

SM3110

Technical Reference

Preliminary

Copyright Notice

Copyright 1999 by Silicon Magic Corporation. All rights reserved.

This technical reference manual is the intellectual property of Silicon Magic Corporation. You may not reproduce, transmit, transcribe, store in a retrieval system or translate into any language or any computer language in any form or by any means - electronic, mechanical, magnetic, optical, chemical, manual or otherwise - any part of this publication without the express prior written permission of Silicon Magic Corporation.

Revision Number	Contents	Contributer / Writer	Date
1.0.0.0	First Release	Staff / Inc	8/12/1999

Table of Contents

1	SM3110 Features Overview	
1.1	Features Overview	1-2
2	Architecture Overview	
2.1	System Configuration	2-2
2.2	Internal Configuration	2-3
2.2.1	Host Interface	2-4
2.2.2	Embedded Memory	2-5
2.2.3	VGA Module	2-8
2.2.4	2D Rendering Engine	2-9
2.2.5	3D Rendering Engine	2-10
2.2.6	Display Pipeline	2-11
2.2.7	Video Input	2-13
2.2.8	Motion Compensation	2-14
2.2.9	ROM Port	2-15
2.3	Additional Features	2-16
2.3.1	Power Management	2-16
2.3.2	Board-level Test Functions	2-16
2.3.3	Clocks	2-16
2.3.4	Inter-IC (I ² C) Interface	2-16

3	Data Sheet	
3.1	Pin Descriptions	3-2
3.1.1	AGP/PCI Interface	3-2
3.1.2	Panel Interface	3-3
3.1.3	ROM Interface	3-4
3.1.4	ZV Port	3-5
3.1.5	CRT Interface	3-5
3.1.6	Reference Clock Input	3-5
3.1.7	Programmable I/O Pins	3-6
3.1.8	Testing Pins	3-6
3.1.9	Reserved Pins	3-6
3.1.10	Digital Power and Ground Pins	3-7
3.1.11	Analog Power and Ground Pins	3-7
3.1.12	Pad Descriptions	3-8
3.1.13	Alternate Pad Function	3-9
3.1.14	Pin Assignment (by pin number)	3-10
3.1.15	Pin Assignment (by signal name)	3-13
3.2	Electrical Specifications	3-18
3.2.1	Maximum DC Specifications	3-18
3.2.2	Normal Operating DC Specifications	3-18
3.2.3	DC Characteristics	3-19
3.2.4	AC Timing	3-21
3.2.4.1	AGP Timing	3-21
3.2.4.2	PCI Timing	3-24
3.2.4.3	PCI Clock Timing	3-26
3.2.4.4	Panel Interface	3-27
3.2.4.5	BIOS ROM Interface	3-28
3.2.4.6	ZV Port	3-32
3.2.4.7	Reference Clock	3-33
3.2.4.8	Reset	3-34
3.3	Mechanical Specifications	3-35
3.4	Ordering Information	3-38

4	Register Summary	
4.1	Control Address Space Overview	4-2
4.2	Register Listing for Control Address Space	4-4
4.2.1	2D Control (Display 0, Display 1) (1FF0000H)	4-5
4.2.1.1	Status/Control (+000H)	4-6
4.2.1.2	Reserved (+100H)	4-14
4.2.1.3	Host Bus Master (+200H)	4-14
4.2.1.4	VGA Registers (+300H)	4-16
4.2.1.5	Peripheral Ports (+400H)	4-16
4.2.1.6	LCD Panel (+500H)	4-22
4.2.1.7	Reserved (+600H)	4-30
4.2.1.8	Reserved (+700H)	4-30
4.2.1.9	2D Rendering Engine (+800H)	4-31
4.2.1.10	Memory Controller (+900H)	4-42
4.2.1.11	Surface Descriptors (+A00H)	4-43
4.2.1.12	Channel Descriptors (+B00H)	4-45
4.2.1.13	Buffer Descriptors (+C00H)	4-48
4.2.1.14	Display Control (+D00H)	4-52
4.2.1.15	Display Viewports and Timing (+E00H)	4-68
4.2.1.16	PCI/AGP Configuration Registers (+F00H)	4-76
4.2.2	3D Control (1FF3000H)	4-84
4.2.2.1	Command Decode Unit (+000H)	4-86
4.2.2.2	Triangle Setup Unit (+040H)	4-91
4.2.2.3	Edge-Walking Unit (+080H)	4-91
4.2.2.4	Texture Mapping Unit (+0C0H)	4-92
4.2.2.5	Z Buffer Unit (+100H)	4-96
4.2.2.6	Per Fragment Unit (+140H)	4-99
4.2.2.7	Memory Interface Unit (+180H)	4-104
4.2.2.8	DMA (+1C0H)	4-108
4.2.2.9	Motion Compensation (+300H)	4-113
4.2.2.10	Stipple Mask Array (+400H)	4-114
4.2.2.11	Texture Map LOD Table Array (+580H)	4-115
4.2.2.12	Temporary Array for Triangle Setup Unit (+5C0H)	4-115
4.2.3	3D Command/Parameter FIFO (1FF4000H)	4-117
4.2.4	2D Command/Parameter FIFO (1FF8000H)	4-117
4.2.5	2D Image Data FIFO (1FFC000H)	4-117

