | Remove data, CS# |

# 19 Command Lists Protocol

## 19.1 Introduction and Overview

The Voodoo3 can execute commands from a list (in the internal documentation, this is called a CMDFIFO) that is located in either frame buffer memory or AGP memory. This is useful since the application can form a list of commands and store it for later (perhaps repetitive) execution. Once the list is available, the application need only direct the attention of Voodoo3 to it for execution. The commands come in a variety of packet types; these are covered in Section 19.3.
There are two completely independent sets of command list control registers. Voodoo3 can execute two completely independent command list concurrently. The registers are covered in Section 11.3.

## 19.2 List Management

In the most straight-forward case, the application constructs and stores an entire list. It then tells Voodoo3 where the list is and that it is to be executed. Voodoo3 executes the list and stalls when it gets to the end. The application waits until Voodoo3 has completed the list and then goes on to the next activity.
Better parallelism can be gotten by allowing Voodoo3 to execute the list at the same time the application is building it. In this case, it is necessary to guarantee that Voodoo3 executes only from locations that contain valid packets. This is complicated by the fact that packets or words within packets may be written out of order, due either to programming expediencies or write re-ordering in the CPU and/or core logic chipset. If the command list is in AGP memory, the application must explicitly manage the list and must explicitly notify Voodoo3 whenever entries have been added. This is called software list management. If the command list is in frame buffer memory, either software or hardware management can be used. Hardware management involves Voodoo3 watching writes to the area containing the list and keeping track of available packets on its own.

### 19.2.1 Software List Management

When the application has formed a useful portion of the command list, it loads cmdBaseAddr, cmdRdPtr (to the beginning of the list) and cmdFifoDepth to the number of valid words in the list. As Voodoo3 executes words from the list, it will increment cmdRdPtr and decrement cmdFifoDepth.
After the Voodoo3 has executed a few words of the list, the situation will be as shown in Figure 19.1. If the application does nothing, eventually Voodoo3 will have decremented cmdFifoDepth to zero and will stall, having reached the end of the usable list (in the figure, this would be the first unwritten word).
Suppose the application now writes the two empty words in the list. It can then flush the pending write buffers in the CPU and core logic to guarantee that the writes have actually taken place. Having done this, it can write cmdBump with number of valid entries available. Voodoo3 will add this value to cmdFifoDepth (which may or may not have been decremented to zero) and press on executing packets.

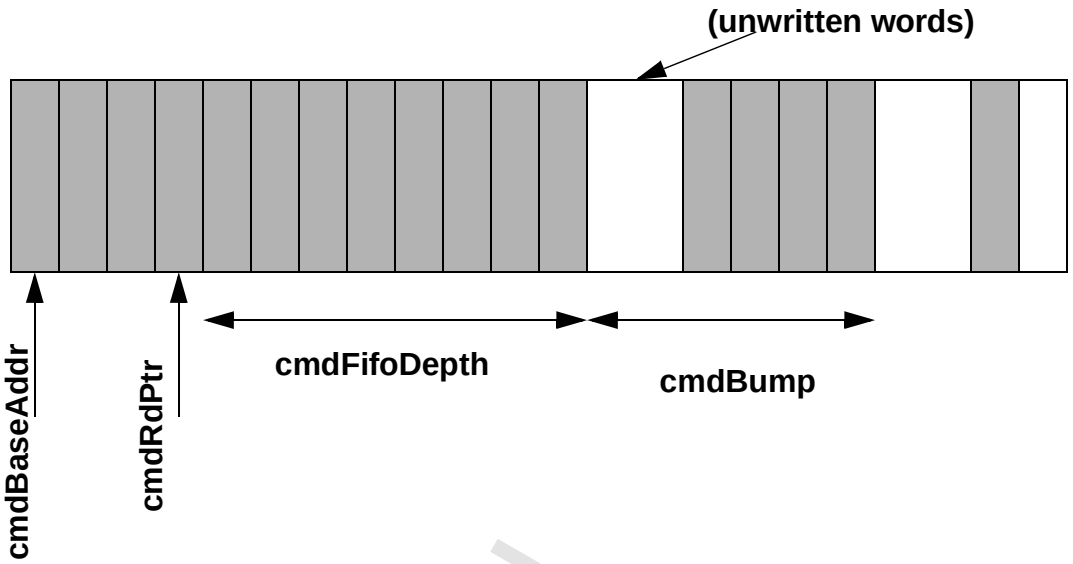



**Figure 19.1 Software List Management**

Since the command list must have a finite length, it is necessary to treat it as a circular list. The cmdBaseSize register specifies the size of the list in terms of 4 Kbyte pages. When the Voodoo3 has processed the last possible entry in the list, it will automatically set the read pointer back to cmdBaseAddr and continue. From the point of view of the maintenance of cmdFifoDepth, this is no different from just executing words in order.

