VS_VP

VS25203

Hardware Reference Manual

Revision 1.03

VLSI Solution

I. Revision History

From Revision 1.02 to Revision 1.03.

1. Trademark page updated.

II. Table of Contents

1. Introduction

This document gives an overview description of the architecture of VLSI Solution's **VS_VP** VS25203 3D graphics accelerator. The document contains 18 chapters.

Chapter 1 is introduction of this document.
Chapter 2 gives some general information about the architecture of VS25203.
Chapter 3 describes the interfacing registers, graphics memory and the PCI bus.
Chapter 4 presents the Geometry Processor.
Chapter 5 describes the Primitive Processor
Chapter 6 describes the Pixel Processor.
Chapter 7 gives some information on how to program the core and video clocks of VS25203.
Chapter 8 describes the VGA -block.
Chapter 9 describes how to calculate screen parameters for video registers, using a screen size of 640×480 pixels as an example.
Chapter 10 gives information about TV-output and its usage.
Chapter 11 provides information about Video Capture unit.
Chapter 12 describes the block transfer unit and its operation.
Chapter 13 lists the main features of the DAC.
Chapter 14 provides some board level application information.
Chapter 15 describes the pin layout and defines the pin signals for 3.3 V system.
Chapter 16 describes the electrical characteristics of the device for 3.3 V system.
Chapter 17 lists a few references for further readings.
Chapter 18 presents the index list.

2. Architecture

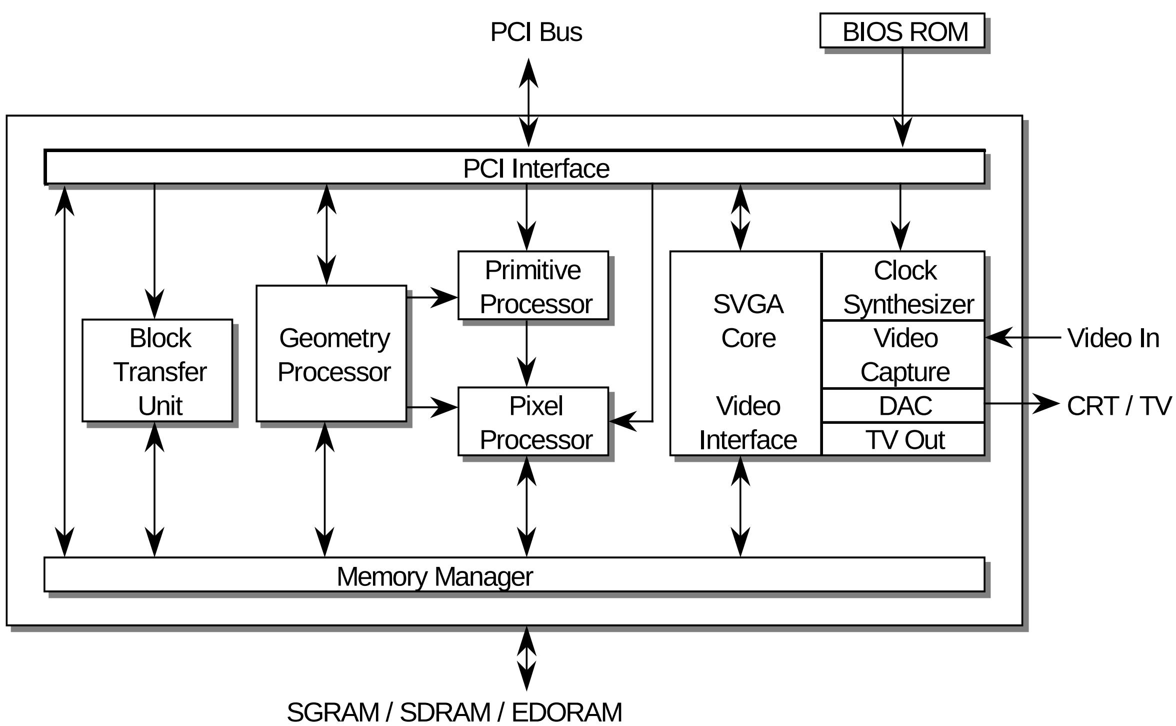
2.1 Overview

VS25203 is a member of VLSI Solution's **VS_VP** family of highly integrated, programmable, and high performance 3D graphics accelerators. It is designed for the acceleration of games, 3D applications and user interfaces. It offers full compatibility with the emerging 3D standards including Direct3D™ for Windows 95™. And OpenGL™ for Windows NT™.

VS25203 integrates on a single chip the Primitive Processor, Pixel Processor, Geometry Processor, PCI bus master interface, memory management unit, video refresh logic, VGA, Block Transfer Unit, clock synthesizer and true-color DAC.

The features of VS25203 form a solid base, on which support for different 3D APIs can be built easily. A full 3D graphics system requires only memory, in addition to VS25203. A low cost system can be constructed with two 256K × 32 SGRAMs.

VS25203 is a single chip implementation of the 3D rendering pipeline. The primitives are first rasterized in the Primitive Processor. The resulting individual pixels are sent to the Pixel Processor, which writes these on the screen through the memory management unit. The chip also contains a PCI interface for communicating with the host processor.



2.2 Key Features

Rendering
Programmable pixel pipeline
User specifiable blending
Perspective correct true-color Gouraud lighting
Perspective correct transparency
Perspective correct fog
Perspective correct texture mapping
Multiple simultaneous textures
Environment mapping
Bump mapping
Stencil operations
Logic operations
Specular highlights
Properly handled lighted textures
Rasterized screen door transparency
Destination blending for transparency effects
Fog and depth cue with vertex level control

Textures
Texture magnification filtering with point sampling or bilinear filtering
Texture minification filtering with point sampling or MIP mapping
Trilinear filtering possible
Texture sizes from 16×16 pixels to 2048×2048 pixels (non-square supported)
Amount of texture maps limited only by available memory
Texture can be looped or have a solid color border
RGB map formats: 32-bit RGBA (32-bit frame buffer) and 16-bit RGB and 16-bit RGBA
YUV map formats: 8-bit alpha + 24-bit VYU (YUV 4:4:4); 32-bit YVYU (YUV 4:2:2)
Indexed map formats: 8-bit and 4-bit
Indexed maps have an internal 256 color 24-bit palette (RGBA 6:6:6:6)
Full blending and filtering possible with indexed maps
Real time texture paging and animation
Rendering directly to texture maps possible

Memory
2-32 Mbytes of SDRAM, SGRAM or EDO DRAM supported
Memory bus width 64 bits or 32 bits
Memory bandwidth up to 800 MBytes/sec with 64-bit bus
Unified memory architecture for frame buffer and textures

Framebuffer
Virtual resolutions up to 2048 × 2048 pixels
24-bit or 16-bit color (dithering supported)
24-bit or 16-bit depth buffer
1-bit stencil mask
Support for double and triple buffering and stereo imaging

SVGA and Video Refresh
100% IBM compatible VGA unit
Support all the existing modes

Display resolutions from 320 × 200 to 1600 × 1200 pixels
Internal video refresh logic
Internal programmable clock generator (up to 200MHz)
Internal true-color DAC (up to 200MHz pixel clock)
TV-output with configurable flicker filter

Geometry Processor
3-issue VLIW architecture
32-bit fixed point vector datapath
Block floating point support
Hardware division unit
Integrated 3 × 128 words 2-port SRAM data memory
4-way set associative instruction cache of 4 × 128 word blocks

Video Capture Unit
8-bit 4:2:2 YUV ITU-R BT.656-3

Block Transfer Unit
Memory copy and fill operations
Supports basic bit copy operations

Physical Characteristics
304-pin BGA packaging
200 MHz operation
I/O interface at 5/3.3V

Compatibility
Drivers for Microsoft Windows 95
Drivers for Microsoft Windows NT 4.0
Drivers for DirectDraw and Direct3D (immediate mode)
Drivers for OpenGL for Windows NT

Estimated Peak Performance (with 300MHz Pentium II)
1,000,000 shaded, 16bpp textured 25 pixel triangles per second
1,000,000 shaded, 16bpp textured, Z-buffered 25 pixel triangles per second
Bilinear pixel fill rate of 60,000,000 pixels per second

2.3 Geometry Processor

The Geometry Processor can be used to accelerate any calculations related to the data stored in the external graphics memory. Normally the task of the Geometry Processor is to process a data stream and calculate values to the Primitive Processor registers.

The Geometry Processor is based on 3-issue VLIW architecture with a packed 32-bit instruction word. It has three Arithmetic Units and additional units for hardware division, logic operations and other tasks. The arithmetic units have three cycle pipelines. As usual for a VLIW processor, the architecture and pipeline in the Geometry Processor are visible to the programmer, and one must take into account all the pipeline effects. This enables one to write maximally efficient code, but requires more care in programming.

The processor also has three integrated data memories, so that there is no need to use the external graphics memory during the calculations. The program that controls the Geometry Processor is given by the user and is stored in the external graphics memory. The program is cached into an on-chip instruction cache.

2.4 Primitive Processor

The Primitive Processor calculates the individual pixels which form each primitive and forwards them to the Pixel Processor. Primitives can be triangles, lines or 2D regions. They are described with their edges and shading information. All these are stored to the Primitive Processor registers by the host processor.

The Primitive Processor determines all pixels that are inside the primitive and calculates the different properties for them. The pixels have 8 properties which are all interpolated in parallel. They can be used as color (R,G,B), transparency, fog intensity, specular intensity, primary texture coordinates (U,V) and secondary texture coordinates (U2,V2).

What really affects the resulting image quality is the accuracy with which this process is carried out. Perspective correction is needed for realistic results and there are no sacrifices in this area. All properties including color, transparency, and fog - not just the texture as in most other 3D systems - are interpolated with full perspective correction without performance restrictions. This guarantees that lighting and texture will fit together seamlessly.

2.5 Pixel Processor

The Pixel Processor performs visibility checking (using the Z buffer), texture data fetching and transparency and color blending. It receives as input a list of pixels along with their properties from the Primitive Processor and writes the resulting colors as an output to the local framebuffer memory.

All calculations in the pixel pipeline are performed with true-color accuracy (24-bit color, 8-bit transparency). The processor can be used for overlay surface color, which can be combined from multiple textures, diffuse light intensity and specular light intensity. The lights can also be independently colored without a performance loss. In addition, special effects including fog, environment mapping and bump mapping are supported in hardware.

In order to maximize image quality without maximizing memory usage, a wide variety of texturing methods are supported. The textures can range from 16×16 pixels with 4 bit indexed color, right up to 2048×2048 pixels, and can be of full 32 bit true-color quality. For indexed textures, the Pixel Processor has an internal 256 color RGBA palette.

The quality can be further increased with texture filtering, as both MIP-mapping and bilinear filtering are directly supported (also in indexed modes). Because of programmability, trilinear filtering is also possible.

There are many different ways in how the pixel properties can be used to derive the final pixel color. So as not to impose any strict limits, the Pixel Processor is fully programmable. The sequence of texture, blending and control operations can be specified. In addition, the pipeline works in parallel with multiple pixels; this guarantees performance even for more complex shading settings.

The resulting pixel color can also be combined with the previous color on the screen. It makes transparency effects possible. In case the display format used is 16-bit color, it is also possible to dither the output (4×4 ordered dither) for better quality.

While programmers can do any kinds of effects they want in software, they are often limited by hardware which lowers their choices considerably so that the needed speed boost can be obtained. With **VS_VP** VS25203, it has been an important design criterion to make the hardware as configurable as possible. As a result, it is possible to generate effects that were only possible previously with advanced software rendering packages, and still do them all in real-time.

2.6 PCI Interface

The **VS_VP** VS25203 can be directly connected to a PCI bus without any extra logic. The PCI interface provides the host with linear access to the frame buffer and registers (which are memory mapped). In addition, bus mastering is supported so that textures and individual triangles can be read from the main memory without host processor overhead.

2.7 Memory Management Unit

All memory is accessed through the Memory Management Unit. This has the advantage that different types of data such as textures and display data can all share the same memory; memory usage can thus be optimized separately for each application. Games, for example, require a lot of texture memory, whereas CAD requires a lot of resolution.
For maximum performance, the memory interface supports SDRAM memory, which can achieve a 800MB/s transfer rate (using 64-bit bus). In addition, a reduced bus width (32 bits) is possible if less memory is desired. It is also possible to use SGRAM, which makes a 2Mbyte configuration with good performance possible. Finally, for a low cost solution it is possible to use EDO DRAM.

The memory management unit also generates commands, which initiate the self-refresh cycles to the SDRAM, SGRAM or EDO DRAM.

2.8 SVGA and Video Refresh

VS252 SVGA Core is 100% compatible with original IBM VGA implementation. It takes use of PCI interface to provide optimizations for standard VGA 256-color mode and extended 8 bit graphics modes. It extends the VGA CRTC counters for larger display modes, and provides linear frame buffer and 64 bit sequencer model.

The video refresh logic supports 16-bit hi-color and 24-bit true-color display formats with resolutions from 320×200 to 1600×1200. With the programmable clock generator, refresh rates can be adjusted without limitations.

TV-output is also supported with configurable flicker filter.

2.9 Video Capture Unit

The independent video capture unit reads 4:2:2 YUV in the ITU-R BT.656-3 format and stores it into the memory for further use.

2.10 Block Transfer Unit

The internal Block Transfer Unit perfoms area copy and fill operation as well as bit copy operations.

2.11 Internal Clocks

VS25203 contains two phase-locked-loop (PLL) frequency synthesizers, one for the video clock and one for the processor.

2.12 Internal VideoDAC

VS25203 contains an internal triple 8-bit VideoDAC, which has a maximum operation frequency of 200 MHz. Internal VideoDAC

2.13 External VideoDAC

It is also possible to use external VideoDAC with the following features:

Triple 8-bit D/A converters
TTL compatible inputs
construction optionally +5 V or +3.3 V.

VS25203 provides pins for an external VideoDAC: 8bit data bus for each color (RGB) and all essential synchronization and blanking signals; see page 233.

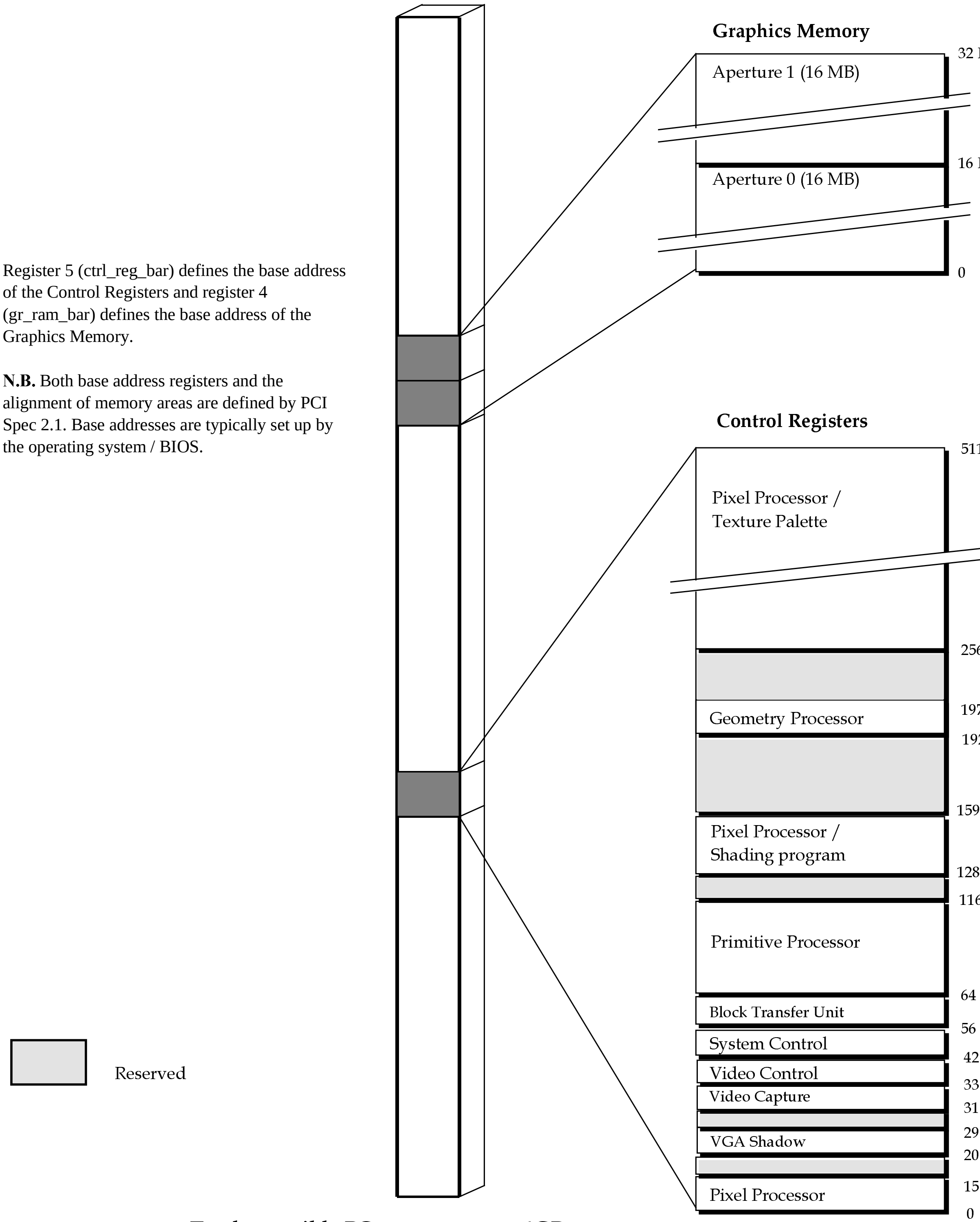
2.14 External BIOS ROM Interface

The eight-bit BIOS ROM contains power-on initialization and mode setup routines. PCI configuration power-on initialization data are also located in BIOS ROM. BIOS ROM shares pins with the external VideoDAC.

2.15 Register Map

Register 5 (ctrl_reg_bar) defines the base address of the Control Registers and register 4 (gr_ram_bar) defines the base address of the Graphics Memory.

N.B. Both base address registers and the alignment of memory areas are defined by PCI Spec 2.1. Base addresses are typically set up by the operating system / BIOS.



Total accessible PC memory space, 4GB.

2.16 Summary of Registers

	Register address	Offset	Register name
Pixel Processor	1	0004h	coef_reg0
	2	0008h	coef_reg1
	3	000Ch	coef_reg2
	4	0010h	coef_reg3
	5	0014h	atex_conf1
	6	0018h	atex_conf2
	7	001Ch	btex_conf1
	8	0020h	btex_conf2
	9	0024h	base_addr
	10	0028h	dither
	11	002Ch	modulation
	12	0030h	ppu_mode
	13	0034h	frame_mode
	14	0038h	ppu_code_start
	15	003Ch	palette_base
VGA Shadow	20 - 29		VGA shadow registers
Video Capture	31	007Ch	capt_base_conf
	32	0080h	capt_w_h
Video Refresh	33	0084h	video_width_height
	34	0088h	screen_width_height
	35	008Ch	video_vblank
	36	0090h	video_hblank
	37	0094h	video_vsync
	38	0098h	video_hsync
	39	009Ch	video_base_conf
	40	00A0h	video_bit_config
	41	00A4h	reserved
System control	42	00A8h	ma_cmd_addr
	43	00ACh	master_state
	44	00B0h	ma_int_addr
	45	00B4h	ma_ext_addr
	46	00B8h	reserved
	47	00BCh	reserved
	48	00C0h	status
	49	00C4h	ref_reg
	50	00C8h	debug_reg
	51	00CCh	io_reg
	52	00D0h	ext_io_reg
	53	00D4h	ext_io_reg2
	54	00D8h	mem_apt0_cfg
	55	00DCh	mem_apt1_cfg
Block Transfer Unit	56	00E0h	blt_src_strd
	57	00E4h	blt_tgt_strd
	58	00E8h	blt_fg_color
	59	00ECh	blt_bg_color
	60	00F0h	blt_params
	61	00F4h	blt_src_addr
	62	00F8h	blt_tgt_addr
	63	00FCh	blt_size

	Register address	Offset	Register name
Primitive Processor	64	0100h	cr_init
	65	0104h	cr_dy
	66	0108h	cr_dx
	67	010Ch	cg_init
	68	0110h	cg_dy
	69	0114h	cg_dx
	70	0118h	cb_init
	71	011Ch	cb_dy
	72	0120h	cb_dx
	73	0124h	ct_init
	74	0128h	ct_dy
	75	012Ch	ct_dx
	76	0130h	atu_init
	77	0134h	atu_dy
	78	0138h	atu_dx
	79	013Ch	atv_init
	80	0140h	atv_dy
	81	0144h	atv_dx
	82	0148h	btu_init
	83	014Ch	btu_dy
	84	0150h	btu_dx
	85	0154h	btv_init
	86	0158h	btv_dy
	87	015Ch	btv_dx
	88	0160h	z_shr
	89	0164h	z_init
	90	0168h	z_dy
	91	016Ch	z_dx
	92	0170h	edge_order
	93	0174h	edge0_init
	94	0178h	edge0_dx
	95	017Ch	edge0_dy
	96	0180h	edge1_init
	97	0184h	edge1_dx
	98	0188h	edge1_dy
	99	018Ch	edge2_init
	100	0190h	edge2_dx
	101	0194h	edge2_dy
	102	0198h	grid_reg
	103	019Ch	p_init
	104	01A0h	p_dy
	105	01A4h	p_dx
	106	01A8h	x_init
	107	01ACh	y_init
	108	01B0h	y_end
	109	01B4h	raster_ext
Pixel Processor	128 -159		Code
	256 - 511		Texture Palette
Geometry Processor	192 -197		Geometry Processor registers

3. Interfaces

3.1 Accessing Internal Registers

The VS25203 internal register ranges are available for access through the PCI interface. The registers are mapped to the PCI memory starting from the memory location specified by the `ctrl_reg_bar` register in PCI configuration space, see page 27.

Because all the registers are 32bit registers, the index of each register must be multiplied by four to get the relative memory address. For example if the PCI BIOS has configured the VS25203 `ctrl_reg_bar` register to the value E0000000h then the `cr_init` (64) register (described in page 106) is mapped to the address E0000000h + 64 × 4 = E0000100h.

Range	Function
0-15	Pixel Processor / General
20-29	VGA Shadow
31-32	Video Capture
33-41	Video Refresh
42-55	System Control
56-63	Block Transfer Unit
64-109	Primitive Processor
128-159	Pixel Processor / Code
192-197	Geometry Processor
256-511	Pixel Processor / Texture Palette

Register ranges not covered or not mentioned above should be considered as *reserved* and not used.

3.2 Memory Apertures

The PCI interface maps the graphics card memory to the PCI bus. Different translations including linear mode and raw frame buffer mode are available.

In order to implement the interfaces to other PCI multimedia devices VS25203 provides two simultaneous apertures to the memory (as suggested by the PCI Multimedia Design Guide revision 1.0). It is possible to configure the apertures to provide different views for the memory.

Linear mode is the similar to standard VESA VGA memory organizations however, when used as a VESA 8 bit linear frame buffer the memory should be accessed in raw mode. This applies to the 8 bit modes only.

Raw mode is VS25203's internal mode for storing data. Raw mode is different from linear mode in that data in raw mode is organized in small 2D regions (e.g. 64 × 16 × 16 bits for hi-color, or 64 × 8 × 32 bits for true-color) so as to take advantage of fast accesses to active rows in a SDRAM to reduce page misses; an active SDRAM row is typically 2048 bytes.

For hi-color (16-bit) pixels, the pixel block is 64 pixels in the horizontal direction, and 16 pixels in the vertical direction, and each pixel is 16 bits. But for 32-bit pixels (i.e. 24-bit color + 8-bit transparency) and depending on the memory configuration used, it is typically 64 pixels wide by 8 pixels high to make it fit into 64×8 at 4 bytes per pixel to give 2048 bytes.

Note that the memory image of the screen is allocated according to integral multiples of the pixel block size. Depending on the screen resolution chosen, the visible area on the screen may be smaller.

For example, to be able to use frame buffer as a texture map, all data must be in the same mode to keep the operations straightforward. In order to see the internal frame buffer format through the PCI bus, the aperture can be configured to convert from raw to linear mode. And to do this, the aperture needs the information of the number of 2048-byte blocks which the screen has in the vertical direction, so that the correct amount of raw mode memory can be skipped when moving from one pixel position to the next neighboring one on the screen. The `apt_width` field of the `mem_apt0_cfg` and `mem_apt1_cfg` registers defines the resulting linear mode row length.

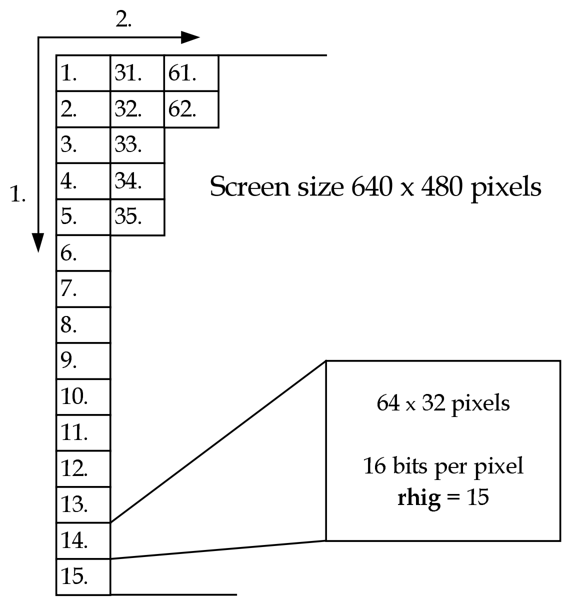
The graphics memory is accessible through a 32MB memory window which is located as specified by the `gr_ram_bar` register, page 26. The uppermost bit of the address in the memory window selects the memory aperture which is used.

3.2.1 Linear Mode

If the aperture selected is in the linear frame buffer mode then the address is first split to x-coordinate and y-coordinate values for the frame buffer (or texture map) memory. The address splitting is done based on the `aperture_width` register value. Address is first zero-level-compensated with `gr_ram_bar` register.

3.2.2 Raw Mode

If the aperture selected is in raw mode then the address is used to access the local graphics memory. In order to support memory configurations larger than 16MB the value of `apt_addr` field $\times$ 2048 can be used to specify the start address of the graphics memory within the aperture; see registers 54 and 55 on page 44.



In raw mode, the screen is split on 64-pixel-column × 32-pixel-row pixel blocks. Blocks are arranged on the screen as shown. The second block is situated below the first one and so on. Depending on screen size, there are different number of blocks on the screen. In the 640 × 480 example, there are 10-block-columns × 15-block-rows = 150 blocks. Within a block, pixels are arranged so that the second pixel is on the right hand side of the first one and the 65th pixel is below the first one. 16- and 32-bit data is arranged in memory as follows:

Pixel 0				Pixel1			
B_0	G_0	R_0	T_0	B_1	G_1	R_1	T_1

byte in memory	0	1	2	3	4	5	6	7
32 bit mode	B_0	G_0	B_1	G_1	R_0	T_0	R_1	T_1
16 bit mode	RGB_0		RGB_1		RGB_2		RGB_3	

3.3 PCI Bus

3.3.1 Overview

VS25203 has a PCI bus interface which conforms to the PCI local bus specification Revision 2.1, see page 246.

The PCI interface of VS25203 contains two base address registers. One register is used to map the internal registers and user controllable internal memories of VS25203 to the PCI bus (register 5, page 27). And the other is used for mapping the graphics memory to the PCI bus (register 4, page 26). See also the page 13. These base addresses are initialized by the PCI BIOS (or the operating system) during boot-up.

PCI bus interface feature summary:

- Fast DEVSEL# assertion

- When acting as a PCI target for write operations, the target does not typically generate wait states. There are no wait states for register writes, but there are in some cases of memory operations.

- Memory on the graphics card is accessible using two independent apertures as suggested by PCI Multimedia Design Guide revision 1.0.

- Memory apertures do not perform color space conversions; YUV conversion is done within the Pixel Processor.

- VS25203 supports the conversions required for the full interoperability level in PCI Multimedia Design Guide rev. 1.0.

3.3.2 Bus Master Functions and Commands

The VS25203 can perform the following operations independently as a PCI master:

- read sequences of triangle parameters for the rendering engine
- upload data for textures and other images to the graphics memory
- synchronize the PCI bus master operation to the operation of the rendering engine.

PCI bus mastering is used as follows:

- initialize bus master command stream to system memory (This refers to host main memory. It is NOT recommended that the graphics board memory be used for this purpose)
- load the start address of the bus master command stream to PCI master command address (42) register (`ma_cmd_addr`)
- write a non zero value to the highest byte of PCI master state (43) register (`master_state`)
- use PCI master state (43) register (`master_state`) to observe when the PCI bus master operation is completed.

Notice that all the addresses referred to in PCI bus mastering are physical addresses. Under virtual memory operating systems like UNIX, Windows 95 and Windows NT, programs typically use virtual memory addresses which must be mapped to physical addresses. Special care must also be given to continuously allocated virtual memory which may not correspond to a continuous block of physical memory.

VS25203 provides the following stream commands as a PCI master. Note that the example sections contain only minimal parts of the whole stream program.

direct command **PCI master stream** **opcode 01h**

Description: Loading values to VS25203 internal registers. Previously known as regload.

Command format: `01aaaabb` `aaaa` is the register_address. `bb` is register_count. Moves next `bb` 32-bit values of the bus master stream to registers of VS25203, beginning with register `aaaa`.

direct command																															
31	30	29	28	27	26	25	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
01h								register_address																register_count							
register_value_1																															
register_value_2																															
register_value_3																															
…																															
register_value_n																															

Special: It is important to synchronize register loading between triangles with jump or wait commands.

Example:

.
.
.
0100402Dh
value for register 64
value for register 65
.
.
.
value for register 108

The direct command (01h) loads 45 (2Dh) register values to the registers of VS25203, beginning at cr_init (64 = 40h) register. 32-bit register values are located after the command line. Note that in this example the last register value is for the y_end (108) register.

Description: Jump conditionally to the specified address when the specified condition is met. This is used for transferring the point where the stream is interpreted to another area in memory. It is also used for synchronizing the PCI master operation with the internal state of the VS25203, for example, to wait until the previous triangle is rendered. It can also be used to generate an interrupt request.

Command format:

`02iXaabb` Wait until the condition is true and then jump to address
`jjjjjjjj` `jjjjjjjj`. The condition is given by:

(status[7:0] xor flag_xor) and flag_mask == 00h)
where status is the status (48) register.

jump command																															
31	30	29	28	27	26	25	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
02h								i	reserved							flag_mask								flag_xor							
jump_address																															

Special: If the interrupt bit i (bit 23) is one, an interrupt is generated when the jump command has been read from the bus master stream. In other words, interrupt is generated when the bus master encounters the jump command, not when the condition is true. X is a reserved field and should be treated as zero.
The execution is halted at the jump command until the condition is true. After the condition is true, the execution continues at the address given by jump_address. Notice that both words of this command are read by the PCI mastering logic before the waiting for the flag values start.

Example:

.
.
.
02800303h
00020000h
.
.
.

The stream program waits until the condition: 03h and (03 xor status[7:0]) == 0 is true and then jumps to address 20000h which is specified right after the command line. Note that this address is an absolute physical address. The program also causes an interrupt since the i-field (bit 23) is one.

See also: Status (48) register, page 39.

Description: Transfers data from main memory address or from another memory mapped PCI device to VS25203-based card. It is used for copying memory data from the host main memory to the graphics board memory. This can be used, for example, for uploading textures. Notice that the PCI interface aperture mapping functions can be utilized with the read command. Also, the destination address is an address on VS25203; it is not a PCI bus physical address.

Command format:

```
03aaaaaa
bbbbbbbb
cccccccc
```

Reads aaaaaa (read_count) 32-bit words (not bytes) from PCI memory beginning with external source address cccccccc and writes them to memory beginning with internal destination address bbbbbbbb.

read command																															
31	30	29	28	27	26	25	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
03h								read_count																							
internal_address																															
external_address																															

Special: It should be noted that bus mastering is not an especially effective way to move data between locations on the local graphics memory. It is targeted for moving data from the host CPU main memory or from other PCI boards/devices to VS25203 memory.

Example:

.
.
.
03010000h
000FF000h
0B100000h
.
.
.

The program transfers a 256KB data block. Destination address is FF000h and source address is B100000h.

wait command | **PCI master stream** | **opcode 04h**

Description: Similar to the jump command. But instead of jumping, it continues to read the current command stream without a jump. The wait command is used for synchronizing the PCI master operation with the internal state of the VS25203, for example, to wait until the previous triangle is rendered. It can also be used to generate an interrupt request.

Command format: `04iXaabb` Wait until the condition is true and then continues the stream processing from the next instruction. The condition is given by: (status[7:0] xor flag_xor) and flag_mask == 00h) where status is the status (48) register.

wait command																															
31	30	29	28	27	26	25	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
04h								i	reserved							flag_mask								flag_xor							

Special: If the interrupt bit i (bit 23) is one, an interrupt is generated when the wait command has been read from the bus master stream. In other words, interrupt is generated when the bus master encounters the wait command, not when the condition is true. X is a reserved field and should be treated as zero.
The execution is halted at the wait command until the condition is true. After the condition is true, the execution continues at the stream location following the address of the wait command.

halt command | **PCI master stream** | **opcode 80h**

Description: Halts PCI bus master operation. It is typically used as the last command in the PCI master command stream.

Command format: 80000000h

halt command																															
31	30	29	28	27	26	25	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
80h								reserved																							

Example:

.
.
.
80000000h
.
.
.

When the stream program reaches the halt command, the PCI bus master halts its operation and its stream execution.

3.3.3 Bus Master Programming Guidelines

Some guidelines for using PCI bus mastering are as follows:

- Use the direct command to load a single register or a group of registers.
- Use jump or wait to synchronize and control program flow within the DMA buffer.
- `wait` is a command similar to `jump`. It continue to run from the next address when the status is met, so it does not contain the jump address.
- Use the interrupt and the halt command to synchronize with the application
- Make sure the interrupt and the halt command are put in place before starting DMA.
- The interrut handler should check that DMA transfer is completed by polling MSB byte of register 43.

Example:

<u>Address</u>	<u>Data</u>	<u>Comment</u>
0x12345678	0x0100402D	load 45 registers starting from register 64 (0x40)
0x1234567C	0x00000000	value for register 64
…		
0x12345730	0x04000F0F	wait until status is 0x0F, then continue
0x12345734	0x04800000	generate an interrupt
0x12345738	0x80000000	halt

To start a DMA,
Write physical address of start of DMA command stream to register 42
Write 0xFF000000 to register 43 to start DMA
An interrupt should be generated when the current DMA command stream is executed

3.3.4 PCI Configuration Space Registers

Register Number	Address Offset	Register name	Description
0	0000h	id_reg	ID register
1	0004h	status_cmd	Status command register
2	0008h	class_rev	Class revision register
3	000Ch	cfg0	Configuration 0 register
4	0010h	gr_ram_bar	Graphics memory base address register
5	0014h	ctrl_reg_bar	Control register base address register
11	002Ch	sub_id	Subsystem ID register
12	0030h	exp_rom_bar	Expansion ROM base address register
15	003Ch	cfg1	Configuration 1 register
16	0040h	core_clk_cfg	Core clock configuration register
17	0044h	mem_cfg	Memory configuration register
18	0048h	video_clk_cfg	Video clock configuration register
19	004Ch	reg_acc_addr	Register access address register
20	0050h	reg_acc_data	Register access data register
21	0054h	feat_reg	Feature register

id_reg	register 0	offset 0000h

Format

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

Fields

Field	Bits	Description
vendor_id	15:0	Manufacturer of the device
device_id	31:16	Device

ID Register contains information about the manufacturer and the device
vendor_id
This field is hardwired to 1292h.
device_id
This field is hardwired to FC04h to identify the device type.

status_cmd	register 1	offset 0004h

Format

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

Fields

Field	Bits	Description
command	15:0	Refer to the PCI local bus rev 2.1
status	31:16	Refer to the PCI local bus rev 2.1

Status Command Register bit 1 enables or disables VS25203 on PCI bus:
0 disable
1 enable
Status Command Register bit 2 (bus master control) enables or disables PCI bus master function:
0 disable
1 enable

class_rev	register 2	offset 0008h

Format

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

Fields

Field	Bits	Description
class_code	31:8	Generic function of the device
revision_id	7:0	Revision

Class Revision Register contains two fields:

class_code
Identifies the generic function of the device; VS25203 is hardwired to 03000000h as a display controller.
revision_id
Device-specific revision identifier.

cfg0	register 3	offset 000Ch

Format

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

Fields

Field	Bits	Description
cache_ls	7:0	Cache line size
lat_tim	15:8	Latency timer
hdr_type	23:16	Header type

Configuration 0 Register contains the following fields:
cache_ls
Cache line size. Field specifies the cache line size in 32-bit units
lat_tim
Latency timer. The value of the latency timer for this bus master in PCI bus clock units.
hdr_type
Header type. Identifies the layout of bytes 0010h to 003Fh, and also whether or not the device contains multiple functions. Hardwired to 0.

gr_ram_bar	register 4	offset 0010h

Format

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

Fields

Field	Bits	Description
gr_ram_bar	31:0	Graphics memory base address

Graphics Memory Base Address Register specifies the graphics memory base address (Aperture 0 and Aperture 1).
See also page 13.

ctrl_reg_bar	register 5	offset 0014h

Format

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

Fields

Field	Bits	Description
ctrl_bar	31:0	Control register base address

Control Register Base Address Register. Specifies the base address of the control register. See also page 13.

sub_id	register 11	offset 002Ch

Format

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

Fields

Field	Bits	Description
sub_ven_id	15:0	Subsystem vendor ID
sub_id	31:16	Subsystem ID

Sub ID Register contains auxiliary information about the manufacturer and the device.

sub_ven_id

This field is read from external ROM during bootup.

subsystem vendor id LSB = ROM(LAST_ADDR-7)
subsystem vendor id MSB = ROM(LAST_ADDR-6)

sub_id

This field is read from external ROM during bootup.

subsystem id LSB = ROM(LAST_ADDR-5)
subsystem id MSB = ROM(LAST_ADDR-4)

exp_rom_bar	register 12	offset 0030h

Format

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

Fields

Field	Bits	Description
rom_bar	31:16	Expansion ROM base address

Expansion ROM Base Address Register has the following field:

rom_bar

Contains base address information for expansion ROM.

cfg1	register 15	offset 003Ch

Format

31–24	23–16
max_lat	**min_gnt**
int_pin	**int_line**
15–8	7–0

Fields

Field	Bits	Description
int_line	7:0	Interrupt line
int_pin	15:8	Interrupt pin
min_gnt	23:16	Minimum grant
max_lat	31:24	Maximum latency

Configuration 1 Register contains the following fields:
int_line
Interrupt line. Contains interrupt line routing information. This field is typically set by the PC motherboard BIOS.
int_pin
Interrupt pin. This field is hardwired to a value 01h to specify that INTA# is the interrupt pin used.
min_gnt
Minimum grant value. Specifies the length of the device's burst period in 250nsec units. Hardwired to 2.
max_lat
Maximum latency value. Specifies how often the device needs to gain access to the bus in 250nsec units. Hardwired to 0 to indicate that VS25203 does not have hard latency requirements.

core_clk_cfg	register 16	offset 0040h

Format

31 / 15	30–16 / 14	13–7	6–0
no			
r_coef (15–14)		**m_coef**	**n_coef**

Fields

Field	Bits	Description
no	31	non-overlap mode
n_coef	6:0	N coefficient for core clock
m_coef	13:7	M coefficient for core clock
r_coef	15:14	R coefficient for core clock

Core Clock Configuration Register.

Controls the internal clock buffer non-overlap time for debugging purposes. 0 for shorter non-overlap, 1 for longer non-overlap. Typical value for non-overlap is 0.

The core clock frequency can be calculated from the formula:

$$F_{OUT} = \frac{m\_coef + 2}{(n\_coef + 2) \times 2^{r\_coef}} \times F_{OSC}$$

where:

n_coef, m_coef, r_coef = coefficients
F_{OSC} = quartz crystal or external clock (MHz)

For additional information see page 152.
After boot-up register contains value of 8000BE87h (Fout = 50 MHz).

Caution: Unsuitable clock frequency parameters may cause permanent damage to the device.

mem_cfg	register 17	offset 0044h

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
dr	sg		2x						ref_rate			depth		wi	ch
mode_reg															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	Bits	Description
dr	31	mm_dram
sg	30	mm_sgram
2x	28	2x memory mode
ref_rate	22:20	refresh rate
depth	19:18	mm_1_2_4_depth
wi	17	mm_16_32_width
ch	16	mm_8_16_chips
mode_reg	15:0	SDRAM mode reg. value/256 & DRAM param.

Memory Configuration Register descriptions:

mm_dram
0 if not using DRAM
1 if using DRAM
The default memory type (both bits 30 and 31 = '0') is SDRAM memory.

mm_sgram
0 if not using SGRAM
1 if using SGRAM

2× memory mode
0 normal memory mode
1 2× memory mode

refresh_rate
000 default memory refresh rate
001 3× rate
010 5× rate

mm_1_2_4_depth
memory "depth" parameter
00 if one level of memory circuits is used
01 if two levels of memory circuits is used
10 if four levels of memory circuits is used
if DRAM memories are used then only supported memory depth is 1,
this field is used to indicate the size of the memory circuits
00 for 256K×16 DRAM
01 for 1M×16 DRAM and also for 4M×16 DRAM
(the 4M×16 DRAM must be of type which uses same amount of CAS bits as typical 1M×16DRAM)

mm_16_32_width
0 if 16 bit wide memory buses are used (SDRAM only parameter)
1 if 32 bit wide memory buses are used
value for this field is 1 for SGRAM boards

mm_8_16_chips
(SDRAM only parameter)
0 if 8 bit wide memory circuits are used
1 if 16 bit wide memory circuits are used
value for this field is 1 for SGRAM boards

mode_reg
SDRAM (SGRAM) mode register / DRAM timing

The mode_reg is shared between two uses:
1) on DRAM configurations to provide timing parameters for the memory accesses.
2) for SDRAM configuration to provide the value which is used in SDRAM mode register configuration.

1) DRAM parameters

The DRAM timing parameters are derived from the mode_reg bits as follows:
(the names of the timing parameters are intended to correspond to the timing parameters in typical DRAM datasheets)

All timings are relative to the core clock frequency

mode_reg(0) T_AS
address set up time (address setup before RAS or CAS)
0 = 0 cycles
1 = 1 cycles

mode_reg(1) T_CAS
CAS# pulse width
0 = 1 cycles
1 = 2 cycles

mode_reg(2) T_CP
CAS precharge time
0 = 1 cycles
1 = 2 cycles

mode_reg(3) T_CSR
CAS to RAS setup time
0 = 0 cycles
1 = 1 cycles

mode_reg(5:4) T_RAS
RAS# pulse width
00 = 4 cycles
01 = 5 cycles
10 = 6 cycles
11 = 7 cycles

mode_reg(6) T_RCD
RAS to CAS delay
0 = 1 cycles
1 = 2 cycles

mode_reg(8:7) T_RP
RAS precharge time
00 = 1 cycles
01 = 2 cycles
10 = 3 cycles
11 = 4 cycles

mode_reg(9) T_RSH
RAS hold time
0 = 0 cycles
1 = 1 cycles

2) SDRAM mode register value

The `mode_reg` parameters are also used to configure the mode register of the external SDRAM or SGRAM memories. The programming is achieved by generating a pseudo read operation during the circuit bootup time. The read address is formed from the `mode_reg` (15:0) value by multiplying it with 256. This read operation is handled so that instead of performing the typical activate and read cycles for the memory a mode-register-set cycle is generated. The mode-register-set is generated so that it replaces the activate cycle so the read address must be correctly aligned as a SDRAM row (activate) address.

The alignement requirements depend on the other memory parameters given in this register. See the examples bellow for more information.

On the current generation of SDRAM/SGRAM circuits the correct value for the mode register is: 030h which corresponds to
burst length "000" = burst length 1
burst type "0" = sequential
cas latency "011" = 3 cycle latency

Configuration examples:

mem_cfg	description
00030180h	SDRAM memory 4×16 bit memories (8 Mbytes) mode_reg = 0180h mm_8_16_chips = 1 mm_16_32_width = 1 mm_1_2_4_depth = 00
000100C0h	SDRAM memory 2×16 bit memories (4 Mbytes) mode_reg = 0C0h mm_8_16_chips = 1 mm_16_32_width = 0 mm_1_2_4_depth = 00
40030180h	2×(256K×32) SGRAM
40070300h	2×2×(256K×32) SGRAM
800001C0h	4×256K×16bit DRAM mode_reg = 01C0h T_AS 0 (0 cycles) T_CAS 0 (1 cycles) T_CP 0 (0 cycles) T_CSR 0 (0 cycles) T_RAS 0 (4 cycles) T_RCD 1 (2 cycles) T_RP 11 (4 cycles) T_RSH 0 (0 cycles)

video_clk_cfg	register 18	offset 0048h

Format

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

Fields

Field	Bits	Description
n_coef	6:0	N coefficient for core clock
m_coef	13:7	M coefficient for core clock
r_coef	15:14	R coefficient for core clock

Video Clock Configuration Register. The video clock frequency can be calculated from the formula:

$$F_{OUT} = \frac{m\_coef + 2}{(n\_coef + 2) \times 2^{r\_coef}} \times F_{OSC}$$

where:

n_coef, m_coef, r_coef = coefficients
F_{OSC} = quartz crystal or external clock (MHz)

For additional information see page 152.

