Advance

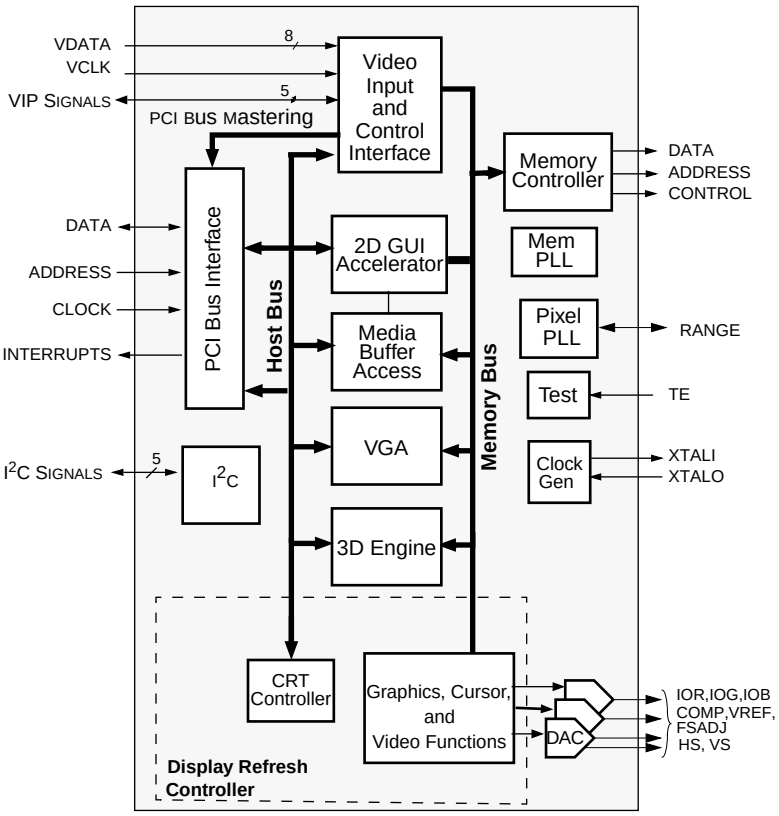
This document contains information on a product under development.
Information is not warranted and is subject to change.

Bt2166

High-Performance PCI/AGP 3D Video/Graphics Controller

Applications

- Microsoft Windows 95 DirectX acceleration
- Consumer appliance PCs
- Living room PCs
- Microsoft Simply Interactive PCs
- Video conferencing
- Multimedia applications



Feature Summary

- Microsoft Windows 95 Direct3D accelerator
  - –Programmable 3D triangle set up coprocessor
  - –Triangle parameter edge traversal/interpolation
  - –Data/address fetch for texture, color, alpha and Z values per triangle or tri-strip
  - –Scaled and rotated sprites
  - –Perspective-correct texture mapping
  - –Bilinear and trilinear texture filtering
  - –MIP mapping
  - –Texture transparency and translucency
  - –True Color and Palletized texture support
  - –Alpha blending translucency, fog, depth queuing
  - –Z-buffering and double buffering
  - –Flat and Gouraud shading
- Microsoft Windows 95 DirectDraw accelerator
  - –High-performance 2D graphics acceleration
  - –64KB GUI command queue
  - –Transparent BLT
- Microsoft Windows 95 DirectVideo accelerator
  - –Hardware-accelerated video playback with sharpness control
  - –multi-tap horizontal/vertical interpolation filtering for video display
  - –Two hardware video windows
  - –Hardware double-buffering
  - –Chroma and color keying
- Host bus interfaces
  - –PCI 2.1 with bus mastering
  - –3.3 V 66 MHz Intel Accelerated Graphics Port (AGP)
- Integrated 173 MHz RAMDAC for all VESA display modes up to 1600 x 1200 at 65 Hz
- 64-bit SGRAM and SDRAM (2 MB to 4 MB)
  - –Supports 85 MHz bus
- TV/MPEG video display and capture support
  - –Glueless connection for Brooktree VideoStream and MPEG decoders
  - –3rd generation TrueView™ technology for interlaced video sources
  - –VBI data capture and decoding for Intercast
- 32 KB to 1 MB of Flash ROM
- Full VGA compatibility
- 208-pin PQFP

Brooktree® | *Rockwell Semiconductor Systems — Brooktree Division • 9868 Scranton Road • San Diego, CA 92121-3707*
619-452-7580 • 1-800-2-BT-APPS • FAX: 619-452-1249 • Internet: apps@brooktree.com • L2166_A

PREFACE

Document Contents

This hardware specification contains the following chapters.

Product Description—Provides an overview of the Bt2166 Graphics/Video Controller, including a summary of the pin assignments.

CPU Address Space Apertures—Describes the programming interface to the Bt2166 Graphics/Video accelerator, the Media Buffer, and BIOS ROM.

VGA Implementation—Describes the compatibility and operation of the VGA within the Bt2166 controller.

I^2C Register Definitions —Specifies the programming interface to the Bt2166 I^2C master and slave modules, and defines the I^2C registers.

PCI Configuration Space Register Definitions —Contains the definitions of registers mapped to the PCI Configuration Space.

GUI Accelerator—Describes the CPU addressing to the GUI and defines GUI accelerator and GUI aperture registers.

Interrupt Register Definitions—Contains the definitions of the memory-mapped interrupt registers.

Configuration and PLL Register Definitions—Contains the definitions of the general configuration and PLL-related registers.

Video Input Subsystem—Describes the interfaces between the Bt2166 Graphics/Video Controller and the VideoStream Decoder, and between the Bt2166 and other video sources. Contains the definitions of the video input and Video Interface Port (VIP) registers.

Display Refresh Controller—Describes the module responsible for fetching display memory data for generating screen outputs and for generating CRT timing signals. This includes back-end video upscaling and mixing, chroma and color keying, and cursor overlay.

3D Accelerator—Describes the 3D graphics accelerator and pixel rendering engine. Contains the definitions for the 3D interface registers and the 3D accelerator and pixel rendering engine registers.

PCI Configuration Space Register Definitions —Contains the definitions of registers mapped to the PCI Configuration Space.

CPU Host Bus Interface—Describes PCI/AGP bus interface between the Bt2166 and the host CPU. Defines the PCI signal pins and provides timing diagrams.

Synchronous Memory Interface—Describes the interface between the Bt2166 and the SGRAM/SDRAM, which provides access to ROM and multimedia data and control information. Includes timing diagrams and connection diagrams.

Bt2166 Operating Specifications—Provides the operating specifications for the Bt2166, including electrical, thermal, and packaging. Provides PLL tables for the pixel, memory, system, and GUI clocks.

Related Documents

Bt829A VideoStream II Decoder Specification — Rockwell Semiconductor Systems, Brooktree Division

PCI Local Bus Specification, Revision 2.1 — PCI Special Interest Group

Accelerated Graphics Port (AGP) Interface Specification, Revision 1.0 — Intel Corporation

The I^2C Bus Reference Guide — Rockwell Semiconductor Systems, Brooktree Division

Richard F. Ferraro, *Programmer's Guide to the EGA and VGA Cards*, Second Edition (Addison-Wesley Publishing Company, Inc., 1990)

Bradley Dyck Kliewer, *EGA/VGA: A Programmer's Reference Guide* (Intertext Publications, McGraw-Hill Publishing Company, New York, 1990)

Notation Conventions

Example	Description
$\overline{\text{RESET}}$	The overbar above a signal or pin (or a portion of the signal or pin) indicates active-LOW.
8’hAD	Hexadecimal notation used by hardware designers, in the format: ***<num bits>’h<value>*** Where: ***<num bits>*** = number of bits in hex value ***’h*** = hexadecimal ***<value>*** = hex value, each alphanumeric digit represents 4 bits
6’b01011	Binary notation used by hardware designers, in the format: ***<num bits> ’b <value>*** Where: ***<num bits>*** = number of bits in binary value ***’b*** = binary ***<value>*** = binary value of each bit
AB31h	Hexadecimal notation used by assembler programmers, in the format: ***<value>h*** Where: ***<value>*** = hex value, each alphanumeric digit represents 4 bits ***h*** = hexadecimal
0x000AC100	Hexadecimal notation used by C programmers, in the format: ***0x<value>*** Where: ***0x*** = hexadecimal ***<value>*** = hex value, each alphanumeric digit represents 4 bits

TABLE OF CONTENTS

List of Figures

List of Tables

PRODUCT DESCRIPTION

Introduction

The Bt2166 is the newest member of the Brooktree family of high-performance media accelerators. It provides high performance 3D graphics acceleration and enhanced 2D graphics and video performance through new host bus and frame buffer memory interfaces. The host bus interface of the Bt2166 is fully compatible with Intel's Accelerated Graphics Port (AGP) for high-speed 3.3V 66MHz connection to the host system. The memory interface supports both Synchronous Graphics RAM (SGRAM) and Synchronous DRAM (SDRAM) memories for optimum price/performance configurations. For details on the memory controller refer to "Synchronous Memory Interface" on page 331.

IMPORTANT

In this Specification, SGRAM and SDRAM are referred to generically as DRAM, unless the distinction is important to the understanding of the system.

The Bt2166 incorporates advanced 3D game acceleration technologies, including dedicated hardware for triangle setup and edge interpolation and a full-featured rendering engine. It also features support for advanced digital video sources such as DVD and digital satellite TV, making the Bt2166 ideally suited for systems targeting the emerging convergence of computer and consumer electronics applications. Using Brooktree's TrueView™ technology, the Bt2166 achieves video playback quality that surpasses that of component (S-video) sources.

The Bt2166 graphics and video subsystem is optimized for acceleration of Microsoft's DirectX™ and ActiveX™ APIs, including DirectDraw™, DirectVideo™, Direct3D™, ActiveMovie, and ActiveVRML™. DirectDraw performance is optimized with a GUI engine capable of full overlap of application and GUI command execution coupled with a high-performance 64-bit, 85 MHz memory interface. DirectVideo acceleration is provided via on-chip color space conversion and horizontal and vertical interpolators for upscaling video sources up to 720 x 480 resolution for advanced MPEG II applications. Up to two video streams can be accelerated in hardware.

By connecting a Brooktree video decoder such as the Bt829A, the Bt2166 system can be used for applications such as video telephony and cable TV. Using the Brooktree Bt829A VideoStream Decoder, the system supports the Intel/Microsoft/NBC Intercast system to encode data within the vertical blanking interval of broadcast TV and deliver high-bandwidth World Wide Web content to the desktop.

High-Performance 3D Graphics Acceleration

The Bt2166 delivers arcade-quality 3D graphics game performance using the powerful combination of a triangle-setup engine and a tightly-coupled, dedicated, 3D rendering engine to accelerate rendering of perspective-correct, texture-mapped, Z-buffered, alpha-blended, Gouraud-shaded triangles. The combination of 3D hardware setup and rendering engines enables a well-balanced, high-performance 3D graphics system. In general, geometry processing will be performed by the CPU and implementation of both triangle setup and rendering hardware in the graphics controller will help prevent the 3D pipeline performance from being limited by the CPU.

The triangle-setup processor provides line slope and color/texture differential calculations for line, triangle and flat top/bottom quadrilateral primitives and retrieves and stores the triangle list vertex data including coordinates, color, texture, depth, and opacity parameters after polygon geometry processing. The Bt2166 triangle set-up engine is implemented as a programmable, 32-bit RISC pipelined processor, enabling parallel-processing with full overlap of the geometry and rendering tasks. It provides the flexibility to either completely off-load or assist the host CPU in 3D triangle processing. Additionally, the setup coprocessor can perform other tasks such as triangle back-face culling and it can be easily optimized to efficiently handle different data structures related to specific 3D software engines and APIs.

The Bt2166 3D triangle setup engine balances the 3D pipeline performance by freeing up additional host CPU resources to apply to geometry processing and game play tasks e.g. artificial intelligence, audio, communications and user interface I/O. With game play absorbing up to 70% of CPU resources in a typical title, limited processor power is left to perform geometry and triangle setup. A 3D pipeline partitioning using a flexible triangle setup processor in hardware facilitates host geometry processing — enabling a more realistic game playing experience than is possible using accelerators providing rendering only hardware.

The 3D rendering engine is a heavily pipelined, pixel processor which supports both unmeshed and meshed (tri-strip) triangles. The system includes a dedicated triangle-edge interpolation engine and a tightly-linked triangle rasterization engine performing span interpolation, texture mapping, Z-buffering, alpha blending, and lighting tasks. The triangle-edge interpolation engine and setup engine perform the scan conversion process needed to transform a triangle defined by three vertices into a set of data that represents the triangle spans (horizontal strips) required for rendering. The edge interpolation process calculates rendering parameter values required by the span interpolation engine. This includes determination of each set of end point values per span and the value increment step per span for color, texture, depth, and opacity. Interpolation is performed with high precision, sub-pixel accuracy.

Perspective-correct texture mapping is supported with the use of bilinear or trilinear filtering with subtexel accuracy to improve the appearance of the texture-mapped surfaces. Filtering avoids rendering 3D objects with a 'blocky' appearance which results from the use of lower performance, simple point sampling. Texture-mapping performance is further enhanced with the Bt2166 through support of bilinear MIP mapping, which uses a selection of images with predefined versions of the same texture map at varying resolutions. The appropriate texture map resolution is chosen to provide the closest match with the size of the target triangle to be rendered. Since the lower resolution textures are pre-filtered, 3D rendering performance and frame buffer bandwidth are increased and the quality of the texturing is enhanced.

The Bt2166 supports both transparency and translucency of texture maps. Texture transparency enables higher levels of detail by allowing complicated objects to be mapped onto transparent polygons e.g. the texture may have holes through which the underlying background should be visible for realistic effect. A translucent texture can also be applied by alpha blending the texture at varying levels of translucence with an underlying object or scene, enabling realistic fog or smoke effects.

Palletized textures are supported in addition to 16-bit and 24-bit RGB textures. Compared to 16-bit RGB texturing, palletized texturing reduces the memory requirements for texture maps by up to 75%. Accordingly, a larger number of textures can be stored in off-screen memory, and the memory bandwidth requirements for manipulation of texture data can be greatly reduced. With this technique a 4-bit (16 color) or 8-bit (256 color) Color Lookup Table (CLUT) is assigned to each texture in a scene. Typically, a pallet of 16 or 256 colors is more than sufficient to accurately render any given texture with 16-bit or 24-bit color resolution. Hence, palletized textures enable the use of more textures and make available additional memory bandwidth for the rendering engine, thereby providing enhanced realism and rendering performance.

The alternatives to using palletized color texturing are to either sacrifice rendering performance using 16-bit RGB textures or to achieve reasonable rendering performance at the expense of reduced realism in the texture color representation. The Bt2166 can read 4-, 8- and 16-bit textures packed into a 64-bit word. Support of packed texture formats provides texture caching by grouping texels which are spatially adjacent. This will improve the rendering engine throughput when bilinear interpolation is being used by limiting the number of required memory accesses to retrieve texture data.

When rendering objects, the Bt2166 rendering engine calculates and stores a Z or depth value for each pixel in the display to support occlusion of surfaces on obscured objects. Z-buffering is supported in resolutions of 8, 16, or 32 bits/pixel. The Bt2166 also supports texture sorting when the Z-buffering algorithm is being used to further optimize texture caching performance.

The Bt2166 rendering engine provides fixed-color alpha-blending in addition to texture alpha-blending to control the transparency of an object to the underlying scene, allowing realistic rendering of translucent surfaces. Alpha-blending is also used to create atmospheric effects such as fogging and depth queueing. Lighting models can be applied using either flat or Gouraud shading. Color dithering is also available to minimize pixelization and contouring effects in object shading caused by use of limited color resolution and can be applied to both palletized and 16-bit images.

The Bt2166 delivers high-performance acceleration for 3D games under Windows 95 with Microsoft's Direct3D API and offers direct support for the BRender and OpenGL 3D graphics APIs.

High-Performance 2D Graphics Acceleration

The 2D GUI engine optimizes application performance under Windows 95 DirectDraw. The 2D GUI engine incorporates a 64KB command queue enabling full overlap of application and GUI command execution. It also includes transparent BLT support, a 2 operand BitBLT engine, text BLT, line drawing, solid/pattern fill, and color expansion.

Video Playback and Display

The Bt2166 supports advanced video playback hardware acceleration and video display processing through color space conversion, scaling, and TrueView technology for interlaced video source processing. DirectVideo acceleration of software video codecs is provided by YCrCb to RGB color conversion and high-quality image upscaling. Native video source formats can be scaled up to 1024 x 768 full screen resolution using an advanced 2-dimensional video filter. Video scaling is implemented using 3-tap horizontal and 3-tap vertical interpolation filtering. Video quality is further enhanced with advanced image filtering to improve picture sharpness. The result is a high-resolution PC video image display which rivals the fidelity of high end TVs.

The Bt2166 enhances TrueView's technology for interlaced-to-progressive scan conversion. TrueView enables the display of full CCIR601 resolution video from an MPEG2 video decoder or Bt829A VideoStream decoder without intermediate processing, hence, preserving the inherent quality of the YCrCb 4:2:2 video stream delivered by the decoder. TrueView uses vertical scaling and filtering and the locking of graphics frame timing to the video source input field timing to overcome visible scan conversion artifacts, such as interfield motion and tearing.

The hardware can support two video windows with double-buffering. Mixing of video and graphics can be implemented using either color keying or chroma keying techniques. This technology directly addresses the requirements of high-fidelity video display on progressive monitors in the PC-TV consumer product category. The hardware can support two video windows with double-buffering. Mixing of video and graphics can be implementing using either color keying or chroma keying techniques.

A flexible video peripheral interface allows a wide range of MPEG video decoder hardware and the Brooktree Bt829A VideoStream decoder to be gluelessly attached to the Bt2166. The Bt2166 is tailored for use with popular MPEG2 video decoder chips for digital satellite TV and DVD movie player applications, and with the Bt829A for analog TV, Intercast, and video phone applications.

Figure 1. Bt2166 Configuration

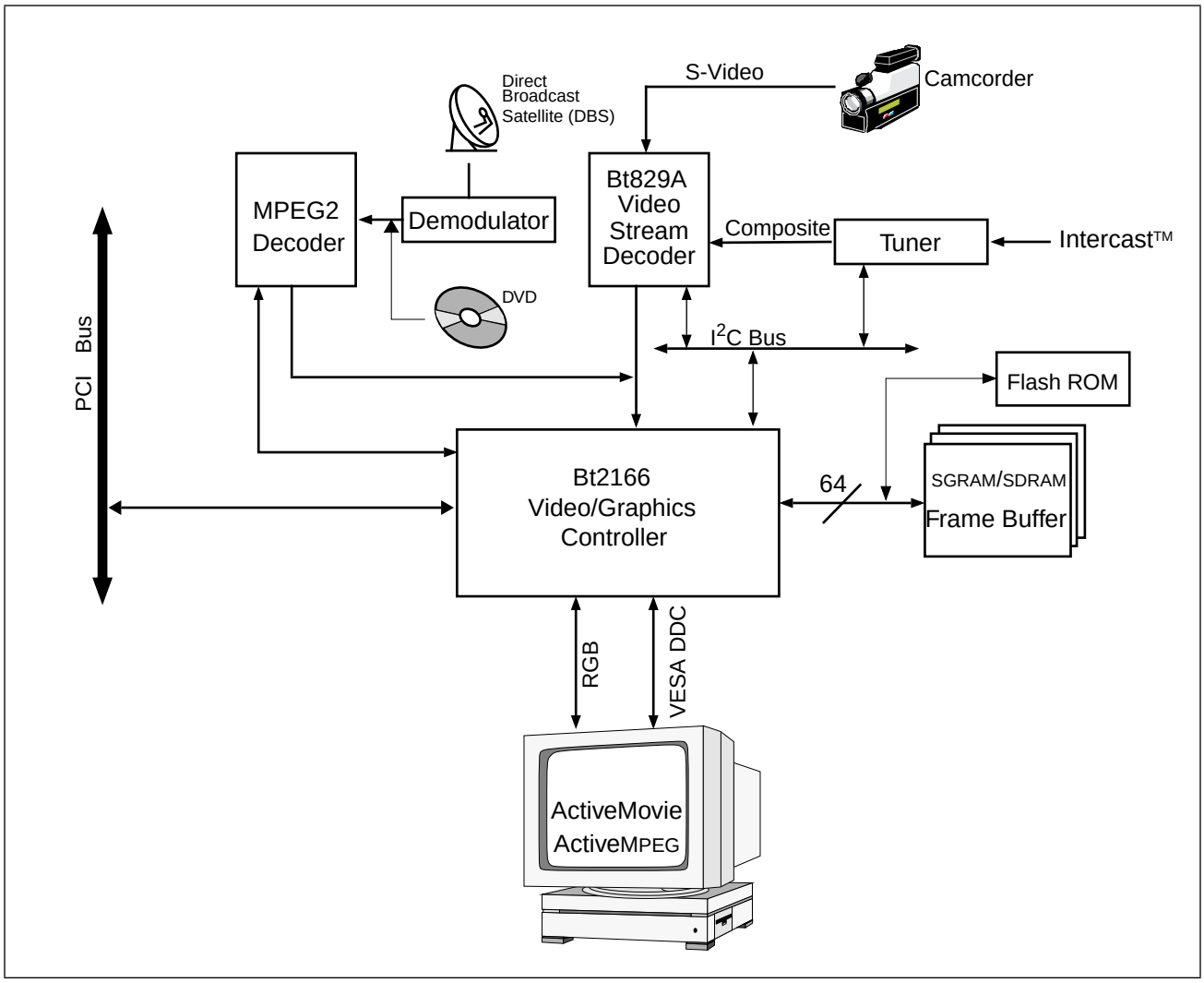
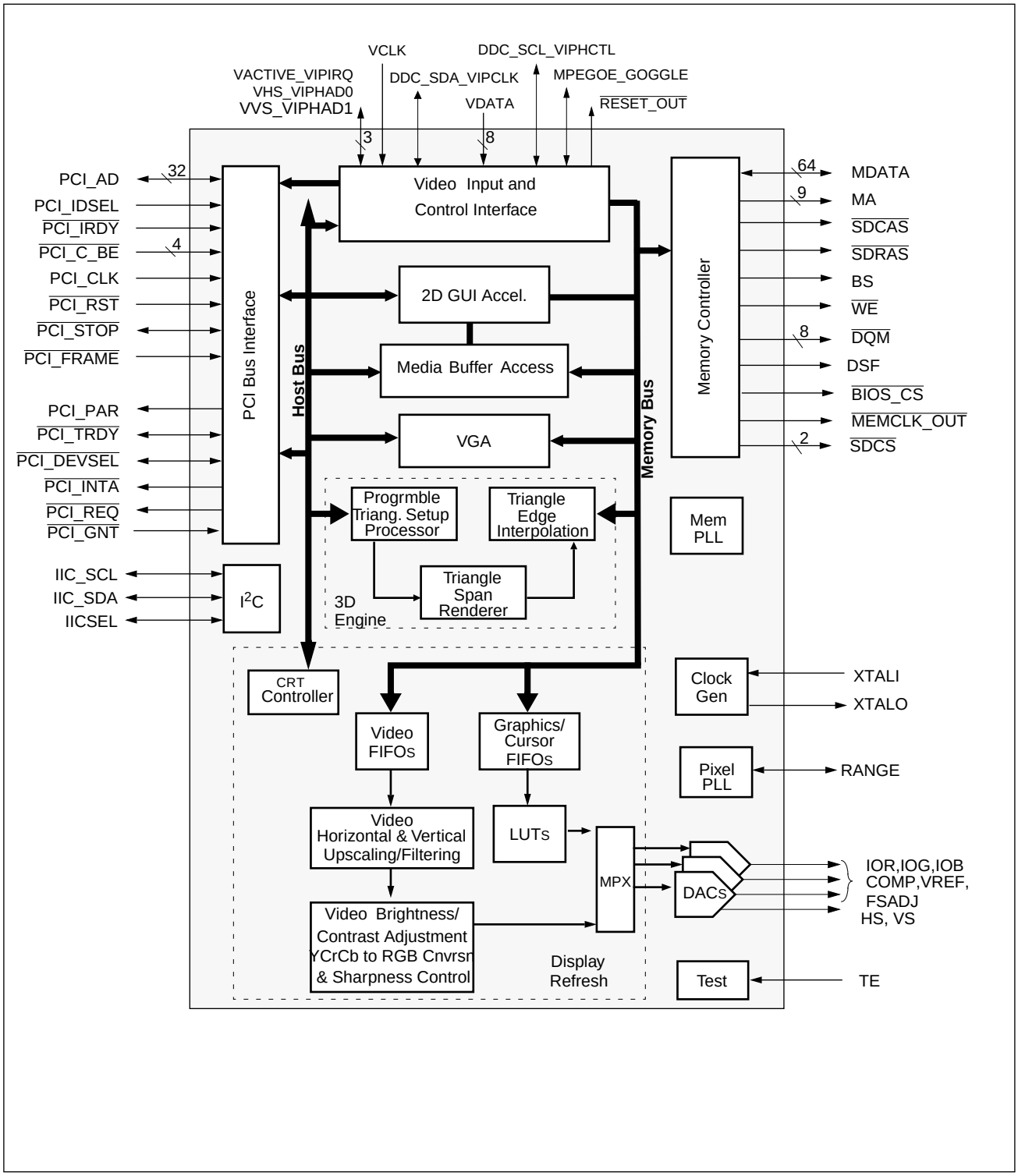


Figure 2 presents a block diagram of the PCI/AGP-bus Bt2166 configuration.

Figure 2. Bt2166 Block Diagram



Interface Description

The Bt2166 provides five hardware interfaces:

- CPU or Host bus interface
- Programmable multimedia interface
- SGRAM/SDRAM memory interface
- BIOS ROM interface
- I^2C interface

These interfaces are described briefly here; however, for more detail refer to subsequent chapters of this document.

CPU/Host Interface

The Bt2166 supports a 32-bit host bus interface configurable for either PCI 2.1 compliant operation or 3.3 V 66 MHz Intel Accelerated Graphics Port (AGP) operation. The Bt2166 is 3.3 V part with 5V-tolerant inputs on the host interface pins. PCI bus mastering capability enables the Bt2166 to efficiently retrieve source texture maps and command lists from system memory without CPU intervention. Similarly, the PCI bus master can deliver captured video directly into system memory in a video phone application without involving the host in the process. The Intel AGP bus configuration allows the Bt2166 to provide optimal 3D graphics performance.

Programmable Multimedia Interface

The Bt2166 provides a flexible, programmable multimedia interface for glueless attachment of video peripherals such as MPEG/DVD video decoder hardware and Brooktree Bt829A VideoStream decoder hardware. The interface contains an 8-bit wide data bus for multiplexed luminance and chrominance input video data, a video pixel clock signal, video synchronization signals and a serial host interface bus. The interface can operate in one of the following four modes.

1. The VIP mode supports the Video Interface Port interface standard. The VIP mode can be used to provide both a video data and host interface connection to an MPEG2 video decoder enabling an entire MPEG2 decoding subsystem to be integrated onto a single PCI/AGP card with the Bt2166 graphics system. In VIP mode, the second I^2C port is not available because the pins are used as part of the VIP interface.

2. ByteStream™ mode requires use of the video clock and video data pins only. This mode is used to interface to Brooktree VideoStream decoders (Bt829/Bt829A).

3. CCIR656 mode requires only the video clock and video data pins. The CCIR656 mode supports EAV/SAV codes for video synchronization information and provides interface compatibility with many MPEG video decoder ICs.

4. A programmable mode provides complete flexibility to tailor the interface to the requirements of any video decoder peripheral hardware. In this mode, the video synchronization control signals can be configured as inputs or outputs depending upon whether the video peripheral device requires the Bt2166 to be the master timing device or slave timing device.

Memory Interface

The Bt2166 supports a 64-bit-wide memory interface for both SGRAM and SDRAM at memory bus speeds up to 85 MHz. The memory bus supports both 2MB and 4MB configurations. The 2MB configuration supports display resolutions up to 1024 x 768 for 16-bit color and 1280 x 1024 for 8-bit color.

A broad range of 16 Mbit 256 K x 32 SGRAM devices are supported including:

- Samsung KM4132G271-12/-15
- Micron MT4KC256K32D4-12/-15
- Hitachi HM5283206FP-12/-15
- IBM IBM038329ON6-12/-15
- NEC UPD481850GF-A10-JBT
- Fujitsu MB11643241A/4A

BIOS ROM Interface

The Bt2166 supports attachment of several different ROM types and the BIOS fits into 32 KB of space. Accordingly, 32 KB ROM may be used. For maximum compatibility, 200 nsec ROMs should be used.

I^2C Interface

The Bt2166 supports two I^2C interfaces. One interface supports VESA DDC1, DDC2A and DDC2B monitor protocols. The other is for communication with I^2C compatible video peripherals such as the Brooktree Bt829A VideoStream decoder and analog TV tuners.

Display Mode Support

The Bt2166 integrates a high-performance 173 MHz RAMDAC and programmable clock synthesizer to support all VESA display modes up to 1600 x 1200 resolution at 65 Hz refresh.

Bt2166 Pin Layout and Descriptions

Figure 3 illustrates the pin assignments for the PCI/AGP bus interface to the Bt2166. The overbar above a signal indicates active-low. A descriptive summary of each pin on the bus is provided in Table 1.

Figure 3. Bt2166 PCI-Bus Pin Layout

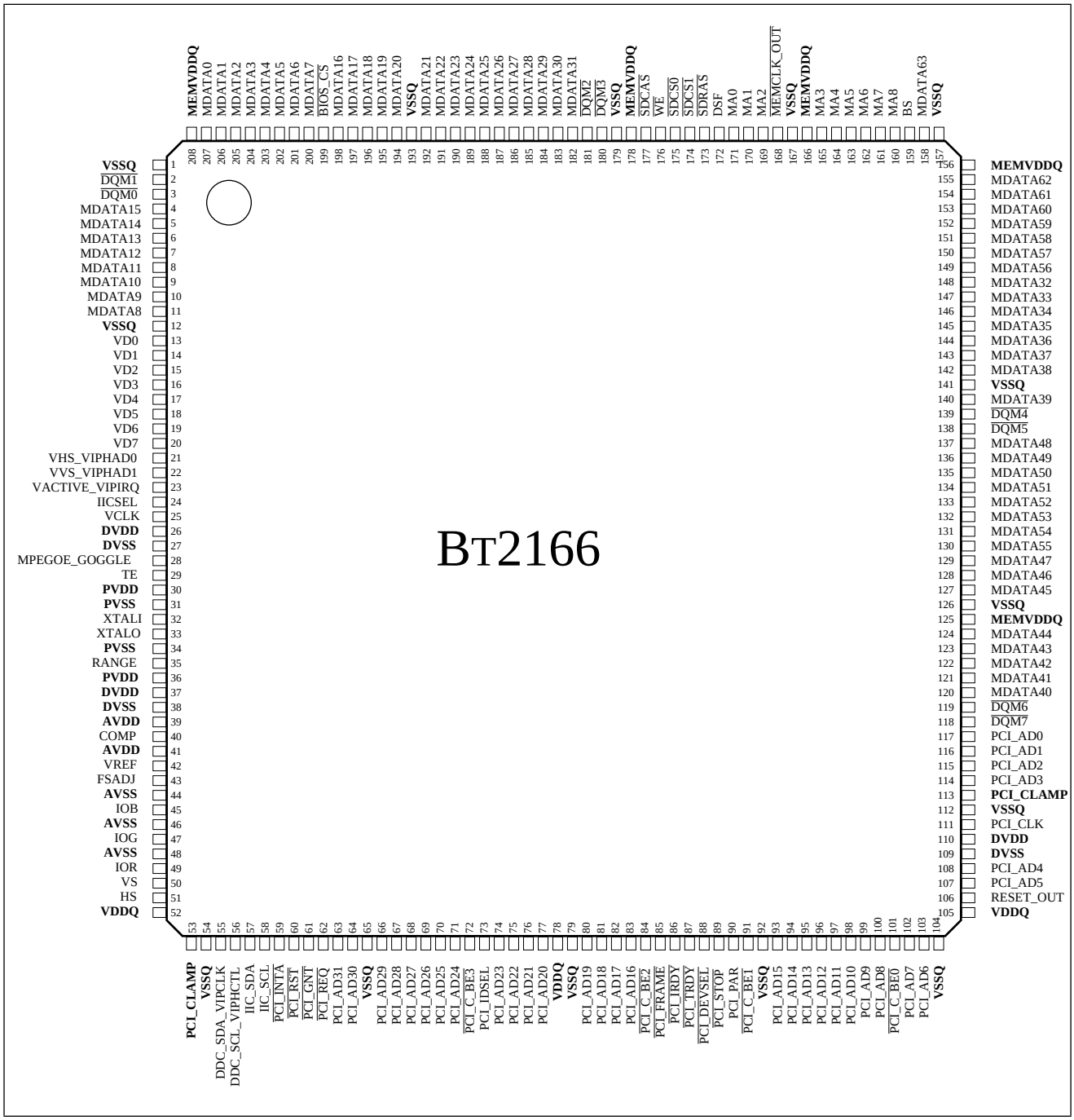


Table 1. Bt2166 PCI/AGP-Bus Pin Descriptions *(1 of 4)*

	Pin Number	Signal Name	I/O	Description
PCI Pins = 48	63,64,66-71,74-77,80-83,93-100,102,103,107,108,114-117	PCI_AD[31:0]	I/O-3S	PCI address and data bus.
	72,84,91,101	$\overline{\text{PCI\_C\_BE}}$[3:0]	I/O	PCI command and byte enables.
	89	$\overline{\text{PCI\_STOP}}$	I/O-3S	PCI stop.
	90	PCI_PAR	O-3S	PCI parity.
	87	$\overline{\text{PCI\_TRDY}}$	I/O-3S	PCI target ready.
	88	$\overline{\text{PCI\_DEVSEL}}$	I/O-3S	PCI device select.
	86	$\overline{\text{PCI\_IRDY}}$	I/O	PCI initiator ready.
	85	$\overline{\text{PCI\_FRAME}}$	I/O	PCI cycle frame.
	73	PCI_IDSEL	I	PCI initialization device select. Connect to PCI_AD[16] when interfacing to AGP.
	59	$\overline{\text{PCI\_INTA}}$	OD	Interrupt pin A.
	62	$\overline{\text{PCI\_REQ}}$	O	PCI master request.
	61	$\overline{\text{PCI\_GNT}}$	I	PCI master grant.
	111	PCI_CLK	I	PCI clock.
	60	$\overline{\text{PCI\_RST}}$	I	PCI reset pin.
Special Pins = 10	57, 58	IIC_SDA, IIC_SCL	I/O	I^2C interface pins used for I^2C bus.
	55, 56	DDC_SDA_VIPCLK, DDC_SCL_VIPHCTL	I/O	I^2C interface pins used for I^2C bus for support of VESA DDC1, 2A, 2B signaling modes, or used for providing clock and control signals for the VIP interface.
	28	MPEGOE_GOGGLE	I/O	Video output enable for external MPEG decoders, or used to drive stereographic headgear.
	24	IICSEL	I/O	This pin detects whether the VIP interface is in use, and selects the external I^2C multiplexer when VIP is used.
	29	TE	I	Test mode enable. Pull low for normal operation.
	106	$\overline{\text{RESET\_OUT}}$	O	Reset to supported peripherals: Bt829A, etc.
	32, 33	XTALI, XTALO	A	Crystal oscillator.

Table 1. Bt2166 PCI/AGP-Bus Pin Descriptions *(2 of 4)*

	Pin Number	Signal Name	I/O	Description
DAC Pins = 9	49,47,45	IOR,IOG,IOB	A	DAC red, green, and blue analog outputs.
	42	VREF	A	Voltage reference. If an external voltage reference is used, it must supply this input with a 1.235 V (typical) reference.
	35	RANGE	A	PLL range adjust.
	51	HS	O	Horizontal Sync output (TTL compatible). Programmable to active high or active low; default active low.
	50	VS	O	Vertical Sync output (TTL compatible). Programmable to active high or active low; default active low.
	40	COMP	A	DAC compensation terminal.
	43	FSADJ	I	Full scale adjust.
Multimedia Bus Pins = 12	20-13	VDATA[7:0]	I	Video input on the multimedia interface bus. Signals are pulled up internally (CMOSP); however, external configuration pulldown resistors are applied to these signals at the board level to tell the Bt2166 whether external video hardware is present, whether the 2166 is attached to a 33MHz PCI or 66MHz AGP bus, and to tell it what type of SGRAM is present on the board.
	21	VHS_VIPHAD0	I/O	External video horizontal sync. In non-interlaced mode, the signal is VHS_VIPHAD0; in interlaced mode, the signal is FIELD. Programmable to active high or active low; default active low. Alternately used has VIP host address/data bit 0.
	23	VACTIVE_VIPIRQ	I/O	External video active signal. Alternately used as VIP Interrupt.
	22	VVS_VIPHAD1	I/O	External video vertical sync. Programmable to active high or active low; default active low. Alternately used as VIP host address/data bit 1
	25	VCLK	I	Multimedia video clock. Used to transfer data on the video input bus. This pin is pulled up internally.

Table 1. Bt2166 PCI/AGP-Bus Pin Descriptions *(3 of 4)*

	Pin Number	Signal Name	I/O	Description
SGRAM Interface Pins = 90	158, 155-149,130-137,129-127,124-120,140,142-148,182-192,194-198, 4-11	MDATA[63:8]	I/O	Bidirectional tri-state memory data. Due to address line loading and pin limitations, address lines for Flash ROM are multiplexed on MDATA[35:16].
	200–207	MDATA[7:0]	I/O	Bidirectional tri-state SGRAM and ROM data.
	160-165,169-171	MA[8:0]	O	Multiplexed memory address lines to SGRAM capable of driving up to 4 SGRAMs.
	159	BS	O	SGRAM bank select (A9)
	174,175	$\overline{\text{SDCS}}$[1:0]	O	SGRAM chip selects.
	118,119,138, 139,180,181,2, 3	$\overline{\text{DQM}}$[7:0]	O	SGRAM data input/output mask.
	176	$\overline{\text{WE}}$	O	Write enable.
	177	$\overline{\text{SDCAS}}$	O	Column access strobe, also used for output enable on ROM.
	173	$\overline{\text{SDRAS}}$	O	Row address strobe.
	172	DSF	O	Define special function output to SGRAM.
	168	$\overline{\text{MEMCLK\_OUT}}$	O	RAM clock. Inverted from internal SYSCLK.
	199	$\overline{\text{BIOS\_CS}}$	O	ROM chip select.

Table 1. Bt2166 PCI/AGP-Bus Pin Descriptions *(4 of 4)*

	Pin Number	Signal Name	I/O	Description
Non-Signal Pins = 39	39, 41	AVDD(1)	P	Analog supply.
	26, 37, 110	DVDD(1)	P	3.3V Digital core supply.
	30, 36	PVDD(1)	P	3.3V PLL supply.
	125, 156, 166, 178, 208	MEMVDDQ(1)	P	3.3V Memory I/O power.
	52, 78, 105	VDDQ(1)	P	3.3V bus I/O power. Hooked to a separate power plane on an AGP bus board.
	53,113	PCI_CLAMP	P	PCI bus I/O clamping pin. Tied to 5V on 5V PCI bus board, 3V on an AGP bus board, or $V_{I/O}$ on a PCI bus universal board.
	44, 46, 48	AVSS(1)	P	Analog ground.
	27, 38, 109	DVSS(1)	P	Digital core ground
	31,34	PVSS(1)	P	PLL ground.
	1, 12, 54, 65, 79, 92, 104, 112, 126, 141, 157, 167, 179, 193	VSSQ(1)	P	Digital I/O ground.

Note: (1). AVSS, DVSS, PVSS, VSSQ pins must be connected to the same ground plane. All VDD (AVDD, DVDD, PVDD, and VDDQ) pins must be connected to the same power plane on a PCI board. VDDQ should be connected to a separate power plane on an AGP bus board, as per the AGP specification.
2. I = Input, O = Output, 3S = Tri-state, OD = Open Drain, A = Analog, NC = No Connect, P = Power in Volts.

CPU ADDRESS SPACE APERTURES

Aperture Space Theory of Operation

This section describes the programming interface to the Bt2166 graphics/video accelerator, the Media Buffer Access (MBA) unit, and the Flash ROM. As in conventional VGA systems, there is a CPU bus aperture at 000A0000h–000BFFFFh through which the VGA frame buffer is accessed, which we shall call the real mode aperture. The VGA ROM is accessed through the PCI Expansion ROM Aperture and can be moved to any 16 MB boundary. The design requires that the Bt2166 is attached to the CPU via a PCI bus. Furthermore it is assumed that the GUI and MBA are accessed from protected mode well above the base memory of the system through the protected mode aperture.

The Bt2166 implements several VGA standard I/O ports for VGA compatibility. Some extensions have been added to the VGA registers to configure the Bt2166 and implement I^2C transactions. Refer to “I/O Mapped Register Definitions” on page 37.

Throughout this document, the hardware register notations GUI_BASE, PM_BASE, PM_ENABLE, and GBASE are continuously referenced. GUI_BASE is a 32 bit internal register that contains the PM_BASE (Protect Mode Base Address), PM_ENABLE (Protect Mode Enable Bit), and the real mode aperture enables and pointers. The GUI_BASE register is accessed through the VGA Extension Registers; refer to “GUI Base Address Registers” on page 46.

PM_BASE (Protect Mode Base Address) is the upper 7 bits of the GUI_BASE[31:25] register. It is also the same as the upper 7 bits of the PCI_BASE0[31:25] register; refer to “PCI Base Address Register Zero” on page 321. Typically, it is system BIOS that sets the value of PM_BASE, through the PCI_BASE0 register, when the system is booted. However, it is possible to read and write PM_BASE through the VGA extension accesses to the GUI_BASE.

The PM_ENABLE (Protect Mode Enable) bit enables the protect mode aperture. This bit is set automatically when the upper 7 bits of the PCI_BASE0 register are written to a non zero value, insuring that the card does not conflict with lower system memory. PM_ENABLE is accessible as part of the GUI_BASE[24] register.

GBASE is the base address of the protected mode aperture in CPU address space and is the same value as the PCI_BASE0. All memory mapped registers, frame buffer memory, and ROM space ARE referenced as an offset from GBASE.

GRP_GUI_BASE is a software graphics indice used for to read or write bytes within the GUI_BASE register.

REMINDER

The identifiers GUI_BASE, GRP_GUI_BASE, and GBASE refer to different entities (as stated above).

As shown in Figure 4, the ROM aperture is relocated to high memory (protected mode) for a PCI/AGP-bus. It is the system BIOS' responsibility to copy the VGA BIOS from this high memory location to system RAM and map it to 000C0000h to 000C7FFFh. PCI systems should never enable the ROM HOLE.

Figure 4. CPU Apertures on a PCI/AGP System

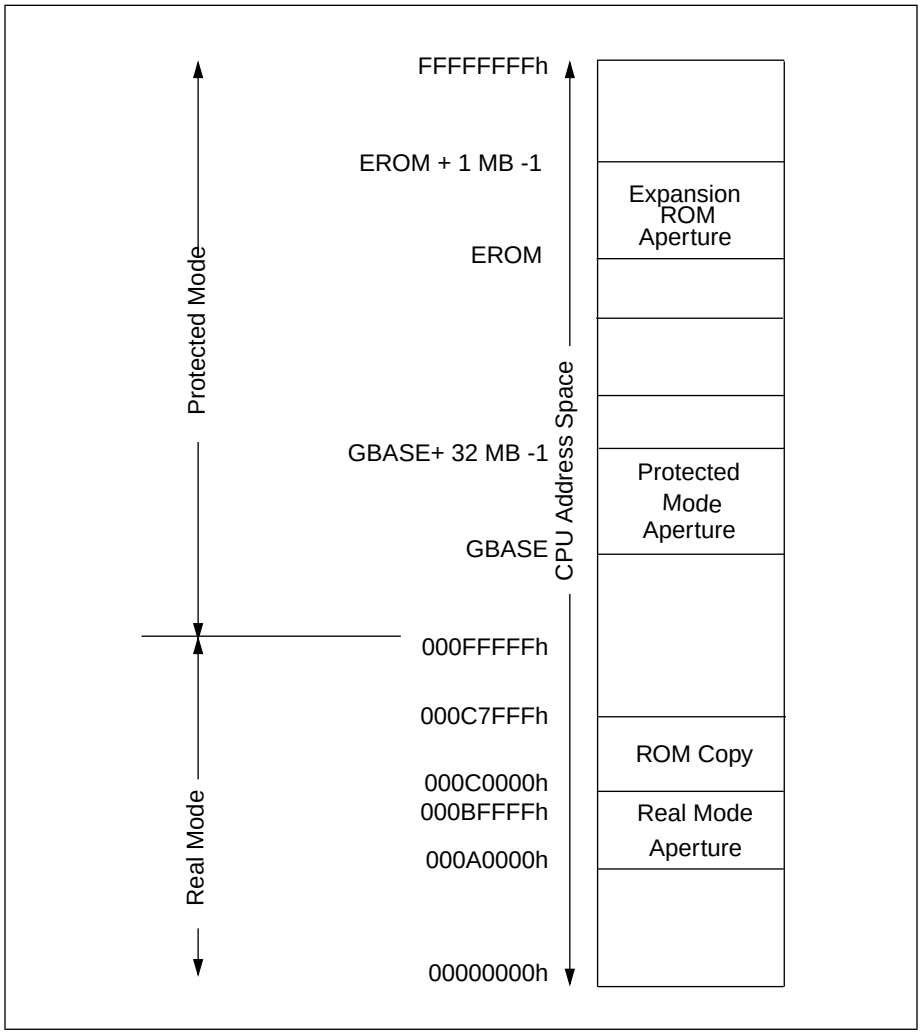
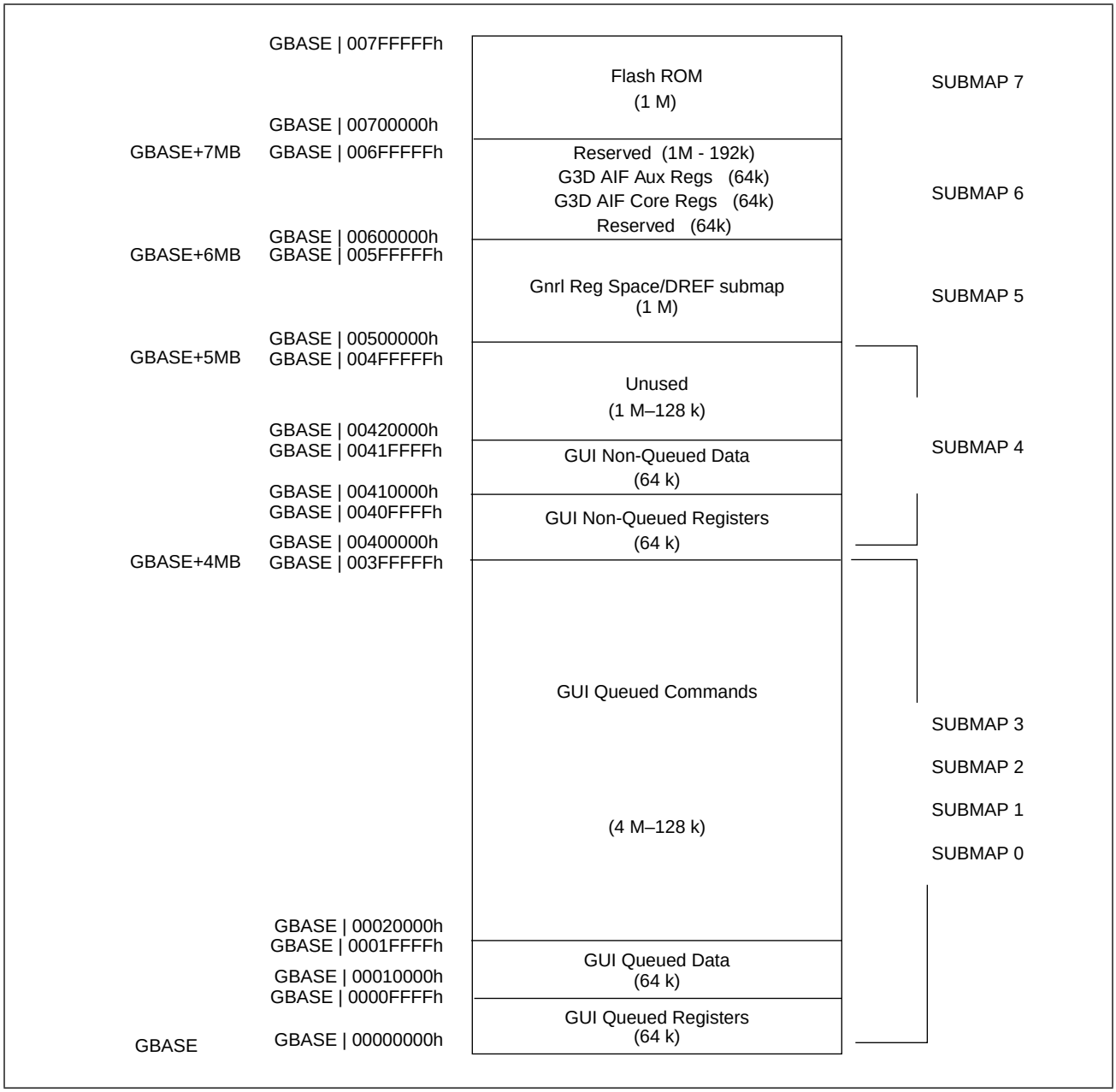


Figure 5 shows the contents of the eight 1MB submaps (numbered 0 through 7) of the GUI memory map. Each submap contains 16 blocks, each 64 K long. Refer to ‘‘GUI Command/Register Map’’ on page 105 for details on GUI submaps zero through four.

Figure 5. Memory Map



Protected Mode Aperture

CPU accesses are applied to the host interface module PCI/AGP bus. The host interface gives rise to an internal tri-state bus called the HBUS. A number of chip subsystems or functional "agents" are attached to the HBUS, such as the VGA controller and the GUI accelerator, shown in Figure 6. The protected mode aperture provides access to any of the functional agents *except* the VGA. VGA and devices on the VGA extension bus are accessed through the VGA I/O registers; refer to ''I/O Mapped Register Definitions'' on page 37.

Figure 6. Bt2166 HBUS Agents

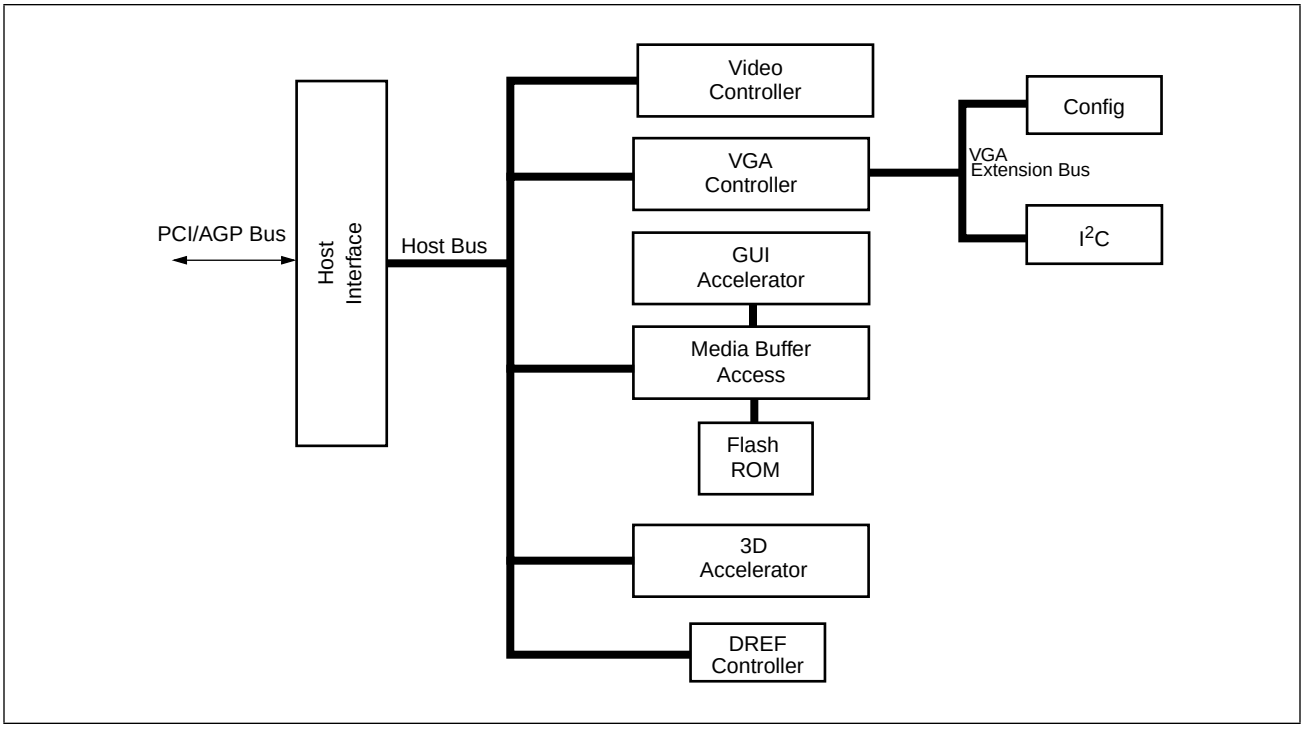
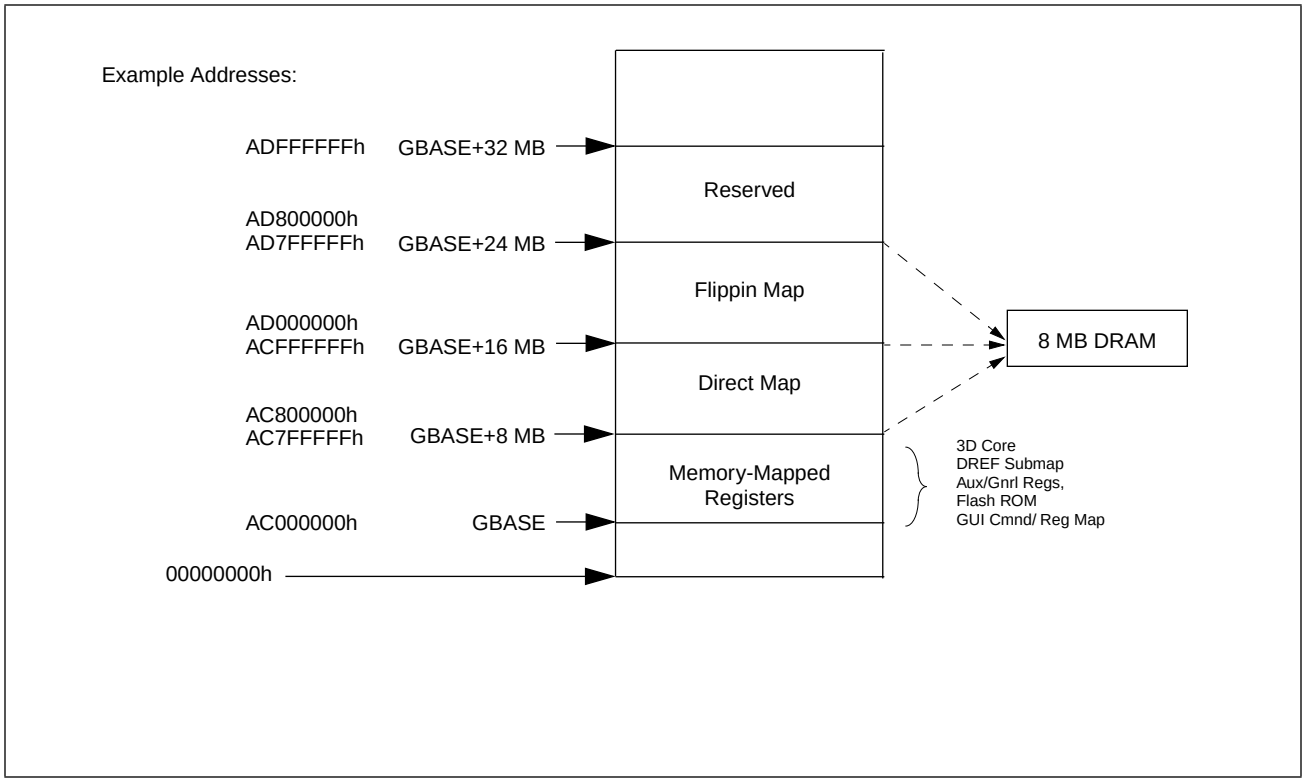


Figure 7 shows how the 32 MB protected mode aperture is broken into four 8 MB maps. Through the lowest addressed map passes all commands and data destined for the GUI accelerator, 3D accelerator, DREF, and all other memory-mapped registers. Two of the maps provide different views of the 8 MB Media Buffer and one map is reserved. Bt2166 supports up to 8 MB of DRAM Media Buffer memory. With an 8 MB Media Buffer address map, the Bt2166 supports video resolutions up to 1280x1024x32.

Looking at the example addresses in Figure 7, AC000000h to AC7FFFFFh maps the GUI command/register space, 3D core /aux registers, general register information, DREF submap, and flash ROM. The map from AC800000h to ACFFFFFFh is a direct mapping of the Media Buffer. AD000000h to AD7FFFFFh, called the Flippin Map, also completely maps the Media Buffer; however, bit, byte, and word data swapping is allowed in this region.

Having the ability to support data bus steering for word, byte, and bit flipping is a distinct advantage for supporting arbitrary image formats. Such steering is controlled from deep within the device driver. Such modes make supporting both *little endian* and *big endian* systems practical from a single controller design.

Figure 7. Protected Mode Aperture Mapping To DRAM



Media Buffer Maps

The following subsections contain explanations and diagrams for the Direct Map and Flippin Map.

Direct Map

The Direct Map covers all of the Media Buffer memory maps and gives a clean, unmodified view of the entire 8 MB the DRAM Media Buffer. Figure 7 depicts the Direct Map covering the DRAM. Bytes, bits, and words are in their natural 486 little endian order with the least significant byte in the 486 EAX register mapping to the left most pixel position when a dword move instruction is used. No word, byte, or bit steering is performed on this Direct Map so regardless of settings in the windows driver for the Flippin Map devices and modules that wish to see an unmodifiable little endian view of the DRAM can *always* do so through the Direct Map.

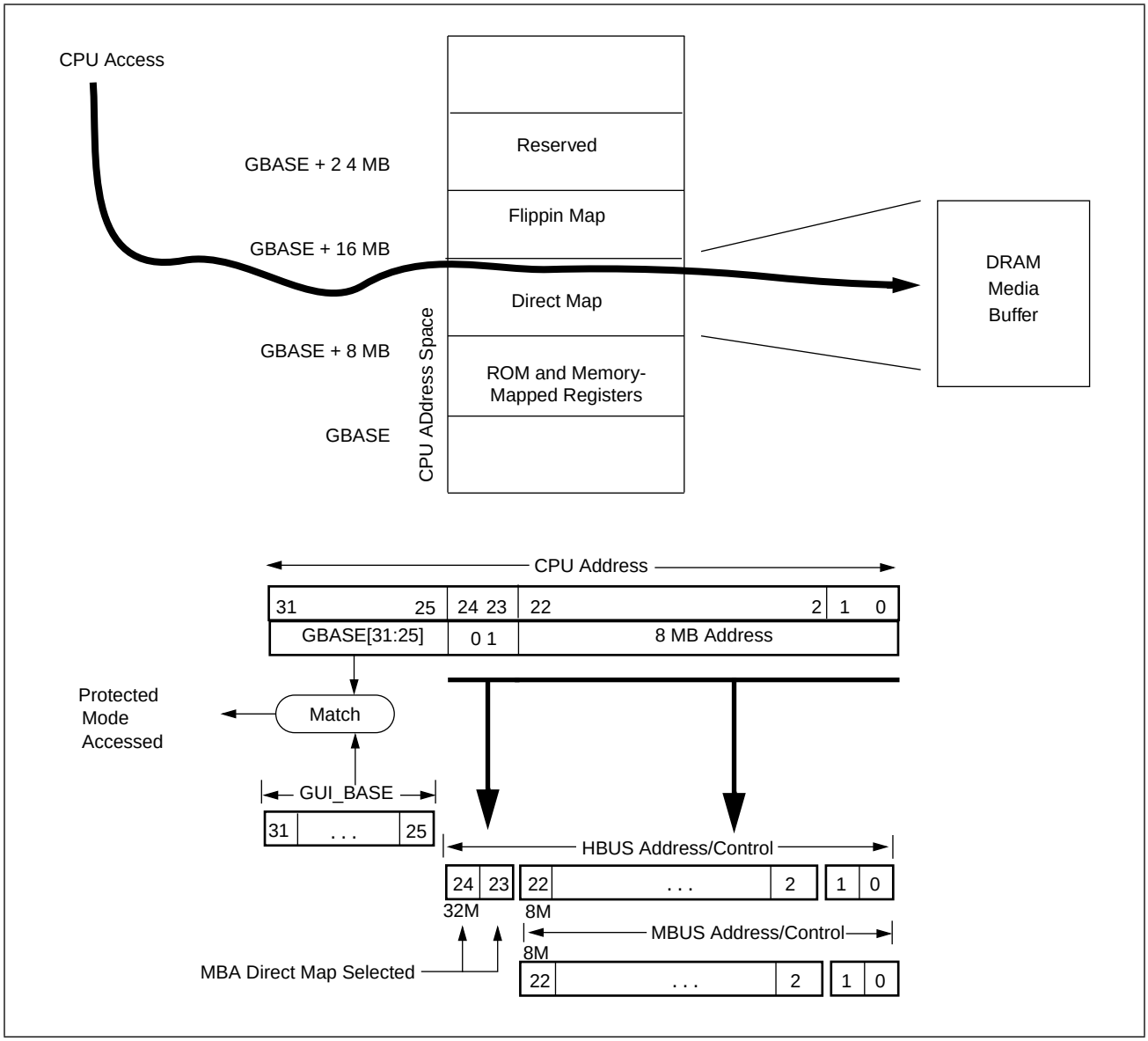
Figure 8 shows that the CPU address bits 31:25 must match GUI_BASE bits 31:25 (PM_BASE) for a protected mode aperture access to be accepted. Once accepted, CPU address bits 24:2 are passed to the internal HBUS, shown in Figure 6. In the Media Buffer access module CPU address bits 24:23 are decoded to determine the type of access to perform. When the Direct Map is selected, CPU address bits 21:2 are passed directly to the internal MBUS for use as a DRAM address. Thus, CPU address bits 24:23 must equal 2'b01 for the Direct Map to be selected.

Finally notice that CPU address bits 21:2, in conjunction with the CPU byte enables determine which byte(s) of the DRAM Media Buffer are accessed through the Direct Map. Because the entire DRAM memory is accessible through the protected mode aperture without having to resort to any page register mechanism, it is referred to as having a "flat view" of the Media Buffer.

Having a separate map that is unchangeable is useful for multimedia accelerators that reside on the PCI. A PCI-resident hardware video codec can get "unmutilated" access to the video-in and video-out buffers in the Media Buffer.

Software video codecs reside in different driver modules from the windows driver code. If multiple drivers are used, the flippin map can be used for video when the Media Buffer is being accessed.

Figure 8. Media Buffer Access Via Direct Map



Flippin Map

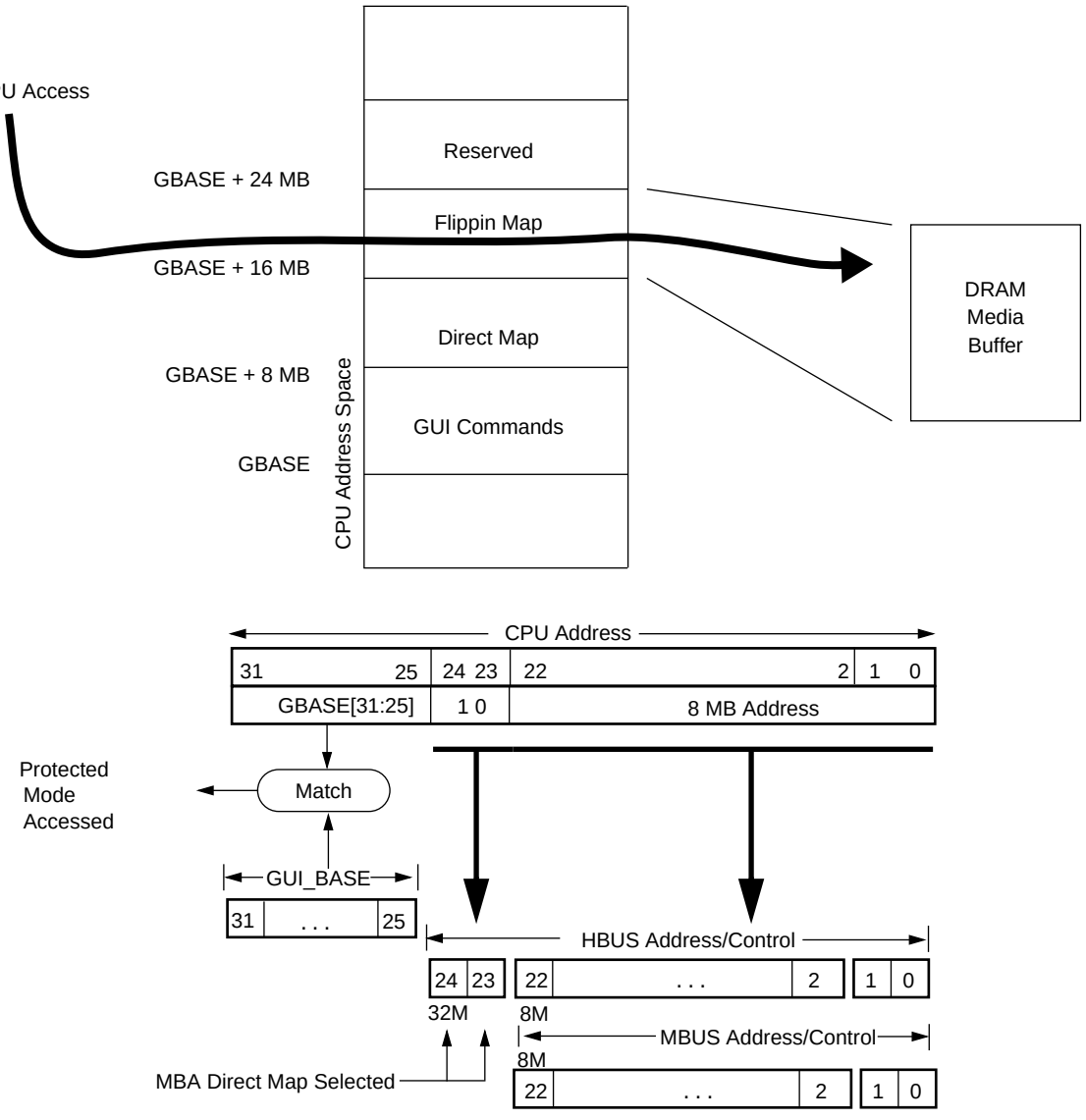
Like the Direct Map, the Flippin Map accesses the Media Buffer memory and gives a flat view of the entire 8 MB DRAM Media Buffer. However, unlike the Direct Map, the windows driver can control the data path steering in Bt2166 so that bits can be swapped within a byte, bytes can be swapped within a word, or words can be swapped within a dword.

Figure 9 depicts the Flippin Map covering the DRAM. In a fashion similar to the Direct Map, notice that the CPU address bits 31:25 must match the GUI_BASE register bits 31:25 (PM_BASE) for a protected mode aperture access to be detected. Notice that CPU address bits 24:23 must equal 2'b10 for the Flippin Map to be selected.

CPU address bits 21:2 are passed to the MBUS and ultimately to the memory controller in an unmodified way. The Flippin Map word and byte flip bits in GUIREG_MBA can cause the data path byte lanes and the byte enables to be re-ordered to effect a byte or word swap within a dword.

Finally, the GUIREG_MBA Flippin Map bit flip bit can cause the data path to bit reverse the word being written or read from the DRAM. The same byte of DRAM can be accessed through either the Direct Map or the Flippin Map; however, the dwords viewed through the Flippin Map may have different orderings of bits, bytes, words, within a dword than the same dword viewed through the Direct Map.

Figure 9. Media Buffer Access Via Flippin Map



Note: Data path steering can cause bit, byte, and/or word flipping.

BIOS ROM Support

Submap 7 at GBASE+7MB is used to access all of the BIOS PROM residing on the DRAM data bus. In the Bt2166, this can be up to 1 MB of ROM. The ROM in this submap is used to hold the Windows NT hardware abstraction layer. The VGA BIOS ROM is visible as the first 32 k of this Flash ROM space.

The BIOS ROM can be one of several types, including Flash ROM, EPROM, or OTP ROM. When Flash ROM is used, it can be read or written through the protected mode aperture. Writing is only enabled when GRP_CFG4[6] is also set.

Note: Flash ROM can only be written after the unlock sequence (see "Unlock Register" on page 60) has been sent to it and must be written in "sectors," refer to the appropriate Flash ROM vendor's specification.In addition, Flash ROM writes are inhibited by the default state of configuration register GRP_CFG4 bit 6 (see Table 26 on page 53).

Figure 10 shows that up to 1 MB of BIOS ROM can be accessed when the CPU address bits 31:25 match the GUI_BASE register bits 31:25 (PM_BASE) and CPU address bits 24:20 equal 5'b00111. CPU address bits 17:2 and the byte enables are passed out the DRAM data bus so that 1 MB is addressed for read or write.

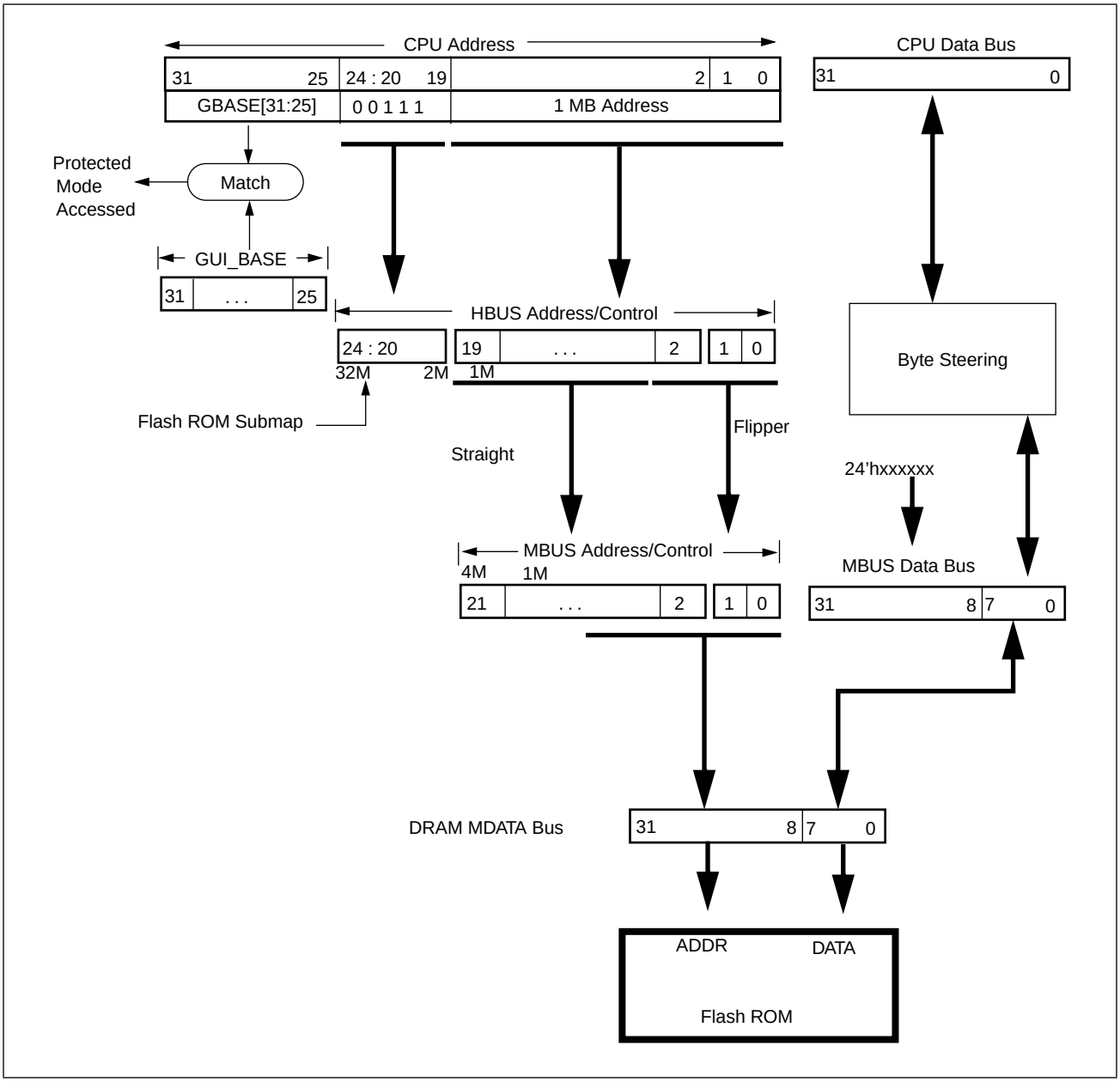
If we make the following C definitions:

```
#define PM_ROM 0x00700000 // PROT MODE ROM
```

Then location 300 in the ROM can be accessed through the protected mode as follows:

```
((unsigned char *) (GBASE | PM_ROM | 0x00000300L))
```

Figure 10. Flash ROM Access Address Mapping



Real Mode Aperture

The VGA aperture mechanism allows access to the 32MB protected mode aperture from 486 real mode. This mechanism involves changing the interpretation of the VGA memory aperture at 000A0000h–000BFFFFh so that it provides access to the various HBUS agents instead of to the VGA frame buffer. Two VGA apertures are provided to offer the most robust support of the VESA BIOS modes.

For 64 K mode, two 64 KB apertures are available for accessing different HBUS agents or for accessing different places within a single HBUS agent. This access of HBUS agents is enabled when GUI_BASE[18] is set to one and GUI_BASE[21] is set to zero. REAL_MODE_APERTURE0 responds to CPU reads/writes in the address range 000A0000h through 000BFFFFh. The HBUS is accessed by a 25-bit address; GUI_BASE[8:0] (REAL_MODE_APERTURE0) defines the upper 9 bits and CPU address [15:0] defines the lower 16 bits to form the HBUS address as follows:

```
HBUS_ADDR[24:0]={GUI_BASE[8:0],CPU_ADDRESS[15:0]}
```

Note: CPU address[1:0] is actually expressed via byte enables

The resulting address is fed to the HBUS decoder as if it had come through the protected mode aperture. Thus when GUI_BASE[8:0] is loaded with 071h and when the CPU accesses VGA_APERTURE0, then CPU reads/writes to address range 000A0000h–000AFFFFh will access the HBUS agent (Flash ROM) at ROM locations 10000 through 1FFFF, i. e., the second 64 K byte block (refer to Figure 5 on page 17).

64K mode REAL_MODE_APERTURE1 responds to CPU reads/writes in the address range 000B0000h through 000BFFFFh. The HBUS is accessed by a 25-bit address; GUI_BASE[17:9] (REAL_MODE_APERTURE1) defines the upper 9 bits and CPU address [15:0] defines the lower 16 bits to form the HBUS address as follows:

```
HBUS_ADDR[24:0]={GUI_BASE[17:9],CPU_ADDRESS[15:0]}
```

This mechanism is useful for accessing all parts of Bt2166 when the CPU is running in real mode, as occurs at power on.

In 32 K mode, access of HBUS agents is enabled when GUI_BASE[20] and GUI_BASE[21] are set to one. REAL_MODE_APERTURE0 responds to CPU reads/writes in the address range 000A0000h and 000A7FFFh. The HBUS is accessed by a 25-bit address; GUI_BASE[9:0] (REAL_MODE_APERTURE0) defines the upper 10 bits and CPU address [14:0] defines the lower 15 bits to form the HBUS address as follows:

```
HBUS_ADDR[24:0]={GUI_BASE[9:0],CPU_ADDRESS[14:0]}
```

32 K mode REAL_MODE_APERTURE1 responds to CPU reads/writes in the address range 000A8000h–000AFFFFh. The HBUS is accessed by a 25-bit address; GUI_BASE[19:10] (REAL_MODE_APERTURE1) defines the upper 10 bits and CPU address [14:0] defines the lower 15 bits to form the HBUS address as follows:

```
{GUI_BASE[19:10],CPU_ADDRESS[14:0]}
```

The real mode aperture mechanism has many applications. For example, it is useful for BIOS routines that want to read more ROM data than is available in the VGA video BIOS space from C0000 to CFFFF, since setting the REAL_MODE__APERTURE register to 070h sets the MBA address to submap 7, i.e., the Flash PROM space. Note that even when the VGA controller is enabled (GRP_CFG4[2]), the REAL_MODE__APERTURE[1:0] takes precedence over the standard VGA aperture.

VGA IMPLEMENTATION

VGA Compatibility

The Bt2166 is fully compatible with VGA applications. Upon configuring and enabling the Bt2166 VGA submodule, the chip implements standard VGA hardware functionality. (For PC platforms, Brooktree's video BIOS properly initializes the chip for VGA operation.) Once the chip is configured, applications may take full advantage of the VGA-defined hardware capability and switch VGA modes either through BIOS calls or by programming the VGA register set directly.

The Bt2166 also supports VESA BIOS extensions to standard VGA modes (Super VGA modes). For these modes, applications must switch modes through BIOS calls. This is reasonable (and standard practice) since the selection of Super VGA pixel clock frequencies is nonstandard in industry VGA implementations, among other issues. However, some VESA BIOS modes will allow the application to directly program those VGA registers that don't determine the basic monitor timing.

Within a register definition, the functional definition is implemented for all of the fields required for VGA compatibility. If the Bt2166 does not support a field's functionality that was present in previous VGA implementations, it is only because that particular functionality was required by the previous implementation, but not required by current implementations. Nevertheless, any software that was written to access that field may write and read the field as before. The written values are stored in the VGA register for later retrieval. The only difference is that the value in the register will have no effect on VGA operation. An example of such fields are the Asynchronous Reset and Synchronous Reset bits in the Sequencer Reset register.

VGA Operation

This specification does not intend to define the VGA architecture and functionality; application programmers should consult a book on the subject. However, the "I/O Mapped Register Definitions" chapter provides a list of VGA registers, including I/O addresses, read/write capability, and a list of the fields within that register. In the meantime, below are some notes about the Bt2166 implementation.

Simultaneous Operation with Other Bt2166 Modules

The BtVista hardware does not preclude the simultaneous operation of the VGA and GUI or VGA and video modules. However, such operations are usually not feasible because VGA applications typically do not include a driver-based interface to GUI or video functionality. The Bt2166 design assumes (but does not require) that GUI and video applications will run with VGA hardware disabled, and provides drivers accordingly. A GUI application could run using a standard VGA driver, but typically the GUI block would be disabled and only the VGA module enabled. Whether motion video applications work with a VGA graphics driver is a design choice in the coding of the Bt2166 low-level video driver.

A full-screen DOS window within Windows would also take advantage of the VGA hardware. However, when switching to a full-screen DOS window, the GUI hardware is typically disabled, so that again GUI and VGA hardware are not actually enabled concurrently.

VGA and DREF

The VGA definition originally assumed that the VGA hardware directly drove the monitor timing signals and directly sequenced pixel data to a palettized RAM-DAC. This was done through a VGA submodule called the CRT Controller. In fact, the Bt2166 VGA module does not directly drive monitor sync signals nor does it directly output data to the DAC. Instead, the Bt2166 implements a Display Refresh (DREF) module (refer to "Display Refresh Controller" on page 179), requiring the Bt2166 VGA module to implement some unique mechanisms to emulate the VGA CRT Controller functionality. The VGA module must automatically program the DREF monitor timing registers based on the VGA CRT register contents. It must also read in the VGA frame buffer and convert it into 8-bit pixel data which it stores in a DRAM buffer area for the DREF to later read.

The source VGA frame buffer is always stored in a 256 KB region starting at address zero of the DRAM media buffer, while the resultant 8-bit-per-pixel screen image is always output to the second 256 KB region of the DRAM media buffer starting at adresss 40000h.

The Bt2166 has dual paths to the DREF registers. One path goes directly from the PCI bus, and the other path goes indirectly through the VGA module. The VGA DREF writes are triggered by writes to a subset of VGA registers. Software can modify the DREF registers directly, but they might be overwritten the next time software writes to a VGA register.

There are three classes of VGA registers that result in DREF register writes. Each one operates somewhat differently: monitor resolution and timing, border color, and color palette. Modifying any one of a set of VGA monitor and timing registers will result in the VGA updating an entire set of DREF monitor and timing registers based on the VGA values. If the VGA Attribute Controller Overscan register is modified, the VGA will send that value to the DREF Border Color Index Register. If software writes a VGA color palette register through the 3c8h/3c9h indexing mechanism, then the VGA will send that write to the DREF color palette. These three classes of register writes have differences in their behavior. The writes to the monitor screen resolution and timing registers are only enabled if both the ENABLE_VGA_CRTC (GRP_CFG4[2]) and ENABLE_VGA_CRTC_GEN (GRP_CFG5[7]) bits are both on. No control bits are required to enable border color or palette writes. Another difference between the classes, is that VGA CRT and Attribute Controller Overscan registers are shadowed in the VGA, while the palette is not. VGA I/O space reads of the CRT and Overscan registers always come from the VGA shadowed copies, while VGA space reads of the palette come from the DREF.

When software writes to any VGA register that affects the screen resolution or timing, the VGA will write not just the corresponding DREF register, but the entire set. It launches into the "CRT programming sequence." This was done because the correlation between the VGA register set and the DREF register set does not have a clean one-to-one mapping. Also, the registers that it does write each time does not include every DREF control register. Table 2 shows the DREF registers that are programmed by VGA CRT programming sequence (in order of programming).

Table 2. DREF Registers Programmed by VGA CRT

DREF Register Name	DREF Register Address
Graphics Size	418h
Graphics Pitch/Burst Size	41Ch
Graphics Horizontal Timing	242h
Graphics Horizontal Active	50Ch
Miscellaneous Timing	520h
Graphics Vertical Timing	428h
Vertical Active	518h
Timing Generator Control	500h
Horizontal Sync	504h
Horizontal Blank	508h
Vertical Sync	510h
Vertical Blank	514h

Table 3 shows the VGA register writes that trigger the CRT DREF programming sequence. Sometimes the sequence is kicked off by a write to an address, regardless of whether any changes are made, and other times, the VGA must detect a change to a bit or field of a register before it kicks off this sequence. You might notice that some writes will unnecessarily trigger DREF CRT programming. For example, writing to the Cursor position register in the VGA CRT Controller space (3d4h-3d5h) in no way affects the DREF register values, yet this will kick off the writes. However, this was the easier algorithm and there is no performance penalty for unnecessary internal writes to the DREF.

Table 3. VGA Register Writes — Trigger CRT DREF Programming Sequence

Register Write Action	Addressing Details
Any CRT register write, modified or not	3d5h, all indices
Modifying the Video Screen disable bit in Sequencer Clocking Mode Register	3c5h, index 01, bit 5
Modifying 9-dot clock bit in Sequencer Clocking Mode Register	3c5h, index 01, bit 0
Modifying clock divide-by-2 bit in Sequencer Clocking Mode Register	3c5h, index 01, bit 3
Modifying horizontal or vertical sync polarity bits in the Miscellaneous Output Register	3c2h, bits 7 and 6
Modifying the graphics mode bit in the Attribute Controller Mode Control Register - this bit has a subtle affect on sync timing	3c0h, index 10, bit 0

The VGA has a dirty bit mechanism to keep track of modification of the CRT registers. It will repeat the DREF programming sequence if an additional VGA write happens before it completes the sequence. It will also keep track of writes that happen while the VGA enable bits are turned off. If VGA CRT writes happen while the CRT DREF programming is disabled, it will kick into the programming sequence as soon as the enable bits are turned back on. Table 4 shows other VGA writes that trigger DREF registers writes.

Table 4. Other VGA Writes that Trigger DREF Registers Writes

VGA Register	Addressing details	Modified DREF Register	Address
Attribute Controller Overscan Register	3c0h, index 11h writes	Border Color Index	404h
VGA Palette	3c9h writes	DREF color palette	000-3FCh

VGA and Pixel Clock PLL

The Bt2166 must provide a general mechanism for selecting the pixel clock frequency for a wide range of screen modes while still being compatible with the VGA standard method of selecting VGA pixel clock frequencies. The Bt2166 provides a programmable PLL to generate a wide variety of pixel clock rates (see "Pixel Clock PLL Rate Selection" on page 358). This register can be written through the protected mode PCI memory aperture. However, VGA software does not know about this register. The VGA way of determining the pixel rate is to write to bits 3 and 2 of the Miscellaneous Output register. It expects the VGA to automatically select between 25 and 28 MHz.

Therefore, when the VGA is enabled (ENABLE_VGA_CRTC is on), the Bt2166 VGA module bypasses the general-purpose pixel clock PLL register and provides its own PLL parameters. The VGA module will drive PLL M, N, and L values that correspond to either a 25 MHz or a 28 MHz clock rate. The actual PLL parameter values that the VGA uses for these clock frequencies are programmable through the GRP_25PLL and GRP_28PLL registers. See Table 15 on page 46.

An additional encoding of bits 3 and 2 of the Miscellaneous Output register was often used to select a nonstandard pixel clock whose frequency is controlled somewhere else. The Bt2166 supports this convention. When these bits are set to b10, the pixel clock PLL's parameters will come from the general-purpose pixel clock PLL register.

VGA Extension Register Control and Status

The Bt2166 provides some additional register fields which are not VGA standard registers but enable software to configure or monitor the VGA. These implementation-specific registers are located in the VGA Graphics Controller Extension space. In other words, they are accessed through the 3CEh, 3CFh indexing mechanism at register indices that lie above those defined for the VGA graphics controller. The chip provides a mechanism to lock or unlock access to these extended registers. However, not all registers in that space are related to VGA; hence not all are listed below.

For instructions on unlocking VGA extension registers, refer to "Unlock Register" on page 60. The VGA fields are summarized below.

Enable VGA CRT Controller Mechanism

The ENABLE_VGA_CRTC field in the GRP_CFG4 register (bit 2, index 44h) enables the VGA module to write pixel data to the frame buffer RAM and acts as master enable for modifying DREF registers. If disabled, no pixel data will be written and DREF registers will not be modified by the VGA hardware. If enabled, the ENABLE_VGA_CRTC_GEN bit (bit 7, index 45h) must also be enabled in order to allow modification of DREF registers. If ENABLE_VGA_CRTC_GEN is off, the VGA will still be allowed to write graphics data to the RAM, but will not disable access to VGA I/O registers or the VGA memory-mapped frame buffer.

Enable VGA CRT Controller DREF Programming

The ENABLE_VGA_CRTC_GEN field in the GRP_CFG5 register (bit 7, index 45h) enables the VGA module to program the DREF timing registers (see Table 25 on page 52). This bit is only effective if the ENABLE_VGA_CRTC bit (bit 2, index 44h) is set also. Setting ENABLE_VGA_CRTC but resetting ENABLE_VGA_CRTC_GEN to zero does have a purpose. It allows software to directly program the monitor timing and resolution while still using the VGA to generate pixel data. This has an application in VGA-compatible VESA BIOS modes. Note that this bit does not disable the generation of pixel data, translation of VGA palette access to DREF palette access, translation of VGA border color writes to VGA border color writes, nor the VGA's ability to override the Pixel Clock PLL register.

25 MHz PLL Select

Registers GRP_25PLL1 (index 1Bh) and GRP_25PLL0 (index 1Ah) specify the M, N, and L values that control the Pixel Clock PLL when the VGA Miscellaneous Output register is set for a 25 MHz pixel clock. Refer to "PLL25 Select Registers" on page 45.

28 MHz PLL Select

Registers GRP_28PLL1 (index 1Eh) and GRP_28PLL0 (index 1Dh) specify the M, N, and L values that control the Pixel Clock PLL when the VGA Miscellaneous Output register is set for a 28 MHz pixel clock. Refer to "PLL28 Select Registers" on page 46.

Attribute Index Status

Bit 0 of the GRP_VGASTAT register (index 2Ch) reflects the status of the Attribute register port (see "VGA Status Register" on page 48). If a zero, writing to 3C0h will write the Attribute Index register. If a one, writing to 3C0h will write the indexed Attribute Data register.

VGA CRT Generation Status

When bit 1 of the GRP_VGASTAT register (index 2Ch) is set to 1, the VGA is busy writing new values to the DREF timing registers. This bit can be used by software to manage mode switching, but it is not necessary for standard VGA operation. Note that the DREF register programming process is independent of the rest of the VGA. It does not hinder CPU access to VGA registers or frame buffer memory, nor does it inhibit the VGA from generating pixel data. This all happens concurrently. Refer to "VGA Status Register" on page 48.

VGA Read Latches

Registers GRP_RDLAT3 (index 2Bh) through GRP_RDLAT0 (index 28h) allow access to the 32-bit VGA Read Latch. The VGA Read Latch (defined by the VGA architecture) is an entity which is loaded with a 32-bit frame buffer value for each VGA memory read. Read Latch register [0] is the least significant byte of the read latch. See "Read Latch Registers" on page 48.

VGA DAC Mode

Bit 0 of the GRP_VGACFG register enables the VGA DAC 8-bit mode (see "VGA Configuration Register" on page 45). The default for the VGA DAC logic is the 6-bit mode. The 8-bit mode means that VGA DAC writes and reads to 3C9h are assumed to be 8-bit DAC values, and are therefore transferred between the DAC register block of the media buffer and the CPU bus unmodified. The 6-bit mode means that DAC VGA access to 3C9h are 6-bit DAC values, and therefore a translation is done between the 6-bit value on the CPU bus and the 8-bit value in the DAC register block of the media buffer. This is done for both reads and writes.

Currently, the DREF palette supports only 6-bits of resolution on all three colors; but these values are left-shifted to be aligned with their eight-bit equivalent. Put another way, the two least significant bits of the palette entries are un-implemented, but can still be considered as 8-bit formatted entries.

Configuration for VGA

The Bt2166 requires the initialization of just a few registers after chip reset. Brooktree provides drivers and BIOS routines to do this for common applications and environments. Below is an outline of the required steps.

1. Program the GRP_25PLL and GRP_28PLL registers with PLL values for 25 and 28 MHz operation. For 25 MHz operation, 241Ch (M,N,L = 28,4,2) is recommended. For 28 MHz operation, 283Fh (M,N,L = 63,8,2) is recommended.
2. Program the DREF Graphics Start Address A register with 40000h.
3. Turn on bit seven in the DREF configuration register (address 0), zero-out the other bits. This will enable the graphics window, select 8 bits-per-pixel mode, and select the border color index register as the source of the border color.
4. Program the standard VGA registers for the initial screen mode. This step is recommended, but not required. It is possible to do this after enabling the VGA, but the first screen won't come up as cleanly. It's also recommended that you leave the screen blanked at the end of this step (3c5, index 01h), and turn them back on after the enabling the VGA. This will produce the smoothest transition to the new screen.
5. Set the ENABLE_VGA_CRTC bit (GRP_CFG4, bit 2) and the ENABLE_VGA_CRTC_GEN bit (GRP_CFG5, bit 7).
6. Optionally, unblank the screen. See step 4.