### 19.2.2 Hardware List Management

If the command list is kept in frame buffer memory, Voodoo3 can keep track of the command list updates by monitoring ('snooping') writes to the area occupied by the list. This method cannot be used if the list is in AGP memory.

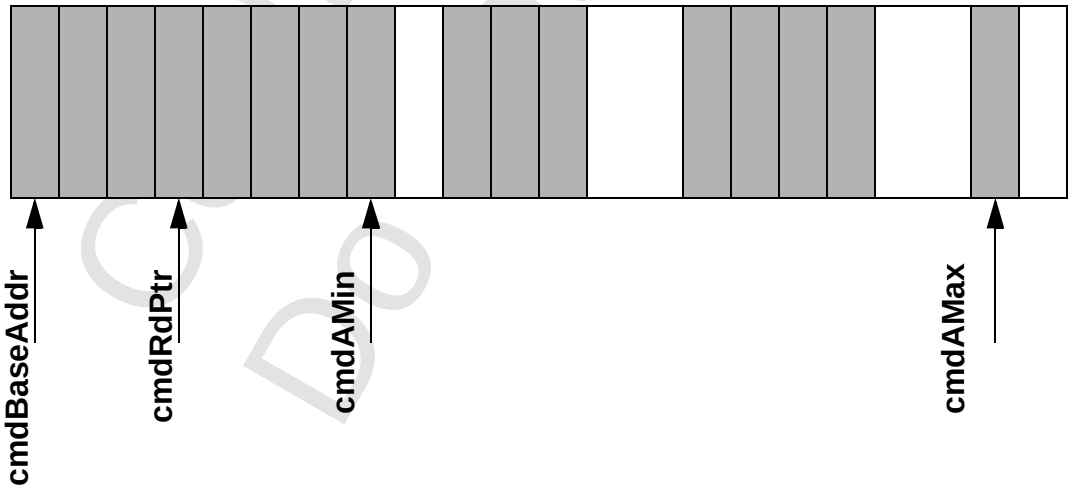



**Figure 19.2 Hardware List Management**

The application must prepare at least four words of the command list. Before it loads anything into the frame buffer, it loads cmdBaseAddr, the cmdRdPtr (to the beginning of the list), cmdAMin (to the address implied in cmdBaseAddr minus four), and cmdAMax (also to the address implied in cmdBaseAddr minus four). It also writes cmdBaseSize to put Voodoo3 into command list mode.

The application now begins writing entries into the frame buffer. The writes can take place in any order. cmdAMax will always contain the highest address that has been written. cmdAMin contains the highest address of the location prior to the lowest *unwritten* location: the address of the last valid packet word. As Voodoo3 executes words from the list, it will increment cmdRdPtr. When (or if) cmdRdPtr reaches cmdAMin, execution will stall until the application/CPU/core logic fills in the hole(s) between cmdAMin and cmdAMax. Then cmdAMin will be loaded with the contents cmdAMax and Voodoo3 can resume executing from the list. Figure 19.2 shows the situation where Voodoo3 has not yet executed up to cmdAMin and there are holes between cmdAMin and cmdAMax.

There is a register called cmdHoleCnt that is used to keep track of the number of holes between cmdAMin and CmdAMax. In addition, cmdFifoDepth keeps track of distance between cmdRdPtr and cmdAMin.

It is possible to generate an interrupt in the situation where a hole has existed for a very long time (this might result from a programming error). See the description of cmdHoleInt.

Since the command list must have a finite length, it is necessary to treat it as a circular list. The cmdBaseSize register specifies the size of the list in terms of 4 Kbyte pages. When the Voodoo3 has processed the last possible entry in the list, it will automatically set the read pointer back to cmdBaseAddr and continue. From the point of view of the maintenance of cmdFifoDepth, cmdAMin, and cmdAMax, this is no different from just executing words in order.