After boot-up register contains value of 0000E087h (Fout = 25 MHz).

Caution: Unsuitable clock frequency parameters may cause permanent damage to the device.

reg_acc_addr	register 19	offset 004Ch

Format

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

Fields

Field	Bits	Description
reg_acc_addr	31:0	register access address

Register Access Address Register contains the address for internal register access.

reg_acc_data	register 20	offset 0050h

Format

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

Fields

Field	Bits	Description
reg_acc_data	31:0	register access data

Register Access Data Register provides an alternative method for accessing the VS25203 internal registers in situations where normal memory mapped register access is not available. Obviously this method is very slow. In order to use the access registers the target register address is written to the address register (19), and the value is written to the data register (20). The actual register write happens when the most significant byte of the access data register is written. This can be done with an 8, 16 or 32-bit configuration register write.

feat_reg	register 21	offset 0054h

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
														ddv	euio
ffe	ffm		fft									vee	vrsl	vrs	vde
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	Bits	Description
ddv	17	disable digital video
euio	16	enable user I/O [6:5]
ffe	15	flicker filter enable
ffm	14	flicker filter mode
fft	12:8	flicker filter thresh
vee	3	VGA extension enable
vrsl	2	VGA refresh select lock
vrs	1	VGA refresh select
vde	0	VGA decode enable

ddv
Disable digital video should be set to "1" when the digital RGB-port is used for other purposes, for example video capture or flash memory programming.
euio
Enable user I/O [5:6], extra enable signal required for user I/O(6) and user I/O(5) signals.
ffe
Flicker filter enable, a bit for activating the flicker filter and interlace module.
ffm
Flicker filter mode, affects the mode of operation for the flicker filter.
0 default value, optimal in most cases.
1 modified algorithm, which might provide better results on 100/120 Hz televisions.
fft
Flicker filter threshold, threshold value for flicker filtering 0 means no threshold (filter always), 16 means no filtering (perform interlace conversion still).
vee
VGA extension enable, enables using of extended VGA registers.
0 only standard VGA registers available.
1 extension registers are also available.
value after reset 0 (extension registers not visible).
vrsl
VGA refresh select lock, refresh register selection lock.
0 automatic selection of refresh registers active.
1 current selection locked.
value after reset 0 (not locked).
vrs
VGA refresh, selects 3D/VGA video refresh control.
0 3D video refresh registers used.
1 VGA refresh registers used.
value after reset 1 (VGA enabled) This bit changes its state automatically if VGA or 3D refresh registers are accesses, unless the select lock is active.
vde
VGA decode enable, activates the decoding of the standard VGA memory and IO ranges.
value after reset 1 (VGA enabled).

3.3.5 Interrupts

VS25203 provides the following interrupt facilities:

Video scanline:
If video interrupt is allowed, video scanline IRQ is enabled when `vq` field of the `ref_reg` register 49 is one. `vi` field of the `status` register 48 is one when interrupt is active. IRQ is triggered when the value of the `video_y_coord` field of the `status` register reaches current video refresh scanline, (`video_y_ref` field of the `ref_reg` register). The interrupt can be reset by writing value "1" into the `vi` field of the `status` register.

PCI master:
When PCI master causes an interrupt, `mi` field of the `status` register is one. The interrupt can be reset by writing value "1" into that same `mi` field of the `status` register. See additional information about the interrupt line and interrupt pin specified on page 28.

Field capture:
When field capture causes an interrupt, `capi` field of the `status` register is one. The interrupt can be reset by writing value "1" into that same `capi` field of the `status` register. For additional information see Video Capture base configuration register (`capt_base_conf` register 31).

VGA interrupt:
Refer to the chapter VGA Interrupt Generation on page 160.

Geometry Processor interrupt:
`the vi` field of the `status` register 48 is one when interrupt is active. The interrupt can be reset by writing value "1" into the `vi` field of the `status` register. The interrupt is set by `status_reg_in` register 194. For additional information see register 194.

3.4 System Control Registers

3.4.1 Overview

The system control registers contain registers which are used to control the PCI master functionality. Also some system debugging and state analysis registers are placed into this category.

PCI master control registers were originally placed at the PCI configuration space. But the present placement offers a more portable high performance interface for accessing them. This register set also contains extra I/O registers (page 41) which can be used to control the general purpose I/O pins of VS25203 (user_io[6:0] pins B9, C10, C12, B12, A12, C13 and B13, see page 234). These pins are used in a system dependent way.

With registers 54 and 55 it is possible to prepare some transformations between apertures, see page 44.

3.4.2 Registers

Register Number	Address Offset	Register name	Description
42	00A8h	ma_cmd_addr	PCI master command address register
43	00ACh	master_state	PCI master state register
44	00B0h	ma_int_addr	PCI master internal address register
45	00B4h	ma_ext_addr	PCI master external address register
46	00B8h	reserved	-
47	00BCh	reserved	-
48	00C0h	status	Status register
49	00C4h	ref_reg	Video reference register
50	00C8h	debug_reg	Debug register
51	00CCh	io_reg	I/O register
52	00D0h	ext_io_reg	Extra I/O register
53	00D4h	ext_io_reg2	Extra I/O register2
54	00D8h	mem_apt0_cfg	Memory aperture-0 configuration register
55	00DCh	mem_apt1_cfg	Memory aperture-1 configuration register

ma_cmd_addr	register 42	offset 00A8h

Format

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

Fields

Field	Bits	Description
ma_cmd_addr	31:0	PCI master command address

PCI Master Command Address Register. This register contains the start/current address of the master stream. User writes the start address of the stream to this register before starting the bus mastering operation. Note that this address is a physical address.

master_state	register 43	offset 00ACh

Format

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

Fields

Field	Bits	Description
master_cnt	23:0	PCI master counter
master_st	31:24	PCI master state

PCI Master State Register. This register, when read, provides debugging information about the current state of the PCI master unit.

master_st

Master state. Non-zero value in master_st starts the PCI master and zero in master_st halts the PCI master. Note that bit 2 (master enable) in the `status_cmd` register (a PCI configuration space register) has to be set to one to enable the master function.

master_cnt

`master_cnt` is read-only, and is used only for driver debugging.

Refer to `status_cmd` (1) register on page 25.

ma_int_addr	register 44	offset 00B0h

Format

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

Fields

Field	Bits	Description
ma_int_addr	31:0	PCI master internal address

PCI Master Internal Address Register. This read-only register contains the source address during the bus master stream read command (opcode 03h); for example, (gr_ram_bar + offset). Note that `ma_int_addr` (read-only) is used only for driver debugging.

ma_ext_addr	register 45	offset 00B4h

Format

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

Fields

Field	Bits	Description
ma_ext_addr	31:0	PCI master external address

This read-only register contains the destination address for the bus master stream read command (opcode 03h); for example, (gr_ram_bar + offset). Note that `ma_ext_addr` (read-only) is used only for driver debugging.

status	register 48	offset 00C0h

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
					video_y_coord										
mi	pv	vi	capi	gpi				gpf	gp0	blti	vc	id1	id2	ok1	ok2
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	Bits	Description
video_y_coord	26:16	Video y coordinate
mi	15	PCI master interrupt active
pv	14	Pixel visible
vi	13	PCI video interrupt active
capi	12	Video Capture interrupt active
gpi	11	Geometry Processor interrupt active
gpf	7	Geometry Processor flag
gp0	6	Geometry Processor stream0 flag
blti	5	Block Transfer Unit idle
vc	4	video compare
id1	3	Primitive processor idle
id2	2	Pixel processor idle
ok1	1	Primitive processor init ok
ok2	0	Pixel processor init ok

video_y_coord
Video y coordinate. Current video refresh scanline.
mi
PCI master interrupt active. This bit is set to "1" when VS25203 has interrupt request active when interrupt is activated by the PCI master block. The interrupt is active until the device driver resets the interrupt. PCI bus master interrupt is reset by writing a value "1" into this field.
pv
Pixel visible. This bit is set to one when a visible pixel has been detected by the Pixel Processor in the `zread` operation. The bit is reset by writing a value "1" into this field. Refer to the `grid_reg` (102) register.
vi
PCI video interrupt active. This bit is one when the device has an active IRQ. This interrupt is caused by the `video_y_ref` field of `ref_reg` (49) register through the video-y comparator. The interrupt is active until the device driver resets the interrupt. Video interrupt is reset by writing a value "1" into this field.
Capi
Video Capture interrupt active. This bit is set to one when the circuit has interrupt request active if the interrupt has originated from the Video Capture unit. The interrupt is active until the device driver resets the interrupt. The video interrupt is reset by writing the value "1" into this register bit.
gpi
PCI Geometry Processor interrupt active. This bit is one when the device has an active IRQ. Video interrupt can be reset by writing a value "1" into this field. See also `Status_reg_in`, register 194.

gpf
Geometry Processor flag.
gp0
Geometry Processor stream 0 flag.
blti
Block Transfer Unit idle. Indicates status of the Block Transfer Unit.

1 idle
0 busy

vc
This bit is one when the `video_y_coord` field value is equal or greater than the `video_y_ref` value of the `ref_reg` , register 49.
id1
Primitive Processor idle. This bit is one when the Primitive Processor is in the idle state.
id2
Pixel Processor idle. This bit is one when the Pixel Processor is in the idle state.
ok1
Primitive Processor initialization ok. This bit is one if initial values are allowed to be written to the Primitive Processor.
ok2
Pixel Processor initialization ok. It is used for finding out when the Pixel Processor can be initialized. In VS25203, it is given by `id1` and `id2`.

ref_reg	register 49	offset 00C4h

Format

31	30	29	28	27	26	25	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
			vgaq	vq	video_y_ref										

Fields

Field	Bits	Description
vgaq	12	VGA IRQ ena
vq	11	Video IRQ
video_y_ref	10:0	Video y reference

vgaq
If this bit is set then the VGA unit generated interrupt is routed to the PCI bus. An interrupt which is initiated by the the VGA block must be reset using the VGA unit.
vq
Video IRQ. When this bit is set to one the device will generate an interrupt request (IRQ) when the `video_y_coord` field value of the `status` register (49) is equal or greater than the `video_y_ref` value.
video_y_ref
Video y reference scanline.

debug_reg	register 50	offset 00C8h

Format

31–16
debug_reg

15–0
debug_reg

Fields

Field	Bits	Description
debug_reg	31:0	System debug register

Hardwired to zero.

io_reg	register 51	offset 00CCh

Format

31–16

15	14–8	7	6–0
	user_io_out[6:0]		user_io_in[6:0]

Fields

Field	Bits	Description
user_io_out[6:0]	14:8	User I/O register (read-only)
user_io_in[6:0]	6:0	User I/O register (read/write)

user_io_out[6:0]
Seven general purpose user_io pins of the VS25203, where the value is driven out from the pins. The actual direction of the pins (input/output) depends on the user I/O enable bits in extra_io_reg (52) register. See also miscellaneous signals on page 234.
user_io_in[6:0]
The value read from the user_io pins.

ext_io_reg	register 52	offset 00D0h

Format

31	30	29	28–24	23–16
tm	eie		usr_io_ena	extra_out_data

15–8	7–0
extra_out_data	extra_in_data

Fields

Field	Bits	Description
tm	31	Test mode
eie	30	Extra I/O enable
usr_io_ena	28:24	User I/O enable[4:0]
extra_out_data	23:8	Extra out data value
extra_in_data	7:0	Extra in data value

extra_in_data
Blue color bus B[7:0] of the DAC or BIOS data.
extra_out_data
Red and green color buses R[7:0], G[7:0] of the DAC or high and low BIOS addresses.
It gives the value driven on the RG pins (digital video), if extra I/O enable is active. See also external DAC signals on page 233, chapter Signal Descriptions.
usr_io_ena
Controls the direction of the user I/O pins [4:0].
0=not driven-input
1=driven-output
each pin has a separate control bit.
eie
Extra I/O enable. Controls the driving of the `extra_out_data` field to the Red/Green DAC signal lines; if eie=0, the value is not driven.
tm
Reserved for test purposes. Should be zero.
Caution: Be careful when setting bits 28:24; as they define the direction of `io_reg` (51) register.

ext_io_reg2	register 53	offset 00D4h

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
edbe															
						usr_io_e2		extra_out_data_b							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	Bits	Description
edbe	31	Extra out data_b_ena
usr_io_e2	9:8	User I/O enable[6:5]
extra_out_data_b	7:0	Extra out data value for B

edbe
Enable signal for extra data out value
usr_io_e2
Controls the direction of the user I/O pins [6:5].
0 not driven-input
1 driven-output; each pin has a separate control bit.

extra_out_data_b
Data out value of B component

mem_apt0_cfg	register 54	offset 00D8h

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
rl0	m0	ws0	bs0		apt0_width					apt0_height					
		apt0_addr													
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	Bits	Description
rl0	31	Aperture 0 raw / linear mode access
m0	30	Aperture 0 16/32 bit mode
ws0	29	Aperture 0 word swap
bs0	28	Aperture 0 byte swap
apt0_width	26:24	Aperture 0 width
apt0_height	21:16	Aperture 0 height
apt0_addr	13:0	Aperture 0 start address

apt0_width
Aperture 0 width. Used in splitting the X and Y coordinates from the memory address in linear mode. The value is the number of bits in X coordinate. Values are translated as presented in the table above.

Value	Texture/screen width (pixels)
0	32
1	64
2	128
3	256
4	512
5	1024
6	2048
7	Reserved

apt0_height
Aperture 0 height in 32 pixel boxes
apt0_addr
Aperture 0 start address in 2048 byte blocks
ws0
Aperture 0 word swap. If this bit is one then the memory accesses will have the 16-bit words swapped, thus e.g. byte ordering 3210 becomes 1032.
bs0
Aperture 0 byte swap. If this bit is one then the memory accesses will have the byte ordering swapped, thus byte ordering 3210 becomes 0123. This can be used in combination with `ws` to produce the ordering 2301 from 3210.
m0
Aperture 0 16/32 bit mode
0 16-bit mode (or packed YUV)
1 32-bit mode (ARGB or YUV-24+α mode)
rl0
Aperture 0 raw/linear mode:
0 the aperture is in raw mode
1 the aperture is in linear frame buffer mode

m0, apt0_width and apt0_height are used only when the aperture is in the linear frame buffer mode.

mem_apt1_cfg	register 55	offset 00DCh

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
rl1	**m1**	**ws1**	**bs1**		**apt1_width**					**apt1_height**					
		apt1_addr													
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	Bits	Description
rl1	31	Aperture 1 raw / linear mode access
m1	30	Aperture 1 16/32 bit mode
ws1	29	Aperture 1 word swap
bs1	28	Aperture 1 byte swap
apt1_width	26:24	Aperture 1 width
apt1_height	21:16	Aperture 1 height
apt1_addr	13:0	Aperture 1 start address

See `mem_apt0_cfg` register (register 54).

3.5 Graphics Memory Type

The preferred memory type for VS25203 is 1M × 16 Synchronous DRAM (SDRAM). Full VS25203 performance can be achieved by connecting VS25203 with four 16-bit wide SDRAM devices. A version with lower performance can be created by using only two 16-bit wide SDRAMs. The performance drop is mostly significant when high resolution and/or true-color modes are used.

It is also possible to use 32-bit wide Synchronous Graphics DRAMs (SGRAM) with VS25203. SGRAM prices are higher than SDRAM prices for the same memory size, but because of their different configuration a graphics system with a smaller total memory can be created by using SGRAMs. Too small a memory will obviously limit the texture capabilities of VS25203.

For lower performance systems VS25203 can also be combined with EDO DRAMs.

Note that for SDRAM and SGRAM, burst access (meaning the transfer of multiple data phases per a single address phase which requires programming the SDRAM/SGRAM mode register with the burst length) is not used in VS25203. In other words, the burst length is only 1. Instead, VS25203 uses pipelined accesses to the current page which has no performance differences with SDRAM/SGRAM burst accesss. Typically on larger triangles the frame and Z buffer accesses use up to 32 consecutive cycles. For textures, the average consecutive access count to a single page is lower.

The following SDRAM commands are used by VS25203:

- mode register write
- precharge
- activate
- read/write
- refresh

VS25203 treats SGRAM accesses similarly and does not currently use any of the SGRAM special features.

4. Geometry Processor

4.1 General Information

The Geometry Processor is based on 3-issue VLIW architecture with a packed 32-bit instruction word. It has three Arithmetic Units (AU) and additional units for hardware division, logic operations and other tasks. The AUs have three-cycle pipelines with multiplication as the first stage, addition as the second and shifting as the final stage. The addition is also used as the second stage of the multiplication, so both cannot be started on the same cycle. The processor also has three integrated data memories, so there is no need to use the external graphics memory during calculations.

The Geometry Processor interfaces to the other blocks of the VS25203B by having nearly full access to the registers of the chip. These registers can be written from the A-register of the AU2. Normally the task of the Geometry Processor is to process a data stream and calculate values to the Primitive Processor and Pixel Processor registers. The stream is read from the external graphics memory to the stream buffer, from where it can be loaded to the AU registers one 64-bit word at a time.

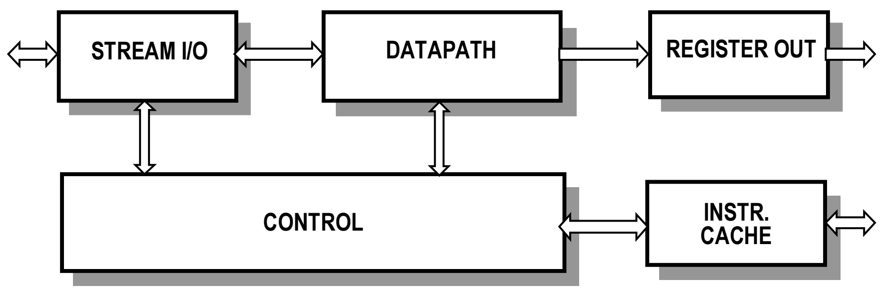


Figure 4.1-1 Geometry Processor, top-level architecture.

Figure 4.1-1 shows the top-level architecture of the Geometry Processor. The processor is optimized for 3D scene processing. Input data comes as a stream and the final result is a set of initialization values for the other units in the VS25203B. The Stream I/O Unit provides the input data for the Geometry Processor. The data can come from either through the PCI interface (direct stream data) or from the external graphics memory through the Memory Manager. The Stream I/O can also be used to write data to the external memory. The Datapath, described in chapter 4.2, does the actual work controlled by the Control block that fetches and decodes the 32-bit instruction words. The Register Out block provides the interface for the other units in the VS25203B chip.

The program that controls the Geometry Processor is given by the user and is stored in the external graphics memory. The address space for the program memory is 16384 words, and the location in the external graphics memory is given by the user in the `CODEBASE` register (see `code_config`, register 193 and the page 99). The program is cached into a 4-way set-associative on-chip instruction cache with a 128 word block size. The program can also initiate prefetching of the cache blocks.

As is usual for a VLIW processor, the architecture and pipeline in the Geometry Processor are visible to the programmer, and he or she must take into account all the pipeline effects. This enables one to write maximally efficient code, but requires more care in programming. To make sure the instruction cache is used efficiently it is important to organize the code properly.

4.1.1 Geometry Processor Bus Structure

The Geometry Processor has one 32-bit wide global data bus called the **General bus**. This bus is used to transfer data between the Control block, Stream I/O and the Datapath. In addition to the General bus, the Stream I/O Unit transfers data read from the stream using the **Stream bus**, which is a combination of three 32-bit buses, one for each of the Arithmetic Units. By using this bus it is possible to read up to three values from the stream in one instruction.

Internal to the Datapath, the Arithmetic Units are connected to the local data memories using **two buses per AU**. This enables transfer of up to six data values in one instruction. **One** of the buses is used to write data from the AU's A-register to any of the memories and read data from the associated memory to the Y-register. The **second bus** is used to read data from any of the data memories to the X-register of the AU. I.e. for each of the AUs, during one instruction execution, it is possible to read one value to the X-register and either read one value to the Y-register or write one value from the A-register.

In addition to these Geometry Processor local buses there are three other buses involved in the operation of the Geometry Processor. The **External Stream Interface bus** has a 64-bit interface to the external graphics memory or to the PCI bus. The **Register Out bus** offers a 32-bit-wide path for writing data to the registers of the other blocks on the VS25203B chip. Finally the program for the Geometry Processor is read through the **Program Memory bus** which has a 64 bit-wide path from the external graphics memory to the instruction cache of the Geometry Processor.

4.2 Datapath Architecture

The Datapath of the Geometry Processor, illustrated in the Figure 4.2-1, does all the calculations needed for executing the users programs. It consists of three parallel 32-bit fixed point Arithmetic Units, a 32-bit wide Logic Unit, a Normalization Unit, and a 32/24-bit Hardware Division Unit. The results of the three AUs can be combined together using a 3-element vector adder. The Logic, Normalization, and Hardware Division Units are closely connected to the Arithmetic Unit 2. They receive input from the X and Y-registers of the AU2 (`X2` and `Y2`), and the result of the Logic unit is written to its A-register (`A2`).

The A-register of the Arithmetic Unit 2 has some special functionality over the A-registers of the other AUs. It is used as a source for the register out instructions (OUT) to write data to the registers in the other units of the VS25203B. The `A2` register is the only one of the A-registers to receive data from the vector adder and the Logic Unit. The instruction encoding additionally restricts the `A2` register to be the only one receiving values from the divide output registers (quotient and remainder), the `STATUS` register (see chapter 4.5.6 on page 63), the `N` register (Result of Normalize instruction), and the `JMPREG`.

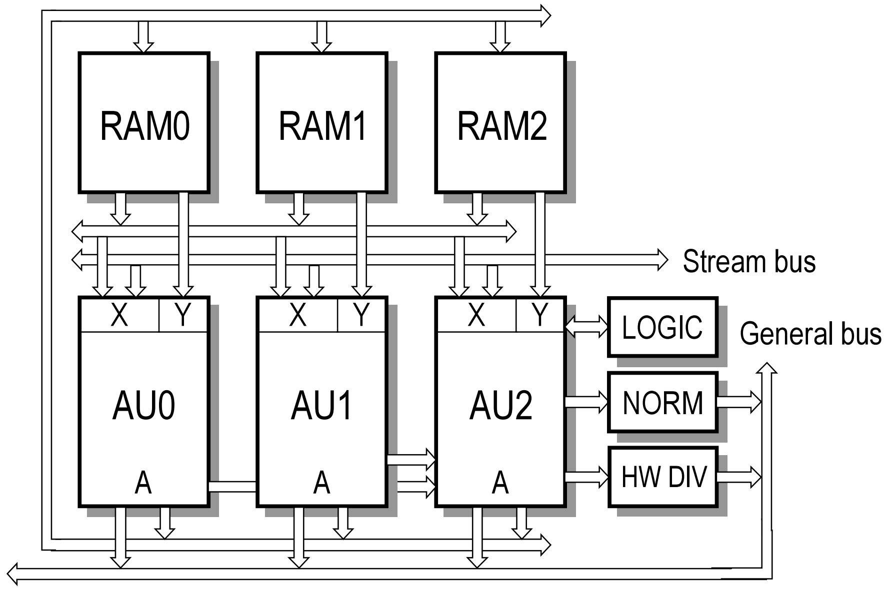


Figure 4.2-1 Block diagram of the Datapath.

4.2.1 Arithmetic Unit

Each of the three Arithmetic Units has a 32-bit fixed point datapath with 24x24 => 48-bit multiplier shown in Figure 4.2-2. The Arithmetic Unit has two input registers: the X-register and the Y-register. The X-register can be loaded with values from any of the local data memory banks, the Stream I/O Unit, any of the A-registers of the AUs, and immediate values from the instructions. The Y-registers may be loaded with values from the corresponding local data memories only, i.e. `Y0` may be loaded from RAM0 only. The Arithmetic Unit can also use the values from the result register A as operands for its instructions. This reduces the impact of the reduced functionality of the Y-registers and makes it possible to effectively perform chained calculations.

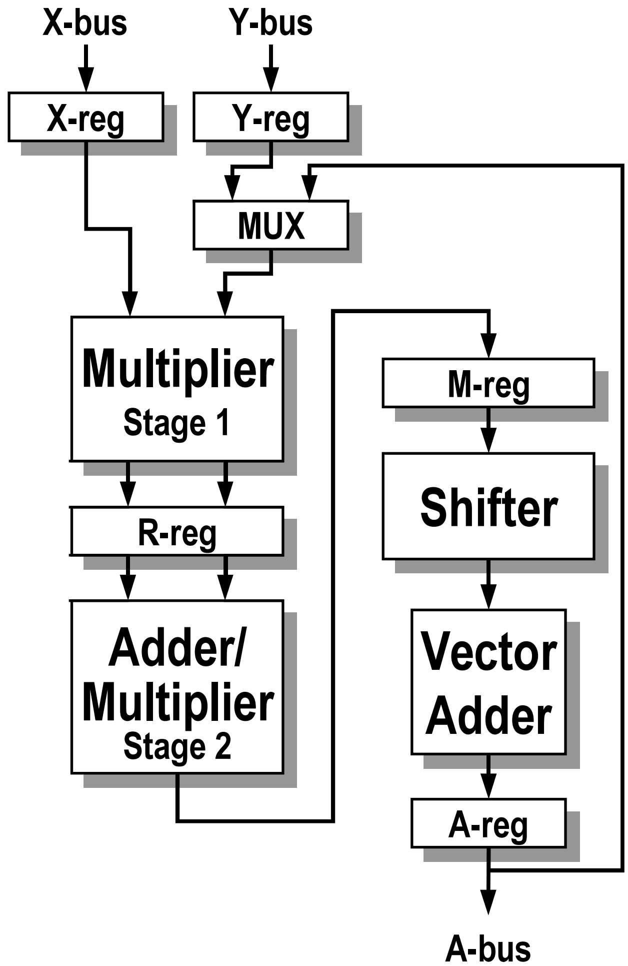


Figure 4.2-2 Block diagram of the Arithmetic Unit.

The multiplier result register R is accessed only through Arithmetic instructions. It receives values through the Multiplier block which performs the first stage of the multiplications. The Multiplier block produces two intermediate values to be processed in the Adder block to produce the final multiplication results. See Figure 4.2-2.

The Multiplier block is also used to format the operands for the other functions performed by the Adder block. Results of the Adder block are normally stored into the M-register for processing by the Shifter block. The A-register finally receives the AU results from the Shifter unit after a latency of three instruction cycles.

For simple operations it is possible to bypass some of the intermediate registers to reduce latency. If multiplication is not used, then the R-register may be bypassed and the values of the X and Y/A-registers are used directly as operands for the instruction. Also if shifting or vector addition of the results is not necessary, the results from the Adder can be directly written to the A-register bypassing the M-register. This makes it possible to perform for example addition of X and Y-register values directly to the A-register in one clock cycle.

The AUs support the following instructions:

ADD	Addition
SUB	Subtraction
MUL	Multiplication
SHIFT	Shift M-register by up to +- 32 bits
SHIFTN	Shift M-register by the value of the N register
NEG	Negation of one value
PASS	Passing through of input values
ZERO	Result is zero
INC	Increment A-register value
DEC	Decrement A-register value
ABS	Absolute value of an AU register
NEG_ABS	Negation if the absolute value of an AU register
ADD_ABS	Add the absolute value of the X-register to A-register
SUB_ABS	Subtract the absolute value of the X-register from A-register
SAT	Saturate value of the A-register to 8-bit signed range
SATU	Saturate value of the A-register to 8-bit unsigned range

In addition of these AU2 also supports the vector adder instructions, which allows addition of any of the shifted values of the M-registers in the AUs. One or two of the M-register values can also optionally be negated. This allows the programmer to perform pipelined 3-element vector dot-products in a single cycle.

The Datapath supports two different numeric formats. Both of the Datapath numeric formats use 2's complement numeric representation. The first format is a normal 32-bit wide integer format. The second is a 32-bit wide fixed-point format where the binary point is between the sign bit and the mantissa. The values that can be represented by these formats are summarized in the table below:

	Integer	Fixed point
Minimum value	-2^{31}	-1
Maximum value	$2^{31} - 1$	$1-2^{-31}$
Smallest difference	1	2^{-31}

Since the inputs of the Multiplier blocks are only 24-bit wide, they cannot use the whole 32-bit data range supported by the Datapath. The multiplication of two integer type operands of 24 bits results in a 47-bit wide result when using 2's complement representation. However the final result of the multiplications should fit to a 32-bit wide register.

For integer data, if the operands are too big, the result can overflow the data range. For the fixed point case, the result can never overflow, but if the values are too small the result of the multiplication can underflow. To support use of these two formats in multiplications, the Multiplier block can multiply either the lower 24 bits of the operands (integer format) or the upper 24 bits (fixed point format). In the case of integer format multiplication, the 32 lower bits of the result contain the desired multiplication result. For the fixed point case, the upper 32 bits contain the result. The instruction set supports direct loading of upper or lower 32 bits of the result to the A-registers. For better control of the data ranges, it is possible to use the Shifter through the M-register.

4.2.2 Logic Unit

The Logic Unit is used to perform normal logical operations between its operands (X2 and Y2), but it can also be used to perform bit-field operations. The result of the Logic operation is loaded to register A2. The Logic Unit instruction bits drive the circuitry directly, so it is possible to use the unit very creatively. The schematic representation of the Logic Unit is shown in the Figure 4.2-3. The shift and mask values are used to form a bitmask containing mask bits shifted left by the value of shift.

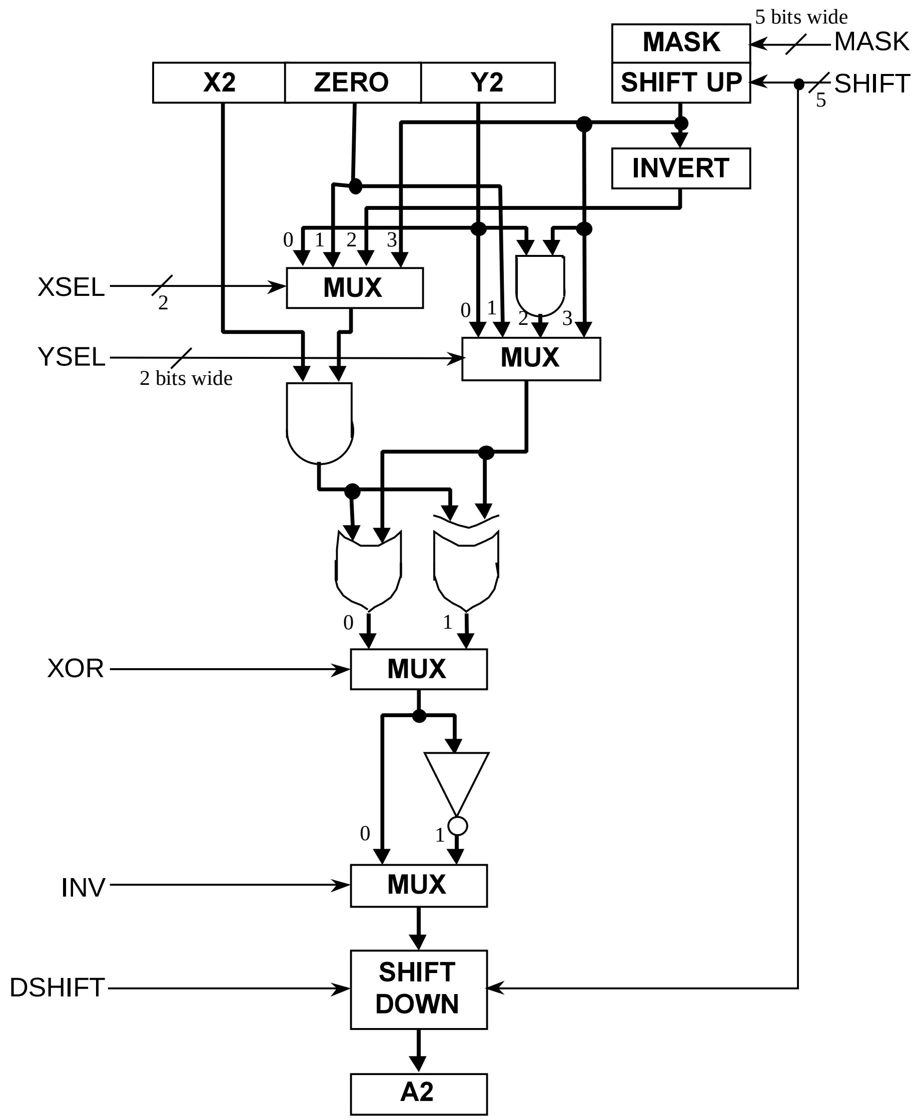


Figure 4.2-3 Block diagram of the Logic Unit.

With the Logic Unit it is possible to produce for example the following operations:

AND	Logical AND between X2 and Y2
NAND	Logical NOT of AND between X2 and Y2
OR	Logical OR between X2 and Y2
NOR	Logical NOT of OR between X2 and Y2
XOR	Logical Exclusive OR between X2 and Y2
XNOR	Logical NOT of Exclusive OR between X2 and Y2
NOT	Logical NOT of X2 or Y2
PASS	Pass X2 or Y2 directly to output
EXTRACT	Extract bit-field from X2 or Y2
BIT_TEST	1-bit version version of the above
COPYBIT	Copy bit-field from Y2 to value of X2
SETBIT	Set bit-field of value of X2 to ones
CLRBIT	Clear bit-field of value of X2 to zeros
NEXTRACT	Extract bit-field from logically negated value of X2
MASK_AND	Logical AND of bit-field of ones and value of X2 or Y2

The Logic operations also affect to the STATUS register bit 3 (see the page 63). It is updated with the LSB (bit 0) of the Logic Unit operation's result value. All the Logic instructions affect to the STATUS register bit. For example, it is possible to test whether X2 or Y2 is even or odd using the PASS Logic operation.

4.2.3 Normalization Unit

The Normalization Unit has two functions. It has an N register which is used in AU operations as the operand for the SHIFT instructions. The N register can be loaded from any of the A-registers, directly via the Immediate Load instructions or executing the Normalize instruction. The Normalize instruction is also mentioned on page 92.

The Normalize instruction calculates the shift value needed to normalize its operand. A number is said to be normalized when its two most significant bits are different.
For fixed point numbers this means that a normalized number is in the range [-1 ; -0.5) or [0.5 ; 1). The Normalize instruction receives its input from the X2 register, and the output goes to the N register. Normalization is useful in implementing block-floating-point-operations and it can also be used to quickly estimate the base 2 logarithm of the absolute value of the operand.

4.2.4 Hardware Division Unit

The Hardware Division Unit implements iterative division of 24- and 32-bit numbers. The division operation needs either 12 or 16 clock cycles, respectively, to complete depending on the operand format. The dividend can be either positive or negative, but the divisor must always be positive.

The Hardware Division Unit operates in parallel with the other units in the Geometry Processor, so that the program needs not to stop to wait for the division to complete. There is no hardware locking to prevent trying to extract the division results too early. If this happens, the program just receives incorrect results.

There is also no protection against starting a second division too soon after the first. After the required clock cycles the division unit freezes the result so it may be extracted at any time after the Division operation is finished, however it should be extracted before a new one is started. I.e. pipelining of divisons is not supported.

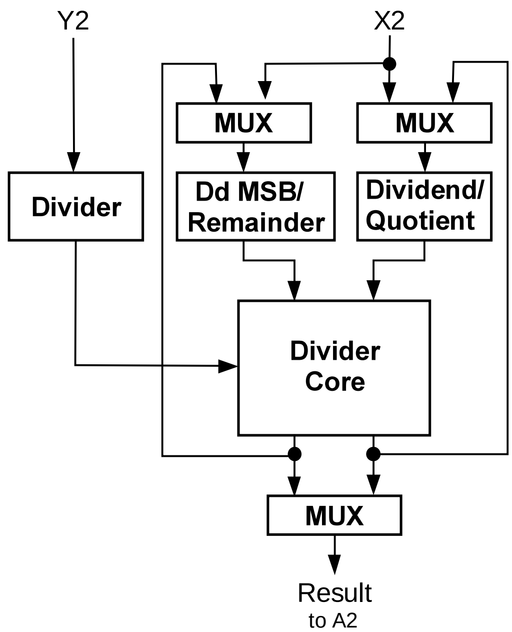


Figure 4.2-4 Block diagram of Hardware Division Unit.

The division unit shown in Figure 4.2-4, uses a non-restoring radix-4 iterative algorithm for the divisions, and both the quotient and the remainder from the operations are available for the programmer. The remainder is, however, not usable directly. The programmer must perform the restoration step to get the real value of the remainder. The restoration algorithm is described in more detail on the page 94.

4.2.5 Data Memory

The Datapath has local data memory for storing intermediate values required by the user's programs. There are three banks of 128 words each. The data word is 32 bits wide. The memories are dual ported so that it is possible to either read two values in one cycle or read one value and write one value. The two ports of the memories are connected in the following way:

port 1	read-only	read to X-registers
port 2	read-write	read to Y-registers write from A-registers

The dual port structure of the memories gives raise to a hazard in the memory operation. If the user's program tries to write to a memory location which is also being read at the same time, the results of the read are unspecified. In addition, the particular memory location in question will contain unspecified value after the operation. However, besides of yielding unspecified results the memories cannot be physically harmed by this. For getting better performance the hazard is left for the programmer to resolve, instead of being handled in hardware.

4.3 Instruction Execution

The Geometry Processor instructions are formed of several fields, and the total width of the instructions is 32 bits. All instructions have the same width, and execute in one clock cycle. Some operations, e.g. Divide, can take more than one clock cycle to complete, but other instructions can be executed in parallel.

The pipeline of the Geometry Processor is visible to the programmer. This means that the programmer should take care of the pipeline by himself / herself. The visible pipeline enables one to write maximally efficient code but causes some overhead in the programming work. The maximum address space for program memory is 14 bits. The on-chip memory is 512 words divided into 4 banks that are cached from the external memory. The location of the Geometry Processor program memory in the external graphics memory is configured through the `CODEBASE` register. Writing to the `CODEBASE` register from the Geometry Processor allows one to have more than one logical address space within the external memory. This can be used to extend the effective program memory address space beyond 14 bits (16384 words). There is no cache flush instruction, so the programmer should take care of the cache effects. See also the page 99.

The Geometry Processor uses a classic 3-stage pipeline consisting of fetch, decode, and execute stages. The normal execution of instructions is shown in the table below:

Instruction	Execution Stage				
Instr1	Fetch	Decode	Execute		
Instr2		Fetch	Decode	Execute	
Instr3			Fetch	Decode	Execute

Time ⟶

Control transfers are implemented as delayed branches with one delay slot. This means that the instruction following the branch instruction is executed always, not depending on whether the jump is actually taken or not. No data moves are delayed, which means that data transferred by one instruction will be available for use during the next instruction. The normal BRANCH execution is shown in the table below:

Instruction	Execution Stage						
Taken Branch	Fetch	Decode	Execute				
Branch+1		Fetch	Decode	Execute			
Target			Fetch	Decode	Execute		
Target+1				Fetch	Decode	Execute	
Target+2					Fetch	Decode	Execute

Time ⟶

There are two major branch categories: jumps and subroutine calls. It is possible to use either unconditional or conditional branches. Conditional branches use the `STATUS` register, described on the page 63, to evaluate the branch conditions. The `STATUS` register contains the sign bits of all the A-registers in the Arithmetic Units, and the BIT_TEST flag from the Logic Unit. Also the branches can be direct or indirect, in which case the branch address is taken from the `JMPREG` register described on the page 62. The block diagram of the program address calculation unit is shown in Figure 4.3-1.

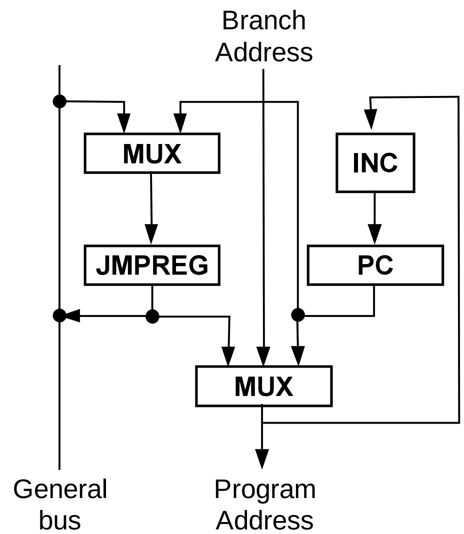


Figure 4.3-1 Block diagram of Program Counter (PC).

The decision for the branch must be done in the decode stage of the branch instructions to be able to fetch the next instruction after the delay slot instruction. Since the branch condition which is derived from the flags is not ready until the execution stage of the branch instruction we cannot be sure whether the branch will be taken or not.

The Geometry Processor uses speculation based on the previous value of the flags to make an early decision of the address of the next instruction. If the prediction was incorrect it is corrected during the next instruction cycle by canceling the decoding and execution of the incorrectly fetched instruction. The execution of a canceled branch is shown in the table below:

Instruction	Execution Stage						
Taken Branch	Fetch	Decode	Execute				
Branch+1		Fetch	Decode	Execute			
Target (cancel)			Fetch				
Branch+2				Fetch	Decode	Execute	
Branch+3					Fetch	Decode	Execute

Time →

The canceling process increases the cycle count of the branch to two cycles from the normal value of one. Since the branch prediction is done based on previous flag values it is possible to optimize the code by not changing the flags on the instruction prior to the branch instruction. This effectively introduces an extra delay slot **before** the branch instruction. If the pre-branch slot cannot be filled with useful code, it can be used for instruction affecting the flags. This causes no penalty to the execution time compared to the case where the pre-branch instruction would be a NOP. On the other hand, it saves one code memory word.

The Geometry Processor has no interrupts. All synchronizing to the external world must be done by polling. Because of the nature of the Geometry Processor tasks this should cause no problems.

When accessing data from external sources the Geometry Processor may need to wait for the data to be ready. There are three possible sources for these wait conditions, which will cause the Geometry Processor to enter a wait state.

First of these conditions arise from a cache miss. In this case the Geometry Processor issues a request for the Memory Manager to fill a block of its code memory from the cache-miss location. During the time the Memory Manager is fetching data from the external graphics memory the Geometry Processor waits and does not process any instructions. After the code load is complete, the Memory Manager notifies the Geometry Processor's Control unit, and the missed instruction is re-fetched, and normal processing continues.

The second case for hardware wait condition can occur while issuing stream fetch commands to the Stream I/O Unit. If the stream FIFO is empty, the Geometry Processor will enter a wait state until a word is ready to be fetched from the stream FIFO. After the wait the stream data word is fetched and normal processing resumes. Normally the Stream I/O Unit will try to keep the FIFO filled, but other units accessing the external graphics memory or the user's program changing the Stream Read address can cause a FIFO empty condition. The Stream I/O operation is described in more detail in chapter 4.7.4.

The third and final wait condition arises if the user sets the Geometry Processor `wait` bit in the Synchronization register (PCI register 192).

It is also possible to completely reset the Geometry Processor independent of the other units in the VS25203B chip by setting the Geometry Processor reset bit (`ge_reset`) in the Synchronization PCI register. In this case all the Geometry Processor registers are set to the initial values, however the local data memories are not affected.

4.4 Addressing Modes

The Geometry Processor support the following Addressing modes

Register Direct	
Immediate	(24-bit)
Short Immediate	(13-bit)
Absolute	(14-bit)
I/O Indirect	(6-bit)
Absolute Data	(4-bit)
Index Register Indirect	(4-bit)

All the Arithmetic instructions use **Register Direct Addressing** mode to provide the operands. This means that all values used by the Arithmetic Units, the Logic Unit, the Normalization Unit, and the Hardware Division Unit need to be loaded to the AU input registers using one of the data-move instructions prior to performing the operations. The Geometry Processor instruction set provides several instructions for this purpose.

The **Immediate 24-bit Addressing** mode is used by the Long Immediate Load instruction (IMMED) to load immediate values to the AU data registers, Stream Read address (`RDADDR`), Stream Write address (`WRADDR`), or to the `JMPREG` register. The values are interpreted as integers and are sign extended, except for loading to the upper part of the A-registers. In that case the values are interpreted as fixed point values and the lower 8-bits are zero filled.

The **Short Immediate 13-bit Addressing** mode is used by the Short Immediate Load instruction (SIMMED) to load immediate values to the Index registers, the `N` register, the IO register address base register (`REGBASE`), or the bus index register (`VTMB`). The 13-bit values are interpreted as integers and are sign extended.

The **Absolute 14-bit Addressing** mode is used for the Branch instructions to provide the target addresses in cases where calculated jump (`JMPREG`) is not used. The PCI register `CODEBASE` is used to give the base address for the program code in the external graphics memory.

The **I/O Indirect Addressing** mode is used to get the addresses for the external registers used as target for the OUT instructions. The register address is calculated by addition of the 10-bit `REGBASE` register and the 6-bit immediate offset from the instruction word. If the resulting value is too large to fit to 10 bits, the 10 lowest bits of the result are used. The 6-bit immediate address part is interpreted as an unsigned value.

The **Absolute 4-bit Data Addressing** mode is used to load values from the local data memories to the Arithmetic Unit input registers. This addressing mode can only address 16 lowest addresses of the data memories, but it can be used to access these locations independent of the values of the Index registers.

The **Index Register Indirect 4-bit Addressing** mode is used to load values from the local data memories to the Arithmetic Unit input registers. This addressing mode can address 16 addresses relative to the data memories. The data address is calculated by addition of the 7-bit Index register and the 4-bit immediate offset from the instruction word. If the resulting value is too large to fit to 7 bits, the 7 lowest bits of the result are used. The 4-bit immediate address part is interpreted as an unsigned value. There are 9 Index registers for each of the possible memory read and write operations. The `XRDBASE#` registers are used when reading data to the X-registers, the `YRDBASE#` registers are used when reading data to the Y-registers, and the `WRBASE#` registers are used when writing values to the data memories from the A-registers.