Support for VESA BIOS Extension Modes

The VESA BIOS Extension (VBE) specification defines standard BIOS function calls which provide software access to resolutions, color depths, and frame buffer organizations beyond the VGA hardware standard. The Bt2166 supports these VESA BIOS extensions. These extensions have often been referred to as Super VGA (SVGA) modes.

VBE Modes

The Bt2166 hardware supports all VBE functions and common modes. The VESA modes that use VGA-compatible memory models allow the application to take advantage of VGA hardware. Those that use 256-color non-chain-4 and direct color memory models are supported, but without VGA hardware. In these modes the Bt2166 behaves as a dumb frame buffer controller supporting 8-bit color depths. VGA IO operations will affect neither the frame buffer accesses nor the display format. Status and control functions should be done through calls to the VBE functions, with the following exceptions: the DAC palette (3C6-3C9h), Attribute Overscan (3c0, index 11), and Input Status 1 (3DAh) still function upon disabling the VGA.

Frame Buffer Models

The VBE specification defines two ways to implement a frame buffer aperture: VGA window or linear/flat model. The set mode function includes an extra bit which specifies which model to use. The Bt2166 supports both.

For the VGA windowing frame buffer model, the Bt2166 provides dual real mode apertures via VGA address space. Each port may be configured for a 32 K or 64 K window size. The 32 K and 64 K mode aperture addresses are defined in "GRP_GUI_BASE Address Register," Table 16 on page 47. Each port includes read/write capability and a paging mechanism which allows an application to “window” into the frame buffer.

For the linear/flat buffer model, the Bt2166 chip provides a protected mode aperture which can be located anywhere within the CPU’s 32-bit address space. For more information, refer to “Real Mode Aperture” on page 25.

DREF register Programming and High-Resolution Modes

The VGA DREF register programming mechanism supports applications that change modes by writing directly to VGA CRT Controller Registers, rather than making a BIOS call. However, the VGA registers do not provide all of the flexibility that programming the DREF registers directly provides. For example, VGA always programs the DREF for 8 or 9 pixel timing. This limits the timing resolution that can be achieved. Also, there is no standard mechanism for switching to high-resolution modes independently of making a BIOS call. Therefore, it is wiser for BIOS or application drivers to program the DREF registers directly rather than rely on the VGA hardware to do it. This also means that the VGA DREF programming mechanism must be disabled by clearing the ENABLE_VGA_CRT_GEN bit (bit 7) in the GRP_CFG5 register (see Table 25 on page 52).

Mode Setting Procedure

The steps required to initialize a VBE mode depend on two factors:

- Does the mode use a VGA-compatible memory model? If not, the VGA hardware must be disabled through the ENABLE_VGA_CRT bit.
- Can the mode use VGA-generated DREF register values? If not, the ENABLE_VGA_CRT_GEN should be set to '0' to disable timing data generation. BIOS must program the DREF registers.

In general, the procedure is as follows:

1. Initialize DREF register set if not already initialized. Initialize the register fields which configure the VGA module.
2. If the mode does not use a VGA-compatible memory model (VBE models 0 through 4), then disable VGA CRT Controller functionality by clearing the ENABLE_VGA_CRT bit (bit 2 of GRP_CFG4). In this case, enable the real-mode or protected-mode aperture.
3. If the resolution exceeds standard VGA modes (such as 720 X 400 or 640 X 480), disable generation of the DREF register values. Program the DREF registers directly with the mode's timing parameters.
4. Modify the Pixel Clock PLL register. If the 25 MHz or 28 MHz pixel clock will be selected through the VGA Miscellaneous Output register, then the pixel clock field does not need to be modified. The VGA will generate the Pixel Clock PLL value automatically. Otherwise, write a 10b to bits 3 and 2 of the Miscellaneous Output register.

Logical Window Control

The VBE functions include logical window setup and control (such as setting the logical line width and adjusting the display start address). These functions can be used for horizontal panning, vertical scrolling, and display buffer switching for animation.

The Bt2166 chip provides the capability to support these functions in hardware. The easiest way to accomplish all of these functions is to change the graphics start address pointer in the DREF by writing to the active graphics start register, the default register being Graphics Start Address A. Alternatively, any one of the DREF's three start address registers (A,B, and C) can be selected manually or automatically between frames. However, manual selection provides little advantage over simply writing a new value to the default active register.

Horizontal panning is a little more difficult if you want more resolution than a quadword's worth of pixels. It is possible though. The DREF architecture differentiates between an "active region" in the screen and the "graphics window". They are usually the same, but they don't have to be. By having the graphics window being larger than the active region, you can simulate panning by moving the graphics window underneath the active region, one pixel width at a time.

DAC Palette Format

The VBE includes a function to set/get the number of bits per palette primary color. For palette access via the VGA 3C9h port, the Bt2166 controller supports both 6-bit and 8-bit formats, which are the most common formats. The selection is made through bit 0 of the GRP_VGACFG register. A '1' indicates an 8-bit mode, while a '0' indicates a 6-bit mode (refer to "VGA Configuration Register" on page 45).

This bit only affects accesses via the VGA 3C9h port. When configured for 6-bit modes, palette operations via 3C9h are translated between 6-bit and 8-bit formats. On the other hand, any palette values programmed directly through the DREF palette registers are always in 8-bit format. However, BIOS can still perform the translation if the palette is accessed through the Load/Unload Palette Data function call.

I/O MAPPED REGISTER DEFINITIONS

Introduction

This section defines the registers that are available through the I/O space interface. I/O space registers include the I/O addresses used by VGA controllers.

Table 5 provides a complete list of the I/O port addresses that are implemented within the Bt2166 and their corresponding alternate decode addresses (see ‘‘Programmable Multimedia Interface’’ on page 175). Notice that all I/O port addresses are present for software compatibility with VGA. Since these I/O spaces were required for software compatibility, they were extended to control some Bt2166-unique functions. In a PCI environment, access to these I/O addresses are requested by declaring through the PCI config register space to have a VGA compatible display controller function. For information on the PCI interface refer to “CPU Host Bus Interface” chapter on page 325.

Table 5. Bt2166 I/O Address Map *(1 of 2)*

I/O Port	Alternate Decoder Port	Write	Read
03B4	33C4	VGA CRT index (mono)	CRT index (mono)
03B5	33C5	VGA CRT data (mono)	CRT data (mono)
03BA	33CA	VGA feature control (mono)	Input status reg 1 (mono)
03C0	23C0	VGA attribute index/data	Attribute index
03C1	23C1	Cycle accepted from bus but no operation performed.	Attribute data
03C2	23C2	VGA miscellaneous output	Input status reg 0
03C4	23C4	VGA Sequencer index	VGA Sequencer index
03C5	23C5	VGA Sequencer data	VGA Sequencer data
03C6	23C6	VGA DAC pixel mask	VGA DAC pixel mask
03C7	23C7	VGA DAC address read mode	VGA DAC state
03C8	23C8	VGA Pixel address write mode	VGA Pixel address write mode
03C9	23C9	VGA Pixel data	VGA Pixel data

Table 5. Bt2166 I/O Address Map *(2 of 2)*

I/O Port	Alternate Decoder Port	Write	Read
03CA	23CA	Cycle accepted from bus but no operation performed	VGA Feature control read
03CC	23CC	Cycle accepted from bus but no operation performed	VGA Miscellaneous output read
03CE	23CE	VGA Graphics controller index	VGA Graphics controller index
03CF	23CF	VGA Graphics controller data	VGA Graphics controller data
03D4	23D4	VGA CRT index (color)	VGA CRT index (color)
03D5	23D5	VGA CRT data (color)	VGA CRT data (color)
03DA	23DA	VGA Feature control (color)	VGA Input status reg 1 (color)

The VGA emulation supports six distinct sets of I/O space addresses: VGA Graphics registers, VGA Sequencer registers, VGA CRTC registers, VGA Attribute registers, VGA Color registers, and VGA External registers.

Similarly, VGA Sequencer registers are accessed by loading the index of the desired register into the Sequencer Index Register at I/O port address 3C4h followed by reading or writing the desired data from the register through the Sequencer Data Register at I/O port address 3C5h. See ‘‘VGA Sequencer/Extension Registers’’ on page 56 for more detail.

The VGA CRT Controller registers are accessed by a similar mechanism with the CRTC Index registers at either I/O port address 3B4h or 3D4h and the CRTC Data Register at either I/O port address 3B5h or 3D5h. See ‘‘VGA CRT Controller’’ on page 62 for more detail.

The VGA Attribute registers are also indexed, but the mechanism has the standard VGA characteristic that both the index and data values pass through the same I/O port address at 3C0h. Reading 3C0h returns the contents of the Indexed Attribute Register. Reading 3C1h returns the contents of the Attribute Index Register. See ‘‘VGA Attribute Controller’’ on page 71 for more detail.

The VGA External registers are accessed at their specified I/O port addresses without further Indexing. See ‘‘VGA General/External Registers’’ on page 76 for more detail.

The VGA Color registers have their own, standard index mechanism involving I/O port addresses 3C6h, 3C7h, 3C8h, and 3C9h. See ‘‘VGA Color Registers’’ on page 79 for more detail.

VGA Graphics Registers

The graphics registers are accessed by loading the index of the desired register into the Graphics Index Register at I/O port address 3CEh followed by reading or writing the desired data from the register through the Graphics Data Register at I/O port address 3CFh. The first nine registers are standard VGA registers and the remainder are Bt2166 extensions.

Graphics Index Port

name: GRP_INDEX
port: 3CEh, read/write
size: 8 bits
function: The value in this register determines which register in the graphics controller block is written or read when performing an I/O cycle to the GRP_DATA address. Only the lower 4 bits of the Index Register are valid when the Bt2166 extensions are locked. When the Bt2166 extensions are unlocked, all 8 bits of the Index Register are used. When locked, bits 7:4 are masked-off (not zeroed); their values remain the same as before being locked.

Graphics Data Port

name: GRP_DATA
port: 3CFh, read/write
size: 8 bits
function: The various VGA graphics registers are accessed through this I/O whenever their corresponding index has been loaded into GRP_INDEX.

Set/Reset Register

name: GRP_SR
index: 3CFh.00h, read/write
size: 8 bits
function: This register is loaded with the pattern to be written to the display planes in Write Mode 0 or Write Mode 3. There is a 1-to-1 correspondence between the bits in this register and the display planes. The bit contents of the Set/Reset Register are shown in Table 6.

Table 6. Set/Reset Register

Bit	Function
7:4	Reserved
3	Set/Reset Plane 3
2	Set/Reset Plane 2
1	Set/Reset Plane 1
0	Set/Reset Plane 0

Enable Set/Reset Register

name: GRP_ESR
index: 3CFh.01h, read/write
size: 8 bits
function: This register enables or disables the individual planes affected by the Set/Reset Register. This register is only used in Write Mode 0. The bit contents of the Enable Set/Reset Register are shown in Table 7.

Table 7. Enable Set/Reset Register

Bit	Function
7:4	Reserved. Hardwired to 0h
3	Enable Set/Reset Plane 3
2	Enable Set/Reset Plane 2
1	Enable Set/Reset Plane 1
0	Enable Set/Reset Plane 0

Color Compare Register

name: GRP_CC
index: 3CFh.02h, read/write
size: 8 bits
function: The color compare bits contain the value to which all 8 bits of the corresponding memory plane are compared. The Color Don't Care Register establishes the mask for the memory planes. This register is only used in Read Mode 1. A '1' is returned for the plane positions where the bits of all 4 planes equal the Color Compare Register. The bit contents of the Color Compare Register are shown in Table 8.

Table 8. Color Compare Register

Bit	Function
7:4	Reserved. Hardwired to 0h
3	Color Compare Plane 3
2	Color Compare Plane 2
1	Color Compare Plane 1
0	Color Compare Plane 0

Data Rotate Register

name: GRP_ROT
index: 3CFh.03h, read/write
size: 8 bits
function: This register contains the Rotate Count and the Function Select fields. The Rotate Count field specifies the number of bit positions to rotate data right by the CPU in Write Mode 0 or 3. The Function Select field selects a Boolean between the write data and the data contained in the Read Latches. The logical functions are enabled for Write Mode 0, 2, and 3. The bit contents of the Data Rotate Register are shown in Table 9.

Table 9. Data Rotate Register

Bit	Function	Detail
7:5	Reserved	Hardwired to b'000
4:3	Function Select 1:0	00 = data unmodified 01 = data ANDed with data in latches 10 = data ORed with data in latches 11 = data XORed with data in latches
2:0	Rotate Count 2:0	Amount to rotate right CPU data

Read Map Select Register

name: GRP_RDPLN
index: 3CFh.04h, read/write
size: 8 bits
function: This register selects which planes are read in Read Mode 0. The bit contents of the Read Map Select Register are shown in Table 10.

Table 10. Read Map Select Reggister

Bit	Function	Detail
7:2	Reserved	Hardwired to 00h
1:0	Map Select 1:0	00 = Plane 0 01 = Plane 1 10 = Plane 2 11 = Plane 3

Mode Register

name: GRP_MODE
index: 3CFh.05h, read/write
size: 8 bits
function: This register controls the VGA Read and Write modes, data shift registers, and other addressing modes. The bit contents of the Mode Register are shown in Table 11.

Table 11. Mode Register

Bit	Function	Detail
7	Reserved	Hardwired to 0
6	256 Color Mode	0 = Configure shift registers for 2, 4, or 16 colors 1 = Configure shift registers for 256 colors
5	Shift Register	0 = Configure shift registers for EGA/VGA compatibility 1 = Configure shift registers for CGA compatibility
4	Odd/Even	0 = Normal frame buffer read addressing 1 = Even/odd addressing - use CPU address bit 0 to select odd or even planes on reads
3	Read Mode	0 = Read Mode 0 - Read data from planes selected by Read Map Select Register (has no effect if bit 3 of the Sequencer Memory Mode Register = 1) 1 = Read Mode 1 - Read comparison of the planes and Color Compare Register
2	Reserved	Hardwired to 0
1:0	Write Mode 1:0	The Write Mode field controls the configuration of the datapath for VGA writes. It controls how the Set/Reset, Enable Set/Rest, Logic Unit, Data Rotator, and Bit Mask Logic are interconnected. Consult a referenced VGA publication for more detailed information. 00 = Write Mode 0 01 = Write Mode 1 10 = Write Mode 2 11 = Write Mode 3.

Graphics Miscellaneous Register

name: GRP_MISC
index: 3CFh.06h, read/write
size: 8 bits
function: This register controls the VGA display mode, monochrome graphics emulation, and odd/even addressing. The bit contents for this register are shown in Table 12.

Table 12. Graphics Controller: Miscellaneous Register

Bit	Function	Detail
7:4	Reserved	Hardwired to 0h
3:2	Memory Map 1:0	00 = 128 KB A0000–BFFFFh 01 = 64 KB A0000–AFFFFh 10 = 32 KB B0000–B7FFFh 11 = 32 KB B8000–BFFFFh
1	Chain Odd/Even	This bit controls whether CPU address A0 is used for memory address bit 0, or replaced with another value (chaining). Consult a VGA text for more detail. 0 = Normal addressing 1 = Chain Odd/Even function is enabled
0	Graphics Mode	0 = Text mode 1 = Graphics mode

Color Don't Care Register

name: GRP_CCX
index: 3CFh.07h, read/write
size: 8 bits
function: This register selects which planes will be used in a color comparison between the values in the Graphics Controller Color Compare Register and the data read from video memory. This comparison is only done while in Read Mode 1. If the map bit = 1, color comparison is enabled for that plane or map. The bit contents of the Color Don't Care Register are shown in Table 13.

Table 13. Color Don't Care Register

Bit	Function
7:4	Reserved - hardwired to 0h
3	Memory Map 3
2	Memory Map 2
1	Memory Map 1
0	Memory Map 0

Bit Mask Register

name: GRP_BITMASK
index: 3CFh.08h, read/write
size: 8 bits
function: This register is a mask for modifying displayed pixels. A bit set to a 1 allows the corresponding pixel to be changed. The LSB corresponds to the rightmost pixel in the group. This register is disabled with Write Mode 1. The bit contents of the Bit Mask Register are shown in Table 14.

Table 14. Bit Mask Register

Bit	Function
7:0	Bit Mask

Extension Registers Accessed Via VGA Space

This section describes general purpose I/O registers accessed through the VGA space. The extension registers are accessed by loading the index of the desired register into the GRP_INDEX register (see page 39) at I/O port address 3CEh followed by reading or writing the desired data from the register through the GRP_DATA register (see page 39) at I/O port address 3CFh. Before being accessed, these registers must be unlocked with the SEQ_UNLOCK register (see page 60).

VGA Configuration Register

name: GRP_VGACFG
index: 3CFh.16h, read/write
size: 8 bits
function: Writing 01h to this register enables the VGA DAC 8-bit mode. The default for the VGA DAC logic is the 6-bit mode. The 8-bit mode means that DAC VGA access to 3C9h are 8-bit DAC values, and therefore transferred to the DREF palette RAM unmodified. The 6-bit mode means that the DAC VGA writes to 3C9h are 6-bit DAC values, and therefore are shifted left 2 bits before being written to the DREF palette RAM. DAC values read from the DREF palette RAM through 3C9h are shifted right 2 bits to convert to a 6-bit value.

Chip ID Registers

name: GRP_CID[1:0]
index: 3CFh.18h:17h, read only
size: 16 bits
function: These registers contain the 16-bit Bt2166 chip ID value. If the chip design were released in September 1997, the chip ID would be 9709h.

PLL25 Select Registers

name: GRP_25PLL[1:0]
index: 3CFh.1Bh:1Ah, read/write
size: 16 bits
function: This register contains the PLL register value that is routed to the pixel clock PLL when the VGA Miscellaneous Output Register (see Table 59 on page 77) is programmed to select a 25 MHz pixel clock for the VGA. See also “Pixel Clock PLL Rate Selection” on page 358.

The bit contents for this register are shown in Table 15.

Table 15. PLL 25 MHz and 28 MHz Select Registers

Bit	Function
23:12	Reserved
13:12	Post scalar factor (variable L) 00 = Divide by 1 01 = Divide by 2 10 = Divide by 4 11 = Reserved
11:8	Value of N, range: 1 to 15 (0h to Fh)
6:0	Value of M, range: 1 to 128 (0h to 7Fh)

PLL28 Select Registers

name: GRP_28PLL[1:0]
index: 3CFh.1Eh:1Dh, read/write
size: 16 bits
function: This register contains the PLL register value that is routed to the pixel clock PLL when the VGA Miscellaneous Output Register (see Table 59 on page 77) is programmed to select a 28 MHz pixel clock for the VGA. See also “Pixel Clock PLL Rate Selection” on page 358.

The bit contents for this register are shown in Table 15.

GUI Base Address Registers

name: GRP_GUI_BASE[3:0]
index: 3CFh.23h:20h, read/write
size: 32 bits
function: These registers provide two function relating to the PCI memory aperture for the VGA-compatible display controller function. First, it shows where the protected-mode memory aperture is mapped by providing a read-only copy of the PCI base register. Second, it provides a mechanism to map real-mode VGA memory space into pages of the 32M internal chip space.

The GUI Base Address Register is a 32-bit register with the least significant byte at index 20h and the most significant byte at 23h so that:

- index 20h has GUI base [07:00]
- index 21h has GUI base [15:08]
- index 22h has GUI base [23:16]
- index 23h has GUI base [31:24]

Table 16 defines the bit contents of the GUI Base Address Register for both 32 K (bits 21:20 = 2’b11) and 64 K (bit 21 = ’b0, 18 = ’b1) apertures. Upon reset, all bits are cleared.

Table 16. GRP_GUI_BASE Address Register

Bits in 64 K apert.	Bits in 32 K apert.	Field	Description
31:25	31:25	PM_BASE	These 7 bits are used to match against the upper 7 bits of all PCI memory addresses. The protected mode aperture is accessed when such a match occurs. In addition, no match will occur unless at least one of the PM_BASE bits is set to one. This insures that the protected mode aperture can not overlay DOS memory from 00000000h to 000FFFFFh (blocking access to the lower 32 MB of the CPU memory address space). Thus software cannot inadvertently cause a collision between system memory and the protected mode aperture. These 7 bits are the same one found in PCI_BASE0[31:25]; see Table 182 on page 321. The protected mode aperture is disabled when this register is set to 0.
24	24	PM_ENABLE	This bit must be set to one to enable the protected mode aperture. This bit controls whether the entire 32 MB aperture is accessible.
23:22	23:22	Reserved	
21	21	32K_APERTURES[(1)]	1= 32 K real mode aperture 0= 64 K real mode aperture
__	20	ENABLE_32K_APERTURES[(1)]	If bit 21 =1 and bit 20 = 1: enable 32 K real mode apertures
20:19	___	Unused	If bit 21 = 0 then these are unused
18	___	ENABLE_64K_REAL[(1)]	If bit 21 = 0 and bit 18 = 1: enable 64 K real mode apertures
17:9	19:10	REAL_MODE_APERTURE1[(1)]	These bits are appended as the high order address bits of a 25-bit address which is applied to the HBUS as if it had come from a protected mode reference to GUI_BASE. The lower 16 bits for 64 K mode or 15 bits for 32 K mode come from the memory address of a read or write to a location determined as follows. 64 K mode: 000B0000h–000BFFFFh 32 K mode: 000A8000h–000AFFFFh
8:0	9:0	REAL_MODE_APERTURE0[(1)]	These bits are appended as the high order address bits of a 25-bit address which is applied to the HBUS as if it had come from a protected mode reference to GUI_BASE. The lower 16 bits for 64 K mode or 15 bits for 32 K mode come from the memory address of a read or write to location determined as follows. 64 K mode: 000A0000h -000AFFFFh 32 K mode: 000A0000h - 000A7FFFh

a. To set 32 K mode, bits 20 and 21 must be 1.. To set 64 K mode, bit 18 must be 1 and bit 21 must be 0. Refer to “Real Mode Aperture” on page 25 for more information.

Read Latch Registers

name: GRP_RDLAT[3:0]
index: 3CFh.2Bh:28h, read only
size: 32 bits
function: These four registers allow access to the 32-bit VGA Read Latch. The VGA Read Latch is an entity defined by VGA criteria which is loaded with a 32-bit frame buffer value for each VGA memory read. Read Latch Register [0] is the least significant byte of the read latch.

VGA Status Register

name: GRP_VGASTAT
index: 3CFh.2Ch, read/write
size: 8 bits
function: Table 17 defines the contents of the VGA Status Register.

Table 17. VGA Status Register

Bit	Function
7:4	Reserved
3	Contains status of VGA display buffer for line repaint. 1 = CRT is currently printing a line of data 0 = CRT is not printing a line of data
2	Contains status of VGA display buffer for the frame repaint. 1 = CRT has completed the frame repaint and is waiting for the Display Refresh Controller to send the frame resync signal to start the next repaint operation. 0 = CRT is still processing a repaint request from the Display Refresh Controller
1	Reports CRT generator status 1 = CRT controller is busy updating the DREF timing registers 0 = CRT controller is finished updating the DREF timing registers
0	Indicates the direction of a write to 3C0h as either the Attribute Index or Attribute Data Register 1 = Attribute Index Register 0 = Attribute Data Register

I/O Aperture Shadow Register

name: GRP_IOAP
index: 3CEh.31h:30h, read-only
size: 16 bits
function: This register shadows the lower 16 bits of the PCI configuration space PCI_BASE1 register, Function 0. This register allows I/O port access to the Bt2166 memory. Reset value is 00h.

This IO aperture provides real-mode software (mainly BIOS) an alternate mechanism to access various parts of the Bt2166's internal address space, including an alias for the 3CE/3CF addresses (IO space) and a mechanism to index into the internal memory map. The upper 16 bits of this register are not available because X86 architectures don't have IO space beyond 16-bits; and, other CPU architectures generally don't have IO space. Refer "PCI Base Address Register 1" on page 322 for details.

GRP_PIO_ENABLE Register

name: GRP_PIO_ENABLE
index: 3CFh.C0h, read/write
size: 8 bits
function: The GRP_PIO_ENABLE register controls the read/write direction of the IICSEL (PIO0), MPEGOE__GOGGLE (PIO1), VHS_VIPHAD0, VVS_VIPHAD1, and VACTIVE_VIPIRQ pins. Setting the GOGGLE_ENABLE bit (GRP_CFGA[4], Table 20 on page 51) will override the GRP_PIO_ENABLE[1] bit. Similarly, VID_GNC[8:6] (see Table 138 on page 176) memory-mapped video register will override GRP_PIO_ENABLE[4:2]. Reset value is 00h. See Table 18.

NOTE

Typically, PIO0 is configured as an output and is connected to the IICSEL pin; and PIO1 is configured as an output and is connected to the MPEGOE__GOGGLE pin.

Table 18. GRP_PIO_ENABLE Register

Bits	Function
7:5	Unused
4	VID_GNC[8] must be off (not over-riding). 1 = Output GRP_PIO_ DATA[4] via VHS_VIPHAD0 0 = Input GRP_PIO_ DATA[4] via VHS_VIPHAD0
3	VID_GNC[7] must be off (not over-riding). 1 = Output GRP_PIO_ DATA[3] via VVS_VIPHAD1 0 = Input GRP_PIO_ DATA[3] via VVS_VIPHAD1
2	VID_GNC[6] must be off (not over-riding). 1 = Output GRP_PIO_ DATA[2] via VACTIVE_VIPIRQ 0 = Input GRP_PIO_ DATA[2] via VACTIVE_VIPIRQ
1	VID_GNC[4] must be off (not over-riding). 1 = Output GRP_PIO_ DATA[1] via MPEGOE__GOGGLE (PIO1) 0 = Input GRP_PIO_ DATA[1] via MPEGOE__GOGGLE (PIO1)
0	1 = Output GRP_PIO_ DATA[0] via IICSEL (PIO0) 0 = Input GRP_PIO_ DATA[0] via IICSEL (PIO0)

PIO Data Register

name: GRP_PIO_DATA
index: 3CFh.C1h, read/write
size: 8 bits
function: Data written to this register is output when the corresponding GRP_PIO_ENABLE[4:0] bits (see ‘‘GRP_PIO_ENABLE Register’’ on page 49) are set to 1. The data is output on the following pins: IICSEL (PIO0), MPEGOE_GOGGLE (PIO1), VHS_VIPHAD0, VVS_VIPHAD1, and VACTIVE_VIPIRQ.

Data read from these bits always reflects the state of PIO pins. The write and read values are equal when the corresponding GRP_PIO_ENABLE bit is set to 1. When GRP_PIO_ENABLE bit is 0, the read bit still reflects the value being received on the pin, and the write data will set a flop, but this flop value will not show up on the pin until the GRP_PIO_ENABLE bit is set to 1. Except for bit 0, these pins can be used to output values which come from other sources. In particular, the GOGGLE_ENABLE bit (GRP_CFGA[4], Table 20 on page 51) will cause the MPEGOE_GOGGLE pin to be driven with the goggle output. Also, VID_GNC[8:6] memory-mapped video register will cause the the VHS_VIPHAD0, VVS_VIPHAD1, and VACTIVE_VIPIRQ pins to be driven with video control signals instead of the PIO data bits.

Reset value is 00h. See Table 19 for contents of GRP_PIO_DATA register.

Table 19. GRP_PIO_DATA Register

Bit	Function
7:5	Unused
4	1 = Read/write data via pin VHS_VIPHAD0
3	1 = Read/write data via pin VVS_VIPHAD1
2	1 = Read/write data via pin VACTIVE_VIPIRQ
1	1 = Read/write data via pin MPEGOE__GOGGLE (PIO1)
0	1 = Read/write data via pin IICSEL (PIO0)

Configuration Registers

name: GRP_CFG[A:0]
index: 3CFh.4Ah:40h, read/write
size: 88 bits
function: These 11 registers allow operating parameters to be set at power-on for the Bt2166 and modified later for desired chip operation through video BIOS. The configuration registers write values 40h–4Ah to the Index/Address Register (03CEh) and read or write data at the Data Register location (03CFh). Address 40h corresponds to GRP_CFG0, address 41h corresponds to GRP_CFG1, etc. Table 20 through Table 30 define the contents of the configuration registers. Figure 11 on page 55 provides a graphical overview of the configuration registers strapping to the VDATA bus.

Table 20. Configuration Register A (GRP_CFGA)

Bit	Function
7	GOGGLE_STATUS—Read only. When set, reports the state of MPEGOE_GOGGLE pin. Reset value 0.
6	GOGGLE_INVERT—When set, this bit inverts the goggle output on the next active vsync edge (specified by GOGGLE_EDGE). Output will remain inverted for one full vsync period or until reset. Inversion of the goggle output is synchronized to the vsync edge to prevent glitches. Since setting this bit doesn't take effect until the next vsync edge, output will remain static for two consecutive vsync periods before resuming the toggle. Note that the same behavior happens when setting this bit back to zero. Reset value 0.
5	GOGGLE_EDGE—Specifies the vsync edge on which to toggle the MPEGOE_GOGGLE output. Reset value 0. 0 = Falling vsync edge 1 = Rising vsync edge
4	GOGGLE_ENABLE—Enables goggle output to drive MPEGOE_GOGGLE pin and starts internal toggle machine. Reset value 0. 0 = Reset goggle state machine, disable output pin 1 = Start toggling the MPEGOE_GOGGLE output pin on the next active vsync edge
3:0	Reserved

Table 21. Configuration Register 9 (GRP_CFG9)

Bit	Function
7:0	Reserved

Table 22. Configuration Register 8 (GRP_CFG8)

Bit	Function
7:0	Reserved

Table 23. Configuration Register 7 (GRP_CFG7)

Bit	Function
7	66 MHz enable — Enables 66 MHz AGP bus interface. When set, the Bt2166 is configured for AGP 66 MHz operation. When zero, the Bt2166 is configured for PCI 33 MHz operation. Since this bit is initialized according to resistor strapping on a VDATA pin, the Bt2166 can be configured for 33MHz or 66 Mhz depending on the board. Reset value: VDATA[7]. 1 = enable AGP bus interface 0 = enable PCI bus interface
6	FCN1_DISABLE — Disables PCI function 1 configuration space. When set, the PCI Function 1 (video) will disappear from the PCI Config address space and the Bt2166 will look like a single-function device. Since this bit is initialized according to resistor strapping on a VDATA pin, the video function can appear in the PCI space when supporting video capture hardware is present in the system. Reset value: VDATA[6] 1 = disable PCI function 1 (video capture disabled) 0 = enable PCI function 1 (video capture enabled)
5:0	DRAM _TYPE— Indicates the state of the VDATA[5:0] pins during reset. Informs software of the type of DRAM present on the board. Reset value: VDATA[5:0]. (See also: “External Configuration/Memory Configuration Resistors” on page 334.)

Table 24. Configuration Register 6 (GRP_CFG6)

Bit	Function
7:1	Reserved
0	Reserved - write to zero

Table 25. Configuration Register 5 (GRP_CFG5)

Bit	Function
7	ENABLE_VGA_CRTC_GEN—Enables VGA CRTC Generator 1 = CRTC structure generation is enabled so that certain VGA I/O cycles will modify Display Refresh Controller registers (refer to “VGA and DREF” on page 28). This bit does not affect the generation of graphics data nor affect palette data writes. GRP_CFG4[2] (Table 26 on page 53) must be set to 1 for ENABLE_VGA_CRTC_GEN to have an effect. 0 = DREF timing registers will not be updated in response to VGA I/O cycles (default).
6:5	DIAG_OUT_SEL—Enables diagnostic state on the VDATA pins. For diagnostics only: must be set to 00.
4:0	Reserved

Table 26. Configuration Register 4 (GRP_CFG4)

Bit	Function
7	Reserved
6	ROM_WRT_ENABLE—ROM Write Enable. 1 = Enable Flash ROM write mode 0 = Disable Flash ROM write cycles (default)
5:3	Reserved
2	ENABLE_VGA_CRTC—Enables VGA CRTC block. The VGA CRTC block maps the VGA frame buffer into the display buffer, according to the current VGA register settings. Setting this bit will cause the VGA CRT controller to begin reading the VGA frame buffer image and output pixels. This bit must be set to enable generation of DREF monitor timing and resolution registers. The ENABLE_VGA_CRTC_GEN bit (GRP_CFG5, bit 7. Table 25 on page 52) must also be set. 1 = Enable the VGA CRT controller. 0 = Disable the VGA CRT controller, force CRTC into idle state (default).
1	Reserved
0	ENABLE_GUI—“GUI portions” are defined as anything having to do with the function or access of the GUI DRAM queue, and anything having to do with reading or writing queued or non-queued registers, including registers that control bit flipping in the Flippin Map. 1 = Enable GUI and GUI portions of FBA logic 0 = Reset all of GUI plus GUI portions of FBA logic (default)

Table 27. Configuration Register 3 (GRP_CFG3)

Bit	Function
7	BIOS_RESET_STATE—Register contains a read/write flop which does not affect chip operation. Allows BIOS to keep track of whether the Init sequence has been executed. 1 = BIOS Init Sequence has been executed 0 = BIOS Init Sequence not executed (default)
6	DECODE_TEST —Enable alternate decode address space so that the VGA real mode aperture A0000–BFFFF becomes 860A0000–860BFFFFh, the VGA ROM aperture at C0000–C7FFF becomes 860C0000h–860C7FFFh. The VGA I/O’s at 03Cxh become 23Cxh. Both mono and color VGA registers at 03Bxh and 03Dxh become 33Cxh. 1 = Enable alternate decodes 0 = Enable normal decodes (default)
5	FUNC1_DEVID_0 — This bit determines bit 0 of the PCI Function 1 Device ID, see “PCI Device ID Registers” on page 317. Reset value = 1.
4:0	Reserved

Table 28. Configuration Register 2 (GRP_CFG2)

Bit	Function
7:0	REFRESH_RATE—This value is used as a down count to determine when to issue refresh cycles to DRAM. Units are MEMCLK/8. Time between successive refresh cycles is ($Period_{MEMCLK}$ • 8) • GRP_CFG2). One row in each of four 32-bit DRAM banks is refreshed per refresh cycle. Reset value: A5h -- should be good for all memories.

Table 29. Configuration Register 1 (GRP_CFG1)

Bit	Function
7:0	Reserved

Table 30. Configuration Register 0 (GRP_CFG0)

Bit	Function
7:0	Reserved

Register Initialization

At reset all bits contained in the configuration registers are initialized. Those bits with reset values definable by the user are initialized based on logic levels present on VDATA pins. Pins can be programmed by strapping the pin to either VDD or VSS; a high value resistor (20 KΩ recommended) should be used for strappings.

Devices attached to the VDATA bus should have their drivers tri-stated at reset to allow programming of the configuration registers. The VDATA pins have weak internal pull-up resistors; consequently if a pin is not pulled down externally, a logical 1 is presented to the configuration register at reset time. If a pin is pulled down with an external 20k resistor, a logical 0 is loaded into the Configuration Register.

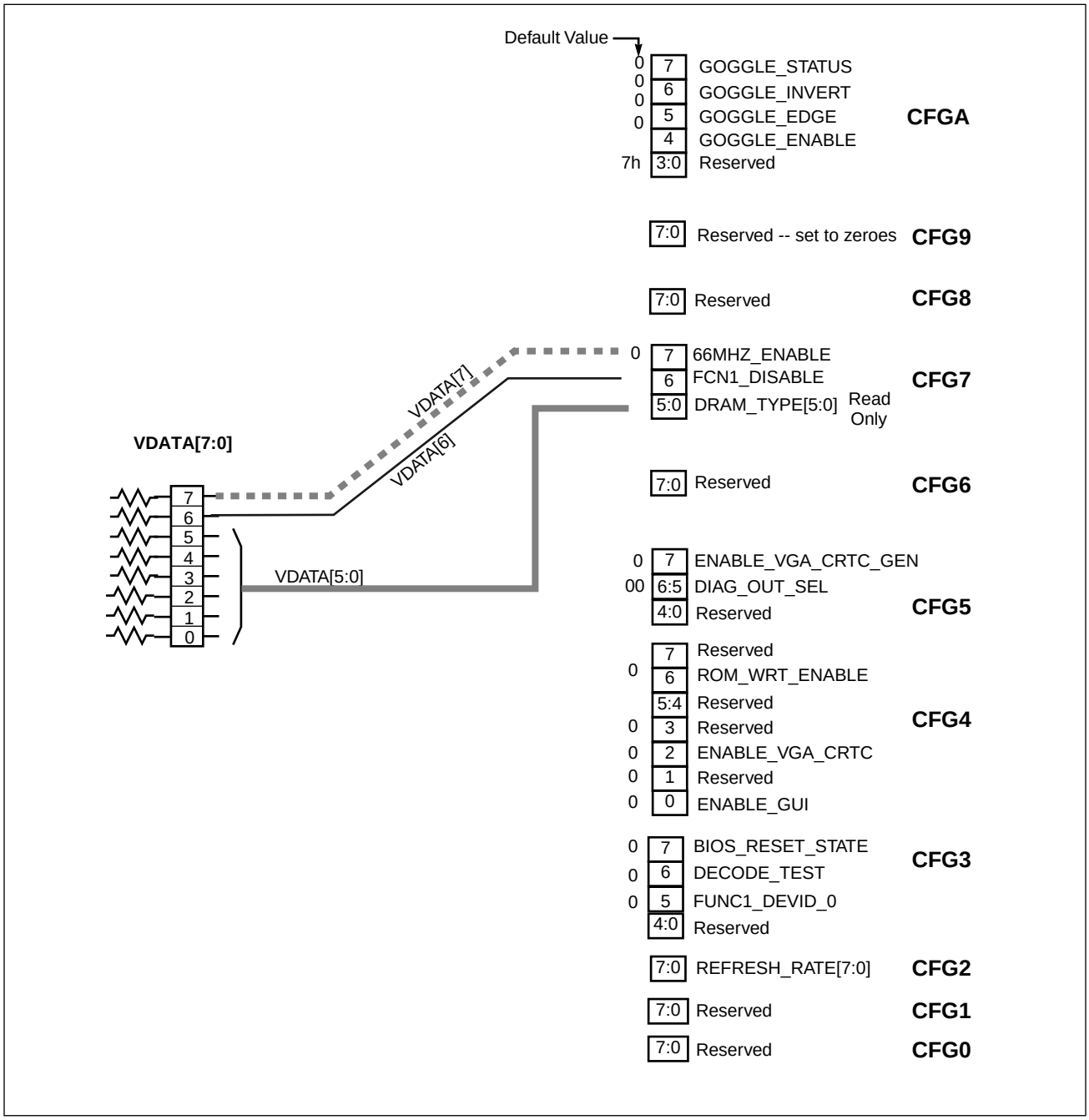
IMPORTANT

To allow programming of the configuration registers, devices attached to the VDATA bus should have their drivers tri-stated at reset.

Figure 11 shows an overview of the pins of the VDATA bus which affect the initialization state of the configuration registers. The values applied to VDATA[5:0] appear in GRP_CFG7[5:0] and contain read-only, DRAM-specific parameters which are used by the VGA BIOS to initialize the card at reset.

For additional SGRAM configuration information, refer to “External Configuration/Memory Configuration Resistors” on page 334)

Figure 11. Configuration Registers (GRP_CFG[A:0]) and Strapping Bits for Bt2166



VGA Sequencer/Extension Registers

The sequencer uses indexed addressing through ports 3C4h/3C5h to access seven different control registers. The first six registers are standard VGA registers. The seventh register is a Bt2166 extension used to unlock the VGA extension registers indexed through 3CEh/3CFh.

Sequencer Index Register

name: SEQ_INDEX
port: 3C4h, read/write
size: 8 bits
function: This register specifies the register in the sequencer block to be accessed by the next I/O read or write to the SEQ_DATA address. Only bits 2:0 are writable. Bits 7:3 are hardwired to zero.

Sequencer Data Port

name: SEQ_DATA
port: 3C5h, read/write
size: 8 bits
function: This port accesses the sequencer data registers. The register accessed depends on the value of SEQ_INDEX.

Reset Register

name: SEQ_RST
index: 3C5h.00h, read/write
size: 8 bits
function: This register has no effect on the behavior of Bt2166. A 00h should be written to this register during mode set; bits 7:2 are hardwired to zero.

Clocking Mode Register

name: SEQ_CLK
index: 3C5h.01h, read/write
size: 8 bits
function: This register contains the fields necessary to establish the video clock timing and control display refresh. The bit contents of the Clocking Mode Register are shown in Table 31. The options for the shift register control are shown in Table 32.

Table 31. Clocking Mode Register *(1 of 2)*

Bit	Function	Detail
7:6	Reserved	Hardwired to 00
5	Display refresh	0 = Normal operation 1 = Display refresh stops
4	Shift & Load 32	Control display shift registers

Table 31. Clocking Mode Register *(2 of 2)*

Bit	Function	Detail
3	Dotclk	0 = Normal operation 1 = Divide by 2
2	Shift and Load 16	Control display shift register
1	Reserved	Hardwired to 0
0	8/9 Dot clocks	0 = 9 dot wide character clock 1 = 8 dot wide character clock

Table 32. Shift and Load Control Bits

Bits 4 and 2	Function
00	Every character clock
01	Every 2nd character clock
10	Every 4th character clock
11	Every 4th character clock

Map Mask Register

name: SEQ_WPMASK
index: 3C5h.02h, read/write
size: 8 bits
function: This register masks write access to the video buffer planes by the host CPU. The bit contents of the Map Mask Register are shown in Table 33.

Table 33. Map Mask Register

Bit	Function	Detail
7:4	Reserved	Hardwired to 0h
3	Map3 enable	0 = no write 1 = write allowed
2	Map2 enable	0 = no write 1 = write allowed
1	Map1enable	0 = no write 1 = write allowed
0	Map0 enable	0 = no write 1 = write allowed

Character Map Select Register

name: SEQ_CFS
index: 3C5h.03h, read/write
size: 8 bits
function: This register determines which character font will be used for display output. Bit 7 of the text attribute byte determines whether the primary or secondary font is used. The bit contents of the Character Map Select Register are shown in Table 34.

Table 34. Character Map Select Register

Bit	Function
7:6	Reserved - hardwired to 00
5	Secondary 0
4	Primary 0
3	Secondary 2
2	Secondary 1
1	Primary 2
0	Primary 1

Table 35 and Table 36 define the Primary and Secondary bits used to select font tables. This is only valid for Text modes. Text fonts are normally loaded into video buffer planes 2 or 3. The offset in the tables below refer to the offset from the base of plane 2 or 3.

Table 35. Secondary Font Selection

Bits 5, 3, 2	Map Selected	Plane Offset
000	0	0 K
001	1	16 K
010	2	32 K
011	3	48 K
100	4	8 K
101	5	24 K
110	6	40 K
111	7	56 K

Table 36. Primary Font Selection

Bits 4, 1, 0	Map Selected	Plane Offset
000	0	0 K
001	1	16 K
010	2	32 K
011	3	48 K
100	4	8 K
101	5	24 K
110	6	40 K
111	7	56 K

Memory Mode Register

name: SEQ_MMODE
index: 3C5h.04h, read/write
size: 8 bits
function: This register controls miscellaneous setup features related to the video buffer and sequencer control. The bit contents of the Memory Mode Register are shown in Table 37.

Table 37. Memory Mode Register

Bit	Function	Detail
7:4	Reserved	Hardwired to 0h
3	Chain 4	0 = Normal addressing 1 = Packed pixel mode; CPU address bits A1 and A0 determine which plane is accessed, and buffer address bits A3, A2 are replaced by CPU address bits A15, A14.
2	Odd/Even	0 = Even CPU addresses access Map0 and Map2, odd CPU addresses access Map1 and Map3 1 = Normal addressing
1	Extended Memory	This bit only affects the odd/even addressing functionality controlled by Chain Odd/Even bit in the VGA Graphics Controller Miscellaneous Register. This bit never needs to be set to zero for Bt2166 systems. 0 = 64 KB of video memory 1 = 256 KB of video memory for VGA modes
0	Reserved	Hardwired to 0

Unlock Register

name: SEQ_UNLOCK
index: 3C5h.07h, read/write
size: 8 bits
function: This register enables or disables access to the Bt2166 extension registers. If the extensions are unlocked, reading this register will return a status value. If the extensions are locked, reading this register will return 0Fh. The bit contents of the Unlock Register are shown in Table 38.

Table 38. Sequencer Unlock Register

Bit	Function	Detail
7:5	Reserved	
4:0	Unlock	Writing 12h to this field unlocks the extension registers. Reads a 12h status. Writing 14h to this field saves VGA context and unlocks the VGA at interrupt service routine entry. Reads a 14h status. Writing 18h to this field restores the VGA context after an interrupt. Reads back the previous status value (never contains 18h)

The Bt2166 implements a context save restore mechanism in the VGA block, that drastically reduces the I/O overhead for entering and leaving the interrupt-service routine.

When 14h is written to the Sequencer Unlock Register (3C5h.7), the current state is saved as follows:

1. Copy current Sequencer Lock Register value to a shadow byte4.
2. Copy current Graphics Index Register (3CE) to a shadow byte0.
3. Copy GUI_BASE0 to a shadow byte1.
4. Copy GUI_BASE1 to a shadow byte2.
5. Copy GUI_BASE2 to a shadow byte3.
6. Force unlock state in Sequencer 7 Register

When 18h is written to the Sequencer Unlock Register, the following occurs:

1. Copy shadow byte 4 back to Sequencer Lock Register
2. Copy shadow byte 0 back to Graphics Index Register
3. Copy shadow byte 1 back to GUI_BASE0
4. Copy shadow byte 2 back to GUI_BASE1
5. Copy shadow byte 3 back to GRP_GUI_BASE2

The ISR input is as follows:

1. Save Sequencer Index Register 3C4h
2. Write sequencer index to address the unlock and unlock, 7->3C4h
3. Write save value (14h) to 3C5h (2Ah and 2Bh are one out16)
4. Write to Graphics Index Register and GRP_GUI_BASE0 (3CFh.20)
5. Write to Graphics Index Register and GRP_GUI_BASE1 (3CFh.21)
6. Write to Graphics Index Register and GRP_GUI_BASE2 (3CFh.22)

Note: For information on GRP_GUI_BASE, refer to ‘‘GUI Base Address Registers’’ on page 46.

7. Read interrupt status through 000A0000h and process

The return from ISR is as follows:

1. Write restore value (18h) to Sequencer Unlock Register to restore Unlock Register and Shadowed Graphics Register
2. Restore Sequencer Index Register 3C4h

VGA CRT Controller

The CRT Controller (CRTC) uses indexed addressing to access the control registers. The offset into the VGA CRTC data space is 3?5h, where "?" is either B (3B5h) or D (3D5h) depending on whether the CRT is monochrome or color, respectively.

CRTC Color Index

name: CRT_INDEX_C
port: 3D4h, read/write
size: 8 bits
function: This address accesses the CRT_INDEX register when VGA Miscellaneous Output Register bit 0 is set for a color display. This register contains the location of the register in the CRTC block to be accessed by the next I/O read or write to the CRT_DATA port.

CRTC Monochrome Index

name: CRT_INDEX_M
port: 3B4h, read/write
size: 8 bits
function: This address accesses the CRT_INDEX register when VGA Miscellaneous Output Register bit 0 is set for a monochrome display. This register contains the location of the register in the CRTC block to be accessed by the next I/O read or write to the CRT_DATA port.

CRTC Color Data

name: CRT_DATA_C
port: 3D5h, read/write
size: 8 bits
function: This port allows access to one of the CRT Data registers, according to the value of the CRT_INDEX register. This register is accessible only for color, MISC_OUT_R[0] = 1 (see Table 59 on page 77).

CRTC Monochrome Data

name: CRT_DATA_M
port: 3B5h, read/write
size: 8 bits
function: This port allows access to one of the CRT Data registers, according to the value of the CRT_INDEX register. This register is accessible only for monochrome, MISC_OUT_R[0] = 0 (see Table 59 on page 77).

Horizontal Total Register

name: CRT_HTOT
index: 3?5h.00h, read/write
size: 8 bits
function: This register sets the horizontal scan time and is loaded in terms of character time. It includes left and right borders, displayed characters, and horizontal retrace time (front porch, back porch, and sync pulse). If 'n' is the total line character count, then this register should be loaded with 'n-5'.

Horizontal Display End Register

name: CRT_HDSP_END
index: 3?5h.01h, read/write
size: 8 bits
function: This register contains the number of displayed characters on a horizontal line, and is loaded in terms of character clocks. If ‘n’ is the character count, then this register should be loaded with ‘n-1’.

Start Horizontal Blank Register

name: CRT_HBLANK_ST
index: 3?5h.02h, read/write
size: 8 bits
function: This register contains the count at which horizontal blanking should begin. It is in terms of character clocks. If ‘n’ is the character count, then this register should be loaded with ‘n-1’.

End Horizontal Blank Register

name: CRT_HBLANK_END
index: 3?5h.03h, read/write
size: 8 bits
function: This register contains the count at which horizontal blanking should end, in terms of character clocks. If ‘n’ is the character count, then this register should be loaded with ‘n-1’. The MSB of the End Horizontal Blank field is in the End Horizontal Sync Register. Together, these two registers specify only the least significant 6 bits of the count at which blanking will end. The actual 8-bit count at which blanking will end is the first time after the start blanking count that the least significant 6 bits of the character counter match these 6 bits. The bit contents of the End Horizontal Blank Register are shown in Table 39.

Table 39. End Horizontal Blank Register

Bit	Function	Detail
7	Compatible Read	This field has no effect on Bt2166 operation.
6:5	Display Enable Skew	This field has no effect on Bt2166 operation.
4:0	HBLANK <4:0>	End Horizontal Blank count. The MSB is in the End Horizontal Sync Register.

Start Horizontal Sync Register

name: CRT_HSYNC_ST
index: 3?5h.04h, read/write
size: 8 bits
function: This register contains the count at which the horizontal sync pulse should begin. It is in terms of character clocks. If ‘n’ is the character count, then this register should be loaded with ‘n-1’.

End Horizontal Sync Register

name: CRT_HSYNC_END
index: 3?5h.05h, read/write
size: 8 bits
function: This register contains the count at which the horizontal sync pulse should end. It is in terms of character clocks. If 'n' is the character count, then this register should be loaded with 'n-1'. This register specifies only the least significant 5 bits of the count at which the sync pulse will end. The actual 8-bit count at which it will end is the first time after the start horizontal sync count that the least significant 5 bits of the character counter match these 5 bits. The bit contents of the End Horizontal Sync Register are shown in Table 40.

Table 40. End Horizontal Sync Register

Bit	Function	Detail
7	HBLANK 5	MSB of HBLANK
6:5	Reserved	
4:0	HSYNC [4:0]	Least significant 5 bits of Start Horizontal Sync Register plus the width of HSYNC in character clocks.

Vertical Total Register

name: CRT_VTOT
index: 3?5h.06h, read/write
size: 8 bits
function: This register sets the number of scan lines in the frame. It contains the 8 least significant bits. The most significant 2 bits are in the Overflow Register. If 'n' is the scan line count, then this register should be loaded with 'n-2'.

Overflow Register

name: CRT_OVERFLOW
index: 3?5h.07h, read-only
size: 8 bits
function: This register contains miscellaneous overflow bits associated with other vertical timing registers. The bit contents of the Overflow Register are shown in Table 41.

Table 41. Overflow Register

Bit	Function
7	Start VSYNC 9
6	End Vertical Display Enable 9
5	Vertical Total 9
4	Line Compare 8
3	Start VBLANK 8
2	Start VSYNC 8
1	End Vertical Display Enable 8
0	Vertical Total 8

Preset Row Scan Register

name: CRT_PRE_RS
index: 3?5h.08h, read/write
size: 8 bits
function: This register is used for line scrolling. The bit contents of the Preset Row Scan Register are shown in Table 42.

Table 42. Preset Row Scan Register

Bit	Function	Detail
7	Reserved	Hardwired to 0
6:5	Byte Pan	This field has no effect on Bt2166 operation.
4:0	Preset Row Scan Count	Specify scan line for 1st character row

Character Height Register

name: CRT_CHEIGHT
index: 3?5h.09h, read/write
size: 8 bits
function: This register specifies the cell height for characters and also contains 2 overflow bits and the scan line double bit. The bit contents of the Character Height Register are shown in Table 43.

Table 43. Character Height Register

Bit	Function	Detail
7	Scan Double	0 = normal operation 1 = activate line doubling
6	Line Compare 9	Overflow bit
5	Start VBLANK 9	Overflow bit
4:0	Cell Height	Number of scan lines / character

Cursor Start Register

name: CRT_CUR_ST
index: 3?5h.0Ah, read/write
size: 8 bits
function: This register sets the first scan line to display the cursor within the character cell which contains the cursor. It also turns the cursor off and on. The bit contents of the Cursor Start Register are shown in Table 44.

Table 44. Cursor Start Register

Bit	Function	Detail
7:6	Reserved	Hardwired to 00
5	Cursor Toggle	0 = cursor on 1 = cursor off
4:0	Top of Cursor	Starting scan line for cursor

Cursor End Register

name: CRT_CUR_END
index: 3?5h.0Bh, read/write
size: 8 bits
function: This register sets the last scan line to display the cursor within the character cell which contains the cursor. The bit contents of the Cursor End Register are shown in Table 45.

Table 45. Cursor End Register

Bit	Function	Detail
7	Reserved	Hardwired to 0
6:5	Cursor Skew	This field has no effect on Bt2166 operation.
4:0	Bottom of Cursor	Ending scanline for cursor

Start Address High Register

name: CRT_SCREEN_STH
index: 3?5h.0Ch, read/write
size: 8 bits
function: This register sets the high order 8 bits of the 16-bit quantity which determines the frame buffer address of the upper left corner of the display screen. The start address is in terms of character positions. The actual frame buffer starting address is scaled by a factor of 1 , 2, or 4 depending on whether the byte, word, or double-word CRT addressing mode is selected. See the CRT Mode Control and CRT Underline Register definitions.

Start Address Low Register

name: CRT_SCREEN_STL
index: 3?5h.0Dh, read/write
size: 8 bits
function: This register sets the low order 8 bits of the 16-bit quantity which determines the frame buffer address of the upper left corner of the display screen. The start address is in terms of character positions. The actual frame buffer starting address is scaled by a factor of 1 , 2, or 4 depending on whether the byte, word, or double-word CRT addressing mode is selected. See the CRT Mode Control and CRT Underline Register definitions.

Cursor Location High Register

name: CRT_CUR_LOCH
index: 3?5h.0Eh, read/write
size: 8 bits
function: This register sets the high order 8 bits of the 16-bit quantity that controls the cursor location. The location is in terms of character positions. It points to the character in the frame buffer which will contain the cursor. The actual frame buffer cursor address is scaled by a factor of 1, 2, or 4 depending on whether the byte, word, or double-word CRT addressing mode is selected. See the CRT Mode Control and CRT Underline Register definitions.

Cursor Location Low Register

name: CRT_CUR_LOCL
index: 3?5h.0Fh, read/write
size: 8 bits
function: This register sets the low order 8 bits of the 16-bit quantity that controls the cursor location. The location is in terms of character positions. It points to the character in the frame buffer which will contain the cursor. The actual frame buffer cursor address is scaled by a factor of 1, 2, or 4 depending on whether the byte, word, or double-word CRT addressing mode is selected. See the CRT Mode Control and CRT Underline Register definitions.

Start Vertical Sync Register

name: CRT_VSYNC_ST
index: 3?5h.10h, read/write
size: 8 bits
function: This register determines the start of the vertical sync pulse. The scan line counter is compared to this register at each horizontal sync time. This register contains the lower 8 bits of this value. Bits 8 and 9 are in the Overflow Register. If ‘n’ is the scan line count, then this register should be loaded with ‘n-1’.

End Vertical Sync Register

name: CRT_VSYNC_END
index: 3?5h.11h, read/write
size: 8 bits
function: This register determines the end of the vertical sync pulse. The horizontal scan line count is compared to the Start Vertical Sync Register. If the values are equal, a vertical sync pulse is ended. This register specifies only the least significant 4 bits of the 10-bit count at which the sync pulse will end. The actual count at which it will end is the first time after the start vertical sync count that the least significant 4 bits of the line counter match these 4 bits. If 'n' is the scan line count, then this register should be loaded with 'n-1'. Miscellaneous control fields are also supported in this register. The bit contents of the End Vertical Sync Register are shown in Table 46.

Table 46. End Vertical Sync Register

Bit	Function	Detail
7	CRTC [7:0] Write Protect	0 = Enable writes to CRT Index registers 7:0h 1 = Disable writes to CRT Index registers 7:0h, the Line Compare bit in the Overflow Register is not affected
6	Reserved	
5	Enable Vertical Sync Interrupt	Enabled with FRAME_SYNC interrupt. See Table 108 on page 138. 0 = Enable vertical sync interrupt 1 = Disable vertical sync interrupt
4	Clear Vertical Sync Interrupt	Enabled with VGA_RETRACE_INT. See Table 108 on page 138. 0 = Clear vertical sync interrupt 1 = Allows vertical sync interrupt generation
3:0	Vertical Sync End	Scan line count to end VSYNC

Vertical Display End Register

name: CRT_VDSP_END
index: 3?5h.12h, read/write
size: 8 bits
function: This register determines the number of scan lines displayed with frame buffer data. This register contains the lower 8 bits of the display end value. Bits 8 and 9 are stored in the Overflow Register. If 'n' is the scan line count, then this register should be loaded with 'n-1'.

Offset Register

name: CRT_OFFSET
index: 3?5h.13h, read/write
size: 8 bits
function: This register specifies the pitch in bytes between adjacent character row or scan lines. The next row start address (dword address) equals the current row start address plus (K * value in Offset Register), where K has a value of 2 in byte mode, 4 in word mode, and 8 in dword mode. Byte or word addressing is set by the CRTC Mode Control Register.

Underline Register

name: CRT_UNDERLINE
index: 3?5h.14h, read/write
size: 8 bits
function: This register includes the scan line location for underlining characters and the double word addressing control bits. The bit contents of the Underline Register are shown in Table 47.

Table 47. Underline Register

Bit	Function	Detail
7	Reserved	Hardwired to 0.
6	Dword Mode	0 = CRT addresses frame buffer in byte or word mode 1 = CRT addresses frame buffer in dword mode.
5	Count by Four	0 = CRT increments frame buffer pointer on every character clock unless Count-by-Two bit in the CRT Mode Control Register is set. 1 = CRT increments frame buffer pointer once every four character clocks
4:0	Underline Location	Scan line value within a character cell where the underline character is displayed. Lines 1 through n are specified by values 0 through n-1.

Start Vertical Blank Register

name: CRT_VBLANK_ST
index: 3?5h.15h, read/write
size: 8 bits
function: This register determines the start of vertical blanking. The scan line counter is compared to this register. When the values are equal, vertical blank begins. The lower 8 bits are loaded in this register. Bit 8 is stored in the Overflow Register and bit 9 is stored in the Character Height Register. If ‘n’ is the scan line count, then this register should be loaded with ‘n-1’.

End Vertical Blank Register

name: CRT_VBLANK_END
index: 3?5h.16h, read/write
size: 8 bits
function: This register determines the end of vertical blanking. The scan line counter is compared to this register. When the values are equal, vertical blank ends. This register specifies only the least significant 8 bits of the 10-bit count at which the blanking period will end. The actual count at which it will end is the first time after the start vertical blank count that the least significant 8 bits of the line counter match these 8 bits. If ‘n’ is the scan line count, then this register should be loaded with ‘n-1’.

Mode Control Register

name: CRT_MODE_CTL
index: 3?5h.17h, read/write
size: 8 bits
function: This register controls several aspects of the display setup. The bit contents of the Mode Control Register are shown in Table 48.

Table 48. Mode Control Register

Bit	Function	Detail
7	Sync enable	0 = HSYNC and VSYNC inactive 1 = HSYNC and VSYNC active
6	Word or Byte Mode	This bit is overridden by the Double-Word Mode bit in CRT Underline Register. 0 = Word mode - Frame Buffer Address is incremented by two 1 = Byte mode. Frame Buffer Address is incremented by one
5	Address Wrap	0 = In word mode, memory address 14 appears at memory address 0 1 = Select memory address 16 for odd/even mode when 256 KB of video memory is present
4	Reserved	
3	Count by Two	0 = Character clock increments memory address counter 1 = Character clock/2 increments memory address counter
2	Vertical Count Double	0 = Vertical line counter increments once per hsync 1 = Vertical line counter increments once every two hsync pulses, effectively doubling all vertical counts.
1	Select Row Scan Counter	0 = Row scan counter bit 1 is output at memory address 14 1 = Bit 14 of the CRTC address counter is output at memory address 14
0	Compatibility Mode	0 = Row scan counter bit 0 is output for memory address 13 1 = Memory address bit 13 unchanged

Line Compare Register

name: CRT_LINECMP
index: 3?5h.18h, read/write
size: 8 bits
function: This register is used for split screen operation. When the scan line counter equals the line compare value, the CRT's frame buffer pointer is cleared causing the lines beneath the compare scan line to be updated from video buffer display location 0. The lower 8 bits are loaded in the Line Compare Register. Bit 8 is in the Overflow Register and bit 9 is located in the Character Height Register. If 'n' is the scan line count, then this register should be loaded with 'n-1'.

VGA Attribute Controller

The Attribute Controller uses indexed addressing to access 20 control registers. Sixteen of these registers are Palette registers. The attribute controller uses a single I/O port at 3C0h to write both the Index Register and the Data registers. Reading the 3C0h address will always return the value of the Index Register. A second port is available at 3C1h to read the value of the Indexed Data Register.

The VGA contains internal state which keeps track of whether the index or data registers will be accessed on the next write. Each write access toggles the state between index and data registers. The Attribute Controller Register write mechanism can be reset to point to a known state. This is done by reading the Input Status #1 Register (3xAh). This resets the 3C0h write port to point to the Index Register.

Attribute Index and Data Port

name: ATT_ADDR
port: 3C0h, read/write
size: 8 bits
function: This register can always be read at 3C0h. However, when writing to 3C0h, the state of the port toggles between writing to the Index Register and the Data Register. The Data Register written to is determined by the value last written to the Index Register.

If the state is unknown, issue a read of the Input Status #1 Register (3xAh) to reset the port for writing the Index Register.

Bit 5 of this register must be a zero to allow the CPU to read or write the palette registers. Otherwise, writes are ineffectual and reads return 3Fh. Also, if bit 0 is not set to one upon completing the palette modifications, then the VGA refresh logic will not be able to access the palette and the screen will be all one color.

The bit contents of the Attribute Index and Data Port Register are shown in Table 49.

Table 49. Attribute Index/Data Port Register

Bit	Function	Detail
7:6	Reserved	Hardwired to 00
5	Video On	0 = CPU drives the Palette address 1 = Refresh logic drives the Palette address
4:0	Attribute Index	0-Fh palette registers 10h mode control 11h overscan color 12h color plane enable 13h horizontal panning 14h color select

Read Data Port

name: ATT_RD
port: 3C1h, read-only
size: 8 bits
function: Data registers are always read from the Read Data Port. Reading the Data Register does not affect the Index Register. The bit contents of the Read Data Port Register are shown in Table 49.

Table 50. Read Data Port Register

Bit	Function
7:5	Reserved.
4:0	Index to the data registers in the Attribute Controller block.

Palette Registers

name: ATT_PAL_REG[15:0]
index: 3C0h.0Fh-00h, read/write
size: 8 bits each
function: These 16 registers comprise the EGA palette. In all BIOS modes, frame buffer color indices are translated to an output color index through this palette. The bit contents of the Palette Register are shown in Table 51.

Table 51. Palette Registers

Bit	Function
7:6	Reserved. Hardwired to 00
5	Secondary Red
4	Secondary Green/Intensity
3	Secondary Blue/Mono
2	Red
1	Green
0	Blue

Mode Control Register

name: ATT_MODE
index: 3C0h.10h, read/write
size: 8 bits
function: This register controls various attribute modes. The bit contents of the Mode Control Register are shown in Table 52.

Table 52. Mode Control Register

Bit	Function	Detail
7	Video Select	0 = Bits 4 and 5 of the Palette registers are used as address bits 1 = Use ATT_CLRSEL[1:0] as address bits
6	PEL Width	This bit determines how pixels are assembled at the output of the EGA palette. When zero, the 6-bit output of the EGA palette is sent on every pixel clock. When one, the least significant 4 bits of EGA palette output from two consecutive pixel clocks are assembled into one 8-bit quantity. This 8-bit value is then sent to the DAC for two consecutive clocks. 0 = Normal mode 1 = 256 color mode
5	Pixel Pan	0 = A line compare has no effect on pixel panning 1 = A line compare terminates pixel panning for remainder of frame.
4	Reserved	hardwired to 0
3	Blink/Intensity	0 = Blink inactive, set background intensity 1 = Blink active
2	Line Graphics Enable	0 = The 9th bit of a 9-pixel character will be the same as the background 1 = The 9th bit of a 9-pixel character will be the same as the 8th bit for character code between C0-DFh
1	Display Type	This bit only effects graphics mode blinking. 0 = Color display attributes 1 = Monochrome attributes
0	Mode Enable	0 = Text mode 1 = Graphics mode

Overscan Color Register

name: ATT_OVRS
index: 3C0h.11h, read/write
size: 8 bits
function: The Overscan Color Register sets the color for the border area of the display. For monochrome displays this register is set to 0.

Color Plane Enable Register

name: ATT_CPE
index: 3C0h.12h, read/write
size: 8 bits
function: This register enables the four planes into the Palette registers. This register also sets the inputs for the diagnostics bits in the Input Status #1 Register. The bit contents of the Color Plane Register are shown in Table 53.

Table 53. Color Plane Enable Register

Bit	Function	Detail
7:6	Reserved	Hardwired to 00
5:4	Video Status Mux	00 = Input Status #1[5:4] = P2-Red, P0-Blue 01 = Input Status #1[5:4] = P5-Secondary Red, P4-Secondary Blue 10 = Input Status #1[5:4] = P3-Secondary Blue, P1-Green 11 = Input Status #1[5:4] = P7, P6 (256-color modes only) See Table 58 on page 77.
3	Enable Plane 3	0 = Disable plane, force pixel bit to 0 before addressing palette 1 = Enable color plane
2	Enable Plane 2	See bit 3
1	Enable Plane 1	See bit 3
0	Enable Plane 0	See bit 3

Horizontal Panning Register

name: ATT_HPAN
index: 3C0h.13h, read/write
size: 8 bits
function: This register is available for text or graphics modes and specifies the number of pixels the display data will be shifted left horizontally. In text mode, characters can be shifted left one pixel less than the cell width. In 256 color modes, up to 3 position pixel shifts may occur. Table 54 gives the bit contents of the Horizontal Panning Register. Table 55 lists the allowable pixel pans.

Table 54. Horizontal Panning Register

Bit	Function
7:4	Reserved. Hardwired to 0h
3:0	Pixel Pan Amount

Table 55. Allowable Pixel Pans

Pixel Pan Amount	9-Bit Character	8-Bit Character	256 Color Mode
0	1	0	0
1	2	1	--
2	3	2	1
3	4	3	--
4	5	4	2
5	6	5	--
6	7	6	3
7	8	7	--
8–Fh	0	--	--

Color Select Register

name: ATT_CLRSEL
index: 3C0h.14h, read/write
size: 8 bits
function: This register contains two fields used to select locations in the video DAC. The bit contents of the Color Select Register are shown in Table 56.

Table 56. Color Select Register

Bit	Function	Detail
7:4	Reserved	Hardwired to 0h
3:2	Color Bits[7:6]	These bits are concatenated with the lower 6 bits from the Palette registers to form an index into the video DAC. These bits are ignored in 256 color modes.
1:0	Color Bits[5:4]	If ATT_MODE[7] = 1, these bits replace the corresponding bits in the Palette registers to form an index into the video DAC. Otherwise, these bits are ignored. These bits are ignored in 256-color modes. These bits are ineffective in 8, 16, and 24 bpp modes.

VGA General/External Registers

The VGA general/external registers are directly accessible from the I/O bus.

Input Status #0 Register

name: INP_STAT0
port: 3C2h, read-only
size: 8 bits
function: The Input Status #0 Register contains the vertical retrace interrupt bit. The bit contents of the Input Status #0 Register are shown in Table 57.

Table 57. Input Status #0 Register

Bit	Function	Detail
7	Vertical Retrace Interrupt	0 = interrupt cleared 1 = interrupt pending
6:5	Reserved	Hardwired to b'00
4	Monitor Detection	This field is undefined. Monitor status is available by accessing registers on the Display Refresh Controller device.
3:0	Reserved	Hardwired to 0h

Color Input Status #1 Register

name: INP_STAT_C
port: 3DAh color, read-only
size: 8 bits
function: This register contains assorted status bits. Reading this register resets the state of the Attribute Controller index/write port for color systems to 3C0h. The bit contents of the Input Status #1 Register (for both color and monochrome) are shown in Table 58. This register is accessible only if MISC_OUT_R[0] = 1 (see Table 59 on page 77).

Monochrome Input Status #1 Register

name: INP_STAT_M
port: 3BAh, read-only
size: 8 bits
function: This register contains assorted status bits. Reading this register resets the state of the Attribute Controller index/write port for monochrome systems to 3C0h. The bit contents of the Input Status #1 Register (for both color and monochrome) are shown in Table 58. This register is accessible only if MISC_OUT_R[0] = 0 (see Table 59 on page 77).

Table 58. Input Status #1 Register

Bit	Function	Detail
7:6	Reserved	Hardwired to b'00
5:4	Video status mux	Multiplexed Video Bits as selected by bits 5:4 of the Attribute Controller Color Plane Enable Register. 00 = P2, P0 01 = P5, P4 10 = P3, P1 11 = P7, P6 See Table 53 on page 74
3	Vertical Retrace Status	0 = Vertical Retrace inactive 1 = Vertical Retrace active
2:1	Reserved	Hardwired to b'10
0	Display Enable Status	0 = Display active 1 = Display inactive-horizontal or vertical retrace active

Write Miscellaneous Output Register

name: MISC_OUT_W
port: 3C2h, write-only
size: 8 bits
function: This register contains write-only fields related to setting up the VGA environment. The bit contents of the Miscellaneous Output registers (for both write and read) are shown in Table 59 on page 77. What is written here is read at 3CCh.

Read Miscellaneous Output Register

name: MISC_OUT_R
port: 3CCh, read-only
size: 8 bits
function: This register contains several read-only fields related to setting up the VGA environment. The bit contents of the Miscellaneous Output registers (for both write and read) are shown in Table 59.

Table 59. Miscellaneous Output Register (Read and Write) *(1 of 2)*

Bit	Function	Detail
7	VSYNC Polarity *(1)*	0 = Active high 1 = Active low
6	HSYNC Polarity*(1)*	0 = Active high 1 = Active low
5	Page Select	This bit determines affects the value used as the least significant bit in Odd/Even addressing modes. It is for diagnostic use, and should be set to one for normal use.
4	Reserved	Hardwired to 0

Table 59. Miscellaneous Output Register (Read and Write) *(2 of 2)*

Bit	Function	Detail
3:2	Clock Select [1:0]	00 = Select 25.180 MHz Pixel Clock 01 = Select 28.325 MHz Pixel Clock 10 = Pixel clock rate determined by PLL value written to media buffer 11 = reserved
1	Enable Frame Buffer Access	0 = Disable CPU access to VGA frame buffer 1 = Enable CPU access to VGA frame buffer
0	CRTC I/O Address	0 = Select monochrome - 3Bh 1 = Select color - 3Dh
Note: (1).Monitors select their vertical scan rate and gain based on HSYNC and VSYNC polarity, which is set by bits [7:6] . The selection is: 0 = Reserved; 1 = 400 line mode; 2 = 350 line mode; 3 = 480 line mode.		

Feature Control Register

name: FC
port: 3CAh read
3xAh write: 3BAh if MISC_OUT_R[0] = 0 (mono), or 3DAh if MISC_OUT_R[0] = 1 (color)
size: 8 bits
function: The Feature Control Register prevents bus timeouts caused by previous code releases; data read is indeterminate.

VGA Color Registers

The Bt2166 supports the following VGA DAC registers (color registers) in addition to the DAC Alias registers in the Graphics Controller extended address space.

DAC Mask Register

name: DAC_MASK
port: 3C6, read/write
size: 8 bits
function: This 8-bit register contains a bit mask of the pixel value going to the DAC lookup table (VGA palette, or color registers). A 1 enables the corresponding bit, while a 0 forces the bit to zero before it addresses the color lookup table. This register does not affect access to the color palette through the 3Cxh CPU ports. It only affects the pixel value coming from the Attribute logic for display on the screen. This register is normally not written by application code but initialized by BIOS or device drivers to establish the color lookup table.

DAC State Register

name: DAC_STATE
port: 3C7h, read-only
size: 8 bits
function: This register shows whether the DAC Data read/write port is in a read or write mode, depending on whether the DAC Read Address Register or DAC Write Address Register was written to most recently. The bit contents of this register are shown in Table 60.

Table 60. Color Registers: DAC State Register

Bit	Function	Detail
7:2	Reserved	Hardwired to b'000000
1:0	State	00 = Write mode 11 = Read mode

DAC Read Address Register

name: DAC_RDADDR
port: 3C7h, write-only
size: 8 bits
function: This register contains an 8-bit address that selects one of 256 video DAC Color registers during a read operation. The next three reads of the DAC Data Register return three 6-bit values for RED, GREEN, and BLUE settings, respectively. After the BLUE value is read, the value in the DAC Read Address Register is incremented to point to the next DAC Color Register. This feature allows the programmer to read all 768 values from the DAC without updating the index value in the DAC Write Address Register. This value will also wrap from 255 to 0. The DAC Read Address can be written with another value at any time between each set of three reads to the DAC Data Register.

DAC Write Address Register

name: DAC_WRADDR
port: 3C8h, read/write
size: 8 bits
function: This register contains an 8-bit address selects one of 256 video DAC Color registers during a write operation. The next three writes to the DAC Data Register output three 6-bit values for RED, GREEN, and BLUE settings, respectively. After the BLUE value is output, the value in the DAC Write Address Register is incremented to point to the next DAC Color Register. This feature allows the programmer to output all 768 values to the DAC without updating the index value in the DAC Write Address Register. This value will also wrap from 255 to 0. The DAC Write Address Register can be written with another value at any time between each set of three writes to the DAC Data Register.

DAC Data Register

name: DAC_DATA
port: 3C9h, read/write
size: 8 bits
function: This register reads or writes 18 bits of palette data depending on the mode of operation. Writes to the DAC Read Address Register put this register in the read mode, while writes to the DAC Write Address Register put this register in the write mode.

Data access to and from this register is in groups of 3 bytes. Each byte contains the 6-bit palette value for the DAC output signal. The first byte contains the red signal value, the second byte contains the green signal value, and the third byte contains the blue signal value. On reads, the most significant two bits of each byte are always set to 0.

I^2C Register Definitions

I^2C Master and Slave Controllers

This chapter specifies the programming interface to both the I^2C master and I^2C slave modules of the Bt2166 Graphics/Video Controller. Because these blocks operate independently of each other, they are documented separately.

The Bt2166 supports two independent I^2C buses, referred to here as IIC and IIC_DDC. Both support ACCESS.bus protocols. IIC_DDC has additional logic to support VESA monitor DDC1, DDC2, and DDC2B signaling. Since there is only one master controller and one slave controller, only one I^2C bus can be designated as the ACCESS.bus. Typically if a DDC2B monitor is detected, it will provide fanout of the ACCESS.bus.

NOTE

If the VIP mode of the video interface is used, the IIC_DDC I^2C bus is no longer available. The remaining I^2C bus will need to be externally multiplexed for DDC2B monitor control.

For additional information, refer to “Video Interface Port” on page 164.

The I^2C block consists of the I^2C master module, the I^2C slave module and a small amount of shared logic. The I^2C master module is capable of transmitting and receiving data over the I^2C bus as an I^2C bus master. This functionality is required for the I^2C links to the and VideoStream Decoder chips and is also required to support the ACCESS.bus protocol which is built on top of the I^2C protocol. The I^2C slave module can receive data over the I^2C bus as an I^2C bus slave. This functionality is not required for the I^2C links to the VideoStream Decoder; it is required solely to support the ACCESS.bus protocol.

Two sets of I^2C pins allow the Bt2166 to support the IIC and IIC_DDC buses. This means that the VideoStream Decoder I^2C bus can be isolated from the ACCESS.bus bus. The I^2C master and slave modules can be independently switched from one bus to the other. The switching of the master and slaves modules to the IIC and IIC_DDC buses is controlled by the GRP_I2C_CTRL(R/W) Control Register, which is defined in the next subsection.

The I²C master module is capable of transmitting and receiving data over the I²C bus and is controlled through two VGA extension registers. Both master transmit and master receive modes are required by the Bt2166 to VideoStream Decoder I²C communication link. This module fulfills all these requirements. Support for the ACCESS.bus spec requires the master transmit and slave receive modes, however this module does not provide the slave receive mode; this must be fulfilled by the I²C slave module.

The I²C slave module is capable of receiving data over the I²C bus as a slave device and is controlled through two VGA extension registers.

This controller is designed to work either polled or interrupt driven in an operating environment that has many other interrupts being serviced for other devices. For example, if a clock interrupt occurs in the middle of an I²C receive using a polling design, no I²C data will be lost.

This module is designed to implement the operations of an I²C and ACCESS.bus interface *with the assistance of system software*. It is possible for the system software to make this hardware perform in violation of either the I²C or ACCESS.bus specs. This design assumes that the system software does not attempt to do any such illegal activities and the design makes no attempt to protect against this eventuality.

NOTE

The I²C interrupt is defined in Table 108 on page 138. For electrical specification and timing for I²C I/O stages and bus lines, refer to the Rockwell Semiconductor Systems, Brooktree Division publication *The I²C Bus Reference Guide.*

I²C Control WRITE Register

name: GRP_I2C_CTRLW
index: 4Bh, write
size: 8 bits
function: This register enables the I²C serial control interface. The bit contents of the GRP_I2C_CTRLW register are shown in Table 61.

Table 61. GRP_I2C_CTRLW Control Register

Bit	Function	Detail
7	I2CMA_SEL	1 = select IIC bus for master module 0 = select IIC_DDC bus for master module
6	I2CSL_SEL	1 = select IIC bus for slave module 0 = select IIC_DDC bus for slave module
5:4	reserved	set to zero.
3	OVR_IIC_SCL	Override SCL clock signal in IIC bus 1= release SCL clock 0= force SCL clock low
2	OVR_IIC_SDA	Override SDA data pins in IIC bus 1= release SDA data 0= force SDA data low
1	OVR_DDC_SCL	Override SCL clock signal in IIC_DDC bus 1= release SCL clock (must be set if VIP video interface is used) 0= force SCL clock low
0	OVR_DDC_SDA	Override SDA data pins in IIC_DDC bus 1= release SDA data (must be set if VIP video interface is used) 0= force SDA data low

As shown in Table 61, bits GRP_I2C_CTRLW[3:0] allow the override the output levels of the SCL clock and SDA data pins for both I²C buses. These overrides could be used to freeze one I²C bus while another is busy; for example, temporarily switch the I²C master module to the other bus for a transmit/receive. Or, these bits could be used to implement an I²C interface purely in software (a CPU-intensive task).

NOTE

If the VIP video interface mode is used, set GRP_I2C_CTRLW[1:0] to 2'b11 and GRP_I2C_CTRLW[7:6] to 2'b11.

Loopback occurs when the master and slave are both switched to the same IIC or IIC_DDC bus. Under the control of software, the master can receive data transmitted by the slave; and the slave can receive data transmitted by the master.

I²C Control READ Register

name: GRP_I2C_CTRLR
index: 4Bh, read only
size: 8 bits
function: This register enables the I²C serial control interface. The bit contents of the GRP_I2C_CTRLR register are shown in Table 61.

Table 62. GRP_I2C_CTRLR Control Register

Bit	Function	Detail
7	I2CMA_SEL	1 = Select IIC bus for master module 0 = Select IIC_DDC bus for master module
6	I2CSL_SEL	1 = Select IIC bus for slave module 0 = Select IIC_DDC bus for slave module
5	I2CMA_DONE	I²C master module DONE. Read-only value from GRP_I2C_MCTRLR
4	I2CSL_DONE	I²C slave module DONE. Read-only value from GRP_I2C_SCTRLR
3	RD_IIC_SCL	Read SCL clock signal in IIC bus (read only)
2	RD_IIC_SDA	Read SDA data pins in IIC bus (read only)
1	RD_DDC_SCL	Read SCL clock signal in IIC_DDC bus (read only)
0	RD_DDC_SDA	Read SDA data pins in IIC_DDC bus (read only)

As shown in Table 61, bits GRP_I2C_CTRLR[3:0] provide for reading the pin values of the four I²C pins. These bits could be used to implement an I²C interface purely in software (a CPU-intensive task). These bits also provide support for VESA monitor DDC1 signaling.

I²C Master Module Software Interface

The software interface to the I²C master block is via two VGA Graphics Extension registers: GRP_I2C_MCTRL (index 4Eh) and GRP_I2C_MDATA (index 4Fh). Both of these registers behave differently when read and written so the interface actually consists of two write-only registers and two read-only registers.

In general it is necessary to access both the GRP_I2C_MCTRL and GRP_I2C_MDATA registers at least once per byte of data transmitted / received. The order of reading the GRP_I2C_MCTRL and GRP_I2C_MDATA registers is important. The data received in the GRP_I2C_MDATA register should not be considered valid until after the DONE bit has been read as "1" in the GRP_I2C_MCTRLR register. Similarly the TRAN or RECV bits in the GRP_I2C_MCTRLW register (start to transmit / receive data) should not be written "1" until after the relevant data has been read from or written to the GRP_I2C_MDATA register.

Note that the I²C master module detects but does not recover from arbitration loss, non-acknowledgment of data, etc. It is the responsibility of the system software to handle any problems like this that occur and resubmit any failed action to the I²C master module.

The software interface to the I²C master module is via the following registers.

I²C Master Data Register

name: GRP_I2C_MDATA
index: 4Fh, read/write
size: 8 bits
function: This register supplies the contents of the input data buffer containing data received over the I²C bus. The GRP_I2C_MDATA register bit contents are given in Table 63.

Table 63. I²C Master Receive Data GRP_I2C_MDATA

Bit	Function	Detail
7:0	DATA	Received data byte/ data byte to transmit

If this register is read while data reception is still in progress the data read will be incorrect (not all bits will be shifted in yet). It is the responsibility of the system software to wait until the data reception on the I²C bus is complete before it considers the data read from the GRP_I2C_MDATA register to be correct. This can be done by examining the DONE bit in the GRP_I2C_MCTRLR register and waiting until it is "1."

When written, the GRP_I2C_MDATA register supplies the data to be transmitted over the I²C bus. If this register is written while data transmission is still in progress the data transmitted will be incorrect. It is the responsibility of the system software to wait until the data transmission on the I²C bus is complete before it attempts to write to the GRP_I2C_MDATA register. This can be done by examining the DONE bit in the GRP_I2C_MCTRLR register and waiting until it is "1" before writing to the GRP_I2C_MDATA register.

Please note that the registers used for data reads and writes are the same, writing data to the GRP_I2C_MDATA register will overwrite any unread data from the previous I²C master module reception. Similarly, any data written to the GRP_I2C_MDATA register could be corrupted by an I²C receive operation. Also, if two consecutive I²C transmit operations are performed with the same data, you still need to rewrite the required data to the GRP_I2C_MDATA register between the two I²C transmissions to remain compatible with future hardware revisions.

Finally, note that upon reset the initialization of the contents of this register is not defined. Assume that after a reset this register contains junk data until data has been written to it by the system software or until data has been loaded into it by an I²C master module receive operation.

I²C Master Control Read Register

name: GRP_I2C_MCTRLR
index: 4Eh, read only
size: 8 bits
function: This register supplies status information about the progress / completion of any I²C bus transaction. The GRP_I2C_MCTRLR register bit contents are given in Table 64.

Table 64. I²C Master Read Control Register GRP_I2C_MCTRLR[7:0] Bit Descriptions

Bit	Function	Detail
7	DONE	The DONE bit is set to “1” whenever the I²C master module has finished its current operation and is ready for another request from the system software. It is set to “0” whenever the I²C master module is busy servicing a request. This bit can be used by the system software to determine whether the I²C master module was the source of an interrupt (interrupts enabled) or to check whether the I²C master finished the current request (software polling, interrupts disabled). This bit is used in the generation of the I²C interrupt (it is ANDed with the INTE interrupt enable bit together with any outstanding requests in the GRP_I2C_MCTRLW register) but it is set independently of the value of INTE (interrupt enable). It is set only when all of the request bits (TSTA, TRAN, RECV and TSTO) in the GRP_I2C_MCTRLW register are cleared. The DONE bit is set under the following conditions. • TSTA = 1: (Start request) The DONE bit is set after the start token is transmitted. • TRAN = 1: (Transmission request) The DONE bit is set after the data is transmitted and the acknowledge bit is received. • RECV = 1: (Receive request) The DONE bit is set after the data is received and the acknowledge bit is transmitted. Note that it is impossible to choose the value of the acknowledge bit dynamically based on the data received. • TSTO = 1: (Stop request) The DONE bit is set after the stop token is transmitted onto the I²C bus. • LOST = 1: (Arbitration lost) The DONE bit is also set whenever the I²C bus arbitration is lost. This will clear any outstanding requests in the GRP_I2C_MCTRLW register, i.e., losing bus arbitration clears the TSTA, TRAN, RECV and TSTO bits. 0 = I²C master module busy 1 = I²C master module finished current operation
6	LOST	Arbitration lost. If the previous action was an I²C transmit, the LOST bit is set when the I²C master module loses the arbitration process. The I²C arbitration process can also be lost during a receive action if the master module tries to send a “1” acknowledge and another contending master tries to send a “0” acknowledge. It is also possible to lose arbitration during the transmission of a stop token onto the I²C bus. 0 = Arbitration not lost 1 = Arbitration lost
5	RACK	Receive acknowledge. The RACK bit contains the value of the acknowledge bit received or transmitted during the last transmit / receive. Value of last I²C acknowledge bit received
4	MAST	Master bus. The MAST bit indicates whether the I²C master module is actually the current I²C bus master (1=master). Once this module becomes bus master it will remain so until either the system software tells it to release ownership by writing a “1” to the TSTO field of the GRP_I2C_MCTRLW register or until it releases ownership due to an arbitration loss. 0 = I²C master module not bus master 1 = I²C master module is bus master
3:0	BITS	Number of bits processed. The BITS value indicates the number of the bit currently being processed, as shown in Table 65. Do not assume that an I²C transaction is finished until this field has the value of 9 (“1001”). However, the correct way to detect the completion of an I²C transaction is when the DONE status bit = 1. This field is actually the contents of an internal counter within the I²C master module that counts the number of bits transmitted / received. Thus, in the case of arbitration loss it will indicate at which bit the arbitration was lost. Number of bits processed. See Table 70

Table 65. GRP_I2C_MCTRLR Current Bit

Bits[3:0]	Value	Meaning
0000	0	1st bit in progress
0001	1	2nd bit in progress
...	...	.....
0110	6	7th bit in progress
0111	7	8th bit in progress
1000	8	Ack bit in progress
1001	9	All bits done

Table 66 gives an example of possible settings of the GRP_I2C_MCTRLR bits.

Table 66. Example of Reading GRP_I2C_MCTRLR Register

DONE	LOST	RACK	MAST	Bits				Description
1	0	X	0	X	X	X	X	Idle, not bus master
0	0	X	1	0	1	0	0	Currently bus master Transmit/receive in progress 3 bits done.
0	0	1	1	1	0	0	1	Currently bus master Transmit/receive done If receive, ACK was 1.
0	0	X	0	0	0	0	0	Transmit/receive in progress. No bits done, not bus master, but waiting for bus mastership.
1	1	X	0	0	0	0	1	Not bus master Lost arbitration on bit 2 of last transmit.

I²C Master Control Write Register

name: GRP_I2C_MCTRLW
index: 4Eh, write only
size: 8 bits
function: The GRP_I2C_MCTRLW register controls the operation of the I²C master module. Writing to this register initializes the DONE and BITS fields in the GRP_I2C_MCTRLR register. Initializing the BITS field in the GRP_I2C_MCTRLR register would destroy any I²C bus transaction in progress, so do not write to the GRP_I2C_MCTRLW register while the I²C master module is busy transmitting or receiving on the I²C bus.

It is the responsibility of the system software to ensure that this register is never written to except when the I²C master module is idle (indicated by MAST = 0 or BITS = 1001 in the GRP_I2C_MCTRLR register). All the bits in this register except for TACK are all set to “0” upon reset. This reset initialization places the I²C master module in an idle state immediately after reset. A TACK value of “0” indicates a not-acknowledged state. The bit contents of the GRP_I2C_MCTRLW register are shown in Table 67.

Table 67. I²C Master Write Control Register GRP_I2C_MCTRLW[7:0] Bit Descriptions *(1 of 2)*

Bit	Function	Detail
7	Unused	This bit is currently unused in the I²C master module, however the system software must always write “0” here to allow for possible future expansions.Must write 0 here.
6	400K	400 kHz mode. When set to “1,” the 400 K control bit causes the I²C master module to transmit at the full speed 400 kHz clock rate. When set to “0,” it uses a more conservative 100 kHz clock rate (actually the rate is only 8 0kHz but the timings are set up to satisfy the I²C 100 kHz mode spec). 0 = 100 kHz mode 1 = 400 kHz mode
5(1)	TSTO	Transmit a stop. When set to “1,” the TSTO control bit causes the I²C master module to transmit a stop token onto the I²C bus. If this I²C master module is not bus master, then this control bit will be ignored. Note that unlike TSTA and TRAN, it is not possible to combine TSTO with TRAN or RECV. A separate command access by the system software is required to transmit a stop to the I²C bus. Thus if TRAN, RECV, or TSTA are set to “1” then TSTO must be “0.” 0 = not transmit a stop 1 = transmit a stop
4(1)	TSTA	Transmit a start. When set to “1” the TSTA control bit causes the I²C master module to transmit a start token onto the I²C bus. If the bus is busy, it will wait for the bus to become free before transmitting the start token. If this I²C master module is already bus master, it will transmit the start token immediately. If the TRAN control bit is also set to “1,” the I²C master will transmit the contents of the GRP_I2C_MDATA register immediately after the start token. The combination where both the TSTA and TSTO bits are written as “1” is illegal and the operation of the I²C master module is not defined. 0 = not transmit a start 1 = transmit a start
3	INTE	Interrupt enable. When set to “1,” the INTE control bit causes the interrupt output from the I²C master module to go to “1” whenever the contents of the DONE field goes to “1” (i.e. at the end of a transmit or receive or arbitration loss). If this bit is set to “0,” the system software must service the I²C bus by means of software polling. This action will not result in the loss of any data but may cause the I²C bus stand idle until the system software next polls this module and sets up the next request. The actual interrupt is generated by ANDing this bit with the DONE bit of the GRP_I2C_MCTRLR Read register. 0 = disable interrupts (software polling) 1 = interrupt enable
2	TACK	Acknowledge to transmit. The TACK bit contains the value that the I²C master module will transmit whenever it needs to transmit an acknowledge bit. (i.e., whenever it needs to acknowledge a I²C reception.). Value of last I²C acknowledge bit received
1(1)	RECV	Receive data. When set to “1,” the RECV bit will start the process of receiving the contents of the GRP_I2C_MDATA register from the I2C bus. Note that the I2C master module must already be bus master (MAST = 1 in the GRP_I2C_MCTRLR register). Note also that since the I2C spec does not allow any circumstance where the I2C master follows a start with anything other than a transmit, this module does not support combining the action of the TSTA bit with the RECV bit as it does with TSTA and TRAN to enable both a start and transmit. The combination where both the RECV and TSTA bits are written as “1” is illegal and the operation of the I2C master module is not defined if this happens. 0 = no receive data 1 = receive data

Table 67. I²C Master Write Control Register GRP_I2C_MCTRLW[7:0] Bit Descriptions *(2 of 2)*

Bit	Function	Detail
0(1)	TRAN	Transmit data. When set to "1," the TRAN bit will start the process of transmitting the contents of the GRP_I2C_MDATA register onto the I2C bus. Note that the I2C master module must already be bus master (MAST = 1 in the GRP_I2C_MCTRLR register) or the system software must write a "1" to the TSTA bit at the same time to initiate bus mastership. Note also that the situation where both this bit and the RECV bit are written as "1" is illegal and will produce some undefined action in the I2C master module. 0 = no transmit data 1 = transmit data
Note: (1). Valid until the action requested is finished, then automatically reset		

Table 68 gives an example of possible settings of the GRP_I2C_MCTRLW bits.