## 19.3 Command List Packet Types

All command list packets begin with a 32-bit packet header that defines the packet contents. In the figures that follow, the packet header is always word 0. Bits [2:0] define the packet header type. All packet headers and data are 32-bit words and begin on 32-bit boundaries. Table 19.1 summarizes the packet types. Reserved bits in all packet header words and all following words must be written as 0 for upward compatibility.

### Table 19.1 Command List Packet Summary

| Type | Description | Link |
|---|---|---|
| 000b | Control transfer | Section 19.3.1 |
| 001b | Write common or consecutive addresses | Section 19.3.2 |
| 010b | Write specific 2D registers | Section 19.3.3 |
| 011b | Write vertex data groups | Section 19.3.4 |
| 100b | Write specific 2D/3D registers | Section 19.3.5 |
| 101b | Write LFB, YUV, 3D LFB, or Texture Port | Section 19.3.6 |
| 110b | Initiate AGP Transfer | Section 19.3.7 |

### 19.3.1 Packet Type 0

Packet Type 0 is used for flow control.

| Bit(s) | 31:29 | 28:6 | 5:3 | 2:0 |
|---|---|---|---|---|
| word 0 | Reserved | Address [24:2] | Func | 000 |
| word 1 | Reserved | Address[35:25] | | |

**Figure 19.3 Command List Packet Type 0**

| Bit | Description |
|---|---|
| 31:29 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 28:6 | **Destination Address[24:2]:** This field specifies bits 24:2 of the destination address. Bits [1:0] of the destination are zero since packets on are 32-bit boundaries. This field is used for JSR and JMP instructions. |
| 5:3 | **Function:** This field specifies the function according to the table. Values not in the table must not be written into this field. |

| Function Code | Description | Note |
|---|---|---|
| 000b | NOP | |
| 001b | JSR | Saves return address |
| 010b | RET | Recovers return address |
| 011b | JMP Local Frame Buffer | |
| 100b | JMP AGP | Requires word 1 |

**NOP** increments the read pointer, decrements the depth counter, and performs no other action. NOPs can be used to pad command lists.
**JSR** is used to call a subroutine. The destination address is in bits [28:6] of word 0. Only a single level of subroutine is supported. If a JSR is executed from within a subroutine, it will overwrite the original return address, leaving Voodoo3 to forever wander in the darkness. Subroutines can be used only when the command list is in frame buffer memory.
**RET** is used to return from a subroutine. Control is returned to the word immediately following the last JSR instruction. The execution of RET without a corresponding JSR is a programming error.
**JMP LFB** is an unconditional jump. The destination address is in bits [28:6] of word 0. This variant of JMP is used when the command list is in the frame buffer (cmdBaseSize[9] = 0).
**JMP AGP** is an unconditional jump. The destination address is in bits [28:6], and bit [10:0] of word 1. This is the only function of Packet Type 0 that requires two words. This variant of JMP is used when the command list in AGP memory (cmdBaseSize[9] = 1).

| | |
|---|---|
| 2:0 | **Packet Type:** This field must be written as 000b to indicate a type 0 packet. |

### 19.3.2 Packet Type 1

Packet Type 1 allows writes to a specific 2D or 3D register address or group of addresses.

| Bit(s) | 31:16 | 15 | 14:3 | 2:0 |
|---|---|---|---|---|
| word 0 | Word Count | inc | Register Base | 001 |
| word 1 | Data | | | |
| word N | Optional Data N (up to 64K words) | | | |

**Figure 19.4 Command List Packet Type 1**

| Bit | Description |
|---|---|
| 31:16 | **Word Count:** This field specifies the number of data words to write to the address or addresses. This is an unsigned integer that is one less that the actual number of words. The minimum value for this field is one, specifying two words. The maximum number of words that can be written 65,536. |
| 15 | **Increment:** If this bit is 0, all the words are written to the address specified in bits 14:3. If this bit is '1', the first words is written to the address specified and subsequent words are written to consecutive addresses. |
| 14:3 | **Register Base:** This field specifies the address of the first or only register address to be written. Bits in this field are used as indicated in the table. |
| 2:0 | **Packet Type:** This field must be written as 001b to indicate packet type 1. |

| Bit(s) | Description | Note |
|---|---|---|
| 14 | Register Set | '1' says 2d, 0 says 3D |
| 13 | Reserved | must be written as 0 |
| 12:11 | Chip Select | See Table 9.2 |
| 10:3 | Register Address | Bits[9:2][a] from Address in Table 7.1 or Table 9.3 |

a. Eight bits (bits[9:2]) of the address offset go into this eight-bit field. Take ten bits of the address field and divide by four.