For the local data memory load and save instructions there is one special feature to consider. In addition of specifying the source and target memories directly in the instruction it is possible to use data driven memory indexes. This feature uses the `VTMB` register. The value of this three bit register is interpreted as three bus index values (VT, VM, VB) according to the table below:

dec	Bin	VT	VM	VB	VT	VM	VB
0	000	0	1	2	00	01	10
1	001	0	1	2	00	01	10*
2	010	0	2	1	00	10	01
3	011	2	0	1	10	00	01
4	100	1	0	2	01	00	10
5	101	1	2	0	01	10	00
6	110	0	1	2	00	01	10*
7	111	2	1	0	10	01	00

*) Not possible as flags value for Derive VTMB.

It is possible then to use these bit indexes to specify from what local data memory bank to load values to the X-registers or to which memory bank to write from the A-registers.

For example if the value of the VTMB register is 011 then the instruction:

```
X2 =  VB[0], X1 = VM[0], X0 = VT[0],
      VB[1] = A0, VT[1] = A1, VM[1] = A2
```

will load X2 with value from memory bank 1, X1 with value from bank 0, and X0 with value from bank 2. It will also write A0 to bank 1, A1 to bank 2, and A2 to memory bank 0.

4.5 Geometry Processor Registers

4.5.1 General

The Geometry Processor contains many internal registers. All the registers, except for the stream related ones are set to zero when the Geometry Processor is reset either with the global chip reset or with the special Geometry Processor reset bit, see the Synchronization register (PCI register 192).

The registers can be divided into four classes: Arithmetic Unit registers, Stream registers, Index registers, and the Control registers. See table below.

Registers	Bits	Description
Arithmetic Unit registers		
X0	32b	AU0X input register
X1	32b	AU1X input register
X2	32b	AU2X input register
Y0	32b	AU0Y input register
Y1	32b	AU1Y input register
Y2	32b	AU2Y input register
R0	48b	Multiply result [x2]
R1	48b	Multiply result [x2]
R2	48b	Multiply result [x2]
M0	48b	Shifter input
M1	48b	Shifter input
M2	48b	Shifter input
A0	32b	AU0 result register
A1	32b	AU1 result register
A2	32b	AU2 result register
Stream registers		
RDADDR	24b	Stream Read address
WRADDR	24b	Stream Write address
STREAM(HI)	32b	Stream data high
STREAM(LO)	32b	Stream data low

Registers	Bits	Description
Index registers		
XRDBASE0	7b	X bus 0 read index register
XRDBASE1	7b	X bus 1 read index register
XRDBASE2	7b	X bus 2 read index register
YRDBASE0	7b	Y bus 0 read index register
YRDBASE1	7b	Y bus 1 read index register
YRDBASE2	7b	Y bus 2 read index register
WRBASE0	7b	Write bank 0 index register
WRBASE1	7b	Write bank 1 index register
WRBASE2	7b	Write bank 2 index register
Control registers		
N	6b	Shift value / Normalization value register
PC	14b	Program counter
JMPREG	14b	Calculated jump address register
REGBASE	10b	IO register address base register
VTMB	6b	Bus index register (special decoded 3b format)
STATUS register	4b	GP's internal register, not same as register 48.

4.5.2 Arithmetic registers

Register	Bits	Description
X0	32b	AU0X input register
X1	32b	AU1X input register
X2	32b	AU2X input register
Y0	32b	AU0Y input register
Y1	32b	AU1Y input register
Y2	32b	AU2Y input register

The Arithmetic Unit input registers are used to provide input to the AUs. In addition the X2 and Y2 registers are used to provide inputs to the Logic unit and the Hardware Division Unit. The X2 register is also used as input for the Normalization unit.

The X-registers can be written to from any of the local data memory banks, the Stream I/O Unit or the General bus. The Y-registers can only be written to from the corresponding local data memory bank.

The instruction set allows to write to all the X-registers at once with a single value. This combination (X012) can be used on General Move instructions, MOVE_REG instructions and the immediate load instructions (IMMED and SIMMED).

Register	Bits	Description
R0	48b	Multiply result [x2]
R1	48b	Multiply result [x2]
R2	48b	Multiply result [x2]

The Multiply result registers are pseudo-registers. The real values and format of these registers is not available directly to the programmer. These registers contain either the intermediate result of the multiplication (before final addition) or the (intermediate) result of an addition operation. These registers cannot be directly written or read, but are accessed indirectly by the Arithmetic instructions. If written through addition operations the R-registers are sign extended from the 32-bit result, so that the effective data is in the lower 32 bits of the R-registers.

Register	Bits	Description
M0	48b	Shifter input
M1	48b	Shifter input
M2	48b	Shifter input

The Shifter input registers cannot be directly written or read, but are accessed indirectly by the Arithmetic instructions. These registers are used as the input for the Shifter. If written through addition operations the M-registers are sign extended from the 32-bit result, so that the effective data is in the lower 32 bits of the M-registers.

Register	Bits	Description
A0	32b	AU0 result register
A1	32b	AU1 result register
A2	32b	AU2 result register

The Arithmetic Unit result registers receive the final results from any arithmetic results. The A-registers can additionally be written through the General bus by the Immediate Load instructions. The `A2` register additionally receives input from the Logic Unit due to the Logic instructions or as a result of several special instructions through the General bus.

The values in the A-registers can be saturated to 8-bit values with the SATURATE instructions, and they can be tested for equality to zero with the ZERODETECT instruction, in which case the result of the test is written to the `STATUS` register. Normally any loading of the A-registers due to arithmetic instructions causes the MSB bits of the A-registers to be written to the `STATUS` register. As the data format for the Geometry Processor is 2's complement, the MSB is one if the data value is negative.

With the Immediate Load instructions there are two ways to write to the A-registers: either to the upper 24 bits or the lower 24 bits. When writing to the lower part, the MSB bits are sign extended to the MSB (bit 23) of the data written. When writing to the upper part of the A-registers, the lower 8 bits are set to zero.

4.5.3 Stream registers

Register	Bits	Description
RDADDR	24b	Stream Read address
WRADDR	24b	Stream Write address
STREAM(HI)	32b	Stream data high
STREAM(LO)	32b	Stream data low

The Stream Read address register (RDADDR) contains the address from where the next stream data item will be read by the next Stream Read operation. The address is automatically incremented by the Stream I/O Unit. When writing to this register the Stream I/O Unit flushes its read cache.

The Stream Write address register (WRADDR) contains the address where the next stream data item will be written by the next Stream Write operation. The address is automatically incremented by the Stream I/O Unit. When writing to this register the Stream I/O Unit flushes its write buffer.

The stream data registers, stream data high (STREAM(HI))and stream data low (STREAM(LO)), contain together the 64-bit data word to be written to the stream by a Stream Write instruction. These registers can be written to by the MOVE_REG and Stream Write instructions.

NOTE! The Stream registers are write only and cannot be read to the buses. See also the page 100.

4.5.4 Index registers

Register	Bits	Description
XRDBASE0	7b	X bus 0 read index register
XRDBASE1	7b	X bus 1 read index register
XRDBASE2	7b	X bus 2 read index register
YRDBASE0	7b	Y bus 0 read index register
YRDBASE1	7b	Y bus 1 read index register
YRDBASE2	7b	Y bus 2 read index register
WRBASE0	7b	Write bank 0 index register
WRBASE1	7b	Write bank 1 index register
WRBASE2	7b	Write bank 2 index register

The RAM Read Index registers are used to allow the programmer to perform indexed memory accesses. The value of the RDBASE registers are optionally added to any local data RAM addresses when reading the memory.

The RAM Write Index registers are used to allow the programmer to perform indexed memory accesses. The value of the WRBASE registers are optionally added to any local data RAM addresses when writing the memory. The index registers are associated with the local data memory banks with the same number i.e. XRDBASE0 is associated with the X read port of the data RAM 0.

NOTE! The registers are write-only and cannot be read to the buses.

The instruction set allows writing to the index registers in a combined way. For each of the groups (X, Y, Write) the 0 and 1 bus registers can be combined, and all registers in the group can be written at once.

4.5.5 Control Registers

Register	Bits	Description
N	6b	Shift value / Normalization value register
PC	14b	Program counter
JMPREG	14b	Calculated jump address register
REGBASE	10b	IO register address base register
VTMB	6b	Bus index register (special decoded 3b format)

The **N** register allows computed shifts to be used. The N register can be loaded from and stored via the bus. It can be used with the Shifters by specifying the N register usage with the Shift control bit in the A-register load operations. The N register can also be loaded with the Normalize operation. This operation finds the number of shifts required to normalize the number in the X2 register. (see Normalization Unit on the page 52)

The N register can be directly written and can be read via the A2 register using the A-register load operations.

The calculated jump address register (**JMPREG**) is used to allow the programmer to perform calculated branches. The value of the JMPREG can be written from the General bus, and then be used as the next value of the PC by using the Calculated Branch operation in the next instructions. Note that using Calculated Branches takes always two instructions to complete. The JMPREG also serves as the storage for return addresses for subrutine calls.

The JMPREG register can be directly written with the General Move instructions and can be read via the A2 register using the A-register load operations. See also Figure 4.3-1 on page 55.

The IO register address base register (**REGBASE**) contains the base address for the register writes. The value of the REGBASE register is added to any register addresses when performing OUT instructions.
NOTE! This register is write only and cannot be read to the buses.

The bus index register (**VTMB**) is used to specify which memory bank will be written by which A-register or which memory bank is used to read each X-register. The purpose is to algorithmically select RAM banks for writing, and this can be used for example when processing the vertices of a triangle currently being drawn in cases where the order of the vertices is not known and affects the algorithm.

The VTMB register values are decoded in a special way described in the Table 4.5-1.

dec	Bin	VT	VM	VB	VT	VM	VB
0	000	0	1	2	00	01	10
1	001	0	1	2	00	01	10*
2	010	0	2	1	00	10	01
3	011	2	0	1	10	00	01
4	100	1	0	2	01	00	10
5	101	1	2	0	01	10	00
6	110	0	1	2	00	01	10*
7	111	2	1	0	10	01	00

Table 4.5-1 Special decoded 3b format. *) Not possible as flags value for Derive VTMB.

The VTMB register can be directly written by the General Move instructions and can read via the A2 register using the A-register load operations.

4.5.6 Status register

The Geometry Processor has a STATUS register which contains the arithmetic flags and a flag describing the result of the BIT_TEST operations. Note that this is internal register of the Geometry Processor and has nothing to do with the status PCI register 48. The STATUS register is written into in two parts. The arithmetic operations affecting to the A-registers load the STATUS bits into the bits 0-2 of the STATUS register, and the Logic operations load LSB of the result into bit 3 of the STATUS register.

The STATUS register bits are allocated as follows:

	3	2	1	0
normal:	bit	neg2	neg1	neg0
zerodetect:	bit	zer2	zer1	zer0

The STATUS register cannot be directly written and can read via the A2 register using the A-register load operations.

4.6 Instruction Encoding

The Geometry Processor instructions are formed from several fields, and the total width of the instructions is 32 bits. All instructions have the same width, and execute in one clock cycle. Some operations, e.g. divide, can take more than one clock cycle to complete, but other instructions can be executed on parallel and can contain more than one parallel operation. The encoding of the instruction words is presented in the table below.

The basic instruction classes are:
* Arithmetic (AU#)
* Parallel Move (LOAD, SLOAD, SAVE and LOAD_SAVE)
* Logic (LOGIC)
* General Move (MOVE_REG, IMMED and SIMMED)
* Branch (BRANCH)
* Miscellaneous:
    - Out (OUT)
    - Stream Write and Read (RD_STRM, WR_STRM and SWR_STRM)
    - Special (SPEC)

<table>
<tr><th></th><th>31</th><th>30</th><th>29</th><th>28</th><th>27</th><th>26</th><th>25</th><th>24</th><th>23</th><th>22</th><th>21</th><th>20</th><th>19</th><th>18</th><th>17</th><th>16</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td colspan="30">LOAD</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td colspan="30">LOAD_SAVE</td></tr>
<tr><td>2</td><td>1</td><td>0</td><td>0</td><td colspan="5">OUT[5:1]</td><td colspan="6">AU6</td><td>O0</td><td colspan="17">SLOAD</td></tr>
<tr><td>3</td><td>1</td><td>0</td><td>1</td><td colspan="12">AU12</td><td colspan="17">SLOAD</td></tr>
<tr><td>4</td><td>1</td><td>1</td><td>0</td><td>0</td><td colspan="28">IMMED</td></tr>
<tr><td>5</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td colspan="27">AU</td></tr>
<tr><td>6</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td colspan="27">BRANCH</td></tr>
<tr><td>7</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>AS</td><td>FS</td><td colspan="6">AU6</td><td colspan="18">RD_STRM</td></tr>
<tr><td>8</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>MR0</td><td>FS</td><td colspan="6">MOVE_REG[6:1]</td><td colspan="18">RD_STRM</td></tr>
<tr><td>9</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>A2</td><td>A1</td><td colspan="6">AU6</td><td>A0</td><td colspan="17">SLOAD</td></tr>
<tr><td>10</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>A2</td><td>A1</td><td colspan="6">AU6</td><td>A0</td><td colspan="17">SAVE</td></tr>
<tr><td>11</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td colspan="2">SPE10</td><td colspan="12">AU12</td><td>AS</td><td colspan="5">SPEC[6:2]</td><td colspan="6">OUT</td></tr>
<tr><td>12</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td colspan="6">AU6</td><td colspan="18">SIMMED</td></tr>
<tr><td>13</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td colspan="12">AU12</td><td>AS</td><td colspan="5">SPEC[6:2]</td><td colspan="2">SPE10</td><td colspan="4">WR_STRM</td></tr>
<tr><td>14</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td colspan="12">AU12</td><td>AS</td><td colspan="7">MOVE_REG</td><td colspan="3"></td><td>WS</td></tr>
<tr><td>15</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td colspan="17">AU_14[0+1+2]</td><td>OS</td><td colspan="6">OUT</td></tr>
<tr><td>16</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td colspan="7">MOVE_REG</td><td colspan="17">LOGIC</td></tr>
<tr><td>17</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td colspan="7">MOVE_REG</td><td colspan="17">SLOAD</td></tr>
<tr><td>18</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td colspan="3"></td><td colspan="4">WR_STRM</td><td colspan="17">SLOAD</td></tr>
<tr><td>19</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td colspan="7">SPEC</td><td colspan="17">SLOAD</td></tr>
<tr><td>20</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td colspan="7">SPEC</td><td colspan="17">SAVE</td></tr>
<tr><td></td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td colspan="24">reserved</td></tr>
<tr><td></td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td colspan="24">reserved</td></tr>
<tr><td></td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td colspan="24">reserved</td></tr>
<tr><td></td><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
</table>

The identification numbers on the first column in the table above are referred to on the more detailed tables to be presented on the next few chapters.

4.6.1 Arithmetic instructions

		31	30	29	28	27	26	25	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
AU	5						A-LOAD2					AU_OP2				R_LOAD2			ML2		AU_OP01				SH	A_LD01			R_LD01			ML01	
AU_14[0+1+2]	15									A2	A1	AU_OP				R_LOAD			ML		A_LOAD_14				SH	A0							
AU12	3				R_LOAD			A12_OP			T12	AU_OP				SH																	
AU12[2/012]	11, 13, 14									SH	T12	AU_OP				R_LOAD			A12_OP			AS											
AU6	2, 12									T6	A6_OP				R6																		
AU6[0+1+2]	9, 10							A2	A1	T6	A6_OP				R6	A0																	
AU6[2/012]	7							AS		T6	A6_OP				R6																		

The numbers in the first column refer to rows in the table on page 64.

Full Arithmetic instructions specify different operations for AUs 0/1 and AU 2. They consist of AU opcode, A-load, M-load and R-load parts. See AU row on table above.

Full Arithmetic instructions for single AUs specify same operations for all the AUs. They consist of AU opcode, A-load, M-load and R-load parts. See row AU_14.

Short Arithmetic instructions specify the same A-load, M-load, and R-load operations for all the AUs. The M-load is a shortened version of the full M-load. See row AU12.

6-bit Arithmetic instructions are the shortest version of the AU instructions. They do not allow the most exotic possibilities of the AUs, and all the AUs execute the same operations. Also the whole instructions is coded together in a 6-bit operation word. See row AU6.

NOTES:
* It is possible to specify in A_LOAD and M_LOAD either AU output or R-register as input. If the selection is not the same for both of the cases, or if one of them is not a NOP, the result is unspecified.

* It is possible to specify the A/Y selection in both AU_OP and R_LOAD. If the selection is not the same for both, or if one of them is not a NOP, the result is unspecified.

* For 6-bit AU, mode 0 operations, note that the Multiplication multiplies the corresponding AU operation operands.

Field descriptions:

AS
Selecting between AU*[2/012] is done using the AS select bit.

Dec	Bin	Operation
0	0	Use only AU2 for the operation
1	1	Use all the AUs for the operation

A0, A1, A2
When selecting individual AUs to perform the AU operations the A0, A1, and A2 bits are used.

Dec	Bin	Operation
0	0	Do not use this AU for any operations
1	1	Use this AUs for the operation

SH
SHIFT select (N_reg/0)

Dec	Bin	Operation
0	0	No SHIFT
1	1	Use N register for shifting

4.6.1.1 Arithmetic Unit opcode

This field selects the operation of the adder in the Arithmetic Unit.

AU_OP, AU_OP2, AU_OP01

Dec	Bin	Operation
0	0000	NOP
1	0001	X+A
2	0010	X-A
3	0011	-X+A
4	0100	X+Y
5	0101	X-Y
6	0110	-X+Y
7	0111	A
8	1000	-A
9	1001	X
10	1010	-X
11	1011	Y
12	1100	-Y
13	1101	0
14	1110	inc(A)
15	1111	dec(A)

4.6.1.2 A-load

These fields select the source of data to be loaded into the A-register.

A_LOAD2
Select source for loading the A2 register.

Dec	Bin	Operation
0	00000	NOP
1	00001	AU result
2	00010	M2
3	00011	hi(R2)
4	00100	lo(R2)
5	00101	Saturate 8b signed
6	00110	Saturate 8b unsigned
7	00111	-M0-M1
8	01000	M2
9	01001	M1
10	01010	M0
11	01011	M0-M1
12	01100	M0-M2
13	01101	M1-M2
14	01110	M0+M1+M2
15	01111	-M0-M1+M2
16	10000	-M2
17	10001	-M1
18	10010	-M0
19	10011	M0+M1
20	10100	-M0+M1
21	10101	M0+M2
22	10110	-M0+M2
23	10111	M1+M2
24	11000	-M1+M2
25	11001	-M1-M2
26	11010	-M0-M2
27	11011	M0-M1+M2
28	11100	M0+M1-M2
29	11101	-M0+M1+M2
30	11110	M0-M1-M2
31	11111	-M0+M1-M2

A_LOAD_14
Select source for loading to A-registers with AU_14 instruction.

Dec	Bin	Operation	Note
0	0000	NOP	
1	0001	AU result	
2	0010	M	
3	0011	hi(R)	
4	0100	lo(R)	
5	0101	Saturate 8b signed	
6	0110	Saturate 8b unsigned	
7	0111	`-M0-M1`	(2)
8	1000	`M2`	
9	1001	`M1`	(2)
10	1010	`M0`	(2)
11	1011	`M0-M1`	(2)
12	1100	`M0-M2`	(2)
13	1101	`M1-M2`	(2)
14	1110	`M0+M1+M2`	(2)
15	1111	`-M0-M1+M2`	(2)

(2) This operation is available only on the AU2. For other AU's it is a NOP.

A_LOAD01
Select source for loading to `A0` and `A1` registers.

Dec	Bin	Operation
0	000	NOP
1	001	AU result
2	010	M
3	011	hi(R)
4	100	lo(R)
5	101	Saturate 8b signed
6	110	Saturate 8b unsigned
7	111	*reserved*

4.6.1.3 M-load

These fields select the source for loading into the M-registers.

M_LOAD, M2_LOAD, M01_LOAD

Dec	Bin	Operation
0	00	NOP
1	01	AU result
2	10	R
3	11	*reserved*

4.6.1.4 R-load

These fields specify the source to be loaded into the R-registers. They also specify the multiplication operation.

R_LOAD, R_LOAD2, R_LOAD01

Dec	Bin	Operation	Notes
0	000	NOP	
1	001	AU result	
2	010	hi(X×A)	Multiply higher 24-bits of the operands
3	011	hi(X×Y)	Multiply higher 24-bits of the operands
4	100	lo(X×A)	Multiply lower 24-bits of the operands
5	101	lo(X×Y)	Multiply lower 24-bits of the operands
6	110	*reserved*	
7	111	*reserved*	

4.6.1.5 AU12

Special fields for the 12-bit Arithmetic instructions.

T12
This field specifies the mode of the AU12 instruction.

Dec	Bin	Operation
0	0	AU12 mode 0: A=AU/R; R=R_LOAD
1	1	AU12 mode 1: A=func(`M0,M1,M2`); M=R; R=R_LOAD

A12_OP
This field specifies the operation to be performed by the AU12 instruction.

AU12 mode 0:

Dec	Bin	Operation
0	000	NOP
1	001	AU result
2	010	M
3	011	hi(R)
4	100	lo(R)
5	101	Saturate 8b signed
6	110	Saturate 8b unsigned
7	111	*reserved*

AU12 mode 1:

Dec	Bin	Operation	Note
0	000	`A=M2`	
1	001	`A=M1`	(2)
2	010	`A=M0`	(2)
3	011	`A=M0-M1`	(2)
4	100	`A=M0-M2`	(2)
5	101	`A=M1-M2`	(2)
6	110	`A=M0+M1+M2`	(2)
7	111	`A=-M0-M1+M2`	(2)

(2) This operation is available only on the AU2. For other AU's it is a NOP.

4.6.1.6 6-bit AU operations (AU6)

Special fields for the 6-bit Arithmetic instructions. **A0, A1 and A2** are described earlier.

T6
This field specifies the operation to be performed by the AU6 instruction.

AU6 mode select:

Dec	Bin	Operation
0	0	Mode 0: A=misc; R=MUL
1	1	Mode 1: A=vect/M; M=R; R=MUL

A6_OP
AU6 operation code, field specifying the operation to be performed by the AU6 instruction.

Mode 0: (A=misc; R=MUL)

Bit(s)	Function
3-0	A-load:

Dec	Bin	Operation	Note
0	0000	NOP	no R-load
1	0001	A=X+A	R=X × A
2	0010	A=X+Y	R=X × Y
3	0011	A=X-Y	R=X × Y
4	0100	A=-X+Y	R=X × Y
5	0101	A=X	R=X × Y
6	0110	A=-X	R=X × Y
7	0111	A=Y	R=X × Y
8	1000	A=-Y	R=X × Y
9	1001	A=0	R=X × Y
10	1010	*reserved*	
11	1011	A=inc(A)	R=X × A
12	1100	A=dec(A)	R=X × A
13	1101	A=hi(R)	R=X × Y
14	1110	A=lo(R)	R=X × Y
15	1111	*reserved*	

Mode 1: (A=vect/M; M=R; R=MUL)

Bit(s)	**Function**
3-1	A-load (MUL = X × Y):

Dec	Bin	Operation	Note
0	000	A=M	
1	001	A=M1	(2)
2	010	A=M0	(2)
3	011	A=M0-M1	(2)
4	100	A=M0-M2	(2)
5	101	A=M1-M2	(2)
6	110	A=M0+M1+M2	(2)
7	111	A=-M0-M1+M2	(2)

(2) This operation is available only on the AU2. For other AU's it is a NOP.

0 SHIFT select (N_reg/0):

Dec	Bin	Operation
0	0	No SHIFT
1	1	Use N register for shifting

R6
AU6 R-load operation

Dec	Bin	Operation	Notes
0	0	R=lo(MUL)	Multiply lower 24-bits of the operands
1	1	R=hi(MUL)	Multiply higher 24-bits of the operands

NOTE! A/Y selection for multiplication is based on the A-load or Y if not otherwise specified.

4.6.2 Parallel Move instructions

These instructions provide means for moving data between local data RAM and the AU registers.

4.6.2.1 Load instructions

<table>
<tr><td></td><td></td><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>LOAD</td><td>0</td><td>0</td><td>0</td><td colspan="6">X-sel</td><td>XI</td><td colspan="5">X01-addr</td><td colspan="5">Y01-addr</td><td colspan="3">Y-sel</td><td colspan="5">X2-addr</td><td colspan="5">Y2-addr</td></tr>
<tr><td>SLOAD</td><td>2,3,9,17,18,19</td><td colspan="15"></td><td colspan="3">XS-sel</td><td>XI</td><td colspan="3">Y-sel</td><td colspan="5">X-addr</td><td colspan="5">Y-addr</td></tr>
</table>

The numbers 0 for LOAD and 2, 3, 9, 17, 18 and 19 for SLOAD refer to rows in the table on page 64.

4.6.2.1.1 Full load (LOAD)

XI

Bit(s)	**Function**
0	R-register indexed load select 0 = fixed buses 1 = indexed buses (VT,VM,VB)

X-Sel

Bit(s)	**Function**
5-4	X2 source select:
3-2	X1 source select:
1-0	X0 source select:

X2 source select (bits 5-4):

XI	**0**	**1**
Xsel		
00	NOP	NOP
01	B0	VT
10	B1	VM
11	B2	VB

X1 source select (bits 3-2):

XI	**0**	**1**
Xsel		
00	NOP	NOP
01	B0	VT
10	B1	VM
11	B2	VB

X0 source select (bits 1-0):

XI	**0**	**1**
Xsel		
00	NOP	NOP
01	B0	VT
10	B1	VM
11	B2	VB

Y-sel

Bit(s)	Function
2	Y2 source select 0 - NOP 1 - Load from associated bus (B2->Y2)
1	Y1 source select 0 - NOP 1 - Load from associated bus (B1->Y1)
0	Y0 source select 0 - NOP 1 - Load from associated bus (B0->Y0)

X01-addr
This field specify the addresses to be used in the X-port of the data memories 0 and 1.

Bit(s)	Function
4	X0,X1 index register select 0=Don't use index register (accessing addresses 0-15) 1=Use index register for memory address generation X0 uses `XRDBASE0`, X1 uses `XRDBASE1`
3-0	Memory address for X0 and X1 moves

X2-addr
This field specifies the address to be used in the X-port of the data memory 2.

Bit(s)	Function
4	X2 index register select 0=Don't use index register (accessing addresses 0-15) 1=Use index register for memory address generation Uses `XRDBASE2`.
3-0	Memory address for X2 move

Y01-addr
This field specify the addresses to be used in the Y-port of the data memories 0 and 1.

Bit(s)	Function
4	Y0,Y1 index register select 0=Don't use index register (accessing addresses 0-15) 1=Use index register for memory address generation Y0 uses `YRDBASE0`, Y1 uses `YRDBASE1`
3-0	Memory address for Y0 and Y1 moves

Y2-addr
This field specifies the address to be used in the Y-port of the data memory 2.

Bit(s)	Function
4	2 index register select 0=Don't use index register (accessing addresses 0-15) 1=Use index register for memory address generation Uses `YRDBASE2`.
3-0	Memory address for Y2 move

4.6.2.1.2 Short LOAD(SLOAD)

XI

Bit(s)	Function
0	X bus indexed mode select 0 = Mode 0, fixed buses 1 = Mode 1, indexed buses (VT,VM,VB)

We have two distinct cases for the **XS-sel** field depending on the indexed mode select value:

XS-sel
X-register source select.

Mode 0 - direct:

Bit(s)	Function
2	X2 select 0 - NOP 1 - Load from associated bus (B2->X2)
1	X1 select 0 - NOP 1 - Load from associated bus (B1->X1)
0	X0 select 0 - NOP 1 - Load from associated bus (B0->X0)

Mode 1 - indexed buses:

Bit(s)	Function
2	X2 select 0 - NOP 1 - Load from VT bus (B[VT]->X2)
1	X1 select 0 - NOP 1 - Load from VM bus (B[VM]->X1)
0	X0 select 0 - NOP 1 - Load from VB bus (B[VB]->X0)

X-addr
This field specifies the address to be used in the X-port of the data memories.

Bit(s)	Function
4	X0,X1,X2 index register select 0=Don't use index register (accessing addresses 0-15) 1=Use index register for memory address generation X0 uses `XRDBASE0`, X1 uses `XRDBASE1`, X2 uses `XRDBASE2`
3-0	Memory address for X0 and X1 moves

Y-addr
This field specifies the address to be used in the Y-port of the data memories.

Bit(s)	Function
4	Y0,Y1,Y2 index register select 0=Don't use index register (accessing addresses 0-15) 1=Use index register for memory address generation Y0 uses `YRDBASE0`, Y1 uses `YRDBASE1`, X2 uses `YRDBASE2`
3-0	Memory address for Y0 and Y1 moves

4.6.2.2 SAVE

<table>
<tr><td></td><td></td><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>LOAD_SAVE</td><td>1</td><td>0</td><td>1</td><td colspan="6">A-sel</td><td>AI</td><td colspan="5">X01-addr</td><td colspan="5">W01-addr</td><td colspan="3">XW-sel</td><td colspan="5">X2-addr</td><td colspan="5">W2-addr</td></tr>
<tr><td>SAVE</td><td>10, 2</td><td colspan="15"></td><td colspan="6">A-sel</td><td>AI</td><td colspan="5">W01-addr</td><td colspan="5">W2-addr</td></tr>
</table>

4.6.2.2.1 SAVE

AI

Bit(s)	Function
0	A-register indexed mode select 0 = Mode 0, fixed buses 1 = Mode 1, indexed buses (VT,VM,VB)

We have two distinct cases for the A-sel field depending on the indexed mode select value:

A-sel
Select which A-register drives which write port of the data memories.

Mode 0 - direct:

Field	Bits	Notes
B2-driver	2b	
B1-driver	2b	
B0-driver	2b	
A-sel	6b	Total

Bit(s)	Function
5-4	B2-driver:

Dec	Bin	Operation
0	00	NOP
1	01	`A0` drives the bus
2	10	`A1` drives the bus
3	11	`A2` drives the bus

3-2 B1-driver:

Dec	Bin	Operation
0	00	NOP
1	01	`A0` drives the bus
2	10	`A1` drives the bus
3	11	`A2` drives the bus

1-0 B0-driver:

Dec	Bin	Operation
0	00	NOP
1	01	`A0` drives the bus
2	10	`A1` drives the bus
3	11	`A2` drives the bus

Mode 1 - indexed:

Field	Bits	Notes
Bus index select	3b	
Write enable	3b	
A-sel	6b	Total

Bit(s)	Function
5-3	Bus index select:

idx	VT	VM	VB
000	A0	A1	A2
001	A0	A2	A1
010	A1	A0	A2
011	A1	A2	A0
100	A2	A1	A0
101	A2	A0	A1
110	A2	A2	A1
111	A1	A2	A2

Bit(s)	Function
2	Write to the VT bus
1	Write to the VM bus
0	Write to the VB bus

W01-addr
This field specifies the address to be used in the write port of data memories 0 and 1.

Bit(s)	Function
4	Bus 0 and 1 index register select 0=Don't use index register (accessing addresses 0-15) 1=Use index register for memory address generation Bus 0 uses `WRBASE0`, bus 1 uses `WRBASE1`
3-0	Memory address for bus 0 and 1 moves

W2-addr
This field specifies the address to be used in the write port of data memory 2.

Bit(s)	Function
4	Bus 2 index register select 0=Don't use index register (accessing addresses 0-15) 1=Use index register for memory address generation Uses `WRBASE2`.
3-0	Memory address for bus 2 move

4.6.2.2.2 Combined LOAD and SAVE (LOAD_SAVE)

XW-sel
X-register source select.

Bit(s)	Function
2	X2 select 0 - NOP 1 - Load from associated bus (B2->X2)
1	X1 select 0 - NOP 1 - Load from associated bus (B1->X1)
0	X0 select 0 - NOP 1 - Load from associated bus (B0->X0)

Other fields as specified in previous instructions.

4.6.3 Logic instructions

Logic instructions are implemented by driving directly the control signals for the Logic Unit. See chapter Logic Unit on page 51 for more information on the Logic Unit implementation.

These are performed by the Logic unit, and include all normal two and one operand logic functions, bit-field extraction and copying operations, and bit-test and set operations. The result of the Logic operations is put into the A2 register.

The SETBIT operation sets bits of the value read from the X2 register. The CLRBIT operation clears bits of the value read from the X2 register. The bits to be set or cleared are indicated by the value of the shift field. The COPYBIT operation copies a bit-field from the Y2 register into the value of X2 register. The field in the Y2 register starts from the value indicated by the shift value and is mask bits long. The field in the X2 register that is to be copied into is at the same location.

		31–17	16–10	9–5	4–0
LOGIC	16		LOGIC_OP	SHIFT	MASK

Bit(s)	Function
16	**DSHIFT** 0 = no shift, 1 = shift down by shift.
15	**INV** 0 = don't invert, 1 = invert data.
14	**XOR** 0 = X OR Y, 1 = X XOR Y
13:12	**YSEL**

Dec	Bin	Operation
0	00	Y = Y2
1	01	Y = 0
2	10	Y = Y2 AND bitmask
3	11	Y = NOT (bitmask)

Bit(s)	Function
11:10	**XSEL**

Dec	Bin	Operation
0	00	X = X2 AND Y2
1	01	X = 0
2	10	X = X2 AND NOT (bitmask)
3	11	X = X2 AND bitmask

Bit(s)	Function
9-5	**SHIFT** Bits to shift for EXTRACT (5 bits -> 0-31 down).
4-0	**MASK** Number of bits to mask (5 bits -> 0-31 bits).

Examples of the useful Logic instructions are presented in the table below, see schematic for more exotic operations. The first three single bin bits present, in the order, DSHIFT, INV and XOR. The next two bin bit series present the YSEL and XSEL.

Bin	Operation	Notes
0 0 0 01 00	AND	
0 1 0 01 00	NAND	
0 0 0 00 10	OR	
0 1 0 00 10	NOR	
0 0 1 00 10	XOR	
0 1 1 00 10	XNOR	
0 1 0 01 10	NOT X	
0 1 0 00 11	NOT Y	
0 0 0 01 10	PASS X	(mask must be 0)
0 0 0 00 11	PASS Y	
1 0 0 01 11	EXTRACT X	
1 0 0 01 11	BIT_TEST X	
1 0 0 10 01	EXTRACT Y	
1 0 0 10 01	BIT_TEST Y	
0 0 0 10 10	COPYBIT(X,Y)	
0 0 0 11 10	SETBIT(X)	
0 0 0 01 10	CLRBIT(X)	
0 0 1 11 11	NEXTRACT	EXTRACT (not X)
0 0 0 01 11	MASK_AND X	
0 0 0 10 01	MASK_AND Y	

4.6.4 General Move instructions

General Move instructions are responsible for moving data other ways than what is possible using the Parallel Move instructions, i.e. not between AU registers and the data memory.

With these operations it is possible to store or load any register in the Geometry Processor (accessible from the buses) to from any other register or memory, and load any register with an immediate data. It is also possible to move data from the A-registers to the stream memory with these operations. The special registers can be loaded from the A-registers using these operations. The amount of data to be moved in one operation is limited with these operations.

NOTE! If a Normalize operation (NORM) is performed at the same time than a move to the `N` register, the `N` register will be written with the results of the NORM operation, and NOT the value which was moved.

4.6.4.1 Move from A-registers to other GP registers

NOTE: If using MOVE_REG together with RD_STRM, and using the X-register targets in register move, the result is unspecified. For the NOP the MOVE_REG fields `SRC` and `REG_DST` must both be zeros.

		S1	S0	REG_DST
MOVE_REG	8	bit 25	bit 23	bits 22–18
MOVE_REG	14	bit 10	bit 9	bits 8–4
MOVE_REG	16, 17	bit 17	bit 23	bits 22–18

SRC [S1,S0]

Dec	Bin	Operation
0	00	NOP
1	01	`A0`
2	10	`A1`
3	11	`A2`

REG_DST

Dec	Bin	Operation	Notes
0	00000	NOP	
1	00001	A0	
2	00010	A1	
3	00011	A2	
4	00100	RDADDR	Stream Read address, 24b
5	00101	WRADDR	Stream Write address, 24b
6	00110	JMPREG	
7	00111	X012	Writes to all X-registers.
8	01000	X0	
9	01001	X1	
10	01010	X2	
11	01011	XRDBASE0	
12	01100	XRDBASE1	
13	01101	XRDBASE2	
14	01110	XRDBASE01	Writes to XRDBASE0 and XRDBASE1
15	01111	XRDBASE012	Writes to all the XRDBASES
16	10000	YRDBASE0	
17	10001	YRDBASE1	
18	10010	YRDBASE2	
19	10011	YRDBASE01	Writes to ÝRDBASE0 and YRDBASE1
20	10100	YRDBASE012	Writes to all the YRDBASES
21	10101	WRBASE0	
22	10110	WRBASE1	
23	10111	WRBASE2	
24	11000	WRBASE01	Writes to WRBASE0 and WRBASE1
25	11001	WRBASE012	Writes to all the WRBASES
26	11010	N	
27	11011	REGBASE	
28	11100	VTMB	Special decoded 3b format, see table below.
29	11101	STREAM(HI)	Stream data high
30	11110	STREAM(LO)	Stream data low
31	11111	*reserved*	

4.6.4.2 Immediate data loads

Immediate Loads move constant data specified in the instruction to Geometry Processor registers.

		31	30	29	28	27	26	25	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
IMMED	4					REG_LONG				IMMED-data 24																							
SIMMED	12																REG_SHORT					IMMED-data 13												

4.6.4.2.1 IMMED

This specifies the Long Immediate Load instruction source and destination.

IMMED-data24
IMMED-data24 is a signed 24-bit data word, which is sign extended for longer registers, and truncated from upper bits for shorter registers.

REG_LONG:

Dec	Bin	Operation	Notes
0	0000	NOP	
1	0001	A0_lower	32b
2	0010	A1_lower	32b
3	0011	A2_lower	32b
4	0100	RDADDR	Stream Read address, 24b
5	0101	WRADDR	Stream Write address, 24b
6	0110	JMPREG	14b
7	0111	X012	Writes to all X-registers.
8	1000	X0	32b (load lower 24b)
9	1001	X1	32b (load lower 24b)
10	1010	X2	32b (load lower 24b)
11	1011	A0_upper	32b
12	1100	A1_upper	32b
13	1101	A2_upper	32b
14	1110	*reserved*	
15	1111	*reserved*	

4.6.4.2.2 SIMMED

This specifies the source and destination for the Short Immediate Load instruction.

IMMED-data13
IMMED-data13 is a signed 13-bit data word, which is sign extended for longer registers, and truncated from upper bits for shorter registers.

REG_SHORT:

Dec	Bin	Operation	Notes
0	00000	NOP	
1	00001	*reserved*	
2	00010	*reserved*	
3	00011	*reserved*	
4	00100	*reserved*	
5	00101	*reserved*	
6	00110	*reserved*	
7	00111	*reserved*	
8	01000	*reserved*	
9	01001	*reserved*	
10	01010	*reserved*	
11	01011	`XRDBASE0`	
12	01100	`XRDBASE1`	
13	01101	`XRDBASE2`	
14	01110	XRDBASE01	Writes to `XRDBASE0` and `XRDBASE1`
15	01111	XRDBASE012	Writes to all the `XRDBASES`
16	10000	`YRDBASE0`	
17	10001	`YRDBASE1`	
18	10010	`YRDBASE2`	
19	10011	YRDBASE01	Writes to `YRDBASE0` and `YRDBASE1`
20	10100	*reserved*	
21	10101	`WRBASE0`	
22	10110	`WRBASE1`	
23	10111	`WRBASE2`	
24	11000	WRBASE01	Writes to `WRBASE0` and `WRBASE1`
25	11001	WRBASE012	Writes to all the `WRBASES`
26	11010	`N`	
27	11011	`REGBASE`	IO register address base register
28	11100	`VTMB`	Special decoded 3b format, see table below.
29	11101	*reserved*	
30	11110	*reserved*	
31	11111	*reserved*	

`VTMB` register bit encoding:

dec	bin	VT	VM	VB	VT	VM	VB
0	000	0	1	2	00	01	10
1	001	0	1	2	00	01	10
2	010	0	2	1	00	10	01
3	011	2	0	1	10	00	01
4	100	1	0	2	01	00	10
5	101	1	2	0	01	10	00
6	110	0	1	2	00	01	10
7	111	2	1	0	10	01	00

4.6.5 Branch instructions

There are two major Branch categories: Jumps and Subroutine Calls. It is possible to use either unconditional or conditional branches. Conditional branches use the STATUS register to evaluate the branch conditions. The STATUS register contains the sign bits of all the A-registers in the AUs, and the BIT_TEST flag from the Logic Unit. Also the branches can be direct or indirect, in which case the branch address is taken from the JMPREG register. All the branches are delayed.

When doing a CALL operation, the return address is saved into the JMPREG. The return address will be *2 + the address of the CALL*, because of the delayed branching. The next instruction following the CALL operation is executed before executing the first instruction of the subroutine. Doing more than one CALL operation after each other without a corresponding return operation between them will overwrite the contents of the JMPREG, and returns after the first one to transfer the control to the same location. It is possible to save the value of the JMPREG to do multilevel CALLs. In this case it is on the programmer's responsibility to save and restore the correct values from and into the JMPREG.

The JUMP condition is evaluated in the following way:

```
calc_mask = (status ^ inv_mask) | ~use_mask;   /* create mask */
if (inv_cond == 1) {
   do_jump = @~&calc_mask;                     /* bitwise NAND reduction
*/
} else {
   do_jump = @&calc_mask;                      /* bitwise AND reduction */
}
```

where the *AND_reduce* function ANDs all the bits from the calc_mask together producing a single value.

For programming the following special cases may be useful:

```
JUMP always:   use_mask = 0000
               inv_cond = 0

JUMP never:    use_mask = 0000
               inv_cond = 1
```

Examples:

```
JUMP N2:       use_mask = 0100
               inv_mask = 0000
               inv_cond = 0

JUMP not(N2):  use_mask = 0100
               inv_mask = 0100
               inv_cond = 0

JUMP N2 & N1:  use_mask = 0110
               inv_mask = 0000
               inv_cond = 0
```

JUMP not(N2) & not(N1):
```
use_mask = 0110
inv_mask = 0110
inv_cond = 0
```

JUMP N2 | N1:
```
use_mask = 0110
inv_mask = 0110
inv_cond = 1
```

JUMP not(N2) | not(N1):
```
use_mask = 0110
inv_mask = 0000
inv_cond = 1
```

Program address space:
Maximum address space for program memory is 14 bits. The on chip memory is 512 words divided into 4 banks that are cached from the out of chip memory (zero address is set by `CODEBASE`). Writing to the `CODEBASE` register from the Geometry Prosessor allows one to have more that one logical address space within the external memory. This can be used to extend the effective program memory beyond 14 bits (16384 words). See also the page 54.

Using the Precache instruction:
There is an A/advice bit within the branch instruction. The advice bit tells that this is not a real branch to be taken, but it advices the cache system to pre-load the cache contents for a forthcoming branch instruction, to save a cache miss. This feature can be used to greatly reduce the amount of time spent in waiting instruction memory cache updates. Only valid information for advice type branch is the immediate address.

Hint: Since the return from the subroutine takes the return address from the `JMPREG`, it is possible to perform non-conditional calculated jumps by using the RETURN operation.

		31	30	29	28	27	26	25	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
BRANCH	6						A	TYPE		IC	USE_MASK				INV_MASK				X	BRANCH-addr													

Bit(s)	Function
26	**A** Advice bit: 0 = Normal branch instruction. 1 = Precache instruction, do not branch.
25-24	**TYPE** Branch type

Dec	Bin	Operation
0	00	immediate jump
1	01	calculated jump
2	10	immediate call
3	11	calculated call

Bit(s)	Function
23	**IC** (INV_COND) Invert condition, invert evaluated branch condition meaning. See before for description of usage.
22-19	**USE_MASK** Condition code usage mask. See before for description of usage.
18-15	**INV_MASK** Condition code negation mask. See before for description of usage.
14	*reserved*
13-0	**BRANCH-address** Not used for calculated (`JMPREG`) branches.

4.6.6 Miscellaneous instructions

These instructions include all the rest of the instruction that was not included into the previous classes. These are OUT, Stream Write, Stream Read and Special instructions.

4.6.6.1 OUT instruction

IO register address space is 10 bits, and the address of the register to be written to is formed always by adding the current value of REGBASE to the address supplied in the instruction. See also chapter 4.7.2.

There is one special case (type 15) where the OUT operation is conditional to the state of the OS bit in the out instruction. In all other cases if the OUT operation is part of the instruction it will be done with no regards to the OUT address.

		31	30	29	28	27	26	25	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
OUT	15																										OS	OUT-addr					
OUT	11																											OUT-addr					
OUT	2				OUT[5:1]											O0																	

NOTE! Register writes can be performed only from A2 register.
OS

Dec	Bin	Operation
0	0	Don't do any OUT operations
1	1	Do the specified OUT operation

4.6.6.2 Stream Write and Stream Read

These instructions read and write data from/to the stream. See also chapter 4.7.4.