Table 68. Example of writing to GRP_I2C_MCTRLW

Unused	400K	TSTO	TSTA	INTE	TACK	RECV	TRAN	Description
0	1	0	0	0	0	0	0	Do nothing.
0	0	0	0	0	0	0	0	Do nothing, in 100 kHz mode.
0	1	0	1	1	0	0	1	Transmit data preceded with start token. Generate interrupt when done.
0	1	0	0	0	1	1	0	Receive data into Acknowledge. No interrupt when finished.
0	1	1	0	0	0	0	0	Transmit a stop token to release I²C bus
1	1	0	0	0	0	0	0	Illegal: Unused bit must not be 1
0	1	0	0	0	0	1	1	Illegal: RECV and TRAN = 1
0	1	1	1	0	0	0	0	Illegal: TSTA and TSTO = 1
0	1	1	0	0	0	1	0	Illegal: TSTA and RECV = 1

Example Bit Setting Sequence

The following is a sample interaction between system software and the I²C master module that results in the I²C master module writing the value 3Fh to the 7-bit address 1Ah on the I²C bus.

1 The system software checks for an idle condition on the I²C bus by reading the GRP_I2C_MCTRLR register, as illustrated below.

	DONE	LOST	RACK	MAST	BITS			
GRP_I2C_MCTRLR	1	X	X	0	1	0	0	1

Note, the I²C master module is idle when MAST = 0. However, this does not mean that the I²C bus is idle; the master module simply does not own the I²C bus and is not transmitting / receiving on it at the moment.

If MAST = 1, the GRP_I2C_MCTRLR register might contain the settings shown below.

	DONE	LOST	RACK	MAST	BITS			
GRP_I2C_MCTRLR	1	0	X	1	1	0	0	1

Here, the I²C master module currently owns the I²C bus (MAST = 1) but is idle at the moment (DONE = 1). Any value other than 1 in DONE would mean that the I²C master module is currently processing a transmit or receive at the moment. The BITS field will indicate which bit it is currently working on: 0000 = MSB, 0111 = LSB, 1000 = ACK bit, etc.

2 The system software can set up the first data to be transmitted by writing 34h to the GRP_I2C_MDATA register. The value 34h is calculated by combining the 7 bit address 1Ah in the 7 MSBs with a 0 (write) in the R/W (LSB) field, as shown below.

	MSB							LSB
GRP_I2C_MDATA	0	0	1	1	0	1	0	0

3 Next the system software can start the transmit process by writing the following data to the GRP_I2C_MCTRLW register, as shown below.

	Unused	400K	TSTO	TSTA	INTE	TACK	RECV	TRAN
GRP_I2C_MCTRLW	0	1	0	1	1	0	0	1

This configuration requests the I²C master module to transmit a start token followed by the data in the GRP_I2C_MDATA register, generating an interrupt when the acknowledge has been received for this transmission. Note that the system software must write “0” to the unused field to allow future extensions.

4 The I²C transmit will now proceed without interaction with the system software. If the system software had not enabled the interrupt, it would need to monitor the progress of the transmit by reading the GRP_I2C_MCTRLR register and taking further action when the transmit finishes. If software polling is used instead of interrupts and if the I²C master module is not serviced for a long period of time after the current transmit / receive is complete, the I²C bus will be idled thereby locking out any other potential I²C bus masters until the system software gives the I²C master module a new command. No data would be lost; just I²C bus bandwidth would be wasted.

Example Bit Setting Sequence

The following is the sequence of events the system software would see if it was monitoring the status of the I²C master module by reading the GRP_I2C_MCTRLR register.

1 Initially the I²C master module would be waiting for an opportunity to transmit the start token (another I²C master module might currently own the bus). While this is the case, the GRP_I2C_MCTRLR register contains the following.

	DONE	LOST	RACK	MAST	BITS			
GRP_I2C_MCTRLR	0	0	X	0	0	0	0	0

2 The "0" in MAST indicates that the I²C master module has not yet gained ownership of the I²C bus. Once the I²C master has ownership (MAST = 1) of the I²C bus and transmitted the start token, the GRP_I2C_MCTRLR register contains the following.

	DONE	LOST	RACK	MAST	BITS			
GRP_I2C_MCTRLR	0	0	X	1	0	0	0	0

3 The "0000" value in BITS indicates that the MSB of the transmit is currently in progress. As more bits are transmitted the value in BITS will increase, as shown below.

	DONE	LOST	RACK	MAST	BITS			
GRP_I2C_MCTRLR	0	0	X	1	0	0	1	1

4 The "0011" value in BITS indicates that 2 bits have been transmitted and that the third bit is in progress. If the I²C master module loses the bus arbitration process to any other master at any time during transmission, the GRP_I2C_MCTRLR register might read as follows.

	DONE	LOST	RACK	MAST	BITS			
GRP_I2C_MCTRLR	1	1	X	0	0	1	0	0

5 The I²C master module lost the bus arbitration process (LOST = 1) while transmitting the 4th bit (BITS = 0100) in the current word. Note that once the I²C master module lost the arbitration process, bus ownership is lost and the MAST bit is reset to "0." Also when the arbitration process ends, the current transmission ends and the DONE bit is set, thereby generating an interrupt if the INTE bit was set. Note that is possible to lose an arbitration while transmitting an ACK bit for a reception so that the following bit pattern is possible.

	DONE	LOST	RACK	MAST	BITS			
GRP_I2C_MCTRLR	1	1	X	0	1	0	0	0

6 Assuming that no arbitration loss occurs, the I²C master module would finish the LSB of the transmission and start receiving the acknowledge bit, as shown below.

	DONE	LOST	RACK	MAST	BITS			
GRP_I2C_MCTRLR	0	0	X	0	1	0	0	0

7 When the acknowledge bit is received, an interrupt will be generated and the transmit is complete.

	DONE	LOST	RACK	MAST	BITS			
GRP_I2C_MCTRLR	1	0	1	1	1	0	0	1

8 The I²C master module is now ready for the next action. The system software would now write the value 3Fh into the GRP_I2C_MDATAW register for the next I²C transmit.

	MSB							LSB
GRP_I2C_MDATA	0	0	1	1	0	1	0	0

9 The system software would again initiate the transmit by writing the following to the GRP_I2C_MCTRLW register.

	Unused	400K	TSTO	TSTA	INTE	TACK	RECV	TRAN
GRP_I2C_MCTRLW	0	1	0	0	1	0	0	1

This configuration is the same as before except the TSTA bit is now “0,” indicating no start token is sent.

10 The system software is interrupted at the end of the transmit; assuming no arbitration loss, the GRP_I2C_MCTRLR register again reads as follows.

	DONE	LOST	RACK	MAST	BITS			
GRP_I2C_MCTRLR	1	0	1	1	1	0	0	1

11 The system software requests the sending of a stop token by writing the following to the GRP_I2C_MCTRLW register.

	Unused	400K	TSTO	TSTA	INTE	TACK	RECV	TRAN
GRP_I2C_MCTRLW	0	1	0	1	0	0	0	0

Note that this time the INTE bit is “0,” disabling interrupts. The I²C master module would proceed to transmit a stop token but would not inform the system software with an interrupt when complete.

I²C Slave Module Software Interface

The software interface to the I²C master block is via 2 VGA graphics extension registers: the GRP_I2C_SCTRL register (index 4Ch) and the GRP_I2C_SDATA register (index 4Dh). The GRP_I2C_SDATA register is read-only (since this module is a slave receiver only and cannot transmit). The GRP_I2C_SCTRL register is both read and write and behaves differently in each case. Consequently, the software interface actually consists of one write-only register and two read-only registers.

In general, it is necessary to access both the GRP_I2C_SCTRL and GRP_I2C_SDATA registers at least once per byte of data received. The order of reading registers is important; the data received in the GRP_I2C_SDATA register should not be considered valid until after the DONE bit has been read as "1" in the GRP_I2C_SCTRLR register. Similarly the GRP_I2C_SCTRLW register should not be written until after the relevant data has been read from the GRP_I2C_SDATA register because writing to this register could initiate another data reception, thereby overwriting the data in the GRP_I2C_SDATA register.

Note that this I²C slave module is capable of losing the arbitration process just like the I²C master module. However this can only happen during transmission of the acknowledge bit. It is the responsibility of the system software to handle any problems like this that occur.

The software interface to the I²C slave module is via the following registers.

I²C Slave Data Register

name: GRP_I2C_SDATA
index: 4Dh, read only
size: 8 bits
function: The GRP_I2C_SDATA register supplies the contents of the input data buffer containing data received over the I²C bus. If this register is read while data reception is still in progress, the data read will be incorrect (not all bits will be shifted-in yet). It is the responsibility of the system software to wait until the data reception on the I²C bus is complete before considering the data read from the GRP_I2C_SDATA register to be correct. Data reception is complete when the DONE bit in the GRP_I2C_SCTRLR register contains a "1."

The bit contents for the GRP_I2C_SDATA register are shown in Table 69.

Table 69. I²C Slave Receive DATA GRP_I2C_SDATA

Bit	Function	Detail
7:0	DATA	Received data byte

I²C Slave Control Read Register

name: GRP_I2C_SCTRLR
index: 4Ch, read only
size: 8 bits

The GRP_I2C_SCTRLR register supplies status information about the progress/completion of any I²C bus transaction. A break must be enabled before the I²C slave can supply status information. The bit contents for the GRP_I2C_SCTRLR register are shown in Table 70. Table 71 provides bit values and meanings. Table 72 gives an example of possible settings of the GRP_I2C_SCTRLR bits.

Table 70. I²C Slave Read Control Register GRP_I2C_SCTRLR[7:0] Bit Descriptions *(1 of 2)*

Bit	Function	Detail
7	DONE	The DONE bit is set to "1" whenever the I²C slave module has finished its current operation and is ready for another request from the system software. It is "0" whenever the I²C slave module is busy servicing a request. This bit can be used by the system software to determine whether the I²C slave module was the source of an interrupt (interrupts enabled) or to check whether the I²C slave has finished its current request (software polling, interrupts disabled). This bit is used in the generation of the I²C interrupt (it is ANDed with the INTE interrupt enable bit) but it is set independently of the value of INTE (interrupt enable). If more than one break request is included in the write to the GRP_I2C_SCTRLW register, the DONE bit will be set when a break conditions occurs. For each of the break conditions, the DONE bit will be set under the following conditions. • BSTA = 1: (Break on start) The DONE bit will be set after the start token is received. • BRCV = 1: (Break on receive) The DONE bit will be set after 8 data bits are received. This request facilitates the system software dynamically, determining the value of the ACK bit to be transmitted based on the data received. • BACK = 1: (Break on acknowledge) The DONE bit will be set after the acknowledge bit has been transmitted. This request facilitates the system software predetermining the value of the ACK bit before any data has been received. • BSTO = 1: (Break on stop) The DONE bit will be set after the stop token is received. Finally, the DONE bit will also be set if the I²C bus arbitration is lost. 0 = module is still busy with last request 1 = finished last request
6	LOST	Arbitration lost. The LOST bit is set to 1 whenever the I²C slave module loses the I²C arbitration process. Note, since this module is only capable of I²C receptions, it can only lose the arbitration process during the transmission of the acknowledge bit 0 = arbitration not lost 1 = arbitration lost
5	STOP	Stop token received. The STOP bit indicates that a stop transaction was received on the I²C bus. This bit is cleared whenever the GRP_I2C_SCTRLW register is written. Therefore if this bit is set, a stop transaction was received since the last GRP_I2C_SCTRLR write. 0 = no stop token received since last read 1 = stop token received since last read

Table 70. I²C Slave Read Control Register GRP_I2C_SCTRLR[7:0] Bit Descriptions *(2 of 2)*

Bit	Function	Detail
4	MAST	Some module is master. A MAST bit value of "1" indicates that some I²C master module is the current bus master. In order to detect a reception or stop transaction on the I²C bus, this bit must be set. If this module is reset (MAST = 0) while some other I²C master module is driving the I²C bus, this module will not be able to receive data or detect a stop transaction until after the current I²C transaction is completed. 0 = bus idle 1 = some module is I²C master
3:0	BITS	Current bit being processed. The BITS value indicates the number of the bit currently being processed during a receive, as shown in Table 71. Do not assume that an I²C reception is finished until this field contains the value of 9 ("1001"). However, the correct way to detect the completion of an I²C transaction when the DONE status bit = 1. This field is actually the contents of an internal counter within the I²C master module that counts the number of bits received. Current bit. See Table 71.

Table 71. GRP_I2C_SCTRLR Current Bit

Bits[3:0]	Value	Meaning
0000	0	1st bit in progress
0001	1	2nd bit in progress
...	...	.....
0110	6	7th bit in progress
0111	7	8th bit in progress
1000	8	Ack bit in progress
1001	9	All bits done

Table 72. Examples of Reading GRP_I2C_SCTRLR[7:0] Bits

DONE	LOST	STOP	MAST	BITS				Description
1	X	0	0	X	X	X	X	Idle, not bus master. This module idle.
0	X	0	0	X	X	X	X	I²C bus idle. This module busy (waiting for start, etc.).
0	0	0	1	0	1	0	0	I²C bus not idle. This module busy (receive in progress, 3 bits done).
1	0	0	1	1	0	0	1	I²C bus not idle. This module done.
1	1	0	1	1	0	0	0	I²C bus not idle. Lost arbitration on ACK bit of last receive
1	0	1	1	X	X	X	X	I²C bus not idle. Stop received since last read.

I²C Slave Control Write Register

name: GRP_I2C_SCTRLW
index: 4Ch, write only
size: 8 bits
function: The GRP_I2C_SCTRLR register controls the operation of the I²C slave module. Writing to this register initializes the DONE and BITS fields in the GRP_I2C_SCTRLR register. Initializing the BITS field in the GRP_I2C_SCTRLR register would destroy any I²C bus transaction in progress, so this register must not be written to while the I²C slave module is busy. It is the responsibility of the system software to ensure that this register is never written to except when the I²C master module is idle (indicated by DONE = 1 in the GRP_I2C_SCTRLR register).

All the bits in this register except for TACK are set to zero upon reset. This reset initialization places the I²C slave module in an idle state immediately after reset. The TACK value of "1" is transmits an ACK bit of "1," which indicates a "not acknowledge" state. With these reset values in the GRP_I2C_SCTRLW register, the I²C slave module will receive data from the I²C bus as it appears but it will never transmit any I²C acknowledge bits and it will never generate an interrupt.

A break (BSTO, BSTA, BRCV, or BACK) must be enabled so that the slave state machine operates properly. A break acts as an interrupt causing the slave to hold SCLK low, stretching the cycles until released. The next start or stop will reset the slave.

The bit contents of the GRP_I2C_SCTRLW register are shown in Table 73.

Table 73. I²C Slave Write Control Register GRP_I2C_SCTRLW[7:0] Bit Descriptions *(1 of 2)*

Bit	Function	Detail
7:6	Reserved	These bits are currently unused in the I²C slave module, however the system software must always write "00" here to allow for possible future expansions. Must write 0's here.
5	BSTO	Break on stop. When set to "1," this control bit causes this module to generate a DONE bit/interrupt after the next stop token reception on the I²C bus. 0 = no break after receive stop 1 = break after receive stop
4	BSTA	Break on start. When set to "1," this control bit causes this module to generate a DONE bit/interrupt after the next start token reception on the I²C bus. 0 = no break after receive start 1 = break after receive start
3	INTE	Interrupt enable. When set to "1," this control bit causes the interrupt output from the I²C slave module to go to "1" whenever the contents of the DONE field goes to "1." If this bit is set to "0," the system software must service this module by means of software polling. This process will not result in the loss of any data but may cause the I²C bus idle until the system software next polls this module and sets up the next request. Note that the actual interrupt is generated by ANDing this bit with the DONE bit of the CTRLR read register. 0 = disable interrupt (software polling) 1 = interrupt enable

Table 73. I²C Slave Write Control Register GRP_I2C_SCTRLW[7:0] Bit Descriptions *(2 of 2)*

Bit	Function	Detail
2	TACK	ACK bit to transmit. This is the value that the I²C slave module will transmit whenever it needs to transmit an acknowledge bit. (i.e. whenever it needs to acknowledge a I²C reception.) 0 = value to transmit for ACK bit 1 = no ACK value to transmit
1	BRCV	Break on receive. When written with "1," this bit will cause this module to generate a DONE bit/interrupt after the next end of receiving 8 bits of data. 0 = no break after receive data 1 = break after receive data
0	BACK	Break on acknowledge. When set to "1," the bit will cause this module to generate a DONE bit/interrupt after the next transmission of the acknowledge bit. 0 = no break after transmit ACK 1 = break after transmit ACK

Determining Break Condition

Sseveral possible break requests can be combined into a single request to the I²C slave module. The interrupt will be generated whenever the first break condition occurs. If more than one break condition is specified, it is usually possible upon receipt of the interrupt to determine which break condition caused the interrupt by reading the GRP_I2C_SCTRLR register.

The BSTA (break on start) request sets MAST = 1 and BITS = 0000 in the GRP_I2C_SCTRLR register.

The BRCV (break on receive) request sets MAST = 1 and BITS = 1001 in the GRP_I2C_SCTRLR register. Data received can be read from GRP_I2C_SDATA, and the acknowledge bit to be transmitted can be controlled by writing to the TACK bit of the GRP_I2C_SCTRLW register.

The BACK (break on acknowledge) request sets MAST = 1 and BITS = 1001 in the GRP_I2C_SCTRLR register. Data received can be read from GRP_I2C_SDATA.

The BSTO (break on stop) request sets STOP = 0 in the GRP_I2C_SCTRLR register, indicating that the interrupt was generated by a stop transaction on the I²C BUS. Using this break condition to generate interrupts which will be used by software is an inherently dangerous procedure. Unlike all the other "break on something" conditions, BSTO *does not* hold the I²C bus to wait for service from the software interrupt service routing (the I²C spec won't allow this). This is especially difficult in a polled (i.e. interrupts disabled) environment because a significant period of time might pass before the I²C module is polled. If this happens and if the interrupt is not serviced in time, the I²C bus could continue off into another transaction, the start of which would be missed by the Bt2166 I²C slave module.

Minimum Stop Transaction

The I²C spec guarantees a minimum of 1.3µs between a stop transaction and a following start transaction on the I²C bus, so if the software interrupt service routing can guarantee to service the interrupt in less than 1.3µs there won't be a problem. If there is any doubt that the software interrupt service routine can service a break on stop interrupt quickly enough then the break on stop condition *must* be combined with another break request (break on start, break on receive or break on acknowledge) to allow the I²C slave module to continue with the next reception without loss of data. When this is done the software interrupt service routine *must* check for the possibility that both break conditions (break on stop and the other break condition) have occurred when servicing the interrupt and take appropriate action if the second break condition has also occurred to avoid missing the second interrupt.

Note also that any of the above break conditions except for BSTO (Break on stop) will cause the I²C slave module to hold the I²C bus idle until the GRP_I2C_SCTRLW register is written again with a new request. It is the responsibility of the system software to ensure that the GRP_I2C_SCTRLW register is written to un-idle the I²C bus within a reasonable period of time. The system software is also responsible for ensuring that the I²C slave module is given a new request sufficiently fast after a break on stop condition. This is necessary because the break on stop does not idle the I²C bus, which could therefore begin another transaction while the I²C slave module is still waiting for system software service, causing the loss of data. Fortunately it is never necessary to detect a break on stop condition in the normal operation of the I²C; the system software could easily ignore the stop token and instead break on start to detect the start of the next I²C bus transaction. Also, the I²C spec requires that the I²C bus remain idle for 1.3µs after the transmission of a stop token, guaranteeing the system software at least 1.3µs to respond to the break on stop request.

Passive I²C Bus Snooper

It is possible to use the I²C slave module as a completely passive I²C bus snooper. Set the GRP_I2C_SCTRLW register to break at the various points of interest in the I²C transactions and set the TACK bit to "1." Data is received as in normal I²C slave module operations, but you can also see what value ACK bit was transmitted by looking at the value of the LOST bit in the GRP_I2C_SCTRLR register. If the ACK bit on the I²C bus was "0," the LOST bit will be set to "1" (since the I²C slave module was trying to send an ACK of 1 and saw an ACK of "0," it sets the LOST bit). Similarly an ACK bit of "1" on the I²C bus will cause the LOST bit in the GRP_I2C_SCTRLR register to remain at zero. To ensure that the I²C slave module is completely passive, all the "break on something" conditions must be serviced immediately. Polling or interrupting is not sufficient because the I²C slave module might halt the I²C bus if it is left waiting for software service for too long. Table 74 gives an example of possible settings of the GRP_I2C_SCTRLW bits.

Table 74. Example of Writing to GRP_I2C_SCTRLW

Unused		BSTO	BSTA	INTE	TACK	BRCV	BACK	Description
0	0	0	0	0	0	0	0	Do nothing
0	0	0	0	1	0	0	0	Pointless: no break requested so no interrupt will be generated
0	0	0	1	1	0	1	0	Interrupt upon reception of a start token or 8-bits of data
0	0	0	0	1	0	0	1	Receive 8 bits of data Acknowledge with ACK=0, then interrupt
0	0	1	1	1	0	0	0	Interrupt upon reception of start or stop token
1	0	0	0	0	0	0	0	Illegal: Unused bit not “0”

Example Bit Settings Sequence

The following is a sample interaction between system software and the I²C slave module during which the I²C slave module receives a series of bytes of data. After the reception of the first two bytes the system software is able to determine that this I²C module is not being addressed by the current I²C transaction and ignore the rest of the transaction.

1 The system software checks whether the I²C bus is idle by reading the GRP_I2C_SCTRLR register.

	DONE	LOST	STOP	MAST	BITS			
GRP_I2C_SCTRLR	1	X	0	0	1	0	0	1

Here it can be seen that the I²C bus is idle (MAST = 0). Note that unlike the MAST bit in the GRP_I2C_MCTRLR register of the I²C master module, this bit indicates whether the I²C bus is idle. In the I²C master module, the MAST bit indicates whether the I²C master module is currently bus master.

If the I²C bus is currently busy (MAST = 1), the GRP_I2C_SCTRLR register might contain the following settings.

	DONE	LOST	STOP	MAST	BITS			
GRP_I2C_SCTRLR	1	0	0	1	1	0	0	1

Any value other than “1” in DONE would mean that the I²C slave module is currently processing a request. During data reception, the BITS field will indicate which bit is currently being received: 0000 = MSB, 0111 = LSB, 1000 = ACK bit being transmitted, etc.

2 The system software can then set up the first request to be processed by writing 0Ah to the GRP_I2C_SCTRLW register. This value will generate an interrupt once 8 bits of data has been received.

	Unused		BSTO	BSTA	INTE	TACK	BRCV	BACK
GRP_I2C_SCTRLW	0	0	0	0	1	0	1	0

3 The I²C slave module can now proceed without interaction with the system software. If the system software had not enabled the interrupt, it would need to monitor the progress of the transmit by reading the GRP_I2C_SCTRLR register and taking further action when the reception finishes. If software polling is used instead of interrupts and if the I²C slave module is not serviced for a long period of time after the current receive is complete, the I²C bus will be idled thereby locking out any other I²C traffic until the I²C slave module interrupt is serviced. Except for the case of detecting stop tokens mentioned above (which can be avoided by correct design of the system software), no data would be lost because of delayed servicing of I²C slave module interrupts / DONE bits - just I²C bus bandwidth would be wasted.

The following is the sequence of events the system software would see if it was monitoring the status of the I²C slave module by reading the GRP_I2C_SCTRLR register.

Initially the I²C slave module would be waiting for an I²C master module to become bus master and transmit a start token. While this was the case, the GRP_I2C_SCTRLR register would contain the following.

	DONE	LOST	STOP	MAST	BITS			
GRP_I2C_SCTRLR	0	0	X	0	X	X	X	X

The “0” in MAST indicates the I²C bus is still idle. Once some I²C master gets ownership of the I²C bus and transmits the start token the GRP_I2C_SCTRLR register would read as follows.

	DONE	LOST	STOP	MAST	BITS			
GRP_I2C_SCTRLR	0	0	X	1	0	0	0	0

The “1” in MAST indicates that some I²C master module now owns the I²C bus. The “0000” value in BITS indicates that the MSB of the transmit is currently in progress. As more bits are received the value in BITS will increase, as shown below.

	DONE	LOST	STOP	MAST	BITS			
GRP_I2C_SCTRLR	0	0	X	1	0	0	1	1

The “0011” value in BITS indicates that 2 bits have been received and that the third bit is in progress. Once all 8 bits have been received, an interrupt will be generated. The I²C slave module will idle the I²C bus while waiting for service by the system software.

4 When responding to the interrupt, the system software can check the progress of the I²C reception by reading the GRP_I2C_SCTRLR register, as shown below.

	DONE	LOST	STOP	MAST	BITS			
GRP_I2C_SCTRLR	1	0	X	1	1	0	0	0

The "1" in the DONE bit indicates the I²C slave module has finished the last request. MAST = 1 indicates the I²C bus is busy. And BITS = 1000 indicates that all 8 bits of data have been received.

The system software can see the data received by reading the GRP_I2C_SDATA register. Then it can setup the I²C slave module to receive the next byte of data by again writing 0Ah to the GRP_I2C_SCTRLW register. This value will transmit an acknowledge bit of "0," then generate an interrupt once another 8 bits of data has been received, as shown below.

	Unused		BSTO	BSTA	INTE	TACK	BRCV	BACK
GRP_I2C_SCTRLW	0	0	0	0	1	0	1	0

Once all 8 bits have been received, an interrupt will be generated. The I²C slave module will again idle the I²C bus while waiting for service by the system software.

5 When responding to this interrupt, the system software can again check the progress of the I²C reception by reading the GRP_I2C_SCTRLR register, as shown below.

	DONE	LOST	STOP	MAST	BITS			
GRP_I2C_SCTRLR	1	0	X	1	1	0	0	0

This indicates that the I²C slave module has done the last request, that the I²C bus is busy and that 8 bits of data has been received.

The system software can see the data received by reading the GRP_I2C_SDATA register. At this stage the system software can determine that the current I²C bus transaction is addressed to some other I²C slave module. It can instruct the I²C slave module to not acknowledge this byte by transmitting an acknowledge value of "1" and to ignore all further I²C activity until the next start token by writing 1Ch to the GRP_I2C_ SCTRLW register, as shown below.

	Unused		BSTO	BSTA	INTE	TACK	BRCV	BACK
GRP_I2C_SCTRLW	0	0	0	1	1	1	0	0

The I²C slave module will ignore all transactions on the I²C bus until it next sees a start token. It will then generate another interrupt and will idle the I²C bus while it waits for service by the system software.

6 The system software can begin again with the new I²C transaction by again writing 0Ah to the GRP_I2C_ SCTRLW register, requesting an interrupt at the end of reception of the 8 bits of data.

GUI ACCELERATOR

GUI Theory Of Operation

To better understand this chapter, first read “Aperture Space Theory of Operation” on page 15 and “Protected Mode Aperture” on page 18 in the “CPU Address Space Apertures” chapter.

The Bt2166 Graphics/Video Controller consists of a programming interface section and an autonomous drawing engine capable of performing BITBLT, character rasterization, and line draw. The command interface to the accelerator is completely queued through a buffer management and command pacing mechanism that requires essentially no reads from the GUI unless expressly transferring a pixmap from the frame buffer to the CPU. The GUI accelerator registers, commands, and parameters are transferred through the lower 5MB of the protected mode aperture. See “BIOS ROM Support” on page 23 for a discussion of the queueing mechanism.

Retained Bitmap Context

One of the acceleration features provided is the caching of bitmap context for the screen as well as for up to three off screen bitmaps. In addition, type information is retained for up to four bitmaps residing in host memory. Source and Destination pixmaps are efficiently identified with a command as selectors of these eight retained pixmap register contexts, see “Bitmap Context Registers” on page 120 for additional discussion on the 8 sets of bitmap context caching registers.

Raster Operation

The Bt2166 GUI has a two-operand raster-op engine which logically combines a source and a destination operand. There are 16 possible combinations; the selection of the desired operation is controlled by a 5-bit field in the GUI configuration register, as shown in the ROP CODE field of Table 93 on page 124. Each command has a COPY and a ROP version. When the COPY version is used, no logical combination is performed; the source is always copied to destination. When the ROP version is used, the logical operation selected by the control value is performed. One of the sixteen operations is copy, so this operation can be performed either way. In addition, the source can be any valid source bitmap: solid colors, patterns, host (mono and color), offscreen, and screen.

Communication of GUI commands, parameters, and register values are paced to the accelerator such that all register modifications which follow a command will be held until that command is completed. For example, the procedure to draw five line segments in five different colors is as follows (assuming the appropriate set-up). Five pairs of foreground register values and their line draw commands are sent. If the accelerator is idle when the first foreground register value is supplied, it will immediately load the foreground color register. When the accelerator receives the first line draw command and its associated parameters, it will go busy and remain that way until completion of the line draw. If the second foreground register value is received while the accelerator is busy, the value is held in the queue until the accelerator goes idle. At this time the new color value is loaded into the foreground register. When the accelerator receives the second line draw command and parameters (either from the CPU or the queue), it will go busy and commence drawing. This sequence is repeated until all five lines are drawn in five different colors. Since each command contains a selector for the bitmap cached context, the lines could also be directed to different destinations with the extents, pitches, and pixel depths automatically managed by the hardware. Notice that any drawing operation can reference host bitmaps as a destination however, only the BLT drawing operator referencing host bitmaps as a destination makes any useful sense.

BLT Transparency

BLT transparency is enabled by setting the transparency control field (bits 13:12) in the GUI Configuration Register (see Table 92 on page 123) and issuing one of the transparency BLT commands. In transparent BLTs, only the destination pixels corresponding to foreground source pixels are modified. In the case of a monochrome source, zeros are considered background pixels and ones are foreground. In the case of a color source, pixels that match the Background Color register are considered background and all other colors are considered foreground pixels. The foreground/background test is made before any raster ops are applied.

GUI Command/Register Map

The GUI command/register map provides access to the GUI accelerator in both a queued and unqueued manner. Refer to ‘‘GUI Commands’’ on page 112 for a detailed functional description of the GUI commands. For GUI register contents, refer to ‘‘GUI Accelerator Registers’’ on page 118 and “GUI Aperture Registers” on page 133.

The command/data interface to the GUI accelerator is mediated by a large queue which is implemented as an on-chip FIFO (the GQUE) with an off-chip overflow FIFO in DRAM (DQUE). The GQUE can hold up to 8 entries; the DQUE can be up to 16 K entries in length. Each queue entry is one dword.

The DQUE is controlled by the GUIREG_FIFO register and its start location and length in DRAM vary based on the driver. To prevent wraparound and corruption of data in the GQUE/DQUE, the driver must limit the addition of items to the queues. The depth of the queues is accessed in the GUI_QUEUE_DEPTH field of the GUIREG_DEPTH register (see Table on page 135).

All GUI commands, parameter values, and write data is queued either on-chip in the GQUE or off-chip in the DQUE. In addition, GUI register values which can be queued will load their specified register “IN-ORDER” with the GUI commands. As shown in Figure 12, there is an on-chip GQUE that gets consumed before the overflow is written to DRAM so that if the GUI accelerator is keeping up with its command arrival rate the DQUE will not be used and the DRAM bandwidth associated with the FIFO will not be consumed. If the CPU attempts to append to the DQUE at the same time that the GUI command processor attempts to retrieve from the DQUE, the CPU is granted access for the higher priority append operation.

Figure 12. Non-queued, Queued And DRAM Queued GUI Interface

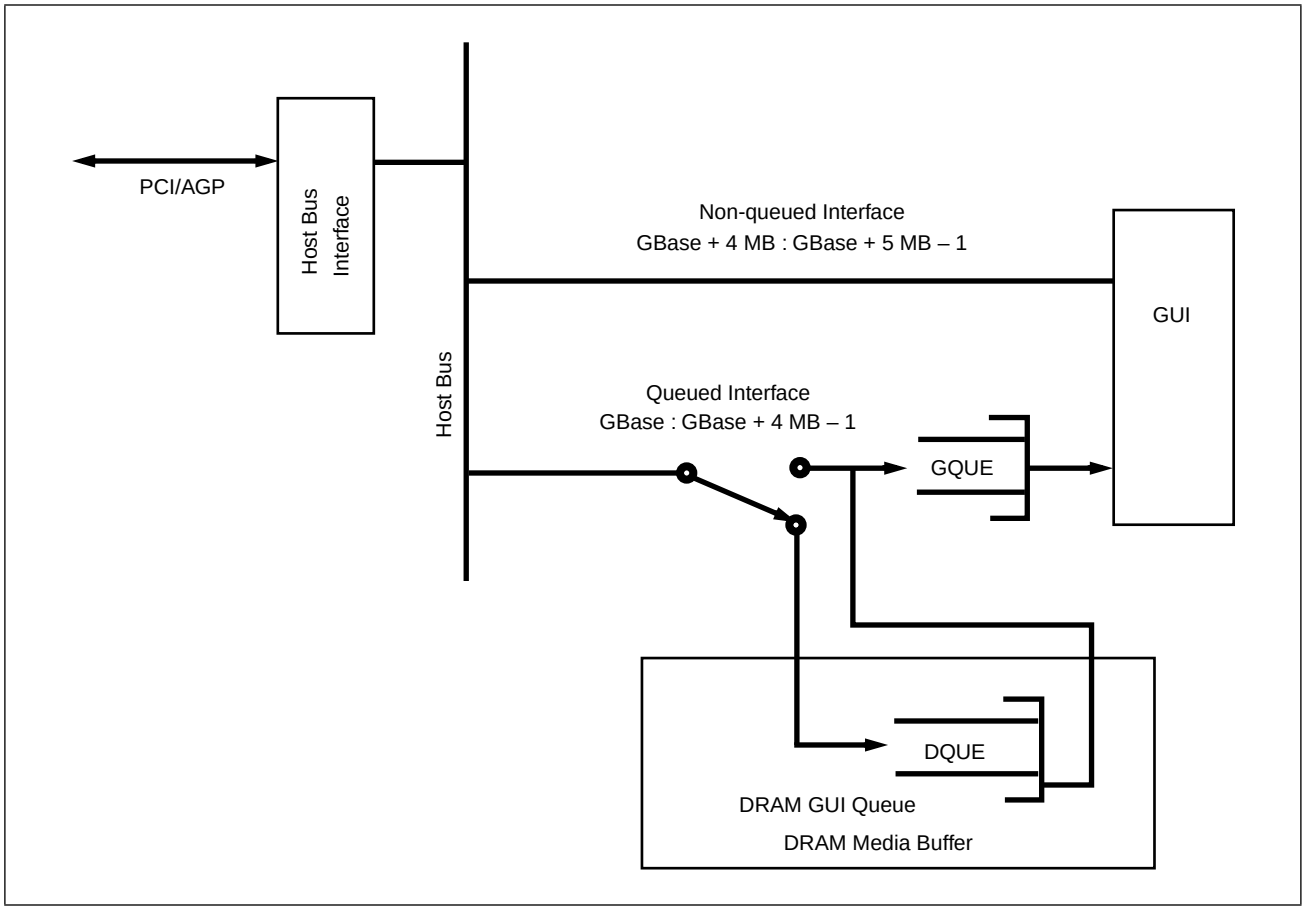


Figure 13 shows the four submaps of the GUI command/register map. Each submap is 1MB and contains 16 blocks, each 64 K long. Submap 4 is a non-queued interface to the GUI accelerator and begins at the GBASE address plus 4 MB. The queued interface map (submaps 3 through 0) begins at GBASE + 0M B. Any command written to the queued interface will use the GQUE on the chip and may use the DQUE in DRAM. All commands to the non-queued interface will "go-around" this FIFO. All reads of any kind go around the FIFO, thus driver software must ensure that the GUI accelerator state is consistent with a read operation by reading the GUIREG_DEPTH to determine whether the GUI is idle. For an overview of the aperture function, refer to ''Aperture Space Theory of Operation'' on page 15.

Figure 13. GUI Command/Register Map

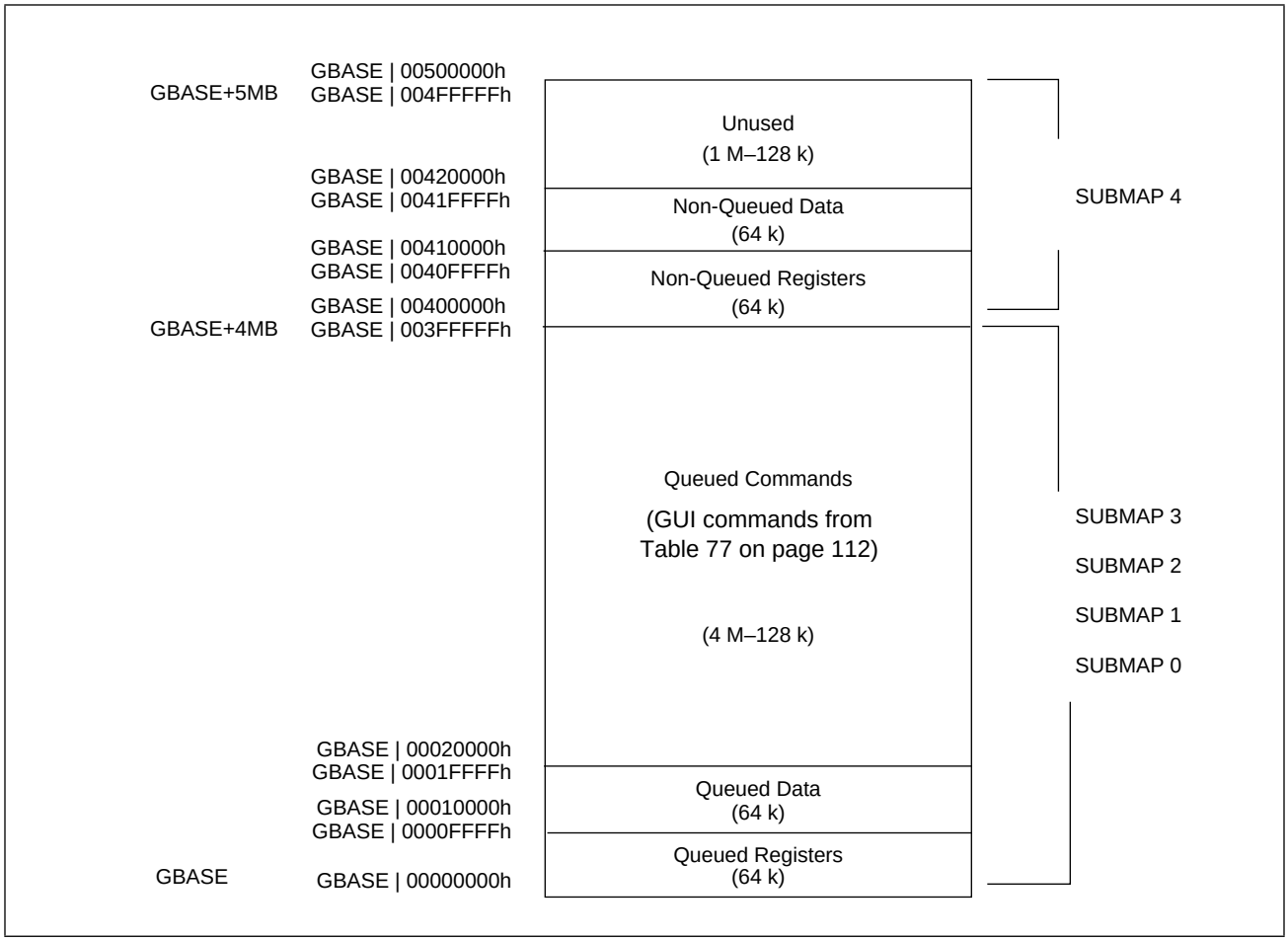


Table 75 indicates the read/write characteristics of the contents of the submaps. Software can read the current depth of the GQUE/DQUE through the non-queued interface to effect efficient management of the GQUE/DQUE. The FIFO is implemented with two address registers and an up/down counter. The address registers implement a ring buffer, wrapping at a power of two boundary, as specified by driver initialization code.

Writing to GUIREG_FIFO will clear the GQUE/DQUE, resetting the counter to zero and reloading the starting address for the DQUE. Notice that each command, register, or data packet in the GQUE/DQUE will contain a 32-bit address followed by zero or more 32-bit data values.

The GQUE/DQUE implements a compression technique to reduce the unnecessary storage of addresses in either the DRAM or on-chip FIFOs. This compression stores the address field of the first word of a command; however, for subsequent parameter dwords no address is stored. Similarly, for host to screen BLTs, the data length is stored in the first location in the FIFO, subsequent data words are stored without inserting address values. For queued register writes, an address is stored for every register data word in the FIFO. Thus an n parameter command would occupy n+1 locations in the FIFO. Similarly, an m dword data transfer would store m+1 locations in the FIFO. In calculating whether a command can be sent to the GUI, one must make sure that there is *at least* n+1 locations remaining in the FIFO by checking the GUIREG_DEPTH register. As reading the GUIREG_DEPTH register is a time consuming proposition, one might keep track of the available space in a variable in the X86 memory.

Table 75. Submap Read/Write Characteristics

SubMap Region	Read	Write
Non-Queued Data	Used in screen-to-host BLTs	Not Allowed
Non-Queued Registers	Allowed	Allowed
Queued Commands	Not Allowed	Allowed
Queued Data	Not Allowed	Used in host-to-screen BLTs
Queued Registers	Not Allowed	Allowed

CPU Addressing of GUI

The 32-bit CPU address controls the GUI queued interface, as shown in Figure 14. CPU address bits 21:2 are used as a selector to determine an action for the memory read or write command when the following signal conditions are met:

- CPU address bits 31:25 match the value of GUI_BASE[31:25] (PM_BASE field).
- GUI_BASE[24] (PM_ENABLE field) is set to 1.
- CPU address bits 24:22 are set to zero.

The meaning of the selector address bits 21:2 is found in Table 76 on page 110, which shows that CPU address bits 21:16 define the GUI command to be performed. Table 76 further shows that there is a different definition for the lower 16 address bits depending on whether the command is a drawing operation, a register access, or a data transfer.

Table 77 on page 112 maps the selector bits 21:16 to a specific GUI command and shows the types of commands supported by the accelerator.

Figure 14. Queued GUI Address Mapping

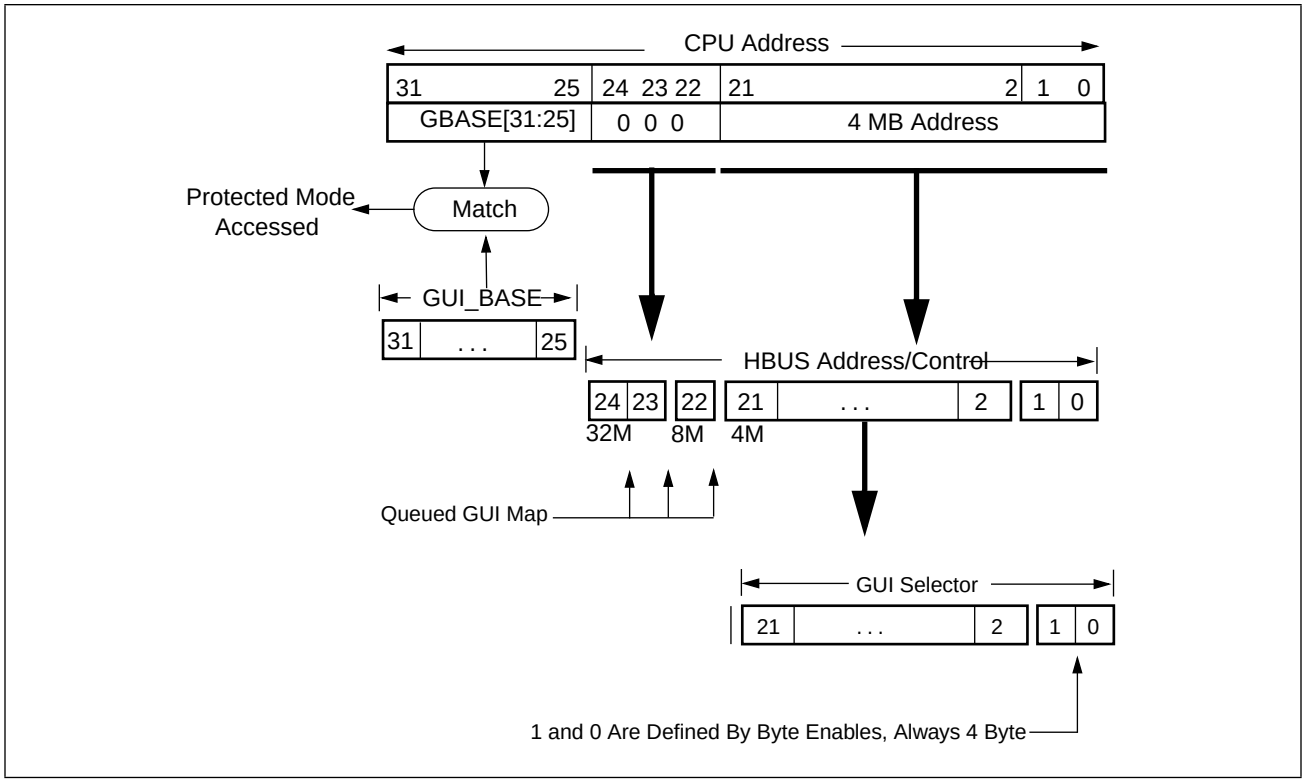


Table 76. Address Fields For GUI Accesses

CPU Address Bits	Field Description		
GUI Selector #	**Register Access**	**CPU-GUI Data Transfers**	**Drawing Operations**
21:16	GUI Command Number = 6'b000000	GUI Command Number = 6'b000001	GUI Command Number > 6'b000001
15:14	Unused	1-16 K Data dword address roll field	Reserved
13:11			Source Bitmap Selector for 2D operations. (1)
10:8			Destination Bitmap Selector, 2D operations(1)
7:5	Register Number		Parameter Count
4:2			Parameter Address Roll Field
1:0	Always set to zero, i.e. a 32-bit transfer		

(1). See Table 86, "Bitmap Context Registers," on page 120

Notice in Table 76 that register accesses occupy the first 64 KB of the GUI access space. This is the queued method of register access described above in which register loads are guaranteed to behave in strict sequential order with respect to the drawing operations. The register access commands are repeated at GBASE+4MB where they are not queued (a register write will occur immediately regardless of whether the accelerator is busy or idle). It is generally too dangerous to access GUI accelerator registers using the non-queued address space while the accelerator is busy.

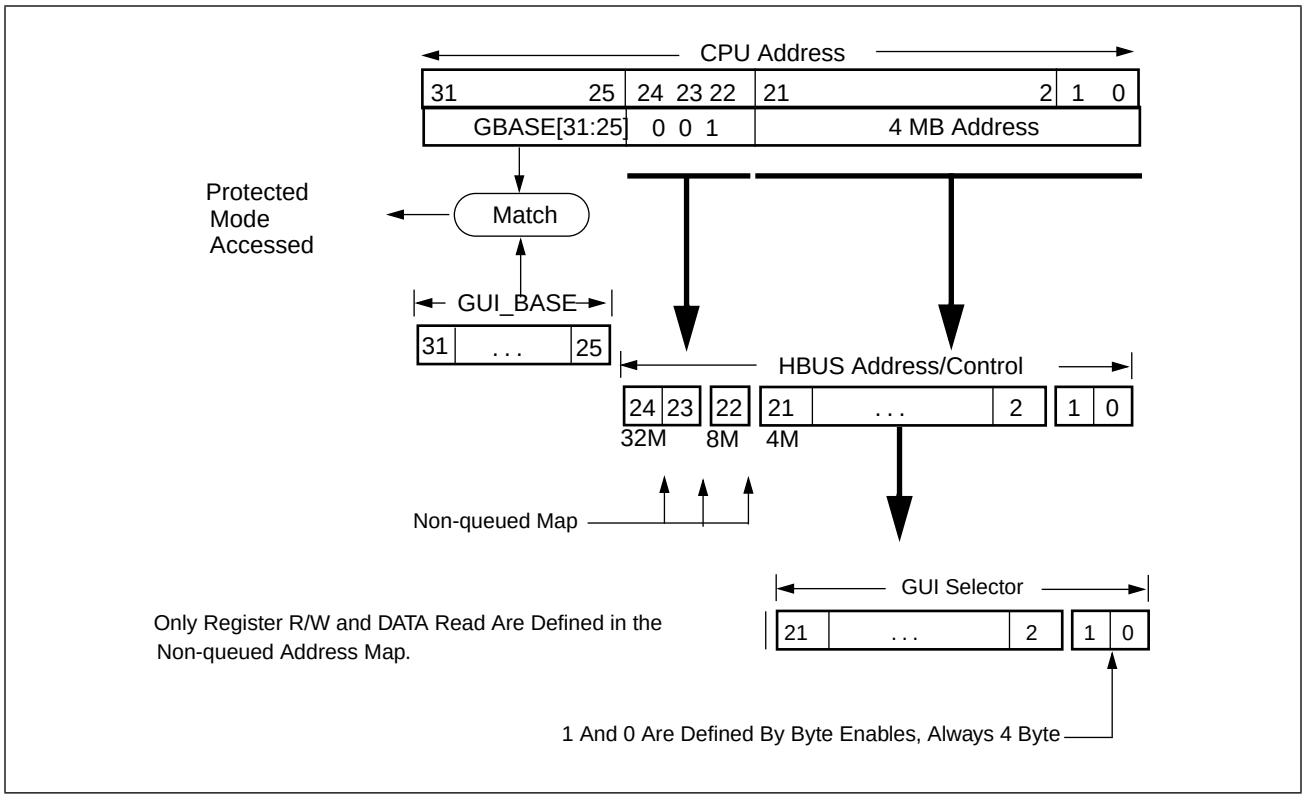
The non-queued method is used for various non-accelerator registers, which are defined in "BIOS ROM Support" on page 23 of the "CPU Address Space Apertures" chapter. The non-queued GUIREG_MBA register controls the type of bit, byte, word, and dword flipping occurs throughout the flippin map of the protected mode aperture and is relatively unrelated to GUI accelerator command queueing. However during driver initialization, this register is loaded with a value that controls whether GUI commands/register/data access are *big endian* or *little endian*.

Shown in Table 76, a bit field is reserved within both drawing operation addresses and data word transfer addresses so that addresses from a REP MOVSD instruction can "roll" through the parameters. Thus you can keep the basic command address in the program, load it into EDI (a 32-bit offset register), then pass out 8K dwords of host-to-screen BLT data using the ADDRESS ROLL field. Thus a drawing command could be started at any one of eight possible addresses corresponding to one of the values of the PARAMETER ADDRESS ROLL FIELD.

Whether a dword is a command or a parameter is *not* encoded in this field. The two are distinguished by their placement in the transfer sequence relative to the previously issued command. Every dword comes over the bus in one cycle and contains an address and a data field. The first parameter (if any) accompanies the first address field. For example, if a drawing command was issued with parameter count 2, the next command of whatever type will appear as the second dword following the drawing command.

Reads from the queued map are unsupported because they have the potential to hang the CPU for very long periods of time. Consequently, to read the final state of the destination XY increment register, the GUI must be completely idle before issuing a non-queued read using the access shown in Figure 15. Because of this hang potential and because a read operation to most of the first 4MB has no meaningful definition, you should always read through the non-queued interface. If reading GUI accelerator registers or screen CPU data, first use the status bits in "GUI FIFO Depth Register" on page 135 to insure that the accelerator is completely idle or actually has read data ready before issuing the first read. Once the accelerator has the first read data word available, driver programs can use REP MOVS instructions to transfer data from screen to host CPU. The accelerator will pace the CPU via the ready line and will, in general, keep up with the CPU demand.

Figure 15. Non-queued GUI Address Mapping



For drawing operations, variable numbers of parameters are permitted (encouraged) using the count mechanism in the command (PCI address bus bits). A zero or one parameter command takes one bus cycle to transfer the command and optional parameter. A two parameter command takes exactly two bus cycles, etc. In addition, a one parameter command takes two word locations in the DRAM FIFO, etc.

When commands are inserted into the DRAM queue software must know how many dwords will be occupied so that the queue depth can be accurately modeled to prevent overflow. Table 77 on page 112 shows how to compute the number of DRAM GUI queue dwords that will be occupied by each command for a given number of parameters or data dwords in the GUI QUESIZE expression.

GUI Commands

The GUI commands are listed in Table 77 and defined in the following pages. The GUI Command Number is determined by CPU address bits [21:16] (see Table 76 on page 110).

Table 77. GUI Commands *(1 of 2)*

GUI Command Number	Command	Raster Ops	No. of Parms.	Queued Access	Non-queued Access	Description
000000	RW Register	n/a	n/a	WRITE	READ WRITE	Read or write one of 64 GUI registers. This command is also available through the non-queued map described above. All writes are queued in sequence with their corresponding GUI commands, e.g. BLT, and no reads are queued. (GUI QUESIZE = 2). See “RWREGISTER Command” on page 114.
000001	RWGUI DATA	n/a	n/a	WRITE	READ	This command allows data to pass to or from the host CPU. On the first cycle of a write the data bus contains a count of the number of dwords to be sent with subsequent RW GUI DATA commands. (GUI QUESIZE = #DWORDS+1). See “RWGUIDATA Command” on page 115.
000010	QMARK	n/a	1	WRITE	NONE	Place a marker in the queue that can be readily recognized by the driver to indicate completion of specific commands. (GUI QUESIZE = #parameters+1)
000011–100000	Reserved	n/a	n/a	NONE	NONE	
100001–100010	Reserved	n/a	n/a	NONE	NONE	
100011	TEXTBLT	COPY	0–4	WRITE	NONE	(GUI QUESIZE = #parameters +1)
100100–100110	Reserved	n/a	n/a	NONE	NONE	
100111	TEXTBLT	Copy Transparent	0–4	WRITE	NONE	(GUI QUESIZE = #parameters +1)
101000–101010	Reserved	n/a	n/a	NONE	NONE	
101011	TEXTBLT	ROP	0-4	WRITE	NONE	(GUI QUESIZE = #parameters +1)
101100–101110	Reserved	n/a	n/a	NONE	NONE	

Table 77. GUI Commands *(2 of 2)*

GUI Command Number	Command	Raster Ops	No. of Parms.	Queued Access	Non-queued Access	Description
101111	TEXTBLT	ROP Trans-parent	0–4	WRITE	NONE	(GUI QUESIZE = #parameters +1)
110000	Reserved	n/a	n/a	NONE	NONE	
110001	TRAP (reserved)	COPY				
110010	LINE	Copy	0–2	WRITE	NONE	(GUI QUESIZE = #parameters +1)
110011	BITBLT	Copy	0–3	WRITE	NONE	(GUI QUESIZE = #parameters+1)
110100	Reserved	n/a	n/a	NONE	NONE	
110101	TRAP (reserved)	Copy Trans-parent				
110110	LINE	Copy Trans-parent	0–2	WRITE	NONE	(GUI QUESIZE = #parameters +1)
110111	BITBLT	Copy Trans-parent	0–3	WRITE	NONE	(GUI QUESIZE = #parameters+1)
111000	Reserved	n/a	n/a	NONE	NONE	
111001	TRAP (Reserved)	ROP				
111010	LINE	ROP	0–2	WRITE	NONE	(GUI QUESIZE = #parameters +1)
111011	BITBLT	ROP	0–3	WRITE	NONE	(GUI QUESIZE = #parameters+1)
111100	Reserved	n/a	n/a	NONE	NONE	
111101	TRAP (reserved)	ROP Trans-parent				
111110	LINE	ROP Trans-parent	0–2	WRITE	NONE	(GUI QUESIZE = #parameters +1)
111111	BITBLT	ROP Trans parent	0–3	WRITE	NONE	(GUI QUESIZE = #parameters+1)

RWREGISTER Command

If we define a C constant to reference the queued register map, it could be coded:

```
#define QUEUED_REG 0x00000000
```

To access the GUIREG_MBA register through the queued register interface we could form an address such as:

```
((unsigned long *)(GBASE| QUEUED_REG | GUIREG_MBA))
```

We can also define a C constant to reference the non-queued register map:

```
#define NON_QUEUED_REG 0x00400000
```

To access the GUIREG_MBA register through the non-queued register interface we could form an address such as:

```
(unsigned long *)(GBASE| NON_QUEUED_REG | GUIREG_MBA)
```

GUI accelerator drawing operators BITBLT , RWGUIDATA, and LINE are discussed below.

BITBLT Command

Many different BLT operations can be very simply encoded with the Bt2166 BLT mechanism. Because the source and destination bitmap context selector within the command can point to either host or media buffer bitmaps, all operations can be specified quite succinctly. A BLT command can be accompanied by up to three parameters so that a maximum BLT command is defined in Table 78.

Table 78. BLT Command

Address Bus	Data Bus	GUI Accelerator Value
BLT	DEST XY ADDRESS	COMMAND + P0
Parameter count and address roll field are don't cares	XY EXTENTS	P1
Parameter count and address roll field are don't cares	SOURCE XY ADDRESS	P2

The destination XY address specifies the XY offset into the bitmap of the first pixel to be written. The XY extents specify the width and height of the rectangle to be written.

The source XY address specifies the XY offset into the source bitmap. In the case of a bitmap in CPU memory, the X value is used to determine the alignment relative to the destination array. The Y value is not used. In the case of a pattern as the source, the X and Y values provide the starting offset into the pattern.

Patterns are stored in packed dword format, e.g. a monochrome 8x8 pattern is stored as two consecutive dwords in the frame buffer. The lowest-addressed element of each line of the pattern is the first pixel displayed for each line of the pattern. In other words, the right-most or least significant element becomes the left-most pixel.

Patterns are aligned relative to the offset frame buffer origin rather than to the starting destination address. The effect is that no matter where a pattern fill is started, the same pixels are drawn in the destination area as would have been if the pattern drawn had started at pixel (0,0) and continued to the destination area. The source (X,Y) address for a pattern fill is always taken to mean the (X,Y) address of the first pixel of the first line of the complete pattern.

When the source of the BLT is a host bit map, an RWGUIDATA command (GUI command number 000001) must immediately follow the BLT command which transfers up to 16 k dwords to the GUI accelerator.

It should be noted that whenever an XY address is specified in a parameter or register it has the format shown in Table 79.

Table 79. XY Address Format

Bit	Field Name	Description
31:28	Reserved	Must be set to zero
27:16	Y ADDRESS	Vertical index or row number
15:12	Reserved	Must be set to zero
11:0	X ADDRESS	Horizontal index or column number

TEXTBLT Command

The TEXTBLT command operates exactly like the BITBLT command except when a patterned source is specified. Instead of aligning relative to the frame buffer origin, the TEXTBLT aligns to the destination address. The intent is to take advantage of the packed nature of patterns in specifying a text character or glyph, thereby reducing the number of source fetches required to render the glyph. Another difference is that the TEXTBLT command does not support drawing multiple copies of the same glyph with a single BLT command.

RWGUIDATA Command

This command is used for host-to-screen or screen-to-host transfers only. Table 80 shows the form that the RWGUIDATA command takes for an N dword transfer.

Table 80. RWGUIDATA Command

Queue Dword No.	Bitmap Dword No.	Dword Use
1	N/A	RWGUIDATA_LENGTH (host-to-screen only)
2	1	Bitmap dword 1
3	2	Bitmap dword 2
...	...	
N	N-1	Bitmap dword N-1
N+1	N	Bitmap dword N

Table 81 defines the RWGUIDATA_LENGTH variable. It is a programming error to supply a RWGUIDATA_LENGTH word for screen-to-host transfers. After sending the command, the host simply reads the required number of dwords from anywhere within the 16 K dword RWGUIDATA address roll space.

Table 81. RWGUIDATA_LENGTH

Bit	Description
31:14	Unused
13:0	Number of dwords per scan line in source (HOST) multiplied by number of scan lines in source (HOST) bitmap (total count -1)

An alternative to the RWGUIDATA command is to transfer data host-to-screen and screen-to-host using PCI bus mastering. This can be accomplished by setting the appropriate control bit in the source or destination bit map context register. Simultaneous transfers from and to main system memory with bus mastering is not supported.

LINE Draw Command

The LINE command, defined in Table 82, supports both single and polyline capabilities.

Table 82. LINE Command

Address Bus	Data Bus	GUI Accelerator Value
LINE	Ending XY point	COMMAND +P0
Parameter count and address roll field are don't cares	Starting XY point	P1

Both end points of the line are specified for single line commands. Only one end point is specified for polylines, where the starting point is the previous endpoint.

The line control register ("LINE Control Register" on page 129) contains control bits that determine whether the first or last pixels of a line segment will be skipped or drawn. Also, this register determines several "styles" of Bresenham line algorithms, including whether the line can be drawn reversibly.

If no parameters are sent, i.e. parameter count field = 0 in the command, then the line drawing registers are assumed to have already been set up (end points, error terms, etc.). The Ending XY (P0) unconditionally gets copied to Starting XY (P1) after every line draw.

QMARK Command

The QMARK command causes a marker to be placed in the GUI command queue so that the driver software can determine when a given command has flushed through the queue. In addition, the QMARK command can, in effect, pace GUI command execution. When the WAIT4_VERTICAL bit is set to one, GUI command execution is paused until the start of vertical retrace. Groups of commands for multiple frames can be issued together, but are executed at only one group per frame. Table 83 lists QMARK command, Table 84 gives the bit definition of Parameter 0 (P0).

Table 83. QMARK Command

Address Bus	Data Bus	GUI Acceleration Value
QMARK Command	32-bit marker value	P0

Table 84. QMARK Command P0 Bit Definition

Bit	Name	Description
31:9	Reserved	
8	WAIT4_VERTICAL	1 = Wait for vertical retrace 0 = Go immediately to next command
7:0	MARKER_VALUE	Copy into marker field of GUI Command Register

The GIUREG_DEPTH[24] status bit (Table 106, “GUI FIFO Depth Register (GUIREG_DEPTH),” on page 135) indicates when the GUI is waiting for vertical retrace.

From QMARK’s point of view, vertical retrace begins at the end of vertical display enable. The complement of the display enable signal used in the GUI can be read in the VGA INP_STAT_M register, bit zero (see "Input Status #1 Register," Table 58 on page 77). If a QMARK command is encountered during a vertical retrace, the GUI pauses until the start of the next vertical retrace period.

GUI Accelerator Registers

This section defines the GUI accelerator registers. Table 85 lists the GUI registers, addresses, and description or reference location. An address in the GUI Register is generated by ORing the GBASE value with the register address to create the offset address (for more information, see "CPU Addressing of GUI" on page 109).

Note: The GUI enable bit in configuration register GRP_CFG4[0] (see Table 26 on page 53) must be set before these registers can be accessed.

Table 85. GUI Register *(1 of 2)*

Queued Address (hex)	Non-queued Address (hex)	Name	Description
00000000	00400000	PARAMETER0	First parm of a command. This is a register RW view of the parameter which is normally sent as part of a command not as a register RW.
00000004	00400004	PARAMETER1	Second parameter of a command.
00000008	00400008	PARAMETER2	Third parameter of a command.
0000000C : 00000018	0040000C : 00400018	Reserved	Reserved
0000001C	0000001C	GUIREG_CMD	"GUI Command Register" on page 123
00000020	00400020	GUIFG_COLOR	"Foreground Color Register" on page 126
00000024	00400024	GUIBG_COLOR	"Background Color Register" on page 126
00000028	00400028	GUILINE_PATT	"Line Pattern Register" on page 130
0000002C	0040002C	GUIDST_XY_INC	"Destination XY Increment Register" on page 127
00000030	00400030	GUIREG_CFG	"GUI Configuration Register" on page 123
00000034	00400034	GUIBLT_CONTROL	"Blit Control (Direction) Register" on page 128
00000038	00400038	GUILINE_CONTROL	"LINE Control Register" on page 129
0000003C	0000003C	Reserved	Reserved
00000040	00400040	BMAP0_TYPE	"Bitmap Context Registers" on page 120
00000044	00400044	BMAP0_PITCH	"Bitmap Context Registers" on page 120
00000048	00400048	BMAP1_TYPE	"Bitmap Context Registers" on page 120
0000004C	0040004C	BMAP1_PITCH	"Bitmap Context Registers" on page 120
00000050	00400050	BMAP2_TYPE	"Bitmap Context Registers" on page 120
00000054	00400054	BMAP2_PITCH	"Bitmap Context Registers" on page 120
00000058	00400058	BMAP3_TYPE	"Bitmap Context Registers" on page 120

Table 85. GUI Register *(2 of 2)*

Queued Address (hex)	Non-queued Address (hex)	Name	Description
0000005C	0040005c	BMAP3_PITCH	“Bitmap Context Registers” on page 120
00000060	00400060	BMAP4_TYPE	“Bitmap Context Registers” on page 120
00000064	00400064	BMAP4_PITCH	“Bitmap Context Registers” on page 120
00000068	00400068	BMAP5_TYPE	“Bitmap Context Registers” on page 120
0000006C	0040006C	BMAP5_PITCH	“Bitmap Context Registers” on page 120
00000070	00400070	BMAP6_TYPE	“Bitmap Context Registers” on page 120
00000074	00400074	BMAP6_PITCH	“Bitmap Context Registers” on page 120
00000078	00400078	BMAP7_TYPE	“Bitmap Context Registers” on page 120
0000007C	0040007C	BMAP7_PITCH	“Bitmap Context Registers” on page 120
00000080	00400080	GUI_BRES0_ADDR	“Bresenham 0, Address Register” on page 130
00000084	00400084	GUI_BRES0_ERR	“Bresenham 0, Error Register” on page 131
00000088	00400088	GUI_BRES0_K1	“Bresenham 0, K1 Register” on page 131
0000008C	0040008C	GUI_BRES0_K2	“Bresenham 0, K2 Register” on page 131
00000090	00400090	GUI_BRES0_INC1	“Bresenham 0, Increment 1 Register” on page 132
00000094	00400094	GUI_BRES0_INC2	“Bresenham 0, Increment 1 Register” on page 132
00000098	00400098	GUI_BRES0_LENGTH	“Bresenham 0, Length Register” on page 132
0000009C : 000000A8	0040009C : 004000A8		Reserved for Bresenham 1
000000AC : 000000EC	000000AC : 000000EC	Reserved	
don’t use	004000F0	GUIREG_MBA	“MBA Control Register” on page 136
don’t use	004000F4	GUIREG_DEPTH	See “GUI FIFO Depth Register” on page 135.
don’t use	004000F8	GUIREG_FIFO	See “GUI FIFO Register” on page 133.
don’t use	004000FC	Reserved	

Bitmap Context Registers

There are eight pairs of 32-bit bitmap registers: BMAP[7:0]_TYPE, BMAP[7:0]_PITCH. Registers BMAP[7:0]_TYPE contain both type and offset information. Registers BMAP[7:0]_PITCH contain pitch information.

The contents of the bitmap context registers are given in Table 86. Table 87 defines the OFFSET field. Table 88 defines the TYPE field. The Bitmap Register Offset Field is given in Table 89. Table 90 defines the PITCH field.

The GUI selector number is determined by the CPU address. Bits [13:11] and [10:8] specify either a source or destination bitmap selector, respectively (see Table 76 on page 110).

Table 86. Bitmap Context Registers

GUI Selector #	32-Bit Register	Upper 12 Bits [31:20]	Lower 20 Bits [19:0]
000	BMAP0_TYPE	[31:24] TYPE [23:20] Reserved	OFFSET
	BMAP0_PITCH	[31:14] Reserved	[13:00] PITCH
001	BMAP1_TYPE	[31:24] TYPE [23:20] Reserved	OFFSET
	BMAP1_PITCH	[31:14] Reserved	[13:00] PITCH
010	BMAP2_TYPE	[31:24] TYPE [23:20] Reserved	OFFSET
	BMAP2_PITCH	[31:14] Reserved	[13:00] PITCH
011	BMAP3_TYPE	[31:24] TYPE [23:20] Reserved	OFFSET
	BMAP3_PITCH	[31:30] Reserved [29:20] UPPER OFFSET	[13:00] PITCH [19:14] Reserved
100	BMAP4_TYPE	[31:24] TYPE [23:20] Reserved	Reserved
	BMAP4_PITCH	Reserved	Reserved
101	BMAP5_TYPE	[31:24] TYPE [23:20] Reserved	Reserved
	BMAP5_PITCH	Reserved	Reserved
110	BMAP6_TYPE	[31:24] TYPE [23:20] Reserved	Reserved
	BMAP6_PITCH	Reserved	Reserved
111	BMAP7_TYPE	[31:24] TYPE [23:20] Reserved	Reserved
	BMAP7_PITCH	Reserved	Reserved

Table 87. Bitmap Register OFFSET Field for DRAM

Register	Bit	Description
BMAP_PITCH	31:20	Reserved, set to zero
BMAP_TYPE	19:0	Offset value. The bitmap's starting location in DRAM (dword address)

As shown in Table 88, the bitmap register type uses the saved context to tell the GUI accelerator how to treat the specified source or destination bitmap.

Table 88. Bitmap Register TYPE Field

Bit	Description
7	Reserved
6	PCI bus mastered data; only valid in bitmap context 3 (BMAP3_PITCH). 1 = PCI bus mastering 0 = Use RWGUIDATA command
5:4	Pattern Size 00 = reserved 01 = 8x8 10 =16x16 11 = 32x32
3	1 = Solid fill with background or foreground color. See "Foreground Color Register" on page 126.
2	1 = Pattern
1	1 = Host
0	1 = Mono

Table 89. Bitmap Register OFFSET Field for PCI

Register	Bits	Description
BMAP_PITCH	31:30	Reserved
BMAP_PITCH	29:20	32-bit PCI offset (dword address, i.e. minus two LS bits) [29:20]
BMAP_TYPE	19:0	PCI offset [19:0]

When using PCI bus mastering, BITMAP_TYPE field bit 6 must be set to one and BITMAP_TYPE field bit 1 must be set to one. In this case, the bit map will be taken from the PCI bus at a location specified by perform XY to linear conversion using the 32 offset field as a base address. To use the GUI queue with RWGUIDATA commands, BITMAP_TYPE field bit 6 must be set to zero and BITMAP_TYPE field bit 1 must set to one.

Note: Only bitmap context 3 (BMAP3_PITCH) supports the upper offset bits required for PCI bus mastering.

Table 90. Bitmap Register PITCH Field

Bit	Description
31:30	Reserved
29:20	Upper offset
19:14	Reserved
13:0	Pitch pixel value For example: 0010 0000 0000 = 200h = 512 pixel 0100 0000 0000 = 400h = 1024 pixel Maximum pixel value is 3FFF = 16,383.

Table 91 shows an example of how the bitmap context registers might be allocated for one instant in the life of a Windows driver. Notice contexts 7:4 are either allocated to host bitmaps or constant source (SRC) data patterns. To change the drawing context, all the driver has to do is send bitmap context register changes down the DRAM queued register write path within the sequence of drawing operations. Suppose the sequence looks like: BLT#1, queued write to context 1 offset, queued write to context 1 pitch, BLT#2. Suppose that both BLT#1 and BLT#2 use bitmap context 1 as their destinations (DST). BLT#1 could be directed to a screen window while BLT#2 could be directed to an offscreen bitmap. Since drawing context is retained and since context changing register writes can be enqueued in sequence there is virtually no need for the Windows driver to synchronize to the actual current drawing state of the GUI accelerator. Therefore the driver can proceed far ahead of the hardware and allow full overlap of application execution with drawing setup and rasterization.

Table 91. Example Bitmap Context Register Allocation

Bit Map #	Selector #	Typical Allocation
0	000	Mono Pattern
1	001	Color Pattern
2	010	Screen Copy
3	011	Spare
4	100	Mono Host*(1)*
5	101	Color Host*(1)*
6	110	Solid Fill*(1)*
7	111	Free*(1)*
Note: (1). Reserved for host and solid fill operations		

If the source (SRC) bitmap has the host bit set but also has the solid fill bit set or if a ROP is specified that doesn't use SRC data, then it is a programming error to supply host data. Unused SRC data will halt all data and command flow through the GUI FIFO, requiring a GUI_RESET to recover.

GUI Command Register

name: GUIREG_CMD
address: GBASE | QUEUED_REG | 0000001Ch, read/write
address: GBASE | NON_QUEUED_REG | 0000001Ch,read/write
size: 32 bits
function: The GUI Command Register is primarily a read-only diagnostic register used to determine the most recently issued GUI command. The complete contents of the GUI Command Register are shown in Table 92.

Table 92. GUI Command Register (GUIREG_CMD)

Bit	Description
31:24	Read-only. Most recent value written by a QMARK command.
23	Read-only. 1 = GUI is waiting for vertical retrace to begin before processing any further GUI commands. 0 = GUI is not waiting for vertical retrace to process commands
22	Reserved
21:16	GUI command number (see Table 77 on page 112)
15:14	Reserved
13:11	Source bitmap context select
10:8	Destination bitmap context select
7:5	Parameter count - number of parameters for last command
4:0	Reserved

GUI Configuration Register

name: GUIREG_CFG
address: GBASE | QUEUED_REG | 00000030h, read/write
address: GBASE | NON_QUEUED_REG | 00000030h, read/write
size: 32 bits
function: The GUI Configuration Register controls the overall context of the GUI accelerator determining such functions as pixel color depth for line or blt, raster operation to be used, etc. The contents of the GUI configuration register are shown in Table 93. At reset all bits are cleared except [18:16], as indicated. Bits [31:27] and [11:9] enable experimental acceleration features and will be removed at a later date.

Table 93. GUI Configuration Register (GUIREG_CFG) *(1 of 2)*

Bit	Description
31	Fetch ahead enable. 1 = Enable fetch for next line source data.
30	Smart FIFO enable. 1 = Enable memory requests in anticipation of FIFOs filling/emptying instead of waiting for actual events.
29	Large store enable. 1 = Enable 32-dword memory request size. 0 = Limit memory request size to 16 dwords
28	Large fetch enable. 1 = Enable 32-dword memory request size. 0 = Limit memory request size to 16 dwords.
27	Force pattern reload. 1 = Reload the pattern on every blit. 0 = Retain pattern in source FIFO if address is same as last.
26	Block write enable. Increases performance by enabling the GUI to send block write requests instead of normal write requests to the memory controller for solid fill operations. This feature must not be enabled by software unless memory implementing block write is present, otherwise data will be rendered incorrectly. The GUI automatically issues color write cycles as required for block write requests. This bit must be set in order for transparent block writes (bit15, page 124) to work. 1 = enable block writes 0 = disable block writes
25	1 = Enable compare of scan line count.
24:20	Reserved
19	Byte 3 write control. 1 = preserve alpha channel 0 = modify alpha channel The Byte 3 write control bit should only be set in 32bpp mode to preserve the “alpha channel” data in byte 3 of 32-bit pixels. When set to 1, the bit disables byte enable 3 on all dwords the GUI writes to DRAM. To ensure predictable results, be sure that the GUI is idle and the result FIFO is empty (GUIREG_DEPTH[19:0] =0) before the byte 3 write control bit is toggled.
18:16	Color depth. 000 = 1bpp (mono) reserved 001 = reserved 010 = 8bpp (reset value) 011 = reserved 100 = 16 bpp (0565) 101 = 16 bpp (1555) 110 = 24 bpp (Note: no 24bpp pattern support) 111 = 32 bpp
15	Transparent block write enable. 1 = enable transparent block writes 0 = disable transparent block writes Increases performance on mono-transparent operations by enabling the GUI to send block write requests instead of normal write requests to the memory controller. Bit 26 (page 124) must also be set in order for this operation to work.

Table 93. GUI Configuration Register (GUIREG_CFG) *(2 of 2)*

Bit	Description
14	Alpha transparency control. 1 = enable alpha transparency 0 = disable alpha transparency Controls handling of the alpha information for color transparent operations (affects only 32bpp and 1555 16bpp color depths). If this bit is set, the alpha content is ignored when detecting transparent pixels. If this bit is disabled, the alpha content must exactly match (on all bits) the corresponding data in the background color register (page 126) in order for a pixel to be considered transparent.
13:12	Transparency control 00 = no transparency 01 = source transparency 10 = reserved 11 = reserved
11	Large source FIFO enable 1 = enable 32-qword FIFO 0 = enable 16-qword FIFO
10	Large destination FIFO enable 1 = enable 32-qword FIFO 0 = enable 16-qword FIFO
9	Large result FIFO enable 1 = enable 32-dword FIFO 0 = enable 16-dword FIFO
8	MONO_FLIP_ENABLE 0 = pass monochrome bitmap bytes through unscathed 1 = reverse the bits within each byte in monochrome bitmaps before alignment
7:5	Reserved
4:0	ROP CODE 00h = S-->D — source replaces dest 01h = S AND D --> D — AND source with dest 02h = S AND $\overline{D}$ --> D — AND source with not dest 03h = 0 --> D — 0s replace dest 04h = S OR $\overline{D}$ --> D — OR source with NOT dest 05h = S XNOR D --> D — XNOR source with dest 06h = $\overline{D}$ --> D — invert dest 07h = S NOR D --> D — NOR source with dest 08h = S OR D --> D — OR source with dest 09h = D --> D — no change 0Ah = S XOR D --> D — XOR source with dest 0Bh = $\overline{S}$ AND D --> D — AND NOT source with dest 0Ch = 1 --> D — 1s replace dest 0Dh = $\overline{S}$ OR D --> D — OR NOT source with dest 0Eh = S NAND D ->D — NAND source with dest 0Fh = $\overline{S}$ --> D — NOT source replaces dest 10h:1Fh—Reserved

Foreground Color Register

name: GUIFG_COLOR
address: GBASE | QUEUED_REG | 00000020h, read/write
address: GBASE | NON_QUEUED_REG | 00000020h, read/write
size: 32 bits
function: The Foreground Color Register is used when expanding monochrome source data into a color destination. A “one” bit of monochrome source data indicates that the foreground color should be written to the corresponding destination pixel. The foreground color is also used for the upper half of 64-bit solid fill operations, which are specified by issuing a transparent bit blit command that selects a solid fill source type. The background color is used for the lower half of 64-bit solid fills. In all modes where the color depth is less than 32 bits per pixel, color values must be replicated to fill the foreground color register. For example, in 8bpp mode, the 8 bit color index to be used for foreground write operations must be replicated across all four bytes of the foreground register. In 24 bpp mode, byte3 must be set equal to byte0. The bit contents of the Foreground Color Register are shown in Table 94.

Table 94. Foreground Color Register

Bit	Description
31:0	Foreground color

Background Color Register

name: GUIBG_COLOR
address: GBASE | QUEUED_REG | 00000024h, read/write
address: GBASE | NON_QUEUED_REG | 00000024h, read/write
size: 32 bits
function: This value is used for writing pixels for solid fills or when in one of the patterned modes, and the pattern bit is a zero. It is also used to detect transparency for non-monochrome sources. If color depth is less than 32 bpp, then color values must replicated in the same manner as for the foreground register. The bit contents of the Background Color Register are shown in Table 95.

Table 95. Background Color Register

Bit	Description
31:0	Background color value

Destination XY Increment Register

name: GUIDST_XY_INC
address: GBASE | QUEUED_REG | 0000002Ch, read/write
address: GBASE | NON_QUEUED_REG | 0000002Ch, read/write
size: 32 bits
function: This value is used to increment the internal destination XY register after the completion of a BLT command, allowing a blt offset to be specified. The bit contents of the destination XY increment register are shown in Table 96.

Table 96. Destination XY Increment

Bit	Description
31:28	y sign extended
27:16	Signed y increment value
15:12	x sign extended
11:0	Signed x increment value

Blit Control (Direction) Register

name: GUIBLT_CONTROL
address: GBASE | QUEUED_REG | 00000034h, read/write
address: GBASE | NON_QUEUED_REG | 00000034h, read/write
size: 32 bits
function: If blit backwards is selected, the DST and SRC Y both point to the last line of the respective fields. Also both Y pointers are decremented throughout the operation. There is no hardware support for blit backwards in the X direction.

The scan line counter enables synchronization of blit operations and CRT screen refresh, especially when blitting from off-screen to on-screen memory. When GUIREG_CFG[25] is enabled (Table 93 on page 124), the GUI compares the value of the scan line count in GUIBLT_CONTROL with the NXT_LINE value reported from the Display Refresh. Before processing each line of a blit operation, the GUI makes sure that the scan line count is less than NXT_LINE, thus avoiding tearing effects by ensuring that the blit operation never gets ahead of the electron beam in the CRT. The scan line count in GUIBLT_CONTROL is incremented as each scan line of a blit operation is completed and must be initialized with the desired value between blit commands. The bit contents of the BLT Control Register are shown in Table 97.

Table 97. BLT Control Register

Bit	Description
31:28	Reserved
27:16	Scan line counter
15:01	Reserved
0	Direction, cleared after each operation 1 = backward 0 = forward

LINE Control Register

name: GUILINE_CONTROL
address: GBASE | QUEUED_REG | 00000038h, read/write
address: GBASE | NON_QUEUED_REG | 00000038h, read/write
size: 32 bits
function: The LINE Control Register defines how several line drawing situations are handled. A single pixel line [(X0,Y0)=(X1,Y1)] with either SKIP_FIRST or SKIP_LAST is effectively a NO-OP (Bresenham registers are modified but not frame buffer, nor is the line pattern stepped). The bit contents of the LINE Control Register are shown in Table 98.

Zero-parameter line draws are not affected by CALC_ONLY, allowing a sequence of (1) DRAW_LINE (CALC_ONLY) (2) modify Bresenham registers (3) DRAW_LINE (no parameters).

Table 98. Line Control Register

Bit	Field	Description
31:11	Unused	
10	XMAJOR(1)	1 = X axis is the major axis 0 = Y axis is the major axis
9	SIGN_DX(1)	1 = DX is negative 0 = DX is positive
8	SIGN_DY(1)	1 = DY is negative 0 = DY is positive
7:6	Unused	
5	X_REVERSIBLE(2)	1 = Draw line reversibly, X-Windows Style. When NT_REVERSIBLE is set, it overrides X_REVERSIBLE.
4	CALC_ONLY	1 = Calculate all constants and addresses but don't draw
3	SKIP_FIRST	1 = Do not draw the first pixel on a line segment
2	SKIP_LAST	1 = Do not draw the last pixel on a line segment.
1	INVERT_ZERO_TEST(2)	1 = For zero-error-term cases, inverts the decision to step in the major and minor or just major axis for NT_REVERSIBLE, X_REVERSIBLE, and non-reversible line draws 0 = Not enable invert zero test
0	NT_REVERSIBLE(2)	1 = Draw line reversibly, NT style. For NT lines always choose lower X or Y value. When set, overrides X_REVERSIBLE. 0 = Disable NT style. If both NT_REVERSIBLE and X_REVERSIBLE are set to 0, non-reversible lines are drawn
Note: (1). Bits 10:8 control the effective drawing octant for zero-parameter lines. These bits are ignored and overwritten when the line draw engine calculates the Breshenham parameters. (2). If bits 0, 1, and 5 are set to "0," the line engine will step both axes when the error term is zero.		

Line Pattern Register

name: GUILINE_PATT
address: GBASE | QUEUED_REG | 00000028h, read/write
address: GBASE | NON_QUEUED_REG | 00000028h, read/write
size: 32 bits
function: This register is used to control the style of lines drawn. When bit 0 is set to 1, the current pixel is drawn in foreground color. When bit 0 is set to 0, the current pixel is drawn in background color. The contents of the line pattern register are rotated one bit position to the right for every line pixel drawn; i.e., bits 31:1 become bits 30:0 and bit 0 becomes bit 31.

The line pattern register is not reset between line draw operations, so it will track through polyline operations. Once the GUI is enabled (GRP_CFG4[0] = 1), the pattern is reinitialized only by writing to this pattern register. The contents of this register are ignored unless a source bitmap with the pattern bit is selected (see Table 88 on page 121).

Bresenham 0, Address Register

name: GUI_BRES0_ADDR
address: GBASE | QUEUED_REG | 00000080h, read/write
address: GBASE | NON_QUEUED_REG | 00000080h, read/write
size: 32 bits
function: The value in this register is normally computed from the XY to linear address logic which converts incoming XY addresses to linear frame buffer addresses. The bit contents are shown in Table 99.

Table 99. Bresenham 0, Address Register

Bit	Description
31:25	Reserved, set to zero
24:5	dword address
4:3	byte address
2:0	Reserved, set to zero
Note: 8-bit address requires byte, 16-bit address requires a word, 24-bit address requires byte, 32-bit address requires dword	

Bresenham 0, Error Register

name: GUI_BRES0_ERR
address: GBASE | QUEUED_REG | 00000084h, read/write
address: GBASE | NON_QUEUED_REG | 00000084h, read/write
size: 32 bits
function: The value in this register is normally computed from the line command setup logic, unless a zero parameter line command was issued. The value is also computed in the CALC_ONLY mode of line draw. The value is set as required for the Bresenham line draw algorithm. The bit contents are shown in Table 100.

Table 100. Bresenham 0, Error Register

Bit	Description
31:0	Error value

Bresenham 0, K1 Register

name: GUI_BRES0_K1
address: GBASE | QUEUED_REG | 00000088h, read/write
address: GBASE | NON_QUEUED_REG | 00000088h, read/write
size: 32 bits
function: The value in this register is normally computed from the line command setup logic, unless a zero parameter line command was issued. The value is also computed in the CALC_ONLY mode of line draw. The value is set as required for the Bresenham line draw algorithm. The bit contents are shown in Table 101.

Table 101. Bresenham 0, Constant K1

Bit	Description
31:0	K1 value

Bresenham 0, K2 Register

name: GUI_BRES0_K2
address: GBASE | QUEUED_REG | 0000008Ch, read/write
address: GBASE | NON_QUEUED_REG | 0000008Ch, read/write
size: 32 bits
function: The value in this register is normally computed from the line command setup logic, unless a zero parameter line command was issued. The value is also computed in the CALC_ONLY mode of line draw. The value is set as required for the Bresenham line draw algorithm. The bit contents are shown in Table 102.

Table 102. Bresenham 0, Constant K2

Bit	Description
31:0	K2 value

Bresenham 0, Increment 1 Register

name: GUI_BRES0_INC1
address: GBASE | QUEUED_REG | 00000090h, read/write
address: GBASE | NON_QUEUED_REG | 00000090h, read/write
size: 32 bits
function: The value in this register is normally computed from the line command setup logic, unless a zero parameter line command was issued. The value is also computed in the CALC_ONLY mode of line draw. The value is set as required for the Bresenham line draw algorithm. The bit contents are shown in Table 103.

Bresenham 0, Increment 2 Register

name: GUI_BRES0_INC2
address: GBASE | QUEUED_REG | 00000094h, read/write
address: GBASE | NON_QUEUED_REG | 00000094h, read/write
size: 32 bits
function: The value in this register is normally computed from the line command setup logic, unless a zero parameter line command was issued. The value is also computed in the CALC_ONLY mode of line draw. The value is set as required for the Bresenham line draw algorithm. The bit contents are shown in Table 103.

Table 103. Bresenham 0, Increment 1 and 2 Registers

Bit	Description
31:18	Sign extend bit[17] (writes bit 17 into this field)
17:3	Increment value
2:0	Set to zero

Bresenham 0, Length Register

name: GUI_BRES0_LENGTH
address: GBASE | QUEUED_REG | 00000098h, read/write
address: GBASE | NON_QUEUED_REG | 00000098h, read/write
size: 32 bits
function: The value in this register is normally computed from the line command setup logic, unless a zero parameter line command was issued. The value is also computed in the CALC_ONLY mode of line draw. The value is set as required for the Bresenham line draw algorithm. The bit contents are shown in Table 104.

Table 104. Bresenham 0 Length Register

Bit	Description
31:12	Reserved, set to zero
11:0	Line pixel count - 1, e.g. (1,1)->(3,3) yields a length register value of 2.

GUI Aperture Registers

This section defines the GUI aperture space registers. Refer to the "CPU Address Space Apertures" chapter starting on page 15 for GUI aperture theory of operation, addressing, and commands.

GUI FIFO Register

name: GUIREG_FIFO
address: GBASE | 004000F8h, read/write
size: 32 bits
function: This register is accessed *only* via the non-queued interface and controls various aspects of the DQUE, including its location and size. Table 105 shows the bit contents of the GUI FIFO register. Upon reset, all bits are cleared. A write to this register will reset all GUI queues to an initial empty state, except when bit [30] is set to allow masking.

Note: The GUI enable bit in configuration register (GRP_CFG4[0]) must be set before this register can be accessed (see Table 26 on page 53).

Table 105. GUI FIFO Register (GUIREG_FIFO) *(1 of 2)*

Bit	Name	Description
31	DQUE_ENABLE	1 = Enable DQUE. Resets status bits in GUIREG_DEPTH. 0 = Disable DQUE. All queued writes go through the GQUE. If the GQUE gets full, additional writes to the GQUE will stall until an empty slot is available. In some circumstances, the stall could be lengthy.
30	EN_FIFO_WR	Write-only. Always reads back as zero. For diagnostics only. 1 = Enable a write only to GUI_STEP and GUI_FREEZE bits without affecting the other bits 0 = Enable write to DQUE_BASE or DQUE_ADDR_MASK. Resets GQUE and DQUE to an initial empty condition.
29:28	Reserved	Must be zero
27:16	DQUE_BASE	Base address of DQUE. Actual address has 8 zeroes appended to the LSB of this address to form an MBUS address.

Table 105. GUI FIFO Register (GUIREG_FIFO) *(2 of 2)*

Bit	Name	Description
15	GUI_FREEZE	1 = Stop sending commands from GQUE to the GUI for diagnostic purposes. EN_FIFO_WR must be set. 0 = Normal operation
14	GUI_STEP	0->1 = step next command from GQUE to the GUI into registers. Must only be used when GUI_FREEZE is a one.
13:9	Reserved	Must be zero
8:7	FAST_WR_SIZE	Controls size of burst writes from the Direct and Flip-pin maps to the frame buffer. 00 = disabled 01 = bursts of 2 dwords 10 = bursts of 4 dwords 11 = bursts of 8 dwords
6	Reserved	Must be zero
5:0	DQUE_ADDR_ MASK	Bits 5:0 mask bits 15:10 of all DQUE address calculations. 0 means that the corresponding address is not modified as the DQUE address pointer is incremented. Thus GUI DQUEs are allocated in powers of 2 dwords (starting at 256). A 6 bit mask value of 01h enables a 512 dword DQUE while a value of 0Fh enables 4096 dword DQUEs. DQUE start addresses are aligned on power of two boundaries at least as large as the DQUE size. dwords — dwords 000000 - 256 — 001111 - 4 k 000001 - 512 — 011111 - 8 k 000011 - 1 k — 111111 - 16 k 000111 - 2 k

GUI FIFO Depth Register

name: GUIREG_DEPTH
address: GBASE | 004000F4h, read only
size: 32 bits
function: This register is *only* accessed via the non-queued interface. The purpose of this register is to provide information about the GUI queues, such as the depth (total number of entries) in the GUI queues (GQUE and DQUE) as well as giving software a way to determine whether the GUI subsystem has completed all requested actions. Table 106 shows the bit contents of the GUIREG_DEPTH register. Upon reset, all bits are cleared.