### 19.3.3 Packet Type 2

Packet Type 2 allows writes to up to 29 specific 2D register addresses. A mask specifies which addresses are to be written.

| Bit(s) | 31:3 | 2:0 |
|---|---|---|
| word 0 | 2D Register Bit Mask | 010 |
| word 1 | Data | |
| word N | Optional Data N (up to 28 words) | |

**Figure 19.5 Command List Packet Type 1**

| Bit | Description |
|---|---|

31:3 **2D Register Bit Mask:** This is a bit significant field; each bit enables the writing of a specific register, according to the table. This field must not be programmed to 0. There must be exactly one data word for every '1' in this field. Data words are ordered from lower destination address to higher destination address.

| Bit | Register Address | Description |
|---|---|---|
| 3 | 0x010 0008 | clip0Min |
| 4 | 0x010 000C | clip0Max |
| 5 | 0x010 0010 | dstBaseAddr |
| 6 | 0x010 0014 | dstFormat |
| 7 | 0x010 0018 | scrColorKeyMin |
| 8 | 0x010 001C | srcColorKeyMax |
| 9 | 0x010 0020 | dstColorKeyMin |
| 10 | 0x010 0024 | dstColorKeyMax |
| 11 | 0x010 0028 | bresError0 |
| 12 | 0x010 002C | bresError1 |
| 13 | 0x010 0030 | rop |
| 14 | 0x010 0034 | srcBaseAddr |
| 15 | 0x010 0038 | commandExtra |
| 16 | 0x010 003C | lineStipple |
| 17 | 0x010 0040 | lineStype |
| 18 | 0x010 0044 | pattern0Alias |
| 19 | 0x010 0048 | pattern1Alias |

| Bit | Register Address | Description |
|---|---|---|
| 20 | 0x010 004C | clip1Min |
| 21 | 0x010 0050 | clip1Max |
| 22 | 0x010 0054 | srcFormat |
| 23 | 0x010 0058 | srcSize |
| 24 | 0x010 005C | srcXY |
| 25 | 0x010 0060 | colorBack |
| 26 | 0x010 0064 | colorFore |
| 27 | 0x010 0068 | dstSize |
| 28 | 0x010 006C | dstXY |
| 29 | 0x010 0070 | command |
| 30 | 0x010 0074 | (reserved) |
| 31 | 0x010 0078 | (reserved) |

| | |
|---|---|
| 2:0 | Packet Type: This field must be written as 010b to specify packet type 2. |

### 19.3.4 Packet Type 3

Packet type 3 is used to transfer up to 16 vertex data groups. While it is possible to transfer a single vertex data group, it is much more efficient to transfer multiple vertex data groups.

| Bit(s) | 31:29 | 28 | 27:22 | 21:10 | 9:6 | 5:3 | 2:0 |
|---|---|---|---|---|---|---|---|
| word 0 | Num | PC | sSetup | Parameter Mask | Num Vertices | CMD | 011 |
| word 1 | Data | | | | | | |
| word N | Optional Data N (up to 223 words) | | | | | | |

**Figure 19.6 Command List Packet Type 2**

| Bit | Description |
|---|---|
| 31:29 | **Dummy Words:** This field specifies that zero to seven dummy words follow the data words. |
| 28 | **Packed Color:** When this bit is ‘1’, it specifies that packed color data follow the X and Y values. When this bit is ‘0’, it specifies that independent red, green, blue, and alpha values follow the X and Y values. |
| 27:22 | **sSetup:** This bit specifies data to be written to the sSetupMode register, according to the table. |

| Mask Bit | sSetupMode Bit | Description |
|---|---|---|
| 22 | 16 | Strip Mode |
| 23 | 17 | Enable Culling |
| 24 | 18 | Culling Sign |
| 25 | 19 | Disable Ping-pong sign correction |
| 27:26 | Unassigned | Reserved: Must be 0 |

### 19.3.4 Packet Type 3 *(cont)*

**Bit** **Description**

21:10 **Parameter Mask:** This field specifies data to be written to sSetupMode, according to the table. This field also specifies the number of data words required for each vertex.