4.3	Standard VGA Registers	4-118
4.3.1	Register Locations	4-118
4.3.2	General Registers	4-120
4.3.3	CRT Controller (CRTC)	4-122
4.3.3.1	VGA CRTC Resolution Limitations	4-122
4.3.3.2	CRTC Index/Data Registers	4-123
4.3.4	Sequencer	4-128
4.3.5	Graphics Controller	4-131
4.3.6	Attribute Controller	4-137
4.3.7	Enhanced VGA Registers	4-140
4.3.7.1	Enhanced Register Access Enable	4-140
4.3.7.2	Enhanced Control Access	4-141
4.3.7.3	Bank Offset	4-142
4.3.7.4	User-Defined Scratch Registers	4-143
4.3.7.5	CRTC Test	4-143

5	BIOS Specifications	
5.1	Overview	5-2
5.1.1	SM3110 Specific Extended BIOS Functions	5-5
5.1.2	Standard VGA BIOS Functions	5-6
5.1.3	VESA BIOS Extended Functions	5-8
5.1.4	SCRATCHPAD Register Usage	5-9
5.2	VGA BIOS Functions	5-12
5.2.1	SM3110 Extended BIOS Functions (5Fh)	5-13
5.2.1.1	Subfunction: 00h - Query BIOS Version	5-13
5.2.1.2	Subfunction: 01h - Query VBE Mode Support	5-13
5.2.1.3	Subfunction: 02h - Set VBE Mode for Second View	5-14
5.2.1.4	Subfunction: 03h - Enable Second View Mode	5-14
5.2.1.5	Subfunction: 04h - Disable Second View Mode	5-14
5.2.2	Standard VGA BIOS Functions	5-15
5.2.2.1	Function: 00h - Set Video Mode	5-15
5.2.2.2	Function: 01h - Set Cursor Type	5-16
5.2.2.3	Function: 02h - Set Cursor Position	5-17
5.2.2.4	Function: 03h - Get Cursor Position	5-17
5.2.2.5	Function: 04h - Light Pen -- (Not Supported)	5-18
5.2.2.6	Function: 05h - Select Active Display Page	5-18
5.2.2.7	Function: 06h - Window Scroll Up	5-19
5.2.2.8	Function: 07h - Window Scroll Down	5-19
5.2.2.9	Function: 08h - Read Character/Attribute at Cursor	5-20
5.2.2.10	Function: 09h - Write Character/Attribute at Cursor	5-21
5.2.2.11	Function: 0Ah - Write Character at Cursor Position	5-22
5.2.2.12	Function: 0Bh - Background/Border, Palette	5-23
5.2.2.13	Function: 0Ch - Write Dot (Pixel)	5-24
5.2.2.14	Function: 0Dh - Read Dot (Pixel)	5-24
5.2.2.15	Function: 0Eh - Write Character to RAM in TTY Mode	5-25
5.2.2.16	Function: 0Fh - Get Video State	5-25
5.2.2.17	Function: 10h - Palette Register (Subfunctions)	5-26
5.2.2.18	Function: 11h - Character Generation (Subfunctions)	5-35
5.2.2.19	Function: 12h - Alternate Select (Subfunctions)	5-52
5.2.2.20	Function: 13h - Write Teletype String	5-57
5.2.2.21	Function: 1Ah - Get/Set Display Combination Code	5-58
5.2.2.22	Function: 1Bh - Collection of Video Information	5-60
5.2.2.23	Function: 1Ch - Video State Information	5-64