NOTE

The GUI enable bit in configuration register (GRP_CFG4[0]) must be set before this register can be accessed (see Table 26 on page 53).

Table 106. GUI FIFO Depth Register (GUIREG_DEPTH)

Bit	Name	Description
31:29	Reserved	
28:25	GUI_GQUE_ DEPTH	Unsigned up-down counter used to track GUI GQUE FIFO depth of usage. Depth counter = 0 for empty GQUE. Count increments for each word written to GQUE; decrements for each word taken from GQUE and sent to GUI.
24	GUIQ_WAIT4_ VERTICAL	1 = GUI GQUE stalled waiting for vertical retrace while in a QMARK command (page 117).
23	GUI_BLT_ UNDERFLOW	1 = GUI screen --> host BLT underflow error
22	Reserved	Must be zero.
21	GUI_BLT_ DATA_RQD	1 = GUI host --> screen write data required
20	GUI_BLT_ DATA_RDY	1 = GUI screen --> host read data ready
19	GUI_BUSY	1 = GUI command processor busy
18	RESULT_FIFO_ BUSY	1 = GUI result FIFO non-empty
17	GUI_QUEUE_ BUSY	1 = GUI queue processor busy
16	GUI_FIFO_ NON_EMPTY	1 = GUI GQUE FIFO non-empty
15:0	GUI_DQUE_ DEPTH	Unsigned up-down counter used to track GUI DQUE FIFO depth of usage. Depth counter = 0 for empty DQUE. Count increments for each word written to FIFO; decrements for each word taken from FIFO and sent to GQUE. To find the total combined depth of the GQUE and DQUE, add this value to the GUI_GQUE_DEPTH value.

MBA Control Register

name: GUIREG_MBA
address: GBASE | 004000F0h, read/write
size: 32 bits
function: The Media Buffer Aperture (MBA) Control Register directs flippin map flip controls and command map flip controls. Table 107 defines the bit contents of the GUI MBA register. Upon reset, all bits are cleared.

NOTE

The GUI enable bit in configuration register (GRP_CFG4[0]) must be set before this register can be accessed (see Table 26 on page 53).

Table 107. GUI MBA Control Register

Bit	Field	Description
31:22	Reserved	Must be 0
21	FLIPPIN_WORD	Flippin Map word flip control, 1=flip
20	FLIPPIN_BYTE	Flippin Map byte flip control, 1=flip
19	FLIPPIN_BIT	Flippin Map bit flip control, 1=flip
18	Reserved	Must be 0
17	CMD_WORD	Command map word flip control, 1=flip
16	CMD_BYTE	Command map byte flip control, 1=flip
15	CMD_BIT	Command map bit flip control, 1=flip
14:0	Reserved	Must be 0

INTERRUPT REGISTER DEFINITIONS

Interrupt Register Definitions

In order for system BIOS to successfully route the PCI_INTA pin from the card, PCI function 1 must be enabled via GRP_CFG7[6] (see Table 23 on page 52). If PCI function 1 is disabled, all interrupts must remain masked (INT_SR[15:0]=15'b0).

Interrupt Status Register

name: INT_SR
address: GBASE | 00500000h, read/write
size: 16 bits
function: On read, this register indicates the active/inactive state for each interrupt device. The bits of this register are ORed together to drive the PCI_INTA pin to the bus. The interrupt state register is bit-wise ANDed with the Interrupt Mask Register to generate this status register.

Bit definitions for interrupt status, interrupt mask, and interrupt state are as shown in Table 108.

A type of "edge" means that the interrupt occurs on the leading edge of the event and will remain until cleared by writing to either the interrupt status or state register. A type of "level" means that the value of the interrupt is passed through (using the mask) and must be reset via some other mechanism, e.g., the VGA vertical interval interrupt mechanism. If type is "edge," writing a 1 to individual bits of the status register will reset that particular interrupt. A type of "pgm" indicates a programmable interrupt.

An interrupt can be forced by writing the numeric value of the bit position of the interrupt type to the Interrupt Firing Register (see page 138). Writing the reserved bit positions will have no effect.

Table 108. Interrupt Status, Mask, and State Register Bits (INT_SR)

Bit	Field	Type	Description
15		N/A	Reserved
14	VIP_FIFO_INT	Edge	VIP FIFO transfer complete interrupt
13	VIP_EXT_INT	pgm	VIP external interrupt source
12		Edge	DREF Y position
11	FRAME_SYNC	Edge	This interrupt is generated by the rising edge of the frame sync signal at the start of every screen refresh.
10	IIC_INT	Level	I^2C controller IRQ
9	VIDEO_CAPTURE_INT	Edge	Video input subsystem, structure requesting IRQ
8	VGA_RETRACE_INT	Level	VGA or DREF requesting IRQ
7:0	Reserved	N/A	Reserved

Interrupt Mask Register

name: INT_MR
address: GBASE | 00500002h, read/write
size: 16 bits
function: This read/write register contains a bit-for-bit mask for each interrupt source. 1 = enable the corresponding interrupt. 0 = mask off the corresponding interrupt type. The contents of this register are bit-wise ANDed with the Interrupt State Register bits to form the interrupt status register and to generate a request to the CPU bus.

Bit definitions for interrupt mask, interrupt status, and interrupt state are as listed in Table 108. Reserved interrupts cannot be masked.

Interrupt State Register

name: INT_STR
address: GBASE | 00500004h, read/write
size: 16 bits
function: This read/write register contains the storage elements for interrupt types that are flagged as events. Interrupt types that are flagged as levels do not actually have a storage element here but are passed through as if they were in this register. The content of this register is bit-wise ANDed with the Interrupt Mask Register bits to present an interrupt request to the CPU bus.

Bit definitions for interrupt state, interrupt status, and interrupt mask are as listed in Table 108.

Interrupt Firing Register

name: INT_FR
address: GBASE | 005003FCh, read/write
size: 8 bits
function: This register can be used to force an interrupt for diagnostic or device initialization purposes. Upon reset, all bits are set to zero.

Table 109 defines the bit contents of the Interrupt Firing Register.

Table 109. Interrupt Firing Register

Bit	Description
7:4	Reserved
3:0	Generate an interrupt for the source that corresponds to the bit number in this register (0-Fh) — refer to Table 108 on page 138 for bit positions.

CONFIGURATION AND PLL REGISTER DEFINITIONS

Configuration and PLL Registers

General Configuration Register

name: GEN_CFG
address: GBASE | 00500010h, read/write
size: 32 bits
function: The general configuration register shown in Table 110 provides configuration information on the ROM, BIOS scratch, monitor sense, sync on green, and VGA sleep.

For information on the ROM configuration portion (bits 17:8) of the register, refer to the section "ROM Interface" on page 338.

Table 110. ROM, BIOS, Monitor, VGA Sleep, Sync Configuration *(1 of 2)*

Bit	Description
31:24	BIOS Scratch
23:18	Reserved
17:16	ROM_ENABLE — sets the Bt2166 memory clock enable interval. Reset value: 01 00 = 1 memory clock interval 01 = 2 memory clock intervals 10 = 3 memory clock intervals 11 = 4 memory clock intervals
15:11	ROM_WAIT — sets the Bt2166 memory clock wait interval from 1 to 32 clocks (00000 = 1 clock **...** 11111 = 32 clocks). Reset value: 00111 (8 clocks)
10:8	ROM_HOLD — sets the Bt2166 memory clock hold interval from 1 to 8 clocks (000 = 1 clock **...** 111 = 8 clocks). Reset value: 001 (2 clocks)

Table 110. ROM, BIOS, Monitor, VGA Sleep, Sync Configuration *(2 of 2)*

Bit	Description
7:3	Reserved
2	Monitor Sense — powers-on monitor sense circuitry in the DAC and allows the monitor sense to be read from GEN_STATUS[0], see Table 112 on page 145. This bit should be turned on only long enough for the read. 1 = monitor sense on 0 = monitor sense off (default)
1	Sync on green — generates a composite sync which is combined with the green levels onto the green signal to the monitor 1 = enabled 0 = disabled (default)
0	VGA Sleep — VGA state at power on 1 = enabled (sleep) 0 = disabled, awake (default)

Memory Configuration Register

name: MEM_CFG
address: GBASE | 00500014h, read/write
size: 32 bits
function: The synchronous memory configuration register shown in Table 111 allows for optimum performance with different memories.

Table 111. Synchronous Memory Configuration Register (MEM_CFG) *(1 of 2)*

Bit #	Name	Default Value	Description
27:26	Reserved	0	Reserved, set to zero
25	tRAS_EXTEND	1	This bit extends the tRC and tRAS by one cycle for normal memory cycles (non-refresh). This bit is independent of RFSH_CYCLE_CNT which is used only for refresh cycles. Reset value: 1.
24	RESET_CFG	0	This bit causes the memory controller to perform another initialization sequence similar to power on reset. When the memory controller determines that this bit has changed state (from 0 to a 1 or 1 to a 0), it will reinitialize the memory and rewrite the sync memory internal config register via the MRS command (see the documentation that accompanied the specific memory device). The initialization sequence will be done at the next refresh request. The memory refresh MUST be enabled in order to use this function Reset value: 0.
23	Reserved	0	This bit is reserved for future use. Set to 0.
22	RAS_PRE_CNT	1	Configures the RAS precharge time (in cycles) used in the memory controller. Reset value: 1. 1 = programs the memory controller for a RAS precharge of 3 cycles 0 = indicates a 2 cycle RAS precharge will be performed
21	RFSH_CYCLE_CNT	1	This bit controls the number of clocks that a refresh cycle will take to complete a refresh operation (tRC). This affects only refresh cycles. Reset value: 1. 1 = configures the memory controller for a refresh cycle count of 10 cycles 0 = configures the memory controller for a refresh cycle count of 9
20	RCD_CNT	1	This bit controls the number of cycles of the RAS to CAS delay. Reset value: 1. 1 = configures the memory controller for a 3 cycle delay 0 = Not applicable in Bt2166 Rev A.
19	RBSTP_LATENCY	1	This bit controls the manner in which the BSTP (Burst Stop) command operates with read operations to the memory. Reset value: 1. 1 = used for memory devices that tristate their output 3 clocks after receiving the BSTP command. 0 = used for memory devices that tristate their output 2 clocks after the clock when a valid BSTP command is received.
18	WBSTP_LATENCY	0	This bit controls the manner in which the BSTP (Burst Stop) command operates with write operations to the memory. Reset value: 0. 1 = used for memory devices that allow writing to the memory the same cycle as the BSTP command and terminate the write burst the next cycle. 0 = used for memory devices that prevent a write the same cycle as the BSTP command.

Table 111. Synchronous Memory Configuration Register (MEM_CFG) *(2 of 2)*

Bit #	Name	Default Value	Description
17	ADR_MODE	0	This bit controls the manner in which the linear address is mapped to the physical address of the memory array. Reset value: 0. 1 = indicates large page mode, which configures the multiple banks of the synchronous memories to provide a large page mode at the expense of having multiple pages open with different row address. 0 = indicates normal page mode, allowing for multiple open pages with different row addresses for each bank. See “Memory Map Information” on page 340.
16:15	RANK1_SIZE	00	These 2 bits encode the size of the memory attached to rank1 (controlled by $\overline{\text{SDCS}}$[1]). Reset value: 00 00 = 2Mbytes 01 = 4Mbytes 10 = 8Mbytes 11 = reserved
14:13	RANK0_SIZE	00	These 2 bits encode the size of the memory attached to rank0 (controlled by $\overline{\text{SDCS}}$[0]). Reset value: 00 00 = 2Mbytes 01 = 4Mbytes 10 = 8Mbytes 11 = reserved
12	RANK1_TYPE	0	This bit indicates the type of memory attached to RANK1. Reset value: 0 1 = indicates attached memory is SGRAM 0 = indicates that the attached memory is SDRAM
11	RANK0_TYPE	0	This bit indicates the type of memory attached to RANK0. Reset value: 0 1 = indicates attached memory is SGRAM 0 = indicates that the attached memory is SDRAM
10:0	SYNCMEM_CFG	‘hex:037	These bits contain the data value that will be written to the internal configuration registers within the synchronous memory during a MRS memory access. The value of these bits are muxed to the memory address bus during a MRS command. Reset value: 037h

General Status Register

name: GEN_STATUS
address: GBASE | 00500018h, read only
size: 32 bits
function: The general status register shown in Table 112 contains information on the monitor sense.

Table 112. General Status Register (GEN_STATUS)

Bit	Description
31:1	Reserved
0	Reports whether the monitor is detected. GEN_CFG[2] (Table 110 on page 141) must be enabled in order to read this bit. 1 = Monitor detected 0 = Monitor not detected

Memory Clock PLL Register

name: MEMCLK_PLL
address: GBASE | 00500020h, read/write
size: 32 bits
function: The memory clock drives the synchronous memory interface. The PLL is accurate within the range of 50Mhz to 175MHz. L is hard-wired to 1. The reset clock value is 65.9 Mhz. See Table 113.

The formula for determining the memory clock frequency is:

$$14.31818 * M/(N*2^{L})$$

For additional information, refer to ‘‘GUI, MEM, and SYS Clock PLL Rate Selection’’ on page 355.

Table 113. Memory Clock PLL Register (MEMCLK_PLL)

Bit	Description
31:12	Reserved
11:8	Value of N. Range: 4 to 15. Reset value: 5h
7	Reserved
6:0	Value of M. Range: 21 to 63. Reset value: 46d

Pixel Clock PLL Register

name: PIXCLK_PLL
address: GBASE | 00500024h, read/write
size: 32 bits
function: The pixel clock is used by the DREF module to drive the display. The PLL is accurate within the range of 50Mhz to 175MHz. The reset clock value is 97.8Mhz. See Table 114.

The formula for determining the pixel clock frequency is:

$14.31818 * M/(N*2^{L})$

For additional information, refer to ‘‘Pixel Clock PLL Rate Selection’’ on page 358.

Table 114. Pixel Clock PLL Register (PIXCLK_PLL)

Bit	Description
31:18	Reserved
17:16	Value of L. Range: 0 to 3. Reset value: 0
11:8	Value of N. Range: 4 to 15. Reset value: 6d
7	Reserved
6:0	Value of M. Range: 21 to 84. Reset value: 41d

GUI Clock PLL Register

name: GUICLK_PLL
address: GBASE | 00500028h, read/write
size: 32 bits
function: The GUI clock drives the GUI module and the 3D module. The PLL is accurate within the range of 50Mhz to 175MHz. L is hard-wired to 1. The reset clock value is 66 Mhz. See Table 115.

The formula for determining the GUI clock frequency is:

$14.31818 * M/(N*2^{L})$

For additional information, refer to ‘‘GUI, MEM, and SYS Clock PLL Rate Selection’’ on page 355.

Table 115. GUI Clock PLL Register (GUICLK_PLL)

Bit	Description
31:12	Reserved
11:8	Value of N. Range: 4 to 15. Reset value: 5 d
7	Reserved
6:0	Value of M. Range: 21 to 63. Reset value: 46d

System Clock PLL Register

name: SYSCLK_PLL
address: GBASE | 0050002Ch, read/write
size: 32 bits
function: The system clock is the general internal clock that drives the VGA module, video module, and other internal buffer logic. The PLL is accurate within the range of 50Mhz to 175MHz. L is hard-wired to 1. The reset clock value is 49.3 Mhz. See Table 116.

The formula for determining the system clock frequency is:

$$14.31818 * M/(N*2^{L})$$

For additional information, refer to ‘‘GUI, MEM, and SYS Clock PLL Rate Selection’’ on page 355.

Table 116. System Clock PLL Register (SYSCLK_PLL)

Bit	Description
31:12	Reserved
11:8	Value of N. Range: 4 to 15. Reset value: 9d
7	Reserved
6:0	Value of M. Range: 21 to 63. Reset value: 62d

Memory Delay Register

name: MEM_DELAY
address: GBASE | 00500030h, read/write
size: 32 bits
function: This register is used to control the delay on internal clock paths for the memory controller. This register should be initialized by BIOS and left unmodified by other software. See Table 117.

Table 117. Memory Delay Register (MEM_DELAY)

Bit	Description
31:8	Reserved
7:0	Delay value. Reset value: 1

VIDEO INPUT SUBSYSTEM

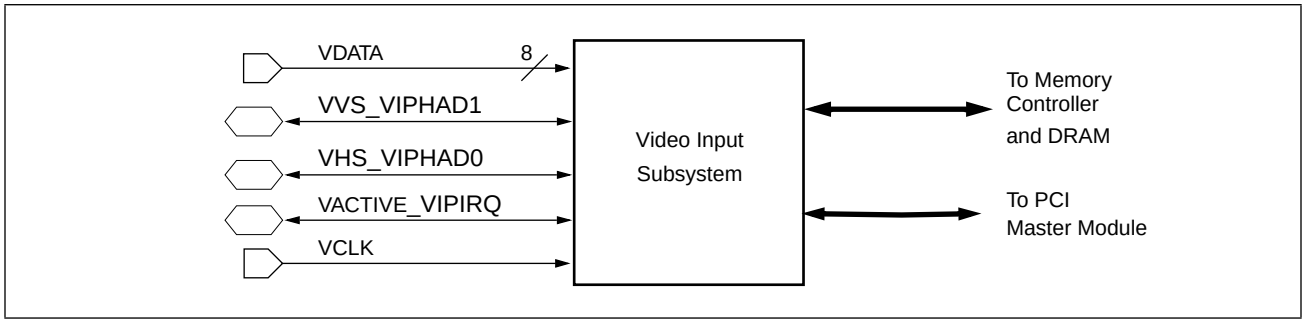
Introduction

The Bt2166 Video Input Subsystem accepts video data from various hardware video sources and formats and stores the video data in DRAM and/or PCI system memory. The input can operate in one of four modes:

1. Video Interface Port (VIP) mode: This mode is used to transfer compressed MPEG data streams from system memory to an external slave device
2. ByteStream mode: This mode is used to interface to Brooktree's VideoStream video decoder. In this mode, the input uses the VDATA[7:0] and VCLK inputs only. The ByteStream encoding uses out-of-range chrominance and luminance values to embed video formatting information into the video stream. The VVS_VIPHAD1, VHS_VIPHAD0, and VACTIVE_VIPIRQ signals can still serve as outputs in this mode but play no part in decoding the incoming video data.
3. Programmable mode: This mode is intended to interface with as many video parts as possible and is extremely programmable. In this mode the input uses the VDATA[7:0] and VCLK signals as inputs and can use the VVS_VIPHAD1, VHS_VIPHAD0, and VACTIVE_VIPIRQ signals as input or outputs, depending on how the Bt2166 is programmed.
4. CCIR656 mode: This mode requires only the video clock and video data pins. The CCIR656 (ITU656) mode supports EAV/SAV codes for video synchronization information and provides interface compatibility with many MPEG video decoder ICs.

Figure 16 shows an overview of the Video Input Subsystem.

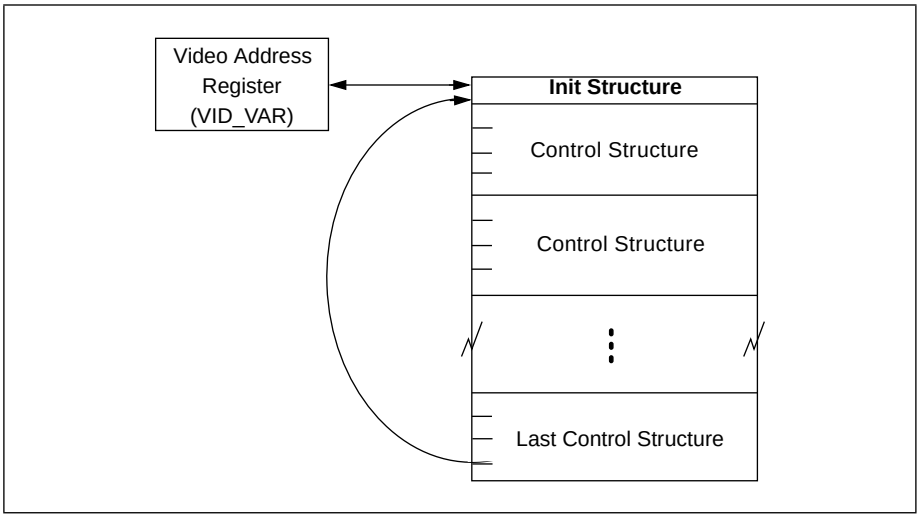
Figure 16. Video Input Subsystem



A control structure list stored in DRAM controls the video input logic. This DMA channel command-like architecture provides a great deal of flexibility to capture and display programs.

The Video Input Subsystem accesses the control structure list via the Video Address Register (VID_VAR), which points to the top of the list. The dword at the top of the list (the INIT STRUCTURE) contains initial control values. Each control structure in the list consists of four sequentially located dwords of DRAM. Thus, Figure 17 represents a typical control structure list.

Figure 17. Video Input Control Structures



The END_OF_LIST bit of the video input control structure indicates the end of the video control structure list. When a control structure is completed with this bit set, theVID_VAR starts the process over again. Any control structure can be marked to execute an interrupt upon completion. In addition, control structures can specify coverage of either vertical active lines or vertical blank lines, such as the vertical interval-time code marker or closed captioning information.

Each control structure can use a different capture format. For example, closed captioning data can be captured in mono format (Y only) while the active vertical lines are captured in 4:2:2 YCrCb format. In addition, a structure can be marked to *drop* specified lines of a specified field by marking off the required time without actually capturing video data into the DRAM, thus, easily skipping over either odd or even fields.

The VideoStream Decoder provides the ByteStream protocol that describes the video image delivered to the Bt2166 controller. The encoded information includes factors such as vertical reset, horizontal scaling, and vertical decimation.

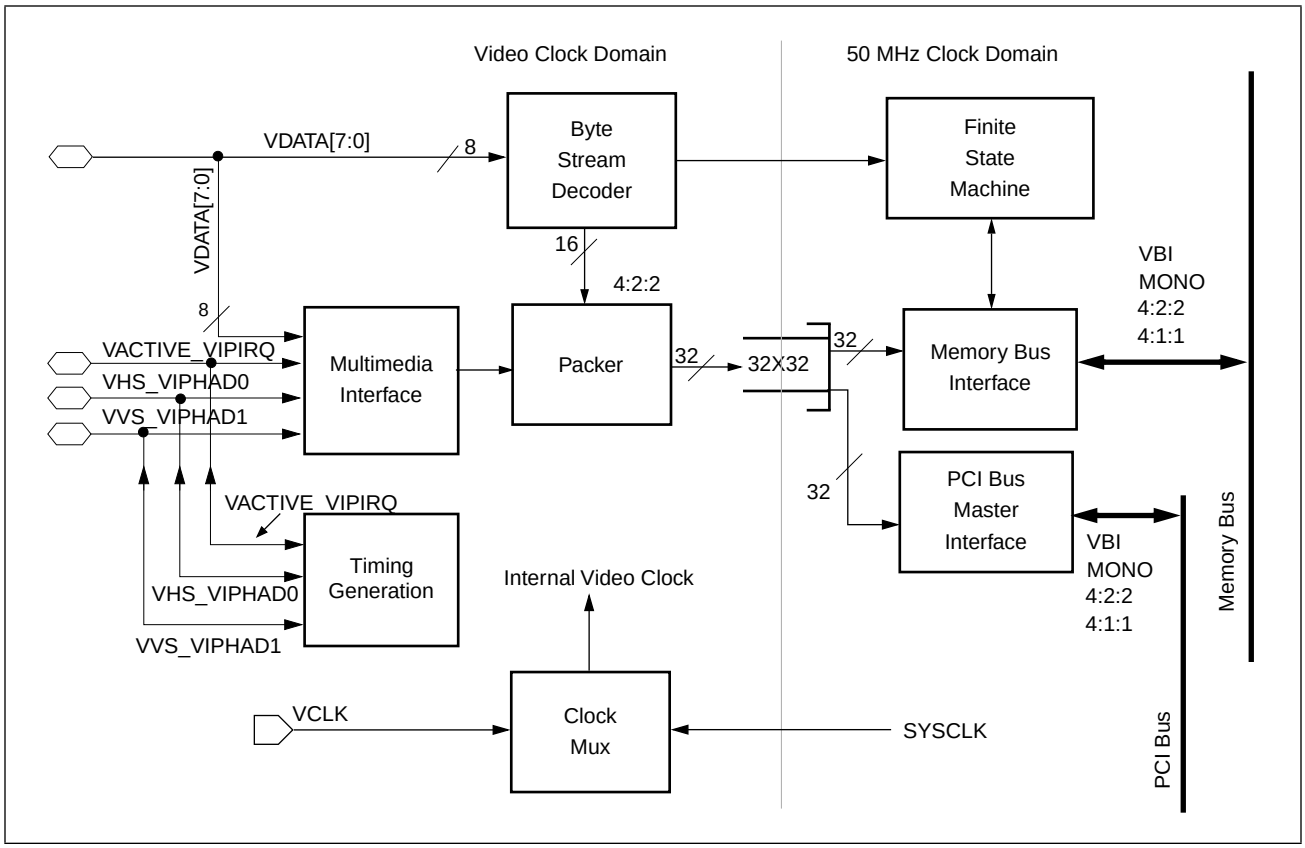
A ByteStream decoder within the Bt2166's video input logic reconstructs the two dimensional video image and any associated data, e.g. closed captioning, from this incoming byte stream. The desired portions of the data can be converted from 4:2:2 YCrCb into 4:1:1 or mono only and then packed into DRAM to minimize size/bandwidth. In addition, the pixels can remain in 4:2:2 format or can be decimated vertically and horizontally by an arbitrary factor. This further decimation is useful for obtaining icon-sized live video or for scaling video from a fixed size source, e.g. some MPEG hardware decompression ICs.

The video input control structure list can be used to control triple-buffering for both the video capture and video display processes of the Display Refresh Controller. For video display, control of the A/B/C buffer selection of the Display Refresh Controller video 1 FIFO, video 2 FIFO and video 3 (graphics) FIFO can be controlled by completion of a capture structure.

As shown in Figure 18, the multimedia data from the VideoStream Decoder is presented to the Byte Stream Decoder block which interprets the packed format into a valid pixel stream with surrounding video frame control signals extracted. The pixel stream is formatted and decimated into dwords destined for the DRAM media buffer. As each dword is formed, it is inserted into the 32x32 FIFO and passed from the video clock domain to the Bt2166 internal SYSCLK domain.

On the SYSCLK side, a finite state machine is in overall control of all aspects of the video input subsystem. It traverses the structure list, fetches each control or init structure in turn and presents the various fields, and handles the writing of the video data from the 32x32 FIFO to DRAM or to the PCI bus master.

Figure 18. Video Input Subsystem Block Diagram



INIT STRUCTURE Definition

The contents of the INIT STRUCTURE are listed in Table 118.

Table 118. INIT STRUCTURE

Bit(s)	Field	Description
31		MSB of MAXPIXEL_LIMIT0.
30:20	MAXPIXEL_LIMIT1	Max number of video source pixels that can be captured on a video line.
19:10	MAXPIXEL_LIMIT0	10 LSBs of the max number of video source pixels that can be captured on a video line (for software compatibility with previous hardware). The MSB is stored in bit 31.
9:0	Reserved	

The VideoStream Decoder's horizontal timing loop can shift the horizontal active to acquire good pixels in fast forward/reverse. The maximum number of pixels to transfer per line is set by the other fields in the INIT STRUCTURE. These fields allow for detection of loss of video lock horizontally and guarantee that only the anticipated amount of memory will be loaded by a control structure, i.e. a badly formatted video source cannot unintentionally overwrite frame buffer RAM or PCI system memory.

The maximum number of pixels in the vertical blanking interval lines and the active video lines can be different; the two choices are stored in MAXPIXEL_LIMIT1 and MAXPIXEL_LIMIT0. For software compatibility with previous hardware, the MSB of MAXPIXEL_LIMIT0 is stored in INIT_STRUCTURE[31]. DWORD0[15] (Table 119 on page 153) specifies which maximum pixel value to select for the video line currently being captured.

The maximum number of pixel fields should be set up to capture an even number of macropixels on each line. In other words, capture 4:1:1 data in 16-pixel multiples and 4:2:2 data in 8-pixel multiples.

Capture Control Structure Definition

The capture structure consists of four dwords of DRAM (DWORD0, DWORD1, DWORD2, DWORD3) as defined in Table 119, Table 120, Table 121, and Table 122.

Table 119. Capture Control Structure DWORD0 *(1 of 2)*

Bit(s)	Field	Description
31	END_OF_LIST	0 = Process the next sequential control structure. 1 = This is the last control structure in the list. Find the next one to process by following the Video Address Register pointer.
30	SET_VIDEO_ INTERRUPT	1 = Set the video interrupt flag at the end of processing this control structure.
29	PAUSE	0 = Continue video list processing at end of processing this structure. 1 = Reset the enable_video bit, VID_VCR[7] at the end of processing this structure and after all data has been stored in DRAM.
28:27	FORMAT	00 = 4:2:2 01 = 4:1:1 10 = mono (Y only) 11 = 4 to 1 horizontal decimation of 4:2:2
26	SCAN	0 = Even field only 1 = Odd field only
25	DROP	0 = Normal operation, write to DRAM 1 = Drop this field
24:23	V2SELECT	Select for video 2 DRAM buffer. 00 = Video 2 use DRAM buffer A for next frame 01 = Video 2 use DRAM buffer B for next frame 10 = Video 2 use DRAM buffer C for next frame
22:21	V1SELECT	Select for video 1 DRAM buffer. 00 = Video 1 use DRAM buffer A for next frame 01 = Video 1 use DRAM buffer B for next frame 10 = Video 1 use DRAM buffer C for next frame
20,16	GSELECT	Select for graphics DRAM buffer. 00 = Graphics use DRAM buffer A for next frame 01 = Graphics use DRAM buffer B for next frame 10 = Graphics use DRAM buffer C for next frame
19	VBI4BIT	0 = Normal mode 1 = Special 4 bits/pixel mode for VBI applications
18:17	DRAM_MEM_BURST	Size of burst size when writing to DRAM memory 00 = 4 dword bursts 01 = 8 dword bursts 10 = 12 dword bursts 11 = 16 dword bursts Note: Burst size to PCI memory is always 8 dwords.

Table 119. Capture Control Structure DWORD0 *(2 of 2)*

Bit(s)	Field	Description
15	PIX_LIMIT_SEL	Select the maximum pixel limit for the current video being captured. See Table 118 on page 152. 0 = MAX_PIXEL_LIMIT0 1 = MAX_PIXEL_LIMIT1
14	VERTICAL_LINE_ EXTENT[10]	MSB of VERTICAL_LINE_EXTENT field in DWORD1. See Table 120 on page 154
13	STARTING_VERTICAL_ LINE[10]	MSB of STARTING_VERTICAL_LINE field in DWORD0. See bits 9:0 below.
12	DDA_ENABLE	0 = Disable horizontal and vertical scaling 1 = Enable horizontal and vertical scaling by the scaling factors set in the VID_DDA register (see Table 134 on page 173)
11:10	CAPTURE_ENABLE	Control where captured data is sent 00 = DRAM only 01 = DRAM and PCI 10 = neither - no capture is done 11 = PCI only
9:0	STARTING_VERTICAL_ LINE	Matched against byte stream decoder vertical line counter to determine when to start data transfer for this control structure. For software compatibility with previous hardware, the MSB is stored in bit 13; see above.

The STARTING_VERTICAL_LINE field is compared to the current vertical line in the current input video field. If the field is the correct type (odd or even), all lines after the specified vertical line are captured until the number of lines specified in the VERTICAL_LINE_EXTENT field (Table 120) is captured or until vertical reset. Consequently, care must be taken when starting capture after reset. The first structure in the capture list may cause incorrect data to be captured if the Bt2166 starts capturing data at a point other than the beginning of the incoming video field. To avoid this problem, set the first capture structure to drop an entire field. Valid capture will then start in the second capture control structure.

Table 120. Capture Control Structure DWORD1

Bit(s)	Field	Description
31:21	PCI_LINE_PITCH	Number of dwords to add to PCI address for start of each video line captured. The 2 LSBs of this address must be zero, i.e., lines of video data must be separated from each other by a multiple of 4 dwords (16 bytes). This is a signed value.
20:10	DRAM_LINE_PITCH	Number of dwords to add to DRAM address for start of each video line captured. The 2 LSBs of this address must be zero, i.e., lines of video data must be separated from each other by a multiple of 4 dwords (16 bytes). This is a signed value.
9:0	VERTICAL_LINE_ EXTENT	Number of vertical lines to be transferred by the control structure. For software compatibility with previous hardware, the MSB is stored in bit 14 of DWORD0; See Table 119 on page 154.

Table 121. Capture Control Structure DWORD2

Bit(s)	Field	Description
31:21	TAG	This tag field is visible in the video address register when this control structure is being processed.
20		Reserved
19:1	DRAM_ADDRESS	DRAM dword address pointer to location in DRAM to store data captured by this control structure. This pointer must point to a 4 dword (16 byte) aligned location in DRAM.
0	DDA_VALUE	Select video 1/video 2 DDA value. 0 = Use value 1 for next frame. 1 = Use value 2 for next frame.

Table 122. Capture Control Structure DWORD3

Bit(s)	Field	Description
31:2	PCI_ADDRESS	PCI dword address pointer to location in PCI address space to store data captured by this control structure. The 2 LSBs of this address (i.e. the pointer) must point to a dword (4 byte) aligned location in PCI address space
1:0	DDA_MODE	Select video 1/video 2 DDA mode. 00 = No update of video 1 or video 2 DDA value. 01 = Update video 1 DDA value for next frame. 10 = Update video 2 DDA value for next frame. 11 = Update video 1 and video 2 DDA values for next frame.

VBI Mode Capture

VBI mode is a special video capture mode in which the video decoder captures data during the vertical blanking interval and passes it into memory completely unfiltered, allowing software to manipulate the data (decompress, error correct, etc.). VBI mode is designed to support cable/modem features in which embedded data is transmitted via the video signal — similar to close caption transmissions, but at a much higher data rate. To implement VBI mode in the video module, set DDA_ENABLE to zero (Table 119 on page 153) and use 4:2:2 format to avoid losing samples. If VBI mode yields more data than the system can reliably handle, use the VBI 4-bit mono mode. Four bit VBI throws away the 4 LSBs of each sample, effectively packing eight samples into a dword and halving the data rate.

Figure 19 illustrates VBI mode, VBI 4-bit mode, and the video formatting choices.

Video Formatting

The FORMAT field in DWORD0 (see Table 119 on page 153) specifies the mode of formatting applied to the signal entering the video module. The four formatting modes are defined below. See Figure 19 for a visual representation of the video formatting modes.

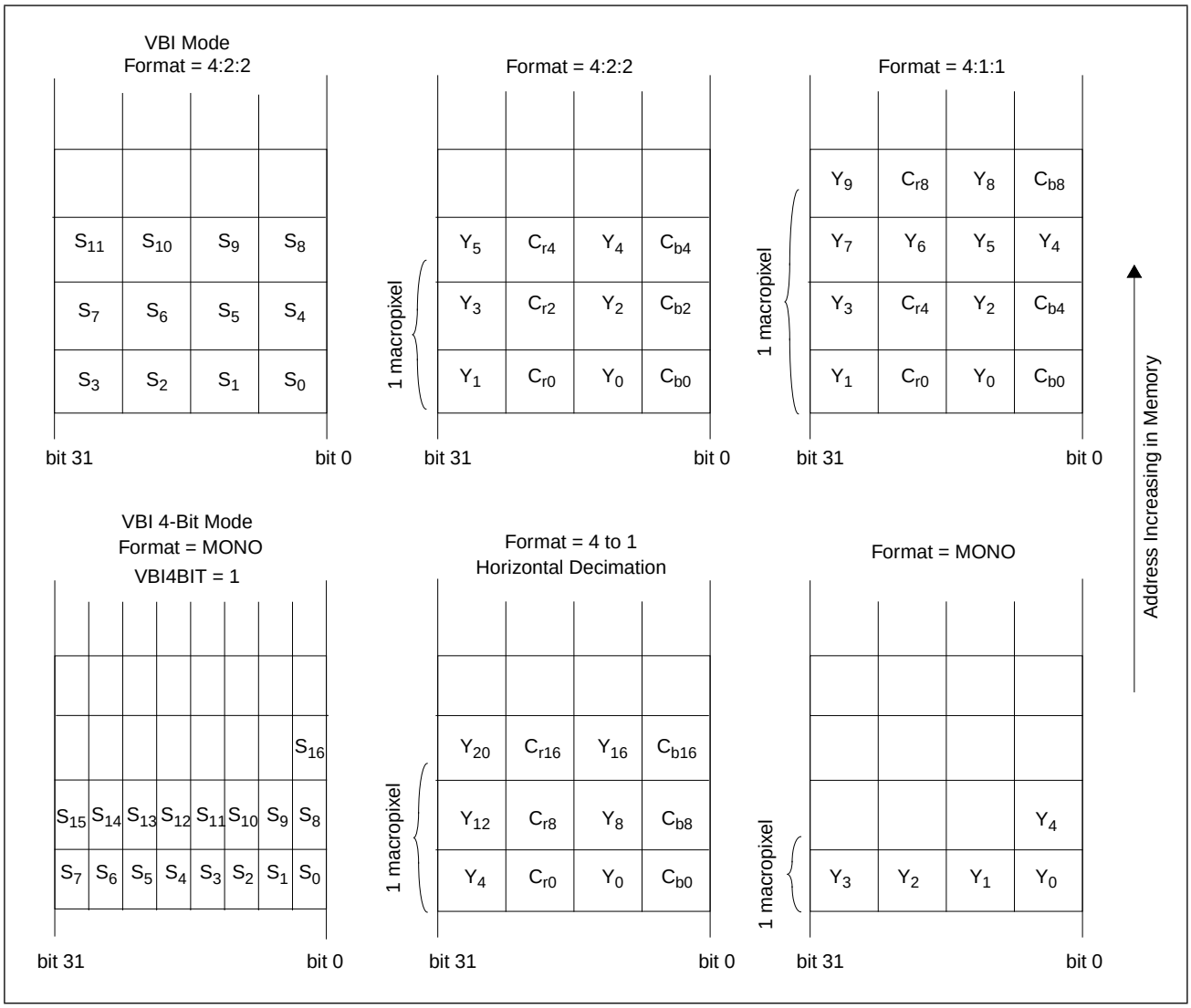
4:2:2 format — The video module receives all video signals in 4:2:2 mode. Selecting 4:2:2 does not reformat the captured video.

4:1:1 format — To format incoming video in 4:1:1 mode, the system throws away the unused data in the received 4:2:2 signal.

Mono format — To format incoming video in mono mode, the system throws away the color data in the received 4:2:2 signal.

4:1 horizontal decimation format — To format incoming video in horizontal decimation mode, the system performs a 4 to 1 downscaling in the horizontal direction only. This format can be used independently of the arbitrary DDA scaling.

Figure 19. Formatting of Video Data in Memory



Line Numbering System

The VideoStream Decoder's vertical timing loop allows the number of lines in the vertical active region to vary if the video chip is decoding a video tape recorder in any mode other than normal play. As a result, the VideoStream Decoder provides an accurate leading edge detection of vertical active. However the number of lines expected may differ, and the actual leading edge of vertical active could be on an unexpected line.

To make the control structure selection of starting line values work correctly, the vertical line counter in the byte stream decoder is reloaded with the value 64 whenever the first vertical active horizontal reset is detected. Normally, active video for an NTSC signal starts on line 22. When a video tape recorder is in a mode other than normal play, the active video may not start until much later (for example, line 28). If a capture control structure is coded to start capture on line 22 but active video does not actually start until line 28, six lines of inactive video would be captured. When using Brooktree's ByteStream interface, this problem can be prevented by loading the line counter with the value 64 on the first line of active video. Then the capture structure can be set to capture active video data starting at line 65.

The VideoStream Decoder has a horizontal and vertical scaling mechanism. Lines that are decimated must still be processed by the VideoStream Decoder so that lines coming to the Bt2166 are marked as either present or not. The marking is done by withholding the start of active pixels from lines that are discarded. The Bt2166 video input subsystem counts only active lines. Thus the line numbering system used in the control structures is based on a post scale count not the actual NTSC or PAL line number. If the DDA scaler included in the video module or the scaler in the VideoStream Decoder is enabled, then all pixel and line numbering is performed after the scaling process.

The VideoStream Decoder only scales vertically inside the active vertical region of the video frame. Consequently, all lines in the inactive region are presented (and counted) by the Bt2166. This ensures that the closed caption data will always be available, independent of the scaling set in the VideoStream Decoder. However, the DDA scaler included in this module does scale the video data from the inactive region of the display. To capture unscaled closed caption data, set up the capture structure to disable the DDA for the relevant lines.

The VideoStream Decoder scales horizontally on all lines. Therefore be sure to account for horizontal scaling in closed caption decoding.

When the VideoStream Decoder or DDA mechanism is set up for scaling, horizontal pixel counts and vertical line counts within the Bt2166 are post-scaled.

Live Video Input

Before performing video capture, the following tasks must be performed:

- VideoStream Decoder must be queried and initialized via the I^2C bus.
- Control structure list is set up in the frame buffer memory.
- Video address register is initialized to point to the control list structure.
- Clock select bit in VID_VCR[6] is set so that the VideoStream Decoder clock is used for capture.
- Video enable bit in VID_VCR[7] is set so that the video capture logic will come out of reset and begin processing the structure list.

For normal live video operation, the PAUSE bit should not be set in any control structure. To display a CIF resolution live video window, set up a control structure list with an INIT STRUCTURE and two CONTROL STRUCTUREs. The first control structure is set to wait for the even field and is set to *drop* the video. The second control structure is setup to wait for the odd field vertical line 32 with a mode 4:1:1 format. The MBUS address in this structure points to the first dword of the target buffer for the video image. The vertical line extent is set for 240 lines. The word line pitch field is set to add the number of dwords per line to the starting address of each line. In this simple case, no double buffering is used so the VID_VCR register should be set so that it controls the VIDEO A/B selection for the Display Refresh Controller.

To capture two fields, the first control structure is set to capture the odd field and points to the first scan line of the frame buffer. The second control structure is set to capture even fields and points to the second line of the buffer. The word line pitch of both control structure is set to twice the number of dwords per scan line.

Closed Captioning Capture With A Video Icon

A large video control structure list and a very small amount of buffer can be used to decode closed captioning while the video window is iconified. To implement this case, set up the list to process 30 separate NTSC frames and mark the last control structure to interrupt the CPU and continue. Each frame is processed by a set of two control structures (4 dwords in each structure). The first structure grabs the closed captioning line by being setup for the odd field line 11 with a format of "MONO" so that only the Y values are grabbed into a buffer. The second structure is setup to grab an iconified version of the video into a single small buffer. This structure's format is set to a 4 to 1 decimation of 4:2:2.

Each closed captioning structure points to a different line buffer while each icon structure points to the same video buffer. Thus the CPU will be interrupted once per second to process an entire second's worth of closed captioning information.

This requires approximately 241 dwords to hold the structure list, 4K dwords to hold the close captioning and 256 dwords to hold the video icon.

To allow for interrupt latency in the CPU, set the actual structures to capture in a double-buffered fashion with two one second sequences of video captioning so that software can ping pong back and forth (i.e., capture 60 frames worth). Set the tag values of the first 30 frames in the control structures to 01h; and set the tags of the second 30 frames to 02h. Thus, allowing software to quickly and easily tell which second's worth of close capturing is currently being processed.

Single Frame Record Example

To simplify single frame capture, set the PAUSE and INTERRUPT bits in the last control structure element. Upon completion of the list, the video control structure list processor will stop and an interrupt will be generated to the CPU.

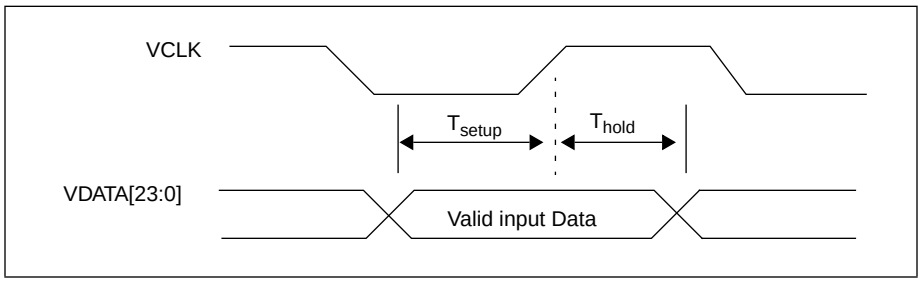
A more complicated example includes both a live video window loop and a capture list. The live video loop occurs first in the control structure list with the END_OF_LIST bit set in the last structure. As video is captured into the live window video buffer, it encounters this EOL bit and restarts at the top.

If the application builds two control structure extensions immediately after the live window portion, the capture part is set to capture full resolution for both fields with its last control structure marked as INTERRUPT and PAUSE. Thus, when the application is instructed to grab a single frame, it turns off the END_OF_LIST bit in the last control structure of the live video window and waits for the interrupt to indicate completion of the capture. With this strategy, the live video that is captured is a fresh, full time coherent frame.

Video Interface Timing

Figure 20 provides the video interface timing diagram.

Figure 20. Video Interface Timing Diagram



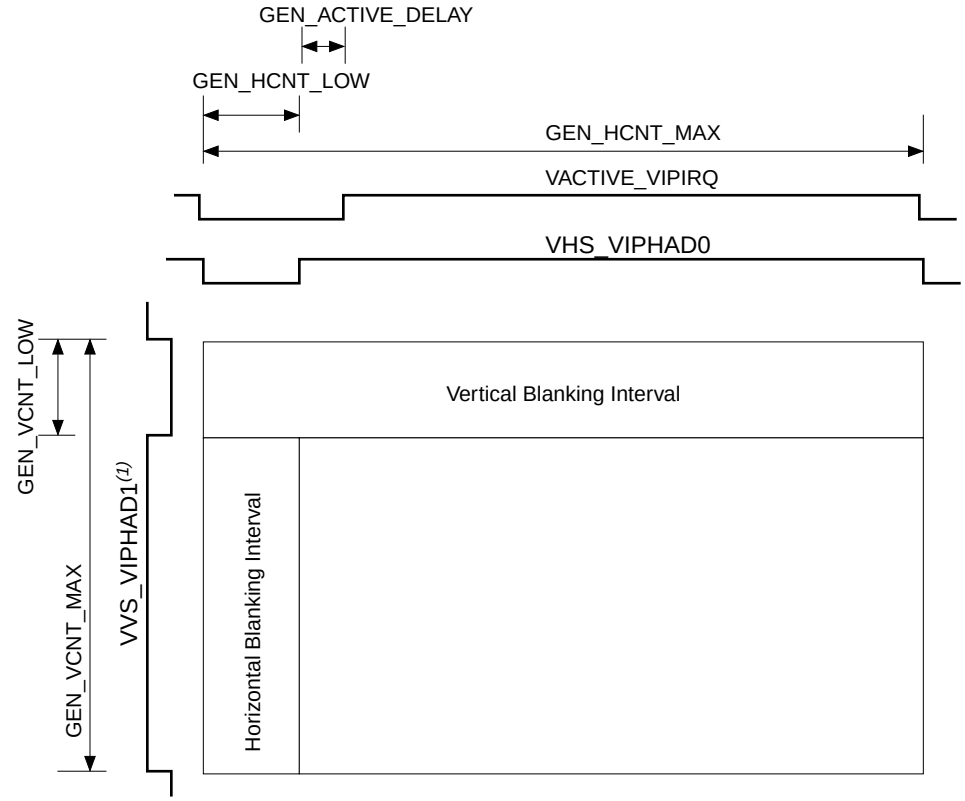
Bytestream and parallel video interface timing:

T_{setup} = 1.0 ns before VCLK rising edge

T_{hold} = 3.0 ns after VCLK rising edge

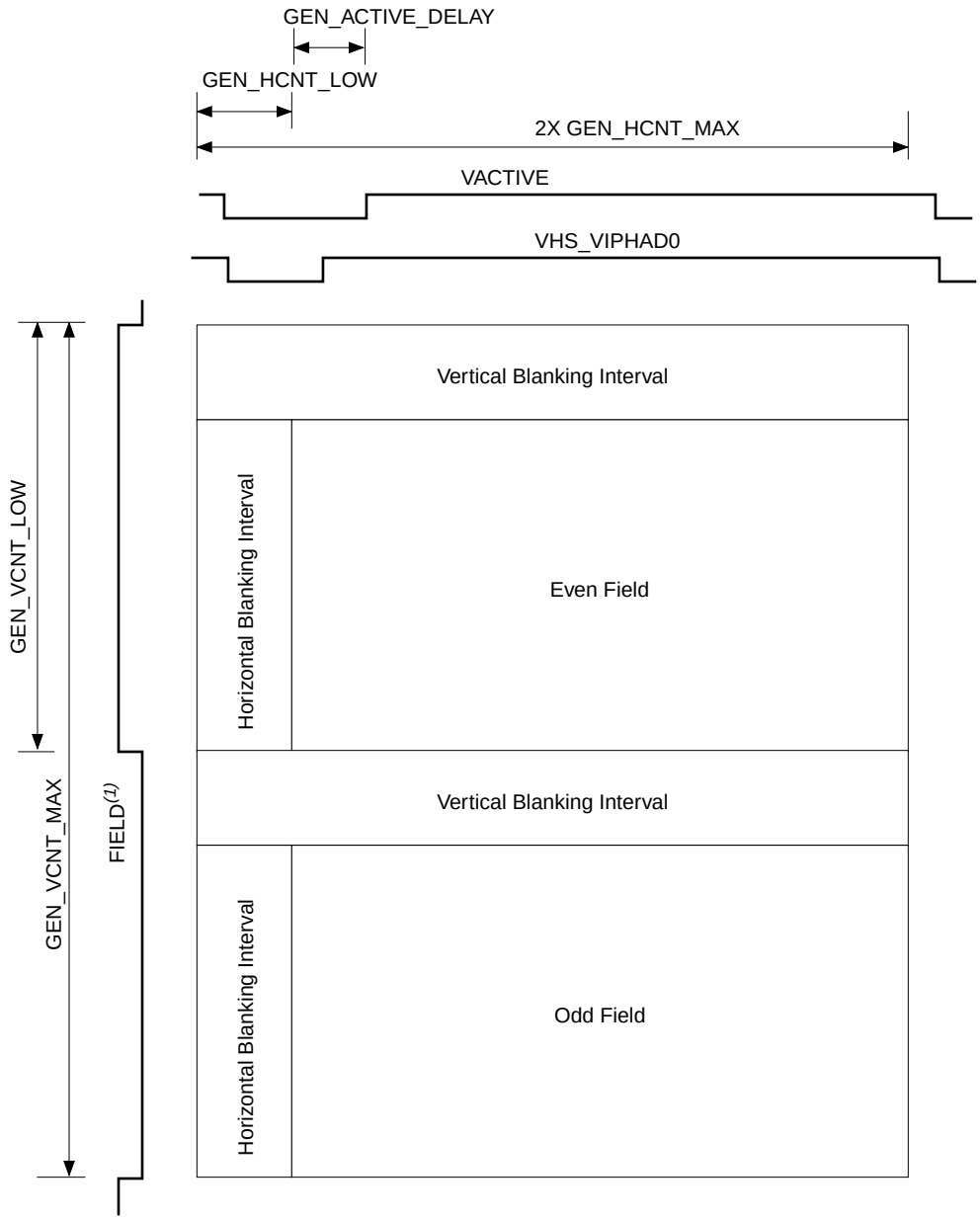
Figure 21 through Figure 23 are the video timing diagrams for interlaced and non-interlaced modes.

Figure 21. Generation of VHS_VIPHAD0 and VVS_VIPHAD1 in Non-Interlaced Mode



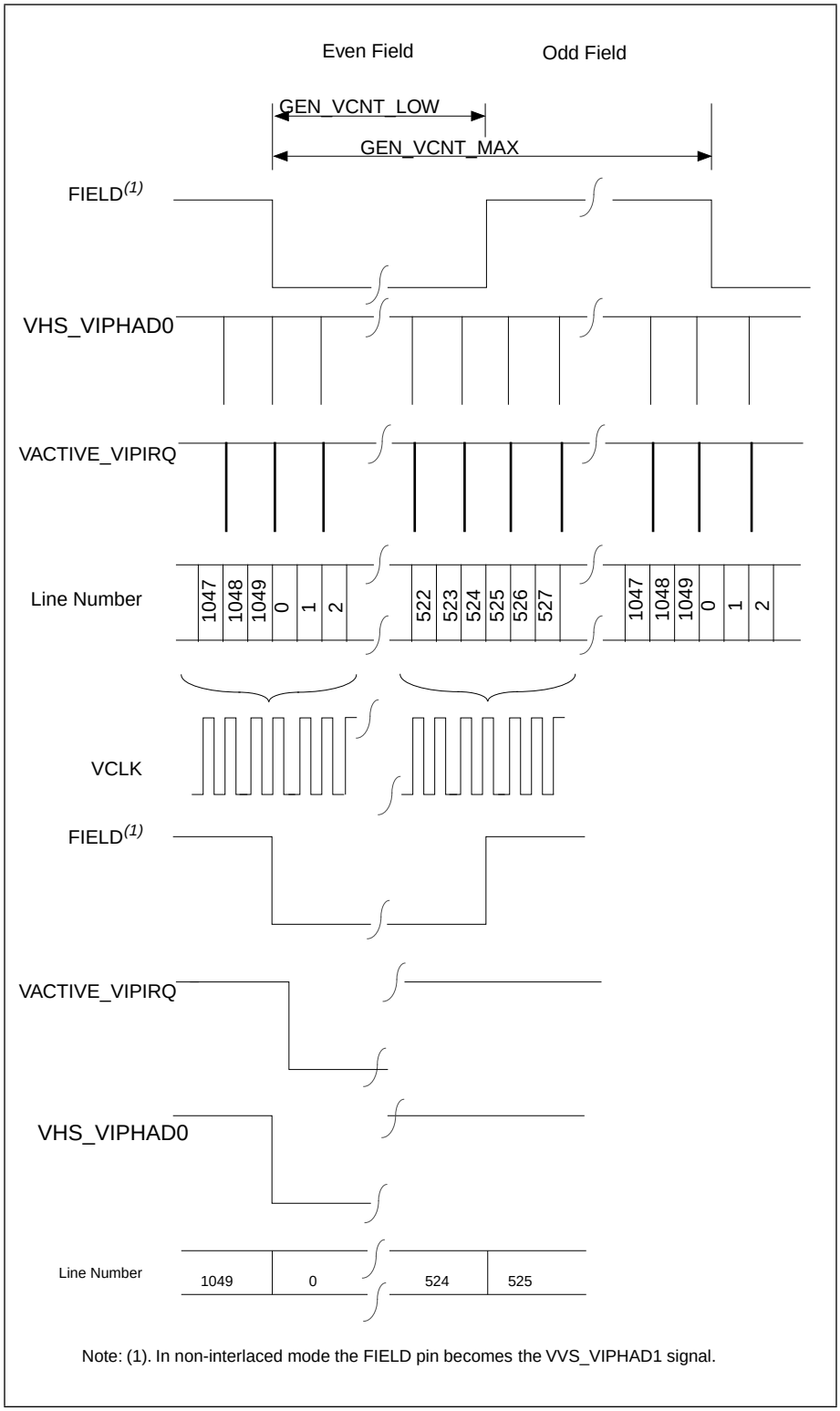
Note: (1). In interlaced mode, the VVS_VIPHAD1 pin becomes the FIELD sig-

Figure 22. Generation of VHS_VIPHAD0 and FIELD in Interlaced Mode



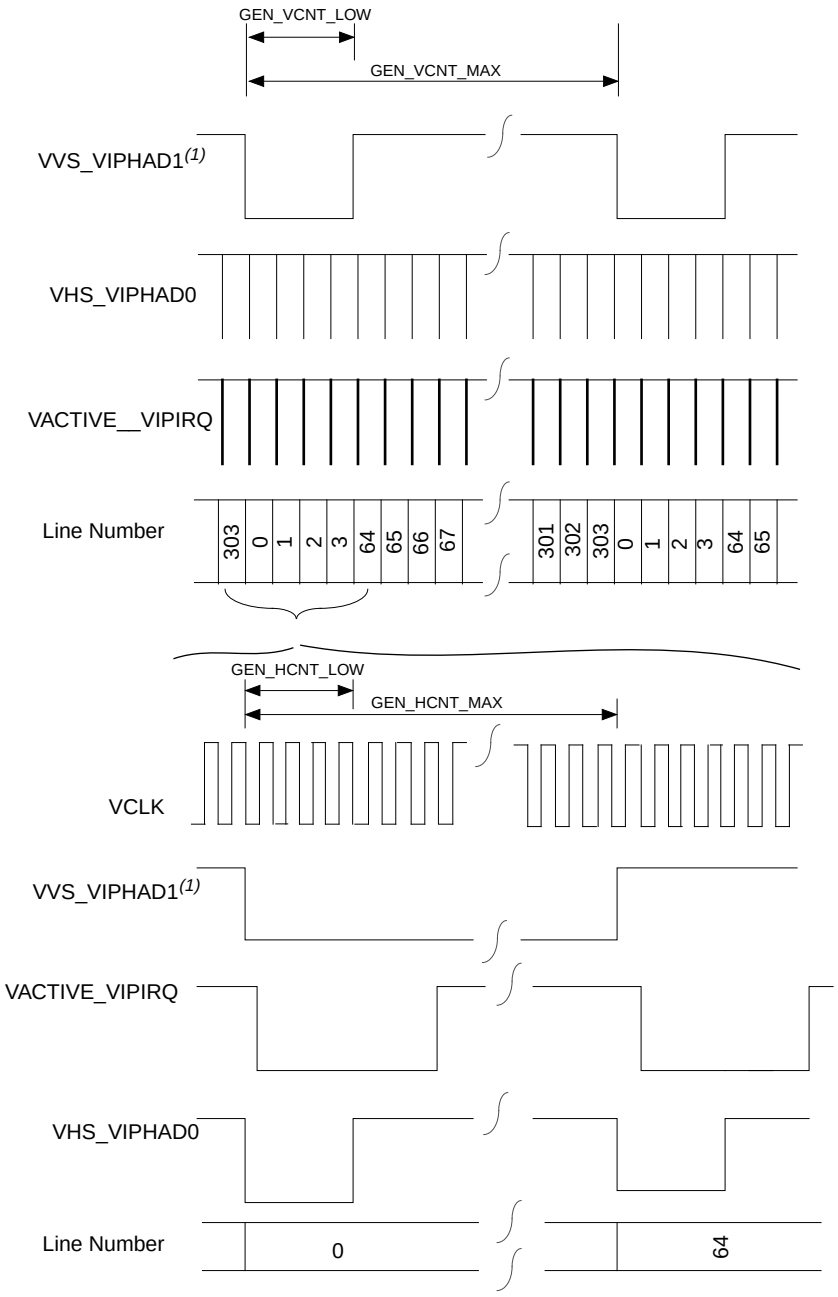
Note: (1). In non-interlaced mode, the FIELD pin becomes the VVS_VIPHAD1 signal.

Figure 23. Timing of VHS_VIPHAD0 and FIELD in Interlaced Mode



Note: (1). In non-interlaced mode the FIELD pin becomes the VVS_VIPHAD1 signal.

Figure 24. Timing of VHS_VIPHAD0 and FIELD in Non-Interlaced Mode



Note: (1). In interlaced mode, the VVS_VIPHAD1 pin becomes the FIELD signal.

Video Interface Port

The Video Interface Port (VIP) can be used to transfer compressed MPEG data streams from system memory to an external slave device. The maximum number of bytes transferred per FIFO-based command is 64k bytes. The maximum number of bytes transferred per register-based command is 4 bytes.

For additional information on the VIP and a list of expected slave devices, refer to the SGS-Thompson Microelectronic's™ *Video Interface Port (VIP) Specification 1.1*.

The VIP interface consists of five basic signals: two data, one control, a clock, and an interrupt. The data and control signals are bidirectional. The clock is a master-based output only signal, while the interrupt is a slave-based output signal. These signals share the Video Input Subsytem's VHS_VIPHAD0, VVS_VIPHAD1, and VACTIVE_VIPIRQ, and the I^2C logic pins DDCSDA_VIPCLK and DDCSCL_VIPCTL. When VIP is enabled with VIP_CNFG[1], the I^2C function of DDCSDA_VIPCLK and DDCSCL_VIPCTL must be disabled via GRP_I2C_CTRLW (see ''I2C Control WRITE Register'' on page 83). Also, VID_VCR[3:2] (see Table 129 on page 170) must be set to 2'b10 to enable the ITU656 interface.

VIP is enabled with VIP_CNFG[1] and the external interrupt is set in VIP_CNFG[6].

Typical FIFO and register transfer sequences are listed below.

FIFO Transfer Sequence:

1. The DRAM address of the first byte of compressed data is written into the FIFO_ADDR register.
2. The number of bytes to transfer minus one (N-1) is loaded into the FIFO_COMMAND register along with the FIFO command (described in the *VIP Specification* page 2.2.6). For FIFO writes, all transfer data must be loaded into the DRAM before starting the transfer.
3. VIP_CNFG[2] is written to 'b1 to start the transfer.
4. VIP_CNFG[2] is cleared by the hardware to indicate the transfer is complete. And, the device interrupt INT_SR[14] (see Table 108 on page 138) is generated, thus signalling the end of a transfer.

Register Write Sequence:

1. Register data for the slave device is written to the REG_WR_DATA register.
2. The number of bytes to transfer minus one (N-1), register command, and register address is loaded into the REG_COMMAND register
3. VIP_CNFG[3] is written to 'b1 to start the transfer.
4. VIP_CNFG[3] is cleared by the hardware, thus signalling the end of a transfer.

Register Read Sequence:

1. The number of bytes to transfer minus one (N-1), register command, and register address is written into the REG_COMMAND register
2. VIP_CNFG[3] is written to 'b1 to start the transfer.
3. VIP_CNFG[4] is set to 'b1 by the hardware to indicate the read is complete. Reading the bit will cause it to be cleared.
4. Register data is now readable from the REG_RD_DATA register.

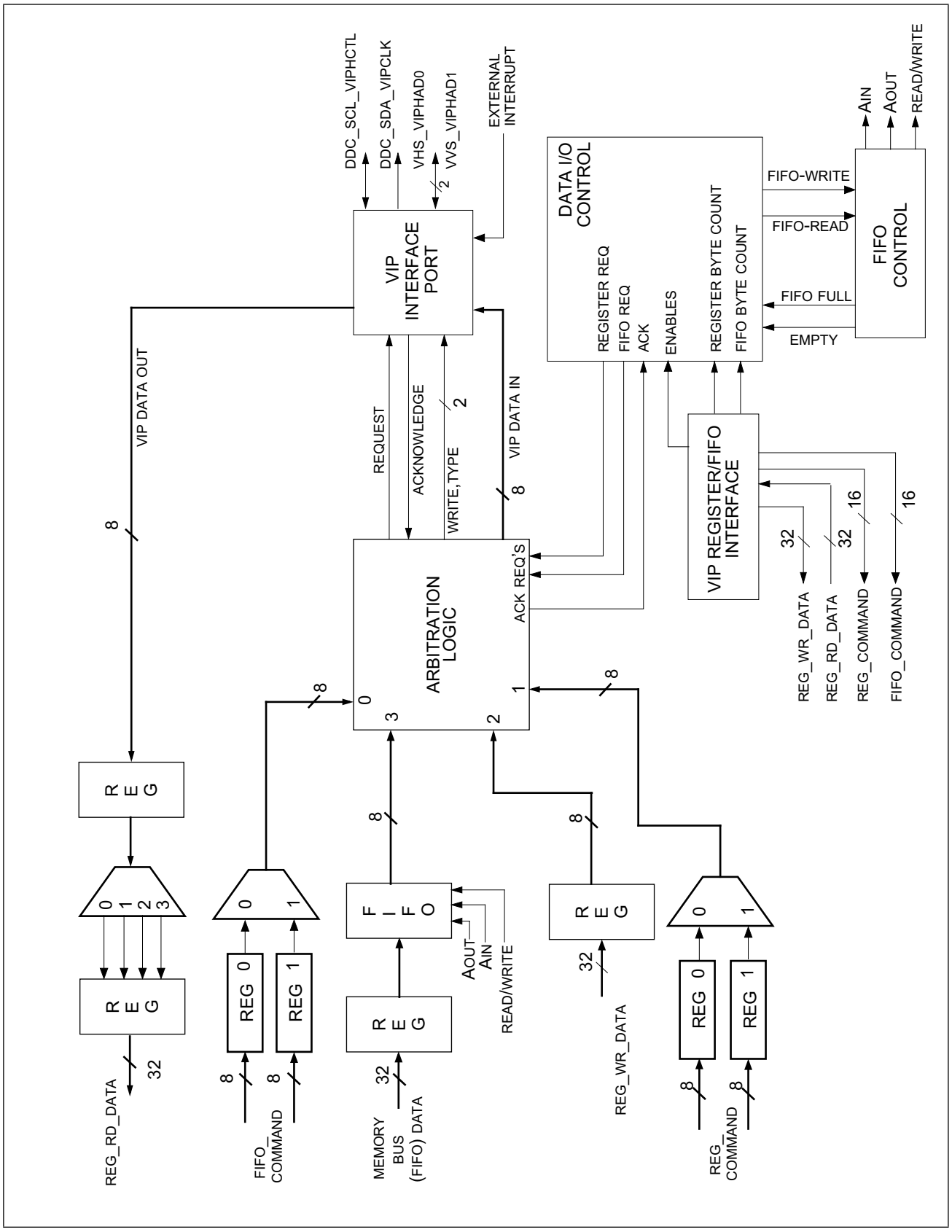
If an interrupt line is present on an external slave device, the line should be connected to the VACTIVE_VIPIRQ pin. And, the type of interrupt (edge/level) should be programmed via VIP_CNFG[6]. The interrupt status is reported in INT_SR[13] (see Table 108 on page 138).

The Brooktree implementation of VIP conforms to the *Video Interface Port (VIP) Specification 1.1*. with the following exceptions.

- The current VIP design supports only one slave device on the VIP bus.
- Section 2.25 of the *VIP Specification* requires that software provide a timeout mechanism; the Bt2166 does not support this feature, therefore does not return a 0xFF condition in response to a Vendor ID register request. Consequently, if software has an active command and VIP_CNFG[3 or 2], as required, is not cleared after the VIP-programmable amount of time, assume a timeout condition has occurred and take the appropriate steps to handle the timeout.
- VBI data is not supported through the VIP interface at this time. ITU-656 VBI data is supported through the Video Input Subsytem's Multimedia interface.

Refer to Figure 25 for an overview of the VIP subsystem.

Figure 25. VIP Subsystem Block Diagram



FIFO Command Register

name: FIFO_COMMAND
address: GBASE | 00500200h, write only
size: 32 bits
function: This register contains the FIFO command and the count of bytes sent with the command as shown in Table 123. Bits 15:0 make up the FIFO command are defined in more detail in the *Video Interface Port Specifcation Version 1.1*, section 2.2.6

Table 123. FIFO Command (FIFO_COMMAND)

Bit	Field	Description
31:16	FIFO byte count	N–1 value to indicate the number of bytes to be sent with this FIFO command.
15	DEVSEL1	Device select 1 -- supports up to 4 devices
14	DEVSEL0	Device select 0 -- supports up to 4 devices
13	READ/WRITE	FIFO write-only 0 = FIFO write
12	FIFO Access	Set to 1 for FIFO access
11:8	FIFO Address	FIFO address -- selects one of 16 FIFO ports
7:0	Reserved	Write to zero

FIFO Address Register

name: FIFO_ADDR
address: GBASE | 00500204h, write only
size: 32 bits
function: Starting address in video memory of FIFO data as shown in Table 124.

Table 124. FIFO Address Register (FIFO_ADDR)

Bit	Field	Description
31:19	Reserved	
18:0	DRAM address	DRAM dword address of the start of FIFO data.

VIP Register Write

name: REG_WR_DATA
address: GBASE | 00500208h, write only
size: 32 bits
function: This register contains registered output data to MPEG/slave device as shown in Table 125.

Table 125. VIP Register Write (REG_WR_DATA)

Bit	Field	Description
31:0	VIP register write data	Register data to be transferred to VIP slave device.

VIP Command and Byte Count Register

name: REG_COMMAND
address: GBASE | 0050020Ch, write only
size: 32 bits
function: This register contains the register command and the count of bytes sent with the command as shown in Table 126. Bits 15:0 make up the register command are defined in more detail in the *Video Interface Port Specifcation Version 1.1*, section 2.2.6

Table 126. VIP Command and Byte Count Register (REG_COMMAND)

Bit	Field	Description
31:19	Reserved	
18:16	Register byte count	N–1 value to indicate the number of bytes to be sent or read with this register command. Valid values: 0 to 3.
	DEVSEL1	Device select 1 -- supports up to 4 devices
14	DEVSEL0	Device select 0 -- supports up to 4 devices
13	READ/WRITE	Register read/write and FIFO read-only. If REG_COMMAND[12] = 1, then: 1 = FIFO read 0 = not used If REG_COMMAND[12] = 0, then: 1 = Register read 0 = Register write
12	Register Access	1 = FIFO read access 0 = register read/write access
11-0	Register Address	Register address bus bits

VIP Register Read

name: REG_RD_DATA
address: GBASE | 00500210h, read only
size: 32 bits
function: This register contains registered input data from MPEG/slave device, as shown in Table 127.

Table 127. VIP Register Read (REG_RD_DATA)

Bit	Field	Description
31:0	VIP register read data	Data received from VIP slave device

VIP Configuration Register

name: VIP_CNFG
address: GBASE | 00500214h, read/write
size: 8 bits
function: This register contains various control bits for the VIP module as shown in Table 128.

Table 128. VIP Configuration Register (VIP_CNFG)

Bit	Field	Description
7	Reserved	
6	External device interrupt	Selects type of external device interrupt 0 = Edge 1 = Level
5	VIPCLOCK Enable	Enable DDC_SDA_VIPCLK 0 = Disable clock 1 = Enable clock Also controls clock pad driver
4	VIP bus read request	0 = Read not complete 1 = Read complete Read only, I/O read operation will clear bit
3	VIP register command	0 = No register command 1 = Command ready (cleared by hardware)
2	VIP FIFO command	0 = No FIFO command 1 = Command ready (cleared by hardware)
1	VIP enable	Enable for VIP module 0 = Reset 1 = Enable
0	Reserved	

Video Buffer Management Hardware

The video input buffer management hardware registers are listed below.

Video Control Register

name: VID_VCR
address: GBASE | 005003C0h, read/write
size: 8 bits
function: This register contains various control bits for the video input function as shown in Table 129.

Table 129. Video Control Register (VID_VCR)

Bit	Field	Description
7	ENABLE_VIDEO	0 = Reset video subsystem 1 = Enable video input processing. Hold at zero until after clock selection is made by VID_VCR[6]. Reset by the pause function in the control structures.
6	CLOCK_SELECT	0 = Use SYSCLK 1 = Use video input clock, i.e. the VCLK. Set to one if a VideoStream Decoder is known to be present from I^2C inquiries. Once the clock bit is set then the enable bit, VID_VCR[7], can be set on a subsequent I/O operation.
5:4	Reserved	Must be 0.
3:2	Enable decoder/MPEG input interfacer	00 = VideoStream Decoder interface 01 = MPEG input interface (see “Programmable Multimedia Interface” on page 175) 10 = ITU 656 input interface 11 = Reserved
1:0	Reserved	

Video One Register

name: VID_V1R
address: GBASE | 005003C1h, read/write
size: 8 bits
function: This register drives the Display Refresh Controller's video1 buffer address logic to select video FIFO 1 pointer A, B, or C. This allows direct software control of tripple-buffering for the video window associated with video FIFO one in the Display Refresh Controller. As shown in Table 130, the state of these bits can be automatically controlled by bits in the video input control structure.

Table 130. Video One Register (VID_V1R)

Bit	Description
7:3	Reserved, must be 0.
2	Select video one DDA_VALUE. (See Table 121, "Capture Control Structure DWORD2," on page 155, bit 0.) 0 = Use video DDA value 1 1 = Use video DDA value 2
1:0	Select Display Refresh Controller Video FIFO 1. Can be updated by the video structures. 00 = Use video buffer A for video FIFO 1 at the top of next frame 01 = Use video buffer B for video FIFO 1 at the top of next frame 10 = Use video buffer C for video FIFO 1 at the top of next frame

Video Two Register

name: VID_V2R
address: GBASE | 005003C2h, read/write
size: 8 bits
function: This register drives the Display Refresh Controller's video2 buffer address logic to select video FIFO 2 pointer A, B, or C. This allows direct software control of tripple-buffering for the video window associated with video FIFO two in the Display Refresh Controller. As shown in Table 131, the state of these bits can be automatically controlled by bits in the video input control structure.

Table 131. Video Two Register (VID_V2R)

Bit	Description
7:3	Reserved, must be 0.
2	Select video two DDA_VALUE. (See Table 121, "Capture Control Structure DWORD2," on page 155, bit 0.) 0 = Use video DDA value 1 1 = Use video DDA value 2
1:0	Select Display Refresh Controller Video FIFO 2. Can be updated by the video structures. 00 = Use video buffer A for video FIFO 2 at the top of next frame 01 = Use video buffer B for video FIFO 2 at the top of next frame 10 = Use video buffer C for video FIFO 2 at the top of next frame

Video Three Register

name: VID_V3R
address: GBASE | 005003C3h, read/write
size: 8 bits
function: This register drives the Display Refresh Controller's video 3 buffer address logic to select video FIFO 3 pointer A, B, or C. This allows direct software control of tripple-buffering for the video window associated with video FIFO three in the Display Refresh Controller. As shown in Table 130, the state of these bits can be automatically controlled by bits in the video input control structure.

Table 132. Video Three Register (VID_V3R)

Bit	Description
7:2	Reserved, must be 0.
1:0	Select Display Refresh Controller graphics FIFO. Can be updated by the video structures. 00 = Use graphics buffer A for graphics FIFO 3 at the top of next frame 01 = Use graphics buffer B for graphics FIFO 3 at the top of next frame 10 = Use graphics buffer C for graphics FIFO 3 at the top of next frame

Video Address Register

name: VID_VAR
address: GBASE | 005003C4h, read/write
size: 32 bits
function: The structure address field in this register is used at the end of every video capture list to know where to begin the next capture list. The bit contents of this registers are shown in Table 133.

Table 133. Video Address Register (VID_VAR)

Bit	Field	Description
31:21	Current Tag	Read-only view of tag field of control structure in process. Reset to zero by chip reset. Undefined when ENABLE_VIDEO (VID_VCR[7]) is not set (see Table 129 on page 170).
20:0	Structure Address	DRAM dword address of first structure in the control structure list. Note: this always points to an INIT STRUCTURE followed by a variable number of control structures.

Video DDA Register

name: VID_DDA
address: GBASE | 005003C8h, read/write
size: 32 bits
function: This register controls video downscaling. The video module uses digital differential analyzer (DDA) techniques and simple decimation to horizontally and/or vertically downscale the video input to an arbitrary dimension. Table 134 shows the bit contents of the VID_DDA register.

Table 134. Video DDA Register (VID_DDA)

Bit(s)	Field	Description
31:28	Reserved	Must be zero
27:16	DDA_VERT	Vertical digital differential analyzer scaling factor
15:12	Reserved	Must be zero
11:0	DDA_HORIZ	Horizontal digital differential analyzer scaling factor

The DDA scaling factors are calculated by multiplying the fraction of video input pixels and/or lines to downscale by the value 0x1000. For example, to downscale both the horizontal and vertical dimensions by 75% (i.e. to 25% of the original size):

```
0.75 X 0x1000 = 0xC00
```

Hence, the DDA_VERT and DDA_HORIZ scale factors are both 0xC00.

To downscale a CIF resolution video to QCIF resolution video (50% downscale), the DDA_VERT and DDA_HORIZ scale factors are both 0x800.

DDA scaling can be performed on a capture-structure by capture-structure basis. To enable DDA scaling, turn on DDA_ENABLE (DWORD0 bit 12, Table 119 on page 153). This allows the software to perform additional functions, such as simultaneously capturing unscaled close caption information and scaled video data. DDA scaling is independent of the selected formatting mode and can be independently enabled or disabled.

Video Status Register

name: VID_STAT
address: GBASE | 005003CCh, read only
size: 32 bits
function: This register allows the software to determine where within the input video frame the video capture module is currently located. Because the contents of this register changes rapidly and the data exists within the video clock domain, reading this register provides unreliable results. To reliably read this register, read it several times until the same value (except for perhaps masking the lower bits of the pixel count field) has been read identically for at least two subsequent reads. The pixel count field is divided by 16 because it changes so fast. However the resulting value still changes too fast to be used for any function other than possibly a mediocre random number generator. Table 135 defines the bit contents of the VID_STAT register.

Table 135. Video Status Register (VID_STAT)

Bits	Field	Description
31		Reserved
30:20	CURR_TAG	Current tag value - same as "Current Tag" field of VID_VAR register. (See Table 133 on page 172.)
19	CURR_ACTIVE	1 = Current structure is actively capturing video input data
18	CURR_FIELD	Odd/even field of current video input data 1 = 0dd field 0 = Even field
17:7	CURR_LINE	Line number of current video input data
6:0	CURR_PIXEL	Pixel number divided by 16 of current video input data

Programmable Multimedia Interface

The Bt2166 can optionally use an alternate decode interface instead of Brooktree's VideoStream Decoder. This option is enabled by setting VID_VCR[3:2] to 2'b01; see Table 129 on page 170. In this mode, the following three registers are in operation.

Video Timing Generation Vertical Register

name: VID_GNV
address: GBASE | 005003D0h, read/write
size: 32 bits
function: This register controls the generation of the VVS_VIPHAD1/FIELD video timing signal. For the bit definition of the Video Timing Generation Vertical Register, refer to Table 136.

Table 136. Video Timing Generation Vertical Register (VID_GNV)

Bits	Field	Description
31:27	Reserved	
26:16	GEN_VCNT_MAX	Vertical count maximum value. For a visual representation of how this value affects video timing generation see Figure 23 on page 162 and Figure 24 on page 163.
15:11	Reserved	
10:0	GEN_VCNT_LOW	Vertical count lower limit value. For a visual representation of how this value affects video timing generation see Figure 23 on page 162 and Figure 24 on page 163.

Video Timing Generation Horizontal Register

name: VID_GNH
address: GBASE | 005003D4h, read/write
size: 32 bits
function: This register controls the generation of the VHS_VIPHAD0 video timing signal. For the bit definition of the Video Timing Generation Horizontal Register, refer to Table 137.

Table 137. Video Timing Generation Horizontal Register (VID_GNH)

Bits	Field	Description
31:27	Reserved	
26:16	GEN_HCNT_MAX	Horizontal count maximum value. For a visual representation of how this value affects video timing generation see Figure 23 on page 162 and Figure 24 on page 163.
15:11	Reserved	
10:0	GEN_HCNT_LOW	Horizontal count lower limit value. For a visual representation of how this value affects video timing generation see Figure 23 on page 162 and Figure 24 on page 163.

Video Timing Generation Control Register

name: VID_GNC
address: GBASE | 005003D8h, read/write
size: 32 bits
function: This register controls the generation of the VACTIVE_VIPIRQ video timing signal, and controls the operation of the programmable multimedia interface. For the bit definition, refer to Table 138.

Table 138. Video Timing Generation Control Register (VID_GNC) *(1 of 2)*

Bits	Field	Description
31:28	Reserved	
27	GEN_VACTIVE_D1	0 = Use active input without changes 1 = Delay active input by one clock cycle
26:16	GEN_VACTIVE_DELAY	Number of clock cycles to delay from VVS_VIPHAD1 rising edge before start of active video.
15:11	Reserved	
10	GEN_INTERLACED	0 = Non-interlaced mode—video timing generation operates as in Figure 23 on page 162. 1 = Interlaced mode — video timing generation operates as in and Figure 24 on page 163.

Table 138. Video Timing Generation Control Register (VID_GNC) *(2 of 2)*

Bits	Field	Description
9	GEN_CR_Y_CB_Y	Swap the order of Y and Cr or Cb pixels in the input video. 0 = First pixel on a video line is Cr or Cb 1 = First pixel on a video line is Y
8	GEN_VVS_EN	Enables the VVS_VIPHAD1/FIELD output pin for timing generation for the VVS_VIPHAD1/FIELD signal
7	GEN_VHS_EN	Enables the VHS_VIPHAD0 output pin for timing generation for the VHS_VIPHAD0 signal
6	GEN_VACTIVE_EN	Enables the VACTIVE_VIPIRQ output pin for timing generation for the VACTIVE_VIPIRQ signal
5	GEN_DAC_POLARITY	Selects which edge of the DAC vertical sync signal to synchronize (genlock) the Bt2166 timing generation
4	GEN_DAC_GENLOCK	Enables genlocking (synchronizing) of the Bt2166 timing generation signal to the vertical synchronization signal of the DAC
3:2	GEN_DAC_COUNT	Specifies which DAC vertical synchronization edges to synchronize to. 00 = Synchronize to every rising or rolling edge 01 = Synchronize to every 2nd rising or rolling edge 10 = Synchronize to every 3rd rising or rolling edge 11 = Synchronize to every 4th rising or rolling edge
1	GEN_VVS_POLARITY	0 = No VVS_VIPHAD1 polarity inversion 1 = Invert polarity of the VVS_VIPHAD1/FIELD output (if VID_GNC[8] enabled) and input signal values
0	GEN_VHS_POLARITY	0 = No VHS_VIPHAD0 polarity inversion 1 = Invert polarity of the VHS_VIPHAD0 output (if VID_GNC[7] enabled) and input signal values

DISPLAY REFRESH CONTROLLER

Introduction

The display refresh module is responsible for fetching display memory data for generating screen outputs and for generating CRT timing signals. This includes back-end video upscaling and mixing, chroma and color keying, and cursor overlay.

Features

- Supports two video planes (any format) with replicative vertical upscaling simultaneously displayed over a graphics plane
- Supports one video 4:2:2 plane w/3-tap vertical upscaling not displayed over graphics
- Supports one video plane (any format) w/2-tap vertical upscaling displayed over a graphics plane
- Supports 3-tap horizontal upscaling in a 4:2:2 format video plane
- Supports 2-tap or replicative horizontal upscaling for video (any format)
- Supports Video Chroma Keying and Color Keying for two video planes
- 4:2:2 YC_rC_b, 15/16/32-bit RGB Video
- 95 Mhz max graphics w/video pixel rate
- 4, 8, 15, 16, 24, 32-bit Graphics
- 173 MHz graphics-only max pixel rate for 4 and 8 bits per pixel.
- 85 MHz max Memory Controller Interface (memclk) Clock Rate
- 66 MHz max Register Interface (sysclk) maximum clock rate
- Video YC_rC_b to RGB Color Space Conversion
- YC_rC_b Gamma/Brightness/Contrast Correction RAM (256 x 8)
- Sharpness control
- Timing Generator
- 8514 Interlaced support w/o frame buffer data re-arrangement
- 256 x 18 Color Palette RAM
- 24 bits/pixel output
- Signature Analysis Register

Internal Memory Map

The display refresh module internal register memory map is shown in Table 139. To allow for easy future expansion, reserved locations should never be accessed, and reserved bits/fields within defined registers should always be written with zeros.

Table 139. Internal Register Memory Mapping *(1 of 5)*

Register Group	GBASE 1 00501xxx (dword)	Register Name	Reset to Zero	Description
Color Palette/ Luma Correction Readback	000h–3FCh	Color Palette	N/A	Address h000 to h3FC map to color palette locations 'h00 to 'hFF, respectively. For writing, din[23:18] is red, din[15:10] is green, din[7:2] is blue. When reading, luma correction data is on dout[31:24],red is on dout[23:18], green is on dout[15:10], and blue is on dout[7:2]. Other dout bits are zero.
Configuration	400h	Configuration Register	Yes	Configuration Register. See Table 140 on page 185 for bit mapping.
Graphics, Border Controls	404h	Border Color Index Register	N/A	Border Color Index Register. See Table 158 on page 203 for bit mapping.
	408h	Border Color Register	N/A	True Color Border Register. See Table 157 on page 203 for bit mapping.
	40Ch	Graphics Start Address A Register	N/A	Graphics starting memory address buffer A register. See Figure 26 on page 187 for bit mapping.
	410h	Graphics Start Address B Register	N/A	Graphics starting memory address buffer B register. See Figure 26 on page 187 for bit mapping.
	414h	Graphics Start Address C Register	N/A	Graphics starting memory address buffer C register. See Figure 26 on page 187 for bit mapping.
	418h	Graphics Size Register	N/A	Graphics Height and Width Registers. Bit mapping is shown in Table 141 on page 187.
	41Ch	Graphics Pitch/ Burst Size Register	N/A	Graphics pitch register. Quadword units. Bit mapping is shown in Figure 28 on page 188.
	420h	Signature Analysis Register	N/A	Signature Analysis Register. Special write timings apply. Bit mapping is the same as the true color border register, shown in Table 157 on page 203.

Table 139. Internal Register Memory Mapping *(2 of 5)*

Register Group	GBASE 1 00501xxx (dword)	Register Name	Reset to Zero	Description
Graphics, Border Controls (cont.)	424h	Graphics Horizontal Timing Register	Yes	Graphics Horizontal Assertion & Negation Point Registers. Bit mapping is shown in Table 155 on page 199.
	428h	Graphics Vertical Timing Register	Yes	Graphics Vertical Assertion & Negation Point Registers. Bit mapping is shown in Table 156 on page 199.
	42Ch-43Ch	Reserved	N/A	Reserved, graphics and border controls, write zeros only.
Cursor Controls	440h-448h	Cursor Color Registers	N/A	Cursor Color 0, 1, 2 Registers. Bit mapping is the same as true color border register.
	44Ch	Cursor Start Address Register	N/A	Cursor Start Address Register. Bit mapping is shown in Figure 26 on page 187.
	450h	Cursor Size/ Pitch/Burst Register	N/A	Cursor Height, Width, Pitch, and Burst Size Register. See Figure 29 on page 188 for bit mapping.
	454h	Cursor Horizontal Timing Register	N/A	Horizontal Assertion, Negation Point Registers. Bit mapping is shown in Table 160 on page 205.
	458h	Cursor Vertical Timing Register	N/A	Vertical Assertion, Negation Point Registers. Bit mapping is shown in Table 161 on page 205.
	45Ch-47Ch	Reserved	N/A	Reserved, cursor controls, do not write.

Table 139. Internal Register Memory Mapping *(3 of 5)*

Register Group	GBASE 1 00501xxx (dword)	Register Name	Reset to Zero	Description
Video Plane 1 Controls	480h	Video 1 Control Register	Yes	Video 1 plane controls. See Table 162 on page 206 for bit mapping.
	484h	Video 1 Scale Increment Registers	N/A	Video 1 x, y-scale increment registers.
	488h	Video 1 Scale Initial Registers	N/A	Video 1 x, y-scale initial registers.
	48Ch	Video 1 Y-scale Init B Register	N/A	Video 1 y-scale initial b register.
	490h	Video 1 Color Key Register	N/A	Video plane 1 color key register. Key is in bits 23:0.
	494h	Video 1 Color Key Mask Register	N/A	Video plane 1 color key mask register. Mask is in bits 23:0.
	498h	Video 1A Start Address Register	N/A	Video plane 1A, 1B, 1C start address registers. See Figure 26 on page 187 for bit mapping.
	49Ch	Video 1B Start Address Register	N/A	
	4A0h	Video 1C Start Address Register	N/A	
	4A4h	Video 1 Size Register	N/A	Video plane 1 source height and width. See Figure 27 on page 188 for bit mapping.
	4A8h	Video 1 Pitch Register	N/A	Video plane 1 line pitch and burst size registers.
	4ACh	Video 1 Horizontal Timing Register	N/A	See Table 147 on page 196 for bit mapping.
	4B0h	Video 1 Vertical Timing Register	N/A	See Table 152 on page 197 for bit mapping.
	4B4h–4BCh	Reserved	N/A	Reserved, video 1 controls, do not write.

Table 139. Internal Register Memory Mapping *(4 of 5)*

Register Group	GBASE 1 00501xxx (dword)	Register Name	Reset to Zero	Description
Video Plane 2 Controls	4C0h	Video 2 Control Register	Yes	Video 2 plane controls. See Table 162 on page 206 for bit mapping.
	4C4h	Video 2 y-, x-scale Increment Register	N/A	Video 2 y, x-scale increment registers.
	4C8h	Video 2 y-, x-Scale Initial Registers	N/A	Video 2 y, x-scale initial registers.
	4CCh	Video 2 y-scale Initial B Register	N/A	Video 2 y-scale initial B register.
	4D0h	Video 2 Color Key Register	N/A	Video plane 2 color key register.
	4D4h	Video 2 Color Key Mask Register	N/A	Video plane 2 color key mask register.
	4D8h	Video 2A Start Address Register	N/A	Video plane 2A, 2B, 2C start address register. See Figure 26 on page 187 for bit mapping.
	4DCh	Video 2B Start Address Register	N/A	
	4E0h	Video 2C Start Address Register	N/A	
	4E4h	Video 2 Size Register	N/A	Video plane 2 source height and width. See Figure 27 on page 188 for bit mapping.
	4E8h	Video 2 Pitch Register	N/A	Video plane 2 line pitch and burst size registers.
	4ECh	Video 2 Horizontal Timing Register	N/A	See Table 147 on page 196 for bit mapping.
	4F0h	Video 2 Vertical Timing Register	N/A	See Table 152 on page 197 for bit mapping.
	4F4h–4FCh	Reserved	N/A	Reserved. Do not write.