| Mask Bit | sSetupMode Bit | Description |
|---|---|---|
| 10 | 0 | Setup red, green, and blue |
| 11 | 1 | Setup Alpha |
| 12 | 2 | Setup Z |
| 13 | 3 | Setup Wb |
| 14 | 4 | Setup W0 |
| 15 | 5 | Setup S0 and T0 |
| 16 | 6 | Setup W1 |
| 17 | 7 | Setup S1 and T1 |
| 21:18 | Unassigned | Reserved: Must be 0 |

### 19.3.4 Packet Type 3 *(cont)*

| Bit | Description |
|---|---|

9:6 **Num Vertex:** This field specifies the number of vertices. The total number of data words in a packet type 3 is this value multiplied the number of words implied in bits 21:10 (and bit 28). The maximum number of data words is 14 words for each of 16 packets, or 224 words. The data words are in the order indicated in the table.

| Word | Parameter |
|---|---|
| 1 | X |
| 2 | Y |
| 3 | Red/Packed ARGB (optional) |
| 4 | Green (optional) |
| 5 | Blue (optional) |
| 6 | Alpha (optional) |
| 7 | Z (optional) |
| 8 | Wbroadcast (optional)‘ |
| 9 | W0: Tmu0 and Tmu1 W (optional) |
| 10 | S0: Tmu0 and Tmu1 S (optional) |
| 11 | T0: Tmu0 and Tmu1 T (optional) |
| 12 | W1: Tmu1 W (optional) |
| 13 | S1: Tmu1 S (optional) |
| 14 | T1: Tmu1 T (optional) |

5:3 **Command:** This field specifies the command code, according to the table. Values not in this table must not be programmed into this field. The last column of this table shows the sequence of implied commands where M indicates Mode Register Write, B indicates sBeginTriCMD, and D indicates sDrawTriCMD.

| CMD | Description | Implied Sequence |
|---|---|---|
| 000b | Independent Triangle | MBDDBDDBDDBDD... |
| 001b | Start new triangle strip (or fan) | MBDDDDDDDDDD... |
| 010b | Continue existing triangle strip (or fan) | DDDDDDDDDDDD... |

2:0 **Packet Type:** This field must be written as 011b to indicate packet type 3.

### 19.3.5 Packet Type 4

Packet type 4 is used to transfer up to 16 words to registers in either the 2D or 3D set.

| Bit(s) | 31:29 | 28:15 | 14:3 | 2:0 |
|---|---|---|---|---|
| word 0 | Num | Register Mask | Register Base | 100 |
| word 1 | Data | | | |
| word N | Optional Data N (up to 15 words) | | | |

**Figure 19.7 Command List Packet Type 2**

| Bit | Description |
|---|---|
| 31:29 | **Pad Words:** This field specifies zero to seven pad words following the data words. |
| 28:15 | **Register Mask:** This is a bit significant field where each bit specifies a register is to be written (‘1’) or skipped (0). This field must be non-zero. The mask is processed from bit 15 to bit 28, when bit 15 corresponds to the register specified in 14:3. |
| 14:3 | **Register Base:** This field specifies the address of the first or only register address to be written. Bits in this field are used as indicated in the table. |
| 2:0 | **Packet Type:** This bit must be written as 100b to indicate packet type 4. |

| Bit(s) | Description | Note |
|---|---|---|
| 14 | Register Set | ‘1’ says 2d, 0 says 3D |
| 13 | Reserved | must be written as 0 |
| 12:11 | Chip Select | |
| 10:3 | Register Address | Low order eight bits in Table 7.1 or Table 9.3 |

### 19.3.6 Packet Type 5

Packet Type 5 is used to load up to 512K words into the frame buffer. Data must be in the correct byte lanes, and the base address must be 32-bit aligned. Transfer to tile space is limited if the tile-stride does not match PCI stride. Tile space rows are not continuous; each tile row must be separated into separate packets.

| Bit(s) | 31:30 | 29:26 | 25:22 | 21:3 | 2:0 |
|---|---|---|---|---|---|
| **word 0** | Space | Byte Enable W2 | Byte Enable Wn | Num Words | 101 |
| **word 1** | Rrsvd | Base Address [24:0] | | | |
| **word 2** | Data | | | | |
| **word N** | Optional Data N (up to 512K words) | | | | |