5.3	VESA VBE BIOS Functions	5-66
5.3.1	Overview	5-66
5.3.2	DDC Implementation	5-66
5.3.2.1	VBE Mode Numbers	5-67
5.3.2.2	VBE Functions Call	5-68
5.3.2.3	VBE Return Status	5-68
5.3.3	VESA BIOS Extended Functions (4Fh)	5-69
5.3.3.1	Subfunction: 00h - Return VBE Controller Information	5-69
5.3.3.2	Subfunction: 01h - Return VBE Mode Information ...	5-73
5.3.3.3	Subfunction: 02h - Set VBE Mode	5-82
5.3.3.4	Subfunction: 03h - Return current VBE Mode	5-83
5.3.3.5	Subfunction: 04h - Save/Restore State	5-84
5.3.3.6	Subfunction: 05h - Display Window Control	5-85
5.3.3.7	Subfunction: 06h - Set/Get Logical Scanline Length ..	5-86
5.3.3.8	Subfunction: 07h - Set/Get Display Start	5-87
5.3.3.9	Subfunction: 08h - Set/Get DAC Palette Format	5-88
5.3.3.10	Subfunction: 09h - Set/Get Palette Data	5-89
5.3.3.11	Subfunction: 0Ah - VBE Protected Mode Interface ...	5-90
5.3.4	VBE Display Power Management Subfunctions(10h) ...	5-93
5.3.4.1	Subfunction: 00h - Report VBE/PM Capabilities	5-93
5.3.4.2	Subfunction: 01h - Set Display Power State	5-94
5.3.4.3	Subfunction: 02h - Get Display Power State	5-95
5.3.5	VBE Display Identification (DDC) Subfunctions(15h) ...	5-96
5.3.5.1	Subfunction: 00h - Report VBE/DDC Capabilities	5-96
5.3.5.2	Subfunction: 01h - Read EDID	5-97
5.3.6	VBE / Flat Panel Subfunctions (11h)	5-98
5.3.6.1	Flat Panel Interface Functions	5-98
5.3.6.2	VBE/FP Completion Codes	5-99
5.3.6.3	Subfunction 00h - Panel Extensions Support	5-100
5.3.6.4	Subfunction 01h - Return Flat Panel Info	5-102
5.3.6.5	Subfunction 03h - Panel Shading Options	5-104
5.3.6.6	Subfunction 06h - Vertical & Horizontal Position ...	5-105
5.3.6.7	Subfunction 07h - Vertical & Horizontal Expansion ..	5-107
5.4	BIOS Data Area Assignments	5-109

6	Programmer's Reference	
6.1	Overview	6-2
6.2	Address Mapping	6-2
6.2.1	Local Memory Address Space	6-3
6.2.2	Memory-Mapped Resources Address Space	6-5
6.2.3	Control Address Space Memory Mapping	6-6
6.2.3.1	2D Control Registers (Display 0/Display 1)	6-7
6.2.3.2	3D Control Registers	6-8
6.2.3.3	3D Command/Parameter FIFO	6-9
6.2.3.4	2D Command/Parameter FIFO	6-9
6.2.3.5	2D Image Data FIFO	6-9
6.2.4	Display Memory Arrays	6-10
6.2.4.1	Surfaces	6-10
6.2.4.2	Texture Memory	6-10
6.2.4.3	Z-Buffers	6-10
6.2.4.4	Memory Buffer Allocation	6-10
6.2.5	Byte Ordering	6-11
6.3	BIOS Functions	6-12
6.3.1	BIOS Initialization: Power-On Self Test (POST)	6-12
6.3.2	Set Mode	6-14
6.4	Driver Functions	6-15
6.4.1	Device detection	6-15
6.4.2	Device configuration	6-16
6.4.3	Memory-mapped control address	6-17
6.4.3.1	Enable/Disable access	6-17
6.4.3.2	2D Command Port selector	6-18
6.4.3.3	2D Image Data Port selector	6-19
6.4.3.4	VGA I/O Ports	6-19
6.4.3.5	A0000H-based 128K VGA memory	6-20
6.4.3.6	PCI configuration space	6-20
6.4.4	Memory Address Allocation	6-21
6.4.4.1	Local memory address space	6-21
6.4.4.2	Display (video) memory address	6-22
6.4.4.3	16-bit Real Mode Addressing	6-23
6.4.4.4	16-bit Protected Mode Addressing	6-23
6.4.4.5	32-bit Mode Addressing	6-24