Table 139. Internal Register Memory Mapping *(5 of 5)*

Register Group	GBASE 1 00501xxx (dword)	Register Name	Reset to Zero	Description
Timing Generator Controls	500h	Timing Generator Control Register	Yes	Timing Generator Control Register. See Table 143 on page 193 for bit mapping.
	504h	Horizontal Sync Register	Reset value is non-zero, consistent with 720 x 400 VGA Text Mode. See bit mapping tables for actual reset values.	Reserved, video 1 controls, write zeros only. See Table 144 on page 195 for bit mapping.
	508h	Horizontal Blank Register		Horizontal Blank Assertion, Negation Point Registers. See Table 145 on page 195 for bit mapping.
	50Ch	Horizontal Active Register		Horizontal Active Assertion, Negation Point Registers. See Table 146 on page 195 for bit mapping.
	510h	Vertical Sync Register		Vertical Sync Assertion, Negation Point Registers. See Table 149 on page 196 for bit mapping.
	514h	Vertical Blank Register		Vertical Blank Assertion, Negation Point Registers. See Table 150 on page 197 for bit mapping.
	518h	Vertical Active Register		Vertical Active Assertion, Negation Point Registers. See Table 151 on page 197 for bit mapping.
	51Ch	Horizontal Midline Register	N/A	Second Horizontal Sync Position (used in interlaced mode only). See Table 148 on page 196 for bit mapping.
	520h	Miscellaneous Timing Register	N/A	Y-Counter Interrupt Register. See Table 154 on page 198 for bit mapping.
	524h–538h	Reserved	N/A	Reserved, timing generator controls, do not write.
	53Ch	Timing Generator Y Counter	See Table 153 on page 197.	Timing Generator Position Counter, read-only, do not write. See Table 153 on page 197 for bit mapping.
Reserved	540h–7F4h	Reserved	N/A	Reserved, do not write.
CSC Diagnostic Register	7F8h	CSC Diagnostic Output	N/A	RGB data output from color space converter. Value read is conversion of true color border register contents. Read-only, do not write.
Status	7FCh	Status Register	See Table 142 on page 191.	Read-only, do not write. See Table 142 on page 191 for bit assignments.
Video Luma Correction Table	800h–BFCh	Video Luma Correction Table	N/A	Luma correction write data is mapped from din[7:0]. These addresses are write-only. Luma data may be read back using the color palette access addresses.
Reserved	C00h–FFCh	Reserved	N/A	Reserved, do not write.

Configuration Register DREF Configuration Register bit assignments are shown in Table 140.

Table 140. Configuration Register Bit Assignments *(1 of 2)*

Bit(s)	Field	Reset Value	Description/Notes
31	Soft Reset	1'b0	Software reset, active high. Setting this bit causes a reset of the module, equivalent to port reset. Note that the assertion of this bit will eventually cause itself to be cleared.
30	Automatic Reset Enable (1'b0) Disabled (1'b1) Enabled	1'b0	Placing a logical one in this field causes the fifos, graphics, video, and cursor paths to be reset at the end (ie: leading vertical blank) of all frames.
29:22	Reserved	N/A	Reserved, write zeros only.
21	Interrupt Enable (1'b0) Disabled (1'b1) Enabled	1'b0	
20:19	Graphics Buffer Bump Mode (2'b00) Manual (2'b01) Auto Bump (2'b10) Port Select (2'b11) Reserved	2'b00	In manual mode, the buffer is bumped by a zero to one transition of the graphics buffer bump bit. In port select mode, the g_buf_select port is sampled at the beginning of each frame. In auto bump mode, the buffer is advanced for each frame according to the modulus.
18	Graphics Buffer Bump (0 to 1) Use next start address (1 to 0) Ignored.	1'b0	When the Graphics Buffer Bump Mode is set to manual control, a zero to one transition on this bit causes the memory controller interface graphics start address to be modified as determined from the bump modulus field.
17:16	Graphics Buffer Bump Modulus (2'b00) Always use start address A (2'b01) Use A & B start addresses only (2'b10) Use A, B, C start addresses	2'b00	
15	Signature Analysis Enable (1'b0) Disabled (1'b1) Enabled	1'b0	Signature Analysis Enable. To produce predictable SAR results, this bit should be changed only during blanking. Special write timings apply.
14:12	Sharpness Correction Coefficient.	3'b000	Field decode is shown in Table 168 on page 217. Reset value will be passthru.
11	Reserved	N/A	Reserved, write zero only.
10	Video Plane Priority (1'b0) Video Plane 1 has higher priority (1'b1) Video Plane 2 has higher priority	1'b0	When both video planes are enabled for display, and the priority can't be othersised determined by chroma or color keying, this bit determines which plane to display.
9	Reserved	N/A	Reserved, write zero only.
8	YC_rC_b Luma Correction Enable (1'b0) No Luma correction applied (1'b1) YC_rC_b Luma correction enabled	1'b0	A logical one causes the luma of YC_rC_b video modes to undergo correction via lookup table.

Table 140. Configuration Register Bit Assignments *(2 of 2)*

Bit(s)	Field	Reset Value	Description/Notes
7	Graphics Enable (1'b0) Disable (1'b1) Enable	1'b0	This bit should be changed only during the front porch of the vertical blanking interval.
6	Graphics RGB Format In 16 bit/pixel mode: (1'b0) 5-6-5 (1'b1) 5-5-5 In 24 bit/pixel mode: (1'b0) Planar RGB, 32 bits/pixel (1'b1) Packed RGB, 24 bits/pixel	1'b0	
5:4	Graphics Pixel Depth (2'b00) 8 bits/pixel (2'b01) 4 bits/pixel (2'b10) 16 bits/pixel (2'b11) 24 bits/pixel	2'b00	
3	Reserved	1'b0	Reserved, write zero only.
2:1	Cursor Mode Select (2'b00) Cursor Disabled (2'b01) Three-Color Cursor (2'b10) Two-Color/Highlight Cursor (2'b11) Two-Color/X-Windows Cursor	2'b00	This bit should be changed only during the front porch of the vertical blanking interval.
0	True Color Border (1'b0) Use Border Color Index Register (1'b1) Use Border Color Register	1'b0	

Memory Control Interface Function

The memory controller interface function is responsible for making memory requests for graphics, video, and cursor data. Fifo outpointers generated in the pixel clock domain are gray-code converted for transfer to the memory controller interface for indicating when the Fifo's are ready to accept new data.

On each frame, at the beginning of the line immediately preceding the first active line of the frame, the memory controller interface function checks the "graphics enable", "video 1 enable", "video 2 enable" and "cursor fetch enable" bits to determine what data is required for each screen. These bits are not re-examined until the next frame.

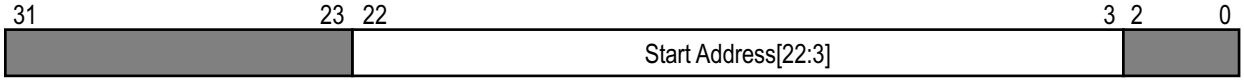
At this point, a full frame of data will be fetched, after which the memory controller interface function will stop fetching all data until the appropriate time in the next frame.

The register file provides the starting and ending address of the memory data, the count, and the line-to-line pitch of the memory data, and the burst size. All data must begin on a quad-word boundary.

Start Address Registers

The bit mapping for the Graphics A/B/C Start Address, Video Plane 1 and 2 A/B/C Start Address, and Cursor Start Address Registers are show in Figure 26.

Figure 26. Graphics/Video/Cursor Start Address Register Bit Mapping



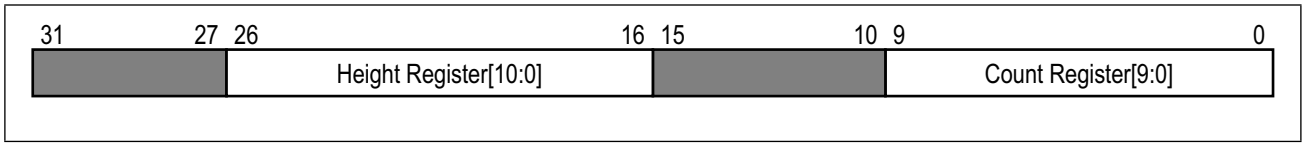
The bit mapping for the Graphics Size Register is shown in Table 141.

Table 141. Graphics Size Register

Bit(s)	Field	Reset	Description
31	Graphics Line Doubling Enable (1'b0) Disabled (1'b1) Enabled	N/A	When this bit is a logical one, each line of graphics data will be fetched twice. Only half as much data will be required for graphics.
30:27	Reserved	N/A	Reserved, write zero only.
26:16	Graphics Height[10:0]	N/A	Number of source graphics lines.
15:10	Reserved	N/A	Reserved, write zero only.
9:0	Graphics Count[9:0]	N/A	Number of source quadwords per graphics line.

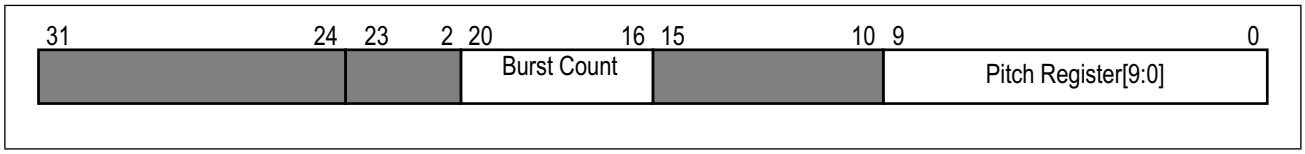
As is the case with all other count-holding registers, the values programmed for the graphics height and count should be one less than the actual amount desired. The bit mapping for the Video Size Register is shown in Figure 27. The count, burst count and height values are programmed to be 1 less than the actual desired amount (eg.: for a burst count of 16, the value 'hF should be loaded).

Figure 27. Video Size Register Bit Mapping



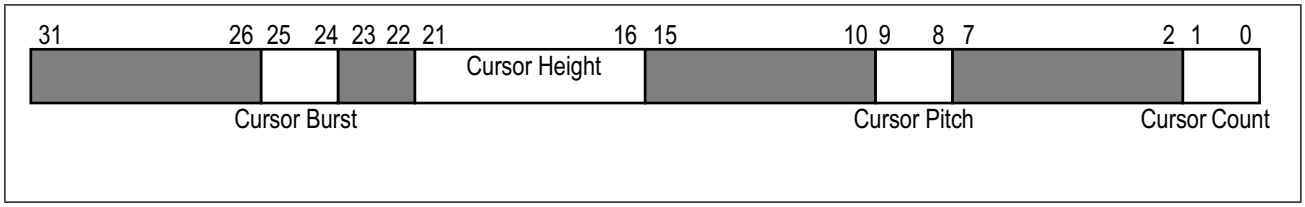
The bit mapping for the Graphics and Video Pitch Registers is shown in Figure 28.

Figure 28. Graphics and Video Pitch and Burst Count Register Bit Mapping



The mapping of the Cursor Size Register is shown in Figure 29.

Figure 29. Cursor Size Register Bit Mapping



Vertical Upscale Handling

The memory controller interface function contains two DDA's for determing when to repeat video 1 and video 2 line fetching. For each line of video data fetched, the increment value is added to the current DDA accumulator value, if a carry out is generated, then the next video source line will be subsequently fetched, otherwise the current line will be repeated.

In interpolated vertical upscaling, the memory controller interface fetches adjacent video source line pairs, with the upper line going into video fifo 1 and the lower line going into video fifo 2 (in this mode only one video window may be displayed).

Interlaced Mode Handling for Graphics

In interlaced mode, the memory controller interface fetches graphics data as required by the current field (even or odd). For even fields, graphics data is fetched starting at the location specified by the graphics start address register; at the end of each line, twice the pitch value is added to obtain the starting point for each subsequent graphics line of data. For odd fields, the uppermost line of graphics if fetched from the (start address + pitch) location. The line to line increment is twice the pitch value as is true for even fields.

Interlaced Mode Handling for Cursor

For cursor data, the first line to fetch depends on both odd_field and the vertical cursor position.

Graphics Line Doubling

When this bit is enabled in the Graphics Size Register, each line of graphics data will be fetched and displayed twice. All registers must be programmed for the full size of the doubled data, but only half the amount of graphics will be used.

Register Interface Function

The register interface function provides internal register read/write, and color palette write, and luma correction table write functionality.

Read Cycles

Reads are asynchronous, and read data output is available combinationally from select active, and a valid internal address presented. Data is not guaranteed valid until reg_ack is active through a rising sysclk. The addr, select, and read inputs must be stable throughtout the read cycle, until the reg_ack is asserted. Select must be inactive for at least one clock between register interface accesses.

When not selected, din[31:0] is driven to dout[31:0] to allow for daisy chained module read capability. Reads from undefined or write-only locations will yield dout[31:0] equal to zero. A read cycle timing diagram is shown in Figure 42 on page 219.

Write Cycles

Having select active with read low through a rising sysclk edge will initiate a write cycle using the din[31:0] data to be written internally at the internal address given by addr[11:0]. The reg_ack signal is used to determine completion of the write cycle. Until reg_ack is asserted, addr, din[31:0], read, and select should be held stable. Dout[31:0] will change to the data written after the rising edge of sysclk. A write cycle timing diagram is shown in Figure 43 on page 219

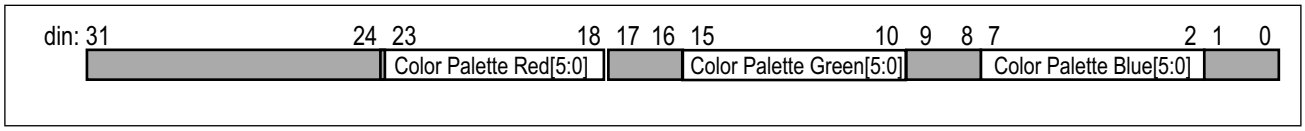
Register Access Timing

Generally, read and write cycles to the registers complete in 2 sysclk cycles, except for reads from the luma/palette ram, and writes to the signature analysis register. Both of these accesses will have delayed reg_ack signals due to synchronization time from sysclk to pclk to sysclk domains.

Color Palette Accessing

The color palette bit mapping is shown in Figure 30.

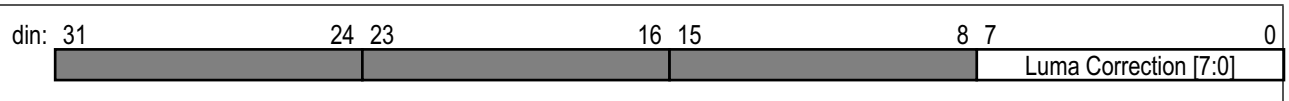
Figure 30. Color Palette Ram Bit Mapping



Luma Correction

Luma correction capability for YC_rC_b video modes is facilitated by providing a 256 x 8 lookup table. The write bit mapping is shown in Figure 31 on page 191. See "YCrCb Luma Correction" on page 210 for more information. The color palette and luma correction data may be read back, although with somewhat different timings from a normal register readback. This is due to the fact that the color palette has only a single read port. Hence, when reading palette or luma data, a cycle must be stolen from the normal pixel path use of the ram. Also, there must be a least one non-ram access sysclk cycle between each ram access cycle. Luma data is read back using the color palette addresses.

Figure 31. Luma Correction Ram Write Bit Mapping



Status Register

The Status Register bit mapping is shown in Table 142.

Table 142. Status Register Bit Mapping *(1 of 2)*

Bit(s)	Field	Reset Value	Description
31	Interrupt	1'b0	Interrupt port state.
30	Vertical Sync	1'b1	Timing Generator is currently within vertical sync interval.
29	Vertical Blank	1'b1	Timing Generator is currently within vertical blanking interval.
28	Horizontal Sync	1'b1	Timing Generator is currently within horizontal sync interval.
27	Horizontal Blank	1'b1	Timing Generator is currently within horizontal blanking interval.
26	Odd Field	1'b1	Timing Generator is currently within odd field (numbering is 1, 2, 1, 2, etc.)
25	Video 1 Fifo Overrun	1'b0	Video 1 Fifo has overrun since last reset.
24	Video 1 Fifo Underrun	1'b0	Video 1 Fifo has underrun since last reset.
23	Video 2 Fifo Overrun	1'b0	Video 2 Fifo has overrun since last reset.
22	Video 2 Fifo Underrun	1'b0	Video 2 Fifo has underrun since last reset.
21	Cursor Overrun	1'b0	Cursor Fifo has overrun since last reset.
20	Cursor Underrun	1'b0	Cursor Fifo has underrun since last reset.
19	Graphics Overrun	1'b0	Graphics Fifo has overrun since last reset.
18	Graphics Underrun	1'b0	Graphics Fifo has underrun since last port or soft reset. Unaffected by auto reset.

Table 142. Status Register Bit Mapping *(2 of 2)*

Bit(s)	Field	Reset Value	Description
17:16	Video 2 Current Start Buffer (2'b00) Using Video 2A Start Address (2'b01) Using Video 2B Start Address (2'b10) Using Video 2C Start Address	2'b00	
15	Video 2 Current Y-scale Initial Pointer (1'b0) Using video 2 Y-scale Initial A (1'b1) Using video 2 Y-scale Initial B	1'b0	
14:13	Video 1 Current Start Buffer (2'b00) Using Video 1A Start Address (2'b01) Using Video 1B Start Address (2'b10) Using Video 1C Start Address	2'b00	
12	Video 1 Current Y-scale Initial Pointer (1'b0) Using video 1 Y-scale Initial A (1'b1) Using video 1 Y-scale Initial B	1'b0	
11:10	Graphics Current Start Buffer (2'b00) Using Graphics 1A Start Address (2'b01) Using Graphics 1B Start Address (2'b10) Using Graphics 1C Start Address	2'b00	
9:0	Reserved	10'h000	Reserved

Timing Generator Function

The timing generator is controlled with a series of registers containing various points defining the boundaries for each of the CRT timing conditions and the Timing Generator Control Register. The bit mapping of the Timing Generator Control Register is shown in Table 143.

Table 143. Timing Generator Control Register Bit Mapping

Bit(s)	Field	Reset Value	Description
31:10	Reserved	N/A	Reserved, write zeros only.
9:8	Programming Unit Select (2'b00) Clocks Units (2'b01) 8 Clock Character Units (2'b10) 9 Clock Character Units (2'b11) Reserved	2'b00	
7	Reserved, previously was Clock Source Select (1'b0) Pixel Clock (1'b1) Pixel Clock divided by 2	1'b0	Reserved, write zero only.
6	Force Blank (1'b0) Normal Operation (1'b1) Blanking is forced	1'b0	
5	Vsync Invert (1'b0) Vsync output is active high (1'b1) Vsync output is active low	1'b0	
4	Hsync Invert (1'b0) Hsync output is active high (1'b1) Hsync output is active low	1'b0	
3	Vsync Disable (1'b0) Normal Operation (1'b1) vsync is held inactive	1'b0	
2	Hsync Disable (1'b0) Normal Operation (1'b1) hsync is held inactive	1'b0	
1	Interlaced Mode (1'b0) Non-Interlaced (1'b1) Interlaced	1'b0	When this bit is a logical one, dref will generate vsync and vblank signals consistent with the 8514 industry standard timings. See VESA monitor timing standards, Version 1.0 Release 0.6a[a]
0	Timing Generator Enable (1'b0) Disable (1'b1) Enable	1'b0	While this bit is zero, the timing generator is held in a reset state.

a. ftp://vesaftp@206.79.240.115/export/home/ftp/vendors/vesa/private/standards/dmt.pdf, page 15.

Non-Interlaced Mode

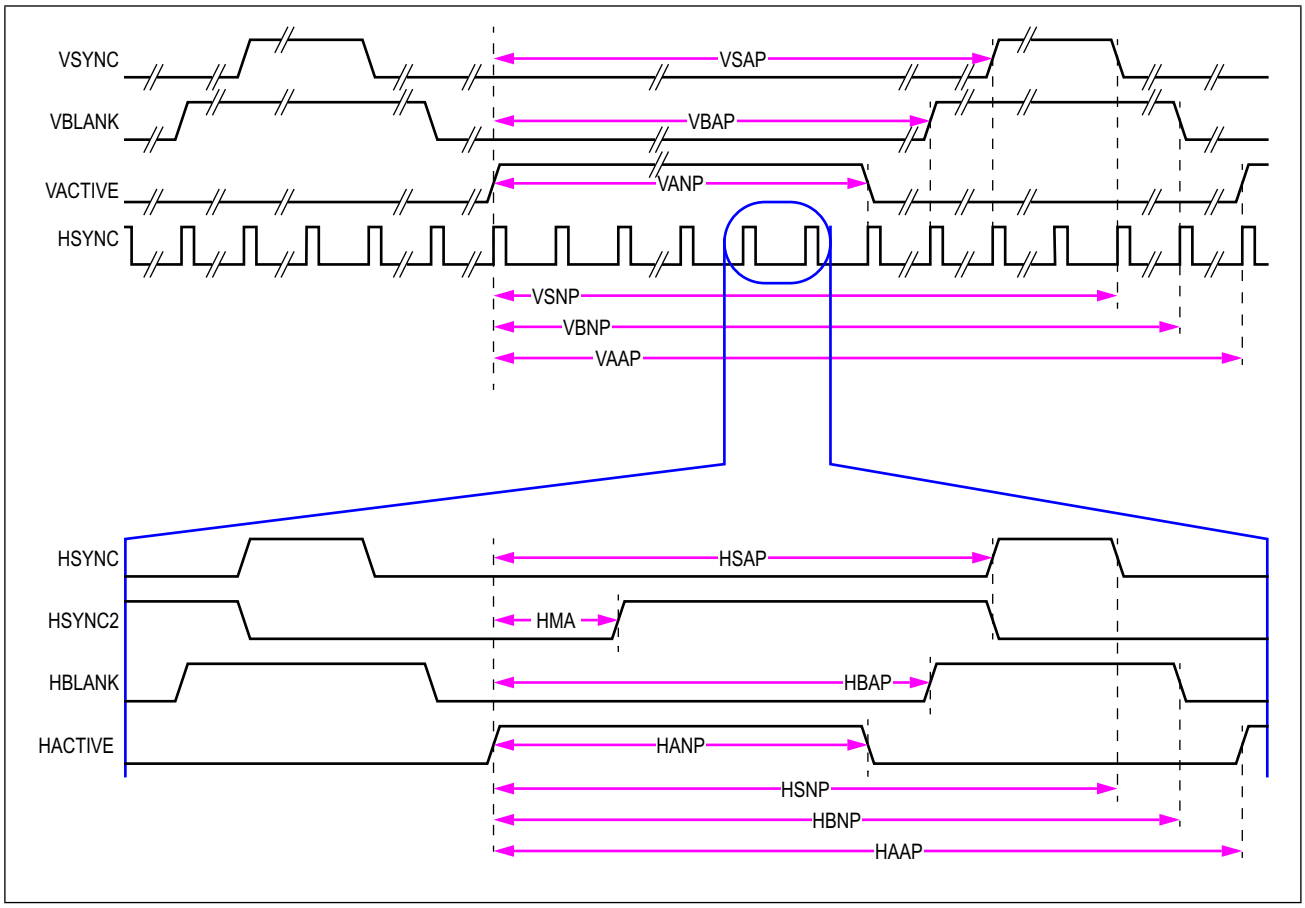
Two counters are used, an x-counter which increments on each pixel clock, and a y-counter which increments on each leading edge of horizontal sync. The x-counter is reset in the pclk cycle following the one that equals the horizontal active assertion pointer. Changes to the vertical state can occur only in the same cycle that the leading edge of hsync occurs.

The changing of each horizontal state of the timing generator requires an equal compare between the x-counter and one of the horizontal register fields occurring in the previous cycle; hence the value programmed into the assertion and negation point registers must be one less than the actual value required. The relationship between programmed registers and timing generator outputs are shown in Figure 32.

On reset, the timing generator registers are set consistent with VGA 720 x 400 resolution text mode.

The odd_field output toggles at each leading edge of vertical blank; the level is of no significance in non-interlaced mode; but will toggle from from frame to frame.

Figure 32. Timing Generator Waveforms, Non-interlaced



Horizontal Timing Generation

Horizontal timings are determined by the programming of the Horizontal Sync Negation Point (HSNP), Horizontal Sync Assertion Point (HSAP) (See Table 144), Horizontal Blank Negation Point (HBNP), Horizontal Blank Assertion Point (HBAP) (See Table 145), Horizontal Active Assertion Point (HAAP), Horizontal Active Negation Point (HANP) (See Table 146), Horizontal Video Assertion Point (HVAP), Horizontal Video Negation Point (HVNP) (See Table 147), and Horizontal Midline Assertion Point (HMAP) (See Table 148) Registers. Horizontal register values are programmed in timing units specified by the programming unit select. The reference position is the horizontal active assertion point. Except as noted below, the HAAP register reflects the line length.

Table 144. Horizontal Sync Register

Bit(s)	Field(s)	Reset Value	Description
31:28		N/A	Reserved, write zeros only
27:16	HSAP	'd773	Horizontal Sync Assertion Point
15:12		N/A	Reserved, write zero only
11:0	HSNP	'd881	Horizontal Sync Negation Point

Table 145. Horizontal Blank Register

Bit(s)	Field(s)	Reset Value	Description
31:28		N/A	Reserved, write zeros only
27:16	HBAP	'd728	Horizontal Blank Assertion Point
15:12		N/A	Reserved, write zero only
11:0	HBNP	'd890	Horizontal Blank Negation Point

Table 146. Horizontal Active Register

Bit(s)	Field(s)	Reset Value	Description
31:28		N/A	Reserved, write zeros only
27:16	HAAP	'd895	Horizontal Active Assertion Point
15:12		N/A	Reserved, write zeros only
11:0	HANP	'd719	Horizontal Active Negation Point

Table 147. Horizontal Video Register

Bit(s)	Field(s)	Reset Value	Description
31:28		N/A	Reserved, write zeros only
27:16	HVAP		Horizontal Video Assertion Point
15:12			Reserved, write zero only
11:0	HVNP		Horizontal Video Negation Point

Table 148. Horozintal Midline Register

Bit(s)	Field(s)	Reset Value	Description
31:28	Reserved, write zeros only		
27:16	HMAP	'd899	Horizontal Midline Sync Assertion Point
15:0	Reserved, write zeros only		

On vertical sync lines, the Alternate Horizontal Active Assertion (AHAAP) register is used for the line length. The bit mapping for this register is shown in Table 154 on page 198.

Vertical Timing Generation

Vertical timings are determined by the programming of the Vertical Sync Negation Point (VSNP), Vertical Sync Assertion Point (VSAP) (See Table 149), Vertical Blank Assertion Point (VBAP), Vertical Blank Negation Point(VBNP) (See Table 150), Vertical Active Assertion Point (VAAP), Vertical Active Negation Point (VANP) (See Table 151), Vertical Video Assertion Point (VVAP), and Vertical Video Negation Point (VVNP) (See Table 152) registers. In non-interlaced mode, these registers are programmed in units of horizontal lines. The reference point for the vertical registers is the vertical active assertion point. Hence the VAAP register reflects the total number of lines that the frame contains.

Table 149. Vertical Sync Register

Bit(s)	Field(s)	Reset Value	Description
31:28		N/A	Reserved, write zeros only
27:16	VSAP	'd412	Vertical Sync Assertion Point
15:12		N/A	Reserved, write zero only
11:0	VSNP	'd414	Vertical Sync Negation Point

Table 150. Vertical Blank Register

Bit(s)	Field(s)	Reset Value	Description
31:28		N/A	Reserved, write zeros only
27:16	VBAP	'd406	Vertical Blank Assertion Point
15:12		N/A	Reserved, write zero only
11:0	VBNP	'd441	Vertical Blank Negation Point

Table 151. Vertical Active Register

Bit(s)	Field(s)	Reset Value	Description
31:28		N/A	Reserved, write zeros only
27:16	VAAP	'd448	Vertical Active Assertion Point
15:12		N/A	Reserved, write zero only
11:0	VANP	'd399	Vertical Active Negation Point

Table 152. Vertical Video Register

Bit(s)	Field(s)	Reset Value	Description
31:28		N/A	Reserved, write zeros only
27:16	VVAP		Vertical Video Assertion Point
15:12			Reserved, write zero only
11:0	VVNP		Vertical Video Negation Point

For coding convenience and diagnostic purposes, the value of the y position counter may be read from the timing position register. The bit mapping is shown in Table 153 on page 197. Note that data read from this register is sourced from the pixel clock domain, hence, values read from this register should not be considered reliable until two consecutively read identical values are obtained.

Table 153. Timing Position Register

Bit(s)	Field	Reset Value	Description
31:28	Reserved	N/A	Reserved
27:16	Y Position	12'h19C	Timing Generator current Y position
15:0	Reserved	N/A	Reserved

Interrupts

The interrupt port can be used to externally signal the timing generator y-counter arrival to a programmed value. The desired y-counter value is programmed into the Miscellaneous Timing Register, whose bit mapping is shown in Table 154.

Table 154. Miscellaneous Timing Register

Bit(s)	Field(s)	Reset Value	Description
31:28	Reserved, write zeros only		
27:16	AHAAP	'd895	Alternate Horizontal Active Assertion Point
15:12	Reserved, write zero only		
11:0	YINT	N/A	Y-Counter Interrupt Value

To enable interrupts, the Configuration Register must have the Interrupt Enable bit programmed to a logical one. The duration of the interrupt signal is one full scan line in non-interlaced mode, or a partial scan line in interlaced mode.

Graphics Data Path

Graphics data is fetched from the frame buffer location as determined by the Graphics Start Address Registers and buffer selection controls. Three start address registers are provided, to allow for triple buffered graphics source data. The start address register used is advanced by a zero to one transition of the graphics bump control in the Configuration Register. The graphics bump modulus controls which of the Graphics Start Address Registers participate in the start address selection. A bump modulus of zero indicates that only start address "A" will be used, a bump modulus of one indicates that only start addresses "A" and "B" will be used, and a bump modulus of two indicates that all three start addresses will be used in the buffer cycling. See Table 140 on page 185.

The graphics plane position within the screen active is determined by the values programmed into the Graphics Horizontal & Vertical timing registers. Typically, these registers will be programmed with the same values as the Horizontal & Vertical Active Registers; however, when graphics data doesn't fill the screen, as may be the case with fullscreen or near fullscreen video, these registers may be programmed to a relatively small area of the screen to reduce the graphics bandwidth required. The bit mapping of the Graphics Horizontal Position registers is shown in Table 155.

Table 155. Horizontal Graphics Position Register

Bit(s)	Field(s)	Reset Value	Description
31:28	Reserved, write zeros only		
27:16	HGAP	'd899	Horizontal Graphics Assertion Point
15:12	Reserved, write zero only		
11:0	HGNP	'd719	Horizontal Graphics Negation Point.

The bit mapping of the Graphics Vertical Position registers is shown in Table 156.

Table 156. Vertical Graphics Position Register

Bit(s)	Field(s)	Reset Value	Description
31:28	Reserved, write zeros only		
27:16	VGAP	'd448	Vertical Graphics Assertion Point
15:12	Reserved, write zero only		
11:0	VGNP	'd399	Vertical Graphics Negation Point.

The graphics formats supported are 4 and 8 bit pseudocolor; and 5-5-5, 5-6-5, and 8-8-8 true color packed and planar. The true color modes bypass the color palette and are scaled to achieve full scale.

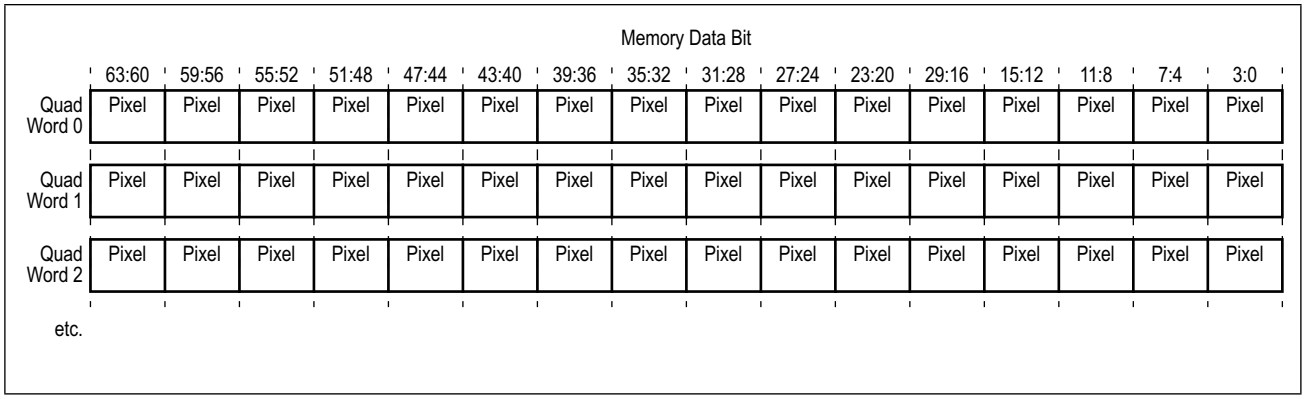
Each line of graphics data must start on a quad-word boundary. The number of quad words per line and the number of lines per frame are programmed into the Graphics Size Register; see Table 141 on page 187 for the bit mapping.

The last quad word of each graphics line may have partial usage. The count register may specify additional unused quadwords; however, these quadwords will cause additional memory bandwidth to be used and will require blanking time to allow for the unloading of this data from the graphics path.

4-bit Pseudocolor

The data format for this mode is shown in Figure 33.

Figure 33. 4-Bit Pseudocolor Graphics Data Format

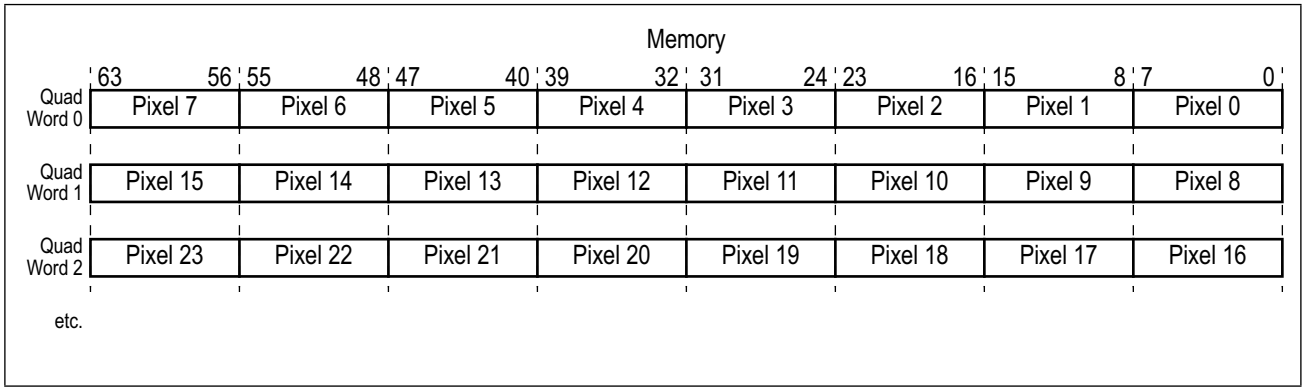


Each 4-bit pixel is used to address an 18-bit RGB value (6 bits/channel) in the color palette, only the first 16 locations of the palette are used. Each 6-bit channel value read from the palette is then scaled to 8 bits. Only the first 16 locations of the color palette are used in the 4-bit/pixel mode. The number of available simultaneous colors is 16 from a palette of 256k colors.

8-bit Pseudocolor

The memory format for this mode is shown in Figure 34 on page 200. Each 8-bit pixel is used to address an 18-bit RGB value (6 bits/channel) in the color palette ram. Each 6-bit channel is then scaled to 8 bits. All 256 locations may be used. The number of available simultaneous colors is 256 from a palette of 256k colors.

Figure 34. 8-Bit Pseudocolor Graphics Data Format

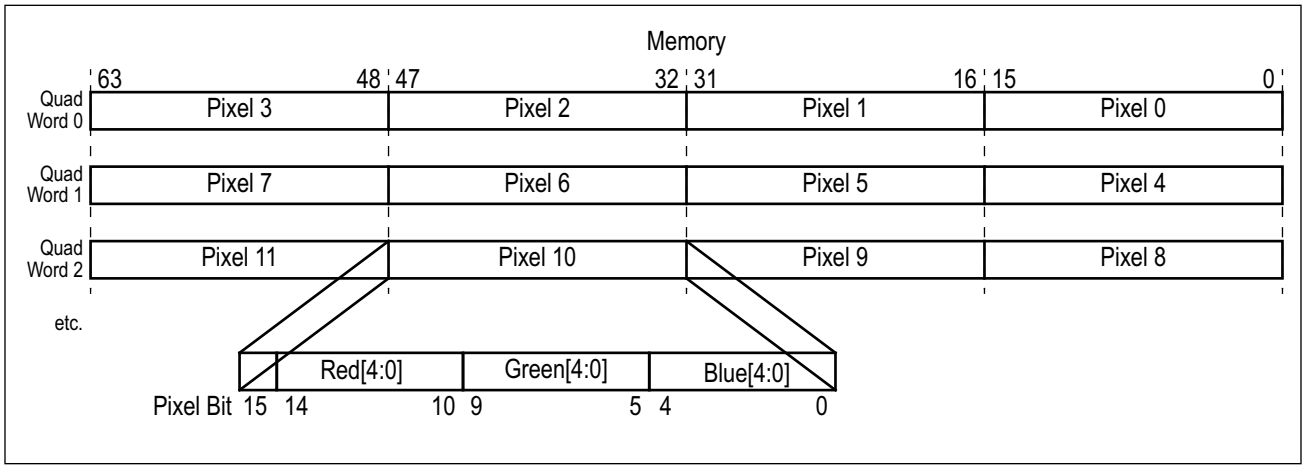


5-5-5 Truecolor

The memory format for this mode is shown in Figure 35. Each 5-bit channel is full-scaled to 8-bits by appending the 3 most significant on the least significant side as shown in the following verilog expressions:

```
assign ch_out[ 7:0] = { ch_in[ 4:0], ch_in[ 4:2]};
```

Figure 35. 16-Bit 5-5-5 Truecolor Graphics/Video Data Format

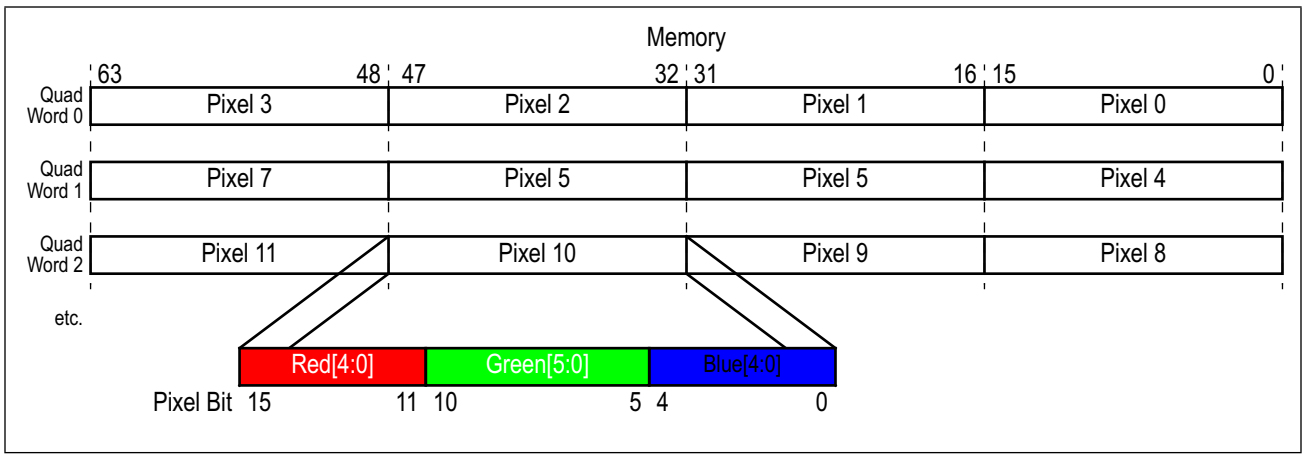


5-6-5 Truecolor

The memory data format for this graphics mode is shown in Figure 36 on page 201. The 5-bit Red and Blue channels are full-scaled as shown for the 5-5-5 format. Scaling for the green 6-bit channel to 8-bit output channel is similiarly accommplished, except that only the two most significant bits are used:

```
assign grn_out[ 7:0] = { grn_in[ 5:0], grn_in[ 5:4]};
```

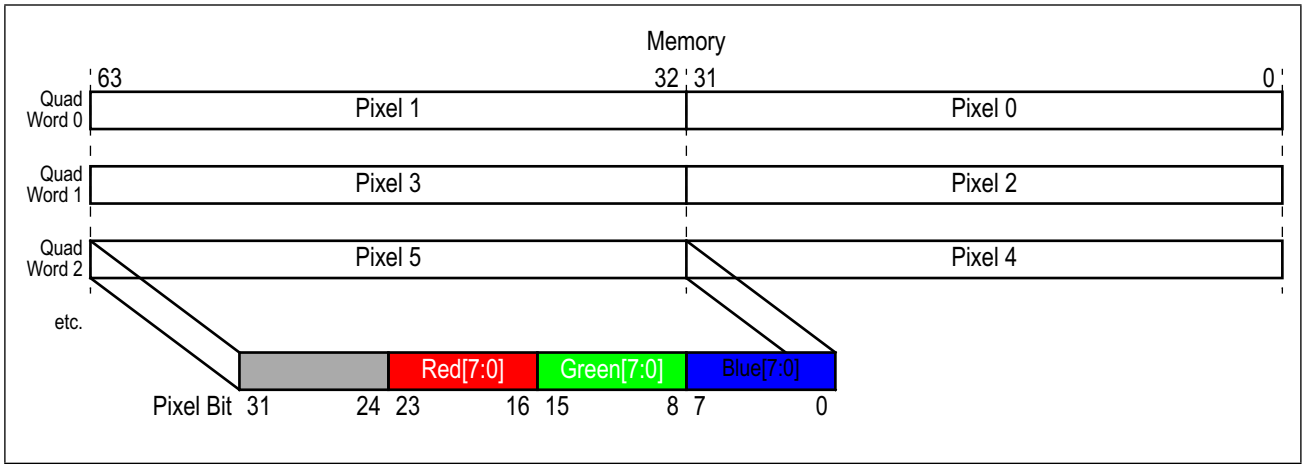
Figure 36. 16-Bit 5-6-5 Truecolor Graphics/Video Data Format



8-8-8 Truecolor, Planar

The memory format for this mode is shown in Figure 37. The color palette is not used in this mode.

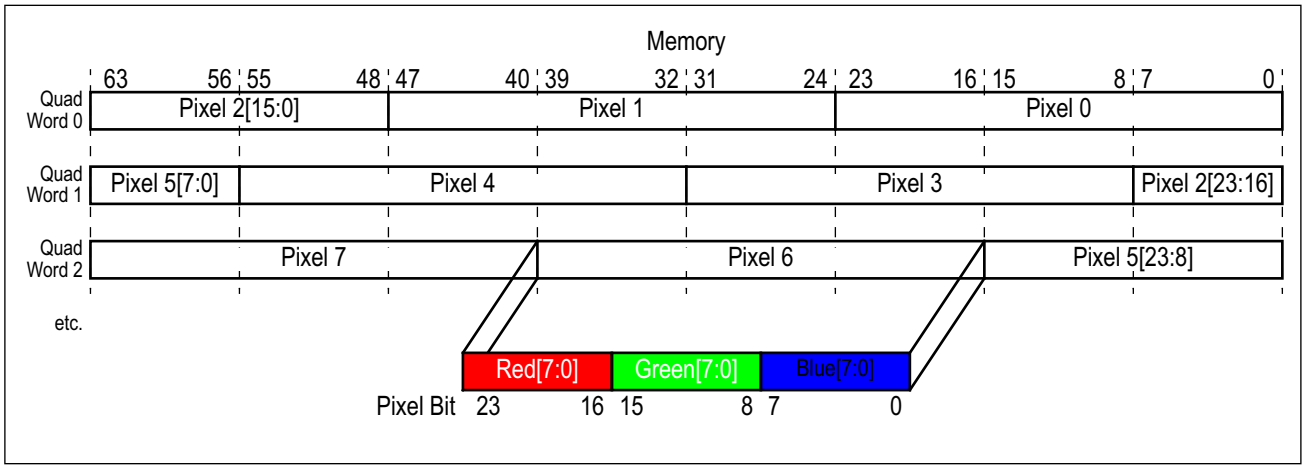
Figure 37. 32-Bit Planar 8-8-8 Truecolor Graphics/Video Data Format



8-8-8 Truecolor, Packed

The memory format for this graphics mode is shown in Figure 38 on page 202. The color palette is not used in this mode.

Figure 38. 24-Bit Packed 8-8-8 Truecolor Graphics Data Format



Border Color Controls

The border color may be selected via an index into the color palette, or a dedicated 24-bit true-color border may be used. The Border Color Register is shown in Table 157.

Table 157. Border/Cursor Color/SAR Register Bit Mapping

Bits(s)	Field	Description
31:24	Reserved	Reserved, write zero only.
23:16	Red[7:0]	Red Field
15:8	Green[7:0]	Green Field
7:0	Blue[7:0]	Blue Field

The Border Color Index Register is shown in Table 158.

Table 158. Border Index Register Bit Mapping

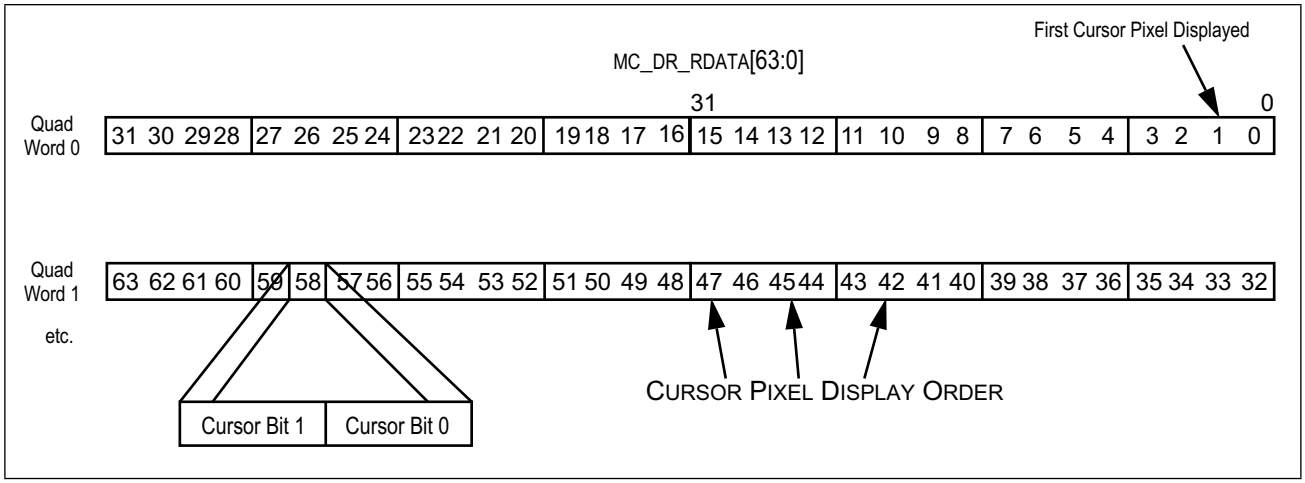
Bit(s)		
31:8	Reserved	Reserved, write zero only.
7:0	Palette Index [7:0]	

Cursor Data Path

The cursor is positioned on the screen by loading the Cursor X and Y Assertion/ Negation Point Registers. The values loaded are compared with the timing generator x- and y-position counters to determine cursor positioning; prior line values may be loaded to move the cursor above or to the left of the active portion of the screen.

The display of the cursor has priority over both video planes and graphics plane within the active region. The cursor never displays within the border region. The cursor mode is selected by writing the Configuration Register (the bit definitions for this register are shown in Table 140 on page 185). The format for cursor data is shown in Figure 39.

Figure 39. Cursor Memory Format



Cursor Modes

The truth table for the colors displayed in each of the cursor modes is shown in Table 159. For the complement case, the rgb data out would be bit-wise complemented.

Table 159. Cursor Color Determination

Cursor [1:0]	Cursor Mode			
	00	01	10	11
00	Cursor is disabled, i.e., always transparent.	Transparent	Cursor Color 1	Transparent
01		Cursor Color 0	Cursor Color 2	Transparent
10		Cursor Color 1	Transparent	Cursor Color 0
11		Cursor Color 2	Complement	Cursor Color 1

Cursor Positioning

The cursor is positioned by programming the horizontal and vertical cursor assertion, and negation point registers. The assertion point registers determine the starting position for the cursor data; the negation point registers determine the ending position for the cursor data. These registers are programmed in the same fashion that the timing generator assertion/negation point registers are programmed, ie.: the programmed values are one less than the number of pixels/lines away from leading hactive/vactive, respectively. To position the cursor beginning at x-position 0 or above, the cursor horizontal assertion point must be programmed with the graphics active assertion point (ie.: the line length) or above. This also applies to the vertical positioning, when moved to the top edge or above the top edge of the active display.

The positions programmed into these register may lie within a blanking or border region; however cursor data will not be displayed outside the active graphics area. This facilitates the positioning of the cursor beyond the edges of the active graphics areas; however, excessively short blanking intervals (as is normally specified for flat panels) may limit how far the cursor may be moved off-screen w/o causing artifacts on prior or subsequent lines.

Leftover cursor pixels in a quadword beyond the cursor negation point are discarded by the pixel mux logic.

The cursor position should be modified only during cursor inactive time, otherwise cursor path errors may result.

Horizontal Cursor Register

The bit mapping for the fields of the Horizontal Cursor Register are shown in Table 160. The units for the values programmed are pixel clocks relative to the leading edge of horizontal active.

Table 160. Horizontal Cursor Register

Bit(s)	Field(s)	Reset Value	Description
31:28	Reserved, write zero only		
27:16	HCAP	N/A	Horizontal Cursor Assertion Point
15:12	Reserved, write zero only		
11:0	HCNP	N/A	Horizontal Cursor Negation Point

Vertical Cursor Register

The bit mapping for the fields of the Vertical Cursor Register are shown in Table 161. The units for the values programmed are lines relative to the leading edge of vertical active.

Table 161. Vertical Cursor Register

Bit(s)	Field(s)	Reset Value	Description
31:28	Reserved, write zero only		
27:16	VCAP	N/A	Vertical Cursor Assertion Point
15:12	Reserved, write zero only		
11:0	VCNP	N/A	Vertical Cursor Negation Point

Video Data Path

Two video planes are supported. The positioning of these video windows are programmed into the video 1 and 2 vertical and horizontal assertion/negation point registers. The values programmed are relative to the active display position.

Video Control Register

The bit mapping for the video control registers is shown in Table 162. There is one video control register for each video plane.

Table 162. Video Control Register Bit Definition *(1 of 2)*

Bit(s)	Field	Reset Value	Description/Comments
31:15	Reserved	21'h0000	Reserved, write zeros only.
14:13	Video Buffer Bump Mode (2'b00) Manual (2'b01) Auto Bump (2'b10) Port Select (2'b11) Reserved	2'b00	In manual mode, the buffer is bumped by a zero to one transition of the video buffer bump bit. In port select mode, the v1_buf_select or v2_buf_select port is sampled at the beginning of each frame. In auto bump mode, the buffer is advanced for each output frame according to the bump modulus.
12	Video Buffer Bump (0 to 1) Use next start address and initial y-scale value. (1 to 0) Ignored.	1'b0	When the buffer bump mode is set to manual, a zero to one transition on this bit causes the memory controller interface and vertical upscale function to use the subsequent video start address and y-scale initial values as determined from the bump modulus field.
11:10	Video Buffer Bump Modulus (2'b00) Always use start & initial A (2'b01) Use A & B start & initial only (2'b10) Use A, B, C start, A, B initial (2'b11) Reserved	2'b00	Note that while there are three start address for video, there are only two y-scale initial values.
9	Video Plane Enable (1'b0) Disable Video (1'b1) Enable Video	1'b0	

Table 162. Video Control Register Bit Definition *(2 of 2)*

Bit(s)	Field	Reset Value	Description/Comments
8:7	Vertical Upscale Mode (2'b00) No upscaling (2'b01) Use replicative upscaling (2'b10) Use 2-tap interpolative upscaling (2'b11) Use 3-tap filtered upscaling	2'b00	Two-tap and three-tap values are valid for video plane 1 only if video plane 2 is not enabled. 3-tap mode is valid only if graphics is not enabled. For video plane 2, this field must always indicate no upscaling or replicative upscaling.
6:5	Horizontal Interpolate Mode (2'b00) No upscaling (2'b01) Use replicative upscaling (2'b10) Use 2-tap interpolative upscaling (2'b11) Use 3-tap filtered upscaling	2'b00	
4	Chroma Key Enable (1'b0) Disable (1'b1) Enable	1'b0	Chroma keying not supported for video 5-6-5 RGB mode.
3	Color Key Enable (1'b0) Disable (1'b1) Enable	1'b0	When color keying is enabled, video will not be displayed unless the underlying graphics pixel meets the color key criteria. If there is no underlying graphics data, then video will be displayed.
2	Reserved		Reserved, write zero only.
1:0	Format (2'b00) 4:2:2 YC_rC_b (2'b01) 16-bit RGB (5-6-5) (2'b10) 16-bit RGB (5-5-5) (2'b11) 24-bit RGB (8-8-8)	2'b00	This field selects the video format for the video plane. When doing two or three tap vertical upscaling, the video plane 2 format should be the same as the video 1 format.

The video data path supports the display of source video in either of the 16-bit true color graphics formats, the 32-bit true color graphics format, or 4:2:2 video formats.

4:2:2 Video Format

The format for this mode is shown in Table 163.

Table 163. 4:2:2 Video Data Format

Quad-Word Sequence	mc_dr_rdata bit							
	[63:56]	[55:48]	[47:40]	[39:32]	[31:24]	[23:16]	[15:8]	[7:0]
0	Y3	C_r2	Y2	C_b2	Y1	C_r0	Y0	C_b0
1	Y7	C_r6	Y6	C_b6	Y5	C_r4	Y4	C_b4
etc.								

Quad-word 0 represents the first quad word at the the memory location pointed to by the video start address register. The pattern is identical for each quad-word. In the 4:2:2 format, each source video pixel has its own Y (luminance) value; however, chroma values exists only for even pixels. The chroma for odd pixels (ie. C_r and C_b) values are linearly interpolated to produce the video stream shown in Table 164.

Table 164. Interpolated 4:2:2 Video Data Stream

Video Pixel Sequence	Y	4:2:2 Video	
		C_r	C_b
0	Y0	C_r0	C_b0
1	Y1	$0.5\ C_r0 + 0.5\ C_r2$	$0.5\ C_b0 + 0.5\ C_b2$
2	Y2	C_r2	C_b2
3	Y3	$0.5\ C_r2 + 0.5\ C_r4$	$0.5\ C_b2 + 0.5\ C_b4$
4	Y4	C_r4	C_b4
5	Y5	$0.5\ C_r4 + 0.5\ C_r6$	$0.5\ C_b4 + 0.5\ C_b6$
6	Y6	C_r6	C_b6
7	Y7	$0.5\ C_r6 + 0.5\ C_r8$	$0.5\ C_b6 + 0.5\ C_b8$
8	Y8	C_r8	C_b8
etc.			

The least significant bit of luminance (ie.: Y[0]) is used for chroma keying in this mode.

Color Key Using Graphics Source

Video color keying from the graphics stream is supported through the use of the 24-bit Color Key and Mask Registers. When a video plane is selected for display using the color key criteria, the serialized pixel graphics data is compared with the color key. If the graphics pixel bits match a video plane's color key bits in the same positions that the color key mask contains logical ones, then the video pixel will be displayed. The color key mask should contain a zero in those bit positions beyond the pixel size. A color key mask containing zero in all bits will match any graphics data.

Chroma Key Using Video Source

Chroma keying from the video source is supported for the 24-Bit True Color, the 5-5-5 16-Bit True Color, and the YC_rC_b 4:2:2 video formats. When the chroma key in the video stream is a logical one, then video is displayed; when the chroma key in the video stream is a logical zero, then the video stream is transparent.

If both chroma and color keying are enabled for a video plane, both criteria must be met for the video pixel to be opaque, otherwise it is transparent.

Display Priority

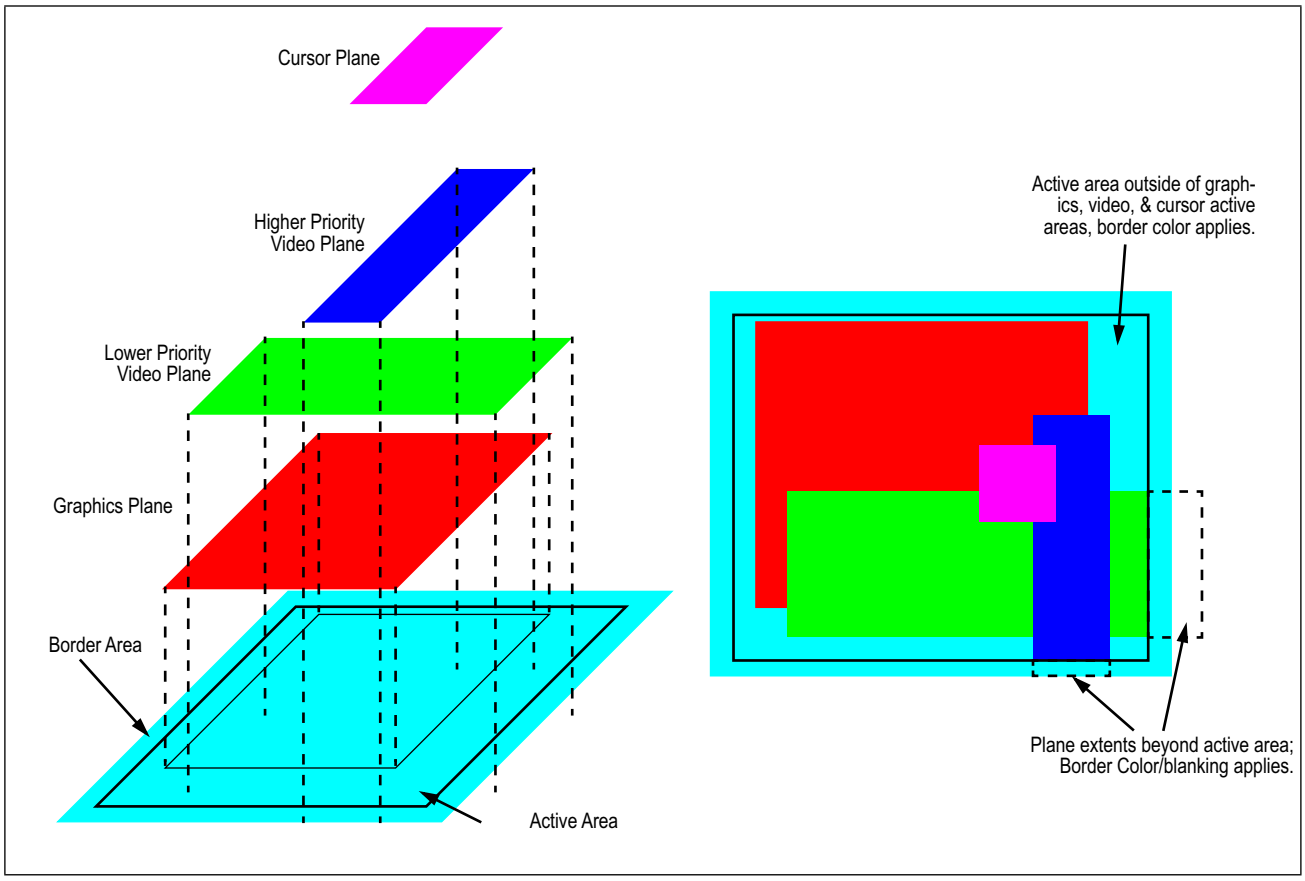
Within the active area of the screen, the data displayed may come from the border color specification or graphics, video 1, video 2, or cursor planes. The cursor plane has highest display priority within the active screen area. Notwithstanding color keying and chroma keying, the video planes always have priority over the graphics plane but are always below the cursor plane. Display priority between the two video planes, when color and/or chroma keying cannot otherwise determine it, is selected by the setting of the video priority bit in the Configuration Register.

The display priority mechanism is graphically depicted in Figure 40 on page 210. This figure shows the resulting display given that color and chroma keying are disabled for both video planes. Also note the following:

- Video pixels which don't overlay the graphics plane always meet the colorkey match criteria.
- Graphics, Video, and/or Cursor Pixels are never displayed outside of the active area; however, the data is still fetched and passed through the pipelines, so the excess data must be containable to the blanking interval.
- Active area pixels which are not within the graphics, video, or cursor planes will display as the border color.

A flowchart showing the video plane selection criteria over a graphics plane for a given video plane priority, chroma, and color keying mode and criteria is shown in Figure 41 on page 215.

Figure 40. Display Priority



YC_rC_b Luma Correction

Luma correction comprises gamma, brightness, and contrast.

Gamma Correction: $Y' = Y^{(1/\gamma)}$
Brightness Correction: $Y' = Y + k$
Contrast Correction: $Y' = k\,Y$

where Y is the normalized (ie.: 0 to 1 valued) pre-correction luma value, and Y' is the normalized corrected luma.

The desired correction is formed by concatenating the gamma, brightness, and contrast corrections and storing the results for each input luma in the luma correction table.

YC_rC_b/RGB Color Space Conversions

For the 4:2:2 video formats, the video path converts the interpolated YC_rC_b video data stream to 24 bits of RGB data (8 bits each) compliant with CCIR Recommendation 601-1 as follows:

$$R = 1.164(Y–16) + 1.596(C_r–128)$$
$$G = 1.164(Y–16)–0.813(C_r–128)–0.391(C_b–128)$$
$$B = 1.164(Y–16) + 2.018(C_b–128)$$

Source Count

Since there is automatic clearing of residual quadwords at the end of each line, the number of quadwords provided may exceed the minimum, however, the amount of data to be cleared must be contained within the interval prior to the beginning of the start of the next displayed line of video. The line count, however, may not cause the source plane to exceed the non-blanked area of the screen. Table 165 on page 211 indicates the minimum required number of load groups for each possible video mode of operation.

Table 165. Video Mode Source Count Requirements

Window Width (pixels)	Minimum Quadwords Required per source video line		
	RGB 5-5-5 or 5-6-5	4:2:2 YC_rC_b	RGB 8-8-8
1	1	1	1
2	1	1	1
3	1	1	2
4	1	2	2
5	2	2	3
6	2	2	3
7	2	2	4
8	2	3	4
9	3	3	5
10	3	3	5
:	:	:	:
n	int[(n+3)/4]	int(n/4) + 1	int[(n+1)/2]
160 (NTSC QCIF)	40	41	80
192 (PAL QCIF)	48	49	96
320 (NTSC CIF)	80	81	160
384 (PAL CIF)	96	97	192
640 (NTSC CCIR601)	160	161	320
768 (PAL CCIR601)	192	193	384

Horizontal Upscaling

Horizontal upscaling may be accomplished in any of the following modes, as select by the Video Control Register:

- Replicative Mode
- 2-Tap Mode
- 3-Tap Mode

Horizontal upscaling is accomplished by using the overflow output of a 12-bit x-scale DDA accumulator to control when to move on to the next pixel or pair of video pixels. At the start of each video output line, the x-scale DDA accumulator is initialized to the x-scale initial value.

For each destination video pixel, the value stored in the x-scale Increment Register is added to the DDA accumulator. If the addition results in a carry, a pixel is unloaded from the respective video stream fi FO. If no carry occurs, the current pixel or pair of pixels will again be used. Both video planes are independently x scaled.

Replicative Horizontal Upscaling

In replicate mode, upscaling is accomplished by repeating source pixels as determined by the x-scale DDA. The DDA increment and initial values must be chosen such that there is sufficient source data provided to prevent video path underrun.

2-Tap Horizontal Upscaling

In 2-tap interpolative upscale mode, the high 3-bits of the DDA are used to select a linear weighing between source pixel pairs. This linear interpolation can yield up to seven unique intermediate values between the pixels being interpolated (also known as 8-phase).

For 4:2:2 and RGB modes, all three channes are linearly interpolated.

In this mode, if chroma keying is enabled, the interpolated pixel will be opaque only if one of the following is true:

- The dda selects 100% of one of the pixels, and the selected pixel is opaque.
- Both pixels being interpolated are opaque.

Lacking both of these conditions will cause the interpolated pixel to be transparent.

3-Tap Horizontal Upscaling

In 3-tap horizontal upscaling, the high 3-bits of the DDA are used to select a linear weighting between the luma values of three adjacent pixels. This mode is valid only for the 4:2:2 video format. The weighting is shown in Table 166 on page 214.

Vertical Upscaling

As is the case with horizontal upscaling, vertical upscaling may be accomplished in any of the following modes:

- Replicative Mode
- 2-Tap Mode
- 3-Tap Mode

To accomplish vertical scaling, the memory controller interface function will automatically reload lines of video as required for vertical upscaling.

Replicative Vertical Upscaling

When upscaling in this mode, the combination of initial and increment values programmed must be such that the number of DDA carry-outs equals the source line count, and the last DDA carry-out occurs while the last source line/pixel is being processed.

2-Tap Vertical Upscaling

When doing 2-tap interpolative video upscaling, only one hardware video window is supported, as the second video path is used for processing the lower line of video line pairs. The video plane 1 path is used for processing the upper line of each interpolated line pair; the video plane 2 path is used for processing the lower line of each interpolated line pair.

The following video plane 1 and 2 registers should contain identical values:

- Video Control Format Field
- Video Size Value
- Video Pitch Values
- Video Count Values
- Video Bump Modulus Values

For interpolative upscaling, the combination of initial and increment values programmed must be such that the number of DDA carry-outs is one less than the number of source line count, and the last DDA carry-out occurs while the last source line/pixel pair is being processed.

The increment value for interpolative upscaling may be computed by using the following expression:

$$\text{Increment} = \frac{4095(\text{SourcePixelCount} - 1)}{\text{DestinationPixelCount}}$$

The source pixel count is the source window height or width, as given by Table 14. The destination pixel count is the size of the onscreen video window plane height or width. The destination pixel count equals the number of pixels for which the video plane extents are defined.

As is true for horizontal 2-tap upscaling, seven intermediate steps of interpolation are provided for all of the video formats; all channels are linearly interpolated in RGB formats. In YC_rC_b formats only luminance is interpolated; chrominance is replicated from top to bottom.

3-Tap Vertical Upscaling

As in 2-tap mode, the video plane 2 path is used for processing the "second" line. The graphics fifo provides the lowest line of a line triad; hence, graphics may not be displayed when doing 3-tap vertical upscaling.

As is true for 3-tap horizontal upscaling, this mode is valid only for 4:2:2 video. The weighting is identical to the weighting shown for horizontal 3-tap upscaling, shown in Table 166 on page 214.

Upscale Initial and Increment Computation

The system needs to pre-calculate and provide the DDA constants required for the desired scale factor and initial starting values and load these values into the appropriate X and Y increment and initial registers.

The amount of video data provided for any of the video modes must be exactly enough to provide any required data used either directly, or partially (i.e. interpolated) by the video logic to generate the video data.

The weights for video scaling for a given DDA accumulator value are shown in Table 166.

Table 166. Video Pixel 2-Tap & 3-Tap Upscaling Weights

DDA Accumulator Bits [11:9]	2-tap Vertical Upscaling	3-Tap Vertical Upscaling	2-tap Horizontal Upscaling	3-tap Horizontal Upscaling
000	V_u	V_u	P_n	P_n
001	7/8 V_u[a] + 1/8 V_m[b]	13/16 V_u + 1/4 V_m – 1/16 V_l[c]	7/8 P_n[d] + 1/8 P_{n+1}[e]	13/16 P_n + 1/4 P_{n+1} – 1/16 P_{n+2}[f]
010	3/4 V_u + 1/4 V_m	5/8 V_u + 17/32 V_m – 5/32 V_l	3/4 P_n + 1/4 P_{n+1}	5/8 P_n + 17/32 P_{n+1} – 5/32 P_{n+2}
011	5/8 V_u + 3/8 V_m	3/8 V_u + 7/8 V_m – 1/4 V_l	5/8 P_n + 3/8 P_{n+1}	3/8 P_n + 7/8 P_{n+1} – 1/4 P_{n+2}
100	1/2 V_u + 1/2 V_m	3/16 V_u + 9/8 V_m – 5/16 V_l	1/2 P_n + 1/2 P_{n+1}	3/16 P_n + 9/8 P_{n+1} – 5/16 P_{n+2}
101	3/8 V_u + 5/8 V_m	9/64 V_u + 71/64 V_m – 1/4 V_l	3/8 P_n + 5/8 P_{n+1}	9/64 P_n + 71/64 P_{n+1} – 1/4 P_{n+2}
110	1/4 V_u + 3/4 V_m	3/32 V_u + 17/16 V_m – 5/32 V_l	1/4 P_n + 3/4 P_{n+1}	3/32 P_n + 17/16 P_{n+1} – 5/32 P_{n+2}
111	1/8 V_u + 7/8 V_m	3/64 V_u + 33/32 V_m – 5/64 V_l	1/8 P_n + 7/8 P_{n+1}	3/64 P_n + 33/32 P_{n+1} – 5/64 P_{n+2}

a. V_u is upper line channel data
b. V_m is lower (2-tap), or middle (3-tap) video line channel data
c. V_l is the bottom (3-tap) video line channel data
d. P_n is the earlier (left) video pixel channel value
e. P_{n+1} is the later (2-tap, right pixel), or middle (3-tap) video pixel channel value
f. P_{n+2} is the latest (rightmost) 3-tap video pixel channel value

Vertical Upscaling Interlaced Format Considerations

In Interlaced format, the vertical upscaling DDA is incremented by twice the increment value for each output display line. Also, for 2-tap mode, the initial value is applied to the first pair of lines on even fields; the initial value plus increment is used on the first pair of lines on odd fields.

5-5-5 True Color Video Format

This format is identical to the 5-5-5 True Color Graphics format shown in Figure 35 on page 201. Chroma keying from the video source is accomplished by using pixel bit 15. A logical one in bit 15 causes the video data to be displayed; a logical zero causes the video pixel data to be transparent.

5-6-5 True Color Video Format

This format is identical to the 5-6-5 True Color Graphics format shown in Figure 36 on page 201. Chroma keying is not supported in this format.

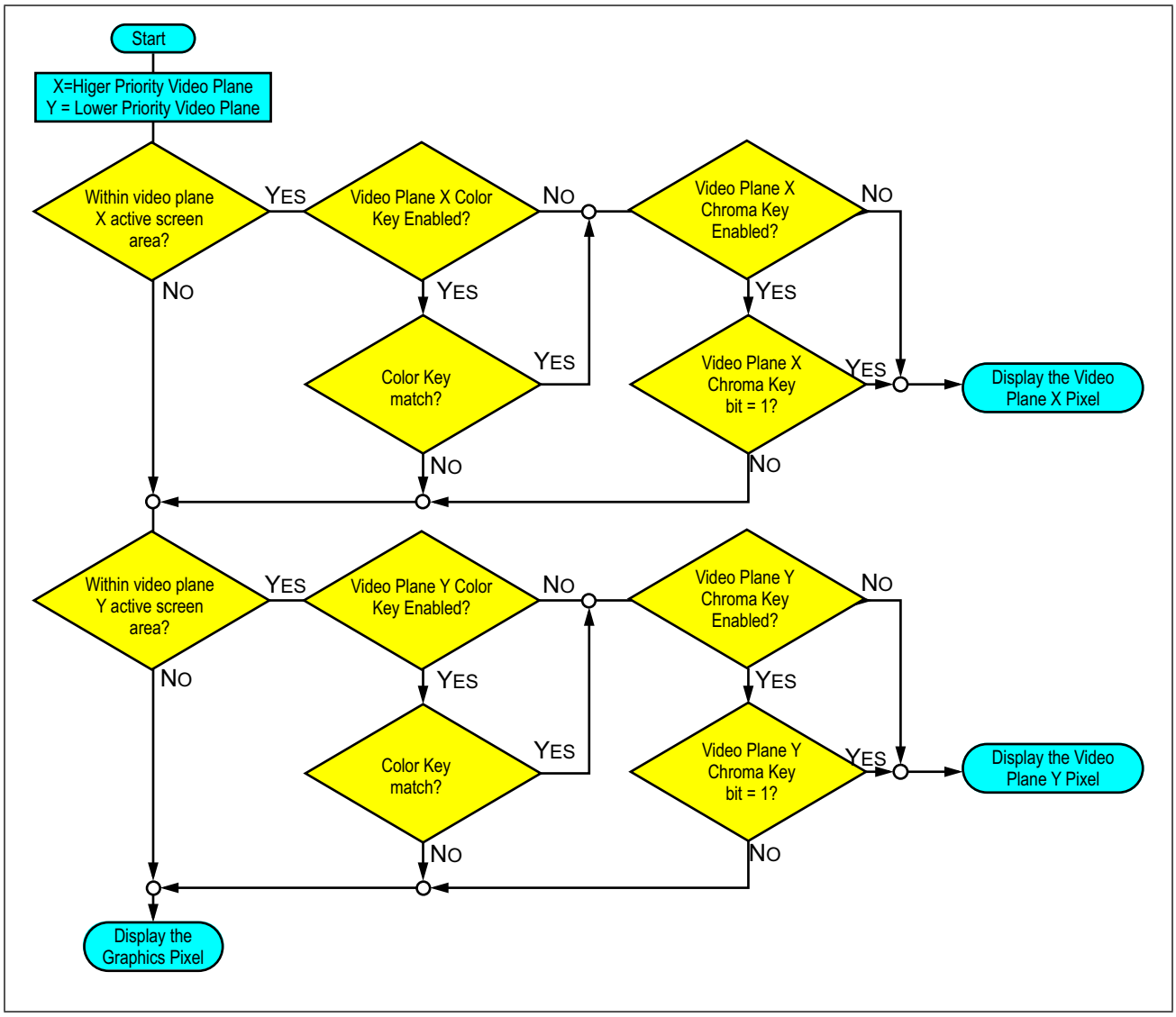
24-Bit True Color Video Format

The data format is identical to the 32-bit planar graphics format illustrated in Figure 37 on page 202, with the added bit 31, used to control chroma keying from the video source. A logical one in bit 31 causes the video data to be displayed; a logical zero causes the graphics pixel data to be displayed.

This format may be used to support a 24-bit True Color cursor. To use the video plane as a true color cursor, the video source should be configured in chroma key enabled mode, with color keying disabled. This will allow transparent showing of underlying graphics data. The video format selected should be a chroma key capable mode, or the video cursor will not be transparent anywhere within its window. The positioning of the video sourced cursor is controlled by the positioning of the video active register.

The overall display selection logic is shown in flowchart form in Figure 41.

Figure 41. Video Plane/Graphics Pixel Source Selection Flowchart



Video Start Buffer and Vertical Upscale Initial Management

Each video plane may be configured as having up to three start buffers within frame buffer memory. All three of these buffers share all other video plane characteristics; ie.: pitch, count, format, etc. The number of buffers is controlled by setting the bump modulus in the corresponding video plane's Video Control Register, see Table 162 on page 206. For a single video plane buffer, only start address A is used; for a double video plane buffer, start addresses A and B are used. For a triple buffered video plane, all three (ie.: A, B, and C) are used.

To request that the subsequent buffer be used, the corresponding video plane's "bump" signal (also contained within the video control register) should be transitioned from zero to one. All buffer changes take effect on the subsequent display frame. The currently active buffer can be determined by reading the Status Register (see Table 142 on page 191).

The vertical upscale initial value is double bufferred; the value to be used is "bumped" in the same manner as the start address buffer; however, at most two values are cycled through. For example, if video plane 1 is triple buffered, the sequence of video start addresses and initial values to be used is shown in Table 167.

Table 167. Triple-Bufferred Video Buffer Bump Example

Bump Transition	Video Start Buffer	Vertical Upscale Initial Value
0 to 1	A	A
0 to 1	B	B
0 to 1	C	A
0 to 1	A	B
0 to 1	B	A
etc.		

When doing two or three tap vertical video upscaling, the start buffers must be identically managed for all paths used.

Sharpness Control Function

- Controlled by bits in the Configuration Register. See Table 140 on page 185.
- Fullscreen only
- Applies to all data types: graphics, video, cursor
- Acts independently on red, green, and blue channel data.
- Filtering is done in x-direction only.
- Filter is as follows:

$$H(z) = \frac{b_0 z^{-1} + b_1 z^{0} + b_2 z^{1}}{a_0}$$

where z^{-1} is the next pixel, z^{0} is the current pixel, and z^{1} is the previous pixel. Coefficients are chosen by programming the sharpness control field in the configuration register as follows:

Table 168. Sharpness Coefficients

Control Bit(s)	Coefficients				Gain (dB)	
	b_0	b_1	b_2	a_0	@1/4 fullscale	@1/2 fullscale
000	0	1	0	1	0	0
001	1	2	1	4	-6	-infinity
010	1	6	1	8	-2.5	-6
011	Reserved					
100	-1	10	-1	8	2	3.5
101	-1	6	-1	4	3.5	6
110	-1	4	-1	2	6	9.5
111	-1	3	-1	1	9.5	14

When sharpness filtering is enabled (ie. the control bits are non-zero), filtering is applied to all pixels having adjacent pixels within the active area.

Signature Analysis Register

When enabled, the signature analysis (SAR) register operates on the 24-bits of output data. The SAR acts as a wide linear feedback shift register on each succeeding output pixel. Signatures are updated for each pixel occurring during the active display time (ie. blank is negated); this includes border-colored pixels.

The SAR is to be used for diagnostic purposes; it should not be enabled for normal operation, as power consumption increases when enabled.

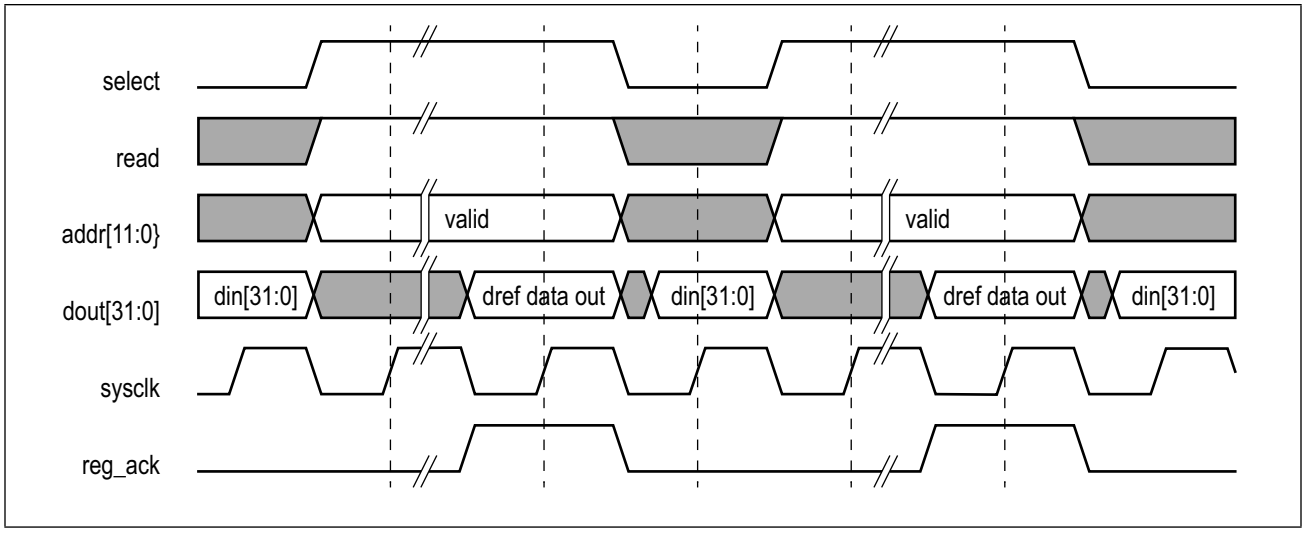
To achieve predictable results, the SAR should be enabled and disabled only during a screen blanking interval. Also, if the graphics path is configured for a pseudocolor mode, or if luma correction is enabled, then no palette/luma ram reads should be attempted, as this will generally corrupt a pixel value. The SAR algorithm may be expressed in verilog as follows:

```
always @(posedge pclk) sar  = {sar, ^(sar & 24'h80000d)} ~^
data;
```

DREF Timing Diagrams

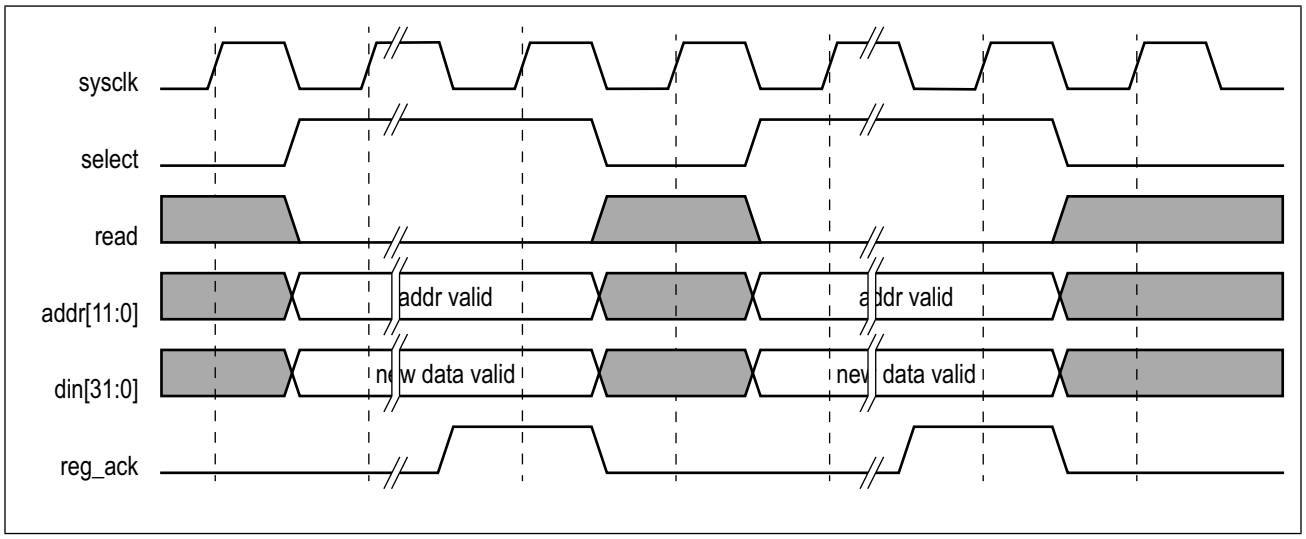
The timing diagram for a read cycle is shown in Figure 42.

Figure 42. Register Interface Read Cycle Timing Diagram



The timing diagram for a write cycle is shown in Figure 43.

Figure 43. Register Interface Write Cycle Timing Diagram



3D GRAPHICS ACCELERATOR

3D Interface

This chapter contains detailed information on the 3D graphics accelerator and Pixel Rendering Engine (PRE). The interface between the Bt2166 and the 3D graphics accelerator is via the following registers.

3D Core Interface Registers

name: 3D_CORE
address: GBASE | 00610000h, read/write
size: 64 32-bit registers
function: The 3D core registers are used to interface between the Bt2166 and the 3D graphics accelerator. These registers provide a secondary access location for the host to access the core registers. These registers are used mainly for debug and are not available when the 3D core is running. See Table 169.

Table 169. 3D Core Interface Register (3D_CORE)

3D Register	Description
0:28	General purpose registers.
29:30	Maskable interrupt link registers used to link back to the position where an interrupt occurred.
31	Branch link register used to link back to the position where a branch occurred.
32:56	Reserved
57	High 32-bit multiply result
58	Middle 32-bit multiply result
59	Low 32-bit multiply result
60	24 bit register. Loop count register used for zero delay loops.
61:63	Reserved

3D Interface Auxiliary Registers

name: 3D_AUX
address: GBASE | 00620000h, read/write
size: 512 32-bit registers
function: The auxiliary registers are used to extend the capabilities of the 3D Core registers. These registers provide a secondary access location for the host to access the auxiliary registers. When the 3D core is running, access is allowed only to the IDENTITY, SEMAPHORE, and STATUS registers. Table 171 defines the auxiliary registers. To access all locations in the table, multiply the hex value by 4 and add to GBASE; for example: SC_RAM starts at GBASE | 00620200h.

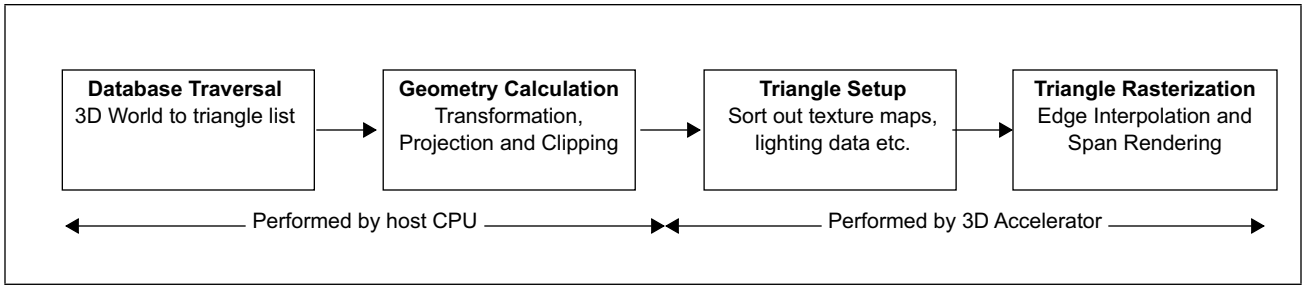
Table 170. 3D Interface Auxiliary Register (3D_AUX)

3D Register # in dec (hex)	3D Register Name	Description
0 (00h)	STATUS	Status/primary control register contains condition codes, interrupt mask bits, halt bit, and program counter.
1 (01h)	SEMAPHORE	Semaphore register controls 3D processor/Bt2166 communication process.
2 (02h)	LP_START	Loop start address register contains start address of the zero delay loop mechanism.
3 (03h)	LP_END	Loop end address register contains end address of the zero delay loop mechanism.
4 (04h)	IDENTITY	3D identification register contains the manufacturer code 2 in bits 31:24 and version number 1.1.1 in bits 23:16 (0x02010101).
5 (05h)	DEBUG	Debug register contains debugging parameters.
6:15 (06h 0Fh)		Reserved
16 (10h)	IVIC	Invalidate instruction cache
17 (11h)	WAG	Waggle port (not connected)
18 (12h)	MULHI	Preload MHI (core register 58)
19 (13h)	AUX_BRST	Block move size
20 (14h)	AUX_SRC_ADR	Block move address in SRAM
21 (15h)	AUX_LOAD_ADR	Block load address in VRAM
22 (16h)	AUX_STORE_ADR	Block store address in VRAM
23:94		Reserved
95 (5F)	PRE_STATUS	Pixel Rendering Engine status
128:255 (80h-FFh)	SC_RAM	Scratch RAM
256:511 (100h-1FFh)	CLUT	Color look-up table, write-only

3D Graphics Pipeline

A typical 3D graphics pipeline can be broken down into four steps: database transversal, geometry calculation, triangle setup, and triangle rasterization. Figure 44 shows these steps in a typical 3D graphics pipeline, and the following subsections define the steps. The Bt2166 3D graphics accelerator performs the triangle setup and triangle rasterization, and the host CPU performs the database transversal and geometry calculation.

Figure 44. Typical 3D Graphics Pipeline



Database Traversal

Assuming that the scene has been processed (i.e. the game-play task has been performed) the 3D model database has to be traversed in order to produce a triangle list for rendering. This triangle list can be a further database of independent triangles or a database of tri-strips. Note that for best hardware accelerator performance, tri-strips are preferred.

Geometry Calculation

The list of triangles that make up the new world have to be transformed, projected to their positions in display space, and clipped:

- Transformation - scaling and rotating of a triangle.
- Projection - The transformed triangle is projected into display space.
- Clipping - Not all objects in the world are visible by an observer. Some objects are outside the cone (actually a pyramid) of vision, and in some cases may be completely obscured by other objects.

Use of the Z buffer helps some of the work for clipping, but not all. The back faces of objects will also not be visible, and these polygons may be removed by the clipping process.

The result of the clipping phase will be a list of polygons, whose vertices are specified in display co-ordinates.

Triangle Material and Lighting Set-up

The various features like textures, fixed color, opacity value have to be set up per triangle or tri-strip. This phase is the fetching of the data and addresses for such features and preparing them into the format required for the triangle rasterization phase.

Triangle Rasterization

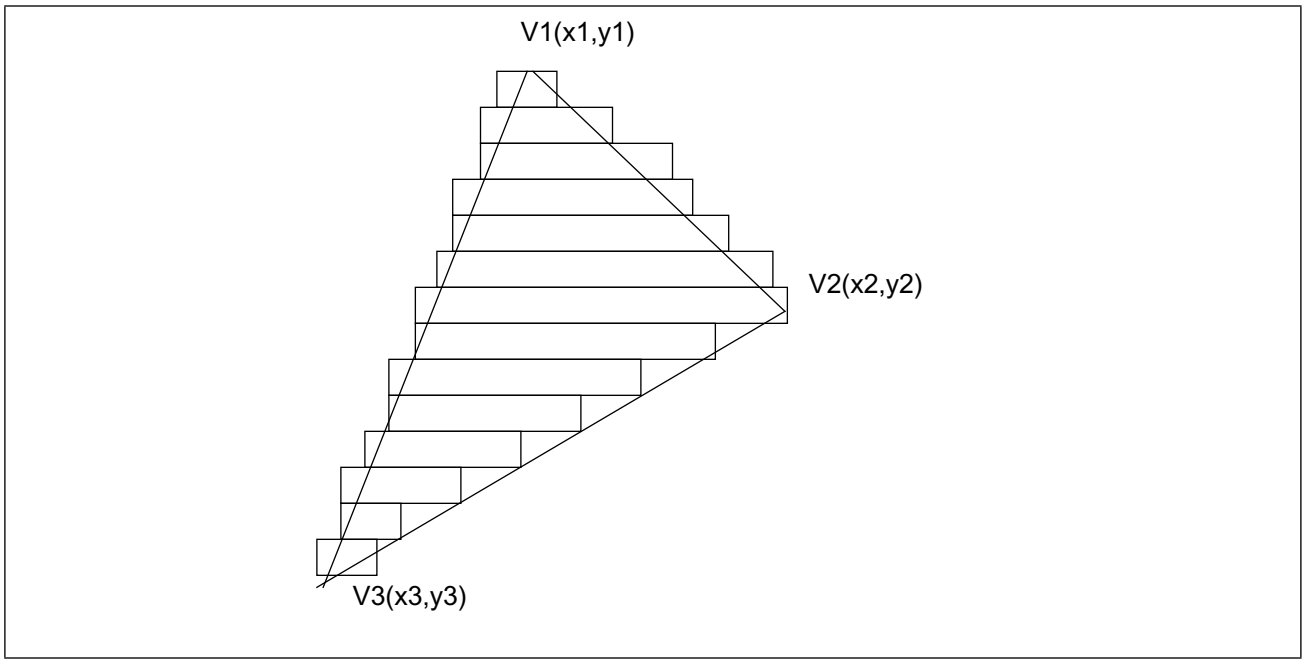
A displayable polygon is represented by the values of several types of co-ordinates at the vertices. The triangle rasterization phase, followed by the rendering phase, takes these vertex co-ordinates and transforms them into pixel values in the screen buffer. Every pixel in the screen buffer which maps to the polygon must be rendered.

Triangle rasterization is the process of transforming a series of vertices into a series of one-dimensional strips of pixels, which make up the polygon. Each of these strips is then rendered separately.

A complication is that polygons which are partially visible, have to be rasterized. This may occur at this stage, or possibly a previous stage may fracture the polygon into several visible polygons.

Assume that we wish to display a triangle whose vertices have been generated in display co-ordinates by the geometry calculation stage, as shown in Figure 45:

Figure 45. Sample Triangle



Vertex Parameter Fetch and Slope Calculation

The triangle setup stage retrieves the vertex parameter data including screen co-ordinate, color, alpha, Z, and texture information. Then it performs slope calculations for the appropriate vertex parameters.

The three triangle vertices are ordered so that the y co-ordinate of each vertex is arranged:

$$y1 < y2 < y3$$

(where the top of the screen is 0, and the bottom is the ‘screen height’-1)

In order to calculate the start and end points of each span, we have to calculate the slopes of each of the edges of the triangle.