**Figure 19.8 Command List Packet Type 2**

| Bit | Description |
|---|---|
| 31:30 | **Space:** This field specifies the address space in the frame buffer, according to the table. The beginning address in the frame buffer is specified in word 1 of the packet. The beginning address must be 32-bit aligned. |

| Space | Description |
|---|---|
| 00b | Linear Frame Buffer |
| 01b | Planar YUV |
| 10b | 3D LFB |
| 11b | Texture Port |

| | |
|---|---|
| 29:26 | **Byte Enable W2:** This field specifies the byte enables for word 2 (the very first data word). These are active low. |
| 25:22 | **Byte Enable Wn:** This field specifies the byte enables for all subsequent word (all data words except the very first). These are active low. |
| 21:3 | **Num Words:** This field specifies the number of data words. The value 0 indicates that one data word is present in the packet. Up to 524,288 words can be transferred in a single packet. |
| 2:0 | **Packet Type:** This field must be written as 101b to indicate Packet Type 5. |

### 19.3.7 Packet Type 6

Packet Type 6 is used to initiate a AGP transfer. This is a fixed-length packet of exactly five words.

| Bit(s) | 31:26 | 25:5 | 4:3 | 2:0 |
|---|---|---|---|---|
| word 0 | Reserved | Transfer Size (bytes) | Type | 110 |
| word 1 | AGP Address [31:0] | | | |
| word 2 | AGP Address [35:32] | AGP Stride[13:0] | Width[13:0] | |
| word 3 | Rsvd | Frame Buffer Offset[25:0] | | |
| word 4 | Rsvd | | Destination Stride[14:0] | |

**Figure 19.9 Command List Packet Type 2**

| Bit | Description |
|---|---|
| 31:26 | **Reserved:** Reserved bits must be written as 0 for upward compatibility. |
| 25:5 | **Transfer Size:** This field specifies the transfer size in bytes. |
| 4:3 | **Type:** This field specifies the memory space, according to the table. |

| Type | Description |
|---|---|
| 00b | Linear Frame Buffer |
| 01b | Planar YUV |
| 10b | 3D LFB |
| 11b | Texture Port |

| | |
|---|---|
| 2:0 | **Packet Type:** This field must be written as 110b to indicate a Packet type 6. |
| Word 1 | **AGP Address[31:0]:** This field specifies the low order 32 bits of the source address in AGP memory. |
| Word2[31:28] | **AGP Address[35:32]** This field specifies the high order four bits of the source address. |
| Word2[27:14] | **AGP Stride:** This field specifies the stride in AGP memory. This value is added to the AGP address after each width bytes have been moved. |
| Word2[13:0] | **Width:** This field specifies the width of the transfer. |
| Word3[25:0] | **Frame Buffer Offset:** This field specifies the beginning address in the frame buffer. |
| Word4[14:0] | **Destination Stride:** This field specifies the stride in frame buffer memory. |

# 20 Memory Notes

The following notes are extracted from the Frame Buffer Application Note (AN1).

## 20.1 Memory Connections

A board can be layed out for a 4-, 8-, or 16-Mbyte SGRAM array or a 16-Mbyte SDRAM array. The designer must choose a specific array. It is essentially impossible to lay out a board that can be field (end user) upgraded.
An SGRAM array can have two banks; one controlled by each chip select term (MCS0, MCS1). An SDRAM array can have a single bank (the MCS terms are used for address bits).
There are two sets of address and control lines (address, RAS, CAS, WE, DSF). The two sets will always contain identical information; they are doubled up to provide additional drive and to make a tighter layout possible. The address and control pins whose names end in A drive the memory devices providing MD[63:0]. The address and control pins whose names end in B drive the memory devices providing MD[127:64]. The reader may refer to the reference designs supplied by 3Dfx Interactive, Inc.

The address line distribution depends on the memory devices, as shown in Table 20.1.

### Table 20.1 Address Assignments

| Address Pin | 8 MBit SGRAM | 16 Mbit SGRAM | 16 Mbit SDRAM |
|---|---|---|---|
| MA_A/B[8:0] | A[8:0] | A[8:0] | A[8:0] |
| MA_A/B[9] | n/c (BA1 on 16M) | BA1 | A[9] |
| MA_A/B[10] | BA0 (band address) | BA0 | BA |
| MCS_0 | (bank select) | | A10 (MD[63:0]) |
| MCS_1 | | | A10 (MD[127:64]) |

## 20.2 Gross Timing Controls

Registers bits allow the timing to be tuned to get the best possible performance. The BIOS supplied by 3Dfx Interactive, Inc. programs these bits. The following information is supplied for customers writing their own BIOS or who wish to override the standard timing.
The gross timing controls are in dramInit0 (see Section 6.3.6).
Table 20.2 shows those timing specifications that are programmed in terms of clock cycles for a typical SGRAM (MOSYS MG802C256). We also show the bit(s) in dramInit0 that are used to program the number of clocks, the maximum and minimum number of clocks, and the default value. The final column shows the value recommended for the MOSYS MG802C512-6.