6.4.5	Physical Buffers for FIFOs	6-26
6.4.6	Linear/Segmented mode	6-27
6.4.7	Cursor control	6-28
6.4.7.1	Cursor Format	6-28
6.4.7.2	Cursor data buffer and surface	6-29
6.4.7.3	Set Monochrome Cursor data	6-30
6.4.7.4	Set Color Cursor data	6-31
6.4.7.5	Cursor Deferred control	6-33
6.4.7.6	Set Cursor Position and Turn Cursor On	6-34
6.4.8	Icon	6-36
6.5	2D Functions	6-37
6.5.1	Display Operations	6-37
6.5.1.1	Define a Surface	6-40
6.5.1.2	Channel Controls - assign surface	6-40
6.5.1.3	Channel Controls - Offsets & viewports location	6-42
6.5.1.4	Enable/Disable Channel Display	6-45
6.5.1.5	Set Display FIFO Controls	6-46
6.5.1.6	Set Channel 0 X,Y Scale Factors	6-47
6.5.1.7	Set Overlay Priority	6-52
6.5.1.8	Set Mix Controls - Blend	6-54
6.5.1.9	Set Mix Controls - Color Keying	6-56
6.5.1.10	Get Displayed Scanline	6-57
6.5.2	2D Command and Data FIFOs	6-58
6.5.2.1	Image/Data Control Port	6-59
6.5.2.2	2D Command Port	6-59
6.5.2.3	FIFO Status	6-60

6.5.3	2D Rendering Engine Commands	6-61
6.5.3.1	Wait Engine Idle	6-61
6.5.3.2	Check FIFO size	6-62
6.5.3.3	Clip Rectangle	6-63
6.5.3.4	Load Mono Pattern	6-64
6.5.3.5	Load Color Pattern	6-64
6.5.3.6	Load General pattern	6-65
6.5.3.7	Block fill (Opaque Rectangle)	6-66
6.5.3.8	Transparent Text	6-67
6.5.3.9	Opaque Text	6-68
6.5.3.10	BitBLT	6-69
6.5.3.11	Pattern BLT Without Source Operand	6-70
6.5.3.12	Display Memory Source BLT with/without Pattern	6-71
6.5.3.13	System Memory Source BLT with/without Pattern	6-72
6.5.3.14	Line Draw	6-76
6.6	3D Functions	6-80
6.6.1	Overview	6-80
6.6.1.1	Feature Set	6-81
6.6.1.2	3D Rendering Engine	6-82
6.6.1.3	3D Rasterization Pipeline	6-83
6.6.2	3D Engine Operation	6-86
6.6.2.1	Initialization	6-86
6.6.2.2	3D Command FIFO	6-88
6.6.3	3D Command Format	6-90
6.6.4	3D Primitive Commands	6-91
6.6.4.1	Triangle Strip/Fan	6-92
6.6.4.2	Triangle List	6-93
6.6.4.3	Point List	6-93
6.6.5	Copy Memory and DMA Command	6-94
6.6.6	Synchronization Command	6-96
6.6.7	Load State Register Command	6-99
6.6.8	Texture Loading	6-100
6.6.9	2D/3D synchronization	6-102
6.6.10	Buffer Issues	6-103
6.6.10.1	Buffer Alignment	6-103
6.6.10.2	Buffer Formats	6-103
6.6.11	Performance Issues	6-104