		31 30 29 28 27 26 25	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
RD_STRM	8, 7		FS				F_X2		X2_mode	F_X1	X1-mode	F_X0	X0-mode		
WR_STRM	13												SRC	SF	WS
WR_STRM	18				SRC	SF	WS								
SWR_STRM	14														WS

4.6.6.2.1 Stream Write (WR_STRM and SWR_STRM)

This instruction writes 32 bits of data to the Stream registers (`STREAM(HI)` and `STREAM(LO)`) and possibly initiates a write to the external graphics memory.

SRC
Source register

Dec	Bin	Operation
0	00	NOP
1	01	`A0`
2	10	`A1`
3	11	`A2`

SF
Data format

Dec	Bin	Operation
0	0	lower 32-bit
1	1	upper 32-bit

WS
Do write

Dec	Bin	Operation
0	0	Don't write stream data out
1	1	Write stream data out

4.6.6.2.2 Stream Read (RD_STRM)

The stream data is stored into a stream fetch register which is a 64-bit register. The stream fetch register is visible through the Stream Read instruction. The fetch register can be read in various formats. The general view is illustrated below:

63							0	
7	6	5	4	3	2	1	0	byte (8 bits)
3		2		1		0		short (16 bits)
1				0				int (32 bits)
I	24b			0				misc

NOTE: If using MOVE_REG together with RD_STRM, and using the X-register targets in register move, the result is unspecified.

Bit(s)	Function
24	**FS** Fetch Stream. Fetch new data to the stream fetch register after reading this value.
23-18	<Unused for RD_STRM>
17-16	**F_X2** Data format for X2 read

Dec	Bin	Operation
0	00	signed
1	01	unsigned
2	10	fixed
3	11	special

Bit(s)	Function
15-12	**X2-mode** Mode for X2 read

Dec	Bin	Numeric	Special
0	0000	NOP	NOP
1	0001	24b	status 1
2	0010	int 1	status 0
3	0011	int 0	
4	0100	short 3	
5	0101	short 2	
6	0110	short 1	
7	0111	short 0	
8	1000	byte 7	8-bit mult by 16 (I)
9	1001	byte 6	8-bit mult by 8 (I)
10	1010	byte 5	8-bit mult by 4 (I)
11	1011	byte 4	8-bit mult by 2 (I)
12	1100	byte 3	float conv -> integer (bits 63-32)
13	1101	byte 2	float conv -> integer (bits 31-0)
14	1110	byte 1	float conv -> fixed (bits 63-32)
15	1111	byte 0	float conv -> fixed (bits 31-0)

Bit(s)	Function
11-10	**F_X1** Data format for X1 read, bit assignments equal to F_X2.
9-6	**X1-mode** Bit assignments equal to X2-mode.
5-4	**F_X0** Data format for X0 read, bit assignments equal to F_X2.
3-0	**X0-mode** Bit assignments equal to X2-mode.

4.6.6.3 Special instructions

This set of operations perform a specialized set of instructions. These include return from subroutine, absolute value compare, etc. The common thing with these operations is that they perform a not very often needed functionality which is done with special functional blocks and does not need any parameters to execute.

		31–26	25–24	23–17	16–11	10–6	5–4	3–0
SPEC	19, 20			SPEC				
SPEC	13					SPEC[6:2]	SPE10	
SPEC	11		SPE10			SPEC[6:2]		

4.6.6.3.1 RETURN instruction

The RETURN instruction returns from a subroutine previously entered with a CALL operation. The `PC` is restored from the `JMPREG`, and the next instruction fetched will be the instruction that is at address *2 + CALL address* (because of the delayed branching). The next instruction following the RETURN operation is executed before returning to the main program. Since the return from the subroutine takes the return address from the `JMPREG`, it is possible to perform non-conditional calculated jumps by using the RETURN operation. See also Figure 4.3-1 on page 55.

4.6.6.3.2 Normalize instruction

The Normalize instruction (NORM) takes the number to normalize from the `X2` register and the normalization shift amount will be placed in the `N` register such that it is easy to perform the actual normalization step by just shifting the number by the value of `N` register.

4.6.6.3.3 Derive VTMB instruction

The Derive VTMB instruction performs the bus index sort operation required in the triangle draw algorithm. It reads the `STATUS` register and uses that to calculate the value for the internal `VTMB` register. The instruction causes the index of the largest value to be written into the VT-part, the index of the middle item into the VM-part and the index of the smallest into the VB-part of the register. The instruction is explained in the table below:

b0<b1	**b1<b2**	**b0<b2**	**top**	**mid**	**bot**	**order**
0	0	0	b0	b1	b2	b0>=b1>=b2
0	0	1	-	-	-	N/A
0	1	0	b0	b2	b1	b0>=b2>b1
0	1	1	b2	b0	b	b2>b0>=b1
1	0	0	b1	b0	b2	b1>b0>=b2
1	0	1	b1	b2	b0	b1>=b2>b0
1	1	0	-	-	-	N/A
1	1	1	b2	b1	b0	b2>b1>b0
AU2	AU1	AU0				

Where the values b0, b1, and b2 correspond to indices 0, 1, and 2 correspondigly. The correct instruction sequence is:

```
                               ; Sort vertices
                               ; a2 = b1-b0, a1 = b2-b1, a0 = b2-b0
x2 = b1[ty], x1 = b2[ty], x0 = b2[ty] !
a = x !
x2 = b0[ty], x1 = b1[ty], x0 = b0[ty] !
a = a - x !

                               ; see table above
d_vtmb !                       ; i.e. a2 = status ! vtmb = a2 !
nop !                          ; nop needed for vtmb change

x2 = vt[ty], x1 = vm[ty], x0 = vb[ty] !

            ; after this sequence it is known that:
            ; x2 <= x1 <= x0, i.e. vt has the lowest value.
```

4.6.6.3.4 Division instruction

The Division instruction performs hardware division. The Hardware Division block is capable to perform a 32-bit by 32-bit division with 32 bits result or 24-bit by 24-bit division giving 24 bits result. The remainder is also available for use. The performed division is a signed integer by unsigned integer division of `X2` by `Y2`. The result of the division will be available after 16 or 12 clock cycles respectively, and can be read into the `A2` register using another of the SPEC instructions. The division block is not pipelined, and thus it is NOT possible to start a new divisions every clock cycle. If new divisions are not started the result of the division will remain for loading at the output of the division block. Refer to the instructions 8 - 15 on table below.

For further information see the chapter on page 52.

4.6.6.3.5 Special AU instructions

Special AU instructions allow loading values of `STATUS`, `N`, and `JMPREG` to `A2`; saturation of A-register values to 8-bit signed and unsigned format; absolute value operations; and comparison of A-register values to zero. Refer to the instructions 16 - 39 on table below.

Defined SPEC instructions:

Dec	Bin	Operation	Note
0	0000000	NOP	
1	0000001	RETURN	
2	0000010	*reserved*	
3	0000011	*reserved*	
4	0000100	Normalize	N=NORM(X2)
5	0000101	*reserved*	
6	0000110	Derive VTMB from STATUS register.	
7	0000111	*reserved*	
8	0001000	Start divide 32-bit X2/Y2	
9	0001001	Start divide 24-bit X2/Y2	
10	0001010	Start divide 32-bit X/(stored)	
11	0001011	Start divide 24-bit X/(stored)	
12	0001100	A2 <= Quotient	
13	0001101	A2 <= Remainder (*)	
14	0001110	*reserved*	
15	0001111	*reserved*	
16	0010000	A2 <= STATUS	
17	0010001	A2 <= N	
18	0010010	A2 <= JMPREG	
19	0010011	*reserved*	
20	0010100	Saturate A012 8b unsigned	
21	0010101	Saturate A012 8b signed	
22	0010110	*reserved*	
23	0010111	*reserved*	
24	0011000	A=abs(X)	
25	0011001	A=abs(A)	
26	0011010	A=abs(Y)	
27	0011011	A=-abs(X)	
28	0011100	A=-abs(A)	
29	0011101	A=-abs(Y)	
30	0011110	A=A+abs(X)	
31	0011111	A=A-abs(X)	
32-39	0100AAA	Zero detect (A2, A1, A0)	
40-47	0101xxx	*reserved*	
48-55	0110xxx	*reserved*	
56-63	0111xxx	*reserved*	
64-127	1xxxxxx	*reserved*	

(*) Remainder is not always valid/useful. Algorithm requires restore step for negative remainders, see example below. Remainder is not valid for cases abs(X)<Y

```
   a2 = remainder !      ; Get actual remainder, may be negative
   jmp(4,4,positive) !   ; Fix negative remainder, otherwise ready
   x2 = b2[divider] !    ; Get divider (delay slot)
   a2 = a2 + x2 !
   a2 = a2 + x2 !        ; 2*remainder + 2*divider
positive:
                         ; A2 now has the remainder value times 2
```

4.7 Geometry Processor External Interface

4.7.1 General information

The Geometry Processor has two interfaces: the Stream I/O and the Register Out bus, see chapter Geometry Processor Bus Structure on page 47. The Stream I/O interface allows the processor to either access the on card memory (read and write) or read data from the host computer through the PCI bus (Stream Read address 0). The Register Out bus allows the processor to write to most of the PCI accessible registers of the VS25203B chip.

The non-accessible registers are:

Register address range	**Function**
20-29	VGA shadow registers
42-55	System control registers

NOTE! There is a delay from issuing the writes on the Register Out bus to the time the values are visible from the PCI registers due to the internal delays of the PCI block and the inherent nature of the PCI bus.

4.7.2 Geometry Processor Interface PCI Register Description

The following registers are available for both the host computer and the Geometry Processor. From the Geometry Processor, the registers can be written with the OUT instruction. The only registers that can be read from the Geometry Processor are the `status_reg_in` (register 194, at stream status 0), and `data_in` (register 196, at stream status 1).

synchronization	**register 192**	**offset 300h**

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
rs	wait											stream_ref			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
					y_ref										

Fields

Field	Bits	Description
rs	31	Geometry Processor reset bit (ge_reset)
wait	30	Geometry Processor wait bit
stream_ref	19:16	Stream reference
y_ref	10:0	Y reference

The `ge_reset` and `wait` bits are normal register bits, i.e. they must be cleared if they have been set. This limits their usability from the Geometry Processor side.

`Y_ref` and `stream_ref` are threshold levels for the screen refresh line counter and the Stream address 0 FIFO respectively. See also chapter Direct stream data on page 101.

code_config	register 193	offset 304h

Format

31	30–17	16
ena		

15–0
CODEBASE

Fields

Field	Bits	Description
ena	31	Program memory enable
CODEBASE	16:0	Geometry Processor code base

The CODEBASE field is padded with 6 LSB zeros to get the actual word address where the code is located in the card memory. See also Chapter 4.7.3.

status_reg_in	register 194	offset 308h

Format

31–27	26–16
	video_y_coord

15	14	13–12	11	10–8	7	6	5	4	3	2	1	0
	pv		gpi		gpf	gp0	blti	vc	id1	id2	ok1	ok2

Fields

Field	Bits	Description
video_y_coord	26:16	Video y coordinate
pv	14	Pixel visible
gpi	11	Geometry Processor interrupt active
gpf	7	Geometry Processor flag
gp0	6	Geometry Processor stream0 flag
blti	5	Block Transfer Unit idle
vc	4	video compare
id1	3	Primitive processor idle
id2	2	Pixel processor idle
ok1	1	Primitive processor init ok
ok2	0	Pixel processor init ok

Status_reg_in register is visible on the Geometry Processor stream interface as status 0. The fields are read-only except fields pv, gpi and gpf that are read/write fields. This is nearly the same register as status register (48).

video_y_coord
Video y coordinate. Current video refresh scanline.
pv
Pixel visible. This bit is set to one when a visible pixel has been detected by the pixel processor in the `zread` operation. The bit is reset by writing a value "1" into this field. Refer to the `grid_reg` (102) register.
gpi
PCI Geometry Processor interrupt active. The interrupt can be caused from the Geometry Processor by writing a value "1" to this bit. This bit should be set back to value "0" after a while, because it is not an automatic operation. This interrupt is reset from the `status` register (48).
gpf
Geometry Processor flag.
gp0
Geometry Processor stream 0 flag.
blti
Block Transfer Unit idle. Indicates status of the Block Transfer unit.
1 idle
0 busy
vc
This bit is one when the `video_y_coord` field value is equal or greater than the `video_y_ref` value of the `ref_reg` , register 49**.**
id1
Primitive processor idle. This bit is one when the primitive processor is in the idle state.
id2
Pixel processor idle. This bit is one when the pixel processor is in the idle state.
ok1
Primitive processor initialization ok. This bit is one if initial values are allowed to be written to the primitive processor.
ok2
Pixel processor initialization ok. It is used for finding out when the pixel processor can be initialized. In VS25203, it is given by `id1` and `id2`.

data_in	**register 196**	**offset 310h**

Format

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

Fields

Field	Bits	Description
data_in	31:0	Data transferring to Geometry Processor

`Data_in` register is PCI writable register for transferring data to Geometry Processor.

It is visible on the Geometry Processor stream interface as status 1 and it can also be written to via the Register Out bus interface.

data_out	register 197	offset 314h

Format

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

Fields

Field	Bits	Description
data_out	31:0	Data transferring from Geometry Processor

The `data_out` register is PCI readable register for transferring data from Geometry Processor and it can be written to via the register bus out interface.

4.7.3 Geometry Processor instruction code interface

The instruction code for the Geometry Processor resides originally in the card memory. The code size is 2^{14} = 16384 = 16 kwords and the word size is 32 bits. Since the card memory is 64 bits wide, the code size there is 8 kwords. The 32-bit instructions are packed in little-endian order, i.e. the word at address with LSB=0 is at the LSB part of the 64-bit word.

Address @ card	Instruction addresses	
CODEBASE+8191	16383	16382
.	.	.
.	.	.
.	.	.
CODEBASE+1	3	2
CODEBASE	1	0
	64 31	32 0

The program memory is cached in the instruction cache which is a 4-way set associative cache with four 128-word blocks. There is currently no other way than jumping through 4 banks to flush the cache. The user should be careful when setting the `CODEBASE` register.

NOTE! The cache is not coherent with the program memory on card.

The recommended way of setting the `CODEBASE` is as follows:
1) set the `gp_reset` bit
2) write the `CODEBASE` register
3) reset the `gp_reset` bit

This causes the instruction cache to start in the initial state, and all the Geometry Processor registers to be reset, however the data memories retain their values.

See also the chapter Instruction Execution on page 54.

4.7.4 Geometry Processor Stream I/O interface

4.7.4.1 Stream I/O

The Stream I/O is controlled by two pointer registers: `RDADDR` and `WRADDR`, which are write-only registers. Writing to these registers sets the corresponding card memory 64 -bit **word** address where the next stream operation will access data. These registers are internally self incrementing. The address 0 is **special** for reading the stream. It causes the stream fetches to fetch data from values supplied through the PCI interface. Also in the address 0 case the **read** address pointer register is **not** self incrementing.

The stream data to be read is stored into a stream fetch register which is a 64-bit register. The value of this registers remains constant until the next time the Stream Read instruction (RD_STRM) sets the fetch stream bit (see `FS` bit on page 91). The stream will have the new data ready for reading at the next instruction. You can have at most every second instruction fetching the stream. The stream fetch register is visible through the Stream Read instruction. The fetch register can be read in various formats. The general view is illustrated below:

<table>
<tr><td colspan="4">63</td><td colspan="4" align="right">0</td><td></td></tr>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>byte (8 bits)</td></tr>
<tr><td colspan="2">3</td><td colspan="2">2</td><td colspan="2">1</td><td colspan="2">0</td><td>short (16 bits)</td></tr>
<tr><td colspan="4">1</td><td colspan="4">0</td><td>int (32 bits)</td></tr>
<tr><td>I</td><td colspan="3">24b</td><td colspan="4">0</td><td>misc</td></tr>
</table>

The numeric formats can use either signed or unsigned integers or signed fixed point numeric formats. The special values are considered to be unsigned except for the floating point numbers.

Dec	Numeric	Special
0	NOP	NOP
1	24b	status 1
2	int1	status 0
3	int0	
4	short3	
5	short2	
6	short1	
7	short0	
8	byte7	8-bit mult by 16 (I)
9	byte6	8-bit mult by 8 (I)
10	byte5	8-bit mult by 4 (I)
11	byte4	8-bit mult by 2 (I)
12	byte3	float conv → integer (bits 63-32)
13	byte2	float conv → integer (bits 31-0)
14	byte1	float conv → fixed (bits 63-32)
15	byte0	float conv → fixed (bits 31-0)

The stream is written through two Geometry Processor special registers: `STREAM(HI)` and `STREAM(LO)`. These Stream registers can be written multiple times, and new data overrides the old one. The data is sent to the stream only when the `WS` bit of the Stream Write instruction is set to 1.

4.7.4.2 Direct stream data

In order to transfer data from the PCI bus to the Geometry Processor, the VS25203B contains a FIFO buffer, with room for 16 data words, each 64 bits wide. This buffer is read by the Geometry Processor by using the Stream Read mechanism and by targeting the reads to the stream address 0. Stream address mechanism also does not auto increment when the reads are done to the address 0, this means that after the read address has been changed to 0 all the following reads will be done from the FIFO until an explicit read address change is done.

The data is written to the FIFO by writing it to the register range 224-255. It does not matter which addresses in this range are used (except for the following even/odd restriction), in any case the data is added to the next position in the FIFO. As the stream consists of 64-bit words, two register writes are needed to generate one stream word. The less significant 32 bits of the word should be written to an even register address and the most significant 32 bits should be written to an odd register address. The less significant 32 bits of the 64-bit word should be written first then the most significant 32 bits. A range of register addresses is used (instead of single address) so that efficient PCI burst writes can be used when adding multiple data words to the FIFO.

In addition to the PCI writes the FIFO can be filled by using the bus mastering mechanism, for further information see chapter PCI Bus starting on page 18.

The FIFO reads are controlled with the basic Stream Read mechanism. This means that the Geometry Processor will stall if it tries to read from an empty FIFO.

There is no similar hardware protection against writing too much data to the FIFO, instead the status of the FIFO is monitored, and this information can be used to control either the software writing to the FIFO or to control the bus master command stream which is filling the FIFO. The FIFO status monitoring is based on two register fields: `stream_ref` which gives the reference value for how many items should be in the FIFO and `stream0_flag` which is set to 1 if the FIFO contains more than `stream_ref` elements and 0 otherwise. The `stream0_flag` is available for the Geometry Processor in register 194 (bit 6). It is also available in register 48, and can be used to control PCI bus master jump and wait commands, see pages 21 and 23.

Because of internal pipelining the actual filling of the FIFO might be delayed relative to the PCI bus write operation. In order to compensate this the `stream0_flag` in register 48 is set to 1 while the internal pipelines contain data. This can cause the `stream0_flag` to be 1 unexpetedly, but the total effect is to protect the FIFO from being overfilled because of the pipeline delays.

5. Primitive Processor

5.1 Overview

The **VS_VP** Primitive Processor is responsible for converting primitives into individual pixels, which are then sent to the Pixel Processor. The primitives can be rectangles, triangles or lines, but in all cases they are described in the same way: The shape of the primitive is specified using edges (`edge0`, `edge1`, `edge2`, pages from 116 to 119) and the minimum and maximum Y-coordinates (registers `y_init, y_end`, page 123), and the contents by giving coefficients for equations that specify the different properties of the pixels inside the edges. The edges are always straight lines, but the pixel properties can be interpolated with either linear interpolation or with perspective correction.

The Primitive Processor handles complete primitives so that there is no need to split them into more simple constructs (such as trapezoids). This makes the initialization process both simpler and faster. A primitive is initialized by loading all the necessary values into the registers of the Primitive Processor. When the last register (`y_end`, maximum Y-coordinate, page 123) is loaded, the rasterization process begins. Because the registers are double buffered, it is possible to start the loading of the next primitive at the same time as the previous one is being rasterized. These registers also preserve their values when the triangle is rasterized. It is therefore unnecessary to reload values which do not change between consecutive triangles.

The Primitive Processor operates in the screen coordinate space; it produces the screen coordinates, z-depth, and up to eight perspective corrected interpolated values for each pixel. Of the eight interpolated values, four have an accuracy of eight bits and are thus suitable for color and transparency values (RGBT, pages from 106 to 109). The other four values have twelve bits of accuracy and are suitable to be used as texture coordinates (ATU/ATV and BTU/BTV, pages from 110 to 113).

Texture coordinates are unsigned 12-bit quantities. The hardware does not handle negative texture coordinates. The Primitive Processor also includes some texture address component manipulation (`grid_reg`)

Basic Formulas

The following formulas describe the edges of a primitive:

$$E0 = (y \times (edge0\_dy \times 8)) + (x \times (edge0\_dx \times 8)) + edge0\_init$$
$$E1 = (y \times (edge1\_dy \times 8)) + (x \times (edge1\_dx \times 8)) + edge1\_init$$
$$E2 = (y \times (edge2\_dy \times 8)) + (x \times (edge2\_dx \times 8)) + edge2\_init$$

where:

x = amount of horizontal pixels, relative to `x_init`.
y = amount of vertical pixels, relative to `y_init`.
Edge deltas dx and dy signify the change of the edge function within the distance of one *subpixel* unit (one eighth of a pixel). Therefore calculation must be done in subpixel units.
Edge0-edge2 are edge interpolators from pages 117 to 119.
See also the **Edge Ordering** section on page 116.

Edge functions are referred with indices from `edge_order` register, page 116. All the following statements have to be true, for a pixel to be inside a primitive:

$$e(left\_1) \geq 0$$

$$e(left\_2) \geq 0$$

$$e(right\_1) > 0$$

$$e(right\_2) > 0$$

e0, e1 and e2 functions give positive or negative results. Positive means that the pixel is inside a primitive, negative means outside, respectively. Note that for the left edges, zero means inside a primitive.

The following formulas are calculated per pixel to perform perspective correction. Next four are for colors and transparency:

$$R = \frac{(x \times r\_dx) + (y \times r\_dy) + r\_init}{\left[(x \times p\_dx) + (y \times p\_dy) + p\_init\right] \times \frac{1}{2^8}}$$

$$G = \frac{(x \times g\_dx) + (y \times g\_dy) + g\_init}{\left[(x \times p\_dx) + (y \times p\_dy) + p\_init\right] \times \frac{1}{2^8}}$$

$$B = \frac{(x \times b\_dx) + (y \times b\_dy) + b\_init}{\left[(x \times p\_dx) + (y \times p\_dy) + p\_init\right] \times \frac{1}{2^8}}$$

$$T = \frac{(x \times t\_dx) + (y \times t\_dy) + t\_init}{\left[(x \times p\_dx) + (y \times p\_dy) + p\_init\right] \times \frac{1}{2^8}}$$

and the next four are for A and B texture interpolators; note the coefficient $\frac{1}{2^{12}}$ - this is because of the 12-bit result.

$$atu = \frac{(x \times atu\_dx) + (y \times atu\_dy) + atu\_init}{\left[(x \times p\_dx) + (y \times p\_dy) + p\_init\right] \times \frac{1}{2^{12}}}$$

$$atv = \frac{(x \times atv\_dx) + (y \times atv\_dy) + atv\_init}{\left[(x \times p\_dx) + (y \times p\_dy) + p\_init\right] \times \frac{1}{2^{12}}}$$

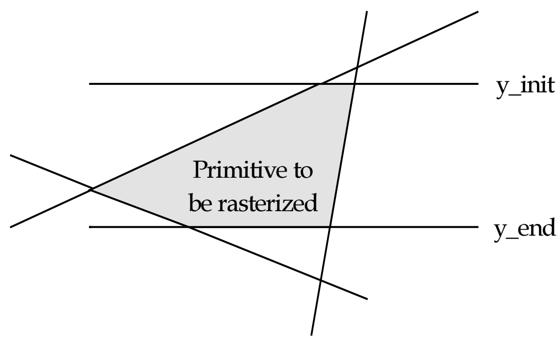
$$btu = \frac{(x \times btu\_dx) + (y \times btu\_dy) + btu\_init}{\left[(x \times p\_dx) + (y \times p\_dy) + p\_init\right] \times \frac{1}{2^{12}}}$$

$$btv = \frac{(x \times btv\_dx) + (y \times btv\_dy) + btv\_init}{\left[(x \times p\_dx) + (y \times p\_dy) + p\_init\right] \times \frac{1}{2^{12}}}$$

x = amount of horizontal pixels, relative to *x_init*.
y = amount of vertical pixels, relative to *y_init*.
p_dx, *p_dy* and *p_init* registers described in page 122.

Note that *dx* and *dy* specify the change of the proper value within the distance of one pixel and *x* and *y* are also pixel coordinates.

All in all: the primitive is defined with three edge functions and with the height of the primitive, (*y_end – y_init*)**.** All the pixels that are within the vertical bounds and have a non-negative value for left edge functions or a positive value for right edge functions are considered to be inside the primitive.



The `edge_order` register specifies the left and right edges. Tetragons can be rasterized by using `y_init` and `y_end` and two edge functions; lines are just narrow tetragons. Note that `y_end` must always be specified even if the edges define the lower end of a primitive. Generally `y_end` should be the first scanline that is not any more drawn.

The Primitive Processor itself does not use the properties in any way. It just performs the calculations described, and the properties finally have effect in the Pixel Processor where they are used to define the final color for the pixel in question. This means that the properties (such as depth) need not be the real values, but can instead be something completely different, if this is used to achieve special effects in the pixel pipeline.

5.2 Primitive Processor Registers

Register address	Offset	Register name
64	0100h	cr_init
65	0104h	cr_dy
66	0108h	cr_dx
67	010Ch	cg_init
68	0110h	cg_dy
69	0114h	cg_dx
70	0118h	cb_init
71	011Ch	cb_dy
72	0120h	cb_dx
73	0124h	ct_init
74	0128h	ct_dy
75	012Ch	ct_dx
76	0130h	atu_init
77	0134h	atu_dy
78	0138h	atu_dx
79	013Ch	atv_init
80	0140h	atv_dy
81	0144h	atv_dx
82	0148h	btu_init
83	014Ch	btu_dy
84	0150h	btu_dx
85	0154h	btv_init
86	0158h	btv_dy
87	015Ch	btv_dx
88	0160h	z_shr
89	0164h	z_init
90	0168h	z_dy
91	016Ch	z_dx
92	0170h	edge_order
93	0174h	edge0_init
94	0178h	edge0_dx
95	017Ch	edge0_dy
96	0180h	edge1_init
97	0184h	edge1_dx
98	0188h	edge1_dy
99	018Ch	edge2_init
100	0190h	edge2_dx
101	0194h	edge2_dy
102	0198h	grid_reg
103	019Ch	p_init
104	01A0h	p_dy
105	01A4h	p_dx
106	01A8h	x_init
107	01ACh	y_init
108	01B0h	y_end
109	01B4h	raster_ext

Note that _dx and _dy register ordering for edges differs from other interpolators.

5.2.1 Red Interpolator

The red interpolator (CR – color red) has three registers describing the values needed by the Primitive Processor. The `cr_init` value specifies the initial value of the red interpolator in the Primitive Processor. `cr_dy` specifies the increment which is added to the red value interpolator, when the Primitive Processor steps one pixel in Y-direction. `cr_dy` specifies the increment which is added to the red value interpolator, when the Primitive Processor steps one pixel in X-direction.

cr_init	register 64	offset 0100h

Format

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

Fields

Field	Bits	Description
cr_init	31:0	Initial value for the red interpolator

cr_dy	register 65	offset 0104h

Format

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

Fields

Field	Bits	Description
cr_dy	31:0	Red delta within one vertical pixel

cr_dx	register 66	offset 0108h

Format

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

Fields

Field	Bits	Description
cr_dx	31:0	Red delta within one horizontal pixel

5.2.2 Green Interpolator

The green interpolator (CG – color green) is similar to the red interpolator (CR) in all ways, except for the property being interpolated. The green interpolator uses three registers: `cg_init`, `cg_dy` and `cg_dx`.

cg_init	register 67	offset 010Ch

Format

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

Fields

Field	Bits	Description
cg_init	31:0	Initial value for the green interpolator

cg_dy	register 68	offset 0110h

Format

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

Fields

Field	Bits	Description
cg_dy	31:0	Green delta within one vertical pixel

cg_dx	register 69	offset 0114h

Format

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

Fields

Field	Bits	Description
cg_dx	31:0	Green delta within one horizontal pixel

5.2.3 Blue Interpolator

The blue interpolator (CB – color blue) is similar to the red interpolator (CR) in all ways, except for the property being interpolated. The blue interpolator uses three registers: `cb_init`, `cb_dy` and `cb_dx`.

cb_init	register 70	offset 0118h

Format

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

Fields

Field	Bits	Description
cb_init	31:0	Initial value for the blue interpolator

cb_dy	register 71	offset 011Ch

Format

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

Fields

Field	Bits	Description
cb_dy	31:0	Blue delta within one vertical pixel

cb_dx	register 72	offset 0120h

Format

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

Fields

Field	Bits	Description
cb_dx	31:0	Blue delta within one horizontal pixel

5.2.4 Transparency Interpolator

The transparency interpolator (CT – color transparency) is similar to the red interpolator (CR) in all ways, except for the property being interpolated. The transparency interpolator uses three registers: `ct_init`, `ct_dy` and `ct_dx`.

ct_init	register 73	offset 0124h

Format

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

Fields

Field	Bits	Description
ct_init	31:0	Initial value for transparency interpolator

ct_dy	register 74	offset 0128h

Format

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

Fields

Field	Bits	Description
ct_dy	31:0	Transparency delta within one vertical pixel

ct_dx	register 75	offset 012Ch

Format

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

Fields

Field	Bits	Description
ct_dx	31:0	Transparency delta within one horizontal pixel

5.2.5 A Texture U Interpolator (ATU)

The ATU interpolator is similar to the red interpolator (CR) in all ways, except for the property being interpolated. Even though the size of the output is different, the initialization process is the same, because a different scale factor is used in the perspective division. ATU interpolator uses three registers: `atu_init`, `atu_dy` and `atu_dx`.

atu_init	register 76	offset 0130h

Format

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

Fields

Field	Bits	Description
atu_init	31:0	Initial value for A texture U interpolator

atu_dy	register 77	offset 0134h

Format

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

Fields

Field	Bits	Description
atu_dy	31:0	A texture U delta within one vertical pixel

atu_dx	register 78	offset 0138h

Format

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

Fields

Field	Bits	Description
atu_dx	31:0	A texture U delta within one horizontal pixel

5.2.6 A Texture V Interpolator (ATV)

The ATV interpolator is similar to the red interpolator (CR) in all ways, except for the property being interpolated. Even though the size of the output is different, the initialization is the same, because a different scale factor is used in the perspective division. ATV interpolator uses three registers: `atv_init`, `atv_dy` and `atv_dx`.

atv_init	register 79	offset 013Ch

Format

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

Fields

Field	Bits	Description
atv_init	31:0	Initial value for A texture V interpolator

atv_dy	register 80	offset 0140h

Format

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

Fields

Field	Bits	Description
atv_dy	31:0	A texture V delta within one vertical pixel

atv_dx	register 81	offset 0144h

Format

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

Fields

Field	Bits	Description
atv_dx	31:0	A texture V delta within one horizontal pixel

5.2.7 B Texture U Interpolator (BTU)

The BTU interpolator is similar to the red interpolator (CR) in all ways, except for the property being interpolated. Even though the size of the output is different, the initialization is the same, because a different scale factor is used in the perspective division. BTU interpolator uses three registers: `btu_init`, `btu_dy` and `btu_dx`.

btu_init	register 82	offset 0148h

Format

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

Fields

Field	Bits	Description
btu_init	31:0	Initial value for B texture U interpolator

btu_dy	register 83	offset 014Ch

Format

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

Fields

Field	Bits	Description
btu_dy	31:0	B texture U delta within one vertical pixel

btu_dx	register 84	offset 0150h

Format

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

Fields

Field	Bits	Description
btu_dx	31:0	B texture U delta within one horizontal pixel

5.2.8 B Texture V Interpolator (BTV)

The BTV interpolator is similar to the red interpolator (CR) in all ways, except for the property being interpolated. Even though the size of the output is different, the initialization is the same, because a different scale factor is used in the perspective division. Btv interpolator uses three registers: `btv_init`, `btv_dy` and `btv_dx`.

btv_init	register 85	offset 0154h

Format

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

Fields

Field	Bits	Description
btv_init	31:0	Initial value for B texture V interpolator

btv_dy	register 86	offset 0158h

Format

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

Fields

Field	Bits	Description
btv_dy	31:0	B texture V delta within one vertical pixel

btv_dx	register 87	offset 015Ch

Format

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

Fields

Field	Bits	Description
btv_dx	31:0	B texture V delta within one horizontal pixel

5.2.9 Z Scale Factor

z_shr	register 88	offset 0160h

Format

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

Fields

Field	Bits	Description
z_shr	4:0	Scaling factor for z-depth calculations

The `z_shr` register specifies the amount of bits the Z interpolator is shifted to the right for Z-Buffering. Since the output Z property is only 24 bits, it is possible to gain extra accuracy to the interpolation by using a `z_shr` of 3. If the `z_shr` is dynamically calculated, even better accuracy is possible.

Maximum shift value is 24.

As `z_init` (register 89) is a 32-bit register, we can write to it a very accurate value of Z. `z_shr` is similar to SUBS in texture space. It is possible to compute dynamically which of the three corners of a triangle has the largest value (nearest), and use more fixed point bits to represent the Z value. As a result, we can choose the fixed point precision dynamically per triangle if we specify the number of bits to right-shift in order to get the actual Z value for Z buffering. Basically, we should OR every Z coordinate together and find the highest value "1" bit of the result. For example, if it is 26, we should multiply the vertex Z values by 16 and use 4 in the `z_shr` register to get the actual Z -value.

5.2.10 Z Interpolator

VS25203 uses "perspective correct" Z-buffer; i.e. using 1/Z values to perform Z buffering. The Z interpolator has three registers describing the depth value of a pixel needed by the Primitive Processor. `z_init` value specifies the initial value of the Z interpolator in the Primitive Processor. z_dy specifies the increment which is added to the Z value interpolator, when the Primitive Processor steps one pixel in Y-direction. z_dx specifies the increment which is added to the Z value interpolator, when the Primitive Processor steps one pixel in X-direction.

z_init	register 89	offset 0164h

Format

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

Fields

Field	Bits	Description
z_init	31:0	Initial value for the Z-depth register

z_dy	register 90	offset 0168h

Format

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

Fields

Field	Bits	Description
z_dy	31:0	Z-depth delta within one vertical pixel

z_dx	register 91	offset 016Ch

Format

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

Fields

Field	Bits	Description
z_dx	31:0	Z-depth delta within one horizontal pixel

5.2.11 Edge Ordering

edge_order	register 92	offset 0170h

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
								right_2		right_1		left_2		left_1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

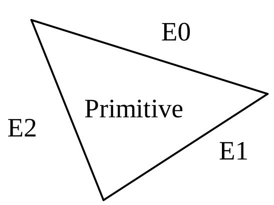
Fields

Field	Bits	Description
right_2	7:6	Second right edge slot (edge index)
right_1	5:4	First right edge slot (edge index)
left_2	3:2	Second left edge slot (edge index)
left_1	1:0	First left edge slot (edge index)

right_2
Specifies which of the edges is used as the second right edge of the primitive.
right_1
Specifies which of the edges is used as the first right edge of the primitive.
left_2
Specifies which of the edges is used as the second left edge of the primitive.
left_1
Specifies which of the edges is used as the first left edge of the primitive.

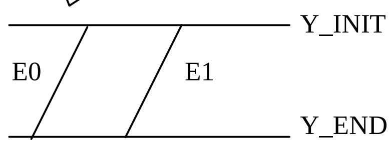
The `edge_order` register is used to specify which of the edges of a primitive are on the left side, and which on the right side. The Primitive Processor supports three edges in total, so one side uses two edges, and the other side uses the remaining one. The bitcode of the latter edge needs to be duplicated when writing it to appropriate fields; any field may not be empty. If a tetragon is to be drawn, two fields have to be duplicated. Upper and lower edges are defined with the registers `y_init`, and `y_end`, page 123. The value in `left1`, `left2`, `right1` and `right2` is a 2-bit number that specifies the index of the edge that belongs to the given slot. Value 0 means that the edge is described with registers `edge0_init`, `edge0_dx`, `edge0_dy`, and similarly with the other values. The number must be in the range of 0-2.(edge0, edge1, edge2 interpolators on pages 117-119.)
See also page 102.

Examples



Contents of fields:

RIGHT_2		RIGHT_1		LEFT_2		LEFT_1	
0	0	0	1	1	0	1	0



Contents of fields:

RIGHT_2		RIGHT_1		LEFT_2		LEFT_1	
0	1	0	1	0	0	0	0

5.2.12 Edge0 Interpolator

Primitive is formed with three edge interpolators. Edge0 interpolator is used to describe the 0th edge of primitive. `edge0_init` provides the initial value of edge interpolator 0. `edge0_dy` is added to the edge interpolator 0 value, when the Primitive Processor steps one pixel to the down. `edge0_dx` is added to the 0^{th} edge interpolator value, when the Primitive Processor steps one pixel to right. Due to subpixel resolution _dx and _dy values are multiplied by eight during interpolation.

edge0_init	register 93	offset 0174h

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
edge0_init															
edge0_init															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	Bits	Description
edge0_init	31:0	Initial value for edge0

edge0_dx	register 94	offset 0178h

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
edge0_dx															
edge0_dx															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	Bits	Description
edge0_dx	31:0	Edge0 delta within one horizontal pixel

edge0_dy	register 95	offset 017Ch

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
edge0_dy															
edge0_dy															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	Bits	Description
edge0_dy	31:0	Edge0 delta within one vertical pixel

5.2.13 Edge1 Interpolator

The Edge1 interpolator is similar to Edge0 interpolator in all ways, except for the number of the edge it controls. Edge1 interpolator uses three registers: `edge1_init`, `edge1_dx` and `edge1_dy`.

edge1_init	register 96	offset 0180h

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
edge1_init															
edge1_init															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	Bits	Description
edge1_init	31:0	Initial value for edge1

edge1_dx	register 97	offset 0184h

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
edge1_dx															
edge1_dx															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	Bits	Description
edge1_dx	31:0	Edge1 delta within one horizontal pixel

edge1_dy	register 98	offset 0188h

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
edge1_dy															
edge1_dy															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	Bits	Description
edge1_dy	31:0	Edge1 delta within one vertical pixel

5.2.14 Edge2 Interpolator

The Edge2 interpolator is similar to Edge0 interpolator in all ways, except for the number of the edge it controls. Edge2 interpolator uses three registers: `edge2_init`, `edge2_dx` and `edge2_dy`

edge2_init	register 99	offset 018Ch

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
edge2_init															
edge2_init															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	Bits	Description
edge2_init	31:0	Initial value for edge2

edge2_dx	register 100	offset 0190h

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
edge2_dx															
edge2_dx															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	Bits	Description
edge2_dx	31:0	Edge2 delta within one horizontal pixel

edge2_dy	register 101	offset 0194h

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
edge2_dy															
edge2_dy															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	Bits	Description
edge2_dy	31:0	Edge2 delta within one vertical pixel

5.2.15 Grid Register

grid_reg	register 102	offset 0198h

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
										bq	bnv	bnu	aq	anv	anu

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		cp		g11	g10	g01	g00			rhig					

Fields

Field	Bits	Description
rhig	5:0	Rendering screen height/32
g00	8	grid mask 00
g01	9	grid mask 01
g10	10	grid mask 10
g11	11	grid mask 11
cp	13	constant_perspective
anu	16	Anu
anv	17	Anv
aq	18	A texture quad loop
bnu	19	Bnu
bnv	20	Bnv
bq	21	B texture quad loop

g00, g01, g10, g11

g00	g10
g01	g11

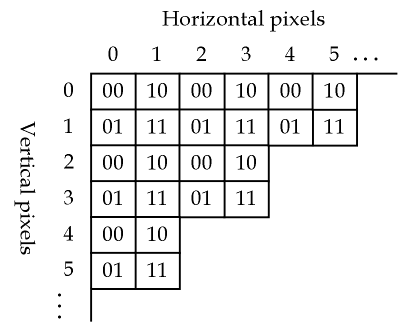
Primitive Processor uses 2 x 2 grid mask to enable or disable pixel visibility. When the field is set to one, the corresponding bit in Primitive Processor is disabled and vice versa. If adjacent fields are set to one, then the Primitive Processor ignores the processing of the corresponding horizontal line, (pairs g00,g10 and g01,g11). This doubles rasterizing speed, and is especially good for doing fast visibility checks using bit 14 in register 48 on page 39.

g11
Grid mask 11. Skips pixels with odd x and y coordinates. Notice that grid skip is especially effective if complete horizontal lines are skipped, otherwise rasterization proceeds at normal speed.
g10
Grid mask 10. Skips pixels with odd x and even y coordinates.
g01
Grid mask 01. Skips pixels with even x and odd y coordinates.
g00
Grid mask 00. Skips pixels with even x and y coordinates.

Grid mask works according to the following:

```
if (grid_mask[x and 1, y and 1]==1)
then kill_pixel
```

It is used, for example, to perform simple motion blur by enabling only one of the four pixels to be drawn every frame, and the bit is changed randomly. This gives the effect of having partial appearance of all four consecutive frames (a simple form of motion blur).



Primitive Processor tiles the whole screen with grid mask, like the diagram on above.

cp
Constant_perspective. If the `cp` = 1 then the perspective correction is constant for R, G, B and T interpolators. If `cp` = 0 then the perspective correction is performed by using P interpolator, refer to page 122.

anu,anv
A_not_Umsb; A_not_Vmsb.
If `anu` = 0, the msb of the A texture interpolator U coordinate is inverted. If `anv` = 0, the msb of the A texture interpolator V coordinate is inverted.
aq
A texture quad loop. Multiplies by four the texture coordinate values for the A texture. If this bit is set to one, loop range is quadruplicated with two step bitwise left shift. The downside of using quad looping is that the two resultant LSB bits are zero which means that only every fourth texel of a texture map can be sampled into the final output. This artifact is visible if a texture is looked at very closely, or if the texture contains some easily recognizable patterns, like text.

bnu,bnv
B_not_Umsb; B_not_Vmsb.
If bnu `= 0`, the msb of the B texture interpolator U coordinate is inverted. If `bnv = 0`, the msb of the B texture interpolator V coordinate is inverted.
bq
B texture quad loop. Multiplies by four the texture coordinate values for the B texture. If this bit is set to one, loop range is quadruplicated with a two step binary left shift.
rhig
Rendering screen height (in number of pixels) divided by 32. It is the height of the rendering area as multiples of 32-pixel blocks.

5.2.16 P Interpolator

When linear interpolation is desired, the P should be initialized to a constant value (normally 7FFFFFFFh for maximum accuracy). For more information on linear and perspective initializations, refer to the `cr_init` and `z_init` registers. The P interpolator uses three registers: p_init, p_dy and p_dx.

p_init	register 103	offset 019Ch

Format

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

Fields

Field	Bits	Description
p_init	31:0	Initial value for the P interpolator

p_dy	register 104	offset 01A0h

Format

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

Fields

Field	Bits	Description
p_dy	31:0	P delta within one vertical pixel

p_dx	register 105	offset 01A4h

Format

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

Fields

Field	Bits	Description
p_dx	31:0	P delta within one horizontal pixel

5.2.17 Start/End Coordinates

These registers specify the point from which the rasterization starts, and at the same time define the y extents for the primitive.

x_init	register 106	offset 01A8h

Format

31–16	
(unused)	

15–11	10–0
	x_init

Fields

Field	Bits	Description
x_init	10:0	Initial x coordinate of the rasterization process

This register is used for describing the starting point of the primitive to be rasterized. Optionally the register should contain the X coordinate of the leftmost visible pixel of the primitive on the row specified by `y_init`.

y_init	register 107	offset 01ACh

Format

31–16	
(unused)	

15–11	10–0
	y_init

Fields

Field	Bits	Description
y_init	10:0	Initial y coordinate of the rasterization process

The `y_init` register is used for describing the starting point of the primitive to be rasterized. It is the initial y coordinate where the triangle rasterization starts. The register should contain the Y coordinate of the first screen row that should be rasterized. Normally this is the row containing the first visible pixel of the primitive, but it is also possible to use larger Y value and in this way skip the topmost part of the primitive. The actual area covered by the triangle depends on the values of the `x_init` and `y_init` registers and on the edge parameters.

y_end	register 108	offset 01B0h

Format

31–16	
(unused)	

15–11	10–0
	y_end

Fields

Field	Bits	Description
y_end	10:0	Bottom row of the primitive

This register gives the maximum Y coordinate of the primitive. It is the first row that is not any more drawn. Writing the `y_end` signals the Primitive Processor that all the other registers are set and it should start processing a new triangle. This means that the `y_end` should be the last register written during the initialization of the primitive.

5.2.18 raster_ext Register

raster_ext	register 109	offset 01B4h

Format

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

Fields

Field	Bits	Description
rst	0	Soft reset for primitive processor

raster_ext (rasterize extra triangle) enables a soft reset. When 1 is written to this register, the Primitive Processor performs a soft reset, aborts the current triangle and starts the rasterization of the next triangle (if any).