In order to perform more realistic rendering, we have to calculate the value of several co-ordinates at each pixel. In order to calculate these during span interpolation, the slope of these co-ordinates in the x direction must be calculated. Note that these x increments do not change over the polygon. Once they have been calculated they may be transmitted to the span interpolator.

Edge Interpolation

The first stage in triangle rasterization is edge interpolation. This process consists of finding the start and end points of each span, given two slopes for the two edges of the portion of the triangle being processed. This process is carried out by the Edge Engine.

Note that there is a change in behavior of the edge interpolator as the y co-ordinate reaches that of vertex 2. The slope of one of the edges changes here. There are two ways in which this change can be dealt with. The triangle could be split in two at this point, so that two triangles are generated for the top and bottom portions. However, this would require a good deal of unnecessary re-calculation. It is more convenient to calculate all values starting from the longest edge: the only change which has to be made at the inflection point is to the x end calculation.

One consequence of performing calculations starting from the longest edge, is that this edge may be on either the left or the right hand side of the triangle. In the latter case, this implies that we must render in a decreasing x direction.

The edge interpolation process must also be able to deal with a ‘flat-topped’ and ‘flat bottomed’ triangles. That is, ones in which the y co-ordinate of V1 and V2, or V2 and V3 are identical.

Span Rendering

The second stage in triangle rasterization is span rendering. This is the phase which actually sends the new pixel values to the screen buffer. This is the part that the Pixel Engine performs: the actual stripe or span drawing process.

The start and end points of a span are passed to the span interpolator, which performs the rendering process for each pixel. The number of pixels written is determined by the difference in the start and end x co-ordinates of the span.

3D Graphics Accelerator

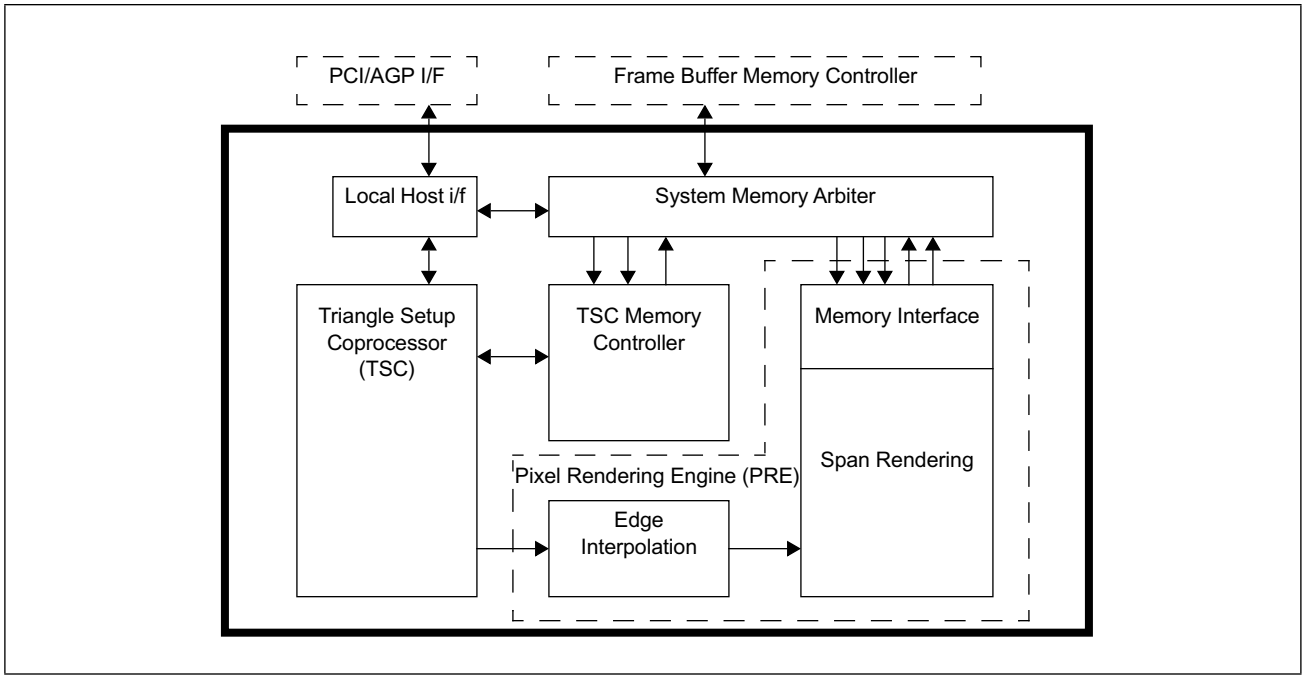
The 3D Graphics Accelerator Engine enhances the graphic capabilities of the host CPU by accelerating the rendering of texture mapped, Z-buffered, alpha blended, Gouraud shaded triangles.

The 3D Accelerator consists of two main parts:

- Pixel Rendering Engine - PRE
- Triangle Setup Coprocessor - TSC

The PRE includes both an edge interpolation engine and a span rendering engine. Figure 46 outlines the architecture of the 3D Accelerator System:

Figure 46. 3D Accelerator Architecture



Pixel Rendering Engine

The Pixel Rendering Engine (PRE) is designed to draw triangles (or polygons) based on input of the triangle vertex information and interpolation values across the triangle. PRE performs the algorithms to produce the sequence of triangle spans and in turn the final pixel values that are written to screen memory. It can perform all the operations required for texture mapping, Gouraud shading, sprite rotation and many other types of rendering. PRE operates in parallel with a triangle setup coprocessor which allows triangle plotting operations to overlap triangle set-up operations.

PRE Key Features

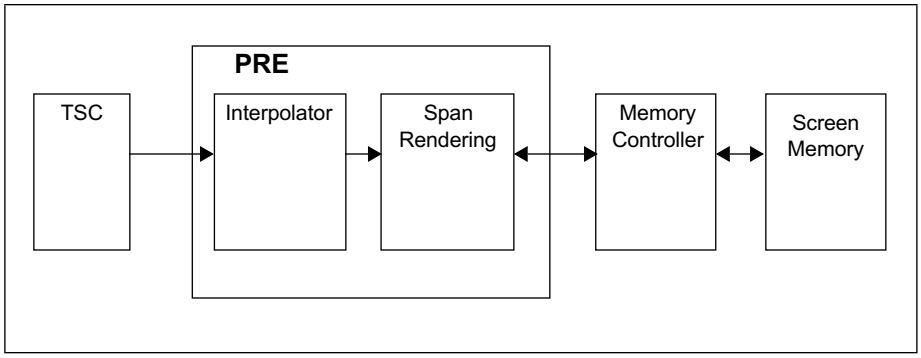
- Direct Shape Draw
  - Triangles
  - Lines
  - Rectangles
  - Flat top and bottom quads
- Parameter Interpolation
  - Edge Engine
  - Span Engine
- Output Pixel Resolutions
  - 8 bit RGB 332
  - 16 bit RGB 565
  - 24 bit RGB 888
  - Dither
- Lighting Algorithms
  - Flat Shading
  - Gouraud Shading
- Texture Mapping Algorithms
  - Transparency
  - Alpha Maps
  - 4 & 8 bit paletted textures
  - 16 & 24 bit RGB
  - 32 bit ARGB
  - Screen Packed Textures
  - Perspective Correct
  - Bilinear Interpolation
- Alpha Blending Algorithms
  - Standard
  - Summed
  - Dimmed
  - Constant Color

- Z Buffering
  - 8, 16 and 32 bit, independent of Pixel Resolution
  - Conditional Z and Pixel Write
- Internal 32 Bit RGB & Alpha ALU
  - 8, 16, 24 bit Pixel Color
  - 8 bit alpha
- Texture Cache

Basic Architecture

PRE is based on two main interpolation blocks: the edge interpolator and the span renderer (see Figure 47). PRE is programmed by the TSC and connects via a memory controller to the screen memory.

Figure 47. PRE Basic Architecture



The edge interpolator traverses the triangle edges and sends commands to the span renderer to draw spans. Upon receipt of commands from the edge interpolator, the span renderer will plot the pixels that make up a triangle span.

Supported Algorithms

During 3D graphics rendering, different effects are applied to the triangle that is to be displayed on the screen. PRE supports the following rendering algorithms:

Scaled and Rotated Sprites

Sprites are basically texture maps that have no transformation in the Z (depth) plane, and hence no shading or perspective applied.

Flat Shading

Flat shading or ‘faceted shading’ is the simplest method of lighting a polygon. An illumination model is applied once per polygon to produce a constant shade of color over the face of the polygon.

Gouraud Shading

Gouraud shading simulates the variation in color intensity as the angle between the light source, the normal of the surface and the viewing direction changes. This is performed by specifying the color intensity at the start of a span, and the change in intensity as the span is rendered from pixel to pixel.

Texture Mapping

Texture maps are areas of memory which contain color ‘pictures’ which can be mapped onto the polygon that is to rendered onto the screen.

Translucency This is the affect of blending a pixel that is to be displayed with a second pixel value which could be a value currently on the screen or a fixed color.

Transparency Texture maps can be used in such a way that they have ‘holes’ in them. When a transparent part of a texture map is detected no pixel is written out so that the previous value in the screen map is left unchanged.

Alpha Mapping It is useful to define textures which also contain translucency, this known as an alpha map. A special type of mode is used on the texture to accommodate this. This translucency data is blended with the texture color.

Light Sourced Texture Mapping Flat shading lighting effects can be applied to textures. The illumination model is applied to the whole of the texture on the surface of the polygon.

Gouraud Shaded Texture Mapping Gouraud shading effects can additionally be applied to textures. The illumination model is applied in different increments across the span.

Perspective Texture Mapping When a texture recedes into the distance perspective is applied to it. Without proper perspective correction applied to the co-ordinates of the texture, artefacts occur on the texture closest to the view point.

Bilinear/Trilinear Interpolated Texture Mapping When displaying texture mapped polygons close to the viewpoint, one 'texel' in the texture map is mapped to many screen pixels. The polygon thus takes on a very 'blocky' appearance. Bilinear or trilinear interpolation is used to smooth out such artefacts, by interpolating the values in the texture map so that it appears more detailed.

Z Buffering When rendering 3-D scenes, some portions of polygons may not be visible. There are two types of cases where this effect occurs. In the first, some polygons which define the surface are facing away from the viewpoint - they are on the back surface of an object. In the second, two objects may overlap or interpenetrate each other.

One solution to the problem of which polygons (and parts of polygons) to render is to use a 'Z buffer'. When a polygon is rendered, at the same time that a color is written to the screen buffer, a value which represents the distance of that point on the polygon from the viewpoint (the Z co-ordinate) is written to a corresponding position in a Z buffer.

Through the use of the Z buffer objects that are not obscured by other objects remain on the screen.

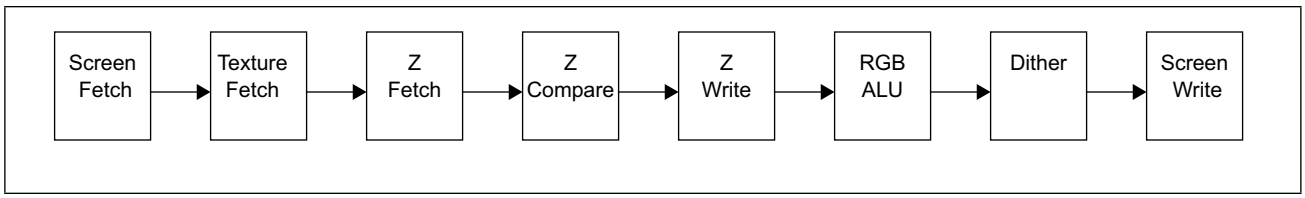
Dithering Shading effects sometimes produce contouring affects on a polygon. Through the use of random dithering on the RGB values, before output to the screen memory, this contouring effect is reduced.

Line Drawing Lines drawing is accomplished by making use of PRE’s ability to draw trapezoids.

PRE Data Flow Pipeline

Figure 48 shows the conceptual data-flow pipeline for the PRE.

Figure 48. PRE Data Flow Pipeline



Screen Fetch Background pixel (or *destination* pixel) fetch, for transparency and blending operations.

Texture Fetch The texture map texel is attained in this phase.

Perspective Correction When enabled, perspective correction modifies the texture fetch address.

Color Lookup If texture compression is enabled then the color look-up table (CLUT) is used to get the 24 bit color value.

Bilinear Interpolation Perform appropriate interpolation on the fetched texture to reduce pixellation effects.

Z Fetch The Z data value is fetched from the Z buffer for the pixel that is about to be rendered (the *source* pixel).

Z Compare If Z comparison returns true, then the Z and screen write sections are disabled for the current pixel.

Z Write The new Z value is written to the Z buffer if the Z comparison returns true.

Color RGB ALU Perform the arithmetic required to apply lighting and blending to the current pixel (the *source* pixel).

Lighting The RGB values are applied to the pixel.

Alpha Blending is applied with current “alpha” value, background, fixed color, or texture translucency.

Transparency For transparency the RGB ALU section allows the screen pixel to remain the same as the background.

Dither The randomization of the pixel is applied.

Screen Write Finally the pixel with the color, blending, and dithering applied, is written to the screen buffer.

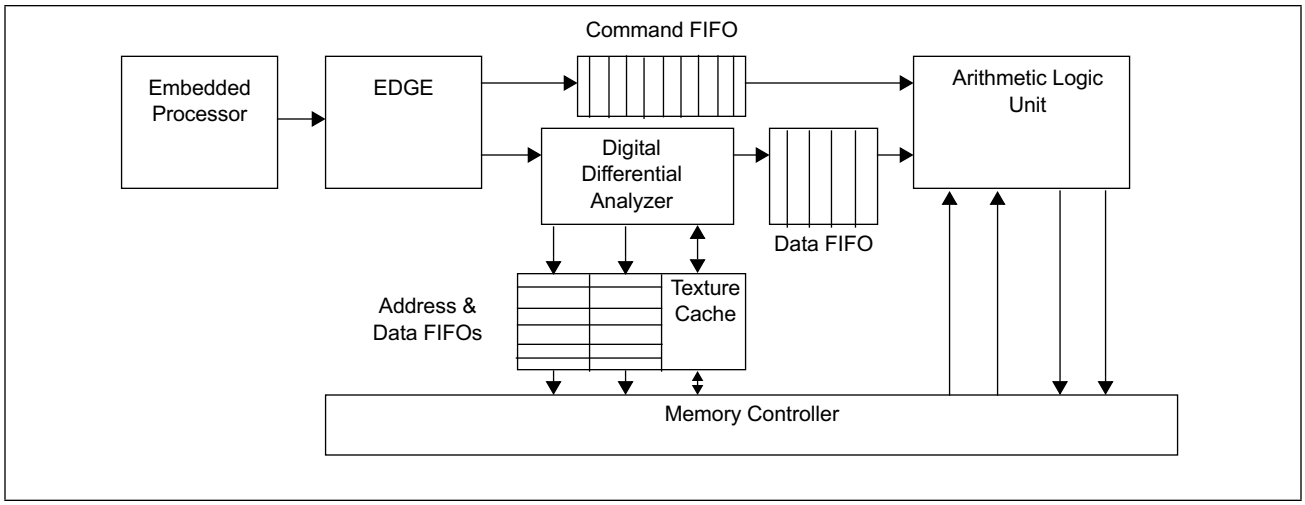
PRE Programming Model

The PRE is implemented in two programmable sections, the DDA block and the RGB ALU block. These two blocks are considered to be separate because the time between issuing a request for texture data and receiving that data may be variable, and it is desirable to be able to cope with as long a delay from the memory system as possible.

The TSC programs both of these blocks, through the edge interpolator and command FIFO, in order get PRE to perform appropriate rendering operations. The DDA block is basically concerned with generating the correct addresses with which to fetch screen, texture and Z data. The RGB ALU block performs the lighting and shading effects on the pixel data that is to be written to the screen. The DDA additionally passes data to the RGB ALU via a data FIFO.

Figure 49 shows the basic architecture of PRE with connections to the TSC and memory controller.

Figure 49. PRE Architecture with Embedded Processor and Memory



PRE is split between the sections which generate addresses for the memory controller to fetch, and which consume texel data. Register writes which refer to registers in the address generation section are piped into one command buffer, and pixel calculation register references into the other. Internal synchronization is maintained by keeping a pixel counter in each section, and feeding these writes to both buffers.

FIFO Update of Registers

PRE conceptually has a register set writable and one register that is readable. However the write interface is different to the read interface in that data sent to a register in PRE is buffered up via the edge interpolator and the command FIFO.

The read interface of PRE will return the current value of the STATUS register.

The embedded processor sets up the triangle parameters and then writes to the S_BOT register. PRE will then perform the edge interpolation and span interpolation processes to draw that triangle.

Double Buffering

If the PRE registers were to be modified directly, it would be necessary to synchronize the communication of parameters between the TSC and PRE by waiting until an operation is finished, transfer parameters, and then restart. The overhead involved is likely to be high when performing short runs of complex rendering operations. PRE registers are therefore double buffered, so that transfer of parameters may take place in one operation. When the embedded processor writes to the S_BOT register, and PRE accepts that register value, the new register values are transferred to the working registers.

Command FIFO Depth

The depth of the command buffer (FIFO) is system dependent but typically 8 or more.

The Register Set

The register set comprises of 3 basic types:

- Control Registers: Turns sections on and off, set up base values and provide limiting values.
- Working Registers: Values that PRE uses for rendering.
- Increment Registers: Values which update the working registers per pixel write.

Control Registers

This set of registers covers the registers that impose limits on the rendering of a triangle or a scene. Typically these are registers like vertex x co-ordinate values, number of spans per triangle, CLUT data, fixed color for blending operations, texture base address, mode register and the S_BOT register.

Working Registers

These values are usually set up on a per span basis and are “current” values used for rendering. For example the RED alpha value is the weighting applied by the RGB ALU to the red component of the pixel.

Increment Registers

These values are set up both edge interpolation and span interpolation processes and are the increment values added to the working registers per rendering cycle. During forward rendering the increment value is added (post-increment) to the working register. During reverse rendering (where the pixel count is negative) the increment value is taken away from (pre-decrement) from the working register. For example, during forward rendering, the RED increment value is the value that is added to the RED alpha value, after the pixel has been written to the screen.

Synchronization with the TSC

Synchronization is provided by reading the STATUS register.

Memory Access

PRE interfaces to the memory through the memory controller interface. PRE is heavily pipe-lined, giving a maximum throughput of one pixel rendered per clock cycle. This figure may be degraded according to the memory architecture. However, PRE provides internal buffering of several DMA channels, in order that memory accesses may be performed in bursts within the same memory 'page'. The sizing of buffers, and their implementation, is dependent on the memory architecture.

In addition, there is a pipeline which keeps track of which word or nibble, is required within each quad-word access. This information is synchronized with the data returning from the memory controller.

Data Organization and Addressing

The definitions of data sizes are as follows:

nibble 4 bits.
byte 8 bits.
word 6 bits.
long-word 32 bits.
quad-word 64 bits.

The data bus presented to and received from the memory controller for each interface is 64 bits

Address Size

PRE provides a 24 bit address bus to the memory controller for each interface.

RGB ALU Data

PRE uses 24 bit color data (RGB888) for the internal calculations and provides the ability to unpack and pack into other color formats.

Screen Buffer

The screen or display buffer is conceptually a two dimensional array of elements, whose value represents the color of a pixel. The screen buffer is the actual display area that is sent to the screen of whatever device PRE is used within.

Screen Buffer Format

There are three possible formats for screen buffer pixels. RGB332 (or 8 bit) color mode, RGB565 (or 16 bit) color mode, and RGB888 (or 24 bit) color mode.

RGB332

8 bit color is made up with the color components represented as RGB332. The resolution of the components is red - bits 7 to 5 (3 bits), green - bits 4 to 2 (3 bits), and blue - bits 1 to 0 (2 bits).

7	6	5	4	3	2	1	0	
r2	r1	r0	g2	g1	g0	b1	b0	0x0000
r2	r1	r0	g2	g1	g0	b1	b0	0x0001

For the RGB332 mode, it is assumed that an increment of one horizontal pixel along the screen, from left to right, is mapped by incrementing the address in the screen buffer by one byte.

Conversely, one pixel down on the screen is mapped by an increment in screen buffer address of the screen width.

If ***s*** represents the start address of a screen buffer, ***w*** represents the width of a screen in pixels, and ***h*** represents the height of the screen in pixels, then the screen mapping is shown below, where all addresses are in bytes:

top left

s	s+1	..	s+w-1
s+w	s+w+1	..	s+2w-1
..			..
s+hw-h	s+hw-(h-1)	..	s+hw-1

bottom right

RGB565

RGB565 pixels are represented in the screen buffer by 16 bit values, which contain red, green and blue pixel color component intensities. The resolution of the components is red - bits 15 to 11 (5 bits), green - bits 10 to 5 (6 bits), and blue - bits 4 to 0 (5 bits).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
r4	r3	r2	r1	r0	g5	g4	g3	g2	g1	g0	b4	b3	b2	b1	b0	0x0000
r4	r3	r2	r1	r0	g5	g4	g3	g2	g1	g0	b4	b3	b2	b1	b0	0x0002
r4	r3	r2	r1	r0	g5	g4	g3	g2	g1	g0	b4	b3	b2	b1	b0	0x0004
r4	r3	r2	r1	r0	g5	g4	g3	g2	g1	g0	b4	b3	b2	b1	b0	0x0006

For the RGB565 mode, it is assumed that an increment of one horizontal pixel along the screen, from left to right, is mapped by incrementing the address in the screen buffer by one word (two bytes). Conversely, one pixel down on the screen is mapped by an increment in screen buffer address of twice the screen width.

If ***s*** represents the start address of a screen buffer, ***w*** represents the width of a screen in pixels, and ***h*** represents the height of the screen in pixels, then the screen mapping is shown below, where all addresses are in bytes:

top left

s	s+2	..	s+2w-2
s+2w	s+2w+2	..	s+4w-2
..			..
s+2wh-2h	s+2wh-(2h-2)	..	s+2hw-2

bottom right

RGB888

Alternatively, pixels may be represented by three 8 bit component values; however, such values are aligned on long-word boundaries. The resolution of the components is red - bits 23 to 16 (8 bits), green - bits 15 to 8 (8 bits), and blue - bits 7 to 0 (8 bits).

31..24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XX	r7	r6	r5	r4	r3	r2	r1	r0	g7	g6	g5	g4	g3	g2	g1	g0	b7	b6	b5	b4	b3	b2	b1	0x0000
XX	r7	r6	r5	r4	r3	r2	r1	r0	g7	g6	g5	g4	g3	g2	g1	g0	b7	b6	b5	b4	b3	b2	b1	0x0004

For the RGB888 mode, it is assumed that an increment of one horizontal pixel along the screen, from left to right, is mapped by incrementing the address in the screen buffer by one long-word (four bytes).

Conversely, one pixel down on the screen is mapped by an increment in screen buffer address of four times the screen width.

If ***s*** represents the start address of a screen buffer, ***w*** represents the width of a screen in pixels, and ***h*** represents the height of the screen in pixels, then the screen mapping is shown below, where all addresses are in bytes:

top left

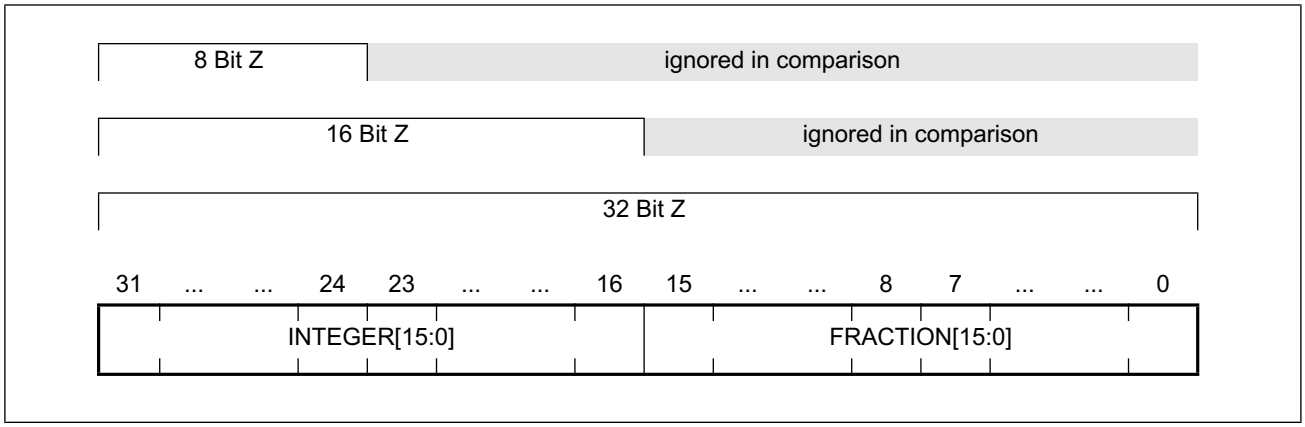
s	s+4	..	s+4w-4
s+4w	s+4w+4	..	s+8w-4
..			..
s+4wh-4h	s+4wh-(4h-4)	..	s+4hw-4

bottom right

Z Data Buffer

The Z data buffer contains the value of Z (or the Z ordinate) for each screen buffer pixel. Z values can be 8, 16 or 32 bits in length. For 8 bit the stored value is the top 8 bits of the 32 bit word; for 16 bit the value in the Z data buffer is the integer part (top 16 bits) of the internal Z value calculated by PRE; and finally in 32 bit Z mode the Z data buffer value is represented the full 16.16 which contains both the integer and fractional parts. During Z comparisons the unused bits for 8 and 16 bit Z are ignored in both the Z data buffer value and the calculated Z. This is shown in Figure 50.

Figure 50. Z Data Buffer



Texture Data Buffer

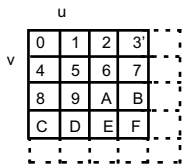
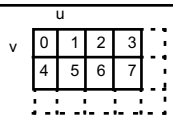
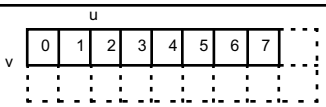
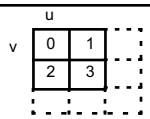
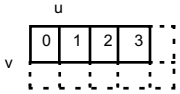
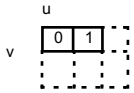
The texture data buffer refers to the area of memory that contains the texture maps. The texture may be taken from any image; one important case is where the texture map is a previously rendered image. However, other texture maps, initialized by the application will be the dominant texture source.

Texture Map Formats

PRE can read textures in different formats. The formats are: packed - 4 bits, 8 bits and 16 bits per texel and linear - 8 bits, 16 bits and 32 bits per texel. The linear formats match up with the screen buffer formats. (See Figure 51.)

The packed formats provide a limited form of texture caching, by grouping texels which are spatially adjacent together. Use of the packed pixel modes can also significantly improve throughput when bilinear interpolation is enabled, by limiting the number of memory fetches.

Figure 51. Texture Map Formats

		Packed: ENLN = 0		Linear: ENLN = 1	
IMODE	**bits/ texel**	**Texel Pack in 64 bit word**	**Texel Cell Format**	**Texel Pack in 64 bit word**	**Texel Cell Format**
0	4		4 x 4 *CLUTed*	N/A	N/A
1	8		4 x 2 *CLUTed*		8 x 1 *Expanded*
2	16		2 x 2 *Expanded*		4 x 1 *Expanded*
3	32	N/A	N/A		2 x 1

Linear Textures

Linear texture maps correspond to the screen buffer formats of the same color resolution, where the texture base address ***t*** of an ***n*** by ***m*** texture correspond to a screen buffer where ***s* = *t*, *n* = *w*** and ***h* = *m*** (see ‘‘Screen Buffer Format’’ on page 233).

8 and 16 bit Linear Textures

8 bit and 16 bit per texel linear format values are expanded to 24 bit RGB888 values and do not make use of the CLUT.

32 bit ARGB8888 Linear Textures

32 bits per texel format corresponds to 24 bit RGB888 with an optional alpha value. Texels are represented by four 8 bit component values aligned on long-word boundaries. The resolution of the components is alpha - bits 31 to 24 (8 bits), red - bits 23 to 16 (8 bits), green - bits 15 to 8 (8 bits), and blue - bits 7 to 0 (8 bits).

31	30	..	..	25	24	23	22	..	..	17	16	15	14	..	..	9	8	7	6	..	..	1	0	
a7	a6	..	..	a1	a0	r7	r6	..	..	r1	r0	g7	g6	..	..	g1	g0	b7	b6	..	..	b1	b0	0x0000
a7	a6	..	..	a1	a0	r7	r6	..	..	r1	r0	g7	g6	..	..	g1	g0	b7	b6	..	..	b1	b0	0x0004

Packed Textures

Packed texture maps get more complicated. They are used because a texture is typically traversed in a non-linear way. Packed textures additionally improve the speed of PRE when bilinear interpolation is enabled.

4 Bit Packed Textures

Texels are represented by a 4 bit value which is an index into the CLUT. The full color look-up address is made from this 4 bit value concatenated with a look-up base address in the MODE register. The value from the look-up table holds a RGB888 value or ARGB8565.

For 4 bit texel mode an increment of one texel in the U (horizontal) direction increments the address bye 0, 1 or 7 depending on the position in the packing. One texel in the V direction (vertical) depends on the position in the packing.

If ***t*** is the texture base address, and ***n*** is the texture map width and ***m*** the texture height then the mapping is shown below, where all addresses are in bytes:

top left

t	t	t+1	t+1	t+8	t+8	t+9	t+9	...
t+2	t+2	t+3	t+3	t+A	t+A	t+B	t+B	...
t+4	t+4	t+5	t+5	t+C	t+C	t+D	t+D	...
t+6	t+6	t+7	t+7	t+E	t+E	t+F	t+F	...
t+2n	t+2n	t+2n+1	t+2n+1	...	...	...	...	...
t+2n+2	t+2n+2	t+2n+3	t+2n+3	...	...	...	...	...
t+2n+4	t+2n+4	t+2n+5	t+2n+5	...	...	...	...	...
t+2n+6	t+2n+6	t+2n+7	t+2n+7	...	...	...	...	...
...	...	...	...	...	...	...	...	...

toward bottom right

8 Bit Packed Textures

Texels are represented by an 8 bit value which is an index into the color look-up table. The value from the look-up table holds a RGB888 value or ARGB8565.

For 8 bit texel mode an increment of one texel in the U (horizontal) direction increments the address by 1 or 5 depending on the position in the packing. One texel in the V direction (vertical) depends on the position in the packing.

If ***t*** is the texture base address, and ***n*** is the texture map width and ***m*** the texture height then the mapping is shown below, where all addresses are in bytes:

top left

t	t+1	t+2	t+3	t+8	t+9	t+A	t+B	...
t+4	t+5	t+6	t+7	t+C	t+D	t+E	t+F	...
t+2n	t+2n+1	t+2n+2	t+2n+3	...	...	...	...	...
t+2n+4	t+2n+5	t+2n+6	t+2n+7	...	...	...	...	...
...	...	...	...	...	...	...	...	...

toward bottom right

16 Bit Packed Textures

Texels are represented by an 8 bit value which is an index into the color look-up table. The value from the look-up table holds a RGB888 value or ARGB8565.

For 16 bit texel mode an increment of one texel in the U (horizontal) direction increments the address by 2 or 15 depending on the position in the packing. One texel in the V direction (vertical) depends on the position in the packing.

If ***t*** is the texture base address, and ***n*** is the texture map width and ***m*** the texture height then the mapping is shown below, where all addresses are in bytes:

top left

t	t+2	t+8	t+A	t+10	t+12	t+18	t+1A	...
t+4	t+6	t+C	t+E	t+14	t+16	t+1C	t+1E	...
t+4n	t+4n+2	...	...	...	...	...	...	...
t+4n+4	t+4n+6	...	...	...	...	...	...	...
...	...	...	...	...	...	...	**toward bottom right**	

Texture Compression (CLUT)

If there are many texture maps, the storage for texture maps will become very large. There is therefore provision for the compression of texture maps. The value of a texel may be defined to be a coded value, a four or 8 bit value, (imode = 0 or imode = 1 above) which defines one of sixteen possible colors or one of 256 possible colors respectively. This value is used as an index into a 256 position RAM CLUT, which holds the actual full resolution (RGB888) color values or RGB565 with 8 bit alpha map color values.

The Lookup RAM is initialized by the embedded processor, by writing values to it via the special CLUT update register.

Alpha Maps

Alpha maps, (texture maps with additional alpha values) are available for 4 and 8 bit packed textures and for 8 and 32 bit linear textures.

Texture Cache

PRE uses a texture cache to improve the texture fetch performance.

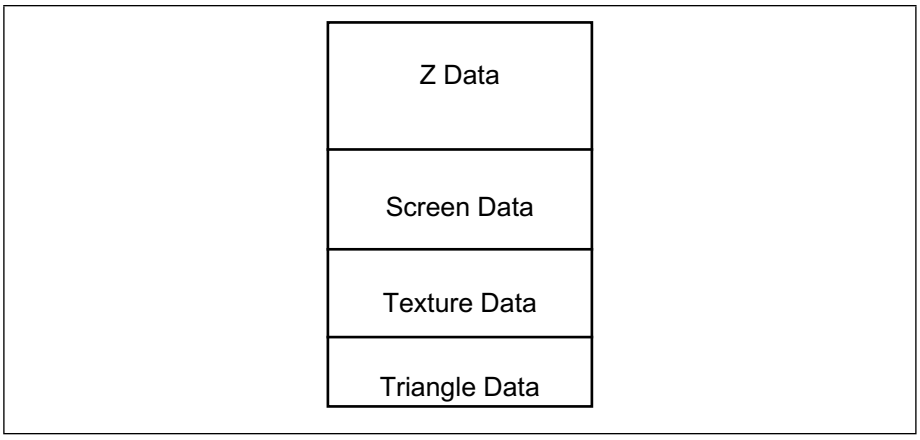
Polygon Data

The embedded processor has to have a list of triangles that need to rendered to the screen. This is typically a triangle list that is contained in memory and is referred to as the triangle data or polygon data list.

Example Memory Map

Figure 52 is a simple conceptual memory map of the different memory buffers required. Each buffer being a contiguous part of memory.

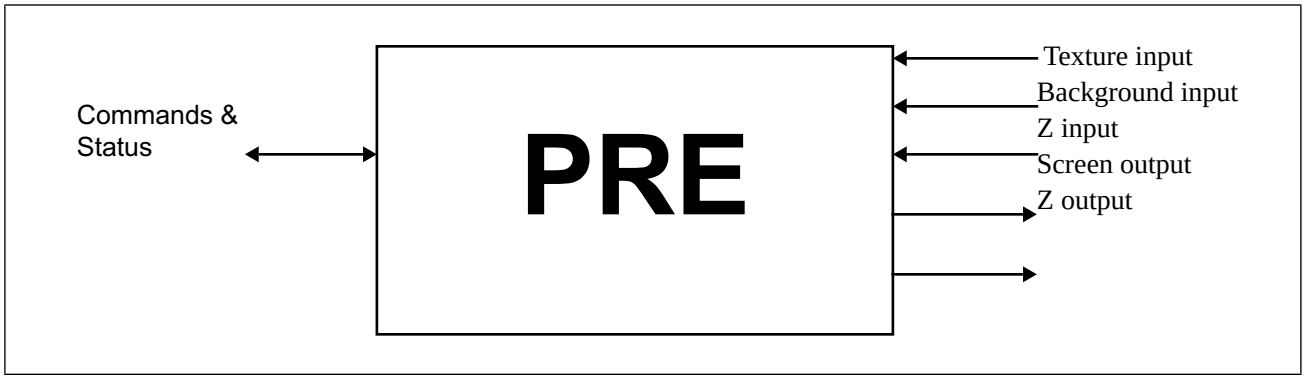
Figure 52. Example Memory Map



PRE Memory Controller Interface

There are five interfaces between PRE and the memory controller, as shown in Figure 53.

Figure 53. PRE Memory Interface



Texture Input

The texture fetch request bus passes an address to the memory, which must fetch the quad-word at that address.

The texture data bus memory interface returns texture words fetched from memory on this bus. Note that because of competition for memory, the request and data may be separated by a considerable delay. In addition, the texture data words may contain variable numbers of desired texels, at variable offsets within the word. The extraction of the data is the responsibility of PRE.

Background Input

The screen pixel source request address bus and source data bus are similar to the texture request buses, though data extraction is less complex.

Z Input

The Z request address bus and Z buffer data bus are similar to the texture request buses, though data extraction is less complex.

Screen Output

The screen pixel write bus consists of pixel data and an address. The memory interface must take the data, which will have been batched up into a memory word, and write bytes which are flagged for write (using a separate write enable bus), at the address specified on the address bus.

Z Output

The Z write bus is similar to the pixel write bus.

Register Table

Table 171 provides a summary PRE register table. The registers are defined in detail in the "Register Set Details" on page 245. The columns are defined as follows:

# = Register number
Name = Register name
Block = Which major section the register relates to
Type = Register type: Control (ctrl), working (work), or increment (inc)
Bits = Number of bits and format
± = Indicates value interpreted as a signed two's complement. (Bit 31 is the sign bit.)

Table 171. PRE Register Table *(1 of 3)*

#	Name	Block	Type	Bits	±	Description
						Per Triangle Parameters
0	MODE	BOTH	ctrl			Mode Register
1	MASK	DDA	ctrl	16.16		U&V Masks for texture map
2	TBASE	DDA	ctrl	24		Texture base address
3	ZBASE	DDA	work	24		Z buffer base address of top span
4	IBASE	DDA	work	24		Input Screen buffer base address of top span
5	SBASE	DDA	work	24		Output Screen buffer base address of top span
						Span Interpolation Registers
6	R_DX	ALU	inc	10.10	±	Red component pixel increment
7	U_DX	DDA	inc	16.16	±	Texture source Xu coordinate pixel increment
8	V_DX	DDA	inc	16.16	±	Texture source Xv coordinate pixel increment
9	G_DX	ALU	inc	10.10	±	Green component pixel increment
10	B_DX	ALU	inc	10.10	±	Blue component pixel increment
11	Z_DX	ALU	inc	16.16	±	Z value pixel increment
12	Q_DX	DDA	inc	1.30	±	Homogenous W value per pixel increment
13	A_DX	ALU	inc	10.10	±	Alpha value pixel increment

Table 171. PRE Register Table *(2 of 3)*

#	Name	Block	Type	Bits	±	Description
						Per Triangle Parameters
14	FIXC	ALU	ctrl	8.8.8		24 bit fixed blend color
15	LKUP	ALU	ctrl	8.8.8.8		CLUT write port(addr.R.G.B)
16	XENDB	BOTH	ctrl	16.16	±	X end coordinate for lower triangle
17	XENDT	BOTH	ctrl	16.16	±	X end coordinate for upper triangle
18	XSTART	BOTH	ctrl	16.16	±	X start coordinate for dominant edge
19	SCRW	N/A	ctrl	12		Screen width in pixels
20	IVTC	DDA	ctrl	-		Invalidate Texture Cache
21	resvd					
						Edge and Span Working Registers
22	RALF	ALU	work	10.10		Red span start value
23	GALF	ALU	work	10.10		Green start value
24	BALF	ALU	work	10.10		Blue start value
25	UVAL	DDA	work	16.16		Texture source Xu span start value
26	VVAL	DDA	work	16.16		Texture source Xv span start value
27	ZVAL	ALU	work	16.16		Z span start value
28	QVAL	DDA	work	1.30		Homogenous 1/W span start value
29	AVAL	ALU	work	10.10		Alpha span start value
30	resvd					
31	STATUS	CMD	ctrl	2.10.10		Status register

Table 171. PRE Register Table *(3 of 3)*

#	Name	Block	Type	Bits	±	Description
						Edge Interpolation Registers
32	XB_DY	DDA	inc	16.16	±	Xend increment per scan line - lower
33	XT_DY	DDA	inc	16.16	±	Xend increment per scan line - upper
34	XS_DY	DDA	inc	16.16	±	Xstart increment per scan line
35	resvd					
36	resvd					
37	resvd					
38	R_DY	ALU	inc	10.10	±	Red start y increment
39	U_DY	DDA	inc	16.16	±	U start y increment
40	V_DY	DDA	inc	16.16	±	V start y increment
41	G_DY	ALU	inc	10.10	±	Green y increment
42	B_DY	ALU	inc	10.10	±	Blue y increment
43	Z_DY	ALU	inc	16.16	±	Z start y increment
44	Q_DY	DDA	inc	1.30	±	W start y increment
45	A_DY	ALU	inc	10.10	±	Alpha start y increment
						Per Triangle Parameters
46	S_TOP	DDA	ctrl	10		Number of spans in top triangle
47	S_BOT	DDA	ctrl	10		Number of spans in bottom triangle
						Reserved for Future Extensions
48	resvd					Reserved
to	...					
63	resvd					Reserved

Register Set Details

The PRE registers are arranged alphabetically according to the register name. The following terms are used in the description of each register:

- **Name:** Long name for the register.
- **Type:** Whether the register is a control, working or increment register.
- **Block:** Which block in the architecture that the register affects: DDA, ALU, CMD.
- **Section:** The register group (or section) that the register belongs to: Run, Mode, Texture Input DDA, Screen Input DDA, RGB ALU, Color, Lookup RAM, Output DMA.
- **Register Number:** The register number (sometimes referred to as the register address).
- **Description:** Comprehensive description of the register.
- **Recommended Initialization Value:** The value that should be sent to the register during the initialization of the PRE. This would usually be a value that disables the feature associated with the register.
- **Register Format:** Diagram of the register and associated register fields.
- **Register Fields:** Description of the fields in the register.

± **Sign Bit**

0 = The value is positive

1 = The value is negative

PRE 3D Registers

Intensity Pixel Increment Register

name: **A_DX**
type: Increment
block: ALU
section: RGB ALU
reg num: 13
function: This register contains the intensity increment, which is interpreted as a 9 bit value with ten bit fraction. The decimal point for these values lies between bit 15 and 16.

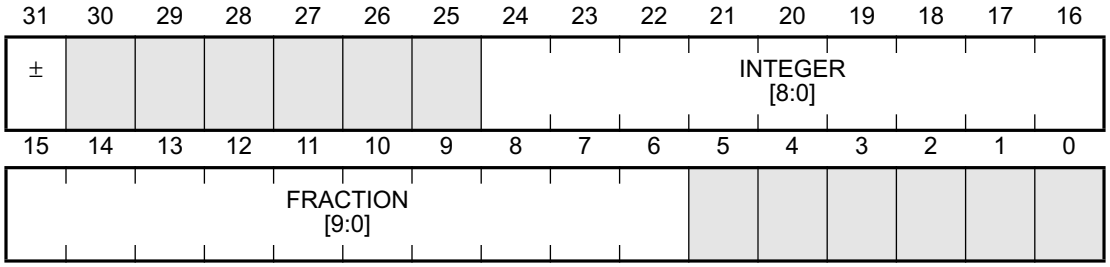
The value is signed such that the ninth bit and bit 31 of the alpha 2 increment, A_DX, is interpreted as a sign bit. At the end of each Pixel Engine cycle, the alpha 2 value (AVAL) is incremented by the increment A_DX:

```
AVAL(i+1)= AVAL(i) + A_DX
```

If the value of AVAL after the increment would be less than zero (underflow), the new value is set to zero. No overflow is detected, since it is assumed that the oversaturation range provides sufficient protection.

Recommended Initialization Value: 0

Register Format:



Register Fields:

INTEGER[8:0]	Integer Part 9 bit integer
FRACTION[9:0]	Fractional Part 10 bit fraction

Intensity Span Increment Register

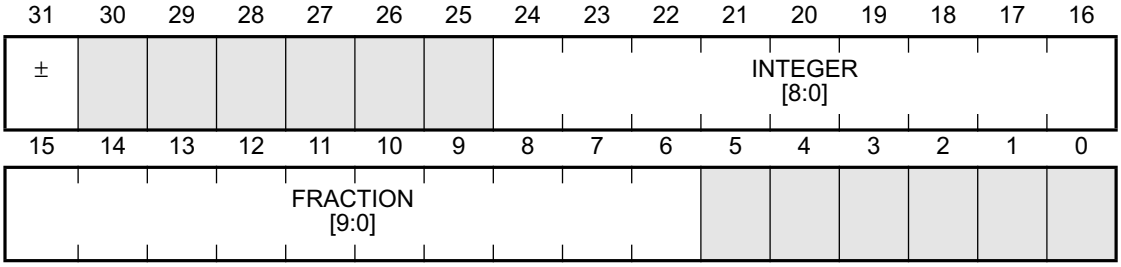
name: **A_DY**
type: Increment
block: ALU
section: RGB ALU
reg num: 45
function: This register contains an intensity increment on a per span basis. It works in a similar way to A_DX and updates AVAL every span. AVAL becomes the previous span start value for AVAL plus A_DY.

The value is signed such that the ninth bit and bit 31 of the alpha increment, A_DY, is interpreted as a sign bit.

See also A_DX and AVAL.

Recommended Initialization Value: 0

Register Format:



Register Fields:

INTEGER[8:0]	Integer Part 9 bit integer
FRACTION[9:0]	Fractional Part 10 bit fraction

Intensity Register

name: **AVAL**
type: Working
block: ALU
section: RGB ALU
reg num: 29
function: This register contains the intensity, which is interpreted as a 9 bit value with ten bit fraction. The decimal point for these values lies between bit 15 and 16.

The pixel ALU only shades using the least significant 8 bits of the integer portion; the ninth bit of alpha thus indicates saturation of the intensity value. The values are still calculated correctly up to an alpha value of 1.FF, so that saturation clipping can be handled correctly.

If the value of AVAL after the increment would be less than zero (underflow), the new value is set to zero. No overflow is detected, since it is assumed that the oversaturation range provides sufficient protection.

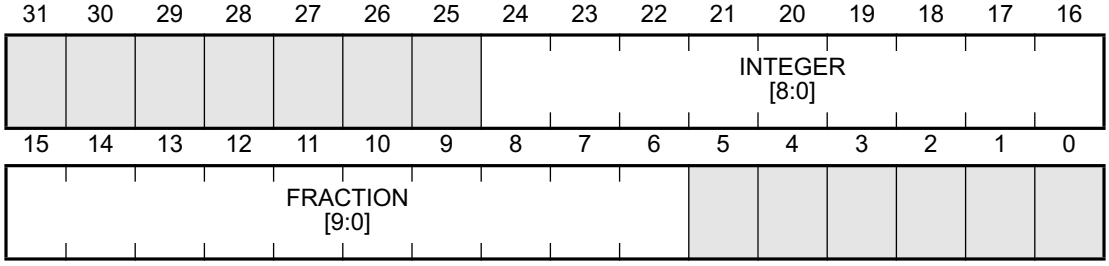
AVAL is used to perform blending operations between the calculated, *source*, pixel and input, *dest*, pixel according to the blend mode set in the MODE register:

Standard	final pixel = source*AVAL + dest*(1-AVAL)	for translucent objects
Summed	final pixel = source*AVAL + dest	for luminous areas
Dimmed	final pixel = dest*(1-AVAL)	for dark areas

If the input screen buffer is disabled then the FIXC value is used in place of *dest*. Alpha blending is disabled by setting AVAL to full saturation ($FFFFFFF), setting A_DX and A_DY to 0 and setting the blending mode to Standard.

Recommended Initialization Value: $FFFFFFFF

Register Format:



Register Fields:

INTEGER[8:0] Integer Part
9 bit integer

FRACTION[9:0] Fractional Part
10 bit fraction

Blue Light-source Pixel Increment Register

name: **B_DX**
type: Increment
block: ALU
section: RGB ALU
reg num: 10
function: This register contains the BLUE light-source increment, which is interpreted as a 9 bit value with ten bit fraction. The decimal point for these values lies between bit 15 and 16.

The value is signed such that the ninth bit and bit 31 of the BLUE alpha increment, B_DX, is interpreted as a sign bit.

At the end of each Pixel Engine cycle, the BLUE light source value (BALF) is incremented by the increment B_DX:

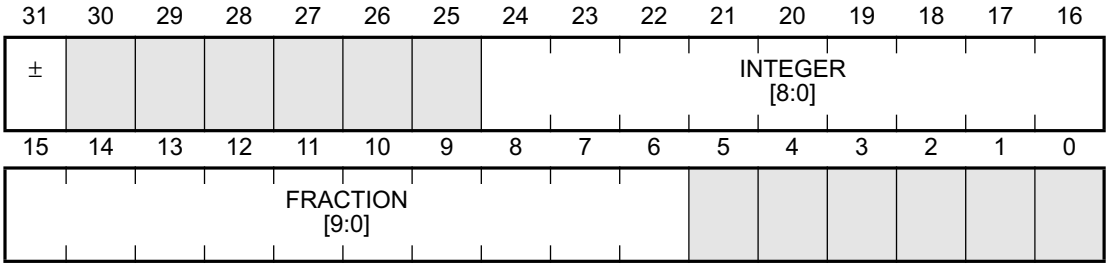
```
BALF(i+1)= BALF(i) + B_DX
```

If the value of BALF after the increment would be less than zero (underflow), the new value is set to zero. No overflow is detected, since it is assumed that the oversaturation range provides sufficient protection.

Additionally, in order to speed up parameter updates when white light sources are used, the green and blue increments are also written when the RED increment (R_DX) is written, thus avoiding the need for 2 parameter writes. To set up coloured light source values, the red parameters must be written first, followed by the blue and green values.

Recommended Initialization Value: 0

Register Format:



Register Fields:

INTEGER[8:0]	Integer Part 9 bit integer
FRACTION[9:0]	Fractional Part 10 bit fraction

Blue Light-source Span Increment Register

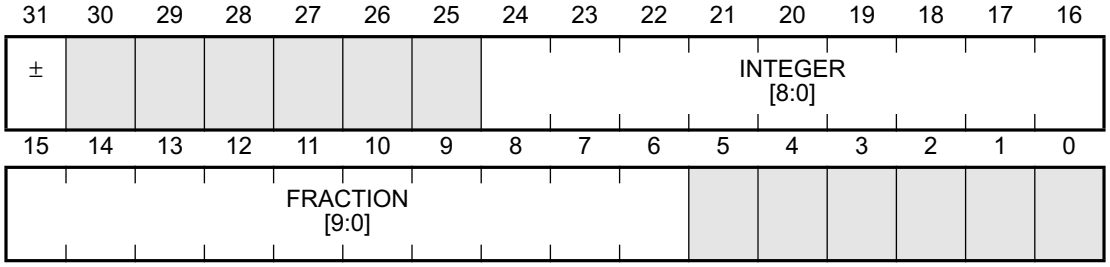
name: **B_DY**
type: Increment
block: ALU
section: RGB ALU
reg num: 42
function: This register contains the BLUE light-source intensity increment on a per span basis. It works in a similar way to B_DX and updates BALF every span. BALF becomes the previous span start value for BALF plus B_DY.

The value is signed such that the ninth bit and bit 31 of the BLUE alpha increment, B_DY, is interpreted as a sign bit.

See also B_DX and BALF.

Recommended Initialization Value: 0

Register Format:



Register Fields:

INTEGER[8:0]	Integer Part 9 bit integer
FRACTION[9:0]	Fractional Part 10 bit fraction

Blue Light Source Component Register

name: **BALF**
type: Working
block: ALU
section: RGB ALU
reg num: 24
function: This register contains the BLUE light source component, which is interpreted as a 9 bit value with ten bit fraction. The decimal point for these values lies between bit 15 and 16.

The pixel ALU only shades using the least significant 8 bits of the integer portion; the ninth bit of alpha thus indicates saturation of the intensity value. The values are still calculated correctly up to an alpha value of 1.FF, so that saturation clipping can be handled correctly.

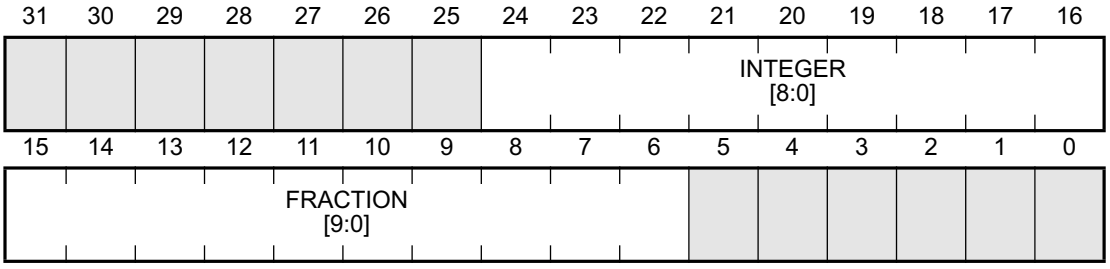
Each colour component has a separate intensity multiplier, so that coloured light sources may be modeled. The colour alpha intensities (RALF, GALF, BALF) perform an unsigned multiply of the pixel colour component values derived from the texture data stream:

```
R' = RALF * R
G' = GALF * G
B' = BALF * B
```

Additionally, in order to speed up parameter updates when white light sources are used, the green and blue components are also written when the corresponding red values (RALF, R_DX) are written, thus avoiding the need for 4 parameter writes. To set up coloured light source values, the red parameters must be written first, followed by the blue and green values.

Recommended Initialization Value: 0

Register Format:



Register Fields:

INTEGER[8:0]	Integer Part 9 bit integer
FRACTION[9:0]	Fractional Part 10 bit fraction

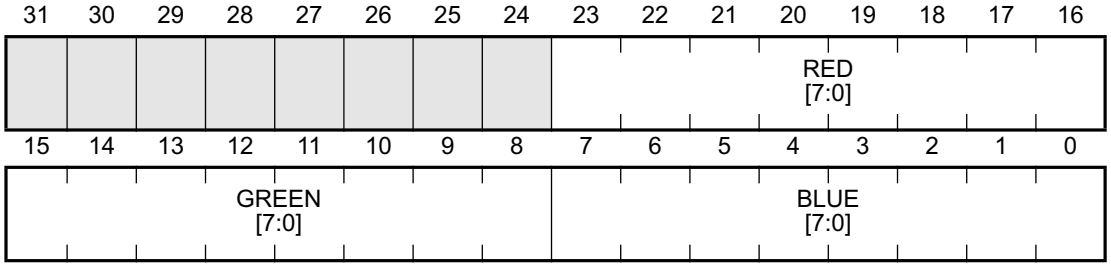
Fixed Background Colour Register

name: **FIXC**
type: Control
block: ALU
section: Colour
reg num: 14
function: This register contains three eight bit colour component values (R,G,B), which provides a fixed colour for drawing flat and shaded polygons. It may also be used to blend pixels with a fixed colour; one use of this is fog effects.

The 24 bit RGB fixed colour value is passed to screen input DMA side of the 24 bit ALU, when that path is not enabled by ENSC in the Mode register MODE.

Recommended Initialization Value: 0

Register Format:



Register Fields:

RED[7:0]	Fixed Colour Red
GREEN[7:0]	Fixed Colour Green
BLUE[7:0]	Fixed Colour Blue

Green Light-source Pixel Increment Register

name: **G_DX**
type: Increment
block: ALU
section: RGB ALU
reg num: 9
function: This register contains the GREEN light-source increment, which is interpreted as a 9 bit value with ten bit fraction. The decimal point for these values lies between bit 15 and 16.

The value is signed such that the ninth bit and bit 31 of the GREEN alpha increment, G_DX, is interpreted as a sign bit.

At the end of each Pixel Engine cycle, the GREEN light source value (GALF) is incremented by the increment G_DX:

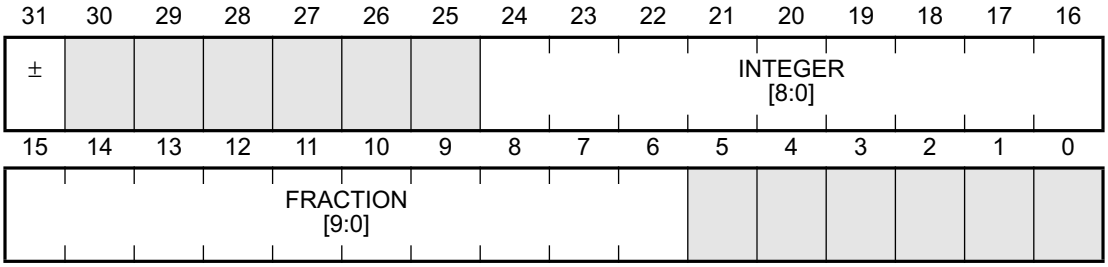
```
GALF(i+1)= GALF(i) + G_DX
```

If the value of GALF after the increment would be less than zero (underflow), the new value is set to zero. No overflow is detected, since it is assumed that the oversaturation range provides sufficient protection.

Additionally, in order to speed up parameter updates when white light sources are used, the green and blue increments are also written when the RED increment (R_DX) is written, thus avoiding the need for 2 parameter writes. To set up coloured light source values, the red parameters must be written first, followed by the blue and green values.

Recommended Initialization Value: 0

Register Format:



Register Fields:

INTEGER[8:0]	Integer Part 9 bit integer
FRACTION[9:0]	Fractional Part 10 bit fraction

Green Light-source Span Increment Register

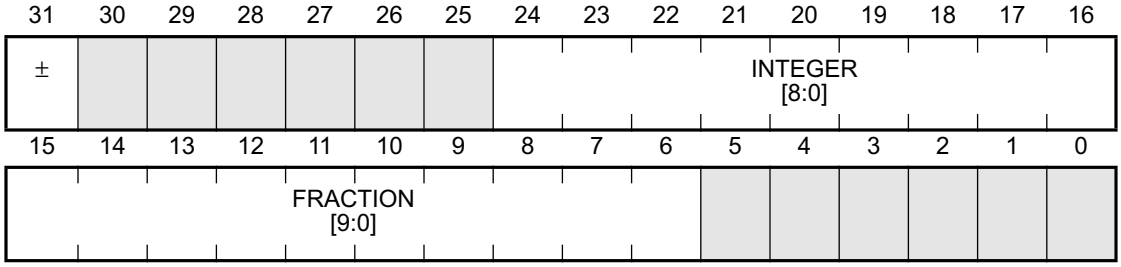
name: **G_DY**
type: Increment
block: ALU
section: RGB ALU
reg num: 41
function: This register contains the GREEN light-source intensity increment on a per span basis. It works in a similar way to G_DX and updates GALF every span. GALF becomes the previous span start value for GALF plus G_DY.

The value is signed such that the ninth bit and bit 31 of the GREEN alpha increment, G_DY, is interpreted as a sign bit.

See also G_DX and GALF.

Recommended Initialization Value: 0

Register Format:



Register Fields:

INTEGER[8:0]	Integer Part 9 bit integer
FRACTION[9:0]	Fractional Part 10 bit fraction

Green Light Source Component Register

name: **GALF**
type: Working
block: ALU
section: RGB ALU
reg num: 23
function: This register contains the GREEN light source component, which is interpreted as a 9 bit value with ten bit fraction. The decimal point for these values lies between bit 15 and 16.

The pixel ALU only shades using the least significant 8 bits of the integer portion; the ninth bit of alpha thus indicates saturation of the intensity value. The values are still calculated correctly up to an alpha value of 1.FF, so that saturation clipping can be handled correctly.

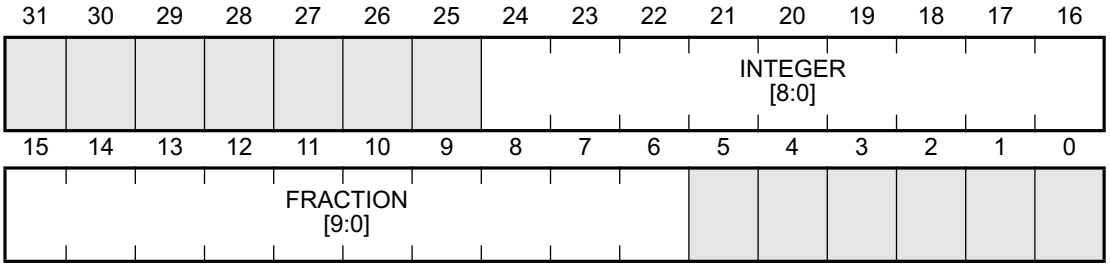
Each colour component has a separate intensity multiplier, so that coloured light sources may be modeled. The colour alpha intensities (RALF, GALF, BALF) perform an unsigned multiply of the pixel colour component values derived from the texture data stream:

```
R' = RALF * R
G' = GALF * G
B' = BALF * B
```

Additionally, in order to speed up parameter updates when white light sources are used, the green and blue components are also written when the corresponding red values (RALF, R_DX) are written, thus avoiding the need for 4 parameter writes. To set up coloured light source values, the red parameters must be written first, followed by the blue and green values.

Recommended Initialization Value: 0

Register Format:



Register Fields:

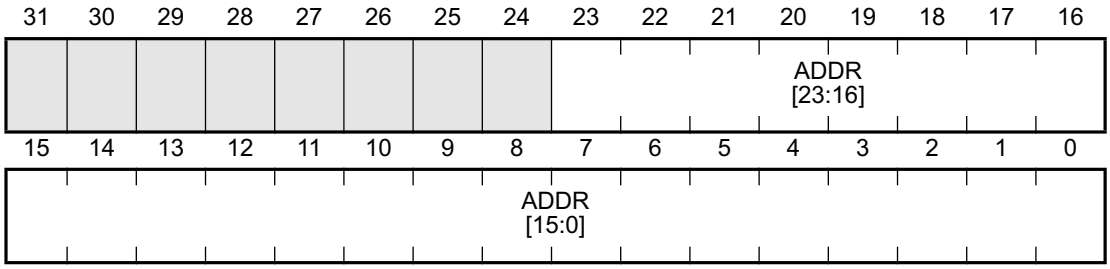
INTEGER[8:0]	Integer Part 9 bit integer
FRACTION[9:0]	Fractional Part 10 bit fraction

Screen Source Address Register

name: **IBASE**
type: Working
block: DDA
section: Screen Input DDA
reg num: 4
function: PRE is used to move or modify data already drawn on the screen, by mixing it with a texture or colour. The address for fetching the screen data comes from this register.

Recommended Initialization Value: 0

Register Format:



Register Fields:

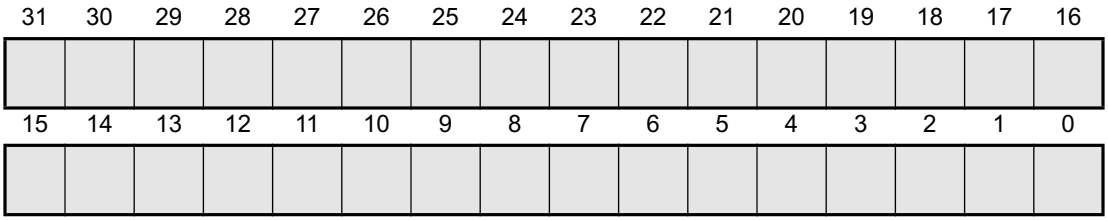
ADDR[24:0] Address
Working address for input screen buffer.

Invalidate Texture Cache Register

name: **IVTC**
type: Control
block: DDA
section: Texture Input DDA
reg num: 20
function: The invalidate texture cache register should be used to clear the texture cache. For example when a new texture replaces a texture in memory that was just used. A write of any value to this register will cause the texture to be invalidated and new values to be loaded on the next texture read.

Recommended Initialization Value: 0

Register Format:



Register Fields:

Not applicable

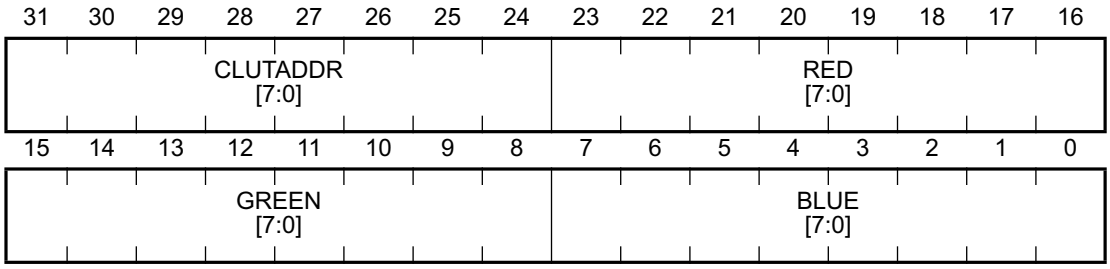
Look-up Write Register

name: **LKUP**
type: Control
block: ALU
section: Lookup RAM
reg num: 15
function: This register is used to alter values in the lookup RAM (CLUT). The value written consists of two parts; the top eight bits which are the address in the RAM in which to write the value, and twenty four bits which are either the full resolution RGB888 value or ARGB8565 value, to be written.

Synchronisation of writes to the lookup RAM are performed in the same way as writes to the count register. A write may not occur until PRE has finished the previous set of pixel operations. Note that this will effectively stop PRE activity during the lookup table load time.

Recommended Initialization Value: 0

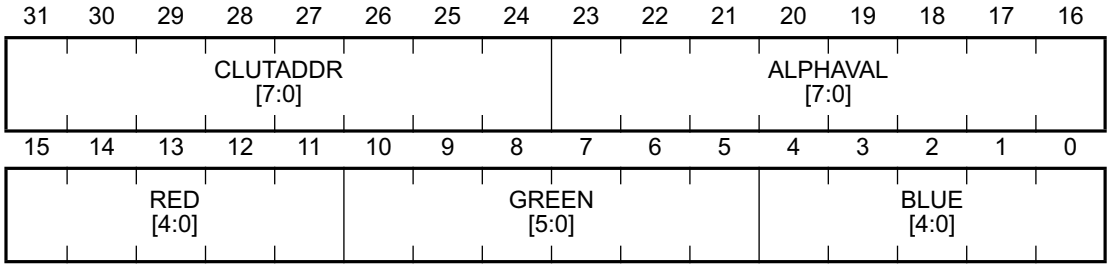
Register Format RGB888:



Register Fields:

CLUTADDR[7:0]	CLUT Address
RED[7:0]	CLUT Red
GREEN[7:0]	CLUT Green
BLUE[7:0]	CLUT Blue

Register Format RGBA5658:



Register Fields:

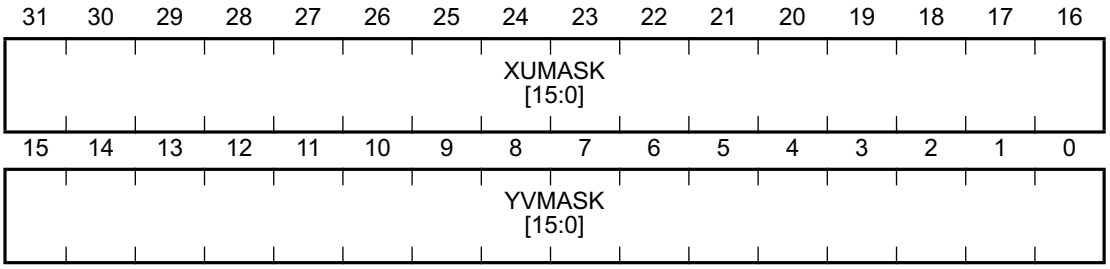
CLUTADDR[7:0]	CLUT Address
RED[4:0]	CLUT Red
ALPHAVAL[7:0]	CLUT Alpha Value
GREEN[6:0]	CLUT Green
BLUE[4:0]	CLUT Blue

Xu and Yv Mask Register

name: **MASK**
type: Control
block: DDA
section: Texture Input
reg num: 1
function: This register contains the mask for the integer parts of Xu and Yv. The Xu and Yv values are logically ANDed with the masks to perform wraparound of the Xu and Vv values and determine the size of the texture map.

Recommended Initialization Value: 0

Register Format:



Register Fields:

XUMASK[15:0] Mask for Xu Integer
16 bit mask

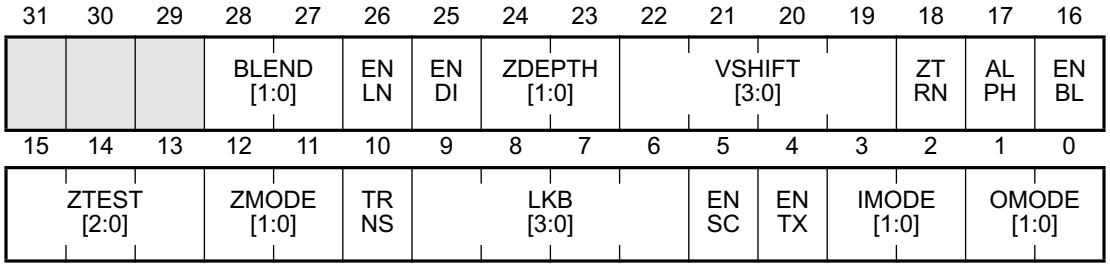
YVMASK[15:0] Mask for Yv Integer
16 bit mask

Mode Register

name: **MODE**
type: Control
block: DDA and ALU
section: Mode
reg num: 0
function: This register contains mode bits which concern the mode of texture pixels, output pixels and Z buffer data.

Recommended Initialization Value: 0

Register Format:



Register Fields:

OMODE[1:0] **Output Mode**
00 = RGB 332
01 = RGB 565
10 = RGB 888 on 32 bit boundaries
11 = *Reserved*

IMODE[1:0] **Texture Input DDA Mode**
00 = 4 bit data in 4 x 4 pixel cells - uses CLUT, packed only
01 = 8 bit data in 4 x 2 pixel cells - uses CLUT
10 = 16 bit RGB 565
11 = 32 bit RGB 888, linear only

ENTX **Enable Input Texture DDA**
0 = Colour is derived from full saturation (white) and RGB Deltas
1 = Texture data is fetched as per IMODE**.**

ENSC **Enable Input Screen DDA**
0 = The 24 bit FIXC register values are passed to the pixel ALU.
1 = Data is fetched as per OMODE

LKB[3:0] **Lookup Base**
High 4 bits of lookup address in 4 bit input mode.

TRNS **Transparency**
0 = A pixel is written for every pixel read.
1 = No pixel written, if result of texture input DDA is zero (black)

ZMODE[1:0] **Z Buffering Mode**
00 = No Z buffering
01 = Write Z always - No Z read (First Pass)
10 = Check Z & conditional pixel write; No Z write. (Last Pass)
11 = Read & compare Z; write Z & Pixel if comparison true

ZTEST[2:0] **Z Comparison Mode**
(calculated Z value versus Z buffer value)
000 = Never(0)
001 = Always(1)
010 = Less Than(N)
011 = Greater or Equal (~N)
100 = Less or Equal(N or Z)
101 = Greater(~N and ~Z)
110 = Equal (Z)
111 = Not Equal (~Z)

ENBL **Enable Bilinear Interpolation**
0 = No Bilinear Interpolation
1 = Enable Bilinear Interpolation

ALPH **Enable Alpha Map Mode**
0 = Alpha from AVAL
1 = Alpha from texture map. 4, 8 and 32 bit textures only

ZTRN **Z Transparency Mode**
0 = A Z value is written, provided Z comparison is true
1 = A Z value is written if Z compare true, and texture not transparent

VSHIFT[3:0] **Yv Shift Value**
Contains a value 0-15 which is the number of bits by which the DDA Y value will be shifted in order to form a texture address. This may be different to the number of bits set in the X portion of the mask register. In this way, a stripe of a texture may be re-used as a narrow texture.

ZDEPTH[1:0] **Z Buffer Depth**
00 = 8 bit Precision Z buffer values.
01 = 16 bit Precision Z buffer values.
10 = 32 bit Precision Z buffer values.
11 = *Reserved*

ENDI **Enable 4x4 Ordered Dither**
0 = Dithering Disabled
1 = Dithering Enabled. 8 and 16 bit pixel modes only

ENLN **Enable Linear Texture Packing**
0 = Packed Textures
1 = Linear Textures

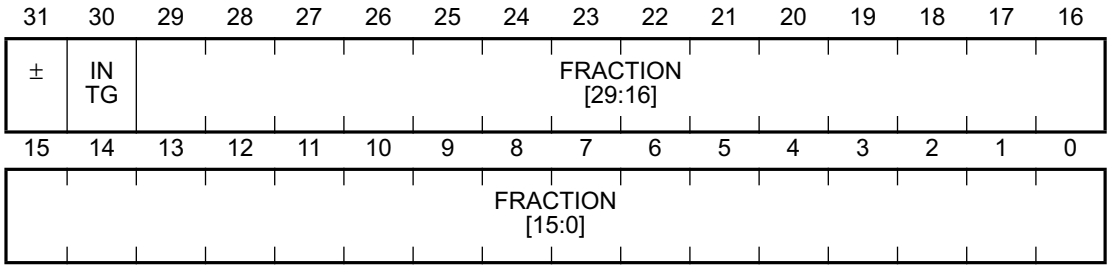
BLEND[1:0] **Alpha Blending Mode**
00 = Standard blend
01 = Summed blend
10 = Dimmed blend
11 = *Reserved*

Homogenous Q Pixel Increment Register

name: **Q_DX**
type: Increment
block: DDA
section: Texture Input DDA
reg num: 12
function: This is the homogenous co-ordinate increment added to QVAL for each pixel read.

Recommended Initialization Value: 0

Register Format:



Register Fields:

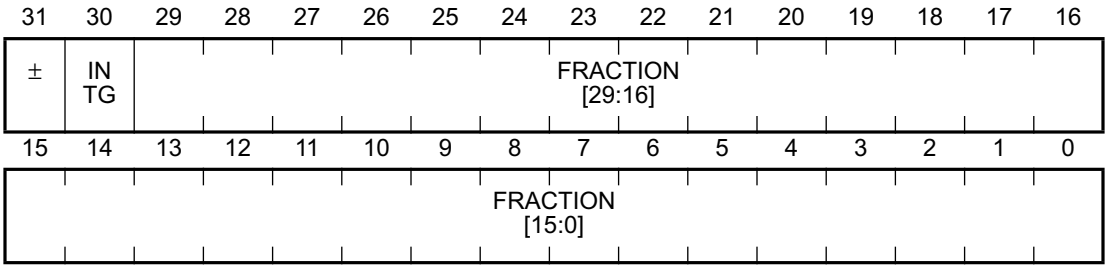
INTG	Integer Part 1 bit integer part
FRACTION[29:0]	Fractional Part 30 bit fractional part

Homogenous Q Span Increment Register

name: **Q_DY**
type: Increment
block: DDA
section: Texture Input DDA
reg num: 44
function: This is the homogenous co-ordinate increment added to QVAL for each span.

Recommended Initialization Value: 0

Register Format:



Register Fields:

INTG	Integer Part 1 bit integer part
FRACTION[29:0]	Fractional Part 30 bit fractional part

Homogenous Coordinate Register

name: **QVAL**
type: Working
block: DDA
section: Texture Input DDA
reg num: 28
function: This is the homogeneous co-ordinate for the pixel, or rather the reciprocal of the homogenous co-ordinate. It is calculated to be H/W, where H is a scaling factor and W is the actual homogenous co-ordinate. It is used with Q_DX and Q_DY to perform perspective corrections. For each span, the Z, Xu, and Yv values are known at the ends of the span.

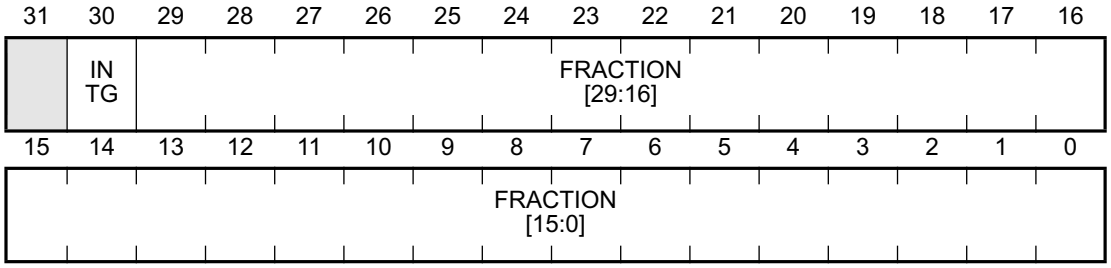
The value of the homogeneous co-ordinate is not fixed, but the relative values at both ends of the span will be determined by the ratio of Z at each end of the span. The dynamic range of QVAL over the span is constrained to lie between 0 and 1 in the PRE implementation. The scan converter therefore calculates the start QVAL and increment Q_DX according to these constraints. Note that the scan converter will need to perform divides in order to calculate these span end values.

The Xu and Yv increments (U_DX and V_DX) are also modified by the homogeneous correction, but these are calculated by the scan conversion program on the embedded processor. The Xu and Yv values for each pixel are then modified to be UVAL=H*Xu/W, VVAL=H*Yv/W, at each pixel position. The corrected values are then used as indices into a texture map. Note that the perspective correction unit in PRE only uses the 10 bit integer and 6 bit fractional part of UVAL and VVAL.

See also UVAL and VVAL.

Recommended Initialization Value: $1.0000000 (i.e. $40000000 should be sent to QVAL). This value points to the reciprocal ROM table where no adjustment is made for perspective correction.

Register Format:



Register Fields:

INTG	Integer Part 1 bit integer part
FRACTION[29:0]	Fractional Part 30 bit fractional part

Red Light-source Pixel Increment Register

name: **R_DX**
type: Increment
block: ALU
section: RGB ALU
reg num: 6
function: This register contains the RED light-source increment, which is interpreted as a 9 bit value with ten bit fraction. The decimal point for these values lies between bit 15 and 16.

The value is signed such that the ninth bit and bit 31 of the RED alpha increment, R_DX, is interpreted as a sign bit.

At the end of each Pixel Engine cycle, the RED light source value (RALF) is incremented by the increment R_DX:

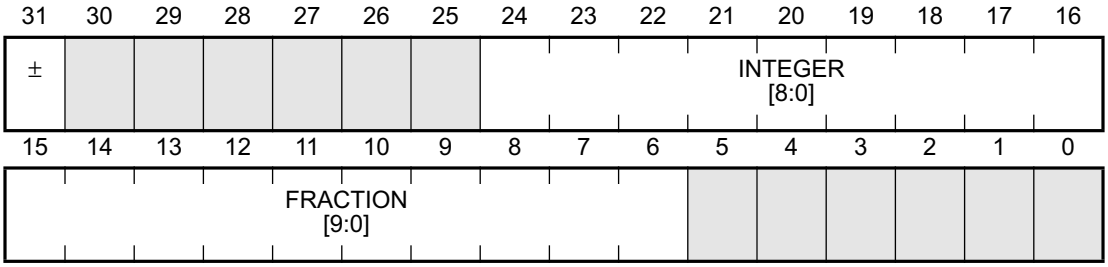
```
RALF(i+1)= RALF(i) + R_DX
```

If the value of RALF after the increment would be less than zero (underflow), the new value is set to zero. No overflow is detected, since it is assumed that the oversaturation range provides sufficient protection.

Additionally, in order to speed up parameter updates when white light sources are used, the green and blue increments are also written when the RED increment (R_DX) is written, thus avoiding the need for 2 parameter writes. To set up coloured light source values, the red parameters must be written first, followed by the blue and green values.

Recommended Initialization Value: 0

Register Format:



Register Fields:

INTEGER[8:0]	Integer Part 9 bit integer
FRACTION[9:0]	Fractional Part 10 bit fraction

Red Light-source Span Increment Register

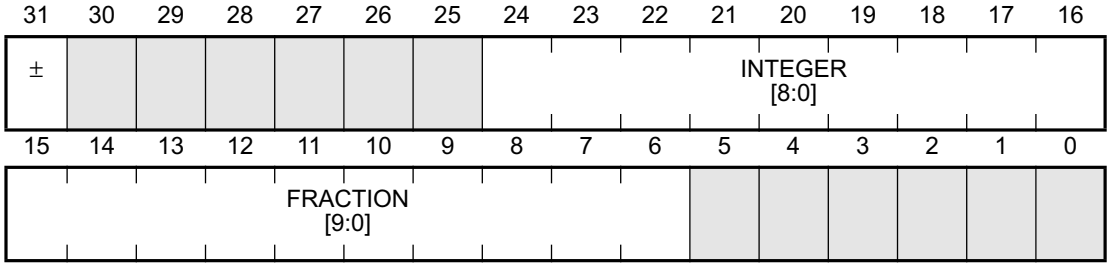
name: **R_DY**
type: Increment
block: ALU
section: RGB ALU
reg num: 38
function: This register contains the RED light-source intensity increment on a per span basis. It works in a similar way to R_DX and updates RALF every span. RALF becomes the previous span start value for RALF plus R_DY.

The value is signed such that the ninth bit and bit 31 of the RED alpha increment, R_DY, is interpreted as a sign bit.

See also R_DX and RALF.

Recommended Initialization Value: 0

Register Format:



Register Fields:

INTEGER[8:0]	Integer Part 9 bit integer
FRACTION[9:0]	Fractional Part 10 bit fraction

Red Light Source Component Register

name: **RALF**
type: Working
block: ALU
section: RGB ALU
reg num: 22
function: This register contains the RED light source component, which is interpreted as a 9 bit value with ten bit fraction. The decimal point for these values lies between bit 15 and 16.

The pixel ALU only shades using the least significant 8 bits of the integer portion; the ninth bit of alpha thus indicates saturation of the intensity value. The values are still calculated correctly up to an alpha value of 1.FF, so that saturation clipping can be handled correctly.

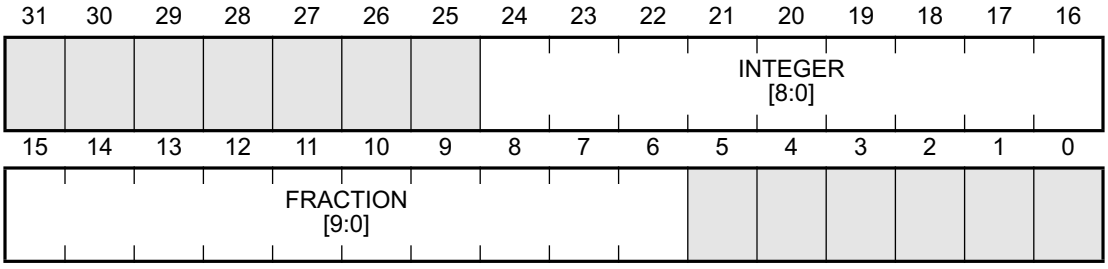
Each colour component has a separate intensity multiplier, so that coloured light sources may be modelled. The colour alpha intensities (RALF, GALF, BALF) perform an unsigned multiply of the pixel colour component values derived from the texture data stream:

```
R' = RALF * R
G' = GALF * G
B' = BALF * B
```

Additionally, in order to speed up parameter updates when white light sources are used, the green and blue components are also written when the corresponding red values (RALF, R_DX) are written, thus avoiding the need for 4 parameter writes. To set up coloured light source values, the red parameters must be written first, followed by the blue and green values.

Recommended Initialization Value: 0

Register Format:



Register Fields:

INTEGER[8:0]	Integer Part 9 bit integer
FRACTION[9:0]	Fractional Part 10 bit fraction

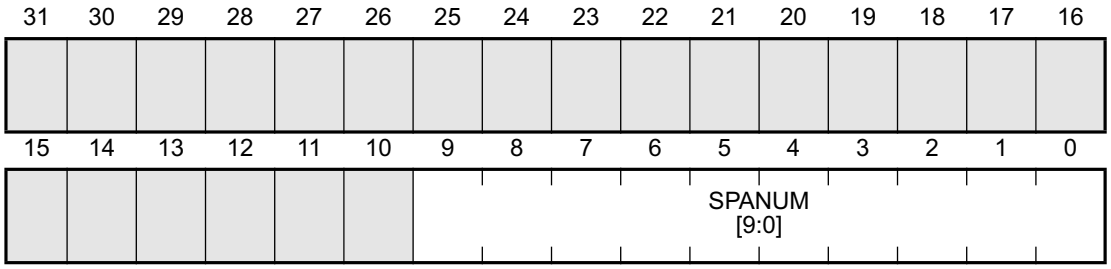
Spans in Bottom Triangle Register

name: **S_BOT**
type: Control
block: DDA
section: DDA and DMA
reg num: 47
function: This register is used in PRE to control the number of spans that are drawn on the screen on a per triangle basis. S_TOP and S_BOT together determine the point of inflection on a triangle.

See also XENDB, XENDT, XSTART, XB_DY, XT_DY and XS_DY.

Recommended Initialization Value: 0

Register Format:



Register Fields:

SPANUM[9:0] Integer number of spans

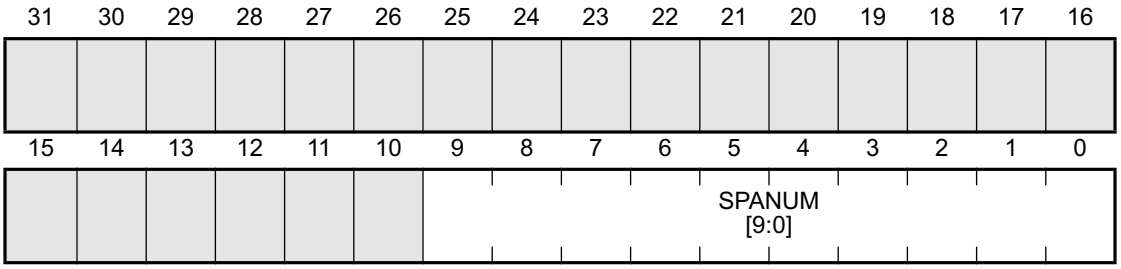
Spans in Top Triangle Register

name: **S_TOP**
type: Control
block: DDA
section: DDA and DMA
reg num: 46
function: This register is used in PRE to control the number of spans that are drawn on the screen on a per triangle basis. S_TOP and S_BOT together determine the point of inflection on a triangle.

See also XENDB, XENDT, XSTART, XB_DY, XT_DY and XS_DY.

Recommended Initialization Value: 0

Register Format:



Register Fields:

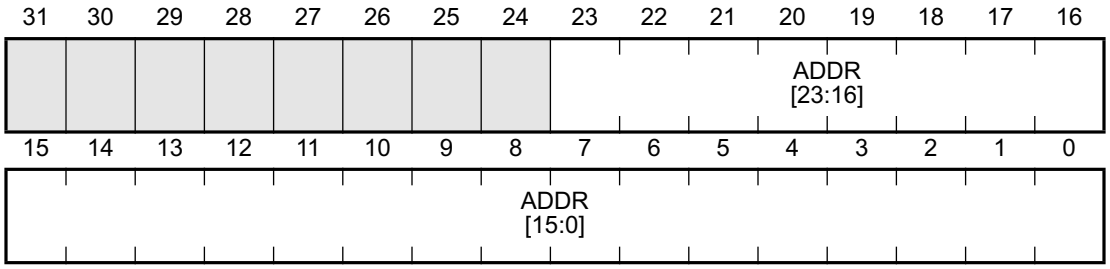
SPANUM[9:0] Integer number of spans

Screen Output Address Register

name: **SBASE**
type: Working
block: DDA
section: Output DMA
reg num: 5
function: The output screen buffer can be at a different address from the input screen buffer (see IBASE). This register contains the address at which the current pixel will be written.

Recommended Initialization Value: 0

Register Format:



Register Fields:

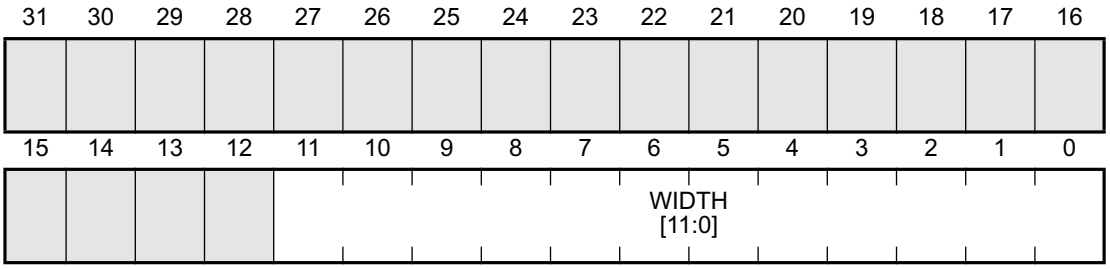
ADDR[24:0] Address
Working address for output screen buffer.

Screen Width in Pixels Register

name: **SCRW**
type: Control
block: DDA
section: DDA and DMA
reg num: 19
function: This register contains the screen width in pixels. The address increments through memory for Z, Input Screen and Output Screen are calculated from SCRW together with the pixel size set in the MODE register.

Recommended Initialization Value: 0

Register Format:



Register Fields:

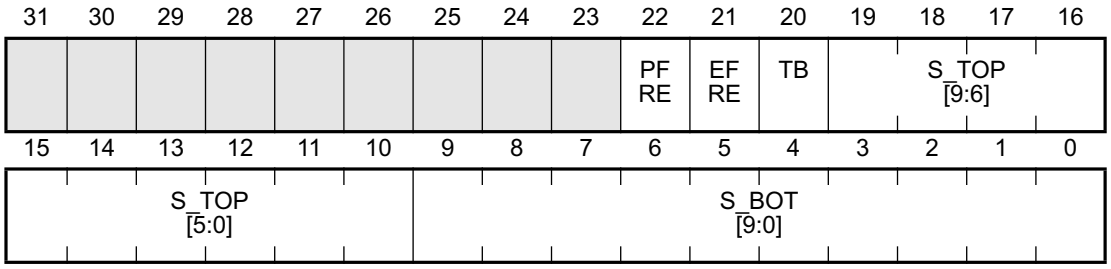
WIDTH[11:0] Screen width in Pixels

Status Register

name: **STATUS**
type: Command
block: CMD
section: Mode
reg num: 31
function: This register when read using the direct register read-back mechanism, contains the triangle span numbers, and whether PRE is active.

A write of any value to this register, through the normal FIFO update mechanism, will cause PRE to halt until a reset has occurred. PRE should only be halted during exceptional circumstances.

Register Format:



Register Fields:

S_BOT[9:0] **Spans in Bottom Triangle**
The number of spans set for the bottom triangle. This is a copy of the S_BOT register

S_TOP[9:0] **Spans in Top Triangle**
The number of spans set for the top triangle. This is a copy of the S_TOP register

TB **Top or Bottom Triangle Running**
0 = The top triangle is being processed
1 = The bottom triangle is being processed

EFRE **Edge Engine Free**
0 = Edge interpolator is busy
1 = The edge interpolator is free to take more commands from the TSC

PFRE **Pixel Engine Free**
0 = Span renderer is busy.
1 = Span renderer is not executing any commands and there are no pending writes to memory.

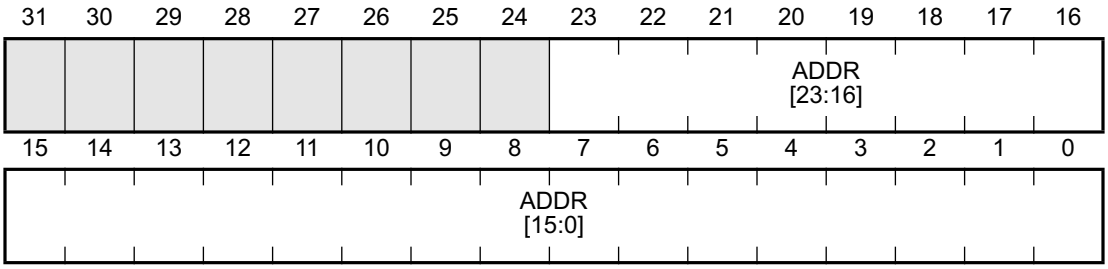
Texture Base Address Register

name: **TBASE**
type: Control
block: DDA
section: Texture Input DDA
reg num: 2
function: This register contains the base address for the current texture.