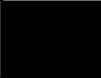
6.7	LCD Panel Programming	6-105
6.7.1	LCD Flat Panel Registers	6-105
6.7.1.1	Initialization	6-107
6.7.1.2	Set to Desired Mode	6-107
6.7.1.3	Standard VGA Modes	6-107
6.7.1.4	Enhanced Modes	6-107
6.7.1.5	Set Mode Sequence	6-109
6.8	Dual View	6-110
6.8.1	Overview	6-110
6.8.2	Enabling Second Display	6-110
6.8.2.1	Enable Simultaneous View	6-111
6.8.2.2	Enable Dual View	6-111
6.9	Motion Compensation	6-112
6.9.1	Overview	6-112
6.9.2	Block Diagram	6-114
6.9.3	Motion Compensation Client Commands	6-115
6.9.3.1	DMCM Commands	6-115
6.9.3.2	DMCM surface allocation	6-116
6.9.4	Registers	6-118
6.9.4.1	Start and End Decode Functions	6-118
6.9.4.2	CDU Start Motion Compensation Command	6-118
6.9.4.3	End of Stream Instruction	6-118
6.9.4.4	Format Conversion Instruction	6-118
6.9.5	Display Memory Requirements	6-120
6.9.5.1	Frame and Format Buffers	6-120
6.9.5.2	Screen Resolutions Supported	6-121
6.9.5.3	System Memory Requirements	6-121
6.10	Power Management	6-122
6.10.1	Overview	6-122
6.10.2	OnNow Power Management	6-122
6.10.3	VESA BIOS Power Management	6-122
6.10.4	Other Power Management	6-122
6.10.5	Registers for OnNow and VESA DPMS	6-123
6.10.5.1	D0 – On	6-123
6.10.5.2	D1 – Standby	6-123
6.10.5.3	D2 – Suspend	6-124
6.10.5.4	D3 – Off	6-124
6.10.6	Registers for Dynamic Power Management	6-125

6.11	Video Capture	6-126
6.11.1	Overview	6-126
6.11.2	I2C device interface	6-126
6.11.3	Video port control	6-129
6.11.4	Interrupt for video capture	6-131
6.12	TV Out.	6-132
6.13	Diagnostics	6-132
6.14	Software Definitions	6-133
6.14.1	Equates	6-133
6.14.2	Macros	6-142
6.14.3	Clock Synthesizer Programming Guide	6-143
6.14.3.1	Theory of Operation	6-144
6.14.3.2	Registers	6-145
6.14.3.3	Memory/Dot/Panel Clock Frequency Control	6-145
6.14.3.4	Memory Clock Synthesizer VCO Control	6-146
6.14.3.5	Dot/Panel Clock Synthesizer VCO Control	6-147
6.14.3.6	Selecting Register Value	6-148
6.14.3.7	Programming Limitations	6-149
6.14.3.8	Register Values for Common Frequencies	6-149

7	Application Notes	
7.1	Overview	7-2
7.2	Board Design	7-2
7.2.1	Power Supply	7-3
7.2.2	Block Diagram	7-4
7.2.3	Host Interface	7-5
7.2.4	Reference Clock	7-6
7.2.5	LCD Panel Interface	7-7
7.2.6	LVDS Transmitter Interface	7-8
7.2.7	CRT Monitor Interface	7-10
7.2.8	BIOS ROM Interface	7-11
7.2.9	Analog and Digital TV Encoder Interfaces	7-12
7.2.10	TV Decoder Interface	7-13
7.3	Power-On Strapping	7-14
7.3.1	Power-on Strapping Pins	7-14
7.3.2	Strapping Method	7-15
7.3.2.1	External Pull Down Resistor	7-15
7.3.2.2	Registers	7-15
7.3.2.3	Strapping Pin Functions	7-16
7.4	Board-level Testing	7-17
7.4.1	Test Mode Control Pins	7-17
7.4.2	Test Modes	7-18
7.5	Programmable (Peripheral) I/O	7-23
7.5.1	Customer I/O	7-23
7.5.1.1	Customer Pins	7-23
7.5.1.2	Customer Registers	7-24
7.5.2	Peripheral I/O	7-25
7.5.2.1	Peripheral I/O Pins	7-26
7.5.2.2	Peripheral I/O Registers	7-27

A	Appendix A	
A.1	Pixel Format Information	A-2
A.1.1	Enhanced Data Format	A-2
A.1.2	RGB Pixel Formats	A-3
A.1.3	Palettized/Color Lookup Table	A-5
A.1.4	CLUT Data Format	A-6
A.2	3D Data Formats	A-7
A.2.1	Coordinates	A-7
A.2.2	RGB Pixel Formats	A-7
A.2.3	ARGB Pixel Formats	A-7
A.2.4	Texel Formats	A-8
A.3	Video Formats	A-10
A.3.1	Cursor	A-11
A.3.2	VGA Data Formats	A-12
A.3.3	Planar	A-13
A.3.4	Text	A-14
A.3.5	Packed Pixel/Linear	A-15
B	Appendix B	
B.1	Graphics Modes Support	B-2
B.1.1	Video Display Window Memory Constraints	B-2
B.1.2	Definitions	B-2
B.1.3	Motion Compensation Memory Requirements	B-3
B.2	Graphics Modes Summary	B-4
B.2.1	Single Display with NTSC Video	B-5
B.2.2	Single Display with PAL Video	B-5
B.2.3	Dual display with NTSC Video	B-6
B.2.4	Dual display with PAL Video	B-7
C	Appendix C	
C.1	References	C-2