6. Pixel Processor

6.1 Overview

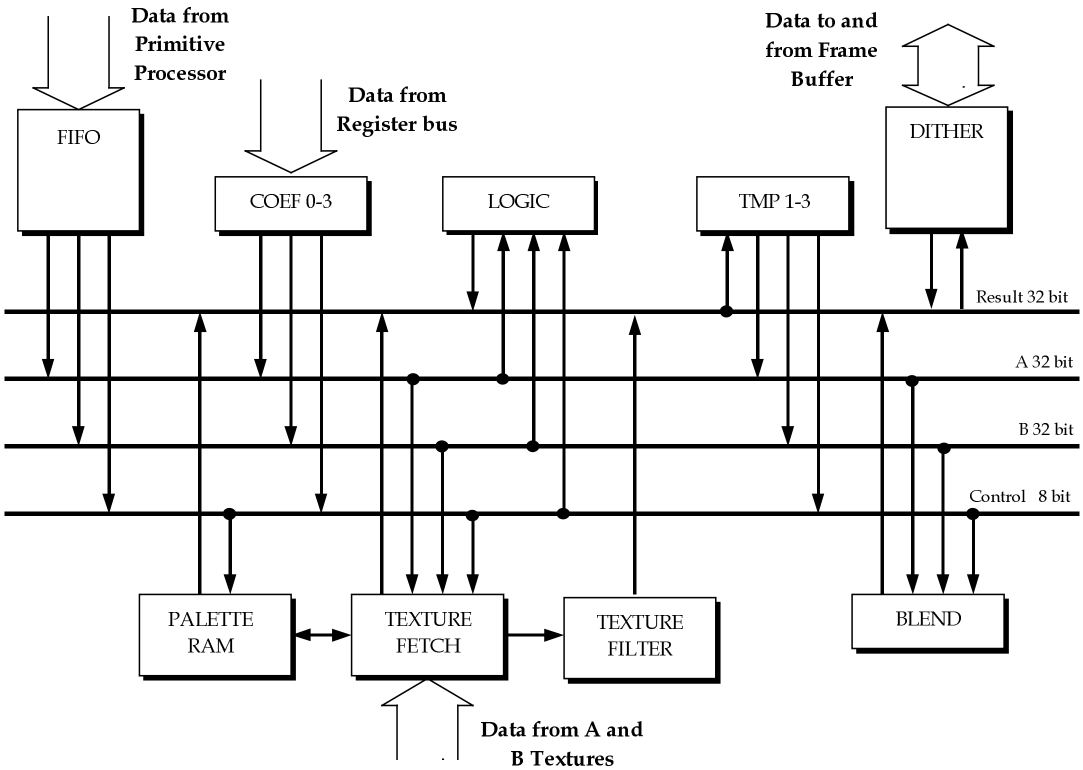
The Pixel Processor is responsible for calculating the final color for all pixels in a primitive. The color is generated by executing a shading program, which is written by the user (or invoked through the device driver). It is executed for each pixel, and combines data from registers, pixel properties in the Primitive Processor FIFO, texture maps and finally the old color in the frame buffer. The shading program is located in code memory of the Pixel Processor (see also Register Map on page 13). Code memory can store up to 32 commands, with many separate shading programs. It is possible to change the shading program for each primitive if needed by determining a new start address or by loading a completely new program to the code memory.

Pixel Processor must be initialized before starting rendering.

The pixel color for the primitive is formed from the color of the primitive surface and from the lighting of the environment. These are described by using registers, pixel properties and texture maps. The maps can be indexed color or true-color (TRGB, where T is most significant byte and B is least significant byte), and can be filtered with bilinear and trilinear filtering regardless of the mode. It is also possible to use two simultaneous textures so that one describes the surface and the other contains information about the lighting, for example, for shadows or highlights.

All these data can be combined by using logic operations and blending in a way controlled by the shading program. Finally, the pixel color can be combined with the old color of the pixel in order to create transparent surfaces or other effects. All these operations are carried out with full true-color accuracy, and the final result can be stored in 32-bit true-color, or 16 bit hi-color format. Dithering is also possible for better color quality.

6.2 Functional Block Diagram



The Pixel Processor consists of four central buses connected to the different functional units as seen above. The A and B buses are the main parameters of the operation, the Control bus is a modifier bus, and the Result bus is used to store the result into one of the three temporary registers (`TMP`1-3) or send to the screen. The main parameters can be colors (TRGB, 8 bits per component where T is the most significant byte and B is the least significant byte), or texture coordinates (UV, 12 bits per component). They can be read from temporary registers, coefficient registers or from the input FIFO fields. The modifier bus width is 8 bits, and the usage depends on the function. Every shading instruction specifies the functional unit to use, and the connections for each bus.

A central feature of the Pixel Processor is the availability of temporary registers. This makes it possible to construct complex shading operations.
The blend unit creates an intermediate color between the A and B colors according to a blending factor in the Control bus. It is also possible to alter the alive-flag of the pixel, based on the resulting transparency of the operation. The alteration can also be based on transparency dither, so that stipple-transparency is possible.

Texture fetch reads a texture color from the external graphics memory, given the texture coordinates in A-bus. The texture data in memory can use 4 to 32 bits per pixel, and the texture unit expands the storage format into full 32 bits. The 4 and 8 bit data are routed through a palette for this purpose. If MIP-mapping is used, the texture fetch operation also modifies the texture coordinate based on the active MIP-level, read from the control bus. The texture coordinates can be looped or clamped.

Texture filtering works together with the texture fetch unit. It fetches multiple texture colors which are combined according to the fractional texture coordinates for bilinear interpolation. Trilinear interpolation can be created with two texture filtering operations and one blend operation. When bilin filtering is used, 4 low order bits of the texture coordinates are used as the fractional bits.

The logic unit provides all possible logic operations between the two colors, as well as arithmetic operations and minimum and maximum operations.

6.2.1 Bus Address Table

Coefficient registers (COEF0-3) are read-only registers for the Pixel Processor; see registers 1 to 4, pages 141 and 142. TMP1-3 registers can be used to store temporary results in a shading program. T signifies transparency, R red, G green, B blue, ATU A texture U coordinate and ATV A texture V coordinate and BTU B texture U coordinate and BTV B texture V coordinate.

Address	**A bus**	**B bus**	**control bus**	**Destination**
0	COEF0 (TRGB)	COEF0 (TRGB)	COEF0 (T)	Frame buffer
1	COEF1 (TRGB)	COEF1 (TRGB)	COEF1 (T)	TMP1
2	COEF2 (TRGB)	COEF2 (TRGB)	COEF2 (T)	TMP2
3	COEF3 (TRGB)	COEF3 (TRGB)	COEF3 (T)	TMP3
4	FIFO (TRGB)	FIFO (TRGB)	FIFO (T)	-
5	TMP1 (TRGB)	TMP1 (TRGB)	TMP1 (T)	-
6	TMP2 (TRGB)	TMP2 (TRGB)	TMP2 (T)	-
7	TMP3 (TRGB)	TMP3 (TRGB)	TMP3 (T)	-
8	FIFO ATU/ATV	*reserved*	FIFO ATV (b.0-7)	-
9	FIFO BTU/BTV	FIFO BTU/BTV	FIFO BTV (b.0-7)	-
10	FIFO Z	*reserved*	TMP2 (R)	-
11	*reserved*	*reserved*	TMP2 (G)	-
12	*reserved*	*reserved*	TMP2 (B)	-
13	*reserved*	*reserved*	FIFO (R)	-
14	*reserved*	*reserved*	FIFO (G)	-
15	ZERO	*reserved*	FIFO (B)	-

Note that the coefficient registers can only be used by either A bus or B bus at the same time. Furthermore, A bus has higher priority than B bus.

Note also that for FIFO ATV and FIFO BTV on the C bus, there is a possibility of shifting the data as it is being transferred to the C bus. Bit selection is controlled by param -field bit 18 (see color_op or logic)

6.2.2 FIFO

Property	Width	Normal usage
CR	8 bits	Red
CG	8 bits	Green
CB	8 bits	Blue
CT	8 bits	Transparency, Blending factor, MIP-map level
ATU	12 bits	First texture X-coordinate
ATV	12 bits	First texture Y-coordinate
BTU	12 bits	Second texture X-coordinate
BTV	12 bits	Second texture Y-coordinate
Z	24 bits	Depth value used for Z-buffering

Red, Green Blue as well as texture coordinates can be used as diffuse, specular and fog intesity.

6.2.3 Coefficient Registers

Property	Width	Normal usage
COEF0	32 bits	Coefficient 0 (T:8, R:8, G:8, B:8)
COEF1	32 bits	Coefficient 1 (T:8, R:8, G:8, B:8)
COEF2	32 bits	Coefficient 2 (T:8, R:8, G:8, B:8)
COEF3	32 bits	Coefficient 3 (T:8, R:8, G:8, B:8)

6.2.4 Temporary Registers

Property	Width	Normal usage
TMP1	32 bits	Temp 1 register (T:8, R:8, G:8, B:8 or U:12, V:12)
TMP2	32 bits	Temp 2 register (T:8, R:8, G:8, B:8 or U:12, V:12)
TMP3	32 bits	Temp 3 register (T:8, R:8, G:8, B:8 or U:12, V:12)

These registers can be used to store temporary results in a shading program.

6.3 Shading Program Format

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
								end	**ppu_oper**				**param**		
param		**a_addr**				**b_addr**				**c_addr**				**dest**	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	**Bits**	**Description**
dest	1:0	Destination address
c_addr	5:2	C bus address
b_addr	9:6	B bus address
a_addr	13:10	A bus address
param	18:14	Parameter field
ppu_oper	22:19	Pixel processor operation
end	23	Shading program end

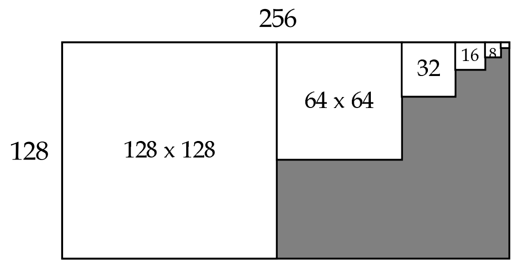
The program instruction contains seven different fields. Desired values are loaded to the appropriate fields, depending on the type of command (ppu_oper).
1) **end** determines the last command of the shading program.
2) **ppu_oper** determines the executed command:

0 reserved
1 color_op
2 stipple_blend
3 logic_op
4 cread
6 zread
7 zwrite
8 textfetch
9 textfetch_modulate
10 bilin
11 tlogic
12 palette

N.B. opcode 5 is reserved.

3) **param** contains parameter, depending on command
4) **a_addr** determines first source bus
5) **b_addr** determines second source bus
6) **c_addr** determines control bus
7) **dest** determines destination address

The Pixel Processor processes every pixel of the primitive, which is generated by the Primitive Processor. Shading program start address is loaded to register 14, page 149. Program run terminates when the value of the `end` field is one (maximum of 32 commands). If the shading program contains `ppu_oper` which may kill pixels (`zread`, `stipple_blend` or `tlogic`), then program execution may terminate and the Pixel Processor begins to process a new pixel of the primitive.



If MIP-mapping is used, the maximum width of the largest texture map is 512. Thus, the series of textures for a MIP-map fits into a combined texture width of 1024. For 128 × 128 texture, a 256 × 128 surface must be created. Smaller level of detail maps must be stored to the right of the highest level of detail map. Example of such a case is above. Note that the grey area is lost due to memory layout.

If MIP-map is enabled (am and bm bits in registers 6 and 8), the Pixel Processor uses four MSB bits from the C (Control) bus to determine what MIP level to use.

6.4 Shading Instructions

color_op **Shading Instructions** **opcode 1**

Description: Handles A and B bus colors with assistance of C (control) bus.
Result.red = A.red + ((B.red - A.red) × C) / 256
Result.green = A.green + ((B.green - A.green) × C) / 256
Result.blue = A.blue + ((B.blue - A.blue) × C) / 256
Result.transp = A.transp + ((B.transp - A.transp) × (C and F0h)) / 256

Special: Note that various parameter combinations are possible; see the example below with `param`=12. The transparency output is computed by only using the 4 topmost bits from the C bus.

Parameters:

0	00000	If C bus = 4 (FIFO) then use the whole transparency value for blend operation.
1	00001	If C bus = 4 (FIFO) then use 4 LSBs of the transparency for blend operation.
2	00010	For transparency component, force blend factor to 0.
4	00100	For color (RGB components), force blend factor to 0.
8	01000	Swap A and C (control) bus values.
16	10000	If C bus address is 8 or 9, value on C bus is shifted left 4 bits (0000xxxx -> xxxx0000).

Inputs: A and B buses contain colors and C bus contains the blend factor.

Outputs: Frame buffer, `TMP1`, `TMP2`, or `TMP3`

Example:

end	ppu_oper	param	a_addr	b_addr	c_addr	dest
1	1	12	0	4	1	0

Performs COEF0 × FIFO_RGB, and stores the result to the frame buffer. With parameter 0, the Result = A + ((B – A) × C) / 256. In the above example, parameters configure the blending unit to zero C bus and then swap A and C buses which makes Result = 0 + ((B – 0) × A) / 256 which is equal to (A × B) / 256. In other words, bit 2 and bit 3 of the parameter field are set (param = 8 + 4 = 12), and this generates A × B because:

Bit 2: forces color blend factor (C bus) to zero
Bit 3: swaps A bus and C bus values

The original formula of A + ((B - A) × C)) / 256 therefore becomes:

0 + ((B - 0) × A)) / 256 = (B × A) / 256

Consider the case of swapping the A bus value with C bus value by setting bit 3 of the `color_op` parameter. Note that C bus only carries 8 bits and A bus carries 32 bits. As `color_op` is performed component-wise (in 8-bit fields), each of the 8-bit fields in A bus is swapped with C bus separately. For instance, if A = 12345678h and C = ABh, the blue component will have the value:

ABh + ((B.blue - ABh) × 78h) / 256

and green would have:

ABh + ((B.green - ABh) × 56h) / 256

Note that for transparency:
Result.transp = ABh + ((B.transp – ABh) × 10h) / 256.

Description: Normal color operation command with some additional functions; see Special.

Special: If rtr field in frame_mode register is set to one, then Pixel Processor kills the pixels after comparing stipple_blend result to the internal transparency dither mask. If tsk field in ppu_mode register is set, pixel is killed depending on transparency result. Note that various parameter combinations are possible.

Parameters:

0	00000	If C bus = 4 (FIFO) then use the whole transparency value for blend operation.
1	00001	If C bus = 4 (FIFO) then use 4 LSBs of the transparency for blend operation.
2	00010	For transparency component, force blend factor to 0.
4	00100	For color (RGB components), force blend factor to 0.
8	01000	Swap A and C (control) bus values.
16	10000	Value on C bus is shifted left 4 bits (0000xxxx -> xxxx0000).

Inputs: A and B buses contain colors and C bus contains the blend factor.

Outputs: TMP1, TMP2, or TMP3. With dest value of 0 result is not written to the frame buffer, but possible pixel kills are proceeded.

Example:

end	ppu_oper	param	a_addr	b_addr	c_addr	dest
1	2	0	4	0	0	1

Read values from FIFO and COEF0, blends and stores the values after comparing them to the internal dither mask.

See also: rtr field (bit 0) in frame_mode (13) register, page 148.

The internal transparency dither mask is hardcoded into stipple_blend; it cannot be modified. It is completely different from the dither mask defined in the dither (10) register.

The decision for the Pixel Processor to kill a pixel is made according to the following:

```
if (raster_transparency ==1 and
      ((pixel.transparency shr 4) >
      ((dithy shr 1 and 1) +
       ((dithy shr 1 and 1) xor (dithx shr 1 and 1)) shl 1+
        (dithy and 1) shl 2+
       ((dithy and 1) xor (dithx and 1)) shl 3)
      )
   )
then kill_pixel;

dithx = ScreenPixelX and 3
dithy = ScreenPixelY and 3
```

Note that this dither mask comparison is not tied to the stencil bits in the ppu_mode (12) register in any way.

The transparency skip parameter in `ppu_mode` register (12) has effect only with the `stipple_blend` instruction. The effect is:

```
if ((transparency_skip==1) and
   ((pixel.transparency shr 4)==15))
then kill_pixel
```

This is to kill only the almost fully transparent pixels if stipple is not wanted.

logic_op **Shading Instructions** **opcode 3**

Description: `logic_op` command performs various logic operations between the A and B bus colors.

Special: Transparency value is taken from C bus, instead of A bus. If `osat` field in `frame_mode` register is one, the result values are clamped between 0-255, otherwise values are looping.

Parameters:

0	00000	A and B
1	00001	A and not B
2	00010	not A and B
3	00011	not A and not B
4	00100	A xor B
5	00101	reserved
6	00110	reserved
7	00111	reserved
9	01001	max(A,B) (finds higher color value of A and B)
12	01100	A + B
13	01101	A - B
16	10000	not (A and B)
17	10001	not (A and not B)
18	10010	not (not A and B)
19	10011	not (not A and not B)
20	10100	not (A xor B)
21	10101	reserved
22	10110	reserved
23	10111	reserved
25	11001	min(A,B) (finds lower color value of A and B)
28	11100	not(A + B)
29	11101	not(A - B)

Inputs: A and B buses; A operand for transparency logic operation is taken from C bus.

Outputs: Frame buffer, `TMP1`, `TMP2`, or `TMP3`

Example:

end	ppu_oper	param	a_addr	b_addr	c_addr	dest
0	8	0	8	0	4	2
1	3	4	4	6	6	0

Performs color XOR operation with A texture and FIFO TRGB. Sends result to the frame buffer.

See also: `frame_mode` (13) register `osat` field (bit 1), page 148.

Note that `max()`, `min()`, addition and subtraction are all component-wise operations. For example:

```
max(11223344h, 44332211h) = 44333344h
min(11223344h, 44332211h) = 11222211h
add(11223344h, 44332211h) = 55555555h
sub(11223344h, 44332211h) = 00001133h
```

For the last subtraction example, we have assumed that the overflow check `osat` (bit 1) in `frame_mode` (13) register is set to one, so that the negative results are clamped to zero.

cread **Shading Instructions** **opcode 4**

Description: Reads color value from frame buffer and stores it to the temporary register.

Special: If cm field in register 13 is set to 1, transparency value will also be read.

There is a Z-buffer mode where the Z value contains one bit of fast clear value. This bit is used to determine on a per-pixel basis whether the pixel is from the current frame or from the earlier frame. According to this information, `zread` first checks if the fast clear bit is different from the fast clear current value (`fcv`) bit in the `frame_mode` (13) register. If it is different, it means that the Z value is not from this frame and should be taken as zero ("fast cleared"). Same has to apply to `cread` as we do not want to get the color from the earlier frame, but get black instead. The `fcv` bit is changed every frame, and for fast clear to work properly, EVERY pixel on the screen has to be drawn every frame. Also, if `fce` in register 13 is set to 1 and the `fcv` comparison fails during `zread`, then return the black value.

Parameters: none

Inputs: Color value from the frame buffer.

Outputs: 32-bit word to any temporary register `TMP`1-3.

Example:

end	ppu_oper	param	a_addr	b_addr	c_addr	dest
1	4	0	0	0	0	1

Reads color value from the frame buffer and stores this value to `TMP`1.

See also: Register 13, bits 3 and 8, page 148.

Description: Kills pixels according to stencil mask, depth compare and fast clear.

Special:

```
If (ppu_mode_red.stencil == 1 and
   fetched_stencil != ppu_mode_reg.sok)
then kill_pixel

If (frame_mode.fce == 1 and
   fetched.fast_clear != frame_mode.fcv)
then kill_pixel
```

If Z equal compare (register 13, bit 2)==1 then Z equal compare will also kill pixels that have the same Z value as the one in the Z buffer.

Parameters: none

Inputs: Bits [23:0] of A bus for z-value, even for z buffer modes less than 24 bits.

Outputs: If the pixel is visible, value on A bus is written according to destination parameter. Valid destinations are `TMP`1-3.

Example:

end	ppu_oper	param	a_addr	b_addr	c_addr	dest
1	6	0	10	0	0	0

Compare FIFO Z (addr value 10) with z value from z-buffer, and skip the rest of the shading program if pixel is not visible.

For more information on fast clear, please refer to the `cread` command above. Stencil mask is very similar to fast clear, except that it can kill a pixel even if the pixel is visible after a Z compare operation. For example in a typical flight simulator game, if the pixels of the cockpit has stencil turned on, it is not necessary to redraw the cockpit in every frame, as it remains untouched from frame to frame.

Description: Stores the z, stencil and `fcv` values of a pixel, depending on the z-buffer mode (i.e. bits in `frame_mode` (13) register).

The fast clear bit is generated by reading from the `fcv` bit (bit 4 of register 13). And the stencil bit is generated from the old stencil value in the Z-buffer modified with the current stencil operation (bits 9-10 of register 12).
Typically the `a_addr` for `zwrite` is the Z-FIFO; in this case, the stencil values are written from internal registers. If the stencil operations `nop` or `invert` are used then the Pixel Processor code must also include `zread` in order to initialize these registers properly. If some other source is used, the stencil and fastclear bits come from that source directly. If the source is one of the `TMP` registers and the `zread` operation is used to initialize the `TMP` register, then the register will have correct stencil etc values deposited by the `zread` operation; although it is possible to change the values with other operations between `zread` and `zwrite.`

Special: -

Parameters: none

Inputs: A bus, the z value to be written.

Outputs: none

Example:

end	ppu_oper	param	a_addr	b_addr	c_addr	dest
1	7	0	10	0	0	0

Write FIFO Z to the z-buffer.

See also: `frame_mode` (13) register, bits 3, and 5-7; page 148.

Description: Makes a texture fecth.

Special: U and V are organized on the bus as follows:
U[11:0] = A bus [23:12]
V[11:4] = A bus [7:0]
V[3:0] = A bus [11:8]
Note that the MIP-map level is taken from the upper 4 bits of C bus. Refer to `bilin` instruction (opcode 10).

When `a/bm` bit (A/B texture MIP-map enable) in `a/btex_conf2` register (6 or 8) is one, the MIP level can be determined by reading it from the C bus of the Pixel Processor. Even when MIP enable is 0, it is still possible to use the `mip_add` bit in the instruction to force one level of MIP-map.

Parameters:

1	00001	Add one bit to V coordinate.
2	00010	Add one bit to U coordinate.
4	00100	Add one to the MIP-map level.
8	01000	Select A/B texture settings. A=0, B=1.

Inputs: Bits [23:0] of A bus. Bits [7:4] of C bus

Outputs: 32-bit TRGB color that can be stored to any temporary register `TMP`1-3.

Example:

end	ppu_oper	param	a_addr	b_addr	c_addr	dest
0	8	0	8	0	4	1
1	1	0	5	4	0	0

Blends A texture with FIFO TRGB, with the amount in C bus.
Sends the result to the frame buffer.

See also: `atex_conf1` (5) and `atex_conf2` (6) registers, pages 142 and 143, and `btex_conf1` (7) and `btex_conf2` (8) registers, page 144.

Description: Performs a texture fetch based on the A bus values; uses the fetched 16-bit value as follows:

Bits 4:0 contain the signed horizontal modulation vector.
Bits 9:5 contain the signed vertical modulation vector.
Bits 15:10 contain the color index that can be used later to address the palette.

Modulates texture components on the B bus.

As a result operation writes modulated component and color index on result bus:
Res 31:26 color index (5:0)
Res 25:24 color index (5:3)
Res 23:12 modulated U component(11:0)
Res 11:4 modulated V component(7:0)
Res 3:0 modulated V cmponent(11:8)

Special: `modulation` (11) register contains the horizontal and vertical modulation coefficients.

Parameters:

1	00001	Add one bit to V coordinate.
2	00010	Add one bit to U coordinate.
4	00100	Add one to the MIP-map level.
8	01000	Select A/B texture settings. A=0, B=1.

Inputs: A bus, containing the U and V coordinates for the source texture.
B bus, containing the U and V coordinates to be modulated.

Outputs: `TMP1`, `TMP2`, or `TMP3`.

Example:

end	ppu_oper	param	a_addr	b_addr	c_addr	dest
0	9	0	8	9	0	1
0	8	0	5	0	0	2
1	4	0	6	4	0	0

Modulates BTU and BTV coordinates with the vector fetched with ATU/TV components texture, and the modulation vector from `modulation` (11) register. Write the result to the `TMP1`. Makes texture fetch using modulated component from `TMP1` and stores TRGB value to `TMP2`. Combines color values from `TMP2` and FIFO to the frame buffer.

See also: `modulation` (11) register, page 147.

Description: Performs bilinear interpoation inside 2×2 texture pixel matrix that is relative to U and V coordinates on A bus. `Bilin` works the same way as `textfetch` except that it performs bilinear filtering for the texel.

Special: Four LSB bits of U and V are used to perform blending. Note that the MIP-map level is taken from the upper 4 bits of C bus. Also, the bits used for blending are the next four which are not used in texture address calculation; which bits these are depends on SUBS and MIP parameters.
When `am` bit (A/B texture MIP-map enable) in `a/btex_conf2` register (6 or 8) is one, the MIP level can be determined by reading it from the C bus of the Pixel Processor. Even when MIP enable is 0, it is still possible to use the `mip_add` bit in the instruction to force one level of MIP-map.

Parameters:

1	0001	add one bit to V-coordinate (not used)
2	0010	add one bit to U-coordinate (not used)
4	0100	add one to MIP level
8	1000	select a/b texture settings

Inputs: Bits [23:0] of A bus.

Outputs: Interpolated 32-bit TRGB color, ready to be stored to temporary register `TMP` 1-3.

Example:

end	ppu_oper	param	a_addr	b_addr	c_addr	dest
0	10	0	8	0	0	1
1	4	0	1	0	0	0

Fetch bilinear filtered texels, and store to `TMP1`. Writes `bilin` result from `TMP1` to the frame buffer.

When using MIP-mapping, the MIP-map is selected with the upper 4 bits of the value on C bus.

7	6	5	4	3	2	1	0
m	m	m	m	n	n	n	n

For example a C-bus value of 8 would select MIP-map 0 (the main texture) because the m-bits are zero, and a C-bus value 22 (=16h) would select MIP-map 1, and so on. The reason for using the upper and not lower bits to select the level is that the lower bits are needed, for example, for trilinear blending between two MIP-map levels.

We consider here the popular method of trilinear filtering as an example. Note that trilinear filtering concerns the bilinear texture fetch instruction only, and not the point-sampled texture fetch instruction. When performing trilinear filtering, two texture fetches are performed. The texture fetch instruction has a parameter that adds 1 to the MIP-map level. For example, a C-bus value of 22 (16h) becomes 38 (16h + 10h =26h) and this selects the next MIP-map. In practice, the Pixel Processor code would look like the following:

; fetch first texture

end	ppu_oper	param	a_addr	b_addr	c_addr	dest
0	10	0	8	0	4	1

; fetch next MIP-map level

end	ppu_oper	param	a_addr	b_addr	c_addr	dest
0	10	4	8	0	4	2

; blend the two textures using the lower 4 bits of MIP-map interpolator, and output to frame ; buffer

end	ppu_oper	param	a_addr	b_addr	c_addr	dest
1	1	1	5	6	4	0

Note that the combination of different shading instructions such as the examples above can implement very versatile and advanced 3D shading and filtering algorithms.

tlogic **Shading Instructions** **opcode 11**

Description: Same as `logic`, opcode 3, with the following exeptions:

Special: The normal logic operation is proceed and if the TRGB result is zero then pixel is killed.

Outputs: `TMP1`, `TMP2`, or `TMP3`. With `dest` value of 0 result is not written to the frame buffer, but possible pixel kills are proceeded.

palette **Shading Instructions** **opcode 12**

Description: Reads the color from the internal palette RAM using the C bus as index.

Special: Palette mask and palette base values are specified in palette_base (15) register. This instruction uses the same A/B palette parameters, and is not limited to the B texture parameters. Result = palette[(C bus and b_palmask) or b_palbase)]

Parameters: 8 01000 Select A/B texture palettebase register A=0 B=1

Inputs: 8-bit index from C bus.

Outputs: `TMP1`-3

Example:

end	ppu_oper	param	a_addr	b_addr	c_addr	dest
0	12	0	0	0	4	1
1	8	0	5	0	0	0

Read the color from the palette using FIFO transparency as index, and store the result to `TMP1`.
Blends this palette with COEF0 register and sends the results to the frame buffer.

See also: palette_base (15) register, page 149.

6.5 Pixel Processor Registers

Register address	Offset	Register name
1	0004h	coef_reg0
2	0008h	coef_reg1
3	000Ch	coef_reg2
4	0010h	coef_reg3
5	0014h	atex_conf1
6	0018h	atex_conf2
7	001Ch	btex_conf1
8	0020h	btex_conf2
9	0024h	base_addr
10	0028h	dither
11	002Ch	modulation
12	0030h	ppu_mode
13	0034h	frame_mode
14	0038h	ppu_code_start
15	003Ch	palette_base

coef_reg0	register 1	offset 0004h

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
transp								**red**							
green								**blue**							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	Bits	Description
blue	7:0	Coefficient 0 blue
green	15:8	Coefficient 0 green
red	23:16	Coefficient 0 red
transp	31:24	Coefficient 0 transparency

coef_reg1	register 2	offset 0008h

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
transp								**red**							
green								**blue**							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	Bits	Description
blue	7:0	Coefficient 1 blue
green	15:8	Coefficient 1 green
red	23:16	Coefficient 1 red
transp	31:24	Coefficient 1 transparency

coef_reg2	register 3	offset 000Ch

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
transp								red							
green								blue							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	Bits	Description
blue	7:0	Coefficient 2 blue
green	15:8	Coefficient 2 green
red	23:16	Coefficient 2 red
transp	31:24	Coefficient 2 transparency

coef_reg3	register 4	offset 0010h

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
transp								red							
green								blue							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	Bits	Description
blue	7:0	Coefficient 3 blue
green	15:8	Coefficient 3 green
red	23:16	Coefficient 3 red
transp	31:24	Coefficient 3 transparency

atex_conf1 register contains the base address for A texture, measured in units of 2048 bytes and texture height in memory in 32 pixel blocks.

atex_conf1	register 5	offset 0014h

Format

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

Fields

Field	Bits	Description
abaseb	13:0	A texture base address in 2048 byte blocks
amhig	21:16	A texture height in 32-pixel blocks

atex_conf2	register 6	offset 0018h

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
asubs					ayl	axl	am				ad	amode			
					aphig								apwid		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	Bits	Description
asubs	31:29	A texture sub pixels
ayl	26	ayloop
axl	25	axloop
am	24	A texture MIP-map enable
ad	20	A texture data same on both memory banks
amode	19:16	A texture mode
aphig	10:8	A texture height in pixels
apwid	2:0	A texture width in pixels

asubs
A texture subpixel accuracy. Determines how many bits are reserved for subpixels; the rest are for the actual pixels. For example 5bits for subpixels and 7bits for actual pixels gives a total of 12bits.

ayl, axl
Ayl controls AV component looping/clamping
Axl controls AU component looping/clamping
If texture looping is enabled ax / yl = 1 texture coordinate larger than the size of the texture will be wrapped around.
If a coordinate is clamped ax / yl = 0 texture coordinate larger than the size of the texture will be forced to zero, and the texture color fetched from this location will be used as a result.

am
A texture MIP-map enable. When `am` is one, the MIP level can be determined by reading it from the C bus of the Pixel Processor. Even when MIP enable is 0, it is still possible to use the mip_add bit in the instruction to force one level of MIP-map.

ad
A-texture double. Pixels are interleaved in memory banks; this bit enables both external memory banks storing the same texels.

This dual bank mode is used to get faster access to texture through the two separate 32-bit buses which take advantage of reducing the access latency at the 64-bit SDRAM interface. If the `ad` bit is one and texture is uploaded so that the even and odd texels are stored to both buses respectively, VS25203 can use faster accesses to fetch the texture without the need for possible texel swapping inside the Pixel Processor. This feature, together with the other more advanced features in VS25203, are quite complex to support in current industry standard 3D APIs. They are used mainly in arcade and specialized applications. A practical way to take advantage of this feature is to have texture fetches twice as wide, where even and odd pixels are cloned horizontally.

amode

A texture mode:

```
0000  8 bit index
0001  4 bit index
0100  RRRRRGGGGGGBBBBB
0101  TRRRRRGGGGGBBBBB
0110  TTTTRRRRGGGGBBBB
1000  TTTTTTTTRRRRRRRRGGGGGGGGBBBBBBBB
1001  AAAAAAAAVVVVVVVVYYYYYYYYUUUUUUUU
1010  YYYYYYYYVVVVVVVVYYYYYYYYUUUUUUUU
```

T signifies transparency, R red, G green and B blue, respectively.

apwid, aphig

A texture pixel width; A texture pixel height. The following list contains calculated values for different texture map sizes:

```
000   16
001   32
010   64
011   128
100   256
101   512
110   1024
111   2048
```

Seven is the maximum value for this field, which gives the maximum texture map size of 2048 times 2048 pixels.

btex_conf1	register 7	offset 001Ch

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
										bmhig					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		bbaseb													

Fields

Field	Bits	Description
bbaseb	13:0	B texture base address in 2048 byte blocks
bmhig	21:16	B texture height in 32-pixel blocks

See atex_conf1.

btex_conf2	register 8	offset 0020h

Format

<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td></tr>
<tr><td colspan="3">bsubs</td><td colspan="2"></td><td>byl</td><td>bxl</td><td>bm</td><td colspan="3"></td><td>bd</td><td colspan="4">bmode</td></tr>
<tr><td colspan="5"></td><td colspan="3">bphig</td><td colspan="5"></td><td colspan="3">bpwid</td></tr>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
</table>

Fields

Field	Bits	Description
bsubs	31:29	B texture sub pixels
byl	26	byloop
bxl	25	bxloop
bm	24	B texture MIP-map enable
bd	20	B texture data same on both memory banks
bmode	19:16	B texture mode
bphig	10:8	B texture height in pixels
bpwid	2:0	B texture width in pixels

See atex_conf2.

base_addr	register 9	offset 0024h

Format

<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td></tr>
<tr><td colspan="2"></td><td colspan="14">zbaseb</td></tr>
<tr><td colspan="2"></td><td colspan="14">cbaseb</td></tr>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
</table>

Fields

Field	Bits	Description
cbaseb	13:0	Frame buffer base address in 2048-byte blocks
zbaseb	29:16	Z-buffer base address in 2048-byte blocks

base_addr register contains the base address for Z-buffer and graphics memory.

dither	register 10	offset 0028h

Format

31–24	23–21	20–18	17–16
	d7	d6	d5

15	14–12	11–9	8–6	5–3	2–0
d5	d4	d3	d2	d1	d0

Fields

Field	Bits	Description
d0	2:0	Dither 0
d1	5:3	Dither 1
d2	8:6	Dither 2
d3	11:9	Dither 3
d4	14:12	Dither 4
d5	17:15	Dither 5
d6	20:18	Dither 6
d7	23:21	Dither 7

VS25203 uses a 4×4 ordered dither matrix. This register describes the dither mask; only bits 0:23 are significant. Dither values of 0:7 are located in the dither mask as follows:

DITHER 0	DITHER 2	DITHER 4	DITHER 6
DITHER 1	DITHER 3	DITHER 5	DITHER 7
DITHER 4	DITHER 5	DITHER 0	DITHER 2
DITHER 6	DITHER 7	DITHER 1	DITHER 3

Dithering is enabled by default, where bit 4 (`nd`: no-dither bit) of `ppu_mode` (12) register is reset to zero.

Dithering is controlled by software functions in the display driver. If zero mask values are passed to the functions, dithering has no effect. The suggested mask value in VS25203 is `007E95A0h`; this is a popular value in dithering literature, but most large random values will have similar results.

The theory behind dithering is that noise is added to the pixel bits below a certain fixed binary point and then the lower order bits are discarded. For example, in RGB 5:6:5 frame buffer format, 3-bit values from the dither matrix are added to the 3:2:3 lower order bits of the pixel value according to the pixel's x and y coordinates. The pixel value then has its 3:2:3 lower order bits truncated before it is written back to the frame buffer. Note that the green component has less bits since human vision requires more green resolution in the pixel value. This is the reason for the use of `shr` bit (bit 5, shift-green-dither-value-right-by-one field) of the ppu_mode (12) register to control a one-bit right shift of the dither value before the value is added to the green component.

modulation	register 11	offset 002Ch

Format

31–24	23–16
modvy	modvx

15–8	7–0
modhy	modhx

Fields

Field	Bits	Description
modhx	7:0	Horizontal X modulation coefficient
modhy	15:8	Horizontal Y modulation coefficient
modvx	23:16	Vertical X modulation coefficient
modvy	31:24	Vertical Y modulation coefficient

This register describes coefficients used to rotate modulation vector which is stored in texture map for the purpose of bump mapping. This register is only used with `textfetch_modulate` command of the Pixel Processor, see page 138.

ppu_mode	register 12	offset 0030h

Format

31–16

15–11	10–9	8	7	6	5	4	3–0
	st_oper	sok	s	tsk	shr	nd	

Fields

Field	Bits	Description
nd	4	No dither
shr	5	Shift green dither value right by one
tsk	6	Transparency skip
s	7	Stencil
sok	8	Stencil reference value
st_oper	10:9	Stencil operation

nd
No dither; disables the internal dithering logic in VS25203.
shr
Shift green dither value right by one. To make dithering work correctly in 16-bit frame buffer mode where the green component has 6 bits, while the red and blue components have only 5 bits.
sok
Stencil reference value used in `zread` instruction

tsk
Transparency skip. It only has effect with the `stipple_blend` shading instruction (opcode 2). The effect is:

```
      if ((transparency_skip==1) and
         ((pixel.transparency shr 4)==15))
      then kill_pixel
```

This is to kill only the almost fully transparent pixels if stipple is not wanted.
s
Stencil. Enable(1)/disable(0) pixel kill by stencil in `zread`.
st_oper
Stencil operation:
00 no operation
01 set stencil mask
10 clear stencil mask
11 invert stencil mask

frame_mode	register 13	offset 0034h

Format

31	30	29	28	27	26	25	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
							cm	zm			fcv	fce	zeq	osat	rtr

Fields

Field	Bits	Description
rtr	0	Raster transparency
osat	1	Overflow saturate
zeq	2	Z equal compare
fce	3	Fast clear enable
fcv	4	Fast clear current value
zm	7:5	Z memory mode
cm	8	C memory mode

rtr
Raster transparency; it has to be set to one for the `stipple_blend` command to do stippling. Refer to the `stipple_blend` command for more details.
osat
Overflow saturate; when it is set it causes the possibly overflowing operations (dithering, logic unit add/subtract) to saturate their results.
zeq
Z equal compare; if this bit is one, Zcompare will ALSO kill pixels that have exactly the same Z value as the one in the Z buffer.
fcv
Fast clear current value; refers to the description of `cread` command (opcode 4).

zm
Defines the z memory mode:
`000   ZZZZZZZZZZZZZZZZ`$_o$
`001   ZZZZZZZZZZZZZZZF`$_o$
`010   ZZZZZZZZZZZZZZFS`$_o$
`011   000000FSZZZZZZZZZZZZZZZZZZZZZZZZ`$_o$
Z signifies z-value, F is fast clear and S is stencil.
cm
Defines the color memory mode:
`0     RRRRRGGGGGGBBBBB`$_o$
`1     TTTTTTTTRRRRRRRRGGGGGGGGBBBBBBBB`$_o$
T signifies transparency, R red, G green and B blue, respectively.

ppu_code_start	**register 14**	**offset 0038h**

Format

31–16	

15–5	4–0
	start_addr

Fields

Field	Bits	Description
start_addr	4:0	Pixel processor code start address

The start address of the shading program for the Pixel Processor unit (ppu) is stored in the `ppu_code_start` register.

palette_base	**register 15**	**offset 003Ch**

Format

31–24	23–16
b_palmask	**b_palbase**

15–8	7–0
a_palmask	**a_palbase**

Fields

Field	Bits	Description
a_palbase	7:0	A texture palette index base
a_palmask	15:8	A texture palette index mask
b_palbase	23:16	B texture palette index base
b_palmask	31:24	B texture palette index mask

palette_base register contains information used in indexing the internal palette memory of VS25203. It uses the following formula:

Result = palette[(C_bus and palmask) or palbase]

See also `palette` command on page 140.

6.6 Pixel Processor Unit Memory Blocks

The Pixel Processor contains a program memory containing 32 24-bit words. This memory is mapped to the address range 128-159.
The Pixel Processor also contains memory for storing the color palette used in some of the texture map modes. This memory is mapped to the address range 256-511.

6.7 VS_VP Bump Mapping Programming Guidelines

The bump mapping modulation matrix is a 2D rotation matrix. It rotates the bump map vectors so that the effect of rotating a triangle (while the environment map stays in the same orientation) can be counteracted. The values in the two vectors inside the matrix are signed 8-bit integers; and the two vectors should be in 90 degrees angle (i.e. mutually orthogonal, with dot product value of zero):

The bumpiness effect of a bump map can be scaled by using smaller modulation values in the matrix. Also, the bump map data loaded into VS25203 should be in X,Y signed delta format (bits 0..4 for X, bits 5..9 for Y). Bump mapping is inherently slow due to random accesses to texture. But the second (conventional diffuse) texture can be stored into the bump map as a 6-bit paletted texture. The palette shading instruction (opcode 12) can then be used to fetch the color that matches the palette.

For a 3D artist to generate a bump map which looks visually realistic, it is best to use a grey scale map with white presenting high. Also, it works best if the map does not contain regular patterns (like text, or logo) as the possible artifacts are more visible on maps like these. One parameter to try while generating a bump map in Photoshop™ is the smoothness (blur) parameter; by trying different versions of the map an artist can find a map that looks best for the application.

The VS_VP bump mapping method always requires a light map to get the effect of a bump surface. Bump + environment map and bump + diffuse map are both supported. The difference between these two is in the way of calculating light map coordinates. In bump + environment map, the light map (environment map in this case) coordinates are generated on the fly from the vertex normal vectors. That is why the environment seems to reflect from the surface, and as we distort the environment map coordinates per pixel, we get the effect of a bumpy surface. In bump + diffuse map, the light map coordinates are static (e.g. in Id Software's Quake™); i.e. the light map does not move or slide on the surface.

It is possible to use the same VS_VP bump mapping method to make diffuse light maps look bumpy, but the benefit of having two separate maps is not significant anymore as these two maps could have been pre-rendered into a single normal texture map for actual use.

Depending on the object, the VS_VP bump mapping modulation matrix works best if it is the same matrix for the whole object. If a modulation matrix is chosen for each individual triangle, we would get discontinuity at the triangle edges. The reason is that we cannot interpolate the matrix over the triangle. The modulation matrix should contain the major horizontal and vertical mapping angles of the bump map on the object. A simple example is an object that has a bump map applied with plane mapping. In this case, the orientation of the 2D plane that is used to access texture coordinates for the bump map should be put into the modulation matrix, with a 90 degree angle between the horizontal and vertical vectors.

Take the example of a car racing game with a car in a blue sky environment, if we intend to have bumps on the racing car surface, the best way to do so is to create a bump texture with the original texture combined to the same map (e.g. car number, stickers etc as a 6-bit palette index). We then use this map to modulate the surrounding sky environment map coordinates:

Atexture = Bump map
Btexture = Sky environment map

The following steps are then carried out through shading instructions for the Pixel Processor:

`1. textfetch_modulate a:Atexture, b:Btexture -> TMP1`
Through this step, we will have bump mapped sky environment map color in TMP1. We also have the 6-bit index (from the bump map) in TMP1_alpha arranged in the bit order of 54321054. This bit layout makes it easier to use TMP1_alpha, for example, to carry out palette look-up or to use it as a blending factor.

`2. textfetch a: TMP1 -> TMP1`
This instruction fetches color information from texture memory using modulated texture component from TMP1 writing the result to TMP1 for further use.

`3. palette c:TMP1_alpha (address 5) -> TMP2`
At this point we have the diffuse color of the surface in TMP2 and specular color in TMP1. Next thing to do is to add these colors together (with saturation turned ON; register 13 bit 1).

`4. logic_op 12 (a+b) a:TMP1, b:TMP2 -> screen`
With the above three-instruction pixel code, the intended combined bump/environment mapping effect is realized.

7. Clock Synthesis and Control

7.1 Overview

VS25203 contains two phase-locked-loop (PLL) frequency synthesizers. They generate clock signals for the processor and for video. VS25203 uses an external crystal, which is connected between the Osc_out and Osc_in pins. The frequency of the crystal is 14.3181818 MHz. Both synthesizers can be programmed separately for up to 200MHz.

7.2 Programming

The frequency synthesized by each PLL is determined by the following equation:

$$F_{OUT} = \frac{m\_coef + 2}{(n\_coef + 2) \times 2^{r\_coef}} \times F_{OSC}$$

where:

n_coef, m_coef, r_coef = clock coefficients
F_{OSC} = quartz crystal or external clock (MHz).

The quartz crystal frequency F_{OSC} is 14.3181818 MHz. The integer values of `n_coef`, `m_coef` and `r_coef` should be between the following values:

`n_coef`:	0-127 (0-32 recommended)
`m_coef`:	0-127
`r_coef`:	0-3

For the best clock stability, there are some guidelines for programming the on-chip frequency synthesizer. The most stable operation for the PLL is achieved when the phase detector frequency in the synthesizer is as high as possible. This condition requires that the `n_coef` counter value is as small as possible. The next guideline is to have high VCO (voltage controlled oscillator) frequency, preferably in the 150-300 MHz range. This condition requires that the value `r_coef` is as large as possible.

Coefficients are defined in `core_clk_cfg` (16) register and `video_clk_cfg` (18) register on pages 28 and 33. The following table contains examples for some F_{out} values. Note that the actual frequency may vary slightly.

Desired Frequency	m_coef	n_coef	r_coef	F_{out} (MHz)
25	125	7	3	25,2557
35	86	7	2	35,0000
45	111	7	2	44,9432
50	125	7	2	50,5114
65	107	4	2	65,0284
75	40	0	2	75,1705
85	117	8	1	85,1932
95	118	7	1	95,4545
105	115	6	1	104,7017
115	110	5	1	114,5455
125	120	5	1	124,7727
135	111	4	1	134,8295
145	119	4	1	144,3750
155	85	2	1	155,7102
165	113	3	1	164,6591
175	120	3	1	174,6818
185	101	2	1	184,3466
200	110	2	1	200,4545

Caution: Unsuitable clock frequency parameters may cause permanent damage to the device.