The other 'Texture Input DDA' registers calculate an offset from this address for the required pixels.

Recommended Initialization Value: 0

Register Format:



Register Fields:

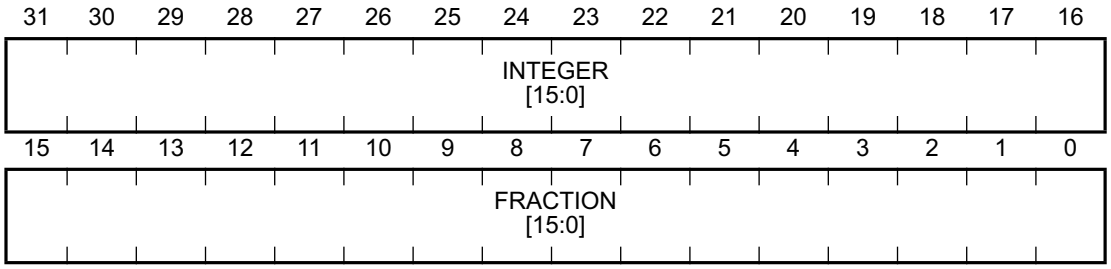
ADDR[24:0] Address
Base address for current texture.

Xu Pixel Increment Register

name: **U_DX**
type: Increment
block: DDA
section: Texture Input DDA
reg num: 7
function: This is the increment added to UVAL for each pixel read.

Recommended Initialization Value: 0

Register Format:



Register Fields:

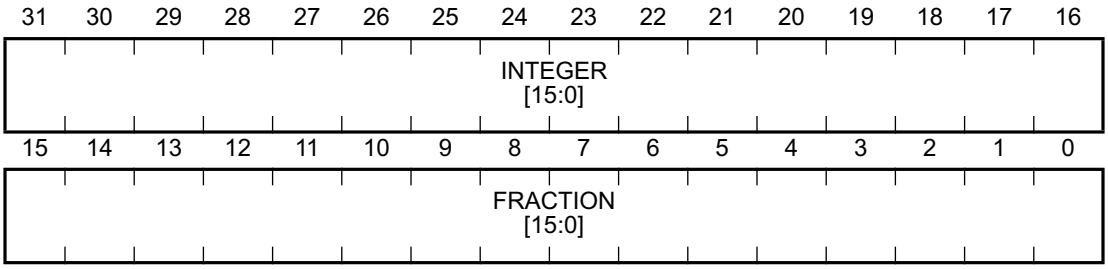
INTEGER[15:0]	Integer Part 16 bit integer part
FRACTION[15:0]	Fractional Part 16 bit fractional part

Xu Span Increment Register

name: **U_DY**
type: Increment
block: DDA
section: Texture Input DDA
reg num: 39
function: This is the increment added to UVAL for each new span.

Recommended Initialization Value: 0

Register Format



Register Fields:

INTEGER[15:0] Integer Part
16 bit integer part

FRACTION[15:0] Fractional Part
16 bit fractional part

Texture Source Xu Register

name: **UVAL**
type: Working
block: DDA
section: Texture Input DDA
reg num: 25
function: This register when masked and combined with VVAL and MASK produces the offset from the base address TBASE. The format of the register is 16.16 with the fractional point between bits 15 and 16.

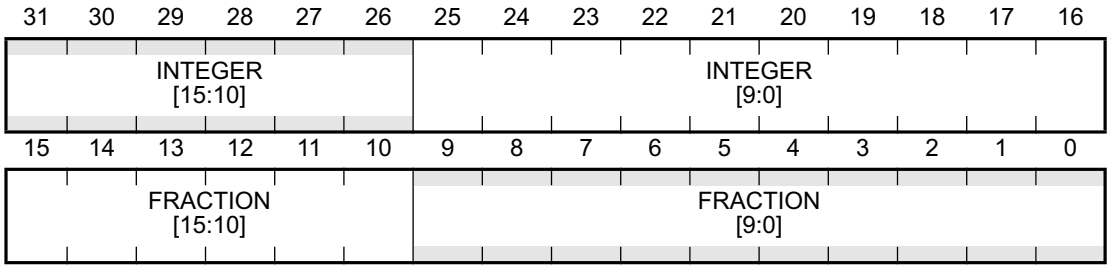
When perspective correction is enabled, the INTEGER[9:0] and FRACTION[15:10] parts of UVAL are modified by the homogenous co-ordinate QVAL, and the remaining parts of UVAL interpreted as 0.

When Bilinear interpolation is enabled the FRACTION[15:10] part is used for the fractional part in the blending Formula.

See also QVAL.

Recommended Initialization Value: 0

Register Format:



Register Fields:

INTEGER[15:0] Integer Part
16 bit integer part

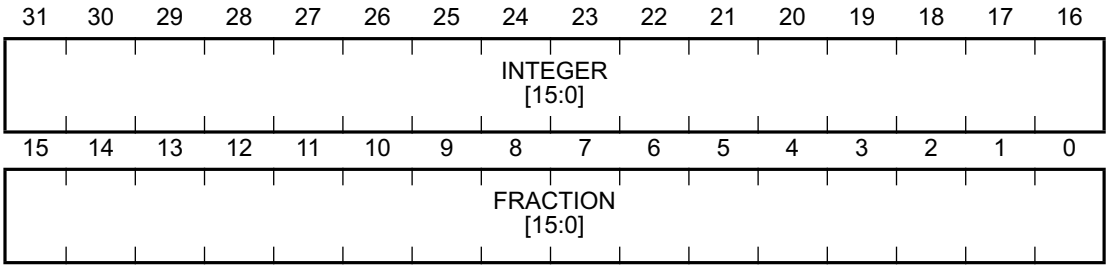
FRACTION[15:0] Fractional Part
16 bit fractional part

Yv Pixel Increment Register

name: **V_DX**
type: Increment
block: DDA
section: Texture Input DDA
reg num: 8
function: This is the increment added to VVAL for each pixel read.

Recommended Initialization Value: 0

Register Format:



Register Fields:

INTEGER[15:0] Integer Part
16 bit integer part

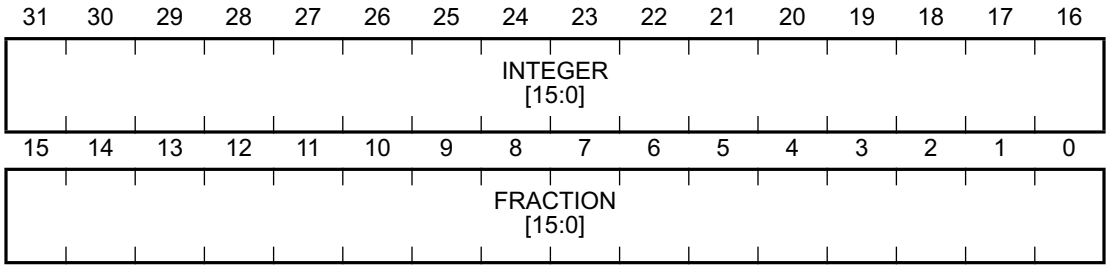
FRACTION[15:0] Fractional Part
16 bit fractional part

Yv Span Increment Register

name: **V_DY**
type: Increment
block: DDA
section: Texture Input DDA
reg num: 40
function: This is the increment added to VVAL for each new span.

Recommended Initialization Value: 0

Register Format:



Register Fields:

INTEGER[15:0] Integer Part
16 bit integer part

FRACTION[15:0] Fractional Part
16 bit fractional part

Texture Source Yv Register

name: **VVAL**
type: Working
block: DDA
section: Texture Input DDA
reg num: 26
function: This register when masked and combined with UVAL and MASK produces the offset from the base address TBASE. The format of the register is 16.16 with the fractional point between bits 15 and 16.

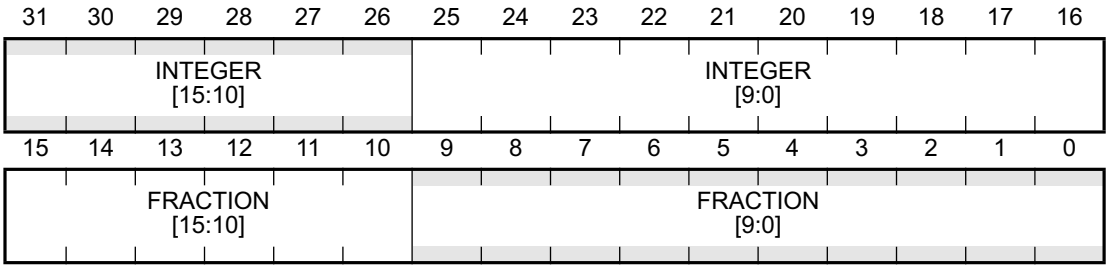
When perspective correction is enabled, the INTEGER[9:0] and FRACTION[15:10] parts of VVAL are modified by the homogenous co-ordinate QVAL, and the remaining parts of VVAL interpreted as 0.

When Bilinear interpolation is enabled the FRACTION[15:10] part is used for the fractional part in the blending formula.

See also QVAL.

Recommended Initialization Value: 0

Register Format:



Register Fields:

INTEGER[15:0] Integer Part
16 bit integer part

FRACTION[15:0] Fractional Part
16 bit fractional part

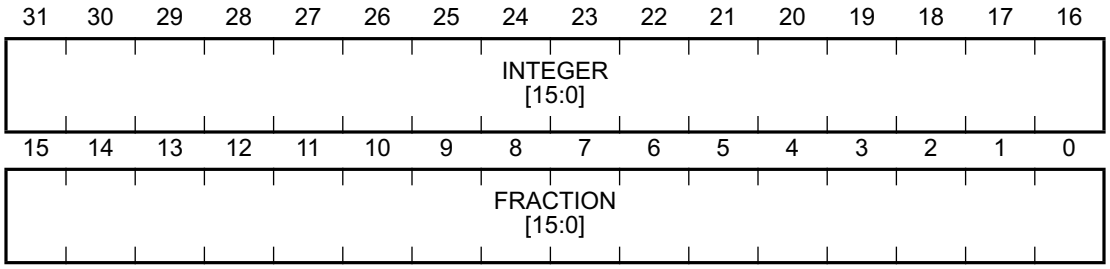
X End Span Increment Register

name: **XB_DY**
type: Increment
block: DDA
section: DDA and DMA
reg num: 32
function: This register contains the gradient of the bottom triangle edge. The other bottom triangle edge gradient is represented by the dominant edge gradient XS_DY.

See also XT_DY and XS_DY.

Recommended Initialization Value: 0

Register Format:



Register Fields:

INTEGER[15:0] Integer Part
16 bit integer part

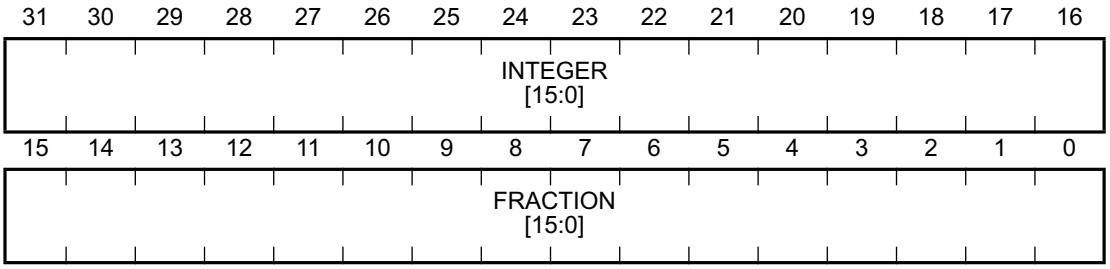
FRACTION[15:0] Fractional Part
16 bit fractional part

X End Coordinate/ Bottom Register

name: **XENDB**
type: Control
block: DDA
section: DDA and DMA
reg num: 16
function: This register contains the X co-ordinate for last pixel of the first span on the bottom triangle. The vertex can be represented as having a sub-pixel X co-ordinate value.

Recommended Initialization Value: 0

Register Format:



Register Fields:

INTEGER[15:0] Integer Part
16 bit integer part

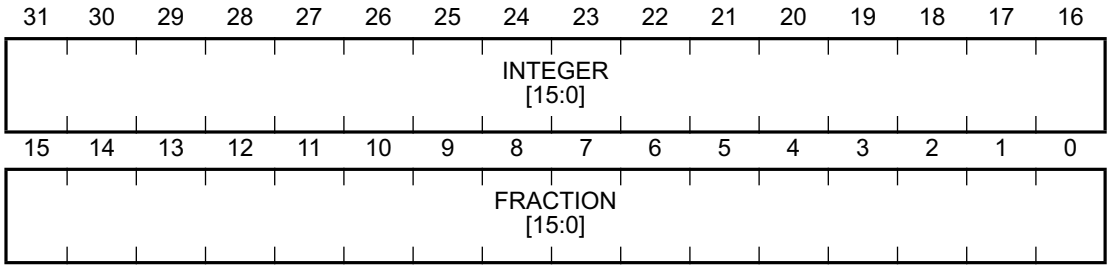
FRACTION[15:0] Fractional Part
16 bit fractional part

X End Coordinate/ Top Register

name: **XENDT**
type: Control
block: DDA
section: DDA and DMA
reg num: 17
function: This register contains the X co-ordinate for the last pixel of the first span on the top triangle. The vertex can be represented as having a sub-pixel X co-ordinate value.

Recommended Initialization Value: 0

Register Format:



Register Fields:

INTEGER[15:0] Integer Part
16 bit integer part

FRACTION[15:0] Fractional Part
16 bit fractional part

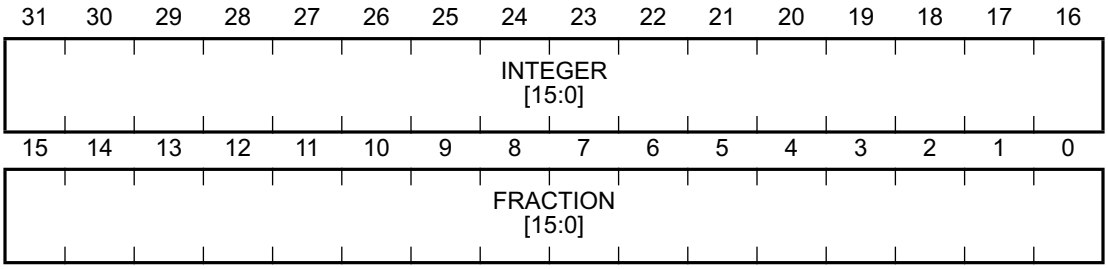
X Start Span Increment Register

name: **XS_DY**
type: Increment
block: DDA
section: DDA and DMA
reg num: 34
function: This register contains the gradient of the dominant edge across the top and bottom triangles. The other triangle edge gradients is represented by the top edge gradient register XB_DY, and the bottom edge gradient register XT_DY.

See also XB_DY and XT_DY.

Recommended Initialization Value: 0

Register Format:



Register Fields:

INTEGER[15:0] Integer Part
16 bit integer part

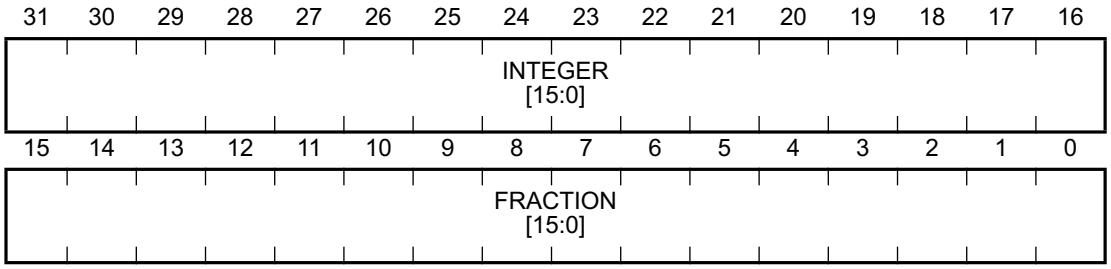
FRACTION[15:0] Fractional Part
16 bit fractional part

X Start Coordinate Register

name: **XSTART**
type: Control
block: DDA
section: DDA and DMA
reg num: 18
function: This register contains the X co-ordinate for the first and top-most vertex for the top and bottom triangle pair. The vertex can be represented as having a sub-pixel X co-ordinate value.

Recommended Initialization Value: 0

Register Format:



Register Fields:

INTEGER[15:0] Integer Part
16 bit integer part

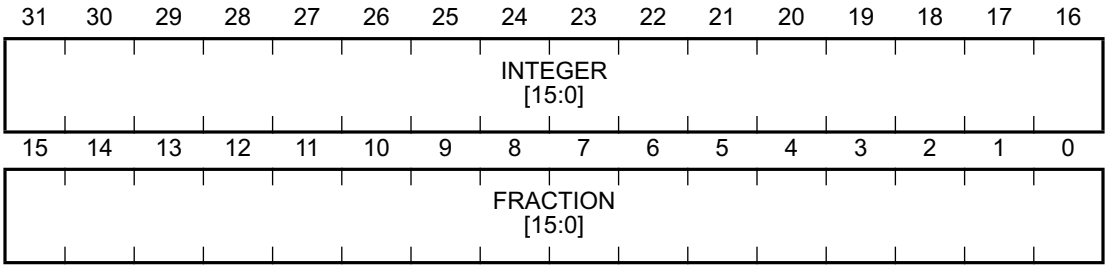
FRACTION[15:0] Fractional Part
16 bit fractional part

X End Span Increment/Top Register

name: **XT_DY**
type: Increment
block: DDA
section: DDA and DMA
reg num: 33
function: This register contains the gradient of the top triangle edge. The other top triangle edge gradient is represented by the dominant edge gradient XS_DY. See also XB_DY and XS_DY.

Recommended Initialization Value: 0

Register Format:



Register Fields:

INTEGER[15:0] Integer Part
16 bit integer part

FRACTION[15:0] Fractional Part
16 bit fractional part

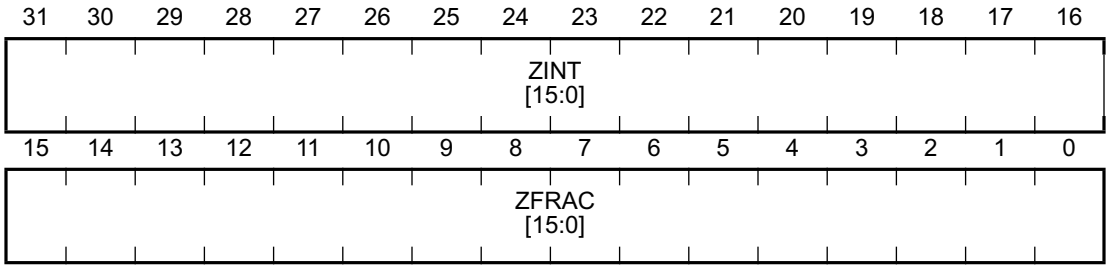
Z Value Pixel Increment Register

name: **Z_DX**
type: Increment
block: ALU
section: RGB ALU
reg num: 11
function: This register is the Z value increment for the current pixel about to be plotted.

See also ZBASE and ZVAL

Recommended Initialization Value: 0

Register Format:



Register Fields:

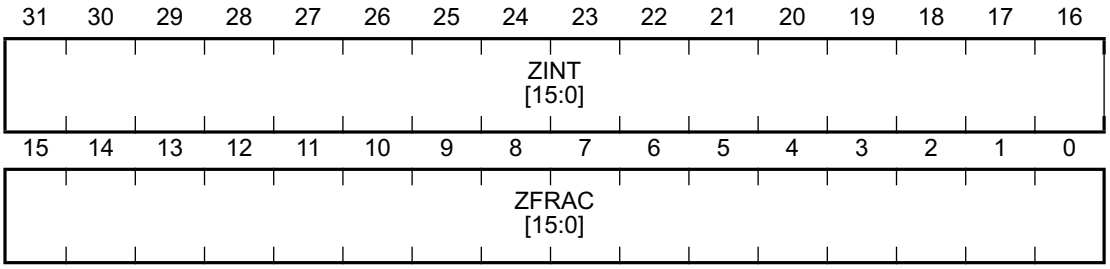
ZINT[15:0]	Integer part of Z Increment
ZFRAC[15:0]	Fractional part of Z Increment

Z Value Span Increment Register

name: **Z_DY**
type: Increment
block: ALU
section: RGB ALU
reg num: 43
function: This register is the Z value increment to be added to ZVAL for each span. See also ZBASE and ZVAL.

Recommended Initialization Value: 0

Register Format:



Register Fields:

ZINT[15:0]	Integer part of Z Increment
ZFRAC[15:0]	Fractional part of Z Increment

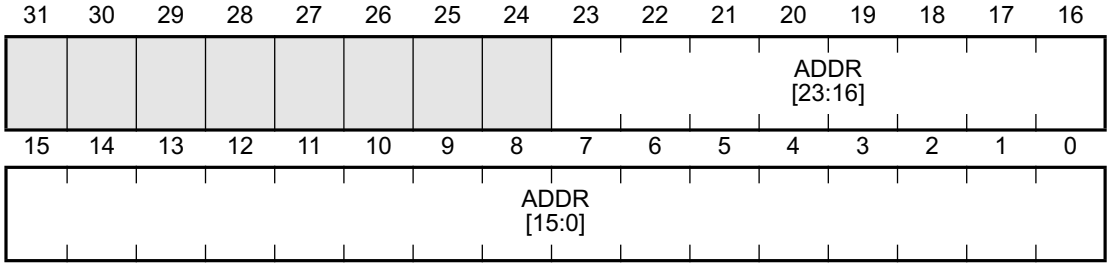
Z Buffer Base Address Register

name: **ZBASE**
type: Working
block: DDA
section: Input DDA
reg num: 3
function: This register contains the current address in the Z buffer. The same address is used for both reading and writing to the Z buffer. This involves some duplication of registers between the DDA and arithmetic blocks.

The incrementers for the input DMA channel using IBASE, is re-used as the Z buffer address increment. This ties the Z value sizes to the same as pixel sizes.

Recommended Initialization Value: 0

Register Format:



Register Fields:

ADDR[24:0] Address
Base address for Z buffer.

Current Z Value Register

name: **ZVAL**
type: Working
block: ALU
section: RGB ALU
reg num: 27
function: This register is the Z value for the current pixel about to be plotted. If Z buffering is enabled, the Z value from the Z source buffer is compared with this Z. If the comparison between the new Z (ZVAL) and the old Z returns true, then the pixel will be written to the screen buffer, and the Z value to the Z buffer.

The Z buffering mode may also be set, so that no Z comparison occurs, and pixels and Z are always written. This is useful in setting backgrounds, when it is known beforehand that no Z masking should occur. In this case, no Z read from the Z buffer occurs, thus economising on memory bandwidth.

A mode which always checks Z, and hence writes pixels conditionally, but never writes Z, is included. This is useful for the last render operation in a frame and translucency effects.

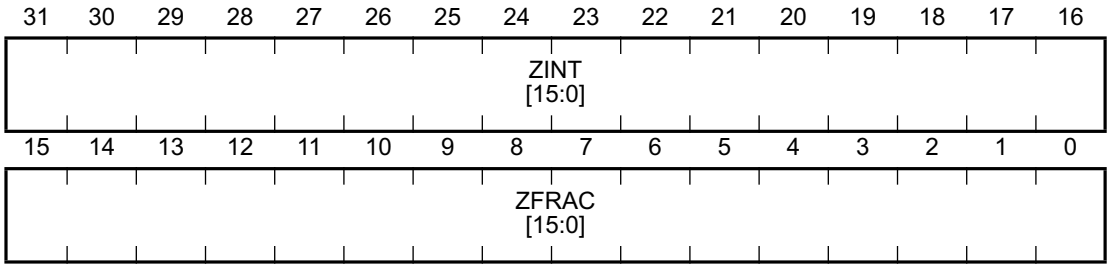
The Z buffering mode is set using the ZMODE bits in the MODE register (MODE).

The Z value is calculated to 32 bits containing both the integer and fractional parts. See also ZBASE.

When 8 and 16 bit Z depth is selected the value from the Z data buffer is the top 8 or 16 bits of the integer part of the internal Z value calculated by PRE. During Z comparisons the unused bits for 8 and 16 bit Z are ignored for both the Z data buffer value and the calculated Z. In 32 bit Z the value returned from the Z data buffer value is 32 bit (16.16) which contains both the integer and fractional parts.

Recommended Initialization Value: 0

Register Format:



Register Fields:

ZINT[15:0]	Integer part of current Z value
ZFRAC[15:0]	Fractional part of current Z value

PRE Programming

The PRE pipeline performs a sequence of operations that are designed to perform the most complex rendering algorithm, namely perspective texture mapped polygon rendering. Other algorithms are special cases of this algorithm.

In order to draw a polygon in the screen buffer, it is first broken down into a set of stripes, one pixel wide; this process is known as scan conversion. Some of this process is performed in software on the TSC, which passes a set of triangle parameters, calculated during the setup phase, to PRE, which then draws the triangle in the screen buffer.

The calculations that are required for each pixel are always the same, and always occur in the same order. Therefore, PRE is arranged as a series of stages (a 'pipeline'), each of which performs one stage of the calculation; the output of each stage is latched or stored in a FIFO RAM before it is passed to the next stage. The throughput of the pipeline is therefore one pixel per PRE clock cycle, although it takes several cycles for an individual pixel to be calculated. Because several pixels are being calculated at once, some care must be taken to change PRE parameters at the correct time.

Initialization and Default Value Set-up

The MODE register is cleared to 0 after a hardware reset. However, the working and increment registers would generally be set up on a per triangle basis anyway and actually reset to the undefined state. To put PRE into a state where all functions are disabled *all* registers should be set up with their recommended initialization value. Of particular note, when using texture maps the QVAL register must be initialized correctly to disable perspective correction.

Starting

Under normal circumstances, all parameter registers must be reset for each new triangle. If this is not done, on the new span, all such parameters will actually take on the previous span start values.

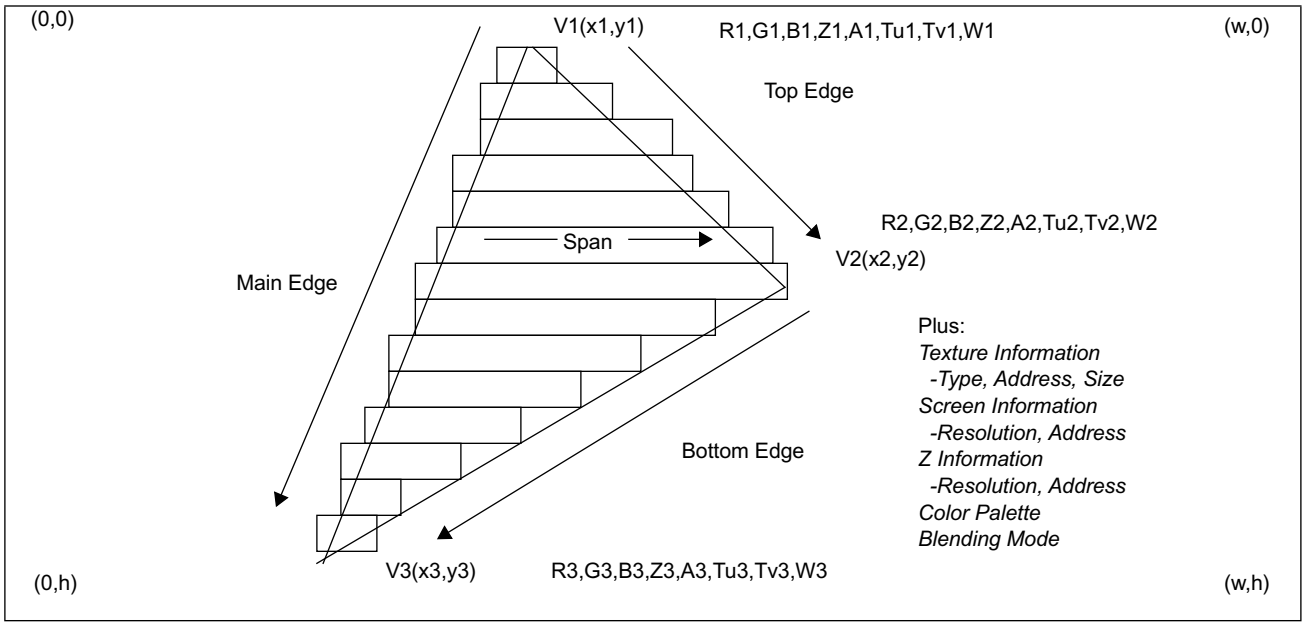
Pixel Output

The final pixel color must be written to the correct position in the screen buffer. The address is calculated by adding an increment, derived from the pixel size and screen width, to a base address at each PRE step.

Before the pixel value may be written to the screen, it is formatted. The calculated pixel value consists of three eight bit precision color values, so the RGB888 value is truncated to RGB565 or RGB332 for 16 bit or 8 bit modes respectively. The truncation process chops off the bottom bits of the color values, i.e. for 16 bit color, Red and Green lose the bottom 3 bits and Green loses the bottom 2 bits.

The pixel values are not written directly to the screen. When each pixel is sixteen bits wide, and is generally adjacent to the previous pixel, the utilization of the bus bandwidth may be minimized by performing the screen writes in bursts. The pixel values are therefore written to a pixel buffer, which stores the values until they are written out by the memory controller (see Figure 54).

Figure 54. Formatted Pixel Output



Register Grouping

The register groupings are defined in the following paragraphs.

Per Triangle Parameters

Based on Figure 54, a triangle with vertices V1, V2, and V3, the triangle parameters can be obtained. Once the dominant edge has been determined it is necessary to calculate the slopes for each edge of the triangle - for this the following calculations must be performed:

$$m1 = (x3 - x1) / (y3 - y1)$$
$$m2 = (x2 - x1) / (y2 - y1)$$
$$m3 = (x3 - x2) / (y3 - y2)$$

These calculations will be performed by the embedded processor core.

Once the slopes have been calculated, PRE must be loaded with a set of values, some of which are derived from these slopes.

The total set of parameters to be loaded into PRE for each triangle are:

- MODE: Mode register
- XENDB: X End coordinate for first span of lower triangle (x2 in Figure 54)
- XENDT: X End coordinate for first span of upper triangle (x1 or x1+1 in Figure 54)
- XSTART: X start coordinate for dominant edge (x1 in Figure 54)
- S_TOP: Number of spans in upper triangle
- S_BOT: Number of spans in lower triangle
- MASK: Texture U & V masks
- TBASE: Texture base address
- ZBASE: Z buffer base address for top span
- IBASE: Input source screen buffer base address for top span
- SBASE: Output screen buffer base address for top span
- FIXC: 24 bit fixed blend color
- LKUP: CLUT write port (addr.R.G.B)
- SCRW: Screen width in pixels

Edge and Span Working Registers

The following parameters represent shared resources between the edge and span engines, and must be loaded into PRE as starting values for the triangle. If the edge interpolator were not present these values would need to be calculated on a per span basis by the TSC. The edge interpolator modifies these values on a per span basis, and the span renderer modifies them on a per pixel basis.

- RALF: Red component alpha
- GALF: Green component alpha
- BALF: Blue component alpha
- UVAL: U start value
- VVAL: V start value
- QVAL: 1/W start value
- ZVAL: Z start value
- AVAL: Start value of background blending coefficient

The start address of span and length of span are generated by the edge interpolator and loaded into the span renderer each span.

Edge Interpolation Registers

The edge interpolation process must calculate the values of the various rendering parameters required by the span renderer for each span of the triangle to be rendered. This involves stepping along the dominant edge of the triangle in Y and obtaining the values for each end of the span at each step. At the point of inflection where the other two edges connect there will be a change in the value of the x increment used, but all other values can remain unaltered.

The following list represents the registers that are used by the edge interpolator for this process.

- XB_DY: Xend increment per scan line for lower triangle (from m3)
- XT_DY: Xend increment per scan line for upper triangle (from m2)
- XS_DY: Xstart increment per scan line for dominant edge (from m1)
- R_DY: Red increment per span
- G_DY: Green increment per span
- B_DY: Blue increment per span
- U_DY: U coordinate increment per span
- V_DY: V coordinate increment per span
- Z_DY: Z increment per span
- Q_DY: 1/W increment per span
- A_DY: Alpha increment per span

Span Interpolation Registers

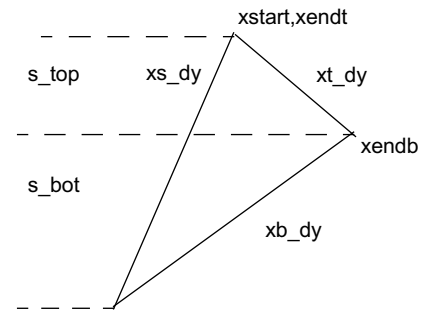
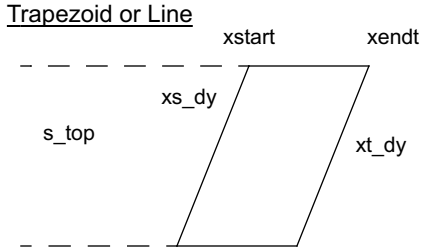
The span interpolation process modifies the working registers across a span. As the span is plotted in the horizontal direction the DX registers are added to the working registers and new values obtained for each pixel that is drawn.

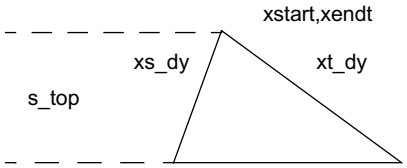
- R_DX: Red increment per pixel
- U_DX: U coordinate increment per pixel
- V_DX: V coordinate increment per pixel
- G_DX: Green increment per pixel
- B_DX: Blue increment per pixel
- Z_DX: Z increment per pixel
- Q_DX: 1/W increment per pixel
- A_DX: Alpha increment per pixel

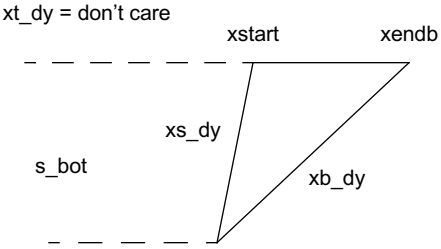
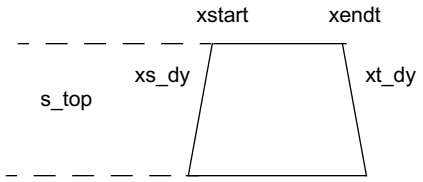
Typical Rendering Shapes

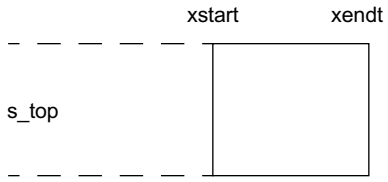
Table 172 shows typical rendering shapes.

Table 172. Typical Rendering Shapes

<table>
<tr>
<td>Triangle
 
</td>
<td>Trapezoid or Line
 

xs_dy = xt_dy
s_bot = 0
xendb = don’t care
xb_dy = don’t care</td>
</tr>
<tr>
<td>Flat Bottomed Triangle
 

s_bot = 0
xendb = don’t care
xb_dy = don’t care</td>
<td>Flat Topped Triangle
s_top = 0
xendt = don’t care
xt_dy = don’t care
 
</td>
</tr>
<tr>
<td>Flat Top and Bottomed Quad
 

s_bot = 0
xendb = don’t care
xb_dy = don’t care</td>
<td>Rectangle
 

xs_dy = 0
xt_dy = 0
s_bot = 0
xendb = don’t care
xb_dy = don’t care</td>
</tr>
</table>

PRE Programming Examples

Flat Shading

Assume we wish to display a triangle of shade ‘white’, with vertices V1, V2 and V3. The screen is ***w*** pixels wide by ***h*** pixels high. The fixed color is made up of FIXC and the RGB light source working components. Note that texture mapping needs to be disabled for flat shading. The steps would be as follows:

1. Set Up Output Mode to be RGB565
   Omode = 1
   EnableTex = 0
   MODE = (EnableTex<<4) + Omode
2. Set Up Output Screen Address
   SBASE = address for V1
3. Set Up Flat Color RGB and Reset Fixed Colors
   RALF = Red Value
   R_DX = 0
   R_DY = 0
   GALF = Green Value
   G_DX = 0
   G_DY = 0
   BALF = Blue Value
   B_DX = 0
   B_DY = 0
   FIXC = 0
   LKUP = 0
4. Work Out Y Increments
   The gradient for the main edge: m1 = (x3-x1)/(y3-y1)
   The gradient for the top edge: m2 = (x2-x1)/(y2-y1)
   The gradient for the bottom edge: m3 = (x3-x2)/(y3-y2)

   XS_DY = m1
   XT_DY = m2
   XB_DY = m3
5. Set Up Co-ordinate Parameters
   SCRW = w
   XSTART = x1
   XENDT = x1+1
   XENDB = x2

6 Start PRE
PRE will start when S_BOT is written.
The height of the top triangle: Ntop = y2-y1
The height of the bottom triangle: Nbottom = y3-y2

S_TOP = Ntop
S_BOT = Nbottom
wait for PRE to finish: test STATUS *register*

Z Buffering

When a polygon is rendered, at the same time that a color is written to the screen buffer, a value which represents the distance of that point on the polygon from the viewpoint (the Z co-ordinate) is written to a corresponding position in a Z buffer. If, at some later time another polygon is rendered, which would write a color value to the same pixel, this Z value is first checked. If the Z value of the new pixel is closer to the viewpoint than the previously rendered pixel, then the new pixel value is written to the screen buffer, and the new Z value written to the Z buffer. If the current pixel is further away than the previously written pixel, the old value is left unchanged.

The Z buffer will thus ensure correct rendering of both overlapping and arbitrarily interpenetrating polygons, even if the polygons are completely unsorted by distance. Higher performance may be achieved for the renderer, however, by rendering objects in the foreground before those in the background. An exception to this is the backdrop which should be painted first in order to ensure the Z buffer takes on a known value at the start of scene rendering.

Only one Z buffer is required for a 3-D rendering system, compared with two screen buffers. This is because the Z buffer does not need to be displayed: it can be re-used for the next scene render.

For this example, assume that the triangle is the same size and position as before, and we use the same gradient results. For Z buffering we also have 3 ‘Z’ values associated with each vertex: Z1, Z2, Z3. The Z buffer address is the start of the Z buffer which associates a Z value for each pixel on the screen. In this case we shall output a pixel if the calculated Z value is greater than the value in the Z buffer. The Z value output format, for this example, will be the 16 bit integer part since 16 bit Z depth is selected.

1 Work Out Area of Triangle
AreaD = 1/((x3-x2)*(y3-y1)-(x3-x1)*(y3-y2))

2 Work out ZΔX
ZdeltaX = ((Z3-Z2)*(y3-y1)-(Z3-Z1)*(y3-y2))*AreaD

3 Work out ZΔY
ZdeltaY = (Z3-Z1)/(y3-y1)

4 Set-up Z Buffer Address
ZBASE = Z address for V1

5 Set-up RGB 565 and Z Comparison Mode
Omode = 0
Zmode = 3
Ztest = 5
Zdepth = 1
MODE = (Zdepth <<23) + (Ztest<<13) + (Zmode<<11) + Omode

6 Set-up Z Interpolation Values
ZVAL = Z1
Z_DX = ZdeltaX
Z_DY = ZdeltaY

7 Start PRE
S_TOP = Ntop
S_BOT = Nbottom
wait for PRE to finish: test STATUS *register*

Gouraud Shading

In PRE, Gouraud shading is performed by specifying the color intensity at the start of the stripe, and the change in intensity as the stripe is rendered from pixel to pixel. The intensity of the pixel will therefore be interpolated linearly between the two end points of the stripe. The intensity of the pixel value is here defined to be a value between 0 and 1 by which the source pixel value is multiplied. A value of '1' specifies maximum illumination.

Each pixel however, has three different color intensities; R, G and B. If all three are shaded by the same intensity value, this has the effect of illuminating the scene with a white light source. In order to simulate colored illumination, different intensities must be used for each of the color components.

PRE therefore contains an incrementing engine for the intensity value of each color component; at each Pixel Engine step it is incremented by an increment value. For example the red component is modelled by a starting intensity RALF, and an intensity increment R_DX. Blue and green components are handled in a similar fashion.

This component alpha value is then used to multiply each of the three color components of the source pixel color.

Again, assume that the triangle is the same size and position as before, and we use the same gradient results and output screen address. For gouraud shading we have an 'RGB' color value associated with each vertex V1, V2, V3. These are: R1, R2, R3 for the color red, G1, G2, G3 for green, and B1, B2, B3 for blue.

Note that in this example texture mapping is disabled.

1 Set Up Output Mode to be RGB565
Omode = 0
EnableTex = 0
MODE = (EnableTex<<4) + Omode

2 Work Out Area of Triangle
AreaD = 1/((x3-x2)*(y3-y1)-(x3-x1)*(y3-y2))

3 Work Out RΔX, GΔX, BΔX

RdeltaX = ((R3-R2)*(y3-y1)-(R3-R1)*(y3-y2))*AreaD
GdeltaX = ((G3-G2)*(y3-y1)-(G3-G1)*(y3-y2))*AreaD
BdeltaX = ((B3-B2)*(y3-y1)-(B3-B1)*(y3-y2))*AreaD

4 Work Out RΔY, GΔY, BΔY

RdeltaY = (R3-R1)/(y3-y1)
GdeltaY = (G3-G1)/(y3-y1)
BdeltaY = (B3-B1)/(y3-y1)

5 Set-up Gouraud Shading Parameters

RALF = R1
R_DX = RdeltaX
R_DY = RdeltaY
GALF = G1
G_DX = GdeltaX
G_DY = GdeltaY
BALF = B1
B_DX = BdeltaX
B_DY = BdeltaY

6 Start PRE

S_TOP = Ntop
S_BOT = Nbottom
wait for PRE to finish: test STATUS *register*

Dithering

If the screen is being stored in RGB565 or RGB332, a dithering operation can be performed on the 8:8:8 value in order to increase the perceived resolution. A 'skewed ordered dither' based on a 4x4 matrix is used. The 4 x 4 matrix is addressed from bits 1, 2, 7 and 8 of the screen output address. The value from this matrix is added to each of the RGB components before output to the screen.

If double buffering is used then both buffer base addresses should lie on a 256 byte boundary, otherwise shimmering artefacts will occur when the buffers are swapped.

Dithering is enabled by setting the enable dither bit in the MODE register, and interpolating as before.

Enable Dithering

Omode = 1
EnableDither = 1
MODE = (EnableDither<<25) + Omode

Translucency

It is sometimes desirable to combine two pixel values. A common instance is the modeling of translucent materials, where the pixel value will be a combination of the pixel value behind the translucent pixel, and the pixel being rendered. Another use is the application of highlights to surfaces.

The required blending operation is achieved in PRE by fetching a second pixel value that is at the position in screen memory that we wish to render to, this is known as the destination pixel (*dest*). The destination pixel (*dest*) is blended with the currently rendered pixel (*source*) according to the equations as defined in the blending mode. The blending factor is a changing intensity value, which for PRE is a single incrementing intensity value is used for all three color components. The maths is carried out in such a way that no overflow occurs; if an overflow is about to occur, the result is set to the maximum pixel value.

The *dest* value is fetched from an address which is calculated using address counter IBASE; no separate DDA calculation is available for the input screen address.

The *dest* value may be replaced by a fixed color FIXC, to be shaded and blended with the *source* value.

AVAL is used to perform blending operations between the calculated, *source*, pixel and input, *dest*, pixel according to the blend mode set in the MODE register (see Table 173).

Table 173. Translucency Equations

Standard	final pixel = source*AVAL + dest*(1-AVAL)	for translucent objects	mode 0
Summed	final pixel = source*AVAL + dest	for luminous areas	mode 1
Dimmed	final pixel = dest*(1-AVAL)	for dark areas	mode 2

In this example the blend mode will be set to Standard.

Alpha blending is disabled by setting AVAL to full saturation ($FFFFFFF), setting A_DX and A_DY to 0 and setting the blending mode to Standard.

1 Work Out Area of Triangle

AreaD = 1/((x3-x2)*(y3-y1)-(x3-x1)*(y3-y2))

2 Work Out AΔX

AdeltaX = ((A3-A2)*(y3-y1)-(A3-A1)*(y3-y2))*AreaD

3 Work Out AΔY

AdeltaY = (A3-A1)/(y3-y1)

4 Set-up Source Screen Buffer Address

IBASE = Input source screen address for V1

5 Set-up RGB 565 and Enable Input Screen

Omode = 0
EnScreen = 1
Blend = 0
MODE = (Blend<<27) + (EnScreen<<5) + Omode

6 Set-up A Interpolation Values
AVAL = A1
A_DX = AdeltaX
A_DY = AdeltaY

7 Start PRE
S_TOP = Ntop
S_BOT = Nbottom
wait for PRE to finish: test STATUS *register*

Texture Mapping

Texture mapping requires that the value of a pixel in the screen buffer be derived from a reference value in a texture map. Each point on the polygon, in *world* co-ordinates, corresponds to a point in the texture map. If the polygon were viewed normal to the surface, and at the same magnification and rotation as the texture map, texture mapping could be performed by copying. However, the polygon will actually be at a different distance and orientation, and so the mapping between the two becomes a more complicated function.

Figure 55. Polygon Texture Mapping

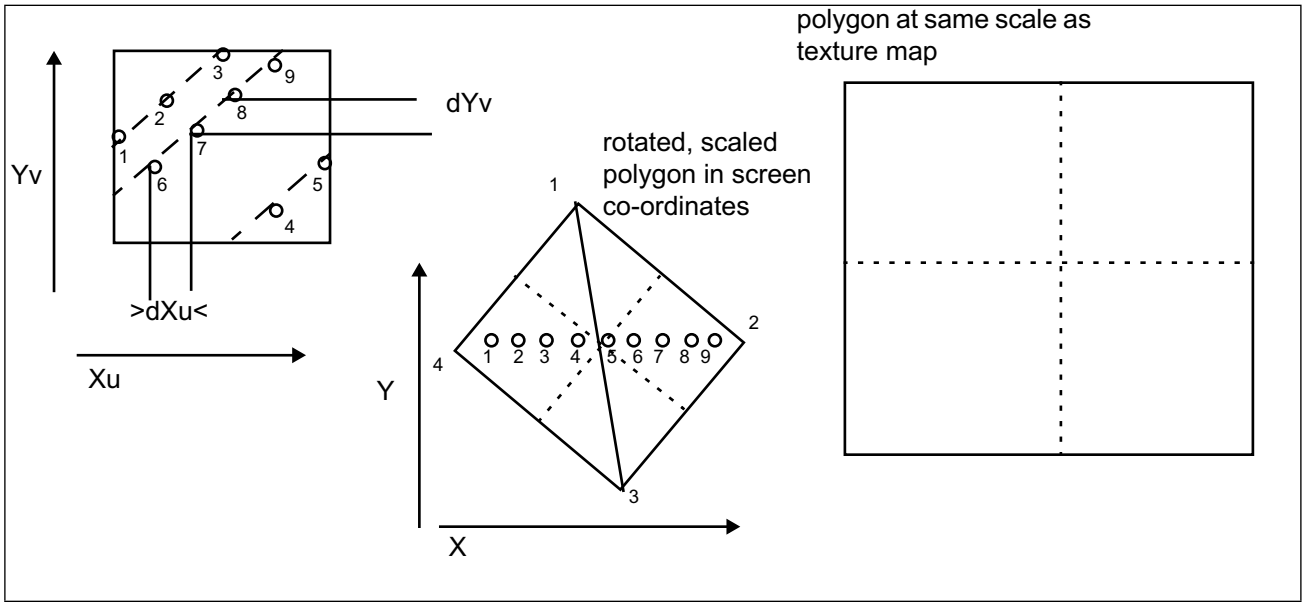


Figure 55 shows how a line through a rotated and scaled polygon maps onto the texture map. When a line is drawn along the surface of the polygon, at a stepping distance of one pixel in screen co-ordinates, the corresponding line through the texture map is at an arbitrary angle and stepping distance. PRE must calculate the address of each texture pixel ('texel') along the path and read it from the memory.

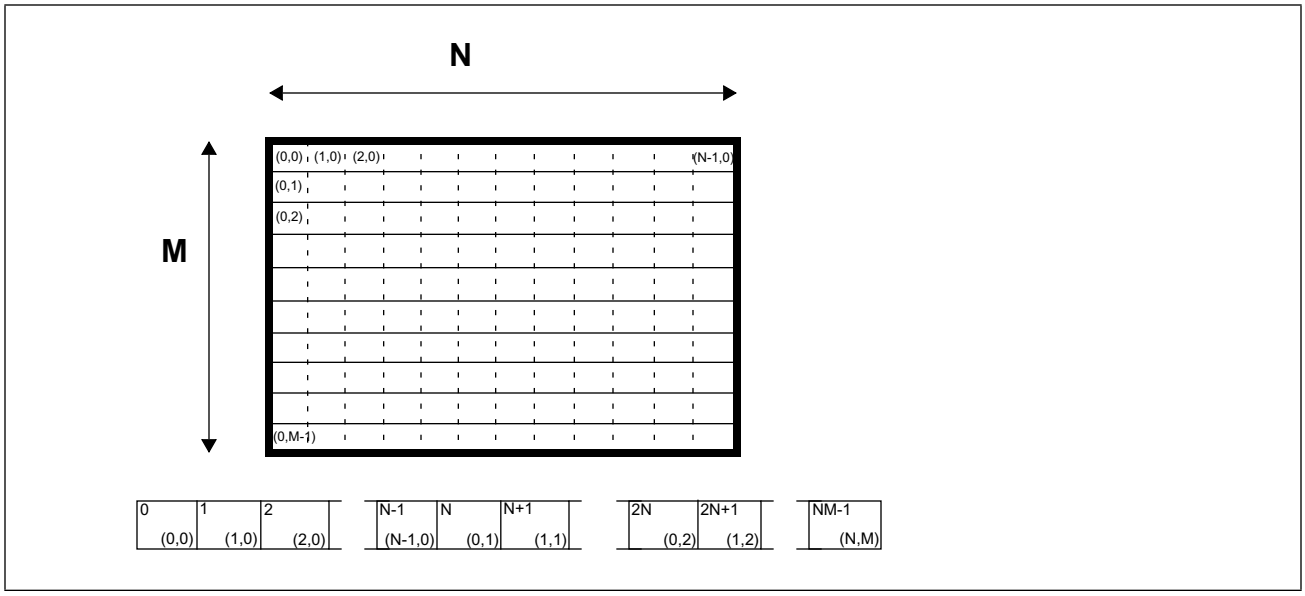
In the scaled and rotated case shown in Figure 55, the Xu and Yv co-ordinates of the texel in map are changed by an amount dXu and dYv at each step. The dXu and dYv deltas, are calculated from the texture co-ordinate values (Tu,Tv) associated with each vertex. In Figure 55, each vertex 1, 2, 3 and 4 have texture associated co-ordinates (Tu1,Tv1), (Tu2,Tv2), (Tu3,Tv3) and (Tu4,Tv4) respectively.

Control of Texture Mapping

At the start of triangle rendering, the embedded processor must initialize the values of Xu, dXu, Yv and dYv (and W and dW for perspective correction), for the texture. Each step, PRE calculates a new Xu and Yv value, which must be transformed into an address in memory. The Xu and Yv values are logically 'ANDed' with masks. This performs wraparound of the Xu and Yv values, and also determines the size of the texture maps, which must be a power of two in each dimension. The logical OR of these two values is then added to the base address of the texture map to obtain the memory address of the desired texel.

Consider a N x M texture map, as in Figure 56. Neglecting texture packing for the moment, this is stored in memory as a contiguous array of texels.

Figure 56. Texture Mapping



The mapping between the Xu and Yv perspective correct coordinates **(u, v)** generated at the output of the perspective correction block and the index of the individual texel in the array is governed by the VSHIFT and MASK values, according to the following formula:

```
Index = (u & X_Mask) + ((v & Y_Mask) << VShift)
Memory Address = Index + TBASE
X_Mask - Mask to set texture width
Y_Mask - Mask to set texture height
VShift - Shift value to move up the V value to convert it
into an offset
TBASE - Address of texture base in memory
```

This is then mapped through the texture unpack stage to produce an actual fetch address and unpacking index within the quadword. It also allows a texture to be tiled across a triangle.

1 Set-up Texture Size, Mode and Base Address from NxM texture map as above. We shall assume 16 RGB screen output and 16 bit texture.

Omode = 0
X_Mask = N-1
Y_Mask = M-1
VShift = log 2 N
Imode = 2
EnableTex = 1

MASKS = (X_Mask << 16) + Y_Mask
TBASE = Base address of Texture Map
MODE = (VShift<<19) + (EnableTex<<4) + (Imode<<2) + Omode

2 Work Out Area of Triangle

AreaD = 1/((x3-x2)*(y3-y1)-(x3-x1)*(y3-y2))

3 Work Out UΔX and VΔX

UdeltaX = ((Tu3-Tu2)*(y3-y1)-(Tu3-Tu1)*(y3-y2))*AreaD
VdeltaX = ((Tv3-Tv2)*(y3-y1)-(Tv3-Tv1)*(y3-y2))*AreaD

4 Work Out UΔY and VΔY

UdeltaY = (Tu3-Tu1)/(y3-y1)
VdeltaY = (Tv3-Tv1)/(y3-y1)

5 Set Up Texture Interpolation Registers

UVAL = Tu1
U_DX = UdeltaX
U_DY = UdeltaY
VVAL = Tv1
V_DX = VdeltaX
V_DY = VdeltaY

6 Start PRE

S_TOP = Ntop
S_BOT = Nbottom
wait for PRE to finish: test STATUS *register*

Texture Compression

If there are many texture maps, the storage for texture maps will become very large. There is therefore provision for the compression of texture maps. The value of a texel may be defined to be a coded value, a 4 or 8 bit value, which defines one of 16 possible colors or one of 256 possible colors respectively. This value is used as an index into a 256 position RAM, the color look-up table (CLUT), which holds the actual full resolution (RGB888) color values or RGB565 with 8 bit alpha map color values.

The Lookup RAM is initialized by the TSC, by writing values to it via the special CLUT update register.

1 Load Up CLUT

for Index *in* Palette_base *to* Palette_top
LKUP = (Index<<24) + palette[Index]
end for Index.

2 Set-up Texture Size, Mode and Base Address from NxM texture map as above. In this case 16 RGB screen output and 4 bit texture.

Omode = 0
X_Mask = N-1
Y_Mask = M-1
VShift = log 2 N
Imode = 0
EnableTex = 1
LookUpBase = 0
MASKS = (X_Mask << 16) + Y_Mask
TBASE = Base address of Texture Map
MODE = (VShift<<19) + (LookUpBase<<6) + (EnableTex<<4) + (Imode<<2) + Omode

3 Set Up Texture Interpolation Registers

UVAL = Tu1
U_DX = UdeltaX
U_DY = UdeltaY
VVAL = Tv1
V_DX = VdeltaX
V_DY = VdeltaY

4 Start PRE

S_TOP = Ntop
S_BOT = Nbottom
wait for PRE to finish: test STATUS *register*

Texture with Perspective Correction

For perspective rendering, the calculations are more complex. As the line is drawn across the polygon, the distance between the viewing plane and the corresponding point on the polygon in world co-ordinates changes. This means that the stepping distance between points in the texture map will change along the path through the texture map.

PRE applies a correction to the Xu and Yv values to approximate the effect of a perspective transform. For a given line scanned across the polygon, an extra co-ordinate, W, is defined; it is the fourth co-ordinate in the homogeneous co-ordinate system.

The W values are derived from the Z values of the vertices of the triangle. It should be noted that the Xu, and Yv initial values are also functions of Z.

The value of W at each point is used to find the perspective corrected **u**, **v** co-ordinates of the texture map pixel, which corresponds to the pixel on the scan line.

Perspective correct **u** and **v** are calculated from Xu and Yv as follows:

u = Xu*W

v = Yv*W

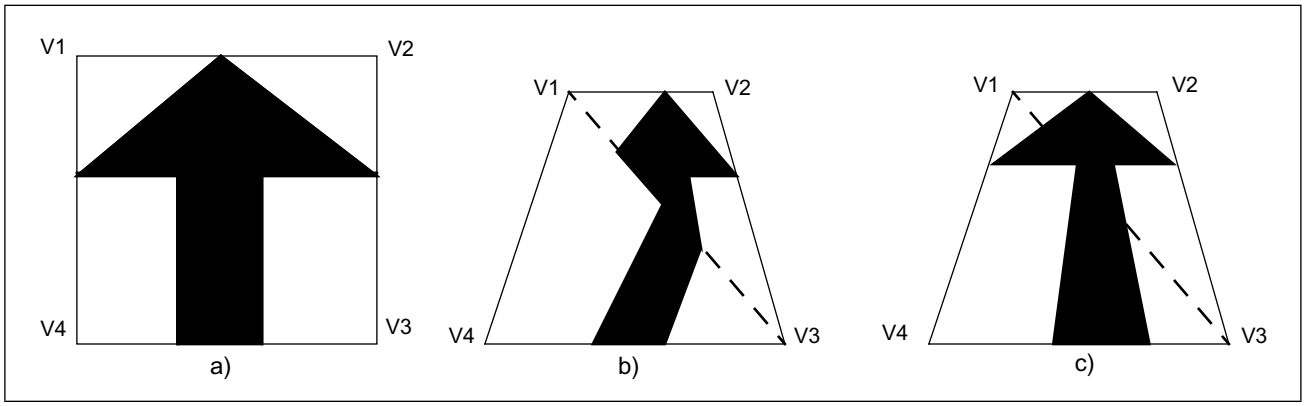
In order to attain the best approximation to perspective correction, PRE interpolates "1/W" values, Q. For PRE, W should also be scaled such that W is the value 1 at one triangle vertex and > 1 at the others. After scaling W, the all Q values will lie in the range $0 < Q < 1$. The perspective correction equations now become:

u = Xu/Q

v = Yv/Q

Such calculations require two divisions per pixel. Internally in PRE, however, they are accomplished by a normalization of the value of Q to lie between 0.5 and 1, followed by a lookup of the reciprocal value of Q in a ROM. The reciprocal is then used to multiply the Xu and Yv values, and hence forming the **u, v** values. Figure 57 shows the sort of problem that might be expected without perspective correction.

Figure 57. Image Without Perspective Correction



The texture square is drawn with 2 triangles having a shared edge shown by the dotted line. The diagrams represent:

a) The direct pixel to texel mapping using algorithms described as before.

b) The result from moving V1 and V2 together after a transformation.

c) The desired output.

The following set-ups will be for the triangle represented by vertices V1, V2 and V3.

1 Set-up Texture Size, Mode and Base Address from NxM texture map as above. We shall assume 16 RGB screen output and 16 bit texture.

Omode = 0
X_Mask = N-1
Y_Mask = M-1
VShift = log 2 N
Imode = 2
EnableTex = 1

MASKS = (X_Mask << 16) + Y_Mask
TBASE = Base address of Texture Map
MODE = (VShift<<19) + (EnableTex<<4) + (Imode<<2) + Omode

2 Work Out Area of Triangle

Area D = 1/((x3-x2)*(y3-y1)-(x3-x1)*(y3-y2))

3 Work Out Scaling Factor for W

H = smallest of W1, W2 or W3

4 Work Out Reciprocal W and New Tu and Tv Values

Q1 = H/W1, QU1 = Tu1*Q1, QV1 = Tv1*Q1
Q2 = H/W3, QU2 = Tu2*Q2, QV2 = Tv2*Q2
Q3 = H/W3, QU3 = Tu3*Q3, QV3 = Tv3*Q3

5 Work Out QΔX and QΔY

QdeltaX = ((Q3-Q2)*(y3-y1)-(Q3-Q1)*(y3-y2))*AreaD
QdeltaY = (Q3-Q1)/(y3-y1)

6 Work Out UΔX and VΔX

UdeltaX = ((QU3-QU2)*(y3-y1)-(QU3-QU1)*(y3-y2))*AreaD
VdeltaX = ((QV3-QV2)*(y3-y1)-(QV3-QV1)*(y3-y2))*AreaD

7 Work Out UΔY and VΔY

UdeltaY = (QU3-QU1)/(y3-y1)
VdeltaY = (QV3-QV1)/(y3-y1)

8 Set Up Texture Interpolation Registers

UVAL = QU1
U_DX = UdeltaX
U_DY = UdeltaY
VVAL = QV1
V_DX = VdeltaX
V_DY = VdeltaY
QVAL = Q1
Q_DX = QdeltaX
Q_DY = QdeltaY

9 Start PRE

S_TOP = Ntop
S_BOT = Nbottom
wait for PRE to finish: test STATUS *register*

Bilinear Interpolation

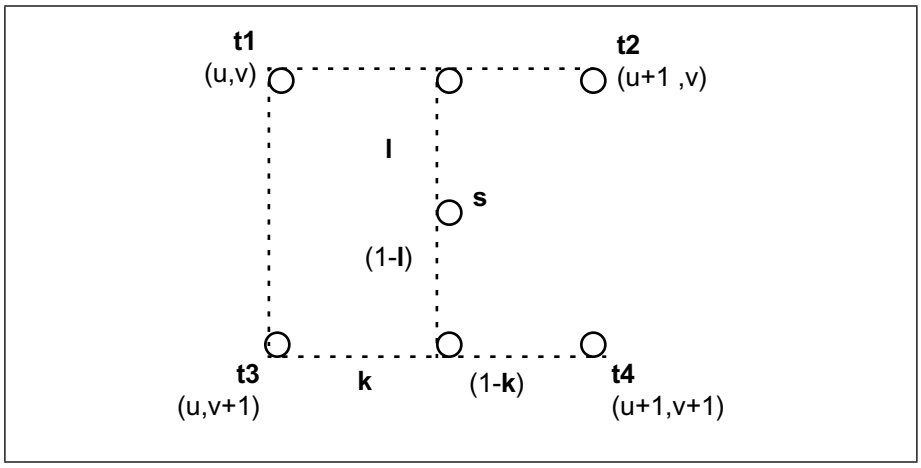
When displaying texture mapped polygons close to the viewpoint, one 'texel' in the texture map is mapped to many screen pixels. The polygon thus takes on a very 'blocky' appearance. Bilinear interpolation is used to smooth out such artefacts, by interpolating the values in the texture map so that it appears more detailed.

Bilinear interpolation is performed by interpolating between the color values defined at four adjacent points in the texture map.

Let the current point on the screen be defined as being located at the point **(u.k, v.l)** in the texture map, where **u** is the integer part of the texture map x co-ordinate, and **k** the fractional part, and **v** and **l** the corresponding integer and fractional values for the texture map y co-ordinate.

The non-interpolated texture map texel associated with the point will be the texel **t1** at **(u, v)**. (See Figure 58.)

Figure 58. Bilinear Interpolation



The bi-linear interpolated texel **s** at (**u.k, v.l**), derived from texels **t1, t2, t3** and **t4** (at (**u,v**), (**u**+1,**v**), (**u,v**+1) and (**u**+1,**v**+1) respectively) is defined as having a value:

$$\mathbf{s} = \{\ \mathbf{l}^*\{\ \mathbf{k}^*t1 + (1-\mathbf{k})^*t2\} + (1-\mathbf{l})\ ^*\{\ \mathbf{k}^*t2 + (1-\mathbf{k})^*t4\}\ \}$$

When bilinear interpolation is enabled in PRE, each time a new texel address (**u**, **v**) is calculated, the DDA requests all four texels required for bilinear interpolation from the memory controller. These are the texels **t1, t2, t3** and **t4**.

PRE performs three blend operations to calculate the effective texel color.

phase1 = k*(t1 - t2) + t2
phase2 = k*(t3 - t4) + t4
final color = l*(phase1 - phase2) + phase2

This complex operation is simply enabled by setting a bit in the MODE register.

Enable Bi-linear

Omode = 0
X_Mask = N-1
Y_Mask = M-1
VShift = log 2 N
Imode = 2
EnableTex = 1
EnableBi = 1

MASKS = (X_Mask << 16) + Y_Mask
TBASE = Base address of Texture Map
MODE = (VShift<<19) + (EnableBi<<16) + (EnableTex<<4) + (Imode<<2) + Omode

Texture Transparency

Texture maps can be used in such a way that they have 'holes' in them. When a transparent part of a texture map is detected no pixel is written out so that the previous value in the screen map is left unchanged. When transparency is enabled, the texel value is compared to a predefined value (in this case, black), and if it is equal to this color, the pixel is not written. Note that when bilinear interpolation is enabled small transparent areas may become opaque since the interpolated color will not be black.

Enable Transparency

Omode = 0
X_Mask = N-1
Y_Mask = M-1
VShift = log 2 N
Imode = 2
EnableTex = 1
EnableTrans = 1

MASKS = (X_Mask << 16) + Y_Mask
TBASE = Base address of Texture Map
MODE = (VShift<<19) + (EnableTrans<<10) + (EnableTex<<4) + (Imode<<2) + Omode

Texture Translucency

It is useful to define textures which also contain translucency (this known as an alpha map). Three texture modes can use alpha maps: 4, 8 and 32 bit textures.

For the 4 and 8 bit textures a special mode is used on the texture look up table to accommodate this type of map. The color value found in the look up table is ARGB8565, and is interpreted as containing alpha, R, G and B information. In this case, alpha is eight bits resolution, and the RGB information is stored in the remaining 16 bits as RGB565.

The texture color is converted from RGB565 to RGB888, and then shaded by the RALF, GALF, BALF working registers, as before.

The CLUT is loaded up the same way as before, but this time with ARGB8565 values.

For 32 bit textures, however, the color from the texture map is interpreted as ARGB8888 and the alpha information passed to the background shading channel as above.

The 8 bit alpha information obtained from the alpha map is passed to the background shading channel, and used to multiply the background color for blending with the texture color.

Note that it is the responsibility of the rendering algorithm to calculate the required values for the RALF, GALF, BALF and their increments. The operation will only be an approximation to the ideal case, because the calculated alpha (AVAL) is not used, and instead the alpha value is the fixed values from the alpha map.

Enable Alpha Map

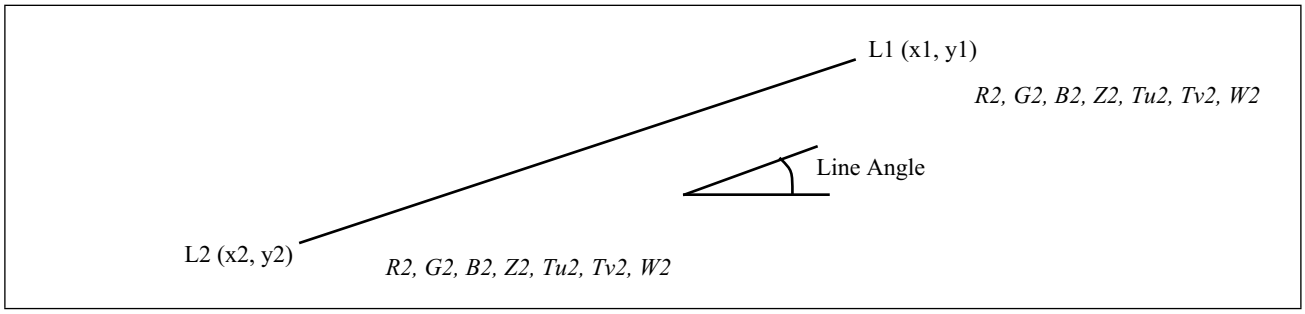
Omode = 0
X_Mask = N-1
Y_Mask = M-1
VShift = log 2 N
Imode = 1
EnableTex = 1
EnableAlphaMap = 1

MASKS = (X_Mask << 16) + Y_Mask
TBASE = Base address of Texture Map
MODE = (VShift<<19) + (EnableAlphaMap<<17) + (EnableTex<<4) + (Imode<<2) + Omode

Line Drawing

Because PRE is able to draw trapezoids we can very simply draw lines. Consider a line with end points L1 and L2 and the same parameters as a triangle, as in Figure 59.

Figure 59. Example Line Drawing



Work Out Gradient, dx/dy

If line is horizontal then gradient is infinity.
Else, if line is vertical then gradient is 0
Otherwise, the gradient of the line m1 = (x2-x1)/(y2-y1).

Parameter Values

For any parameter P (either: *R,G,B,Z,U,V,W)*, the delta along the line is Pdelta = (P2-P1)/(line_length), and PRE will have three registers for interpolation of that parameter say *PVAL, P_DY, P_DX*.

Vertical Line, Gradient 0
Pdelta = (P2-P1)/(y2-y1)
PVAL = P1
P_DY = Pdelta
P_DX = 0
XSTART = x1
XENDT = x1+1
XENDB = don't care
XS_DY = 0
XT_DY = 0
XB_DY = don't care
S_TOP = y2-y1
S_BOT = 0

Horizontal Line, Gradient Infinity

Pdelta = (P2-P1)/(x2-x1)
PVAL = P1
P_DY = 0
P_DX = Pdelta
XSTART = x1
XENDT = x2
XENDB = don't care
XS_DY = don't care
XT_DY = don't care
XB_DY = don't care
S_TOP = 1
S_BOT = 0

Line Angle ≤45°, |Gradient| ≤1

Pdelta = (P2-P1)/(y2-y1)
PVAL = P1
P_DY = Pdelta
P_DX = 0
XSTART = x1
XENDT = x1+1
XENDB = don't care
XS_DY = m1
XT_DY = m1
XB_DY = don't care
S_TOP = y2-y1
S_BOT = 0

Line Angle < 45°, |Gradient|>1

Pdelta = (P2-P1)/(x2-x1)
PVAL = P1
P_DY = 0
P_DX = Pdelta
XSTART = x1
XENDT = x1+ (*rounded down* m1)
XENDB = don't care
XS_DY = m1
XT_DY = m1
XB_DY = don't care
S_TOP = y2-y1
S_BOT = 0

Supplemental Information

Terms

Alpha Map This is the term used to describe textures that contain alpha (translucency) information.

Database Traversal This is the process of accessing the world database of 3D models and providing a list of triangles to be rendered.

Destination Pixel This is the value that is currently in the screen buffer

DDA The hardware which performs the calculation of the addresses for texels and pixels is known as a 'DDA' which stands for Digital Differential Analyzer

Line One of the 3D primitives that PRE is able to draw.

Polygon This is a facet of a 3D model. For example a cube could be defined as 6 squares and each square defined as 2 triangles. PRE can be used as a triangle rendering engine but is not restricted only to triangles hence use of the term polygon.

Quad A four sided shape. In PRE this would probably be decomposed to 2 triangles

Quad-word This is a 64 bit word, sometimes referred to as Oct-Byte. It is 4 x 16 bit words or 8 x 8 bit bytes wide.

PRE Pixel Rendering Engine

Pre-decrement Where a working register is decremented before it is used.

Post-increment Where a working register is incremented after it is used.

Rasterization Or "triangle rasterization" is often referred to as "Scan Conversion". This is the process of transforming a series of vertices into one dimensional strips of Pixels.

Reverse Rendering Typically, this is the process of drawing a span from right to left.

Scan Conversion See Rasterization

Scan Line This is the horizontal line that is sent to the display. The scan line length is the display width.

Screen Input DDA Part of the DDA which generates the addresses for screen data fetch

Source Pixel This is the pixel that is calculated after texel fetch and color calculations.

Span A polygon is made of horizontal stripes that are one pixel high. These stripes are known as spans.

Trapezoid A regular four sided shape with flat top and bottom.

Tri-Strip This is a list of triangles that share common material and are joined by a series of vertices to form a triangle strip.

Texel This is the term for the texture pixel obtained from a texture map from memory.

Vertex This a point that makes up a corner of a triangle.

Z Source DMA Part of the DDA which generates the addresses for Z buffer data fetch

Parameter Nomenclature

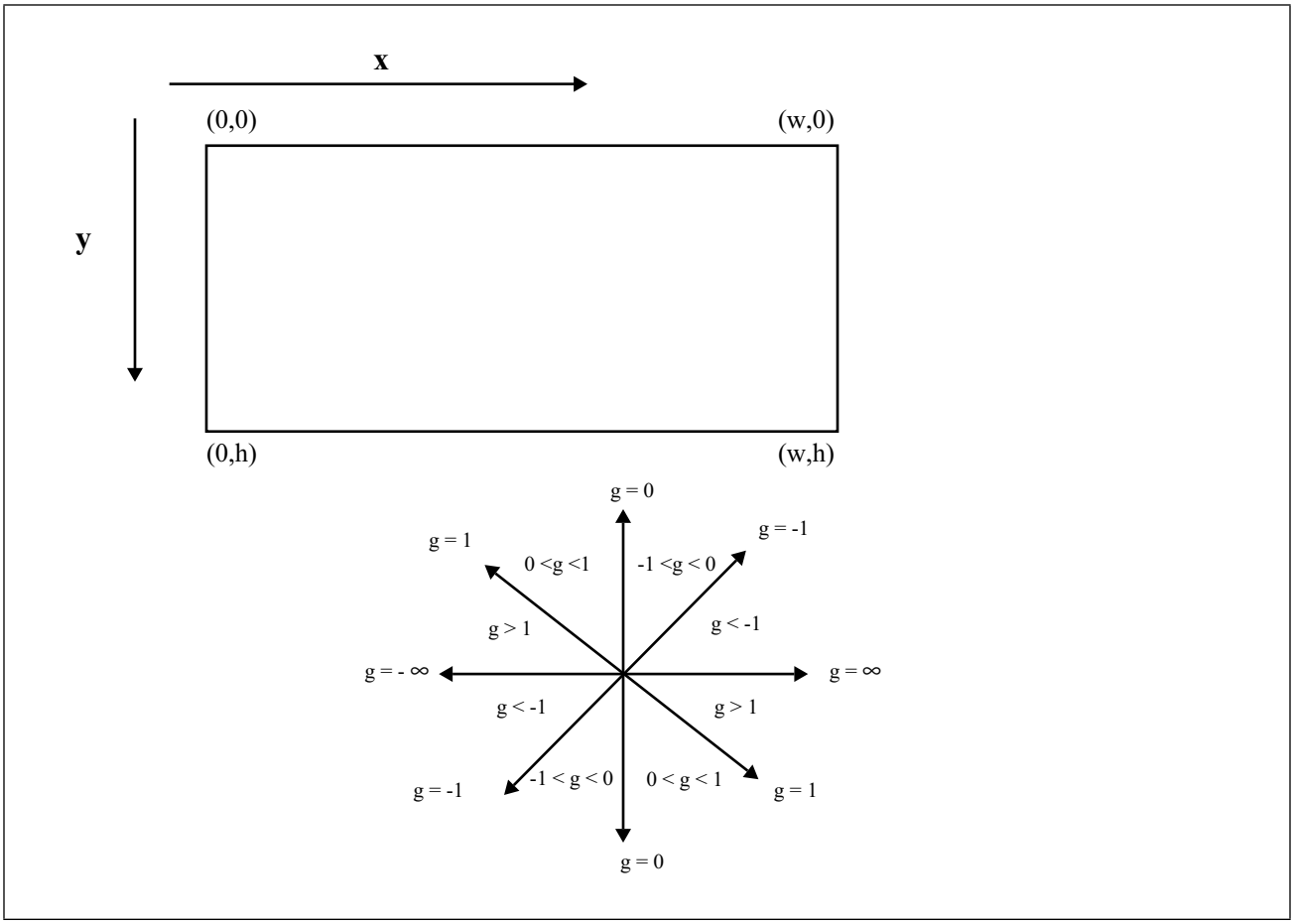
A	A1 A2 A3	Alpha component associated with each vertex
B	B1 B2 B3	Blue component associated with each vertex
G	G1 G2 G3	Green component associated with each vertex
h		screen height
H		homogenous W co-ordinate scaling factor so 0 < Q < 1.
k		fractional part of x co-ordinate of a texture map
l		fractional part of y co-ordinate of a texture map
L	L1 L2	Each end point of a line
m	m1 m2 m3	gradients for main edge, top edge and bottom edge respectively
M		Height of a texture map
N		Width of a texture map
P	P1 P2	Arbitrary parameter ‘P’ associated with each end point of a line.
Q	Q1 Q2 Q3	1/W associated with each vertex
Qu	Qu1 Qu2 Qu3	Tu/W associated with each vertex
Qv	Qv1 Qv2 Qv3	Tv/W associated with each vertex
R	R1 R2 R3	Red component associated with each vertex
s		Bilinear interpolated screen texel at s(u.k, v.l)
t		texel co-ordinate e.g. t(u,v)
t	t1 t2 t3 t4	texels required for Bi-linear math stages
Tu	Tu1 Tu2 Tu3	Texture x co-ordinate associated with each vertex
Tv	Tv1 Tv2 Tv3	Texture y co-ordinate associated with each vertex
u		**Post** perspective x co-ordinate of a texture map
V	V1 V2 V3	Each vertex of a triangle
v		**Post** perspective y co-ordinate V in a texture map
W	W1 W2 W3	W homogenous co-ordinate associated with each vertex

w		screen width
x		screen x co-ordinate
Xu		**Pre** perspective x co-ordinate of a texture map
y		screen y co-ordinate
Yv		**Pre** perspective y co-ordinate of a texture map
Z	Z1 Z2 Z3	Z co-ordinate associated with each vertex

Gradient Nomenclature in PRE

Consider a screen **w** wide and **h** high, as in Figure 60. Any line drawn on the screen in any direction will have a gradient **g** from the calculation **dx/dy**.

Figure 60. Example Gradient Lines



PCI CONFIGURATION SPACE REGISTER DEFINITIONS

PCI Configuration Space

The PCI configuration space is an address space that is accessible only through PCI config cycles. This address space is defined to contain configuration registers for the purpose of executing PCI plug-and-play algorithms. This section defines the organization of the registers within the 256 byte space of each PCI function. The Bt2166 supports the 64 byte predefined header as well as an additional 5 bytes in the user-defined portion of the configuration space. Figure 61 shows the configuration space header.

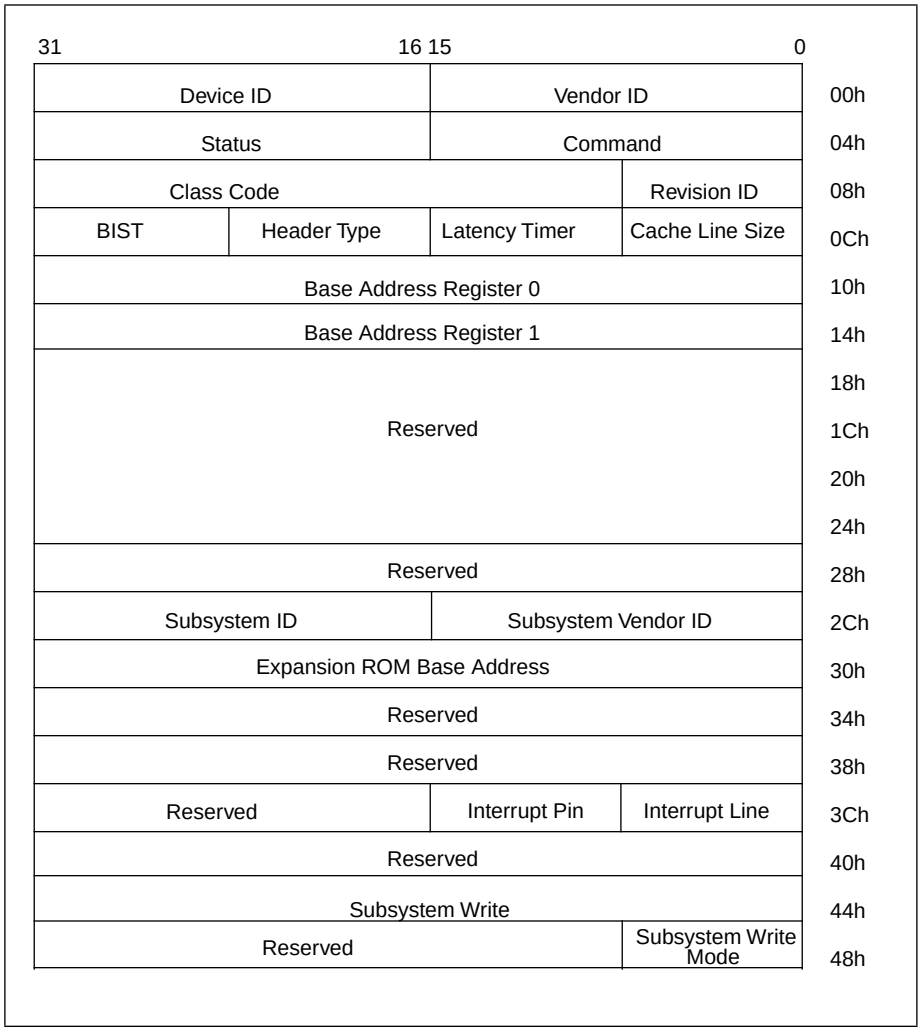
The Bt2166 is a multifunction device. Function 0 responds as a VGA compatible display controller. Function 1 responds as a multimedia video device. Each funtion has its own address space. The register definitions below apply to both functions unless otherwise specified.

The configuration space registers are stored in dwords and defined by byte addresses. Therefore a register one byte in length can have a bit definition other than 7:0 (for example 31:24), depending on its location in the configuration space. For a discussion on configuration cycle addressing, refer to Section 3.7.4 of the *PCI Local Bus Specification, Revision 2.1*.

All writable bits are reset to 0 by the system reset, unless otherwise specified. After reset, the Bt2166 is disabled and will only respond to CFGWR and CFGRD cycles. All reserved and unimplemented registers are read as zeroes.

For details on the PCI bus, refer to the *PCI Local Bus Specification, Revision 2.1*.

Figure 61. PCI Configuration Space Header



PCI Vendor ID Register

name: PCI_VENDOR_ID
address: 01h:00h, read only
size: 16 bits
function: This register contains the unique vendor ID assigned to Brooktree Corporation. This field will always return the value of 109Eh. As shown in Figure 61, the PCI vendor ID resides in bits 15:0 of 00h; and the PCI device ID resides in bits 31:16.

PCI Device ID Registers

name: PCI_DEVICE_ID0, PCI_DEVICE_ID1
address: 03h:02h, read only
size: 16 bits
function: This register contains the unique device IDs assigned by Brooktree Corporation. The function 0 register will return the Video/Graphics Controller model number in hexadecimal. The function 1 register will return an ID value which has the same 15 most signifcant bits as the function 0 value. The least significant bit is configurable.

As shown in Figure 61, the PCI vendor ID resides in bits 15:0 of 00h; and the PCI device ID resides in bits 31:16. Table 174 defines Function 0 bits; Table 175 defines Function 1 bits.

Table 174. PCI Device ID Register Function 0 (PCI_DEVICE_ID0)

Bit	Field	Detail
31:16	PCI Device ID	2166h

Table 175. PCI Device ID Register Function 1 (PCI_DEVICE_ID1)

Bit	Field	Detail
31:17	PCI Device ID[15:1]	15 most significant bits of ‘2166’h
16	PCI Device ID[0]	Value from GRP_CFG3[5] (FUNC1_DEVID_0), see Table 27 on page 53). Value is writable through VGA extension register space. Reset to 1. (i.e., Default Device ID is 2167.)

PCI Command Registers

name: PCI_COMMAND0, PCI_COMMAND1
address: 05h:04h, read/write as shown below
size: 16 bits
function: This register controls the ability to respond to PCI cycles. The bit contents for function 0 (PCI_COMMAND0) are shown in Table 176, and function 1 (PCI_COMMAND1) in Table 177.

Table 176. PCI Command Register 0 (PCI_COMMAND0)

Bit	Field	Detail
15:10		Read only. Contains all 0’s.
9	FAST BACK-TO-BACK	1 = Enable fast back-to-back 0 = Disable fast back-to-back
8:6		Read only. Contains all 0’s.
5	PCI_SNOOP_DAC	1 = Palette snooping is enabled, i.e.device does not respond to palette writes, but does snoop data. 0 = Handles palette accesses normally.

Table 176. PCI Command Register 0 (PCI_COMMAND0)

Bit	Field	Detail
4:3		Read only. Contains all 0's
2	PCI_MASTER_EN	PCI master enable for function 0 1 = Enable bus master cycles 0 = Disable bus master cycles
1	PCI_MEM_EN	PCI memory enable for function 0 1 = Enable memory cycles 0 = Disable memory and ROM cycles
0	PCI_IO_EN	PCI I/O enable for function 0 1 = Enable IO cycles in the VGA address space or alternate address space 0 = Disable IO cycles in the VGA address space or alternate address space

Table 177. PCI Command Register 1 (PCI_COMMAND1)

Bit	Field	Detail
15:10		Read only. Contains all 0's.
9	FAST BACK-TO-BACK	1 = Enable fast back-to-back 0 = Disable fast back-to-back
8:3		Read only. Contains all 0's.
2	PCI_MASTER_EN	PCI master enable for function 1 1 = Enable bus master cycles 0 = Disable bus master cycles
1	PCI_MEM_EN	PCI memory enable for function 1 1 = Enable memory cycles 0 = Disable memory and ROM cycles
0	PCI_IO_EN	PCI I/O enable for function 1 1 = Enable IO cycles in the VGA address space or alternate address space 0 = Disable IO cycles in the VGA address space or alternate address space

PCI Status Registers

name: PCI_STATUS
address: 07h:06h, read only
size: 16 bits
function: This register records status information for PCI bus related events. Except for bits 29 and 28, function 0 and function 1 share the same PCI status bits. Refer to Table 178.

Table 178. PCI Status Register 0 and 1 (PCI_STATUS0, PCI_STATUS1)

Bit	Field	Detail
31:30		Read-only.
29	PCI_MASTER_ ABORT	When the Bt2166, functioning as a PCI bus master, terminates a transaction with a master-abort, this bit is set to one. Reset to zero when a one is written to this bit position.
28	PCI_TARGET_ ABORT	When the Bt2166, functioning as a PCI bus master, has its transaction terminated by a target-abort, this bit is set to one. Reset to zero when a one is written to this bit position.
27		Read-only. Returns 0.
26:25	PCI_DEVSEL_ TIMING	Read-only, set to 01. Medium, DEVSEL always in third clock or faster.
24		Read-only. Returns 0.
23	PCI_BK_TO_BK	Read-only, set to 1, supports back-to-back cycles.
22		Read-only. Returns 0.
21	66MHZ_CAPABLE	Read-only value from GRP_CFG7[7], 66MHZ_ENABLE. See Table 23 on page 52. Reset to value on VDATA[7] pin (see Figure 11 on page 55). When Bt2166 is attached to a PCI bus, set to 0; when attached to an AGP bus, set to 1. 1 = Bt2166 is capable of operating at 66MHz 0 = Bt2166 operates at 33MHz only
20:16		Read-only. Returns 0.

PCI Revision ID Register

name: PCI_REVISION_ID
address: 08h, read only
size: 8 bits
function: This register contains the revision ID for the Bt2166.

PCI Class Code Registers

name: PCI_CLASS0_ID, PCI_CLASS1_ID
address: 0B:09h, read only
size: 32 bits
function: This register identifies the function of the Bt2166. The bit contents for function 0 (PCI_CLASS0_ID) are shown in Table 179, and for function 1 (PCI_CLASS1_ID) in Table 180.

Table 179. PCI Class Code Register—Function 0

Bit	Detail
31:24	In normal operation, value is 03h (display controller). When alternate decode is enabled with GRP_CFG3[6] (see Table 27 on page 53), value is FFh (undefined class).
23:8	VGA - 0000h

Table 180. PCI Class Code Register—Function 1

Bit	Detail
31:24	Multimedia device - 04h
23:8	Video device - 0000h

PCI Cache Line Size

name: PCI_CACHE_SZ
address: 0Ch, read-only
size: 8 bits
function: This field is hardwired to 00h, indicating no cache.

PCI Latency Timer

name: PCI_LATENCY_TIMER0, PCI_LATENCY_TIMER1
address: 0Dh, read/write
size: 8 bits
function: This register specifies, in units of PCI bus clocks, the value of the Latency Timer for this PCI bus master. The most significant 5 bits of this register have flops, while the least significant three are tied to zero. This produces a timer granularity of eight clocks. A value of 00h indicates that a burst will terminate as soon as PCI_GNT is deasserted.

Table 181. PCI Latency Timer

Bit	Detail
15:11	Latency Timer [7:3]
10:8	000b

PCI Header Type

name: PCI_HEADER_TYPE
address: 0Eh, read-only
size: 8 bits

This byte identifies the layout of the second part of the predefined header (beginning at byte 10h in Configuration Space) and also whether or not the device contains multiple functions. A value of 80h indicates a multi-function device.

The FCN1_DISABLE bit (GRP_CFG7[6], see Table 23 on page 52) determines how the chip will respond. If set, the Bt2166 will respond as a single function device. Access to all function numbers will be mapped into function 0. If FCN1_DISABLE is zero, the Bt2166 will respond as a multi-function device. This means that function numbers will be decoded, such that function 0 accesses will be mapped into function 0 registers, function 1 accesses get mapped into function 1 registers, and accesses to other functions are ignored. Reset is the value on the VDATA6 pin (see Figure 11 on page 55).

PCI BIST

name: PCI_BIST
address: 0Fh, read-only
size: 8 bits
function: This optional register is used for control and status of BIST (Built-in Self Test). The Bt2166 returns 00h, indicating that it does not support BIST.

PCI Base Address Register Zero

name: PCI_BASE0
address: 10h, read/write
size: 32 bits
function: This register contains the memory base addresses for functions 0 and 1. See Table 182 for PCI_BASE0 bit content. For more information on the address space, refer to "CPU Address Space Apertures" starting on page 15.

Table 182. PCI Base Zero Address Register (PCI_BASE0)

Bit	Detail
31:25	If all zeroes, this aperture is disabled. If non-zero, specifies the address of the 32MB aperture space. For function 0, this field is also readable through the GRP_GUI_BASE[3] register. Refer to "GUI Base Address Registers" on page 46. The base 0 registers point to the same internal 32 MB space for both functions 1 and 0.
24:4	Read only, all 0's
3	Read only. 0 indicates not prefetchable.
2:1	Read only. 00 indicates locatable anywhere in 32-bit address space.
0	Read only. 0 = memory space

PCI Base Address Register 1

name: PCI_BASE1
address: 14h, read/write
size: 32 bits
function: This register specifies the I/O base address for functions 0 and 1. See Table 182 for PCI_BASE1 bit content. For more information on the address space, refer to "CPU Address Space Apertures" starting on page 15.

Table 183. PCI Base 1 Address Register (PCI_BASE1)

Bit	Detail
31:4	If all zeroes, this address space is disabled. If non-zero, specifies the starting address of a 16-byte I/O space. The base 1 registers point to the same physical 16 byte space for both functions 1 and 0.
3:1	Read only, all 0's
0	Read only. 1 = I/O space

PCI_BASE1[15:0] is shadowed by the I/O-mapped register GRP_IOAP ("I/O Aperture Shadow Register" on page 48). These 16 bits contain an I/O base address that points to a 16 byte I/O block that is implemented as follows.

first 4 bytes: contains address field that points to Bt2166 memory

second 4 bytes: contains a data field

Once the address field is written, the I/O read/writes from the data field will read/write the Bt2166 memory at that address. Reads/writes from address and data fields must be in dwords.

next 6 bytes: are unused

last 2 bytes: shadow the 3CE/3CFh VGA registers

An I/O read/write of the next-to-last byte of the I/O block is equivalent to an I/O read of 3CEh.