1 **SM3110 Features Overview**



Features Overview

2D/3D/Video LCD/CRT Graphics Accelerator

4 Mbytes of embedded graphics RAM

Bi/TriLinear MIPmap 3D Engine

Advanced Floating Point 3D setup engine
Z-buffering

3 Op 2D Engine

256 ROP on 8, 16, 24 bpp with transparency

Video playback acceleration

Color and Chroma key
Bi-linear (X,Y) interpolated scaling
Color space conversion
Motion Compensation
Bob and Weave

Industry standard Video input support

ZV port, I²C support

MyView™ dual display support

Independent LCD and CRT display

LCD panels and CRT support

TFT up to 1280x1024x24 bpp
DSTN up to 1280x1024x16bpp
CRT up to 1600x1200x16 bpp

AGP 2x and 66 MHz PCI 2.2 host interface with bus mastering

SmartPower™ power management system

Full VESA 2.0 DPMS and DDC2B support
2.5V and 3.3V operation with 5V tolerant I/O

304 pin PBGA package

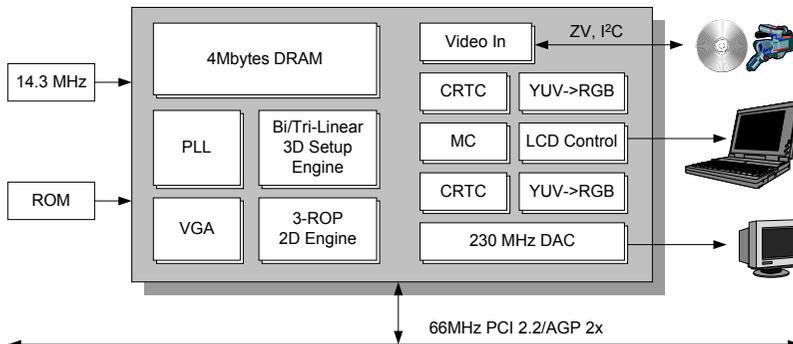
The SM3110 provides an optimal graphics sub-system for notebook computers. Its unique design embeds rich features and superior performance in a *single chip*.

Embedded Performance and Savings

The SM3110 integrates an advanced 3D engine, a robust 3 Op 2D accelerator, hardware video playback acceleration with interpolated scaling, independent dual display channels for LCD and CRT, a true color 230 MHz DAC and 4 Mbytes of embedded DRAM optimized for bandwidth availability and usage.

By integrating the optimal memory and functionality required by popular notebook applications, the SM3110 multimedia solution eliminates two or more external components, while attaining highest possible performance. For notebook computers this all adds up to ease of design and valuable savings in PCB space, weight, and sub-system costs.

Moreover, its SmartPower™ power management system ensures ultra-low power consumption during all operations and adheres to the ACPI goals for 1999 mobile computers which adds significant time to notebook battery life.



3D Engine

- Floating point 3D setup engine
- Trilinear MIPmapping
- Bilinear filtering
- True divide-per-pixel perspective correction
- 16-bit Z-buffer support
- Alpha blending
- Transparency
- Texture blending
- Colored fog
- Specular lighting
- Gouraud shading
- Logical OP, dithering
- Stipple support
- On-chip texture palette
- Supports 8, 16 or 32 bpp
- 1, 2, 4, 8, 16 bit texture map support
- Up to 1M Byte Command/Parameter buffer

Embedded 4M Bytes DRAM

- Embedded DRAM configurable for display buffers, Z-buffers, texture buffers or command buffers

2D Engine

- 3 OP bitBLT engine
- 256 ROP
- Color expansion
- Clipping
- Mono/Color pattern fill
- Opaque rectangle
- Transparent text
- Opaque text
- Color compare for color keying/transparency control
- 8, 16, 24bpp color
- Frame synchronization
- 32KB data FIFO
- 32KB command FIFO
- 256x256 monochrome or 32x32 color hardware cursor