8. VGA Core

8.1 Introduction

VS252 VGA Core is 100% IBM® VGA compatible, and has extensions for supporting SVGA modes with higher refresh rates and larger screen dimensions. The VGA Core is highly integrated circuit, taking full advantage of PCI and refresh logic providing maximum bandwidth from bus to memory and from memory to DAC. Internally, the VGA core is divided, as traditionally, into two components: host and video interfaces, see Fiqure 8.1-1 below. Frame buffer, the on-board video memory, is shared with other units. The host interface communicates with PCI Controller and takes care of memory writes, reads and I/O mapped register access. The video interface is a complex state machine, which carries the data flow from the memory to the DAC. The video interface operation can be subdivided into alphanumeric and graphics modes.

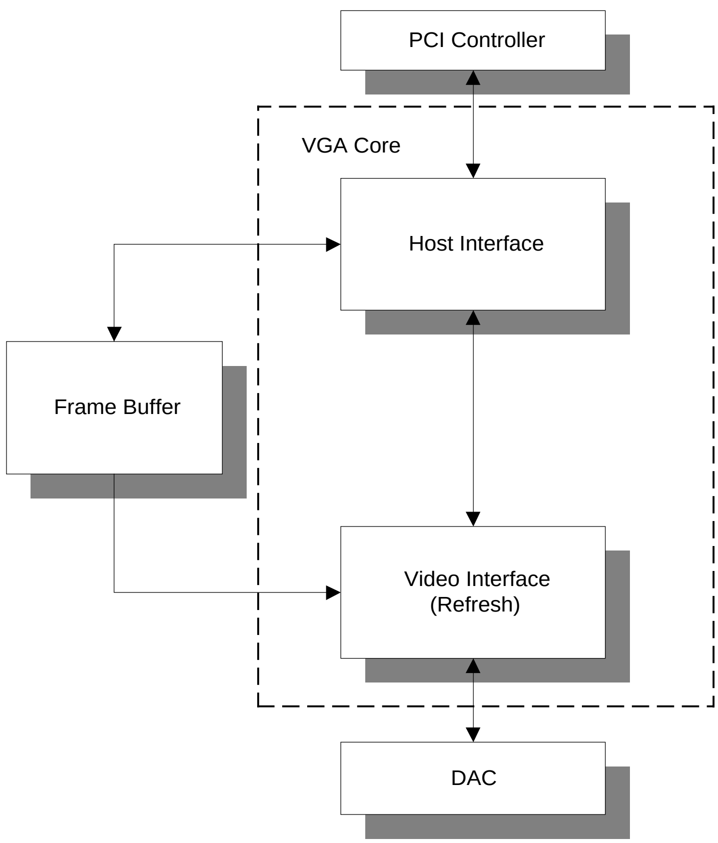


Figure 8.1-1. VGA Core external interfaces.

As mentioned, VS252 VGA Core does not include just a VGA. It optimizes indexed 256-color modes in a way that allows the full potential of PCI bus to be harnessed. Acceleration is provided for standard VGA mode 13h and VESA linear 256 color modes.

8.2 VGA Memory and Register Mapping

8.2.1 Introduction

During system startup IBM compatible PC‘s memory is organized like in the Table 8.2.3-1. VGA memory window is located at memory range A0000h-BFFFFh, BIOS ROM is located in memory range C0000h-C7FFFh and registers are mapped in the I/O space registers from 3B4h to 3Dah and into PCI aperture’s shadow area. Decoding of the I/O and the memory addresses can be disabled from the PCI configuration space.

8.2.2 VGA Register mapping

VGA registers are mapped in I/O space locations from 3B4h to 3DAh. Table 8.2.2-1 describes the VGA register map. Location of CRTC00-CRTC45, FEATCTRL and INPUTS1 registers are dependent of the MISCOUT register’s bit 0. When MISCOUT bit 0 is ‘1’, these registers are mapped into ports 3Dxh, otherwise into ports 3Bxh. Mapping of the other registers is fixed. The I/O space decoding can be disabled by setting PCI configuration space register 4 bit 0 to zero.

Accessing of VGA registers is not as straightforward as accessing the memory mapped registers of the other units. CRTC, Graphics and Sequencer Registers are indexed registers, which are accessed by writing register index into index port (for example CRTCINDEX) and then writing the data into the data port (CRTCDATA). Writing an indexed register can be done with two 8-bit operations or by a singe 16-bit write, where the index is located in the low-byte and the data in the high byte. Reading an indexed register must be done by first writing the index and then reading the data. CRTC registers from 0 to 7 can be locked from writing by CRTC11 bit 7 (reading of them is still possible). CRTC registers from 40 to 45 (extended registers) can be written only if PCI apertures register 21 (Feature Register) bit 3 is ‘1’. All the other indexed registers can be read and written all the time.

General Registers, MISCOUT, FEATCTRL, INPUTS0 and INPUTS1 are accessed directly by reading and writing their corresponding I/O port. Input Status registers are read only registers, while MISCOUT and FEATCTRL are read/write.

Attribute registers are accessed by first reading INPUTS1 register and then writing the attribute register index into port 3C0h and then writing the data into 3C0h. If the bit 5 of the index is not set, the refresh logic does not have access into attribute registers, and the screen will be black. If the bit 5 of the index is set, the refresh can access the palette and attribute registers all the time. Attribute register data can be read from the port 3C1h and the index can be read from port 3C0h.

Color palette can be accessed by first writing the index into 3C8h and then writing red, green and blue values sequentially into port 3C9h. The index of the color palette is incremented after writing the blue component. Thus the whole 256-color palette can be written by writing first the index and then writing all 768 color component values into port 3C9h. The read operation is similar. The CPWADDR can be read, but the CPRADDR can not. CPSTATE is read only port, and CPMASK can be directly read and written. This is the basic operation of writing the 256-indexed color palette. It is not possible to write only red and green components of the color palette register, but the whole RGB value must be written before the actual palette register is updated.

VGA memory and I/O space can be accessed through PCI aperture. Since PCI apertures are relocatable, VGA can be used without fixed I/O and memory ranges. Memory mapped VGA registers, called shadow registers, are located in PCI apertures register space at address 50h. This means, that each VGA register can be accessed like using the normal VGA, but instead of making an I/O write/read to a port 3XXh, one makes a memory write/read to address 50 + (3XXh-3C0h). For example, if one wants to write CRTC register index, one makes a byte write to address 64h (50h + 3D4h-3C0h) in the memory mapped register space. The value of Miscellaneous Output Register does not affect the memory aperture register mapping. Otherwise, the behavior of the shadow registers is similar to their behavior in the I/O space.

I/O ADDRESS	OFFSET	WRITE REGISTER	READ REGISTER
3B4	64h	CRTCINDEX	CRTCINDEX
3B5	65h	CRTCDATA	CRTCDATA
3B6			
3B7			
3B8			
3B9			
3BA	6Ah	FEATCTRL	INPUTS1
3BB			
3BC			
3BF			
3C0	50h	ATTRIDX	ATTRIDX
3C1	51h		ATTRDATA
3C2	52h	MISCOUT	INPUTS0
3C3			
3C4	54h	SEQINDEX	SEQINDEX
3C5	55h	SEQDATA	SEQDATA
3C6	56h	CPMASK	CPMASK
3C7	57h	CPRADDR	CPSTATE
3C8	58h	CPWADDR	CPWADDR
3C9	59h	CPDATA	CPDATA
3CA	5Ah		FEATCTRL
3CB			
3CC	5Ch		MISCOUT
3CD			
3CE	5Eh	GFXINDEX	GFXINDEX
3CF	5Fh	GFXDATA	GFXDATA
3D0			
3D1			
3D2			
3D3			
3D4	64h	CRTCINDEX	CRTCINDEX
3D5	65h	CRTCDATA	CRTCDATA
3D6			
3D7			
3D8			
3D9			
3DA	6Ah	FEATCTRL	INPUTS1

Table 8.2.2-1. VGA Register Mapping.

8.2.3 VGA Memory Mapping

VGA memory is mapped in 80x86 real mode as described in the Table 8.2.3-1. The alternative method for accessing the VGA memory is through the PCI memory apertures, which provide direct access to the frame buffer. The physical address (bus address) of the linear frame buffer can be obtained from the PCI configuration space register 4.

VGA memory space decoding is enabled at boot time, and can be disabled by setting PCI configuration space register 21 bit 0 to zero. Possible maps are from A0000h to BFFFFh, from A0000h to AFFFFh, from B0000h to B7FFFh and from B8000h to BFFFFh. The map used is selected from GFX6 bits 2 to 3. When the whole 128kb memory window is used, the addresses over B0000h are aliased to memory space starting at A0000h, unless the bit 6 of CRTC41 is set. Writes outside the memory map are discarded by the PCI interface. Writes through the VGA memory window can be disabled by MISCOUT register's bit 1.

When data is written through the VGA memory space, it is handled by the VGA host interface. There are several host interface configurations, which determine the format in which the data is actually written to the frame buffer. There are four different write modes and two read modes combined with several control registers, for example plane and bit masks that make the VGA host interface rather complex. The most important configuration registers are Graphic Controller Registers and Sequencer Register 2 and 4.

In linear modes, VGA host interface is bypassed and data is written to memory without modifications. This can be done either by enabling the VGA linear mode in the VGA memory window, or by writing the data directly to the frame buffer through PCI apertures. In the linear mode, each byte represents index to the 256-color palette. Linear host interface is enabled by CRTC41 bit 3. Refresh logic can be also configured in linear mode by setting refresh to standard VGA mode 12h byte addressing mode and enabling 64-bit sequencer model by setting CRTC41 bit 4 to '1'.

Memory	Explanation
FE0000-	Shadow ROM BIOS
10000-FDFFFF	Extended memory
F0000	Planar BIOS
E0000	Expansion BIOS and motherboard video BIOS
D8000	Voice Communication BIOS/LIM EMS page map area
D0000	Network BIOS
CC000	LIM EMS page map area
C8000	Hard disk BIOS
C0000	**VGA BIOS**
A0000	**VGA Memory Window**
00600	System RAM
00400	BIOS Data Area
00000	Interrupt Vector Tables

Table 8.2.3-1. IBM PC Memory Layout during system startup.

8.3 VGA Subsystem Configuration

VGA Subsystem can be enabled and disabled using PCI configuration space register 21, which provides four bits to control the state of the VGA Core. See the Table 8.3-1 below. The functionality and of these bits is described in table. The boot time configuration corresponds the IBM VGA setup where extension registers are hidden and VGA decoding is enabled.

Feature register bit 0 activates the VGA I/O and memory range decoding. At boot time the VGA decoding is enabled by default. Feature register bit 1 selects between VGA and 3D refresh. VGA refresh is used for 256 color or less output and 3D refresh is used for true- and high-color output. Unless the bit 2 of the Feature register is set, the selection between VGA and 3D refresh changes if these registers are written. Feature register bit 3 enables writes to extended VGA registers. The extended registers (CRTC40-CRTC45) can be read all the time.

bit	Explanation	Reset value
3	VGA extension enable 0 enables using of extended VGA registers 1 extension registers are also available	0
2	VGA refresh select lock 0 selection of refresh registers active 1 selection locked	0
1	VGA refresh select 0 3D video refresh registers used 1 VGA refresh registers used Selects 3D/VGA video refresh control. This bit changes its state automatically if VGA or 3D refresh registers are accesses, unless the select lock (bit 2) is active	1
0	VGA decode enable Activates the decoding of the standard VGA memory and IO ranges.	1

Table 8.3-1. PCI Configuration space register 21, `Feature` register.

8.4 VGA Clock Configuration

8.4.1 Introduction

VGA host and video clock is the same as system video and system core clock. The value for the system core and video clocks can be calculated as:

Clock Configuration Register	Coefficient
Bits 14-15	*r_coef*
Bits 7-13	*m_coef*
Bits 0-6	*n_coef*

$$F_{OUT} = \frac{m\_coef + 2}{(n\_coef + 2) \times 2^{r\_coef}} \times F_{OSC}$$

F_{OSC} is the frequency of external clock, usually 14.318MHz.

WARNING! Unsuitable clock parameters may cause permanent damage to the device.

8.4.2 Host Interface

VGA host interface clock is same as system Core Clock. This is defined in PCI configuration space register 16, Core Clock Config. See general clock programming guidelines for programming this register.

8.4.3 Video Interface

Video clock is derived from the system video clock, which is set from PCI configuration space register 18. If VGA refresh is enabled, the system video clock can be set indirectly by programming MISCOUT register. Writing this register with bit 3 as ‘0’, the system video clock will be programmed to a new value. If bit 3 is set, the write into MISCOUT does not affect video clock setting.

8.5 VGA Interrupt Generation

VGA generates vertical retrace interrupts, if System Control Register bit 12 is set. The vertical retrace interrupt must be cleared using VGA register CRTC11. See Table 8.5 below.

`ref_reg`	Explanation
VGA IRQ ena (bit:12)	if this bit is set then the vga unit generated interrupt is routed to the pci bus. An interrupt which is initiated by the vga block must be reset using by the vga unit video_irq (bit:11)
Video IRQ (bit:11)	if this bit is set to one the circuit will generate an interrupt request when the video_y value reaches the video_y_ref value
video_y_ref (bits:10-0)	

Table 8.5 System Control Register 49, `ref_reg`.

8.6 VGA Registers

8.6.1 General Registers

MISCOUT - Miscellaneous Output Register	offset 005Ch / 0052h	Standard VGA

Access

Read Address	3CCh
Write Address	3C2h
Index	-
Access Type	R/W

Format

7	6	5	4	3	2	1	0
VSP	HSP	PS	R	C		EVDM	P I/O B

Fields

Field	Bits	Description
VSP	7	Vertical Sync Polarity
HSP	6	Horizontal Sync Polarity
PS	5	Page Select
R	4	Reserved
C	3:2	ClockSelect
EVDM	1	Enable VGA Display Memory
P I/O B	0	Port I/O Base

Register overall description:
This registers is an important control register, which controls sync polarities, I/O addressing, pixel clock settings and memory access. Write to this register programs the Video Refresh clock, if the bit 3 of the value written is zero.

Field description:

Vertical Sync Polarity
If set to zero (0), the vertical sync is a signal going from low to high (0 -> 1).
If set to one (1), the vertical sync is a signal going from high to low (1 -> 0).

Horizontal Sync Polarity
If set to zero (0), the horizontal sync is a signal going from low to high (0 -> 1).
If set to one (1), the horizontal sync is a signal going from high to low (1 -> 0).

For some VGA monitors following table indicates the vertical resolution used with corresponding Horizontal and Vertical sync values:

VSYNC Polarity	HSYNC Polarity	Vertical Size
0(+)	0(+)	Reserved
0(+)	1(-)	400
1(-)	0(+)	350
1(-)	1(-)	480

Page Select
If display memory configuration is in so called in odd/even mode, internally only odd or even memory addresses are used. The selection between odd and even memory addresses is done according to the value of Page Select bit.
If Page Select bit is set to ‘1’, only even memory locations are accessed in odd/even modes.
If Page Select is set to ‘0’, only odd memory locations are accessed in odd/even modes.

See also GFX5[4] for setting VGA to odd/even modes and GFX6[1] for switching into chain odd/even modes.

Clock Select
This field selects the pixel clock. Internally selected pixel clock frequencies are:

MISCOUT[3]	MISCOUT[2]	Frequency
0	0	25 MHz
0	1	28 MHz
1	0	External
1	1	External

Writing to this field reprograms video clock to a certain frequency. User must take care of setting to other pixel clock values than 25 or 28 MHz through modifying the system video clock, from which the video clock frequency is actually derived. Writing value 2 or 3 to this bit-field means *external clock frequency* and does not change the video clock frequency.

Enable VGA Display Memory
This bit enables or disables enables VGA memory accesses from the host. This bit must be set to ‘1’, to obtain access to the VGA memory.

Port I/O Base
If set to zero, VGA emulates Monochrome I/O Addresses. If monochrome I/O addresses are used, the color I/O ports are not decoded and vice versa. The port mappings in either mode are:

Field:	0	1
INPUTS1	3BA	3DA
FEATCTRL	3BA	3DA
CRTCINDEX	3B4	3D4
CRTCDATA	3B5	3D5

FEATCTRL - Feature Controller	offset 005Ah / 006Ah	Standard VGA

Access

Read Address	3CAh
Write Address	3DAh (color), 3BAh (mono)
Index	-
Access Type	R/W

Format

7	6	5	4	3	2	1	0
R				Vss	R		

Fields

Field	Bits	Description
R	7:4	Reserved
Vss	3	Vertical sync select
R	2:0	Reserved

Register overall description:
This register contains vertical sync control bit. The write port address of the register is determined by MISCOUT[0].

Field description:

Vertical sync select
If set to zero (0), normal vertical sync is generated. If set to one (1), vertical sync is logical OR of the vertical sync and the vertical display enable. Vertical display enable is controlled by CRTC12, CRTC07[1], CRTC07[6] and CRTC40[7]

INPUTS0 - Input Status 0	offset 52h / -	Standard VGA

Access

Read Address	3C2h
Write Address	-
Index	-
Access Type	R

Format

7	6	5	4	3	2	1	0
VRI	FS1	FS0	SS	R			

Fields

Field	Bits	Description
VRI	7	Vertical Retrace Interrupt
FS1	6	Feature Status 1
FS0	5	Feature Status 0
SS	4	Switch Sense
R	3:0	Reserved

Register overall description:
If vertical interrupts are enabled, the status of the interrupt line can be read through this register. Software can use Switch Sense bit to determine the type of the connected monitor. This register is also used when writing to Attribute registers: dummy reading from this register resets ATTRIDX to point to index.

Field description:

Vertical Retrace Interrupt
Reports the status of vertical interrupt. If vertical interrupt has been generated, it must be cleared by the interrupt handler using register CRTC11.
1 = Vertical interrupt is pending
0 = Interrupt line clear

Feature Status 1
Hardwired to zero (0).

Feature Status 0
Hardwired to zero (0).

Switch Sense
Reports the status of the switch sense inside the DAC. This field can be used to determine the monitor type. Typically, software uses Switch Sense to determine whether monochrome or color monitor is connected. This is done by driving a high intensity color values through the DAC. If red, green or blue wire to the monitor is not connected, the current will go so high that the switch sense will be enabled. Since the output is implemented as inverted, the actual Switch Sense value goes low.

INPUTS1 - Input Status 1	offset 006Ah / -	Standard VGA

Access

Read Address	3DAh (color), 3BAh (mono)
Write Address	-
Index	-
Access Type	R

Format

7	6	5	4	3	2	1	0
R		D		VR	LPSw	LPSt	DEN

Fields

Field	Bits	Description
R	7:6	Reserved
D	5:4	Diagnostic
VR	3	Vertical Retrace
LPSw	2	Light Pen Switch
LPSt	1	Light Pen Strobe
DEN	0	Display Enable Not

Register overall description:
This register contains debugging lines for color palette registers, vertical retrace and display enable bits. Vertical Retrace and Display Enable Not are used by software to synchronize to the screen refresh.

Field description:

Diagnostic
Diagnostic field indicates the value of two of the eight address lines to color palette. The address lines which are read are be selected by ATTR12[4-5] according to the following table:

ATTR12[5]	ATTR12[4]	INPUTS1[5]	INPUTS1[4]
0	0	line 2	line 0
0	1	line 5	line 4
1	0	line 3	line 1
1	1	line 7	line 6

Vertical Retrace
1 = vertical retrace is occurring
0 = vertical retrace is not occurring

Light Pen Switch
Hardwired to one (1).

Light Pen Strobe
Hardwired to zero (0).

Display Enable Not
0 = video is in display mode
1 = either blank or border is active

8.6.2 Sequencer Registers

SEQINDEX - Sequencer Index Register	offset 0054h	Standard VGA

Access

Read Address	3C4h
Write Address	3C4h
Index	-
Access Type	R/W

Format

7	6	5	4	3	2	1	0
I							

Fields

Field	Bits	Description
I	7:0	Index

Register overall description:
This register specifies index of the sequencer register to be accessed with the next I/O read or write operation to port 3C5h.

SEQ0 - Sequencer Reset	offset 0055h	Standard VGA

Access

Read Address	3C5h
Write Address	3C5h
Index	0
Access Type	R/W

Format

7	6	5	4	3	2	1	0
R						SR	AR

Fields

Field	Bits	Description
R	7:2	Reserved
SR	1	Syncronous Reset
AR	0	Asyncronous Reset

Register overall description:
Sequencer reset register. This register is implemented for compatibility only, and does not affect the functionality of VGA Core.

Field description:

Synchronous Reset
0 Hold sequencer in reset state
1 Release reset

Asynchronous Reset
0 Hold sequencer in reset state
1 Release reset

SEQ1 - Clocking Mode	offset 0055h	Standard VGA

Access

Read Address	3C5h
Write Address	3C5h
Index	1h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
R		SO	SbF	DC	SbT	B	8/9 DC

Fields

Field	Bits	Description
R	7:6	Reserved
SO	5	Screen Off
SbF	4	Shift by Four
DC	3	Dot Clock
SbT	2	Shift by Two
B	1	Bandwidth
8/9 DC	0	8/9 Dot Clocks

Register overall description:
Clocking mode register defines some important characteristics of the refresh. One can turn screen off to achieve greater memory bandwidth to frame buffer from the host side. Shift by four is used to divide serializer load frequency by four. Dot Clock is used to divide the pixel clock by two for displaying low resolution modes. Shift by Two is used to divide the serializer load frequency by two. 8 or 9 pixel wide characters can be selected using this register in alphanumeric modes.

Field description:

Screen Off
The screen off prevents the display refresh logic from accessing the frame buffer. This results in greater bandwidth to the memory from the host side, and can be used during high speed memory transfer.
0 Screen on
1 Screen off

Shift by Four
0 Load serializers at every character cycle
1 Load serializers at every fourth character cycle

Dot Clock
If this bit is set to ‘1’, dot clock is divided by two, and two consequent pixels are output with same color. It is used to create low resolution modes, for example 320 pixels per scanline. If set to zero, dot clock is not affected.

Shift by Two
0 Load serializers at every character cycle
1 Load serializers at every second character cycle, if Shift by Four is not used.

Bandwidth
Writing to this bit has no effect. It's purpose is to force the memory bandwidth between host and refresh interfaces.

8/9 Dot Clocks
1 character width is 8 pixels.
0 character width is 9 pixels.
9 pixel characters can be used only in alphanumeric modes. Selection between graphics and alphanumeric modes is done by ATTR10[0].

SEQ2 - Plane Mask	offset 0055h	Standard VGA

Access

Read Address	3C5h
Write Address	3C5h
Index	2h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
R				PM			

Fields

Field	Bits	Description
R	7:4	Reserved
PM	3:0	Plane Mask

Register overall description:
The register selects the planes which can be accessed by the standard VGA host write operations.

Field description:

Plane Mask
If bit corresponding to plane number is '1', the plane can be accessed by host write operations. Correspondingly, bit '0', corresponding to plane number, means that the plane can't be accessed by host write operations. Bit zero stands for plane '0' and bit three stands for plane '3'. Plane mask is not used in linear write mode.

SEQ3 - Character Map Select	offset 0055h	Standard VGA

Access

Read Address	3C5h
Write Address	3C5h
Index	3h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
R		SAH	SAB	SA		SB	

Fields

Field	Bits	Description
R	7:6	Reserved
SAH	5	SAH
SAB	4	SAB
SA	3:2	SA
SB	1:0	SB

Register overall description:
This register selects the character maps used. Character map A is used if the character attribute bit 3 is '0'. Character map B is used if bit 3 is '1'. Normally, the two character maps are the same (and the register is programmed to zero). Position of the character map is calculated by the following formulas:

Character map A start = SA $\times$ 16384 + SAH $\times$ 8192
Character map B start = SB $\times$ 16384 + SBH $\times$ 8192

Character map is resided in the memory plane 3, and each character consists of 32 consequent bytes. Thus, a single 256 character map requires 8192 bytes of memory.

SEQ4 - Memory Mode	offset 0055h	Standard VGA

Access

Read Address	3C5h
Write Address	3C5h
Index	4
Access Type	R/W

Format

7	6	5	4	3	2	1	0
R				CF	OE	EMP	GAm

Fields

Field	Bits	Description
R	7:4	Reserved
CF	3	Chain Four
OE	2	Odd/Even
EMP	1	Extended Memory Present
GAm	0	Graphics/Alphanumerics mode

Register overall description:
Memory Mode register controls host side odd/even mode behavior and has memory size flag indicating the size of the video memory. Chain Four is used to enable special host mode, where four display planes are chained together.

Field description:

Chain Four
If this bit is set to '1' address to VGA memory is formed in a special way. Two low order bits of address are ignored and will select the display plane where the data is written or read. Two most significant bits of the address will become two least significant bits. For example, if the memory write is to address 8007h, then the data is written to plane 3 of memory address 8005h. This corresponds the double word addressing mode in VGA refresh, enabled by CRTC14[6].Data is written only if corresponding bit in map mask (SEQ2) is enabled. This setting takes priority over chain odd/even in GFX6[1] and odd/even in GFX5[4].

0 Normal operation
1 Chain Four mode

Odd/Even
This bit selects between odd/even and normal addressing modes. The value of GFX5[4] should always be set to complement of this bit.

0 Odd/even enabled
1 Odd/even disabled

Extended Memory Present

0 Extended memory not present, memory size 64Kb
1 Extended memory present, memory size > 64Kb

8.6.3 CRTC Registers

CRTCINDEX - CRTC Register Index	offset 0064h	Standard VGA

Access

Read Address	3D4h (color), 3B4h (mono)
Write Address	3D4h (color), 3B4h (mono)
Index	-
Access Type	R/W

Format

7	6	5	4	3	2	1	0
I							

Fields

Field	Bits	Description
I	7:0	Index

Register overall description:
This register specifies the CRTC register which is accessed through port 3B5h/3D5. The mapping of CRTCINDEX and other CRTC registers is determined by MISCOUT[0]. Writes to CRTC registers 0-7 can be disabled by setting CRTC11[7] to ‘1’.

CRTC00 - Horizontal Total	offset 0065h	Standard VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	0
Access Type	R/W

Format

7	6	5	4	3	2	1	0
HT							

Fields

Field	Bits	Description
HT	7:0	Horizontal Total

Register overall description:
This register together with CRTC40[2] determines the screen width - 5 in characters including borders and blanking. This register can be written only if CRTC11[7] is zero.

Field descriptions:

Horizontal Total
These bits together with CRTC40[2] determine the total width of display area - 5. This includes borders and blanking. The actual resolution depends on character width, which may be 8 or 9 pixels.

CRTC01 - Horizontal Display End	offset 0065h	Standard VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	1h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
HDE							

Fields

Field	Bits	Description
HDE	7:0	Horizontal Display End

Register overall description:
This register together with CRTC40[3] determine the total width -1 of display area in characters.
This register can be written only if CRTC11[7] is zero.

Field descriptions:

Horizontal Display End
These bits together with CRTC40[3] determine the visible display area - 1 in characters. The actual resolution depends on character width, which may be 8 or 9 pixels.

CRTC02 - Horizontal Blanking Start	offset 0065h	Standard VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	2h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
HBS							

Fields

Field	Bits	Description
HBS	7:0	Horizontal Blanking Start

Register overall description:
This register together with CRTC40[0] determines the start of horizontal blanking period.
This register can be written only if CRTC11[7] is zero

Field descriptions:

Horizontal Blanking Start
These bits together with CRTC40[0] determine the start of the horizontal blanking period in characters.

CRTC03 - Horizontal Blanking End	offset 0065h	Standard VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	3h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
R	DES		HBE				

Fields

Field	Bits	Description
R	7	Reserved
DES	6:5	Display Enable Skew
HBE	4:0	Horizontal Blanking End

Register overall description:
This register together with CRTC05[7], CRTC41[1-2], CRTC41[7] determine the end of horizontal blanking period in characters. This register can be written only if CRTC11[7] is zero.

Field descriptions:

Display Enable Skew
Defines the number of characters by which horizontal display enable is delayed.

Horizontal Blanking End
These bits together with CRTC05[7], CRTC41[1-2] and CRTC41[7] determine the end of horizontal blanking period.

If CRTC41[7] is ‘0’, the horizontal blanking period ends, when the character counter’s 6 lowest bits do equal the 6 low-order bits of horizontal blanking end value. With this setting bits form CRTC41[1-2] are not included into horizontal blanking end value. This is the standard VGA operation.

If CRTC41[7] is ‘1’ horizontal blanking ends, when 8 low-order bits correspond the Horizontal Blanking End value. With this setting bits CRTC41[1-2] are included in Horizontal Blanking End value.

CRTC04 - Horizontal Sync Start	offset 0065h	Standard VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	4h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
HSS							

Fields

Field	Bits	Description
HSS	7:0	Horizontal Sync Start

Register overall description:
This register together with CRTC40[1] determines the start of horizontal retrace period in character clocks. This register can be written only if CRTC11[7] is zero.

Field descriptions:

Horizontal Sync Start
These bits together with CRTC40[1] determine the start of the horizontal retrace period in characters.

CRTC05 - Horizontal Sync End	offset 0065h	Standard VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	5h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
HBE5	HRD		HRE				

Fields

Field	Bits	Description
HBE5	7	Horizontal Blanking End[5]
HRD	6:5	Horizontal Retrace Delay
HRE	4:0	Horizontal Retrace End

Register overall description:
This register determines the end of horizontal retrace period, horizontal retrace delay and the sixth bit of end horizontal blanking value. This register can be written only if CRTC11[7] is zero.

Field descriptions:

Horizontal Blanking End[5]
This is the sixth bit of horizontal blanking end value. See CRTC03 for details.

Horizontal Retrace Delay
This is the number of characters to delay the horizontal retrace. These bits are added to the horizontal retrace start value.

Horizontal Retrace End
This is a MOD 32 value determining end of the horizontal retrace period. When 5 low-order bits of the character counter equal Horizontal Retrace End value, the horizontal retrace period ends.

CRTC06 - Vertical Total	offset 0065h	Standard VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	6h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
VT							

Fields

Field	Bits	Description
VT	7:0	Vertical Total

Register overall description:
This register together with CRTC07[0], CRTC07[5] and CRTC40[6] determine the total height of display -2 including borders and blanking. This register can be written only if CRTC11[7] is zero

Field descriptions:

Vertical Total
These bits together with CRTC07[0], CRTC07[5] and CRTC40[6] determine the total height of display -2 including borders and blanking.

CRTC07 - CRTC Overflow Register	offset 0065h	Standard VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	7h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
VSSB9	VDEB9	VTB9	LCB8	VBSB8	VSSB8	VDEB8	VTB8

Fields

Field	Bits	Description
VSSB9	7	Vertical Sync Start Bit 9
VDEB9	6	Vertical Display End Bit 9
VTB9	5	Vertical Total Bit 9
LCB8	4	Line Compare Bit 8
VBSB8	3	Vertical Blanking Start Bit 8
VSSB8	2	Vertical Sync Start Bit 8
VDEB8	1	Vertical Display End Bit 8
VTB8	0	Vertical Total Bit 8

Register overall description:
This register includes bits which extend vertical counters. This register can be written only if CRTC11[7] is zero, with the exception of bit 4 which can be written normally regardless of CRTC11[7] setting.

Field descriptions:

Vertical Sync Start Bit 9
The ninth bit of vertical sync start (CRTC10). Other extension bits are CRTC07[2] and CRTC40[5].

Vertical Display End Bit 9
The ninth bit of vertical display end (CRTC12). Other extension bits are CRTC07[1] and CRTC40[7].

Vertical Total Bit 9
The ninth bit of vertical total (CRTC06). Other extension bits are CRTC07[0] and CRTC40[6].

Line Compare Bit 8
The ninth bit of line compare (CRTC18). Other extension bits are CRTC09[6] and CRTC41[0]. This bit can be written even if CRTC11[7] is ‘1’.

Vertical Blanking Start Bit 8
The eighth bit of vertical blanking start (CRTC15). Other extension bits are CRTC09[5] and CRTC40[4].

Vertical Sync Start Bit 8
The eighth bit of vertical sync start (CRTC10). Other extension bits are CRTC07[7] and CRTC40[5].

Vertical Display End Bit 8
The eight bit of vertical total (CRTC06). Other extension bits are CRTC07[5] and CRTC40[6].

Vertical Total Bit 8
The eighth bit of vertical display end. Other extension bits are CRTC07[5 and CRTC40[6].

CRTC08 - Preset Row Scan	offset 0065h	Standard VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	8h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
R	BP		PRS				

Fields

Field	Bits	Description
R	7	Reserved
BP	6:5	Byte Panning
PRS	4:0	Preset Row Scan

Register overall description:
This register controls character horizontal byte panning by adding 0-3 to the address of the first character on the screen. Smooth vertical scrolling can be done using Preset Row Scan, which determines the first displayed scanline of the first character row.

Field descriptions:

Byte Panning
This field defines how many characters are panned from the left edge of the screen. If the value of this field is ‘0’, screen is displayed normally. If the value is ‘3’, then first three characters are skipped. The result is screen being scrolled left 3 characters.

Preset Row Scan
This field determines the first displayed scanline on the first character row. Using this register, it is possible to make smooth vertical scrolling across a character.

CRTC09 - Character Cell Height	offset 0065h	Standard VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	9h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
CSD	LCB9	VBSB9	CH				

Fields

Field	Bits	Description
CSD	7	CRTC Scan Double
LCB9	6	Line Compare Bit 9
VBSB9	5	Vertical Blanking Start Bit 9
CH	4:0	Character Height

Register overall description:
CRTC Scan Double allows doubling of the scanlines, dividing the vertical resolution by 2. Two extension bits, for Line Compare and Vertical Blanking, are in bits 5 and 6. Character height is determined by bits 0 to 4.

Field descriptions:

CRTC Scan Double
If this bit is ‘1’, every scanline is displayed twice, dividing the vertical resolution by 2.

Line Compare Bit 9
The tenth bit of line compare field. See CRTC18 for details.

Vertical Blanking Start Bit 9
The tenth bit of vertical blank start. See CRTC15 for details.

Character Height
This field determines the height of the character. Character height is between 1-32 pixels. The value in this field is Character Height - 1.

CRTC0A - Cursor Start	offset 0065h	Standard VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	Ah
Access Type	R/W

Format

7	6	5	4	3	2	1	0
R		CH	CS				

Fields

Field	Bits	Description
R	7:6	Reserved
CH	5	Cursor Hide
CS	4:0	Cursor Start

Register overall description:
This register disables/enables the cursor and defines the first scanline of the cursor.

Field descriptions:

Cursor Hide
0 cursor on
1 cursor off.

Cursor Start
This field determines the starting scanline of the cursor inside character box. if cursor start is greater than cursor end (CRTC0B), cursor is not displayed.

CRTC0B - Cursor End	offset 0065h	Standard VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	Bh
Access Type	R/W

Format

7	6	5	4	3	2	1	0
R	CS		CE				

Fields

Field	Bits	Description
R	7	Reserved
CS	6:5	Cursor Skew
CE	4:0	Cursor End

Register overall description:
Defines the scanline where cursor ends. Cursor Skew field specifies how many characters cursor is skewed to right.

Field descriptions:

Cursor Skew
The number of characters the cursor is delayed from the cursor start address (CRTC0E and CRTC0F).

Cursor End
This field determines the ending scanline of the cursor inside character box. if cursor end is smaller than cursor start (CRTC0A), cursor is not displayed.

CRTC0C - Start Address High	offset 0065h	Standard VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	Ch
Access Type	R/W

Format

7	6	5	4	3	2	1	0
SSAH							

Fields

Field	Bits	Description
SSAH	7:0	Screen Start Address High

Register overall description:
This register together with CRTC0D and CRTC43[0-4] define display start address.

Field descriptions:

Screen Start Address High
Bits 8 to 15 of the display start address

CRTC0D - Start Address Low	offset 0065h	Standard VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	Dh
Access Type	R/W

Format

7	6	5	4	3	2	1	0
SSAL							

Fields

Field	Bits	Description
SSAL	7:0	Screen Start Address Low

Register overall description:
This register together with CRTC0C and CRTC43[0-4] define the location in display memory where the screen refresh starts.

Field descriptions:

Screen Start Address Low
Bits 0 to 7 of the display start address

CRTC0E - Cursor Location High	offset 0065h	Standard VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	Eh
Access Type	R/W

Format

7	6	5	4	3	2	1	0
CLH							

Fields

Field	Bits	Description
CLH	7:0	Cursor Location High

Register overall description:
This register together with CRTC0F determines the cursor's location in the display memory.

Field descriptions:

Cursor Location High
Bits 8 to 15 of the cursor location.

CRTC0F - Cursor Location Low	offset 0065h	Standard VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	Fh
Access Type	R/W

Format

7	6	5	4	3	2	1	0
CLL							

Fields

Field	Bits	Description
CLL	7:0	Cursor Location Low

Register overall description:
This register together with CRTC0E determines the cursor's location in the display memory.

Field descriptions:

Cursor Location Low
Bits 0 to 7 of the cursor location.

CRTC10 - Vertical Sync Start	offset 0065h	Standard VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	10h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
VSS							

Fields

Field	Bits	Description
VSS	7:0	Vertical Sync Start

Register overall description:
This register determines the eight least-significant bits of the vertical sync start value. The other bits can be found from CRTC07[2], CRTC07[7] and CRTC40[5].

Field descriptions:

Vertical Sync Start
Bits 0 to 7 of the vertical sync start. The other bits can be found from CRTC07[2], CRTC07[7] and CRTC40[5].

CRTC11 - Vertical Sync End	offset 0065h	Standard VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	11h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
PR0-7	B	DVI	CVI	VSE			

Fields

Field	Bits	Description
PR0-7	7	Protect Registers 0-7
B	6	Bandwidth
DVI	5	Disable Vertical Interrupt
CVI	4	Clear Vertical Interrupt
VSE	3:0	Vertical Sync End

Register overall description:
This register determines the end of the vertical retrace period. Registers CRTC00-CRTC07 can be protected from writing by bit 7. Vertical interrupt disable and clear flags are in bits 4 and 5.

Field descriptions:

Protect Registers 0-7
If set to ‘1’, the registers from CRTC00 to CRTC07 are protected from writing, with the exception of CRTC07[4]. if set to ‘0’ registers can be written normally

Disable Vertical Interrupt
If set to ‘1’ the vertical interrupt is disabled, and INPUTS0[7] never creates vertical interrupt flag..
If set to ‘0’, interrupt is generated normally.

Clear Vertical Interrupt
When set to ‘0’ the vertical interrupt flag in **INPUTS0** [7] is cleared, and interrupt can not occur. When set to ‘1’ interrupt can occur again.

Vertical Sync End
This register determines the end of vertical retrace period. When bits 0-3 of row scan counter equal these bits the vertical retrace period ends.

CRTC12 - Vertical Display End	offset 0065h	Standard VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	12h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
VDE							

Fields

Field	Bits	Description
VDE	7:0	Vertical Display End

Register overall description:
This register defines eight least-significant bits of display vertical resolution. Other bits can be
found from CRTC07[1], CRTC07[6] and CRTC40[7].

Field descriptions:

Vertical Display End
Bits 0 to 7 vertical display end value.

CRTC13 - Offset Register	offset 0065h	Standard VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	13h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
O							

Fields

Field	Bits	Description
O	7:0	Offset

Register overall description:
This register defines least-significant bits of display memory offset value. This is the difference between successive scanlines in display memory. Extended bits of the offset value are CRTC43[5-7].

Field descriptions:

Offset
These bits define how many bytes difference exists between successive scanlines. The actual value is multiplied by two, four or eight, depending on the addressing mode.

CRTC14 - Underline Register	offset 0065h	Standard VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	14h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
R	DWM	CbF	UL				

Fields

Field	Bits	Description
R	7	Reserved
DWM	6	Double Word Mode
CbF	5	Count by Four
UL	4:0	Underline Location

Register overall description:
This register defines the position of the underline in character. The Count by Four and double word addressing modes are controlled by this register.

Field descriptions:

Double Word Mode
If this bit is set to ‘1’, double word addressing is used. In double word addressing, address to frame buffer increments in 4 byte steps. This is achieved by rotating the address to the frame buffer left by 2. By using this approach, two most significant bytes will become two least significant bytes. If double word addressing is not enabled CRTC17[6] controls whether byte or word addressing is used.

Count by Four
If this bit is set to ‘1’, address to the frame buffer is incremented on every fourth character clock. This is normally used together with CRTC14[6] in 256-color modes to allow the sequencer to process all the four pixels loaded from a single double word aligned address. If this bit is set to ‘0’, character counter is incremented normally.

Underline Location
This field specifies the scanline inside the character box, where the underlining occurs.

CRTC15 - Vertical Blank Start	offset 0065h	Standard VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	15h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
VBS							

Fields

Field	Bits	Description
VBS	7:0	Vertical Blank Start

Register overall description:
The register defines start of the vertical blanking period.

Field descriptions:

Vertical Blank Start
This register defines bits 0 to 7 of the vertical blanking start value. Other bits can be found from CRTC07[3], CRTC09[5] and CRTC40[4].

CRTC16 - Vertical Blank End	offset 0065h	Standard VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	16h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
VBE							

Fields

Field	Bits	Description
VBE	7:0	Vertical Blank End

Register overall description:
The register defines end of the vertical blanking period.

Field descriptions:

Vertical Blank End
When vertical counters 8 least-significant bits correspond Vertical Blank End value, the vertical blanking period ends.

CRTC17 - Mode Control Register	offset 0065h	Standard VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	17h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
HWR	WBM	AW	R	CT	DVC	SRSC	CMS

Fields

Field	Bits	Description
HWR	7	Hardware reset
WBM	6	Word/Byte Mode
AW	5	Address Wrap
R	4	Reserved
CT	3	Count by two
DVC	2	Double Vertical Counters
SRSC	1	Select Row Scan Counter
CMS	0	Compatibility Mode Support

Register overall description:
The register defines end of the vertical blanking period.

Field descriptions:

Hardware Reset
0 refresh logic is deactivated.
1 refresh is activated.

Word/Byte Mode
If this bit is ‘0’ frame buffer addresses are rotated left by one and the frame buffer is accessed in two byte (word) steps. This setting takes priority over double word addressing (CRTC14[6]). When the address is rotated most significant byte gets to the least significant byte

If the bit is ‘1’, frame buffer address is not multiplied by two and frame buffer is accessed in byte steps. However, if CRTC14[6] is enabled, then double word addressing is used.

Address Wrap
If Word addressing (CRTC17[6]) is used, this field determines whether address is rotated or simply shifted. If set to ‘1’ rotation is done to 16-least significant bits of the address. This means, that the bit 15 is rotated to bit 0. If set to ‘0’ rotation is done to 14-least significant bits of the address. This means, that the bit 13 is rotated to bit 0.

Count by Two
If set to '1', counter to frame buffer is incremented on every other character clock. This is for refresh cycles only. This setting takes priority over Count by Four in CRTC14[5].

Double Vertical Counters
1 vertical counter values are doubled by incrementing vertical scanline counters at every other horizontal retrace.
0 vertical counters are clocked normally.

Select Row Scan Counter
This bit is provided for hercules compatibility.
0 scanline counter bit 1 is substituted for frame buffer address bit 14.
1 no substitution is performed.

Compatibility Mode Support
If set to '0', frame buffer address bit 13 is substituted for scanline counter bit 0. This provides for CGA compatibility.
If set to '1', no substitution is performed.

CRTC18 - CRTC Line Compare	offset 0065h	Standard VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	18h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
LC							

Fields

Field	Bits	Description
LC	7:0	Line Compare

Register overall description:
This register sets the scanline from where the screen refreshing gets back to display memory location 0.

Field descriptions:

Line Compare
This field defines 8 least-significant bits of the line compare value. Other bits can be found from CRTC07[4] and CRTC09[6] andCRTC41[0]. When scanline counter equals line compare the refresh starts from memory location 0. The ATTR10[5] selects whether pixel panning is reset to zero or not during when line compare match occurs.

CRTC40 - CRTC Extension Register 1	offset 0065h	Extended VGA

Access

Read Address	**3D5h (color), 3B5h (mono)**
Write Address	**3D5h (color), 3B5h (mono)**
Index	**40h**
Access Type	**R/W**

Format

7	6	5	4	3	2	1	0
VDEb1	**VTb1**	**VSSb1**	**VBSb1**	**HDEb8**	**HTb8**	**HSSb8**	**HBSb8**

Fields

Field	Bits	Description
VDEb1	7	Vertical Display End bit 10
VTb1	6	Vertical Total bit 10
VSSb1	5	Vertical Sync Start bit 10
VBSb1	4	Vertical Blank Start bit 10
HDEb8	3	Horizontal Display End bit 8
HTb8	2	Horizontal Total bit 8
HSSb8	1	Horizontal Sync Start bit 8
HBSb8	0	Horizontal Blanking Start bit 8

Register overall description:
This register extends VGA horizontal and vertical refresh counters.

CRTC41 - CRTC Extension Register 2	offset 0065h	Extended VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	41h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
HBE	AA	R	SAM	LA	HBE		LCB1

Fields

Field	Bits	Description
HBE	7	Horizontal Blanking Extension
AA	6	Address Aliasing
R	5	Reserved
SAM	4	Sequencer Addressing Mode
LA	3	Linear Addressing
HBE	2:1	Horizontal Blanking Extension
LCB1	0	Line Compare Bit 10

Register overall description:
This register extends some VGA refresh counters so that larger displays can be defined.