### Table 20.2 SGRAM Timing Definitions

| Symbol | Definition | dramInit0 Bit(s) | min - max | default | Typical Value |
|---|---|---|---|---|---|
| $t_{RRD}$ | row active to row active | 1:0 | 1-4 | 01b | 01b (2 cycles) |
| $t_{RCD}$ | Ras to CAS delay | 3:2 | 1-4 | 10b | 10b (3 cycles) |

**Table 20.2 SGRAM Timing Definitions *(cont.)***

| Symbol | Definition | dramInit0 Bit(s) | min - max | default | Typical Value |
|---|---|---|---|---|---|
| $t_{RP}$ | row precharge | 5:4 | 1-4 | 10b | 10b (3 cycles) |
| $t_{RAS}$ | minimum RAS active | 9:6 | 1-16 | 0100b | 0011b (4 cycles) |
| $t_{RC}$ | minimum row cycle | 13:10 | 1-16 | 0111b | 0110b (7 cycles) |
| $t_{CAS}$ | CAS latency | 15:14 | 1-4 | 10b | 01b (2 cycles) |
| $t_{MRS}$[a] | mode and spec reg cycle | 16 | 1-2 | 1b | 1b (2 cycles) |
| $t_{DQR}$ | read to DQM assertion | 17 | 0-1 | 1b | 0 for 2-CAS<br>1 for 3-CAS |
| $t_{BWC}$ | block write cycle time | 18 | 1-2 | 1b | 1b (2 cycles) |
| $t_{WL}$ | write to precharge | 19 | 1-2 | 0b | 1b (2 cycles) |
| $t_{BWL}$[b] | block write to precharge | 21:20 | 1-4 | 01b | 01b (2 cycles) |
| $t_{RL}$[c] | read to precharge | 22 | 1-2 | 1b | 0b (1 cycle) |

a. This is tLRC in the Mosys data sheet.
b. This is tBPL in the Mosys data sheet.
c. This is not specified in the Mosys data sheet.

## 20.3 Fine Timing Controls

Bits in dramInit1 can be programmed to select the delay from the internal MCLK term to the clock output pin and from the MCLK input pin to the term that samples the SGRAM read data. The following diagrams are intended to show the functionality of the register bits and do not necessarily represent any actual implementation.
The fine timing controls are in dramInit1 (see Section 6.3.7).