PCI Subsystem Vendor ID

name: PCI_SUBSYS_VENDOR_ID
address: 2Dh:2Ch, read-only
size: 16 bits
function: This register is used to identify the vendor of the add-in board or subsystem where the PCI device resides. The value read from this space can be fetched from ROM, or can come from an internal register which can be written through the PCI_SUBSYSID_WR address (see "PCI Subsystem ID Write Register" on page 324). Whether the value comes from a ROM fetch or an internal register is controlled by the PCI_SUBSYSID_MODE register. Since the default is to come from an attached ROM, the reset state of this logical register is undefined.

PCI Subsystem ID

name: PCI_SUBSYS_ID
address: 2Fh:2Eh, read-only
size: 16 bits
function: This register is used to identify the add-in board or subsystem where the PCI device resides. The value read from this space can be fetched from ROM, or can come from an internal register which can be written through the PCI_SUBSYSID_WR address (see "PCI Subsystem ID Write Register" on page 324. Whether the value comes from a ROM fetch or an internal register is controlled by the PCI_SUBSYSID_MODE register. Since the default is to come from an attached ROM, the reset state of this logical register is undefined.

PCI ROM Base Address Register

name: PCI_ROM_BASE
address: 33h:30h, read/write
size: 32 bits
function: These registers specify a 32 KB space for the Bt2166 to respond to ROM cycle requests. These registers are valid for function 0 only. For function 1, these registers are not implemented and read back as 32'h00000000. The bit contents for PCI_ROM_BASE0, function 0, is shown in Table 184.

Table 184. PCI ROM Base Register—Function 0

Bit	Detail
31:15	Specifies a 32 KB space for ROM cycles
14:01	Reserved
0	PCI_ROM_EN, Read/write 1 = ROM cycles enabled 0 = ROM cycles disabled

PCI Interrupt Line Register

name: PCI_INT_LINE0, PCI_INT_LINE1
address: 3Ch, read/write
size: 8 bits
function: This register specifies which system interrupt controller input is connected to the Bt2166's interrupt pin. This register is valid for function 1 only.

PCI Interrupt Pin Register

name: PCI_INT_PIN0, PCI_INT_PIN1
address: 3Dh, read only
size: 8 bits
function: This register returns the value 8'h01, which indicates a response on $\overline{\text{PCI\_INTA}}$. This register is valid for function 1 only; function 0 will return a value of 0.

PCI Subsystem ID Write Register

name: PCI_SUBSYSID_WR
address: 47h:44h, write only
size: 32 bits
function: This register provides a value for reads to both the PCI_SUBSYS_ID and PCI_SUBSYS_VENDOR_ID addresses. The values written to this address will only be used for PCI_SUBYS_ID and PCI_SUBSYS_VENDOR_ID reads if PCI_SUBSYS_MODE is set to one. The reset value for this register is 32'h2166109E, which is equivalent to the PCI Device IDs and PCI Vendor IDs registers.

PCI Subsystem ID Mode Register

name: PCI_SUBSYSID_MODE
address: 48h, read/write ony
size: 8 bits
function: When bit one is set, reads to the PCI_SUBSYS_ID and PCI_SUBSYS_VENDOR_ID addresses comes from the internal register written through the PCI_SUBSYSID_WR address. Otherwise, data for these reads is fetched from ROM. Function 0 reads go to ROM address offset 20h, and function 1 reads go to ROM address offset 24h. Bits 7:1 are not writable and are always zero. The reset value is 0.

CPU HOST BUS INTERFACE

PCI/AGP Local Bus Interface

The PCI local bus is an architectural, timing, electrical, and physical interface that allows the Bt2166 to interface to the local bus of a host CPU.

The supported PCI bus cycles are as follows:

- I/O Read
- I/O Write
- Memory Read (slave and master)
- Memory Write (slave and master)
- Configuration Read
- Configuration Write
- Memory Read Multiple (slave and master)
- Memory Read Line
- Memory Write and Invalidate

Memory Write and Memory Write and Invalidate are treated in the same manner. Memory Read and Memory Read Multiple are treated in the same manner.

The unsupported PCI bus cycles are as follows:

- Interrupt Acknowledge
- Special Cycle
- Dual Address Cycle

All I/O, memory, and configuration writes are three clock cycles unless held off by a full host FIFO. Subsequent writes in the same transaction are zero wait states as long as the internal FIFO is not full.

The PCI Bus interface pins are defined in the following paragraphs. The overbar above a signal indicates active-LOW.

The term "sustained tri-state™" is used in this section. It is defined as an active-LOW, tri-state signal owned and driven by a single agent at a time. The agent that drives this pin low must drive it high for at least one clock before allowing it to float. A new agent cannot start driving a sustained tri-state signal sooner than one clock after the previous owner tri-states it. To sustain the inactive status until a new agent drives it, a pullup must be provided by the central resource.

PCI/AGP Functionality

The Bt2166 is a 66MHz device designed to attach to either a PCI bus running up to 33MHz or an AGP bus running at 66MHz (using AGP timings). In all cases, the Bt2166 performs as a PCI device; however, its ability to operate at 66MHz enables it to function in an AGP bus environment. The Bt2166 implements only the AGP functionality that is part of the *PCI Local Bus Specification, Revision 2.1*. It does not implement any of the AGP New Capabilities Registers.

PCI Address and Data Bus

PCI_AD[31:0]—These tri-state, bi-directional, IO pins handle both address and data information. A bus transaction consists of an address phase followed by one or more data phases for either read or write operations.

The address phase is the clock cycle in which $\overline{\text{PCI\_FRAME}}$ is first asserted. During the address phase, PCI_AD[31:0] contains a byte address for IO operations or a dword address for configuration and memory operations. During data phases, PCI_AD[7:0] contains the least significant byte and PCI_AD[31:24] contains the most significant byte.

Read data is stable and valid when $\overline{\text{PCI\_TRDY}}$ is asserted; write data is stable and valid when $\overline{\text{PCI\_IRDY}}$ is asserted. Data is transferred during the clocks when both $\overline{\text{PCI\_TRDY}}$ and $\overline{\text{PCI\_IRDY}}$ are asserted.

PCI Bus Command and Byte Enables

$\overline{\text{PCI\_C\_BE}}$[3:0]—These tri-state, bidirectional, IO pins handle both bus command and byte enable information. During the address phase of a transaction, $\overline{\text{PCI\_C\_BE}}$[3:0] contain the bus command. During the data phase, $\overline{\text{PCI\_C\_BE}}$[3:0] are used as byte enables. The byte enables are valid for the entire data phase and determine which byte lanes carry meaningful data. $\overline{\text{PCI\_C\_BE}}$[3] refers to the most significant byte and $\overline{\text{PCI\_C\_BE}}$[0] refers to the least significant byte.

PCI Parity

PCI_PAR—This tri-state, bidirectional, IO pin provides even parity across PCI_AD[31:0] and $\overline{\text{PCI\_C\_BE}}$[3:0]. This means that the number of 1's on PCI_PAR, PCI_AD[31:0], and $\overline{\text{PCI\_C\_BE}}$[3:0] equals an even number.

PCI_PAR is stable and valid one clock after the address phase. For data phases, PCI_PAR is stable and valid one clock after either $\overline{\text{PCI\_TRDY}}$ is asserted on a read or $\overline{\text{PCI\_IRDY}}$ is asserted on a write. Once valid, PCI_PAR remains valid until one clock after the completion of the current data phase. PCI_PAR and PCI_AD[31:0] have the same timing, but PCI_PAR is delayed by one clock. For any transaction, the target drives PCI_PAR for read data phases; the master drives PCI_PAR for address and write data phases.

PCI Clock

PCI_CLK—This input provides timing for all PCI transactions. All PCI signals except $\overline{\text{PCI\_RST}}$ and the optional interrupt are sampled on the rising edge of PCI_CLK, and all other timing parameters are defined with respect to this edge. The Bt2166 supports a PCI clock of up to 66MHz.

PCI Reset

$\overline{\text{PCI\_RST}}$—This input resets all functions on the Bt2166 before execution. $\overline{\text{PCI\_RST}}$ may be asynchronous to PCI_CLK when asserted or deasserted.

PCI Cycle Frame

$\overline{\text{PCI\_FRAME}}$—This sustained tri-state signal is driven by the current master to indicate the beginning and duration of an access. $\overline{\text{PCI\_FRAME}}$ is asserted to signal the beginning of a bus transaction. Data transfer continues throughout assertion. At deassertion, the transaction is in the final data phase.

PCI Initiator Ready

$\overline{\text{PCI\_IRDY}}$—This sustained tri-state signal indicates the bus master's readiness to complete the current data phase. $\overline{\text{PCI\_IRDY}}$ is used in conjunction with $\overline{\text{PCI\_TRDY}}$. When both $\overline{\text{PCI\_IRDY}}$ and $\overline{\text{PCI\_TRDY}}$ are asserted, a data phase is completed on that clock. During a read, $\overline{\text{PCI\_IRDY}}$ indicates when the initiator is ready to accept data. During a write, $\overline{\text{PCI\_IRDY}}$ indicates when the initiator has placed valid data on PCI_AD[31:0]. Wait cycles are inserted until both $\overline{\text{PCI\_IRDY}}$ and $\overline{\text{PCI\_TRDY}}$ are asserted together.

PCI Target Ready

$\overline{\text{PCI\_TRDY}}$—This sustained tri-state signal indicates the target's readiness to complete the current data phase.

$\overline{\text{PCI\_IRDY}}$ is used in conjunction with $\overline{\text{PCI\_TRDY}}$. When both $\overline{\text{PCI\_IRDY}}$ and $\overline{\text{PCI\_TRDY}}$ are asserted, a data phase is completed on that clock. During a read, $\overline{\text{PCI\_TRDY}}$ indicates when the target is presenting data. During a write, $\overline{\text{PCI\_TRDY}}$ indicates when the target is ready to accept the data. Wait cycles are inserted until both $\overline{\text{PCI\_IRDY}}$ and $\overline{\text{PCI\_TRDY}}$ are asserted together.

PCI Stop

$\overline{\text{PCI\_STOP}}$—This sustained tri-state signal indicates the target is requesting the master to stop the current transaction.

PCI Initialization Device Select

PCI_IDSEL—This input is used to select the target during configuration read and write transactions.When connecting to an AGP bus, this pin should be connected to PCI_AD[16].

PCI Device Select

$\overline{\text{PCI\_DEVSEL}}$—This sustained tri-state signal indicates device selection. The Bt2166 always asserts $\overline{\text{PCI\_DEVSEL}}$ in the third clock cycle.

When actively driven, $\overline{\text{PCI\_DEVSEL}}$ indicates the driving device has decoded its address as the target of the current access.

PCI Interrupt A

$\overline{\text{PCI\_INTA}}$—This signal is an Open Drain interrupt output, which signals an interrupt to the host processor.

PCI Master Request

$\overline{\text{PCI\_REQ}}$—This signal tells the PCI bus arbiter that the Bt2166 wants to use the bus. This point-to-point signal is driven by a PCI compatible driver.

PCI Master Grant

$\overline{\text{PCI\_GNT}}$—This signal tells the Bt2166 that access to the bus has been granted. This point-to-point signal is received by a standard PCI input cell.

PCI/AGP Bus Timing

This section presents timing diagrams for key PCI/AGP waveforms. Table 185 defines the minimum and maximum values for the specified parameters.

Table 185. PCI/AGP Bus Timing

Symbol	Parameter	33MHz PCI		66MHz AGP		Units
		Min	Max	Min	Max	
Input (see Figure 62)						
t_1	Address/control hold from PCI_CLK	0	—	0.5	—	ns
t_2	Address/control setup to PCI_CLK	7	—	5	—	ns
t_3	PCI_CLK rise slope	1	4	1.5	4	V/ns
t_4	PCI_CLK fall slope	1	4	1.5	4	V/ns
t_5	PCI_CLK high period	11	—	6	—	ns
t_6	PCI_CLK low period	11	—	6	—	ns
t_7	PCI_CLK period	30	—	15	—	ns
Output (see Figure 63)						
t_8	Read data/control delay from PCI_CLK (= t_{VAL})	—	11	1.5	6	ns
t_9	Read data hold from PCI_CLK	0	—	0.5	—	ns

The following timing diagrams depict the associated PCI time intervals.

Figure 62. PCI/AGP Input Timing Diagram

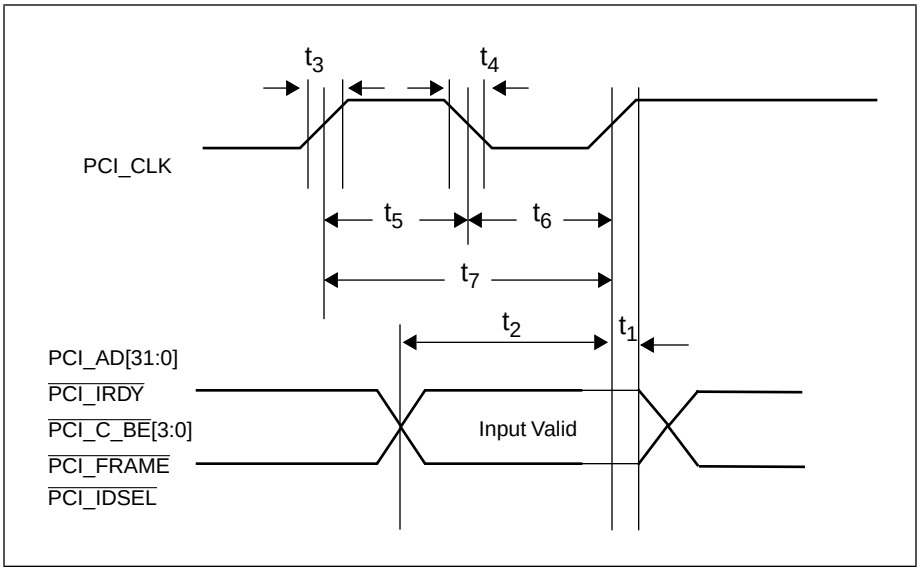
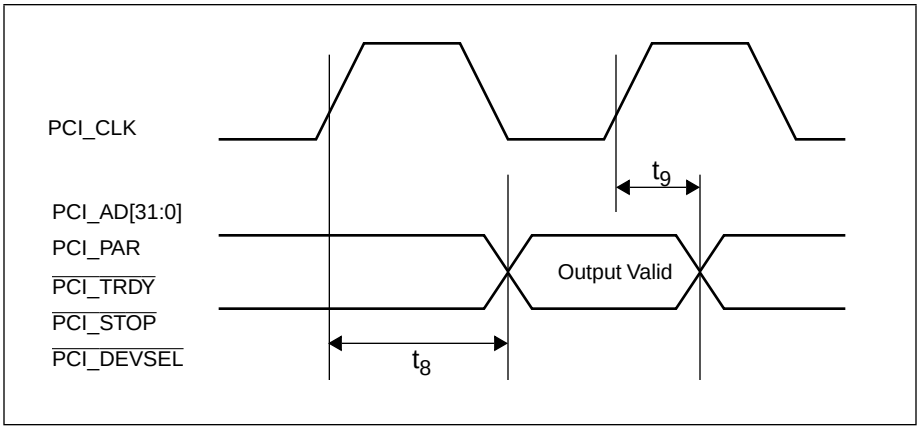


Figure 63. PCI/AGP Output Timing Diagram



SYNCHRONOUS MEMORY INTERFACE

Overview

The Bt2166 multimedia adapter supports only synchronous dynamic memory for frame buffer memory. Either synchronous dynamic graphics RAM (SGRAM) or synchronous dynamic RAM (SDRAM) may be used. The frame buffer may be either 1 or 2 ranks. A rank is defined as an amount of memory that is controlled with a single $\overline{\text{SDCS}}$ (Chip Select) and provides 64 bits of data to the Bt2166. The term RANK is used to avoid confusion with the internal banks of synchronous memories, which are called banks in this documentation.

The Bt2166 only supports a 64 bit interface to memory. The control signals for the memories are comprised of the signals $\overline{\text{SDRAS}}$, $\overline{\text{SDCAS}}$, $\overline{\text{WE}}$, DQM[7:0], MA[8:0], BS and $\overline{\text{SDCS}}$[1:0]. All signals except $\overline{\text{SDCS}}$[1:0] are common to both ranks. The column address for the memories is presented on bits [7:0] of the memory address bus, while the row address is presented on bits [8:0] of the memory address bus. Both row and column accesses utilize the BS (bank select) signal to determine the selected internal bank.

The signals $\overline{\text{SDRAS}}$, $\overline{\text{SDCAS}}$, $\overline{\text{WE}}$ and $\overline{\text{SDCS}}$ comprise the command for the synchronous memories. The $\overline{\text{SDRAS}}$ signal is analogous to the $\overline{\text{RAS}}$ signal in an asynchronous system, while the $\overline{\text{SDCAS}}$ signal is analogous to the $\overline{\text{CAS}}$ signal. The $\overline{\text{WE}}$ signal determines the operation type and the DQM signals provide a byte addressable masking ability. For complete details on the command encoding, refer to the documentation of the specific synchronous memory being used.

The Bt2166 memory interface can operate at speeds up to 83Mhz, allowing for burst rates of over 660 MB/s to and from the frame buffer. The Bt2166 also uses the performance features of the synchronous memories whenever possible, including hidden precharging and the ability to maintain multiple open pages. Additional benefits are available when SGRAM is used. In this mode, a special graphics feature called block write enables data expansion at rates over 1.7 Gbytes/s into the frame buffer.

The Bt2166 is designed primarily for use with SGRAM, but commodity SDRAM could be used to achieve alternative price/performance targets. When 16Mbit SDRAM is used, only a single rank of 8Mbytes is supported. This mode uses the $\overline{\text{SDCS}}$ pins as additional address bits to access the increased memory range. Table 186 on page 332 summarizes the supported memory configurations and memory types used.

Table 186. Supported Memory Configurations

Total Frame Size Buffer	RANK0		RANK1		Notes:
	Size	Type	Size	Type	
2Mbytes	2MB	(2) 8Mbit SGRAM (256Kx32)	0	Not Installed	Single Rank
4Mbytes	2MB	(4) 8Mbit SGRAM (256Kx32)	2MB	8Mbit SGRAM (256Kx32)	Dual Ranks
8Mbytes	8MB	(4) 16Mbit SDRAM (1Mx16)	0	Not Installed	Single Rank Not fully simulated in pass A

The Bt2166 uses the memory interface pins for ROM accesses also. A special pin, $\overline{\text{BIOS\_CS}}$, distinguishes between ROM and RAM accesses. The ROM data lines are attached to the lower memory data pins while the ROM address is connected to the upper address pins. The ROM timings are programmable to allow for multiple vendors. ROM operation is discussed in more detail in “ROM Interface” on page 338.

Memory-Related Pin Descriptions

Table 187 lists the pins used by the internal memory controller for frame buffer and ROM accesses.

Table 187. Frame Buffer and ROM Access Pins

Name	Pin #	Description	I/O	Max Load (pf)
$\overline{\text{SDCS}}$[1:0]	174, 175	Chip selects for each rank. SDCS[0] controls rank0 and SDCS[1] controls rank. If SDRAM mode is enabled (see "Internal Configuration" on page 337), these pins are used as additional address pins. In this mode, SDCS[0] becomes MA[9] and SDCS[1] becomes MA[10]. See the SDRAM connection section for more information.	O	
$\overline{\text{SDRAS}}$	173	Synchronous $\overline{\text{RAS}}$ (Row Address Strobe) signal. This signal connects to the $\overline{\text{RAS}}$ pin of all synchronous memories installed.	O	
$\overline{\text{SDCAS}}$	177	Synchronous $\overline{\text{CAS}}$ (Column Address Strobe) signal. This signal connects to the $\overline{\text{CAS}}$ pin of all synchronous memories installed. This signal is also used as the $\overline{\text{OE}}$ for the ROM.	O	
DQM[7:0]	118, 119, 138, 139, 180, 181, 2, 3	Data Q Mask Enables. These signals connect to the DQM pins for each rank of memory installed. DQM[0] is the mask for the low order byte within the 64 bit memory quadword. DQM[7] is the mask for the high order byte within the 64 bit memory quadword.	O	
$\overline{\text{WE}}$	176	Write Enable Signal. This signal is the write enable for all ranks and banks. This signal is also used as the write enable for ROM accesses.	O	
$\overline{\text{MEMCLK\_OUT}}$	168	The external clock for connection to all synchronous memory chips. This is a very timing critical signal and must be routed within the PCB carefully. All control signals and data are clocked relative to the rising edge of this clock.	O	
DSF	172	Data Special Function. This pin enables special functions with the SGRAM. Leave unconnected when using with SDRAM. Connect to all DSF pins in all ranks/banks when used with SGRAM.	O	
MDATA[63:0]	158, 140, 155-149, 130-137, 129-127, 124-120, 142-148, 182-192, 194-198, 4-11, 200-207	Memory data bus. These signals comprise the 64 bit data interface for the Bt2166.	I/O	
BS	159	Bank Select. This pin selects the appropriate bank within the synchronous memory and should be connected to the bank select input of all synchronous memory chips installed.	O	
MA[8:0]	160-165, 169-171	Memory Address bus. These signals must be connected to the memory address pins of all synchronous memory chips installed.	O	
$\overline{\text{BIOS\_CS}}$	199	ROM chip select. This signal, in conjunction with $\overline{\text{SDCAS}}$ and $\overline{\text{WE}}$, enables reading and writing to/from a ROM connected to the memory data bus.	O	

External Configuration/Memory Configuration Resistors

The Bt2166 supports a variety of synchronous memories from various vendors. The memory subsystem is optimized with 6 external configuration bits that the system BIOS reads at reset. The values are indexes into a look-up table and set different configurations of the internal memory configuration registers (see Table 188). The 6 external configuration bits are connected to the video data bus bits 5 through 0 and are tied to VDD or Gnd with a 20K resistor. When connected to VDD through the resistor, the external configuration bit is a binary "1". When connected to Gnd through the resistor, the external configuration bit is a binary "0".

Table 188. External Configuration Bits

Bit #	Pin #	Pin Name	Description
0	13	VDATA[0]	Memory type installed bit 0 (MEM_TYPE[0])
1	14	VDATA[1]	Memory type installed bit 1 (MEM_TYPE[1])
2	15	VDATA[2]	Memory type installed bit 2 (MEM_TYPE[2])
3	16	VDATA[3]	Memory type installed bit 3 (MEM_TYPE[3])
4	17	VDATA[4]	Memory type installed bit 4 (MEM_TYPE[4])
5	18	VDATA[5]	Memory controller speed 0 = 83Mhz 1 = 66Mhz

The 5 bit (MEM_TYPE[4:0]) memory type installed vector causes the Bt2166 BIOS to update the synchronous memory configuration register, MEM_CFG, to the values listed in Table 189 on page 335. All other bits are preserved or set by additional BIOS routines.

NOTE

VDATA[5:0] is written to the configuration register GRP_CFG7[5:0] as the read-only DRAM_TYPE. See Table 23 on page 52 and "Register Initialization" on page 54.

Table 189. BIOS Reset to Memory Configuration Register

Memory Type	Memory Configuration Bit #									Notes
	[27:26]	[25]	[22]	[21]	[20]	[19]	[18]	[12]	[11]	
	SDRAM	RAS	RP	REF_RC	RCD	RBSTP	WBSTP	TYPE1	TYPE0	
00000	00	0	1	0	1	1	1	1	1	Samsung SGRAM
00001	00	0	1	0	1	0	0	1	1	Hitachi, Micron SGRAM
00010	00	0	1	0	1	1	0	1	1	Fujitsu SGRAM
00011	00	0	1	0	1	0	0	1	1	IBM SGRAM
00100	00	1	1	1	1	1	0	1	1	NEC SGRAM
00101	00	1	1	1	1	0	0	1	1	Generic SGRAM0
00110	00	1	1	1	1	0	1	1	1	Generic SGRAM1
00111	00	1	1	1	1	1	0	1	1	Generic SGRAM2
01000	00	1	1	1	1	1	1	1	1	Generic SGRAM3
01001	11	1	1	1	1	0	0	0	0	Generic SDRAM0
01010	11	1	1	1	1	0	1	0	0	Generic SDRAM1
01011	11	1	1	1	1	1	0	0	0	Generic SDRAM2
01100	11	1	1	1	1	1	1	0	0	Generic SDRAM3
01101-11111	Reserved									

The SGRAMs listed in Table 190 have been simulated at release time.

Table 190. Currently Simulated SGRAMs

Vendor	Part Number	Type	Speed Option	Memory Type (binary)
Samsung	KM4132G271	SGRAM	-12	00000
Hitachi	HM5283206	SGRAM	-12	00001
IBM	IBM038329NQ6	SGRAM	-12	00011
Micron	MT41LC256K32D4	SGRAM	-12	00001
NEC	UPD481850GF	SGRAM	-12	00100
Fujitsu	MB81G83222	SGRAM	-12	00010

NOTE

No SDRAM models have been simulated at REV A release time.

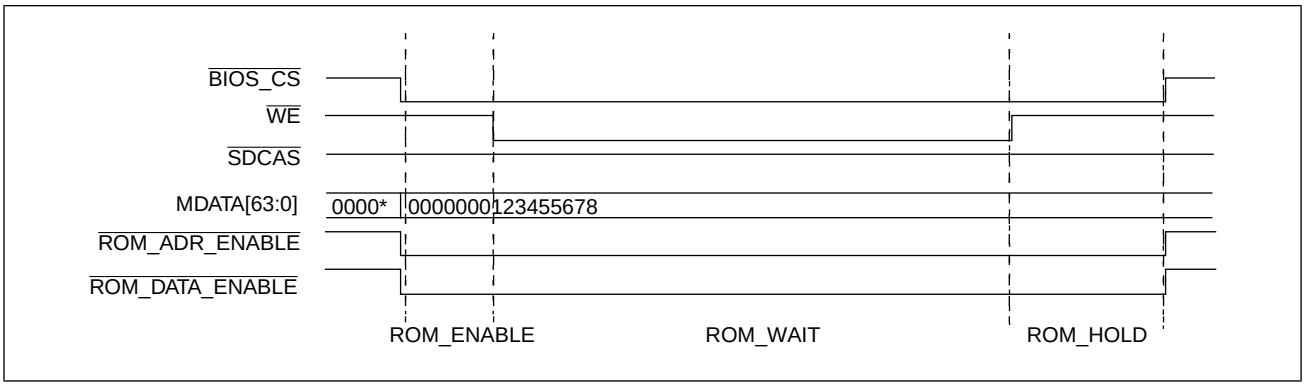
Internal Configuration

The Bt2166 provides a programmable method of accommodating new RAM types and different speed ROMs. This is accomplished with 2 configuration registers: MEM_CFG controls the operation of the memories and GEN_CFG controls the timing for ROM accesses. Refer to “Configuration and PLL Register Definitions” on page 141 for details.

ROM Interface

The Bt2166 supports a glue-less interface for 8 bit wide 3V ROM devices. Only 3V OTP or 3V writable Flash ROMs are supported. The Bt2166 does not support any device that requires greater than 3.3 volts to read or write the device. Connection to the ROM is made through the Bt2166 memory data bus only. Bits 35:16 of the memory data bus contain the address for the ROM during ROM cycles, while memory data bits [7:0] contain the data (read or write). The ROM is selected by means of the $\overline{\text{BIOS\_CS}}$ pin (pin 199). During ROM cycles, the frame buffer memory is deselected via the $\overline{\text{SDCS}}$[1:0] pins. Signal pins $\overline{\text{WE}}$ and $\overline{\text{SDCAS}}$ are also used for the ROM $\overline{\text{WE}}$ and $\overline{\text{OE}}$, respectively. The $\overline{\text{ROM\_DATA\_ENABLE}}$ and $\overline{\text{ROM\_ADR\_ENABLE}}$ signals are internal chip signals controlling the output enable of the signals. See Figure 64.

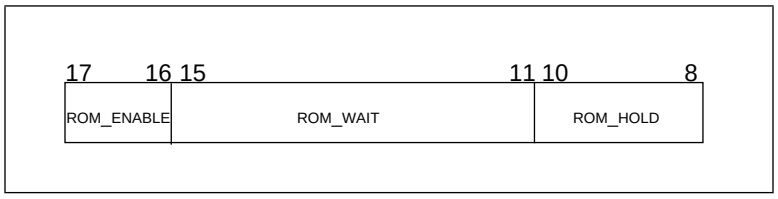
Figure 64. ROM Write Cycle



The Bt2166 memory controller incorporates a flexible method for reading ROM and writing flash ROM. The address and data bus for the ROM are connected to the Bt2166 memory data bus only. The Bt2166 memory address bus is not used during ROM cycles. A typical ROM write cycle is shown in Figure 64. The address and data for the ROM write are available at the falling edge of $\overline{\text{BIOS\_CS}}$.

The access times for the ROM is programmed using the GEN_CFG[17:8] register; refer to “General Configuration Register” on page 141 for more information. The format of the ROM configuration bits is shown in Figure 65.

Figure 65. ROM Configuration Bits (GEN_CFG[17:8])

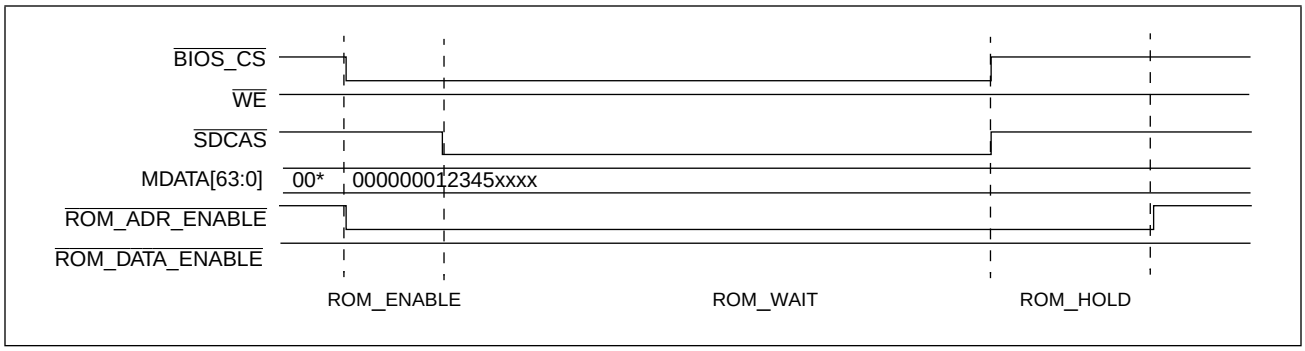


GEN_CFG[17:8] consists of three 3 programmable cycles: ROM_ENABLE, ROM_WAIT, and ROM_HOLD. The minimum time for each interval is one memory clock cycle. ROM_ENABLE is programmable from a minimum of 1 clock to a maximum of 4 clocks in length and is controlled by GEN_CFG[17:16]. A value of 2'b00 in these bits configures the Bt2166 controller for a 1 memory clock ROM_ENABLE interval. A value of 2'b11 in these bits configures the interval for 4 memory clocks in length.

ROM_WAIT is controlled by GEN_CFG[15:11]. A value of 5'b00000 in these bits will program the interval for 1 memory clock cycle in length. The maximum length of this interval is 32 clocks.

ROM_HOLD is controlled by GEN_CFG[10:8]. The length of this interval is from 1 to 8 clocks in length. ROM reads are controlled by the same bits as ROM writes, although different signals are toggled. A typical ROM read cycle is shown in Figure 66.

Figure 66. ROM Read Cycle



Memory Map Information

The Bt2166 can support up to 8Mbytes of attached frame buffer. The linear address is modified according to the addressing mode to accelerate better performance in different situations. There are 2 different addressing modes available, large page mode (fine) and normal page mode (lumpy). In large page mode, the linear address is altered to provide a large page size by moving the low order row address bit to the bank address position. This has the effect of increasing the page size of the access to twice the normal size. Table 191 lists the AC Timing Specifications.

Table 191. AC Timing Specifications

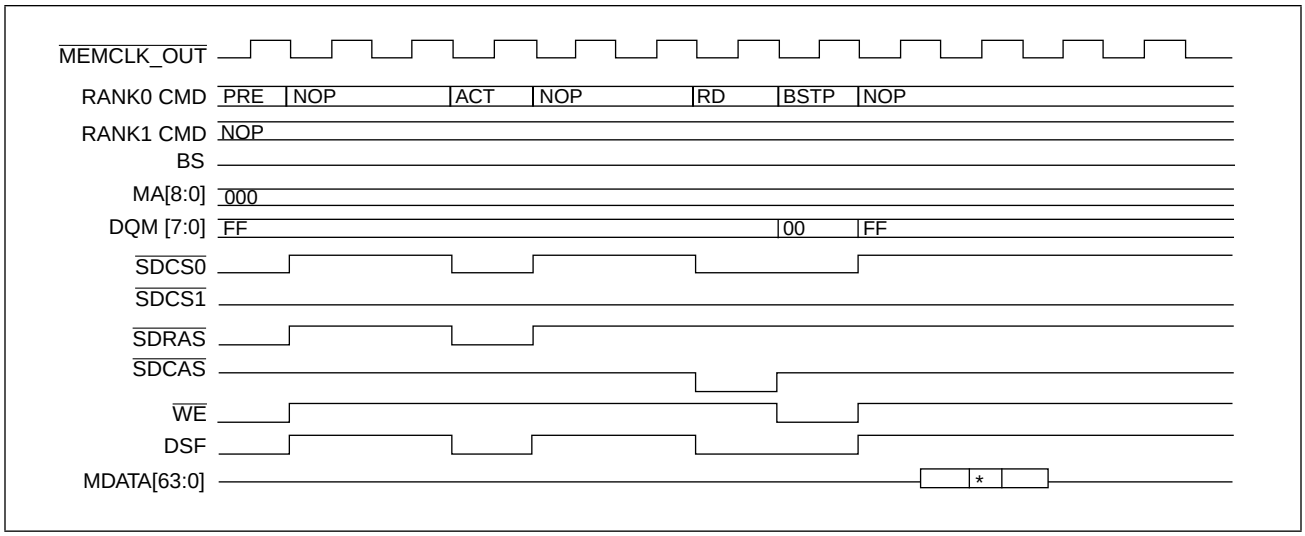
#	Parameter	Symbol	Min	Max	Unit	Note
1	$\overline{\text{MEMCLK\_OUT}}$ cycle time	tcc	12	--	ns	
2	CAS Latency (read)	tcl	3	3	clks	
3	Output data setup time to $\overline{\text{MEMCLK\_OUT}}$ rising edge	tdos	3.5		ns	CL=3 supported only
4	Output data hold time from $\overline{\text{MEMCLK\_OUT}}$ rising edge	tdoh	1.5		ns	
5	Data input setup time	tss	1.0		ns	
6	Data input hold time	tsh	3.0		ns	
7	$\overline{\text{SDRAS}}$, $\overline{\text{SDCAS}}$, $\overline{\text{WE}}$, DSF, $\overline{\text{SDCS}}$, DQM setup to $\overline{\text{MEMCLK\_OUT}}$ rising edge	tcms/tcs	3.5		ns	
8	SDRAS, SDCAS, $\overline{\text{WE}}$, DSF, $\overline{\text{SDCS}}$, DQM hold from $\overline{\text{MEMCLK\_OUT}}$ rising edge	tcmh/tch	1.5		ns	
9	Row active to Row active delay	trrd	2		clks	
10	$\overline{\text{RAS}}$ to $\overline{\text{CAS}}$ delay	trcd	3		clks	
11	Row precharge time	trp	2/3		clks	
12	Row active time	tras	6/7		clks	
13	Row cycle time	trc	9/10		clks	Configurable
14	Last data in to new column address delay	tcdl/tdal	1		clks	Configurable
15	Last data in to row precharge	trdl	2		clks	
16	Last data in to burst stop	tbdl	0/1		clks	Configurable
17	Column address to column address delay	tccd	1		clks	
18	Block write cycle time	tbwc	3		clks	
19	Block write data in to PRE command	tbpl	2		clks	
20	Mode register set cycle time	trsc	3		clks	
21	Data in to PRE command period	tdpl/trdl	2		clks	

Timing Diagrams

All the timing diagrams shown assume that the memory has been previously initialized with a CAS Latency of 3 and is operating in FULL PAGE mode.

A typical read cycle is shown in Figure 67. The read cycle is initiated by performing a READ command after a bank has been previously activated with the ACT command. The Bt2166 will always expect the read data to be valid on the third clock edge after the READ command was issued (CAS Latency = 3). The read command is terminated with the BSTP command. The timing of the BSTP command for reads is under control of MEM_CFG[19] (see Table 111 on page 143). A value of 0 will cause the Bt2166 to issue the BSTP command 1 cycle prior to the last read data required. A value of 1 in this bit will cause the BSTP to be issued 2 cycles prior to the last read data required. The DQM signals for the read operation always operate with a latency of 2 clocks.

Figure 67. Single Quadword Read Cycle - Page Miss



The Bt2166 will burst read whenever possible, as shown in Figure 68. The burst size is governed by the internal requesting unit and can be up to the maximum page size without any intervening wait states.

Figure 68. Burst Read Cycle - Page Miss

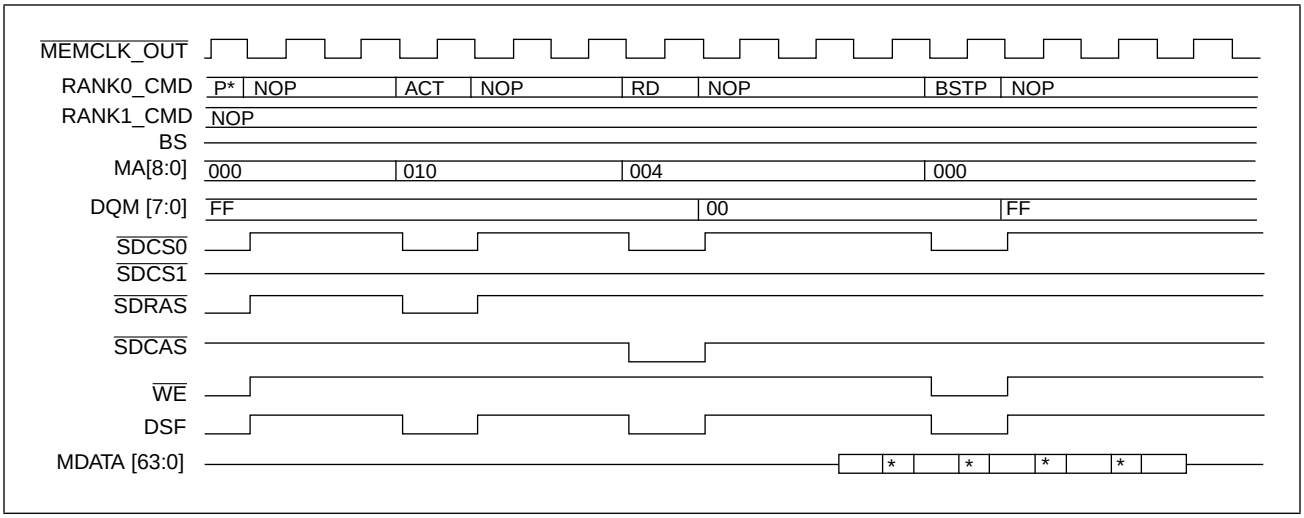
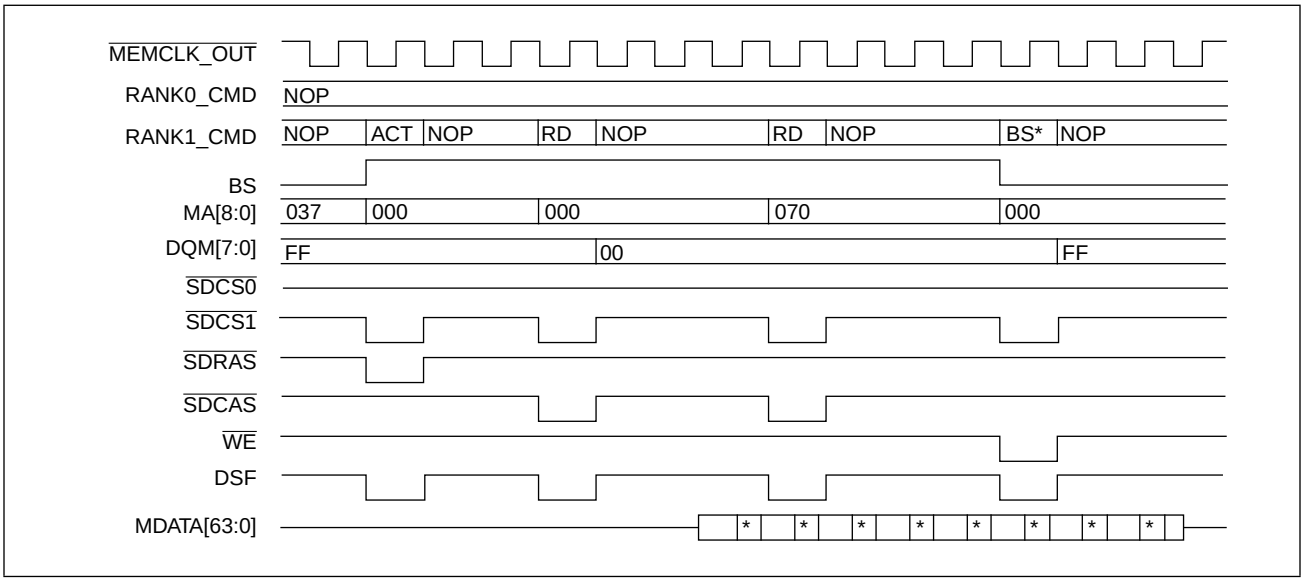


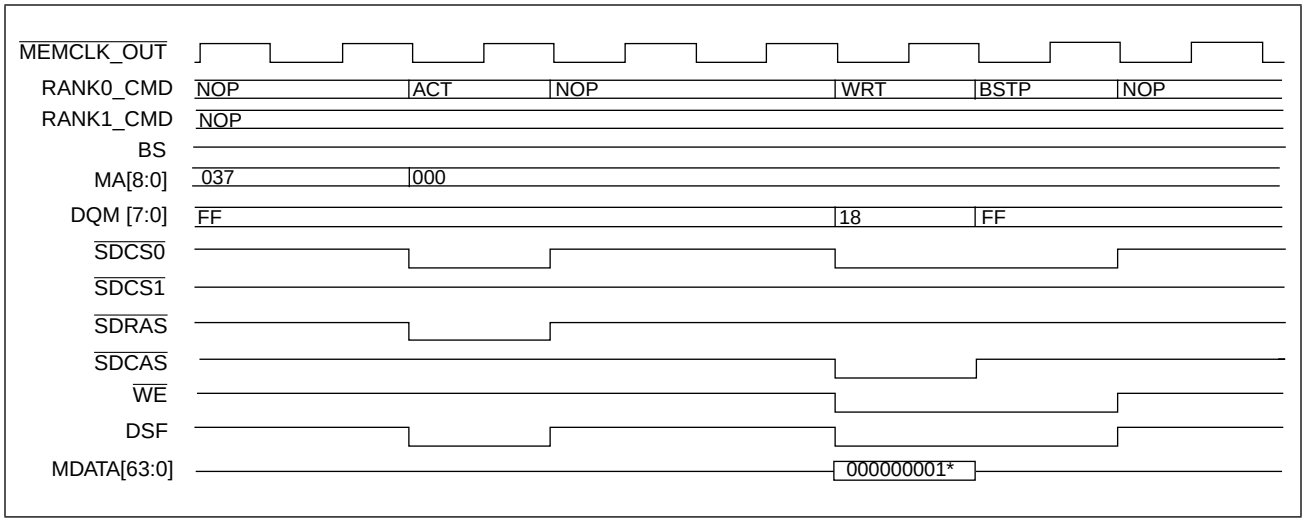
Figure 69 shows page mode read cycle. The Bt2166 memory controller maintains all open pages based on the previous accesses. No auto precharge cycles are performed.

Figure 69. Page Mode Read Cycle



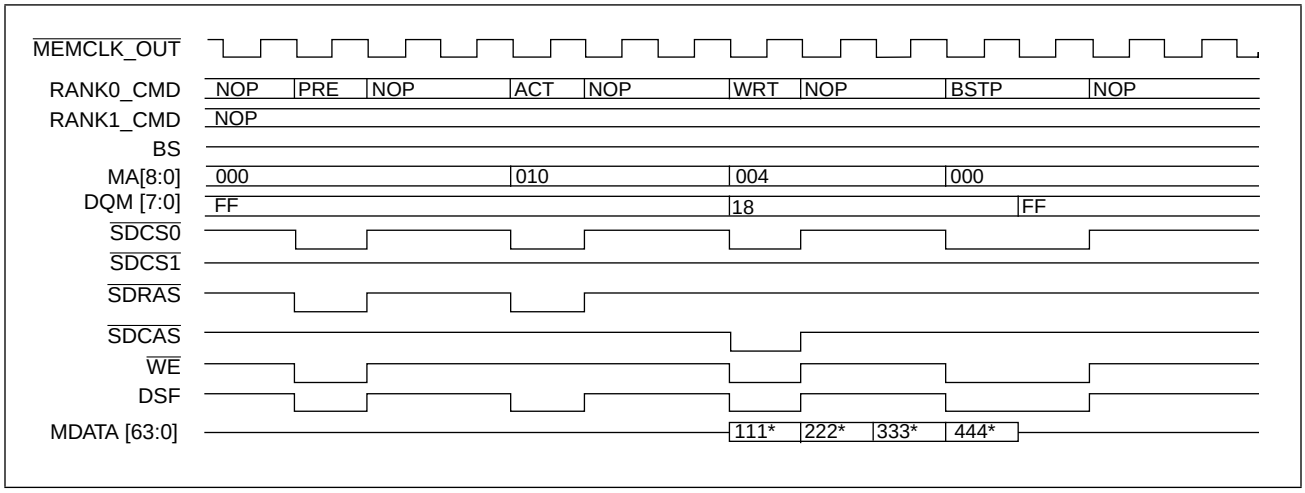
A typical write cycle is shown in Figure 70. The write operation always operates with a 0 clock latency, for both the data and the DQM signals. A BSTP is always used to terminate a write operation when the Bt2166 is idle. If other requests are made internally, the write operation either terminate with a BSTP command or by a PRECHARGE command to the same rank and bank. Bit 18 of the MEM_CFG controls whether the data presented during a BSTP command is expected to be written or not. This is a memory vendor specific issue dependent on the memory type installed.

Figure 70. Single Quadword Write Cycle



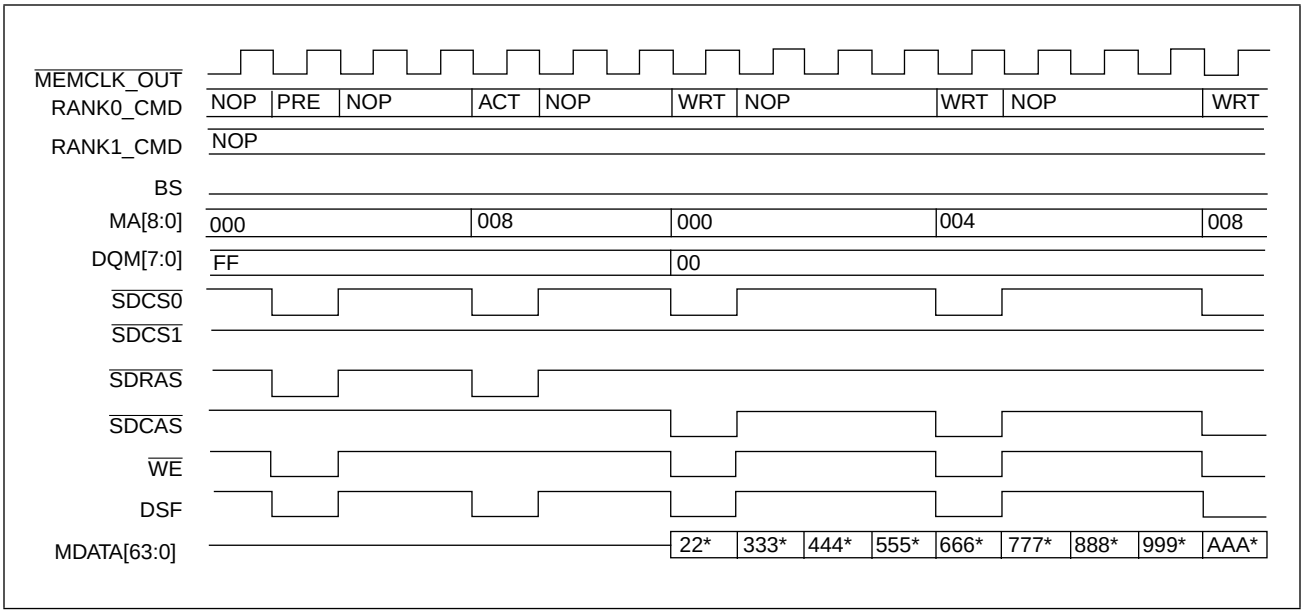
The Bt2166 will burst whenever possible, as shown in Figure 71. The burst size is governed by the internal requesting unit and can be up to the maximum page size without any intervening wait states.

Figure 71. Burst Write Cycle - Page Miss



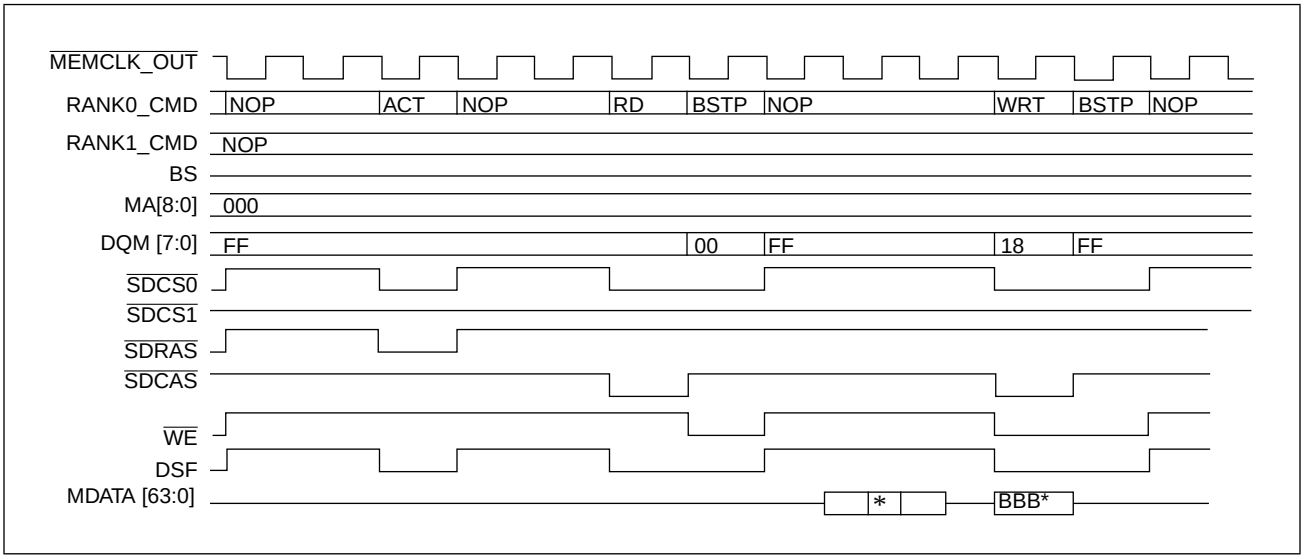
If a requested page has been previously activated, subsequent writes may be done in page mode, as shown in Figure 72.

Figure 72. Page Mode Write Cycle



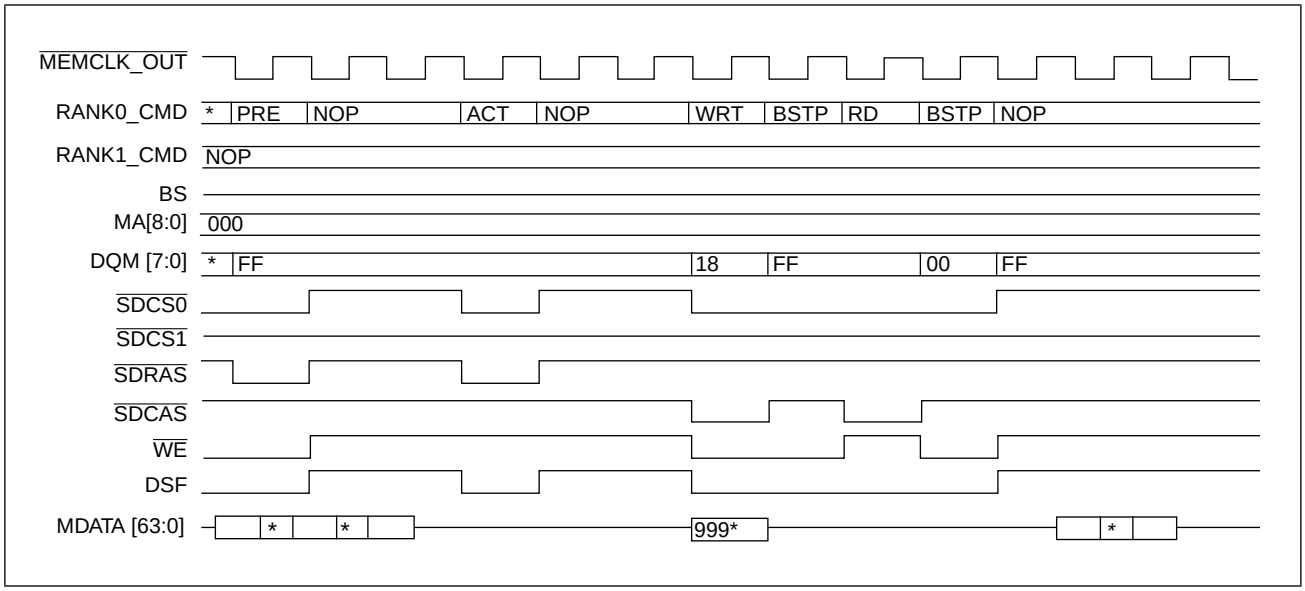
Page mode is also maintained between read and write operations; Figure 73 shows a read followed by a write. The Bt2166 allows one cycle between the final read and the first write to avoid any bus contention.

Figure 73. Page Mode Read Followed by Write



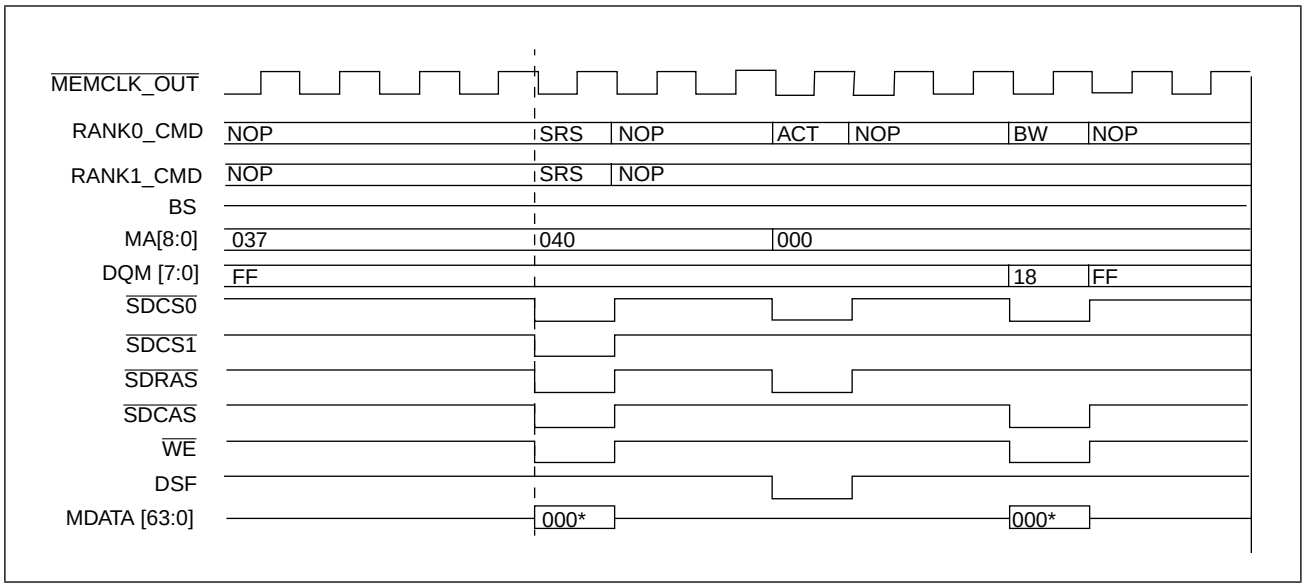
Because of the latencies associated with read operations, a read operation can immediately follow a write operation, as shown in Figure 74.

Figure 74. Page Mode Write Followed by Read



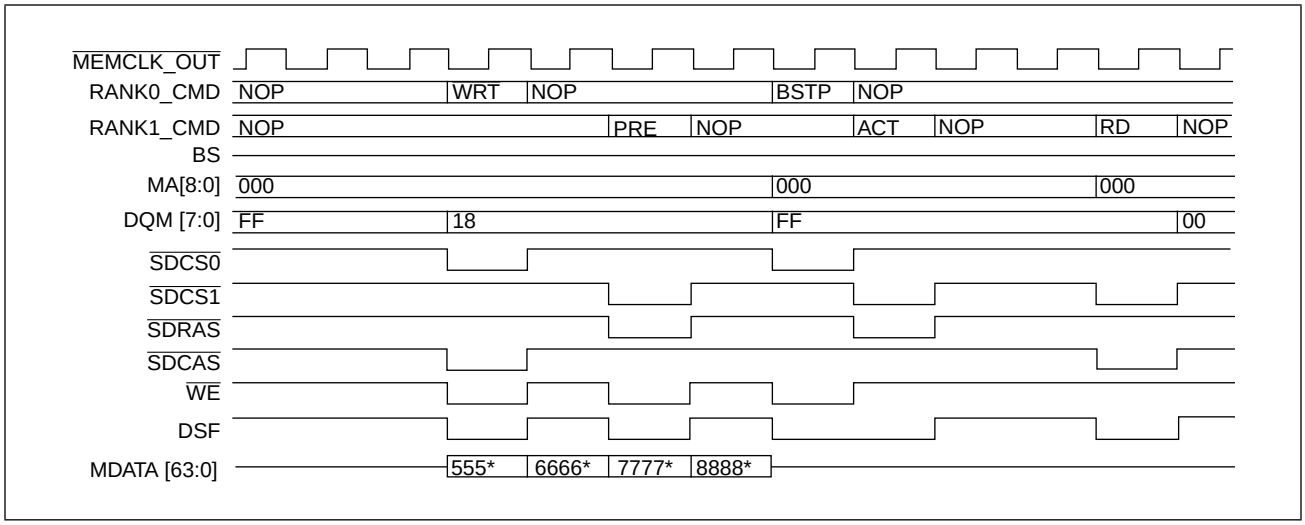
The Bt2166 supports a single color register when SGRAM is installed. If required, this register is updated prior to a block write type operation. A sample LCR command followed by a block write is shown in Figure 75.

Figure 75. LCR Command Followed by Block Write



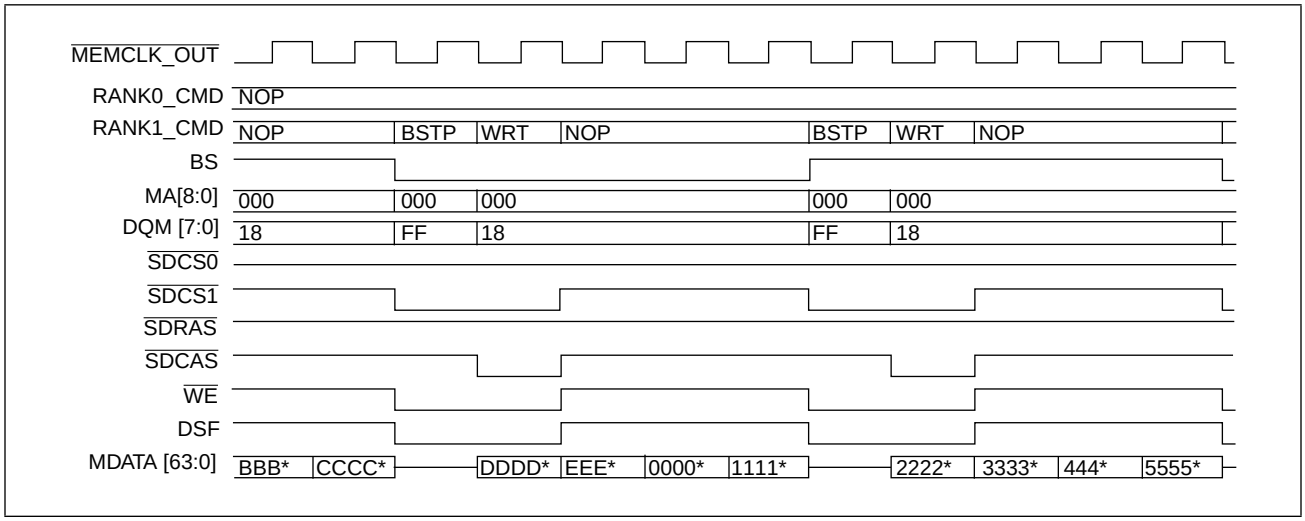
One performance advantage of the synchronous memory system over its asynchronous counter-part is the ability to hide precharging. If an operation is in progress to one bank or rank of the memory system, another rank or bank can be precharged in preparation of a page activation, as long as the command bus is idle and the operation is to a different rank or bank than the current operation. Figure 76 on page 346 shows an example of a write operation occurring while a precharge command to another rank is performed.

Figure 76. Burst Write with Hidden Precharge Command



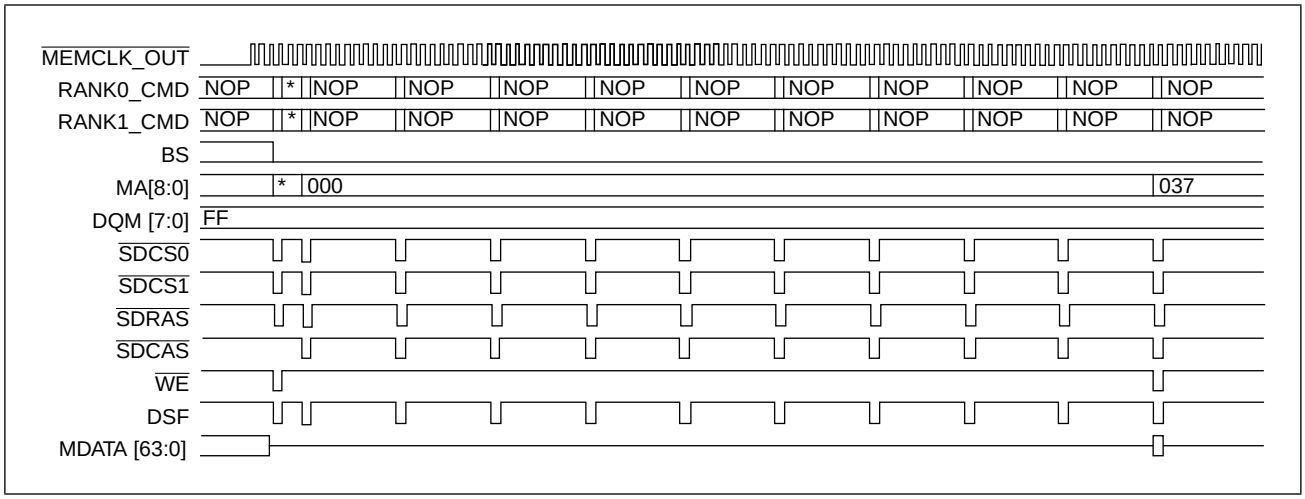
Another advantage of a multibank memory device is the ability to have more than one open page, as illustrated in Figure 77. In this situation, both banks of the SGRAM have been previously activated with non sequential rows. If a write operation is required to both of these banks, the Bt2166 will chain all writes together with only a BSTP command in between.

Figure 77. Multibank Write



The initialization sequence performed by the Bt2166 after a reset or after bit 24 of the MEM_CFG has changed values is shown in Figure 78 on page 347. This is a PRECHARGE ALL command to both ranks, followed by 9 refresh (CBR) cycles, followed by the MRS command. Note that the data used in the MRS command comes from the MEM_CFG[10:0].

Figure 78. Bt2166 Memory Initialization Sequence



OPERATING SPECIFICATIONS

The parametric values in this chapter are undergoing testing. The values listed are preliminary and approximate in this release.

Absolute Maximum Ratings

Table 192. Absolute Maximum Ratings

Parameter	Symbol	Min	Typical	Max	Units
VAA, VDD, VDDQ(measured to GND)		3.0	3.3	5.5	V
Voltage on Any Signal Pin		GND – 0.5		VDD + 0.5	V
Ambient Operating Temperature	T_A	0		+70	˚C
Storage Temperature	T_S	–65		+150	˚C
Junction Temperature	T_J			+125	˚C
Vapor Phase Soldering (1 minute)	T_{VSOL}			220	˚C

Note: Stresses above those listed in this table may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those listed in the operational sections of this specification are not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability. This device employs high-impedance CMOS devices on all signal pins. It should be handled as an ESD-sensitive device. Voltage values on any signal pin that extend beyond the power supply rails by more than the amount(s) specified above can cause destructive latchup.

Recommended Operating Conditions

Bt2166 is designed to drive up to 4 RAM modules. Table 193 gives the recommended operating conditions for the Bt2166.

Table 193. Recommended Operating Conditions

Parameter	Min	Max	Units
Load Capacitance on Digital Outputs:			
$\overline{\text{BIOS\_CS}}$		20	pF
$\overline{\text{RESET\_OUT}}$		40	pF
$\overline{\text{DQM}}$[7:0]		20	pF
MA[8:0], BS		50	pF
MDATA[63:0]		40	pF
$\overline{\text{SDCAS}}$, $\overline{\text{WE}}$, $\overline{\text{SDRAS}}$, DSF		60	pF
$\overline{\text{SDCS}}$[1:0]		tbd	pF
$\overline{\text{MEMCLK\_OUT}}$		tbd	pF
Supply Voltage	3.135	3.465	V
Ambient Temperature T_a	0	70	˚C

Target DC Characteristics

Target DC characteristics are shown in Table 194, I^2C characteristics in Table 195, and DDC characteristics in Table 196.

Table 194. Target DC Characteristics *(1 of 2)*

Parameter	Symbol	Min	Typical	Max	Units
Analog Outputs (IOR, IOG, IOB)					
Accuracy					
Integral Linearity Error	IL			±1	LSB
Differential Linearity Error	DL			±1	LSB
Monotonicity			Guaranteed		
Coding					Binary
Analog Output (VIDEO)					
Resolution		8			Bits
Integral Linearity Error	IL			±1	LSB
Differential Linearity Error	DL			±1	LSB
Monotonicity			Guaranteed		
Coding					Binary
Digital Inputs					
Input High Voltage Input Low Voltage	V_{IH} V_{IL}	2.0 GND–0.5		V_{DD} + 0.5 0.8	V V
Input High Current (Vin = 2.4 V) Input Low Current (Vin = 0.4 V)	I_{IH} I_{IL}	–1 –1		1 1	µA µA
Input Capacitance (v = 1 MHz, Vin = 2.4 V)	C_{IN}		7		pF
Digital Outputs (VS, HS)					
Output High Voltage (I_{OH} = –400 µA)	V_{OH}	2.4			V
Output Low Voltage (I_{OL} = 3.2 mA)	V_{OL}			0.4	V
Three-State Current (0–2.4V)	I_{OZ}			50	µA
Output Capacitance (f = 1 MHz, Vout = 2.4 V)	CDOUT		7		pF

Table 194. Target DC Characteristics *(2 of 2)*

Parameter	Symbol	Min	Typical	Max	Units
Analog Outputs (IOR, IOG, IOB)					
Output Current (Standard RS-343A)					
White Level Relative to Black(1)		16.74	17.62	18.5	mA
Black Level Relative to Blank(1)			1.44		mA
SETUP = 7.5 IRE		0.95	1.44	1.90	mA
SETUP = 0 IRE		0	5	50	µA
Blank Level		6.29	7.62	8.96	mA
Sync Level		0	5	50	µA
LSB Size			68.8		µA
RGB DAC-to-DAC Matching		0	2	5	%
Output Compliance	V_{OC}	–0.3		+1.5	V
Output Impedance	RAOUT		10		kΩ
Output Capacitance (f = 1 MHz, Iout = 0 mA, Vout = 1.0 V)	CAOUT		30		pF
Onboard VREF(2)	VREFOUT		1.235		V
Power Supply Rejection Ratio (COMP = 0.1 µF, f = 1 kHz)	PSRR		.5		% / % ΔVAA

Note: (1).When the internal voltage reference is used, RSET may require adjustment to meet these limits. Also, the "gray-scale" output current (white level relative to black) will have a typical tolerance of ±10% rather than the ±5% specified above.

(2).Onboard VREF numbers and use subject to change upon completion of characterization and yield studies.

3. Test conditions to generate RS-343A standard video signals (unless otherwise specified): "Recommended Operating Conditions" using external voltage reference with SETUP = 7.5 IRE, RSET = 442 Ω, VREF = 1.235 V, or alternatively, SETUP = 0 IRE, RSET = 409 Ω. As the above parameters are guaranteed over the full temperature range, temperature coefficients are not specified or required.

Table 195. DC Characteristics for IIC_SDA and IIC_SCL I/O (Fast Mode)

Parameter	Symbol	Min.	Max.	Units
Low level input voltage	V_{IL}	–0.5	.8	V
High level input voltage	V_{IH}	2.0	V_{DD} + 0.5	V
Spike suppression	t_{SP}	0	50	ns
Low level output voltage (open drain) at 3 mA sink current	V_{OL1}		0.4	V
Output fall time (10–400 pF bus capacitance)	t_{OF}	$20 + 0.1C_b$ (1)	250 (2)	ns
Input current (Vi = 0.4–0.9 $V_{DD\ max.}$)	I_i	–10	10	µA
Note: (1).C_b = capacitance of one bus line in pF. (2).SDA and SCL lines will not be obstructed if V_{DD} is switched off.				

Table 196. DC Characteristics for DDC_SDA and DDC_SCL I/O (Standard Mode)

Parameter	Symbol	Min.	Max.	Units
Low level input voltage	V_{IL}	–0.5	.08	V
High level input voltage	V_{IH}	2.0	V_{DD} + 0.5	V
Spike suppression	t_{SP}	0	50	ns
Low level output voltage (open drain) at 3 mA sink currentt	V_{OL1}		0.4	V
Output fall time (10–400pF bus capacitance)	t_{OF}	$20 + 0.1C_b$ (1)	250 (2)	ns
Input current (Vi = 0.4–0.9$V_{DD\ max.}$)	I_i	–10	10	µA
Note: (1).C_b = capacitance of one bus line in pF. (2).SDA and SCL lines will not be obstructed if V_{DD} is switched off.				

GUI, MEM, and SYS Clock PLL Rate Selection

Table 197 lists the selectable GUI clock, MEM clock, and SYS clock PLL rates for 14.31818 crystal frequency. The formula for determining the clock frequency is:

$$14.31818 * M/(N*2^{L})$$

where:
M min = 21
M max = 63
N min = 4
N max = 15
L = 1
VCO min = 50 MHz
VCO max = 175 MHz

Refer to “Configuration and PLL Register Definitions” on page 141 for defintions of the GUICLK_PLL, MEMCLK_PLL and SYSCLK_PLL registers. For Pixel clock PLL rates, refer to “Pixel Clock PLL Rate Selection” on page 358.

Table 197. GUI. MEM. and SYS Clock PLL Rate Selection *(1 of 3)*

Frequency MHz	M	N	L
49.0909	48	7	1
49.2187	55	8	1
49.3182	62	9	1
50.1136	28	4	1
51.0085	57	8	1
51.1364	50	7	1
51.3068	43	6	1
51.5454	36	5	1
51.9034	29	4	1
52.1591	51	7	1
52.5000	44	6	1
52.7983	59	8	1

Table 197. GUI. MEM. and SYS Clock PLL Rate Selection *(2 of 3)*

Frequency MHz	M	N	L
52.9773	37	5	1
53.1818	52	7	1
53.6932	30	4	1
54.2045	53	7	1
54.4091	38	5	1
54.5881	61	8	1
54.8864	46	6	1
55.2273	54	7	1
55.4829	31	4	1
55.8409	39	5	1
56.0795	47	6	1
56.2500	55	7	1
56.3778	63	8	1
57.2727	32	4	1
58.2954	57	7	1
58.4659	49	6	1
58.7045	41	5	1
59.0625	33	4	1
59.3182	58	7	1
59.6591	50	6	1
60.1364	42	5	1
60.3409	59	7	1
60.8523	34	4	1
61.3636	60	7	1
61.5682	43	5	1
62.0454	52	6	1
62.3864	61	7	1
62.6420	35	4	1
63.0000	44	5	1
63.2386	53	6	1
63.4091	62	7	1
64.4318	36	4	1
65.6250	55	6	1
65.8636	46	5	1

Table 197. GUI. MEM. and SYS Clock PLL Rate Selection *(3 of 3)*

Frequency MHz	M	N	L
66.2216	37	4	1
66.8182	56	6	1
67.2954	47	5	1
68.0114	38	4	1
68.7273	48	5	1
69.2045	58	6	1
69.8011	39	4	1
70.1591	49	5	1
70.3977	59	6	1
71.5909	40	4	1
72.7841	61	6	1
73.0227	51	5	1
73.3807	41	4	1
73.9773	62	6	1
74.4545	52	5	1
75.1704	42	4	1
75.8864	53	5	1
76.9602	43	4	1
77.3182	54	5	1
78.7500	44	4	1
80.1818	56	5	1
80.5398	45	4	1
81.6136	57	5	1
82.3295	46	4	1
83.0454	58	5	1
84.1193	47	4	1
84.4773	59	5	1
85.9091	48	4	1
87.3409	61	5	1

Pixel Clock PLL Rate Selection

Table 198 lists the selectable pixel PLL rates for 14.31818 MHz crystal frequency. The formula for determining the clock frequency is:

$$14.31818 * M/(N*2^{L})$$

where:
M min = 21
M max = 84
N min = 4
N max = 15
L min = 0
L max = 3
VCO min = 50 MHz
VCO max = 175 MHz

Refer to “Configuration and PLL Register Definitions” on page 141 for a defintion of the PIXCLK_PLL register. For GUI clock, memory clock, and system clock PLL rates refer to “GUI, MEM, and SYS Clock PLL Rate Selection” on page 355.

Table 198. Pixel Clock PLL Rate Selection Table *(1 of 13)*

Frequency MHz	M	N	L
49.0909	48	7	1
49.2187	55	8	1
49.3182	62	9	1
49.3977	69	10	1
49.4628	76	11	1
49.5170	83	12	1
50.1136	21	6	0
50.5909	53	15	0
50.6643	46	13	0
50.7645	39	11	0
50.8295	71	10	1
50.9091	32	9	0
51.0085	57	8	1
51.1364	25	7	0
51.3068	43	6	1
51.4153	79	11	1

Table 198. Pixel Clock PLL Rate Selection Table *(2 of 13)*

Frequency MHz	M	N	L
51.5454	36	5	1
51.7045	65	9	1
51.7657	47	13	0
51.9034	29	4	1
52.0661	40	11	0
52.1591	51	7	1
52.2614	73	10	1
52.5000	22	6	0
52.7169	81	11	1
52.7983	59	8	1
52.8671	48	13	0
52.9773	37	5	1
53.1818	26	7	0
53.2954	67	9	1
53.3678	41	11	0
53.4545	56	15	0
53.6932	30	4	1
53.9685	49	13	0
54.0186	83	11	1
54.0909	34	9	0
54.2045	53	7	1
54.4091	38	5	1
54.5881	61	8	1
54.6694	42	11	0
54.8864	23	6	0
55.0699	50	13	0
55.1250	77	10	1
55.2273	27	7	0
55.3636	58	15	0
55.4829	31	4	1
55.6818	35	9	0
55.8409	39	5	1

Table 198. Pixel Clock PLL Rate Selection Table *(3 of 13)*

Frequency MHz	M	N	L
55.9711	43	11	0
56.0795	47	6	1
56.1713	51	13	0
56.2500	55	7	1
56.3182	59	15	0
56.3778	63	8	1
56.4773	71	9	1
56.5568	79	10	1
57.2727	24	6	0
57.9886	81	10	1
58.0682	73	9	1
58.1676	65	8	1
58.2273	61	15	0
58.2954	57	7	1
58.3741	53	13	0
58.4659	49	6	1
58.5744	45	11	0
58.7045	41	5	1
58.8636	37	9	0
59.0625	33	4	1
59.1818	62	15	0
59.3182	29	7	0
59.4204	83	10	1
59.4755	54	13	0
59.6591	25	6	0
59.8760	46	11	0
59.9574	67	8	1
60.1364	21	5	0
60.3409	59	7	1
60.4545	38	9	0
60.5769	55	13	0
60.8523	34	4	1

Table 198. Pixel Clock PLL Rate Selection Table *(4 of 13)*

Frequency MHz	M	N	L
61.0909	64	15	0
61.1777	47	11	0
61.2500	77	9	1
61.3636	30	7	0
61.5682	43	5	1
61.6783	56	13	0
61.7472	69	8	1
62.0454	26	6	0
62.3864	61	7	1
62.4793	48	11	0
62.6420	35	4	1
62.7797	57	13	0
62.8409	79	9	1
63.0000	22	5	0
63.2386	53	6	1
63.4091	31	7	0
63.5369	71	8	1
63.6364	40	9	0
63.7810	49	11	0
63.8811	58	13	0
63.9545	67	15	0
64.4318	27	6	0
64.9091	68	15	0
64.9825	59	13	0
65.0826	50	11	0
65.2273	41	9	0
65.3267	73	8	1
65.4545	32	7	0
65.6250	55	6	1
65.8636	23	5	0
66.0227	83	9	1
66.0839	60	13	0

Table 198. Pixel Clock PLL Rate Selection Table *(5 of 13)*

Frequency MHz	M	N	L
66.2216	37	4	1
66.3843	51	11	0
66.4773	65	7	1
66.8182	28	6	0
67.1165	75	8	1
67.1853	61	13	0
67.2954	47	5	1
67.5000	33	7	0
67.6859	52	11	0
67.7727	71	15	0
68.0114	38	4	1
68.2867	62	13	0
68.4091	43	9	0
68.5227	67	7	1
68.7273	24	5	0
68.9062	77	8	1
68.9876	53	11	0
69.2045	29	6	0
69.3881	63	13	0
69.5454	34	7	0
69.6818	73	15	0
69.8011	39	4	1
70.0000	44	9	0
70.1591	49	5	1
70.2892	54	11	0
70.3977	59	6	1
70.4895	64	13	0
70.5682	69	7	1
70.6364	74	15	0
70.6960	79	8	1
71.5909	25	5	0
72.4858	81	8	1

Table 198. Pixel Clock PLL Rate Selection Table *(6 of 13)*

Frequency MHz	M	N	L
72.5454	76	15	0
72.6136	71	7	1
72.6923	66	13	0
72.7841	61	6	1
72.8926	56	11	0
73.0227	51	5	1
73.1818	46	9	0
73.3807	41	4	1
73.5000	77	15	0
73.6364	36	7	0
73.7937	67	13	0
73.9773	31	6	0
74.1942	57	11	0
74.2756	83	8	1
74.4545	26	5	0
74.6591	73	7	1
74.7727	47	9	0
74.8951	68	13	0
75.1704	21	4	0
75.4091	79	15	0
75.4959	58	11	0
75.6818	37	7	0
75.8864	53	5	1
75.9965	69	13	0
76.3636	32	6	0
76.7045	75	7	1
76.7975	59	11	0
76.9602	43	4	1
77.0979	70	13	0
77.3182	27	5	0
77.5568	65	6	1
77.7273	38	7	0

Table 198. Pixel Clock PLL Rate Selection Table *(7 of 13)*

Frequency MHz	M	N	L
77.9545	49	9	0
78.0992	60	11	0
78.1993	71	13	0
78.2727	82	15	0
78.7500	22	4	0
79.2273	83	15	0
79.3007	72	13	0
79.4008	61	11	0
79.5454	50	9	0
79.7727	39	7	0
79.9432	67	6	1
80.1818	28	5	0
80.4021	73	13	0
80.5398	45	4	1
80.7025	62	11	0
80.7954	79	7	1
81.1364	34	6	0
81.5035	74	13	0
81.6136	57	5	1
81.8182	40	7	0
82.0041	63	11	0
82.3295	23	4	0
82.6049	75	13	0
82.7273	52	9	0
82.8409	81	7	1
83.0454	29	5	0
83.3058	64	11	0
83.5227	35	6	0
83.7063	76	13	0
83.8636	41	7	0
84.1193	47	4	1
84.3182	53	9	0

Table 198. Pixel Clock PLL Rate Selection Table *(8 of 13)*

Frequency MHz	M	N	L
84.4773	59	5	1
84.6074	65	11	0
84.7159	71	6	1
84.8077	77	13	0
84.8864	83	7	1
85.9091	24	4	0
87.0105	79	13	0
87.1023	73	6	1
87.2107	67	11	0
87.3409	61	5	1
87.5000	55	9	0
87.6989	49	8	0
87.9545	43	7	0
88.1119	80	13	0
88.2954	37	6	0
88.5124	68	11	0
88.7727	31	5	0
89.0909	56	9	0
89.2133	81	13	0
89.4886	25	4	0
89.8140	69	11	0
90.0000	44	7	0
90.2045	63	10	0
90.3147	82	13	0
90.6818	38	6	0
91.1157	70	11	0
91.2784	51	8	0
91.4161	83	13	0
91.6364	32	5	0
91.8750	77	12	0
92.0454	45	7	0
92.2727	58	9	0

Table 198. Pixel Clock PLL Rate Selection Table *(9 of 13)*

Frequency MHz	M	N	L
92.4173	71	11	0
92.5175	84	13	0
93.0682	26	4	0
93.7190	72	11	0
93.8636	59	9	0
94.0909	46	7	0
94.2614	79	12	0
94.5000	33	5	0
94.8579	53	8	0
95.0206	73	11	0
95.4545	40	6	0
95.9318	67	10	0
96.1364	47	7	0
96.3223	74	11	0
96.6477	27	4	0
97.0454	61	9	0
97.3636	34	5	0
97.6240	75	11	0
97.8409	41	6	0
98.1818	48	7	0
98.4375	55	8	0
98.6364	62	9	0
98.7954	69	10	0
98.9256	76	11	0
99.0341	83	12	0
100.2273	28	4	0
101.5289	78	11	0
101.6591	71	10	0
101.8182	64	9	0
102.0170	57	8	0
102.2727	50	7	0
102.6136	43	6	0

Table 198. Pixel Clock PLL Rate Selection Table *(10 of 13)*

Frequency MHz	M	N	L
102.8306	79	11	0
103.0909	36	5	0
103.4091	65	9	0
103.8068	29	4	0
104.1322	80	11	0
104.3182	51	7	0
104.5227	73	10	0
105.0000	44	6	0
105.4339	81	11	0
105.5966	59	8	0
105.9545	37	5	0
106.3636	52	7	0
106.5909	67	9	0
106.7355	82	11	0
107.3864	30	4	0
108.0372	83	11	0
108.1818	68	9	0
108.4091	53	7	0
108.8182	38	5	0
109.1761	61	8	0
109.3388	84	11	0
109.7727	46	6	0
110.2500	77	10	0
110.4545	54	7	0
110.9659	31	4	0
111.3636	70	9	0
111.6818	39	5	0
112.1591	47	6	0
112.5000	55	7	0
112.7557	63	8	0
112.9545	71	9	0
113.1136	79	10	0

Table 198. Pixel Clock PLL Rate Selection Table *(11 of 13)*

Frequency MHz	M	N	L
114.5454	32	4	0
115.9773	81	10	0
116.1363	73	9	0
116.3352	65	8	0
116.5909	57	7	0
116.9318	49	6	0
117.4091	41	5	0
117.7273	74	9	0
118.1250	33	4	0
118.6363	58	7	0
118.8409	83	10	0
119.3182	50	6	0
119.9148	67	8	0
120.2727	42	5	0
120.6818	59	7	0
120.9091	76	9	0
121.7045	34	4	0
122.5000	77	9	0
122.7273	60	7	0
123.1363	43	5	0
123.4943	69	8	0
124.0909	52	6	0
124.7727	61	7	0
125.2841	35	4	0
125.6818	79	9	0
126.0000	44	5	0
126.4773	53	6	0
126.8182	62	7	0
127.0738	71	8	0
127.2727	80	9	0
128.8636	36	4	0
130.4545	82	9	0

Table 198. Pixel Clock PLL Rate Selection Table *(12 of 13)*

Frequency MHz	M	N	L
130.6534	73	8	0
130.9091	64	7	0
131.2500	55	6	0
131.7273	46	5	0
132.0454	83	9	0
132.4432	37	4	0
132.9545	65	7	0
133.6363	56	6	0
134.2329	75	8	0
134.5909	47	5	0
135.0000	66	7	0
136.0227	38	4	0
137.0454	67	7	0
137.4545	48	5	0
137.8125	77	8	0
138.4091	58	6	0
139.0909	68	7	0
139.6023	39	4	0
140.3182	49	5	0
140.7954	59	6	0
141.1363	69	7	0
141.3920	79	8	0
143.1818	40	4	0
144.9716	81	8	0
145.2273	71	7	0
145.5682	61	6	0
146.0454	51	5	0
146.7613	41	4	0
147.2727	72	7	0
147.9545	62	6	0
148.5511	83	8	0
148.9091	52	5	0

Table 198. Pixel Clock PLL Rate Selection Table *(13 of 13)*

Frequency MHz	M	N	L
149.3182	73	7	0
150.3409	42	4	0
151.3636	74	7	0
151.7727	53	5	0
152.7273	64	6	0
153.4091	75	7	0
153.9204	43	4	0
154.6363	54	5	0
155.1136	65	6	0
155.4545	76	7	0
157.5000	44	4	0
159.5454	78	7	0
159.8863	67	6	0
160.3636	56	5	0
161.0795	45	4	0
161.5909	79	7	0
162.2727	68	6	0
163.2273	57	5	0
163.6363	80	7	0
164.6591	46	4	0
165.6818	81	7	0
166.0909	58	5	0
167.0454	70	6	0
167.7273	82	7	0
168.2386	47	4	0
168.9545	59	5	0
169.4318	71	6	0
169.7727	83	7	0
171.8182	48	4	0

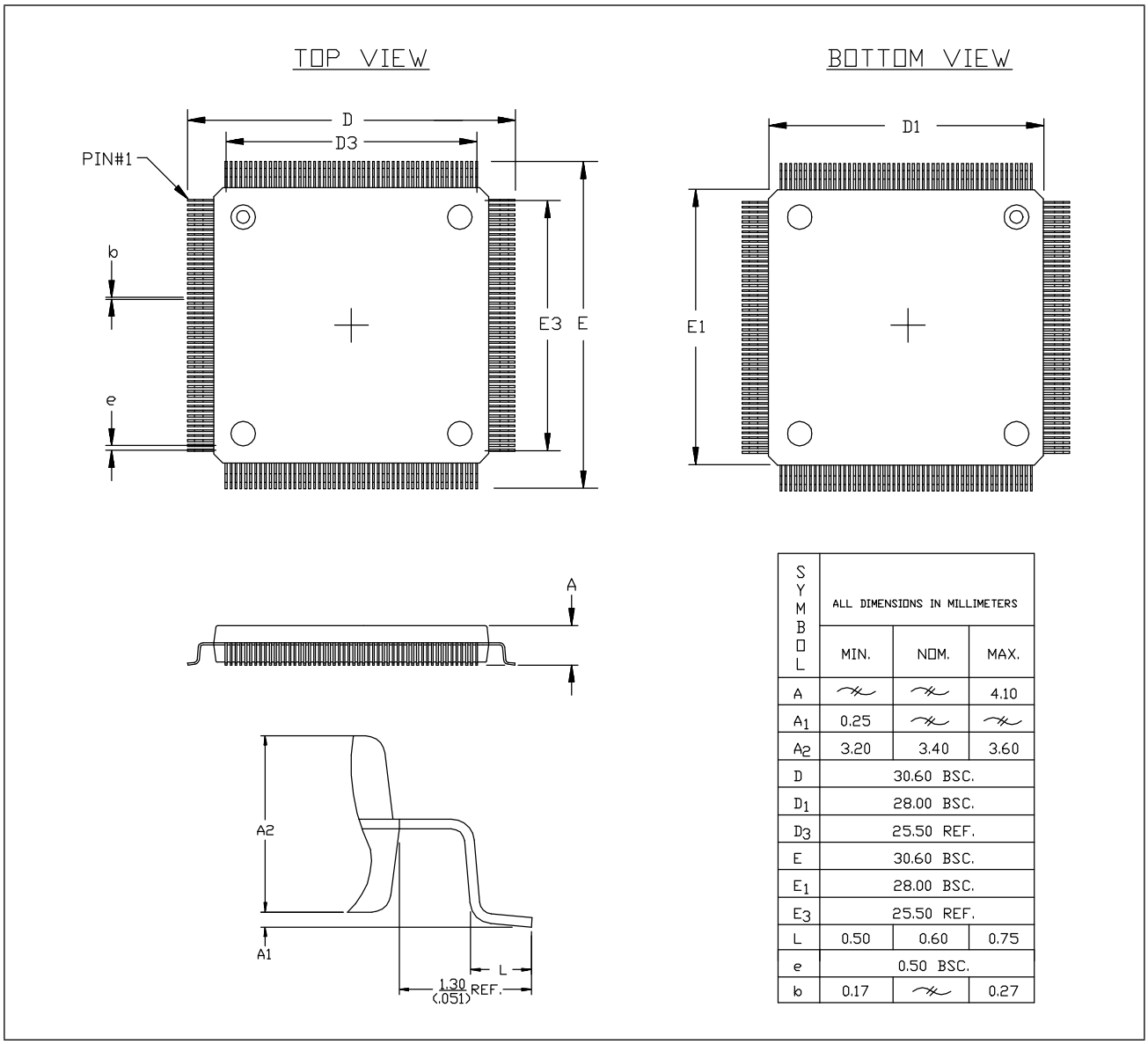
Bt2166 Packaging Specifications

Table 199 shows the thermal resistance for the package shown in Figure 79. Figure 79 shows model Bt2166AHF: Metric Quad Flat Pack (MQFP).

Table 199. Package Thermal Resistance+

Package	Airflow (Linear Feet per Minute)					Units
	0	50	100	200	400	
208-pin MQFP with heat spreader	21	19	17	16	14	˚C/W
208-pin MQFP	28	26	24	23	20	˚C/W

Figure 79. Bt2166AHF Packaging Diagram



SYMBOL	ALL DIMENSIONS IN MILLIMETERS		
	MIN.	NOM.	MAX.
A			4.10
A_1	0.25		
A_2	3.20	3.40	3.60
D		30.60 BSC.	
D_1		28.00 BSC.	
D_3		25.50 REF.	
E		30.60 BSC.	
E_1		28.00 BSC.	
E_3		25.50 REF.	
L	0.50	0.60	0.75
e		0.50 BSC.	
b	0.17		0.27

Brooktree®

Brooktree Division
Rockwell Semiconductor Systems, Inc.
9868 Scranton Road
San Diego, CA 92121-3707
(619) 452-7580
1(800) 2-BT-APPS
FAX: (619) 452-1249
Internet: apps@brooktree.com
L2166_*A*