Field descriptions:

Horizontal Blanking Extension
If this bit is set to ‘0’, horizontal blank end acts as MOD 64 counter to horizontal character clock. In this mode, the horizontal blanking period ends, when 6 low-order bits correspond the horizontal blanking end 6 low-order bits from CRTC3[0-4] and CRTC5[7].

If set to ‘1’, horizontal blank end acts as MOD 256 counter to horizontal character clock. In this mode, the horizontal blanking period ends, when 8 low-order bits correspond the horizontal blanking end 6 low-order bits from CRTC3[0-4], CRTC5[7] and CRTC41[1-2].

See CRTC03 for details.

Address Aliasing
0 frame buffer accesses are aliased to 256kb memory.
1 whole memory can be accessed through the VGA memory window using banking registers.

Sequencer Addressing Mode
If this bit is set to ‘1’ the sequencer is extended to 64 bit to provide faster 8-bit linear refresh. 8 or 9 8-bit pixels are loaded per sequencer fill cycle.
If this bit is set to ‘0’ the sequencer is in VGA mode.

Linear Addressing
1 frame buffer is accessed linearly.
0 frame buffer is accessed in VGA fashion.

Horizontal Blanking Extension
If horizontal blanking extension is used (CRTC41[7]), these bits are the bits 6-7 of the Horizontal Blanking End value.

CRTC42 - CRTC Extension Register 3	offset 0065h	Extended VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	42h
Access Type	R

Format

7	6	5	4	3	2	1	0
R							VRE

Fields

Field	Bits	Description
R	7:1	Reserved
VRE	0	VGA Refresh Enable

Register overall description:
This is read only register that indicates whether VGA refresh is enabled. If this bit is zero, true/high color refresh is active.

Field descriptions:

VGA Refresh Enable
1 VGA Refresh is enabled.
0 Refresh is in 16-bit or higher mode.

CRTC43 - CRTC Extension Register 4	offset 0065h	Extended VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	43h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
ORB			DSA				

Fields

Field	Bits	Description
ORB	7:5	Offset Register Bits 8-10
DSA	4:0	Display Start Address bits 16-20

Register overall description:
This register extends display start address and offset registers.

Field descriptions:

Offset Register, bits 8-10
Details in CRTC13.

Display Start Address bits 16-20
Details in CRTC0C and CRTC0D.

CRTC44 - Read Bank Start Address	offset 0065h	Extended VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	44h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
RBSA							

Fields

Field	Bits	Description
RBSA	7:0	Read Bank Start Address

Register overall description:
This register determines VGA memory read bank’s start address.

Field descriptions:

Read Bank Start Address
When VGA memory is read, the actual address is address + (Read Bank Start Address) × 65536.

CRTC45 - Write Bank Start Address	offset 0065h	Extended VGA

Access

Read Address	3D5h (color), 3B5h (mono)
Write Address	3D5h (color), 3B5h (mono)
Index	45h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
WBSA							

Fields

Field	Bits	Description
WBSA	7:0	Write Bank Start Address

Register overall description:
This register determines VGA memory write bank's start address.

Field descriptions:

Write Bank Start Address
When VGA memory is written, the actual address is address + (Write Bank Start Address) $\times$ 65536.

8.6.4 Graphics Registers

GFXINDEX - Graphics Register Index	offset 005Eh	Standard VGA

Access

Read Address	3CEh
Write Address	3CEh
Index	-
Access Type	R/W

Format

7	6	5	4	3	2	1	0
I							

Fields

Field	Bits	Description
I	7:0	Index

Register overall description:
This register specifies the register to be accessed by the next I/O read or write to address 3CFh.

GFX0 - Set / Reset Register	offset 005Fh	Standard VGA

Access

Read Address	3CFh
Write Address	3CFh
Index	0h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
R				SRP			

Fields

Field	Bits	Description
R	7:4	Reserved
SRP	3:0	Set/Reset Plane

Register overall description:
One of settings for write mode 0 or 3.

Field descriptions:

Set/Reset Plane
In write mode 0, if enable set/reset (GFX1) is enabled for corresponding plane, then the plane is written with the value of the bit assigned to the plane in this field.

In write mode 3 each plane is written with the value of the bit assigned to the plane in this field, before ALU operations, latch combination and plane masking are done.

GFX1 - Enable Set / Reset Register	offset 005Fh	Standard VGA

Access

Read Address	3CFh
Write Address	3CFh
Index	1h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
R				ESR			

Fields

Field	Bits	Description
R	7:4	Reserved
ESR	3:0	Enable Set/Reset

Register overall description:
One of settings for write mode 0.

Field descriptions:

Enable Set/Reset
In write mode 0, if enable set/reset is enabled for corresponding plane, then the plane is written with the value of the bit assigned to the plane in GFX0[0-3]

GFX2 - Color Compare	offset 005Fh	Standard VGA

Access

Read Address	3CFh
Write Address	3CFh
Index	2h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
				CC			

Fields

Field	Bits	Description
	7:4	Reserved
CC	3:0	Color Compare

Register overall description:
One of settings for read mode 1.

Field descriptions:

Color Compare
This register defines the color that is compared against latch bytes in read mode 1. If the color matches, then corresponding bit is ‘0’, otherwise, ‘1’. (this applies only for 8 pixels/byte modes). The color’s bit can be forced to match with GFX7[0-3], color don’t care.

GFX3 - Data Rotate	offset 005Fh	Standard VGA

Access

Read Address	3CFh
Write Address	3CFh
Index	3h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
R			AFS		DR		

Fields

Field	Bits	Description
R	7:5	Reserved
AFS	4:3	ALU Function Select
DR	2:0	Data Rotate

Register overall description:
Selects ALU function for write modes 0,2 and 3 and data rotation for write modes 0 and 3.

Field descriptions:

ALU Function Select
ALU functions are operational in write modes 0, 2 and 3.
0 = no operation
1 = AND written data with latches
2 = OR written data with latches
3 = XOR written data with latches

Data Rotate
Write mode 0 and 3 specific setting for rotating data before it’s written. The data is rotated right.

GFX4 - Read Map	offset 005Fh	Standard VGA

Access

Read Address	3CFh
Write Address	3CFh
Index	4h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
R						RM	

Fields

Field	Bits	Description
R	7:2	Reserved
RM	1:0	Read Map

Register overall description:
Selects read map in read mode 0.

Field descriptions:

Read Map
This field specifies the map that is read from the address. Applies only for read mode 0.

GFX5 - Mode Register	offset 005Fh	Standard VGA

Access

Read Address	3CFh
Write Address	3CFh
Index	5h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
R	SM		OE	RM	R	WM	

Fields

Field	Bits	Description
R	7	Reserved
SM	6:5	Sequencer Mode
OE	4	Odd/Even
RM	3	Read Mode
R	2	Reserved
WM	1:0	Write Mode

Register overall description:
Miscellaneous VGA host side functions are defined by this register.

Field descriptions:

Sequencer Mode
These bits define how the sequencer is loaded to for palette or color palette accesses
0 = standard VGA output format
1 = CGA output format
2 = MCGA output format

Odd/Even
If this bit is set to '1', graphics controller is set to odd/even mode. This means that odd memory accesses address odd memory planes, and even memory accessed address even memory planes. This bit should be set to complement of SEQ4[2] to enable odd/even addressing.

Read Mode
If set to '0', read mode 0 is used. In read mode 0, single plane (determined by GFX4) is read. If display is in chain four mode, odd-even or chain odd/even mode, then the plane read is determined similarly to write.

If set to '1', read mode 1 is used. In read mode 1, color don't care (GFX7) and color compare (GFX2) are used to determine how data is read.

Write Mode
Defines which of the four write modes is used.

GFX6 - Miscellaneous Register	offset 005Fh	Standard VGA

Access

Read Address	3CFh
Write Address	3CFh
Index	6h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
R				MM		COE	GM

Fields

Field	Bits	Description
R	7:4	Reserved
MM	3:2	Memory Map
COE	1	Chain Odd Even
GM	0	Graphics Mode

Register overall description:
Miscellaneous VGA host side functions are defined by this register.

Field descriptions:

Memory Map
This field specifies the memory map of the VGA.

Memory Map Value	Memory Start	Memory End
0	A0000h	BFFFFh
1	A0000h	AFFFFh
2	B0000h	B7FFFh
3	B8000h	BFFFFh

Chain Odd Even
If this bit is set to ‘1’, then even addresses access planes 0 and 2, and odd addresses access planes 1 and 3.
If this bit is set to ‘0’, no chaining occurs.

Graphics Mode
0 alphanumeric mode of operation
1 graphical mode of operation

GFX7 - Color Don't Care	offset 005Fh	Standard VGA

Access

Read Address	3CFh
Write Address	3CFh
Index	7h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
R				CDt C			

Fields

Field	Bits	Description
R	7:4	Reserved
CDt C	3:0	Color Don't Care

Register overall description:
Read mode 1 specific register.

Field descriptions:

Color Don’t Care
This register is used in conjunction with GFX2, in read mode 1. Setting a bit to ‘1’ means that corresponding plane is taken into comparison. Setting a bit to ‘0’ means that corresponding plane is ignored, as if it had matched.

GFX8 - Write Mask	offset 005Fh	Standard VGA

Access

Read Address	3CFh
Write Address	3CFh
Index	8h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
WM							

Fields

Field	Bits	Description
WM	7:0	Write Mask

Register overall description:
Bit mask for writing. Applies to modes 0 and 2.

Field descriptions:

Write Mask
In write mode 0, these bits control whether corresponding bit is written to the frame buffer or not.
1 write
0 do not write.
In write mode 2, these bits select which of the bits are written from the host data and which are taken from the latches.
1 host data
0 latched data.

8.6.5 Attribute Controller Registers

ATTRIDX - Attributer Index	offset 0051h / 0050h	Standard VGA

Access

Read Address	3C0h
Write Address	3C0h
Index	-
Access Type	R/W

Format

7	6	5	4	3	2	1	0
R		ERA	I				

Fields

Field	Bits	Description
R	7:6	Reserved
ERA	5	Enable Refresh Access
I	4:0	Index

Register overall description:
This register defines the index to the attributer registers. ATTRIDX has internal flip flop, so that every write switches between attribute register index and attribute register data. The filp flop is resetted to point to index by I/O read from INPUTS1 register. If bit 5 of this register is zero, attribute registers are locked from refresh logic. This means that every time attribute register is accessed, the bit 5 must be set to ‘1’ enable screen refresh.

ATTRPAL - Palette Registers	offset 0051h / 0050h	Standard VGA

Access

Read Address	3C1h
Write Address	3C0h
Index	0-fh
Access Type	R/W

Format

7	6	5	4	3	2	1	0
R		CI					

Fields

Field	Bits	Description
R	7:6	Reserved
CI	5:0	Color Index

Register overall description:
Each of these registers define VGA palette color for a 16-color palette index, that is indexes to 256-color palette registers.

Upper 2 bits of the color palette index are taken from ATTR14[2-3], and if ATTR10[7] is ‘1’, the bits 4-5 of the color palette index are taken from ATTR14[0-1]. The extension for bits 4-5 do not apply for 256 or higher color modes.

ATTR10 - Attribute Controller Mode	offset 0051h / 0050h	Standard VGA

Access

Read Address	3C1h
Write Address	3C0h
Index	10h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
IPS	PDCS	PPC	R	BE	LGE	DT	GA

Fields

Field	Bits	Description
IPS	7	Internal Palette Size
PDCS	6	Pixel Double Clock Select
PPC	5	Pixel Panning Compatibility
R	4	Reserved
BE	3	Blink Enable
LGE	2	Line Graphics Enable
DT	1	Display Type
GA	0	Graphics/Alphanumeric

Register overall description:
This register determines various settings for refresh logic.
See ATTRIDX for information about writing to attribute registers.

Field descriptions:

Internal Palette Size
If this field is ‘1’, the bits 4-5 of the palette register value are taken from ATTR14[0-1]. In 256 color modes this register is ignored.

Pixel Double Clock Select
If this field is selected, the attribute controller bypasses palette registers and pixels are generated from 8-bit index formed by two consequent 4-bit values from the sequencer. This means, that it requires two cycles to generate a single pixels from 4 bit wide sequencer, and pixel of same color is displayed two times before the color can change.

Pixel Panning Compatibility
If set to ‘1’, line compare match will reset the pixel panning value to ‘0’. This makes possible to scroll the upper partition of the screen independently.
If set to ‘0’, line compare does not affect the scrolling.

Blink Enable
1 character blinking is enabled.
0 character blinking is disabled.

Line Graphics Enable
This field applies only for 9-bit wide characters in alphanumeric modes.
If this bit is set to ‘1’, the ninth bit is copied from the eight bit for character codes in the range C0h-DFh.
If this bit is set to ‘0’, the ninth bit is set to same as the background color.

Display Type
If this bit is set to ‘1’, monochrome display attributes are used. The attribute codes for the monochrome adapter are:

Attribute Code	Attribute
7h	Normal
Fh	Intense
1h	Underline
9h	Underline intense
70h	Reverse
F0h	Blinking to Reverse

Graphics/Alphanumeric
0 alphanumeric mode
1 graphics mode

ATTR11 - Overscan Color Register	offset 0051h / 0050h	Standard VGA

Access

Read Address	3C1h
Write Address	3C0h
Index	11h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
OC							

Fields

Field	Bits	Description
OC	7:0	Overscan Color

Register overall description:
This register determines the border color, which is defined as color between display end and blanking.
See ATTRIDX for information about writing to attribute registers.

Field descriptions:

Overscan Color
8 bit color palette index to overscan color.

ATTR12 - Color Plane Enable Register	offset 0051h / 0050h	Standard VGA

Access

Read Address	3C1h
Write Address	3C0h
Index	12h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
R		VSM		CPE			

Fields

Field	Bits	Description
R	7:6	Reserved
VSM	5:4	Video Status MUX
CPE	3:0	Color Plane Enable

Register overall description:
Register has mux for diagnostic field in INPUTS1 and mask for display planes.
See ATTRIDX for information about writing to attribute registers.

Field descriptions:

Video Status MUX
Selects the lines used for diagnostic field in INPUTS1[4-5].

Color Plane Enable
If corresponding plane is set to ‘1’, the plane is enabled for the refresh accesses.
If corresponding plane is set to ‘0’, the plane can’t be accessed by the video refresh.

ATTR13 - Horizontal Pixel Panning	offset 0051h / 0050h	Standard VGA

Access

Read Address	3C1h
Write Address	3C0h
Index	13h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
R				HPP			

Fields

Field	Bits	Description
R	7:4	Reserved
HPP	3:0	Horizontal Pixel Pan

Register overall description:
This register defines how many pixels characters are panned horizontally.
See ATTRIDX for information about writing to attribute registers.

Field descriptions:

Horizontal Pixel Pan
This field defines how many pixels screen is scrolled to the left. In 9-dot wide alphanumeric modes the screen is scrolled the field value - 1 pixels, and with value ‘0’ eight pixels.

ATTR14 - Color Select Register	offset 0051h / 0050h	Standard VGA

Access

Read Address	3C0h
Write Address	3C1h
Index	14h
Access Type	R/W

Format

7	6	5	4	3	2	1	0
R				CPA67		CPA45	

Fields

Field	Bits	Description
R	7:4	Reserved
CPA67	3:2	Color Palette Address, Bits 6 and 7
CPA45	1:0	Color Palette Address, Bits 4 and 5

Register overall description:
The bits of this field extend the ATTRPAL register values to full 8-bit color palette index. See ATTRIDX for information about writing to attribute registers.

Field descriptions:

Color Palette Address, Bits 6 and 7
These bits extend the palette index from ATTRPAL registers to full 8-bit palette index. In 256-color, or higher, modes this field is ignored.

Color Palette Address, Bits 4 and 5
If ATTR10[7] is ‘1’, this the palette register bits 4-5 are substituted for these bits.

8.6.6 Color Palette Registers

CPWADDR - Color Palette Write Address	offset 0058h	Standard VGA

Access

Read Address	3C8h
Write Address	3C8h
Index	-
Access Type	R/W

Format

7	6	5	4	3	2	1	0
CPWA							

Fields

Field	Bits	Description
CPWA	7:0	Color Palette Write Address

Register overall description:
Write address to the color palette registers.

Field descriptions:

Color Palette Write Address
Selects one of the 256 color palette registers for writing. Writing is done through CPDATA register. Write to this field resets the palette index to point to red color component.

CPRADDR - Color Palette Read Address	offset - / 0057h	Standard VGA

Access

Read Address	-
Write Address	3C7h
Index	-
Access Type	W

Format

7	6	5	4	3	2	1	0
CPRA							

Fields

Field	Bits	Description
CPRA	7:0	Color Palette Read Address

Register overall description:
Read address to the color palette registers.

Field descriptions:

Color Palette Read Address
Selects one of the 256 color palette registers for reading. Reading is done through CPDATA register. Write to this field resets the palette index to point to red color component.

CPDATA - Color Palette Data	offset 0059h	Standard VGA

Access

Read Address	3C9h
Write Address	3C9h
Index	-
Access Type	R/W

Format

7	6	5	4	3	2	1	0
R		CCV					

Fields

Field	Bits	Description
R	7:6	Reserved
CCV	5:0	Color Component Value

Register overall description:
The palette entries are read and written through this port.

Field descriptions:

Color Component Value
The color palette consists of 256 18-bit registers having intensities for red, green and blue. Palette index points to one color component of a register. When data is read or written the index autoincrements
to point to next component. The first component is red, and the last is blue. When the blue component has been read or written the index moves to the next 18-bit register, or if the register index overflows, returns to the register 0.

CPSTATE - Color Palette State	offset 0057h / -	Standard VGA

Access

Read Address	3C7h
Write Address	-
Index	-
Access Type	R

Format

7	6	5	4	3	2	1	0
R						SR	

Fields

Field	Bits	Description
R	7:2	Reserved
SR	1:0	State Register

Register overall description:
Reports the status of the color palette accesses.

Field descriptions:

State Register
0 = Color palette write register (CPWADDR) was accessed last
3 = Color palette read register (CPRADDR) was accessed last

CPMASK - Color Palette Mask	offset 0056h	Standard VGA

Access

Read Address	3C6h
Write Address	3C6h
Index	-
Access Type	R/W

Format

7	6	5	4	3	2	1	0
CPM							

Fields

Field	Bits	Description
CPM	7:0	Color Palette Mask

Register overall description:
Color palette address lines can be forced to zero with this register.

Field descriptions:

Color Palette Mask
The final color palette index is anded with this mask before the RGB value read from the color palette register. Setting a bit in this field to ‘0’ clears the palette index's address line. Usually this field is programmed to FFh to enable all the 256-color palette indexes.

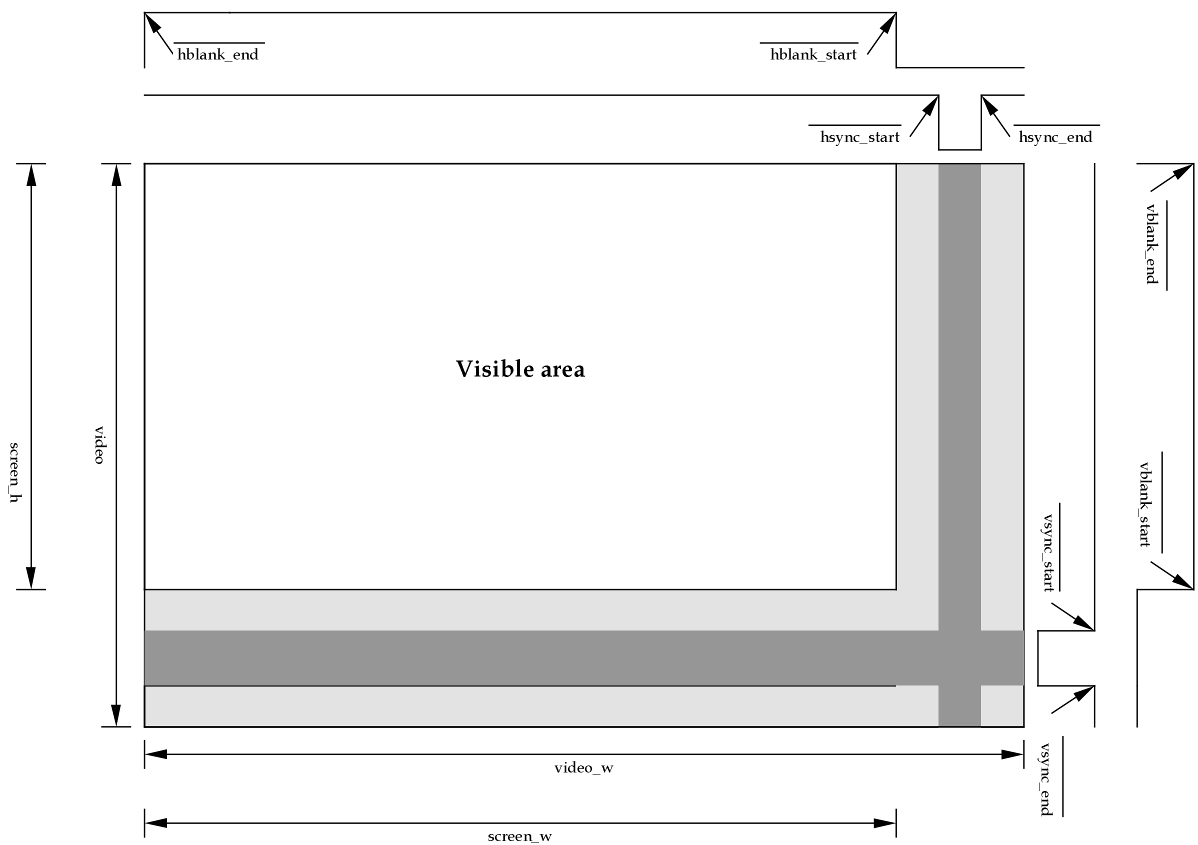
9. Video Control

9.1 Overview

VS25203 provides a full internal video control logic unit. The video refresh logic supports 16-bit hi-color and 24-bit true-color display formats with all resolutions from 320 × 200 to 1600 × 1200 pixels. Note also that it is possible to use other screen ratios than the normal 4:3 screen aspect ratio. The only restriction is that blank areas have to be after a visible area.

9.2 Refresh Timing

The following figure illustrates the relationship between horizontal and vertical timing signals. Terms used in the figure are the fields of the video registers.



`hblank_end`, `hblank_start` are fields of the `video_hblank` register.

`hsync_end`, `hsync_start` are fields of the `video_hsync` register.

`vblank_end`, `vblank_start` are fields of the `video_vblank` register.

`vsync_end`, `vsync_start` are fields of the `video_vsync` register.

`video_w`, `video_h` are fields of the `video_w_h` register.

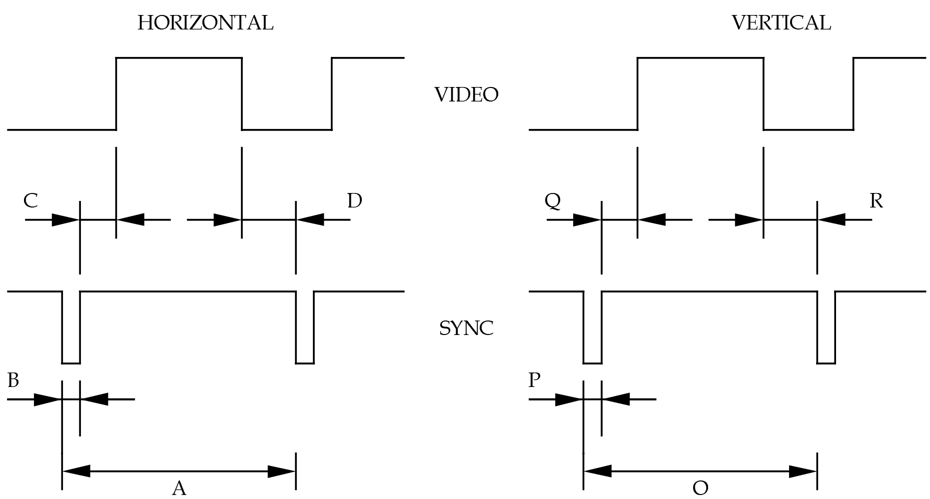
`screen_w`, `screen_h` are fields of the `screen_w_h` register.

Refer to the video interface register definitions on page 211. Note that all count type registers uses the convention of count-from-0, and not count-from-1; once the maximum count is reached, the value wraps around to 0.

9.3 640 × 480 Calculation Example

VESA standards have predefined timing parameters for a set of chosen screen resolutions. For non-VESA resolutions, this section illustrates the procedures for determining the values to be placed into the fields of the VS25203 video registers which are defined and described on page 211.

The monitor manual's spec page will usually give the following information:



Horizontal:

A	Scanline duration: 31.778 µs
B	Sync duration: 3.813 µs
C	Back porch: 1.589 µs
D	Front porch: 0.953 µs

Vertical:

O Frame duration : 16.683 ms
P Sync duration : 64 µs
Q Back porch : 1.017 ms
R Front porch : 350 µs

f_v Vertical frequency: $\frac{1}{16.683ms}$ = 59.94 Hz

1. Calculate the estimated display size, which is, as a rule of thumb, about:
$640 \times 1.25 = 800$
$480 \times 1.25 = 600$

2. Calculate the clock frequency:
$800 \times 600 \times 59.94$ Hz = 28.8 MHz

3. Horizontal front porch:
0.953µs × 28.8 MHz = 27 pixels

4. Horizontal sync duration:
3.813µs × 28.8 MHz = 110 pixels

5. Horizontal back porch:
1.589µs × 28.8 MHz = 46 pixels

6. Calculate `video_w`:
640 + 27 + 110 + 46 = 823 pixels

7. Vertical front porch:
350µs × 28.8 MHz = 10080 pixels

This means: $\frac{10080\,pixels}{823\,{}^{pixels}/_{line}} = 12$ lines

8. Vertical sync duration:
64 µs × 28.8 MHz = 1843 pixels

This means: $\frac{1843\,pixels}{823\,{}^{pixels}/_{line}} = 2$ lines

9. Vertical back porch:
1.017 µs × 28.8 MHz = 29290 pixels

This means: $\frac{29290\,pixels}{823\,{}^{pixels}/_{line}} = 35$ lines

10. `video_h` size:
480 + 12 + 2 + 35 = 529 lines

11. Insert the values into appropriate registers:

`screen_w` = 640	register 34, page 212
`screen_h` = 480	register 34, page 212
`video_w` = 823	register 33, page 212
`video_h` = 529	register 33, page 212
`hblank_start` = 641	register 36, page 214
`hsync_start` = 668	register 38, page 215
`hsync_end` = 778	register 38, page 215
`hblank_end` = 0	register 36, page 214
`vblank_start` = 481	register 35, page 213
`vsync_start` = 493	register 37, page 214
`vsync_end` = 495	register 37, page 214
`vblank_end` = 0	register 35, page 214

12. Calculate the clock coefficients for the desired clock frequency. The video clock frequency can be calculated from the formula:

$$F_{OUT} = \frac{m\_coef + 2}{(n\_coef + 2) \times 2^{r\_coef}} \times 14.3181818\text{MHz}$$

where:

n_coef, m_coef and *r_coef* are count coefficients for the on-chip frequency synthesizer.

With `m_coef` = 126, `n_coef` = 14 and `r_coef` = 2, we get a pixel frequency of 28.6MHz.

13. Insert the coefficient values into `video_clk_cfg` (18) register, page 33.

9.4 Video Interface Registers

Register address	Offset	Register name
33	`0084h`	video_width_height
34	`0088h`	screen_width_height
35	`008Ch`	video_vblank
36	`0090h`	video_hblank
37	`0094h`	video_vsync
38	`0098h`	video_hsync
39	`009Ch`	video_base_conf
40	`00A0h`	video_bit_config
41	`00A4h`	reserved

video_w_h	register 33	offset 0084h

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
										video_h					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
video_h					video_w										

Fields

Field	Bits	Description
video_w	10:0	video_width
video_h	21:11	video_height

video_w_h register specifies the size of the area scanned by the video-x and video-y counters. The visible screen occupies a portion of this memory, starting from the (0,0)-point.

video_w

Specifies the last value which video-x counter reaches. This value is the total width minus one. For example, if 800 is desired for the video total width, value 799 is specified in this field.

video_h

Specifies the last value which video-y counter reaches. This value is the total height minus one. For example, if 525 is desired for the video total height, value 524 is specified in this field.

screen_w_h	register 34	offset 0088h

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
										screen_h					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
screen_h					screen_w										

Fields

Field	Bits	Description
screen_w	10:0	screen_width
screen_h	21:11	screen_height

screen_w_h registers specifies the size of the actual displayed screen. Pixels are sent to the display as long as the values of video-x counter is less than `screen_w` and as long as the video-y counter is less than `screen_h`.

screen_w

Specifies the width of the displayed screen minus one. (*See* `video_w_h` register 33, `video_w` field). For example, if 640 is desired for the screen width, value 639 is specified in this field.

screen_h

Specifies the height of the displayed screen minus one. (*See* `video_w_h` register 33, `video_h` field). For example, if 400 is desired for the screen height, value 399 is specified in this field.

video_vblank	register 35	offset 008Ch

Format

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

Fields

Field	Bits	Description
vblank_end	10:0	vertical blank end
vblank_start	21:11	vertical blank start

video_vblank register specifies the timing of the vertical blank signal relative to the video-y counter.

vblank_end

Specifies the video-y counter value which ends the vertical blank signal. This value is the vertical blank end minus one. (*See* `video_w_h` register 33, `video_h` field). For example, if 490 is desired for the vertical blank to end, value 489 is specified in this field.

vblank_start

Specifies the video-y counter value which starts the vertical blank signal. This value is the vertical blank start minus one. (*See* `video_w_h` register 33, `video_h` field). For example, if 410 is desired for the vertical blank to start, value 409 is specified in this field.

The blank area can be made to overlap the screen area. It is also possible to initialize `vblank_end` to a lower value than `vblank_start`. This causes the blank area to wrap around the bottom of the video coordinates.

video_hblank	register 36	offset 0090h

Format

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

Fields

Field	Bits	Description
hblank_end	10:0	horizontal blank end
hblank_start	21:11	horizontal blank start

video_hblank register specifies the timing of the horizontal blank signal relative to the video-x counter.

hblank_end

Specifies the video-x counter value which ends the horizontal blank signal. This value is the horizontal blank end minus one. (*See* `video_w_h` register 33, `video_w` field). For example, if 790 is desired for the horizontal blank to end, value 789 is specified in this field.

hblank_start
Specifies the video-x counter value which starts the horizontal blank signal. This value is the horizontal blank start minus one. (*See* `video_w_h` register 33, `video_w` field). For example, if 650 is desired for the horizontal blank to start, value 649 is specified in this field.
The blank area can be made to overlap the screen area. Also it is possible to initialize the `hblank_end` to lower value than `hblank_start`. This causes the blank area to wrap around the right edge of the video coordinates.

video_vsync	register 37	offset 0094h

Format

31–22	21–16
	vsync_start

15–11	10–0
vsync_start	vsync_end

Fields

Field	Bits	Description
vsync_end	10:0	vertical sync end
vsync_start	21:11	vertical sync start

vsync_end
Specifies the video-y counter value which ends the vertical sync signal. This value is the vertical sync end minus one. (*See* `video_w_h` register 33, `video_h` field). For example, if 480 is desired for the vertical sync to end, value 479 is specified in this field.
vsync_start
Specifies the video-y counter value which starts the vertical sync signal. This value is the vertical sync start minus one. (*See* `video_w_h` register 33, `video_h` field). For example, if 420 is desired for the vertical sync to start, value 419 is specified in this field.
The sync area can be made to overlap the screen area. It is also possible to initialize the `vsync_end` to a lower value than `vsync_start`. This causes the sync area to wrap around the bottom edge of the video coordinates.

video_hsync	register 38	offset 0098h

Format

31–22	21–16
	hsync_start

15–11	10–0
hsync_start	hsync_end

Fields

Field	Bits	Description
hsync_end	10:0	horizontal sync end
hsync_start	21:11	horizontal sync start

hsync_end
Specifies the video-x counter value which ends the horizontal sync signal. This value is the horizontal sync end minus one. (*See* `video_w_h` register 33, `video_w` field). For example, if 780 is desired for the horizontal sync to end, value 779 is specified in this field.

hsync_start
Specifies the video-x counter value which starts the horizontal sync signal. This value is the horizontal sync start minus one. (*See* `video_w_h` register 33, `video_w` field). For example, if 660 is desired for the horizontal sync to start, value 659 is specified in this field.
The sync area can be made to overlap the screen area. Also it is possible to initialize the `hsync_end` to lower value than `hsync_start`. This causes the sync area to wrap around the right edge of the video coordinates.

video_base_conf	register 39	offset 009Ch

Format

Bits	31–22	21–16
		mh

Bits	15	14	13–0
	mh		scr_addr

Fields

Field	Bits	Description
scr_addr	13:0	Screen base address
mh	21:15	Screen memory height

The **video_base_conf** register contains information about how the screen data is stored in the memory.

mh
The screen memory height can be calculated with the following formulas:
mh = Screen height / SIZE
16-bit pixels: SIZE = 32
32-bit pixels: SIZE = 16

scr_addr
Specifies the start address of the screen memory as a multiple of 2048 bytes.

video_bit_config	register 40	offset 00A0h

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
						hbm	vwm	dpl	dpp	dt	hbp	vbp	hsp	vsp	pw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	Bits	Description
pw	0	Pixel width 16-bit / 32-bit
vsp	1	Vertical sync. polarity
hsp	2	Horizontal sync. polarity
vbp	3	Vertical blank polarity
hbp	4	Horizontal blank polarity
dt	5	DAC test (Reserved)
dpp	6	Duplicate pixel
dpl	7	Duplicate line
vwm	8	Video width msb
hbm	9	Hblank width msb

video_bit_config register contains a collection of bits used for configuring the behavior of the video interface.

pw

0 pixel width is 16 bits

1 pixel width is 32 bits

vsp

Specifies the polarity of the `vsync` signal.

hsp

Specifies the polarity of the `hsync` signal.

vbp

Specifies the polarity of the `vblank` signal.

hbp

Specifies the polarity of the `hblank` signal.

dpp

If set to one each line is twice. Useful for displaying low resolution screen on a high resolution display

Dpl

If set to one each pixel displayed twice. Useful for displaying low resolution screen on a high resolution display

vwm

MSB for video width register (Added to cover high resolution modes)

hbm

MSB for horizontal register (Added to cover high resolution modes).

10. TV Output Unit

10.1 Overview

TV output unit works parallel with video refresh and can be turned on/off whether TV-signaling is used. Required screen size as well as synchronization signaling for the TV are programmed to the Video refresh block. Video-refresh output is fed to TV-output unit that performs interlacing and flicker filtering to produce final output signaling.

10.2 Usage

The TV-output unit is controlled through PCI Configuration Space Register 21 (`feat_reg`). The TV-output unit is turned on by setting `ffe` field. This starts interlacing process as well as flicker filtering. Flicker filter threshold (field `fft`) is adjustable and there exists also 100 Hz TV set flicker filter enhancement field `ffm`.

Note: Flicker filter halves the line frequency and doubles the horizontal blank time. It does not add the horizontal sync. This must be taken into account when setting the video parameters.

10.3 TV Output Unit Register

PCI Configuration Space Register 21, feature register, is presented below.

feat_reg	register 21	offset 0054h

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
														ddv	euio
ffe	ffm		fft									vee	vrsl	vrs	vde
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	Bits	Description
ddv	17	disable digital video
euio	16	enable user I/O [6:5]
ffe	15	flicker filter enable
ffm	14	flicker filter mode
fft	12:8	flicker filter thresh
vee	3	VGA extension enable
vrsl	2	VGA refresh select lock
vrs	1	VGA refresh select
vde	0	VGA decode enable

ffe
Flicker filter enable, a bit for activating the flicker filter and interlace module.
ffm
Flicker filter mode, affects the mode of operation for the flicker filter.
0 default value, optimal in most cases.
1 modified algorithm, which might provide better results on 100/120 Hz televisions.
fft
Flicker filter threshold, threshold value for flicker filtering 0 means no threshold (filter always), 16 means no filtering (perform interlace conversion still).

11. Video Capture Unit

11.1 Overview

The video capture block is totally independent functional block, which stores data captured through digital RGB pins into on-board memory for further use. 4:2:2 YUV data format described in ITU-R BT.656-3 standard is supported.

11.2 Usage

The video capture unit takes the inputs from digital RGB (set the disable digital video-bit to "1" from the Feature Register, register 21) and user_io[0] pins. When 8 bit mode is used captured data is read from pin b[7:0] and in 16 bit mode 8 MSB bits are read from g[7:0].

ITU-R BT.656-3 standard capture data stream contains information about vertical and horizontal synchronization, but optionally external synchronization can be used (`capt_w_h` -register `ssel` -field). Vertical synchronization is read through r[1] pins and horizontal r[0] respectively. The Rising edge of these signals is considered as start of the line/field (hor/ver). When using this mode current field information is carried in r[2] pin.

The video capture unit uses external clock that is driven through user_io[0] pin. Capture data sampling is done at the rising edge of this clock. The capture unit is working up to 35 MHz (capture clock). If `capt_w_h` -register bit `cq` is on, data sampling is done when clock qualifier signal r[3] is active.

Captured data is stored in 4:2:2 YUV format into on-board memory location defined by capture base address.

If the interrupts are enabled (`capt-base-conf`- reg `irq1` and `irq2` fields), the video capture unit indicates its interrupts by setting `capi` field of `status` (48) register. The interrupt can be acknowledged by writing 1 to the same field.

11.3 Video Capture Unit inputs

Input	**Description**
b[7:0]	capture_data_in[7:0]
g[7:0]	capture_data_in[15:8] if 16-bit mode is used
r[0]	vertical sync_in (if ssel bit is set)
r[1]	horizontal sync_in (if ssel bit is set)
r[2]	field information (contains the field (odd/even) field identification) 0 during field1, 1 during field2 (if ssel bit is set)
r[3]	clock qualifier
user_io[0]	capture_clk

11.4 Video Capture Unit Registers

Register address	Offset	Register name
31	007Ch	capt_base_conf
32	0080h	capt_w_h

capt_base_conf	register 31	offset 007Ch

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ena	**irq1**	**irq2**							**mem_height**						
		base_addr													
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	Bits	Description
ena	31	video capture enable
irq1	30	Interrupt request 0 enable
irq2	29	Interrupt request 1 enable
mem_height	22:16	Memory height
base_addr	13:0	Base address

capt_base_conf contains control bits for video capture block.

ena

Video capture enable/disable (1/0).

Irq1

This field enables/disables interrupt of detected odd fields from the capture source.

Irq2

This field enables/disables interrupt of detected even fields from the capture source.

mem_height

This field gives the memory height of the target in 2048 byte blocks.

base_addr

The capture base address specifies the start address of the capture memory as multiple of 2048 bytes.

capt_w_h	register 32	offset 0080h

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
cdt	cq	ssel	dei			cap_height									
						cap_width									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	Bits	Description
cdt	31	Capture data type
cq	30	capture clock qual
ssel	29	capture sync hs vs
dei	28	capture deinterlace
cap_height	26:16	capture height
cap_width	9:0	capture width

capt_w_h contains information to control the video capture block. For typical ITU-R BT.656-3 standard usage `cdt` = 0, `cq` = 0 and `ssel` =0.

cdt

The capture data type

0 8 bit

1 16 bit.

cq

Selection bit whether the clock qualifier input is used in data synchronisation.

ssel

Selection bit whether the synchronisation signals are used during capture process.

dei

This bit defines whether both of the fields are captured or the odd frame duplicated and capture is done at half speed.

Cap_height

Cap height defines the video capture area height in pixels.

Cap_width

Cap width defines the video capture area width in pixels.

12. Block Transfer Unit

12.1 Overview

VS25203 includes totally independent Block Transfer Unit, which performs area fill and copy operations as well as bit copy operations.

12.2 Usage

The Block Transfer Unit is controlled by register 56 - 63. After writing the register 63 the unit starts the operation defined by other register. Status information can be read from `status` register (register 48) `blti` field. Byte base addressing is used with all the Block Transfer Unit addresses.

12.3 Block Transfer Unit Registers

Register Number	Address Offset	Register name	Description
56	00E0h	blt_src_strd	Source stride
57	00E4h	blt_tgt_strd	Target stride
58	00E8h	blt_fg_color	Foreground color
59	00ECh	blt_bg_color	Background color
60	00F0h	blt_params	Parameters
61	00F4h	blt_src_addr	Source address
62	00F8h	blt_tgt_addr	Target address
63	00FCh	blt_size	Block size

blt_src_strd	register 56	offset 00E0h

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
								x64_source_stride							
			Y_source_stride												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	Bits	Description
x64_source_stride	23:16	Source stride of x-direction
Y_source_stride	12:0	Source stride of y-direction

blt_src_strd contains information block source strides.
X64_source stride
This field gives x-direction offset for physical memory address.
Y_source_stride
This fiels gives the offset for physical memory when stepping in y-direction.

blt_tgt_strd	register 57	offset 00E4h

Format

31–24	23–16
	x64_target_stride

15–13	12–0
	Y_target_stride

Fields

Field	Bits	Description
x64_target_stride	23:16	Target stride of x-direction
Y_target_stride	12:0	Target stride of y-direction

blt_tgt_strd contains information block target strides.
X64_target stride
This field gives x-direction offset for physical memory address.
Y_target_stride
This fiels gives the offset for physical memory when stepping in y-direction.

blt_fg_color	register 58	offset 00E8h

Format

31–16
fg_color

15–0
fg_color

Fields

Field	Bits	Description
fg_color	31:0	Foreground color

Blt_fg_color contains information about foreground color when making bit copy operation.
Fg_color
Specifies foreground color. When using 16 or 8 bit color modes the whole register should be filled by duplicating the desired color. Alternatively different colors can be specified for the vertical lines on the screen by specifying different values to the 8 and 16 bit sections of the register.

blt_bg_color	register 59	offset 00ECh

Format

31–16
bg_color

15–0
bg_color

Fields

Field	Bits	Description
bg_color	31:0	Background color

Blt_bg_color contains information about background color when making bit copy operation.

Bg_color
Specifies background color. When using 16 or 8 bit color modes the whole register should be filled by duplicating the desired color. Alternatively different colors can be specified for the vertical lines on the screen by specifying different values to the 8 and 16 bit sections of the register.

blt_params	register 60	offset 00F0h

Format

31	30	29	28	27	26	25	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
						blt_oper								pxl_type	

Fields

Field	Bits	Description
blt_oper	9:8	Block transfer operation
pxl_type	1:0	Pixel type

Blt_params contains general information for Block Transfer Unit.
Blt_oper
Defines the operation:
00 fill
Fill the target are with foreground color.
01 copy
Copies data from the source area to target area.
10 bit copy (fg and bg)
Makes bit copy operation using both foreground and background color
11 bit copy (only fg)
Make bit copy operation using only foreground. Background bytes are left empty.
Pxl_type
Defines pixel type for operation:
00 8 bit (VGA)
01 16 bit (3D)
10 32 bit (3D)

blt_src_addr	register 61	offset 00F4h

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
src_addr															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
src_addr															

Fields

Field	Bits	Description
src_addr	31:0	Source address

Blt_src_addr contains address for source data.
Src_addr The byte address for the first source byte to be handled.

blt_tgt_addr	register 62	offset 00F8h

Format

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

Fields

Field	Bits	Description
tgt_addr	31:0	Target address

Blt_tgt_addr contains address for target of data.
Tgt_addr The byte address for the first target byte.

blt_size	register 63	offset 00FCh

Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ydir					height										
xdir					width										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fields

Field	Bits	Description
ydir	31	Y direction
height	26:16	height of the block
xdir	15	X direction
width	10:0	width of the block

Blt_size contains information about the block size and direction of the specified operation.
ydir defines y direction for the operation.
0 from top to bottom
1 from bottom to top.
xdir defines x direction for the operation.
0 form left to right
1 from right to left
height defines the height of the block.
Width defines the width of the block.

13. Internal / External DAC

VS25203 contains an internal triple 8-bit Video DAC, which has a maximum operating frequency of 200MHz.

It is also possible to use an external Video DAC with the following features:

triple 8-bit D/A converters
TTL compatible inputs
construction optionally +5 V or +3.3 V.

14. Application Notes

14.1 PCI Bus Rerference Design

14.1.1 Introduction

The schematic examples and layout guidelines referred in this section are intended for engineers implementing the VS25203 board. The information presented here is for reference only and is subject to change. Designers should contact VLSI Solution for the latest schematics and further information before production.

The schematics represent a sample PCI Bus implementation of VS25203 with detailed discussions of components and their board placement. The layout discussion provides guidelines for specific layout issues such as analog and digital ground separation and recommended trace width restrictions.

14.1.2 Power-Up Configuration Summary

On system reset or power-up, the video subsystem configuration information is latched into VS25203's internal configuration registers from the data stored in the on-board EPROM. This data is read using the pixel bus data lines, BTB[0..7] for data and BTR[0..7] and BTG[0..7] for address. The EPROM is enabled with the USE_ROM signal from VS25203. The on-board Video BIOS contains data for the following:

Enable VS25203
Program DAC/Clock to a value of 90MHz
Set display memory data path width (32 or 64 bits) and DRAM type

14.1.3 Content of EPROM

The last eight addresses from the EPROM are automatically loaded into the internal registers as follows:

```
0     subsystem vendor id bits  7:0
1     subsystem vendor id bits 15:8
2     subsystem id        bits  7:0
3     subsystem id        bits 15:8
4     memory config reg   bits  7:0
5     memory config reg   bits 15:8
6     memory config reg   bits 23:16
7     memory config reg   bits 31:24
```

(address 7 means the last address of the EPROM)

The last 8 addresses are implemented by VS25203 to access locations FFF8h – FFFFh. If a 32KB or smaller EPROM is used the top-most address bits are ignored. The maximum EPROM size is 64 KB. The EPROM contents can also be accessed using the normal PCI expansion ROM access mechanism. The EPROM address is provided by the digital Red/Green/Blue (RGB) pins so that the R-bus contains the top (MSB) bits, and G-bus the lower (LSB) bits. B-bus is used for reading the data.