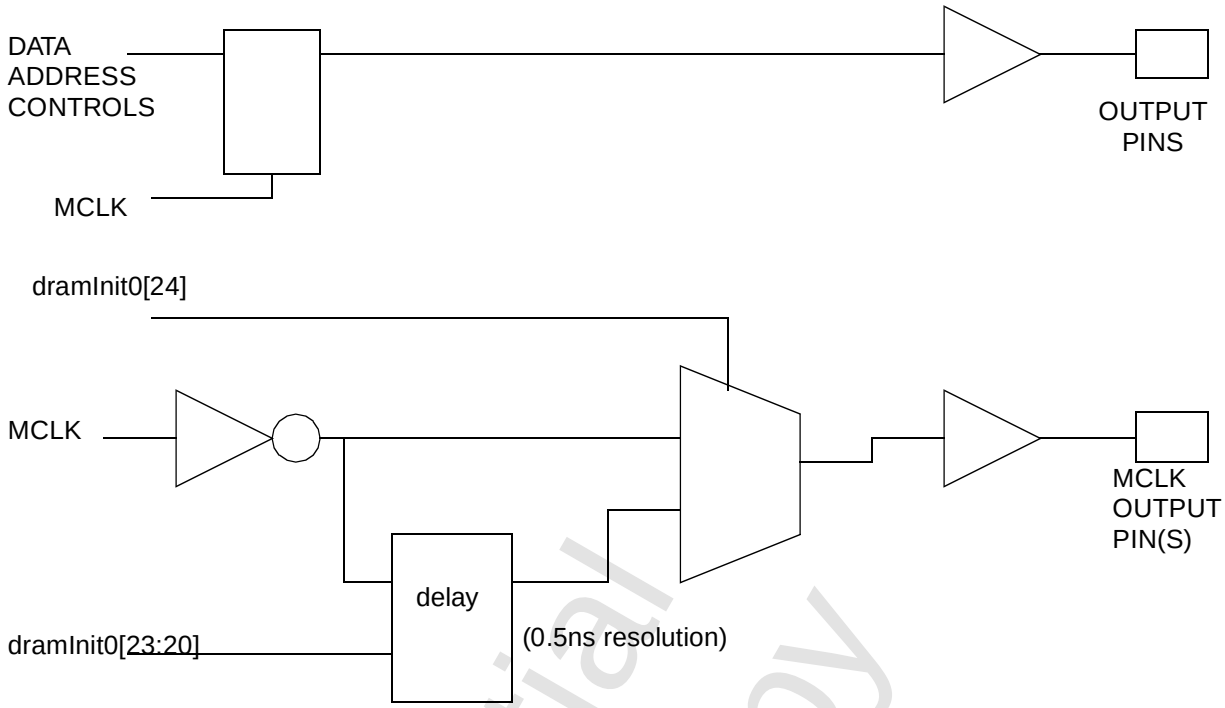



**Figure 20.1 MCLK Out Delay Logic**

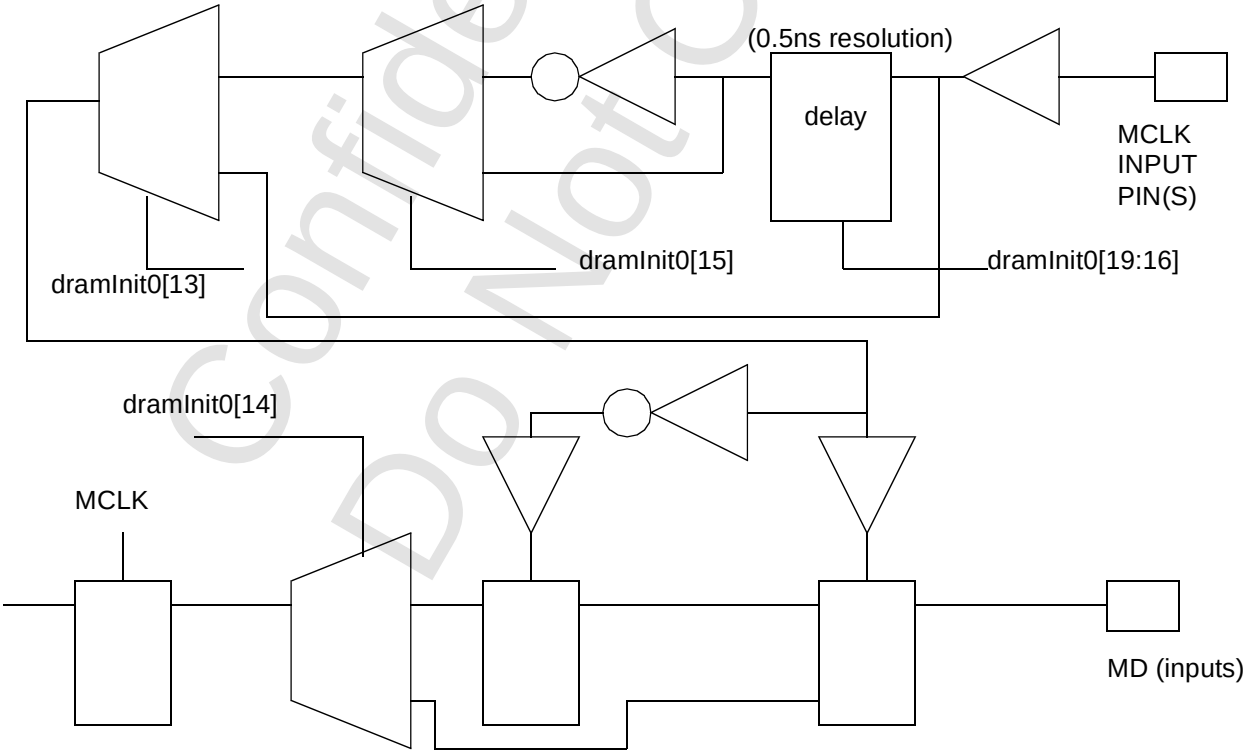



**Figure 20.2 Read Data Sample Clock Logic**

## Keyword Index

# Register Index