14.1.4 PCI Bus Interface

VS25203 is designed for a glueless interface to the PCI bus. The pins on VS25203 are directly connected to similarly named pins on the PCI bus. This is summarized below.

PCI Signal Names	
Address/data bus	AD[31..0]
	C/BE[3..0]#
	PAR
Control	FRAME#
	STOP#
	IRDY#
	TRDY#
	IDSEL
	DEVSEL#
System	CLK
	RST#
Bus Master Control	REQ#/
	GNT#
Interrupt	INTA#
Error Reporting	PERR#
	SERR#

The pin assignments on VS25203 are carefully optimized to allow short and direct connections between the bus pins and VS25203 pins. VS25203 should be placed within an inch of the PCI connector and approximately centered on the connector.

14.1.5 Memory Interface

VS25203 features a fully integrated 64-bit synchronous DRAM memory interface. VS25203 supports 256K×16 EDO DRAM, SDRAM and SGRAM memory chips. The memory size can range from 2 MB to 32MB.

Memory timing adjustment through software will be clarified in the next revision.

DRAM Interface Signal Names

Signal Name	BankA	BankB
Address	AA0[11..0]	BA0[11..0]
Data	ADQ[15..0]	BDQ[15..0]
Control	ARAS	BRAS
	ACAS	BCAS
	AWE	BWE
	ACS0	BCS0
	AMEMCLK	BMEMCLK
	ADQM[3..0]	BDQM[3..0]

14.1.6 Monitor Interface

Proper signal conditioning with carefully selected component values is critical for providing good crisp video at high frequencies and minimizing EMC (radio frequency interference) emissions.

RGB Lines

RGB lines are nominally terminated in 75Ω to DAC ground, thus providing half of the 37.5Ω DC load; the other half is in the monitor. Z filters on each RGB line control edge rates and reduce EMC to an acceptable level. The z filter's cutoff frequency should be as high as possible to prevent signal degradation but as low as possible to provide for reduced emissions. The 75Ω RGB termination resistors should be located as close as possible to VS25203 and the Z filters should be located very close to the output DB-15 connector. The traces between VS25203 and the filters should be direct, with no vias or sharp corners. These traces must be designed with a characteristic impedance as close as possible to 75Ω. During high refresh rate operations, the signal edge rates are fast enough that a trace as short as a few inches begins to behave as a transmission line.

Sync Lines

The `hsync` and `vsync` signals are isolated with in-line 75Ω resistors. Future VS_VP reference designs will rely on LC filters of ferrite bead (17Ω at 100 MHz) and 220-pF capacitor to further reduce EMC emissions. The LC filter outputs connect directly to the DB-15 output connector.

DDC2B Support

The graphics subsystem requires information on the monitor's display capabilities for selecting optimum refresh rates. This information is obtained from the monitor via a serial bi-directional bus from VS25203 to the monitor. VS25203 provides a serial clock (SCL) and reads serial data (SDA) from a VESA DDC2B compliant monitor.

14.1.7 Power Distribution and Conditioning

The most common reason for poor quality video is the failure on the part of the board designer to properly manage power distribution and conditioning. For this reason, dedicated power and ground planes are very strongly recommended for boards based on VS25203.

VS25203 operates at 3.3 V supply power. PCI bus and video interfaces are also 5 V compatible. Selection is done with pin A6, (AGP / PCI). When using 5 V interfaces additional 5 V pad power is fed through pins D4 and AA11, (VDD_Clamp).

Decoupling capacitors
Bypass capacitors are used to minimize power sags caused by current spikes and reduce the power distribution impedance. Bulk bypassing is present in the area where power comes onto the board, around the DRAM array, and near the EPROM. The bulk bypass is usually a tantalum or an aluminum electrolytic capacitor which at very high frequencies becomes inductive, rendering it unsuitable for fast switching signals. For this reason local bypassing capacitors are distributed as needed next to each high-speed IC. When an un-bypassed IC switches current into a load, the current comes from the supply line, exits the output pin, and flows through the load into the ground line. Any series impedance in the supply and ground lines causes large local glitches in both lines. The role of the bypass capacitor is to supply fast transient currents to the IC, so they do not have to come through the supply-line series impedance.

A bypass capacitor can do its job efficiently only if it is mounted in close proximity to the pins that draw the fast transient currents. And if it is some distance from the IC, the series inductance of the PCB traces gives the transients an opportunity to develop glitches. For this reason uncased multilayer ceramic (MLC) surface mount components are used exclusively in the design. High operating frequencies of the VS25203 board are affected not only by the inductance due to the length of the PCB traces but also the lead length of the bypass capacitors.

Dedicated Ground Plane
A dedicated ground plane minimizes differential ground offsets and more nearly approximates the ideal notion of ground. Additionally, a ground plane is necessary to predict and control the characteristic impedance of those traces that must be treated as transmission lines.

Analog and digital ground separation is very critical for mixed signal devices such as VS25203. The ground plane on the VS25203 design has cuts to partially isolate the critical analog ground sections from the relatively noisy digital ground associated with SDRAM memory and the PCI bus interface. The schematic reflects three ground planes, a digital ground and two isolated analog grounds; one for the DAC and one for the clock synthesizer. Traces for analog grounds should not have any digital connections.

14.1.8 Clock Synthesizer

VS25203 on-chip clock synthesizer requires a quartz crystal of the following characteristics:

Crystal characteristics	
Frequency	14.31818 MHz +/- 0.1% Fundamental resonance
ESR	25 to 45Ω
Load Capacitance	15 to 40 pF, parallel resonance

The crystal should be connected across Osc_in (pin A4) and Osc_out (pin C6) of VS25203. If a 14.318 MHz oscillator is used instead of a crystal, then the clock output of the oscillator should be connected to Osc_in only, and Osc_out should be left open. If a crystal is used, both sides of the crystal should have soldering pads to allow grounding of the case and attaching of the crystal to a quiet ground plane. The parallel resonant crystal requires 22pF balance capacitors and a 1MΩ shunt resistor to initiate stable oscillation.

15. Pinouts and Signal Descriptions

15.1 Pinout

The following two figures describe the pin configuration of VS25203. The chip is packaged in a 304-pin thermally enhanced ball grid array (BGA) package. Signals are grouped so that pins for external memory chips are on both sides of VS25203, pins for the PCI bus are on the lower part of the processor and the remaining pins (DAC, PLL, Osc, etc.) are on the upper part.

To reduce communication delay, it is recommended putting the external memory chips on the right hand side and on the left hand side of the processor.

	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	
A	B[0]	B[1]	B[2]	B[5]	R[0]	R[2]	R[4]	R[7]	G[2]	G[5]	use_rom#	usr_io[2]	vsync_in	vsync	PCLK	clk_tst [0]	clk_tst [1]	AGP/PCI	VDD_syn	Osc_in	Vref	GND_dac	BCS[3]	A
B	ACS#[0]	ACS#[1]	B[3]	B[6]	R[1]	R[3]	R[5]	G[0]	G[3]	G[6]	usr_io[0]	usr_io[3]	hsync_in	cblank	usr_io[5]	csync	Video_clk	GND_syn	filter_pros	GND_bias	IGreen	IBlue	BCS#[2]	B
C	ADQ [31]	ADQ [30]	B[4]	B[7]	VDD_CORE	GND_CORE	R[6]	G[1]	G[4]	G[7]	usr_io[1]	usr_io[4]	GND_CORE	usr_io[6]	VDD_CORE	hsync	filter_video	Osc_out	IRed	VDD_dac	BDQ [00]	BDQ [01]	BDQ [02]	C
D	ADQ [29]	ADQ [28]	ADQ [27]	ADQ [26]	VDD_PADv	GND_dig	VDD_PADv	GND_dig	VDD_PADv	GND_dig	VDD_PADv	VDD_PADv	VDD_PADv	VDD_PADv	GND_dig	Pros_clk		Rres	GND_dig	VDD_Clamp	GND_digb	BDQ [03]	BDQ [04]	D
E	ADQ [25]	ADQ [24]	ADQ [23]	GND_diga																VDD_PADb	BDQ [05]	BDQ [06]	BDQ [07]	E
F	ADQ [22]	ADQ [21]	ADQ [20]	VDD_PADa																GND_digb	BDQ [08]	BDQ [09]	BDQ [10]	F
G	ADQ [19]	ADQ [18]	ADQ [17]	GND_diga																VDD_PADb	BDQ [11]	BDQ [12]	BDQ [13]	G
H	ADQ [16]	Amem clk	Amem clkin	VDD_PADa																VDD_PADb	BDQ [14]	BDQ [15]	BDQM [3]	H
J	AWE#	ARAS#	GND_CORE	GND_diga																VDD_CORE	GND_CORE	BDQM [2]	BA[11]	J
K	ADQM [0]	ADQM [1]	ACAS#	VDD_CORE																GND_digb	BA[10]	BA[09]	BA[08]	K
L	AA[00]	AA[01]	AA[02]	VDD_PADa																VDD_PADb	VDD_PADb	BA[07]	BA[06]	L
M	AA[03]	AA[04]	AA[05]	GND_diga																GND_digb	BA[05]	BA[04]	BA[03]	M
N	AA[06]	AA[07]	VDD_PADa	VDD_PADa																VDD_PADb	BA[02]	BA[01]	BA[00]	N
P	AA[08]	AA[09]	AA[10]	GND_diga																VDD_CORE	BCAS	BDQM [1]	BDQM [0]	P
R	AA[11]	ADQM [2]	GND_CORE	VDD_CORE																GND_digb	GND_CORE	BRAS#	BWE#	R
T	ADQM [3]	ADQ [15]	ADQ [14]	VDD_PADa																VDD_PADb	Bmem clkin	Bmem clk	BDQ [16]	T
U	ADQ [13]	ADQ [12]	ADQ [11]	VDD_PADa																GND_digb	BDQ [17]	BDQ [18]	BDQ [19]	U
V	ADQ [10]	ADQ [09]	ADQ [08]	GND_diga																VDD_PADb	BDQ [20]	BDQ [21]	BDQ [22]	V
W	ADQ [07]	ADQ [06]	ADQ [05]	VDD_PADa																GND_digb	BDQ [23]	BDQ [24]	BDQ [25]	W
Y	ADQ [04]	ADQ [03]	GND_diga	VDD_PADp	GND_digp	VDD_CORE	VDD_PADp	GND_digp	GND_digp	VDD_PADp	GND_CORE	VDD_CORE	VDD_PADp	GND_digp	VDD_PADp	GND_CORE	VDD_PADp	GND_digp	VDD_PADp	BDQ [26]	BDQ [27]	BDQ [28]	BDQ [29]	Y
AA	ADQ [02]	ADQ [01]	ADQ [00]	GND_CORE	PCI_AD[06]	PCI_C/BE#[0]	PCI_AD[11]	VDD_PADp	PCI_AD[15]	VDD_CORE	GND_digp	GND_CORE	VDD_Clamp	PCI_AD[17]	PCI_AD[21]	VDD_CORE	VDD_PADp	GND_digp	GND_digp	PCI_AD[29]	PCI_CLK	BDQ [30]	BDQ [31]	AA
AB	ACS#[2]	PCI_AD[00]	PCI_AD[02]	PCI_AD[05]	PCI_AD[07]	PCI_AD[09]	PCI_AD[12]	PCI_AD[14]	PCI_PAR	PCI_PERR#	PCI_DEVSEL#	PCI_IRDY#	PCI_C/BE#[2]	PCI_AD[18]	PCI_AD[20]	PCI_AD[23]	PCI_C/BE#[3]	PCI_AD[25]	PCI_AD[27]	PCI_AD[30]	PCI_GNT#	BCS#[1]	BCS#[0]	AB
AC	ACS#[3]	PCI_AD[01]	PCI_AD[03]	PCI_AD[04]	PCI_AD[08]	PCI_AD[10]	PCI_AD[13]	PCI_C/BE#[1]	PCI_SERR#	PCI_STOP#	PCI_TRDY#	PCI_FRAME#	PCI_AD[16]	PCI_AD[19]	PCI_AD[22]	PCI_IDSEL	PCI_AD[24]	PCI_AD[26]	PCI_AD[28]	PCI_AD[31]	PCI_REQ#	PCI_RST#	PCI_INTA#	AC

VS25203
Bottom View

(PINS UP)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
A	BCS#[3]	GND_dac	Vref	Osc_in	VDD_syn	AGP/PCI	clk_tst[1]	clk_tst[0]	PCLK	vsync	vsync_in	usr_io[2]	use_rom#	G[5]	G[2]	R[7]	R[4]	R[2]	R[0]	B[5]	B[2]	B[1]	B[0]
B	BCS#[2]	IBlue	IGreen	GND_bias	filter_pros	GND_syn	Video_clk	csync	usr_io[5]	cblank	hsync_in	usr_io[3]	usr_io[0]	G[6]	G[3]	G[0]	R[5]	R[3]	R[1]	B[6]	B[3]	ACS#[1]	ACS#[0]
C	BDQ[02]	BDQ[01]	BDQ[00]	VDD_dac	IRed	Osc_out	filter_video	hsync	VDD_CORE	usr_io[6]	GND_CORE	usr_io[4]	usr_io[1]	G[7]	G[4]	G[1]	R[6]	GND_CORE	VDD_CORE	B[7]	B[4]	ADQ[30]	ADQ[31]
D	BDQ[04]	BDQ[03]	GND_digb	VDD_Clamp	GND_dig	Rres		Pros_clk	GND_dig	VDD_PADv	VDD_PADv	VDD_PADv	VDD_PADv	GND_dig	VDD_PADv	GND_dig	VDD_PADv	GND_dig	VDD_PADv	ADQ[26]	ADQ[27]	ADQ[28]	ADQ[29]
E	BDQ[07]	BDQ[06]	BDQ[05]	VDD_PADb																GND_diga	ADQ[23]	ADQ[24]	ADQ[25]
F	BDQ[10]	BDQ[09]	BDQ[08]	GND_digb																VDD_PADa	ADQ[20]	ADQ[21]	ADQ[22]
G	BDQ[13]	BDQ[12]	BDQ[11]	VDD_PADb																GND_diga	ADQ[17]	ADQ[18]	ADQ[19]
H	BDQM[3]	BDQ[15]	BDQ[14]	VDD_PADb																VDD_PADa	Amem clkin	Amem clk	ADQ[16]
J	BA[11]	BDQM[2]	GND_CORE	VDD_CORE																GND_diga	GND_CORE	ARAS#	AWE#
K	BA[08]	BA[09]	BA[10]	GND_digb																VDD_CORE	ACAS	ADQM[1]	ADQM[0]
L	BA[06]	BA[07]	VDD_PADb	VDD_PADb																VDD_PADa	AA[02]	AA[01]	AA[00]
M	BA[03]	BA[04]	BA[05]	GND_digb																GND_diga	AA[05]	AA[04]	AA[03]
N	BA[00]	BA[01]	BA[02]	VDD_PADb																VDD_PADa	VDD_PADa	AA[07]	AA[06]
P	BDQM[0]	BDQM[1]	BCAS#	VDD_CORE																GND_diga	AA[10]	AA[09]	AA[08]
R	BWE#	BRAS#	GND_CORE	GND_digb																VDD_CORE	GND_CORE	ADQM[2]	AA[11]
T	BDQ[16]	Bmem clk	Bmem clkin	VDD_PADb																VDD_PADa	ADQ[14]	ADQ[15]	ADQM[3]
U	BDQ[19]	BDQ[18]	BDQ[17]	GND_digb																VDD_PADa	ADQ[11]	ADQ[12]	ADQ[13]
V	BDQ[22]	BDQ[21]	BDQ[20]	VDD_PADb																GND_diga	ADQ[08]	ADQ[09]	ADQ[10]
W	BDQ[25]	BDQ[24]	BDQ[23]	GND_digb																VDD_PADa	ADQ[05]	ADQ[06]	ADQ[07]
Y	BDQ[29]	BDQ[28]	BDQ[27]	BDQ[26]	VDD_PADp	GND_digp	VDD_PADp	GND_CORE	VDD_PADp	GND_digp	VDD_PADp	VDD_CORE	GND_CORE	VDD_PADp	GND_digp	GND_digp	VDD_PADp	VDD_CORE	GND_digp	VDD_PADp	GND_diga	ADQ[03]	ADQ[04]
AA	BDQ[31]	BDQ[30]	PCI_CLK	PCI_AD[29]	GND_digp	GND_digp	VDD_PADp	VDD_CORE	PCI_AD[21]	PCI_AD[17]	VDD_Clamp	GND_CORE	GND_digp	VDD_CORE	PCI_AD[15]	VDD_PADp	PCI_AD[11]	PCI_C/BE#[0]	PCI_AD[06]	GND_CORE	ADQ[00]	ADQ[01]	ADQ[02]
AB	BCS#[0]	BCS#[1]	PCI_GNT#	PCI_AD[30]	PCI_AD[27]	PCI_AD[25]	PCI_C/BE#[3]	PCI_AD[23]	PCI_AD[20]	PCI_AD[18]	PCI_C/BE#[2]	PCI_IRDY#	PCI_DEVSEL#	PCI_PERR#	PCI_PAR	PCI_AD[14]	PCI_AD[12]	PCI_AD[09]	PCI_AD[07]	PCI_AD[05]	PCI_AD[02]	PCI_AD[00]	ACS#[2]
AC	PCI_INTA#	PCI_RST#	PCI_REQ#	PCI_AD[31]	PCI_AD[28]	PCI_AD[26]	PCI_AD[24]	PCI_IDSEL	PCI_AD[22]	PCI_AD[19]	PCI_AD[16]	PCI_FRAME#	PCI_TRDY#	PCI_STOP#	PCI_SERR#	PCI_C/BE#[1]	PCI_AD[13]	PCI_AD[10]	PCI_AD[08]	PCI_AD[04]	PCI_AD[03]	PCI_AD[01]	ACS#[3]

VS25203
Top View

(PINS DOWN)

15.2 Signal descriptions

The signals for the VS25203 device are described in this section. The following tables list each signal, its pin location, the operating mode (input, output, analog, power) and provide some descriptions. The signals are grouped according to their functional purpose.

15.2.1 External DAC Signals

External Video DAC Pin Signals

Signal name	Pin	Mode	Description
B[0]	A23	I/O	8-bit data bus for blue color / BIOS data. this bus is used as the data bus (input) when performing ROM accesses. It is also possible to utilize it as an extra digital input resource if the digital RGB outputs are not used.
B[1]	A22	I/O	
B[2]	A21	I/O	
B[3]	B21	I/O	
B[4]	C21	I/O	
B[5]	A20	I/O	
B[6]	B20	I/O	
B[7]	C20	I/O	
cblank	B10	O	Composite-blanking signal, created from the horizontal and vertical blank signals.
csync	B8	O	Composite sync signal out, created from the horizontal and vertical sync signals.
G[0]	B16	O	8 bit data bus for green color / BIOS low order bits address. It is used as the low order address bits when performing ROM accesses. It is also possible to utilize the bus as an extra digital output resource if the digital RGB outputs are not used.
G[1]	C16	O	
G[2]	A15	O	
G[3]	B15	O	
G[4]	C15	O	
G[5]	A14	O	
G[6]	B14	O	
G[7]	C14	O	
hsync	C8	O	Horizontal sync signal.
PCLK	A9	O	Delayed clock signal for external DAC.
R[0]	A19	O	8 bit data bus for red color / BIOS high order bits address. This bus is used as the high order address bits when performing ROM accesses. It is also possible to utilize it as an extra digital output resource if the digital RGB outputs are not used.
R[1]	B19	O	
R[2]	A18	O	
R[3]	B18	O	
R[4]	A17	O	
R[5]	B17	O	
R[6]	C17	O	
R[7]	A16	O	
vsync	A10	O	Vertical sync signal.

15.2.2 PLL Signals

PLL Signals			
Signal name	Pin	Mode	Description
clk_tst[0]	A8	I	clk_tst[0] configures the direction of the Video_clk pin.
clk_tst[1]	A7	I	clk_tst[1] configures the direction of the Pros_clk pin. clk_tst[1:0]=00 internal PLL generated clock pins are not active. clk_tst[1:0]=01 clock pins used as clock inputs. clk_tst[1:0]=10 internal PLL generated clock pins used as outputs. clk_tst[1:0]=11 reserved.
filter_pros	B5	O	Core PLL external RC loop filter, typical component values C=100nF, R=400 ohms.
filter_video	C7	O	Video PLL external RC loop filter, typical component values C=100nF, R=400 ohms.
Pros_clk	D8	I/O	Processor clock. Normally not connected, can be used either as a clock input or as a clock output depending on the clk_tst signals.
Video_clk	B7	I/O	Video clock. Normally not connected, can be used either as a clock input or as a clock output depending on the clk_tst signals.
Osc_in	A4	analog	External chrystal connection for the internal clock generator.
Osc_out	C6	analog	Typical crystal frequency is 14.3181818 MHz.

15.2.3 Internal Video DAC Signals

Internal Video DAC Pin Signals			
Signal name	Pin	Mode	Description
hsync_in	B11	I	Horizontal synchronization input. VS252 will detect the transition from non-active to active state on this line, and synchronize its internal operation to it.
vsync_in	A11	I	Vertical synchronization input. VS252 will detect the transition from non-active to active state on this line, and synchronize its internal operation to it.
IBlue	B2	O	Blue, green and red analog (current mode) outputs; RS-343-A compatible.
IGreen	B3	O	
IRed	C5	O	
Rres	D6	analog	Resistor reference of 1100 ohms should be connected between this pin and ground.
Vref	A3	analog	Voltage reference for the video DAC. This is the output of VS252's internal voltage reference (1.23V). The output is relatively high impedance (10kohms); it is possible to override it with an external voltage reference. It is recommended that a bypass capacitor is attached to this pin.

15.2.4 Miscellaneous Signals

Miscellaneous Signals			
Signal name	Pin	Mode	Description
use_rom#	A13	O	Use ROM (active low)> VS252 can use a ROM which is connected to the digital RGB lines for boot configuration and as a BIOS ROM. The use_rom# line is used to differentiate between the normal digital video usage and the ROM access usage. It should be connected to the ROM chip select and output enable lines; both signals should be active and the ROM used must set the data pins to high impedance state when it is not selected.
AGP / PCI	A6	I	Pad operation mode selection. "0" = AGP and "1" = PCI. See Supply Signals VDD_Clamp.
usr_io[0]	B13	I/O	User configurable general purpose I/O pins. These pins can be read and written, and their direction changed using internal registers.
usr_io[1]	C13	I/O	
usr_io[2]	A12	I/O	
usr_io[3]	B12	I/O	
usr_io[4]	C12	I/O	
usr_io[5]	B9	I/O	
usr_io[6]	C10	I/O	

15.2.5 A-Memory Signals

A-memory Signals			
Signal name	**Pin**	**Mode**	**Description**
AA[00]	L23	O	A Memory Address. It is a 12-bit address bus. When used with SDRAM or SGRAM the memory address bus is also used to transfer configuration data and to perform bank select operations, so it is essential that the relevant address pins are connected to the corresponding address pins on the memories (it is not ok to swap the address pins).
AA[01]	L22	O	
AA[02]	L21	O	
AA[03]	M23	O	
AA[04]	M22	O	
AA[05]	M21	O	
AA[06]	N23	O	
AA[07]	N22	O	
AA[08]	P23	O	
AA[09]	P22	O	
AA[10]	P21	O	
AA[11]	R23	O	
ACAS#	K21	O	A Memory Column Address Select. Drives the CAS input of external memory. Used on SDRAM/SGRAM memory configuration. On EDO or FPM DRAMs, the DRAM's CAS lines should be connected to ADQM# lines.
ACS#[0]	B23	O	Chip select signals for memory banks. These lines are needed on large memory configurations. The chip selects are decoded so that the first memory device should be connected to the ACS#[0], the second to the ACS#[1] etc.
ACS#[1]	B22	O	
ACS#[2]	AB23	O	
ACS#[3]	AC23	O	
ADQ[00]	AA21	I/O	32-bit A-memory Data Bus. The normal configuration for the A-Data Bus is 32 bits wide (+ 32 bits for the B-Data Bus), but it is possible to create a system with 16 (+ 16) wide interface when using SDRAM as the basic element of the memory subsystem.
ADQ[01]	AA22	I/O	
ADQ[02]	AA23	I/O	
ADQ[03]	Y22	I/O	
ADQ[04]	Y23	I/O	
ADQ[05]	W21	I/O	
ADQ[06]	W22	I/O	
ADQ[07]	W23	I/O	
ADQ[08]	V21	I/O	
ADQ[09]	V22	I/O	
ADQ[10]	V23	I/O	
ADQ[11]	U21	I/O	
ADQ[12]	U22	I/O	
ADQ[13]	U23	I/O	
ADQ[14]	T21	I/O	
ADQ[15]	T22	I/O	
ADQ[16]	H23	I/O	
ADQ[17]	G21	I/O	
ADQ[18]	G22	I/O	
ADQ[19]	G23	I/O	
ADQ[20]	F21	I/O	
ADQ[21]	F22	I/O	
ADQ[22]	F23	I/O	
ADQ[23]	E21	I/O	
ADQ[24]	E22	I/O	
ADQ[25]	E23	I/O	
ADQ[26]	D20	I/O	
ADQ[27]	D21	I/O	
ADQ[28]	D22	I/O	

A-memory Signals			
Signal name	**Pin**	**Mode**	**Description**
ADQ[29]	D23	I/O	
ADQ[30]	C22	I/O	
ADQ[31]	C23	I/O	
ADQM#[0]	K23	O	A-Memory Data Byte Enables. These are connected to the DQM lines of the SDRAM or SGRAM, and to the CAS lines of EDO or FPM DRAMs.
ADQM#[1]	K22	O	
ADQM#[2]	R22	O	
ADQM#[3]	T23	O	
Amemclk	H22	O	A Memory Clock. It is the clock output for memory synchronization used by synchronous memories. For non-synchronous memory, this signal is not used.
Amemclkin	H21	I	A Memory Clock Input. Used for controlling the latch in of the external data. This pin must be connected to the Amemclk pin. The connection must be made even in configurations with non-synchronous memories.
ARAS#	J22	O	A-Memory Row Address Select. Drives the RAS input of external (either synchronous or non-synchronous) memory.
AWE#	J23	O	Write Enable. Drives the WE# input of external (synchronous or non-synchronous) memory.

15.2.6 B-Memory Signals

B-memory Signals			
Signal name	**Pin**	**Mode**	**Description**
BA[00]	N1	O	B-Memory Address. It is a 12-bit address bus. When used with SDRAM or SGRAM the memory address bus is also used to transfer configuration data and to perform bank select operations, so it is essential that the relevant address pins are connected to the corresponding address pins on the memories (it is not ok to swap the address pins).
BA[01]	N2	O	
BA[02]	N3	O	
BA[03]	M1	O	
BA[04]	M2	O	
BA[05]	M3	O	
BA[06]	L1	O	
BA[07]	L2	O	
BA[08]	K1	O	
BA[09]	K2	O	
BA[10]	K3	O	
BA[11]	J1	O	
BCAS#	P3	O	B-Memory Column Address Select. Drives the CAS input of external memory. Used on SDRAM/SGRAM memory configuration. On EDO or FPM DRAMs, the DRAM's CAS lines should be connected to ADQM# lines.
BCS#[0]	AB1	O	Chip select signals for memory banks. These lines are needed on large memory configurations. The chip selects are decoded so that the first memory device should be connected to the BCS#[0], the second to the BCS#[1] etc.
BCS#[1]	AB2	O	
BCS#[2]	B1	O	
BCS#[3]	A1	O	
BDQ[00]	C3	I/O	32-bit B-Memory Data Bus. The normal configuration for the B-Data Bus is 32 bits wide (+ 32 bits for the A-Data Bus), but it is possible to create a system with 16 (+ 16) wide interface when using SDRAM as the basic element of the memory subsystem.
BDQ[01]	C2	I/O	
BDQ[02]	C1	I/O	
BDQ[03]	D2	I/O	
BDQ[04]	D1	I/O	
BDQ[05]	E3	I/O	
BDQ[06]	E2	I/O	
BDQ[07]	E1	I/O	
BDQ[08]	F3	I/O	

B-memory Signals			
Signal name	**Pin**	**Mode**	**Description**
BDQ[09]	F2	I/O	
BDQ[10]	F1	I/O	
BDQ[11]	G3	I/O	
BDQ[12]	G2	I/O	
BDQ[13]	G1	I/O	
BDQ[14]	H3	I/O	
BDQ[15]	H2	I/O	
BDQ[16]	T1	I/O	
BDQ[17]	U3	I/O	
BDQ[18]	U2	I/O	
BDQ[19]	U1	I/O	
BDQ[20]	V3	I/O	
BDQ[21]	V2	I/O	
BDQ[22]	V1	I/O	
BDQ[23]	W3	I/O	
BDQ[24]	W2	I/O	
BDQ[25]	W1	I/O	
BDQ[26]	Y4	I/O	
BDQ[27]	Y3	I/O	
BDQ[28]	Y2	I/O	
BDQ[29]	Y1	I/O	
BDQ[30]	AA2	I/O	
BDQ[31]	AA1	I/O	
BDQM#[0]	P1	O	B-Memory Data Byte Enables. These are connected to the DQM lines of the SDRAM or SGRAM, and to the CAS lines of EDO or FPM DRAMs.
BDQM#[1]	P2	O	
BDQM#[2]	J2	O	
BDQM#[3]	H1	O	
Bmemclk	T2	O	B-Memory Clock. It is the clock output for memory synchronization used by synchronous memories. For non-synchronous memory, this signal is not used.
Bmemclkin	T3	I	B-Memory Clock Input. Used for controlling the latching-in of the external data. This pin must be connected to the Bmemclk pin. The connection must be made even in configurations with non-synchronous memories.
BRAS#	R2	O	B-Memory Row Address Select. Drives the RAS input of external (either synchronous or non-synchronous) memory.
BWE#	R1	O	B-Memory Write Enable. Drives the WE# input of external (synchronous or non-synchronous) memory.

15.2.7 PCI-Bus Signals

PCI Bus Signals			
Signal name	**Pin**	**Mode**	**Description**
PCI_AD[00]	AB22	I/O	32-bit multiplexed Address and Data Bus.
PCI_AD[01]	AC22	I/O	
PCI_AD[02]	AB21	I/O	
PCI_AD[03]	AC21	I/O	
PCI_AD[04]	AC20	I/O	
PCI_AD[05]	AB20	I/O	
PCI_AD[06]	AA19	I/O	
PCI_AD[07]	AB19	I/O	
PCI_AD[08]	AC19	I/O	
PCI_AD[09]	AB18	I/O	
PCI_AD[10]	AC18	I/O	
PCI_AD[11]	AA17	I/O	
PCI_AD[12]	AB17	I/O	
PCI_AD[13]	AC17	I/O	
PCI_AD[14]	AB16	I/O	
PCI_AD[15]	AA15	I/O	
PCI_AD[16]	AC11	I/O	
PCI_AD[17]	AA10	I/O	
PCI_AD[18]	AB10	I/O	
PCI_AD[19]	AC10	I/O	
PCI_AD[20]	AB9	I/O	
PCI_AD[21]	AA9	I/O	
PCI_AD[22]	AC9	I/O	
PCI_AD[23]	AB8	I/O	
PCI_AD[24]	AC7	I/O	
PCI_AD[25]	AB6	I/O	
PCI_AD[26]	AC6	I/O	
PCI_AD[27]	AB5	I/O	
PCI_AD[28]	AC5	I/O	
PCI_AD[29]	AA4	I/O	
PCI_AD[30]	AB4	I/O	
PCI_AD[31]	AC4	I/O	
PCI_C/BE#[0]	AA18	I/O	Multiplexed Bus Command and Byte Enables.
PCI_C/BE#[1]	AC16	I/O	Used to transmit the command on the first cycle of the transaction and the byte
PCI_C/BE#[2]	AB11	I/O	enables on the following cycles.
PCI_C/BE#[3]	AB7	I/O	
PCI_CLK	AA3	I	PCI Clock Signal. Supports PCI clock frequencies in the range 0-33 MHz.

PCI Bus Signals			
Signal name	**Pin**	**Mode**	**Description**
PCI_DEVSEL#	AB13	I/O	Device Select. Used by transaction target to indicate that it has decoded a recognized address of the transaction.
PCI_FRAME#	AC12	I/O	Cycle Frame.Driven by the transaction initiator to indicate the beginning and the duration of an access.
PCI_GNT#	AB3	I	Grant Bus Ownership. Indicates to the agent that the arbiter has granted access to the bus when VS252 operates as a bus master.
PCI_IDSEL	AC8	I	Initialization Device select. Used as a chip select during the configuration transactions. Note that configuration transactions do not use the normal PCI address decoding.
PCI_INTA#	AC1	O	Interrupt A. Indicates an interrupt request. The wiring of this interrupt line is motherboard and operating system dependent. The interrupt is reset by resetting the corresponding status register bit.
PCI_IRDY#	AB12	I/O	Initiator Ready. Indicates the initiating agent's ability to complete the data phase of the transaction, and is ready to transfer data on the current clock cycle. Pin direction depends on whether VS252 is participating in the transfer as a target or as an initiator.
PCI_PAR	AB15	I/O	Parity. Indicates even parity across PCI_AD[31:0] and PCI_C/BE#[3:0].
PCI_PERR#	AB14	I/O	Parity Error. Indicates a data parity error in AD, C/BE#, and PAR signal lines during the data phase.
PCI_REQ#	AC3	O	Request bus ownership. Used when operating as the initiator for requesting bus ownership; indicates to the arbiter that this agent desires use of the bus.
PCI_RST#	AC2	I	PCI Reset. Forces the PCI sequencer of VS252 to a known state.
PCI_SERR#	AC15	I/O	System Error. Reports address or data parity errors or any other catastrophic error.
PCI_STOP#	AC14	I/O	StopTransaction; used by the target when it needs to stop a transaction. Typical usage does not indicate any kind of error condition.
PCI_TRDY#	AC13	I/O	Target Ready. Indicates the target agent's ability to complete the current data phase, and is ready to transfer data on the current clock cycle. Pin direction depends on whether VS252 is participating in the transfer as a target or as an initiator.

15.2.8 Supply Signals

Supply Signals			
Signal name	**Pin**	**Mode**	**Description**
gnd_core	J21	power	Core ground pads.
gnd_core	R21	power	
gnd_core	AA20	power	
gnd_core	Y13	power	
gnd_core	AA12	power	
gnd_core	Y8	power	
gnd_core	R3	power	
gnd_core	J3	power	
gnd_core	C11	power	
gnd_core	C18	power	
GND_diga	E20	power	Ground for A memory pads.
GND_diga	G20	power	
GND_diga	J20	power	
GND_diga	M20	power	
GND_diga	P20	power	
GND_diga	V20	power	
GND_diga	Y21	power	
GND_digp	Y19	power	Ground for PCI pads.
GND_digp	Y16	power	
GND_digp	Y15	power	
GND_digp	AA13	power	
GND_digp	Y10	power	
GND_digp	AA6	power	
GND_digp	Y6	power	
GND_digp	AA5	power	
GND_digb	W4	power	Ground for B memory pads.
GND_digb	U4	power	
GND_digb	R4	power	
GND_digb	M4	power	
GND_digb	K4	power	
GND_digb	F4	power	
GND_digb	D3	power	
GND_dac	A2	power	Ground for internal DAC.
GND_bias	B4	power	
VDD_dac	C4	power	Analog Vdd.
GND_DIG	D5	power	Ground for core.
GND_DIG	D9	power	
GND_DIG	D14	power	
GND_DIG	D16	power	
GND_DIG	D18	power	
	D7		Not used.
vdd_core[0]	K20	power	Core $V_{dd.}$
vdd_core[1]	R20	power	
vdd_core[2]	Y18	power	
vdd_core[3]	AA14	power	
vdd_core[4]	Y12	power	
vdd_core[5]	AA8	power	
vdd_core[6]	P4	power	

Supply Signals			
Signal name	**Pin**	**Mode**	**Description**
vdd_core[7]	J4	power	Core $V_{dd.}$
vdd_core[8]	C9	power	
vdd_core[9]	C19	power	
VDD_syn	A5	power	V_{dd} for PLL.
GND_syn	B6	power	Groung for PLL.
VDD_Clamp	AA11	power	Clamp diode terminal. Note 3.3 volt for AGP and 5 or 3.3 volt for PCI. See also pin A6 AGP / PCI in Miscellaneous Signals.
VDD_Clamp	D4	power	
VDD_PADa	F20	power	V_{dd} for A memory pads.
VDD_PADa	H20	power	
VDD_PADa	L20	power	
VDD_PADa	N21	power	
VDD_PADa	N20	power	
VDD_PADa	T20	power	
VDD_PADa	U20	power	
VDD_PADa	W20	power	
VDD_PADb	V4	power	V_{dd} for B memory pads.
VDD_PADb	T4	power	
VDD_PADb	N4	power	
VDD_PADb	L3	power	
VDD_PADb	L4	power	
VDD_PADb	H4	power	
VDD_PADb	G4	power	
VDD_PADb	E4	power	
VDD_PADp	Y20	power	V_{dd} for PCI pads.
VDD_PADp	Y17	power	
VDD_PADp	AA16	power	
VDD_PADp	Y14	power	
VDD_PADp	Y11	power	
VDD_PADp	Y9	power	
VDD_PADp	AA7	power	
VDD_PADp	Y7	power	
VDD_PADp	Y5	power	
VDD_PADv	D10	power	V_{dd} for video pads.
VDD_PADv	D11	power	
VDD_PADv	D12	power	
VDD_PADv	D13	power	
VDD_PADv	D15	power	
VDD_PADv	D17	power	
VDD_PADv	D19	power	

16. Electrical Specifications

16.1 Electrical Characteristics and Operating Conditions

16.1.1 Absolute Maximum Conditions

Beyond these limits damage may occur to the device.

Symbol	Parameter	Condition	Min	Typ	Max	Unit
V	Supply voltage		-0.25		4.0	V
T_S	Storage temperature		-40		125	°C

16.1.2 DC Operating Conditions

Valid for 25 °C ambient temperature and 3.3 V supply unless otherwise stated.

Symbol	Parameter	Condition	Min	Typ	Max	Unit
V_{dd1}	Supply voltage		3.0	3.3	3.6	V
V_{dd2}	Supply voltage		3.0	3.3	3.6	V
Avd	Analog Supply Voltage		3.15	3.3	3.45	V
CLK	Crystal Frequency			14.318		MHz

16.1.3 General Specifications

Symbol	Parameter	Condition	Min	Typ	Max	Unit
V_{il}	TTL input LO	$V_{dd} = 3.3$ V	-0.5		0.3 * V_{dd}	V
V_{ih}	TTL input HI	$V_{dd} = 3.3$ V	0.5 * V_{dd}		V_{dd} + 0.5	V
I_{il}	Input leakage	$0 < V_{in} < V_{dd}$	-10		10	uA
V_{ol}	Low Level Output	pad: I_{ol} = 1500 uA, V_{dd} = 3.3 V			0.1 * V_{dd}	V
V_{oh}	High Level Output	pad: I_{oh} = -500 uA, V_{dd} = 3.3 V	0.9 * V_{dd}			V
I_{oz}	High Z leakage	$0 < V_{in} < V_{dd}$	-10		10	uA

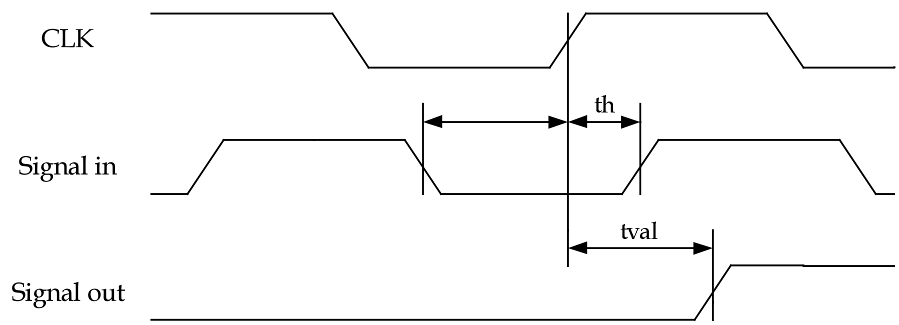
16.1.4 Electrical Specifications

Symbol	Parameter	Condition	Min	Typ	Max	Unit
I_{VDD}	Digital Supply Current (using CMOS -level clock)	Power up RESET = Logic **0**		150	TBA	mA
I_{AVD}	Analog Supply Current	Power up RESET = Logic **0**		40	TBA	mA
I_{VDDPD}	Digital Supply Current	Power down RESET = Logic **1**		1		uA
I_{AVDPD}	Analog Supply Current	Power down RESET = Logic **1**		1		uA

16.2 Timing Parameters

16.2.1 PCI Interface

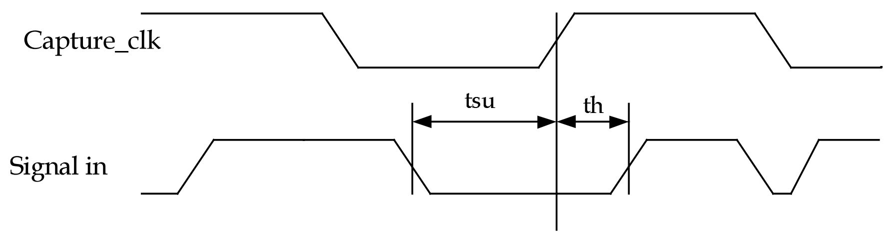
The PCI interface is designed to be compatible with PCI Local Bus Specification rev. 2.1.



Symbol	Parameter	Min	Max	Unit
tsu	Input set up time to CLK			
	bused signals	7		ns
	point to point	10		ns
th	Input hold time from CLK	0		ns
tval	CLK to output valid delay			
	bused signals	2	11	ns
	point to point	2	12	ns

16.2.2 Video Capture

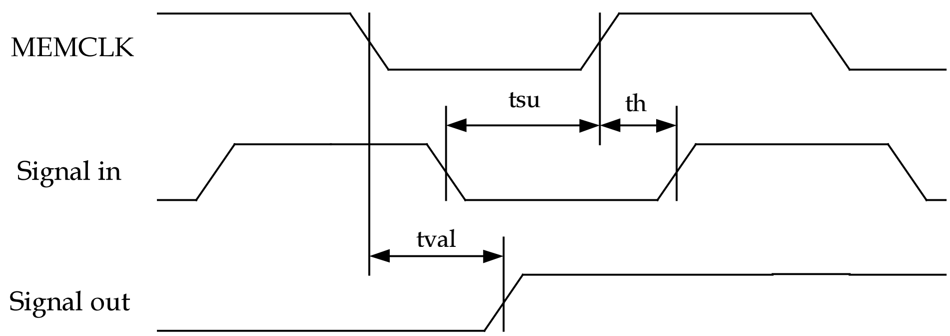
Video capture unit is designed to be working at least up to 35 MHz capture clock frequencies.



Symbol	Parameter	Min	Max	Unit
tsu	Input set up time to CLK	0	8	ns
th	Input hold time from CLK	0	4	ns

16.2.3 Memory Interface

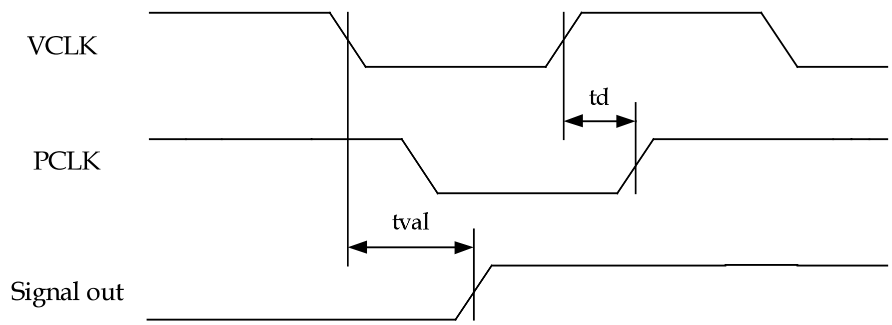
All timings are relative to the MEMCLK created by VS25203. Memory interface is designated to be compatible with SGRAM and SDRAM devices with clock frequencies up to 100MHz.



Symbol	Parameter	Min	Max	Unit
tsu	Input set up time to CLK	1		ns
th	Input hold time from CLK	1		ns
tval	CLK fall to output valid delay	0	1	ns

16.2.4 Video Interface

All timings are relative to the MEMCLK created by VS25203.



Symbol	Parameter	Min	Max	Unit
td	PCLK delay from VCLK	0	3	ns
tval	CLK fall to output valid delay	0	1	ns

17. Further Readings

PCI Local Bus Specification, rev. 2.1.
PCI Multimedia Design Guide rev. 1.0.
PCI System Design Guide, rev. 1.0.

PCI Special Interest Group, PO Box 14070, Portland, OR 97214,
tel.no. 1 800 433 5177 (503 234 6762 int.) 1 503 234 6762 fax.

18. Index