High-performance AGP Interface

- AGP 2.0 interface, 1x and 2x with Sideband addressing
- 32-bit, 66MHz PCI 2.2 interface
- DMA bus mastering
- Linear addressable frame buffer
- Bi-endian data format support
- 3.3V host interface with 5V tolerance

Video Playback Acceleration

- Hardware destination color and chroma key support
- Colorspace conversion
- Motion compensation to ensure >30fps during MPEG/DVD playback
- Bob and weave
- Bi-linear (X, Y) interpolated scaling
- Independently scalable hardware windows for each display channel
- YUV-RGB conversion (4:2:2 -> CLUT8, 555, 565, 888)

Industry Standard Video Input

- ZV port
- I²C support
- Glueless interface to video digitizers
- Real-time 720x525 video capture
- On-the-fly horizontal downscaling

SmartPower™ = longer battery life

- Ultra-low power consumption achieved with dynamic control during all operations
- Comprehensive support for suspend, standby and hibernation modes
- VESA DPMS 2.0 and DDC2B support for both GreenPC and CRT Plug-and-play
- 2.5V internal operation, 3.3V I/O with 5V tolerance

Display Output

MyView™ dual display support

- Allows true multitasking by displaying different images on LCD and CRT simultaneously
- Independent resolution and refresh rate for each display for optimal display quality

High resolution support up to 1600x1200 CRT

- DSTN up to 1280x1024x16bpp
- CRT up to 1600x1200x16bpp

True color 230MHz palette DAC with gamma correction

- 4, 8, 16, 24, 32bpp support up to 1600x1200 resolution at 85Hz
- DDC2B support

DSTN, TFT flat panel support with gamma correction

- OEM-customizable Hardware Display Icon
- 9, 12, 18, 24 bpp TFT LCD panel support with dithering in resolutions up to 1280x1024 at 85Hz
- 8, 16, 24 bit DSTN with dithering and FRC support in resolutions up to 1280x1024 at 85Hz
- 12, 18, 24 and 36-bit panel interface
- 3,4,6 and 8-bit dithering support
- VGA mode LCD scaling and centering support
- Automatic panel power sequence control
- Fast shutdown for Hot-Battery-Switching
- Configurable default display at power-up
- Two independent hardware cursors

LVDS or TMDS Interface

Digital or Analog TV encoder interface

Maximum display resolutions supported

Mode	Resolution (w pixels x h pixels) x Color Depth (bits/pixel)	
Single Display	CRT	1600 x 1200 x 16
	CRT or TFT	1280 x 1024 x 24
	DSTN	1280 x 1024 x 16
	CRT or LCD	1024 x 768 x 32
Single Display w/NTSC DVD (full-screen)	CRT or LCD	640 x 480 x 24
MyView™ Dual Displays	LCD	CRT
	1280 x 1024 x 16	800 x 600 x 16
	1280 x 1024 x 8	1600 x 1200 x 8
	1024 x 768 x 24	800 x 600 x 24
	1024 x 768 x 16	1024 x 768 x 16
	1024 x 768 x 8	1600 x 1200 x 8
	800 x 600 x 32	800 x 600 x 32

Optimized drivers, BIOS and utilities

The SM3110 is designed to accelerate Microsoft's DirectX architecture including DirectDraw and Direct3D, plus OpenGL in the Windows environment. Optimized drivers and utilities for Windows 95/98, NT 4.0, and Windows 2000 operating systems back the high performance hardware. An ICD driver for OpenGL is also available.

The SM3110 is supported by the industry standard VESA VBE-compliant VGA BIOS to provide optimum performance and functionality in VGA and VESA extended display modes.

The SM3110 is backed by a strong suite of utilities to help with the customization and update of BIOS. A utility to generate and fine tune the timing configuration of LCD panels in the BIOS is also available.

Trademarks are the property of their respective holders.

Software Drivers	Resolutions Supported	No. of colors
Microsoft Windows 95, 98, 2000, NT 4.0, Direct3D, OpenGL	640x480, 720x480, 800x600, 1024x768, 1280x1024, 1600x1200	256 or 65536
	640x480, 720x480, 800x600, 1024x768, 1280x1024	16.8 million
Microsoft WDM/VPE/VPM Driver	Resolution-independent	-
DVD Playback	800x600	65536
	640x480	16.8 million

2 Architecture Overview



2.1 System Configuration

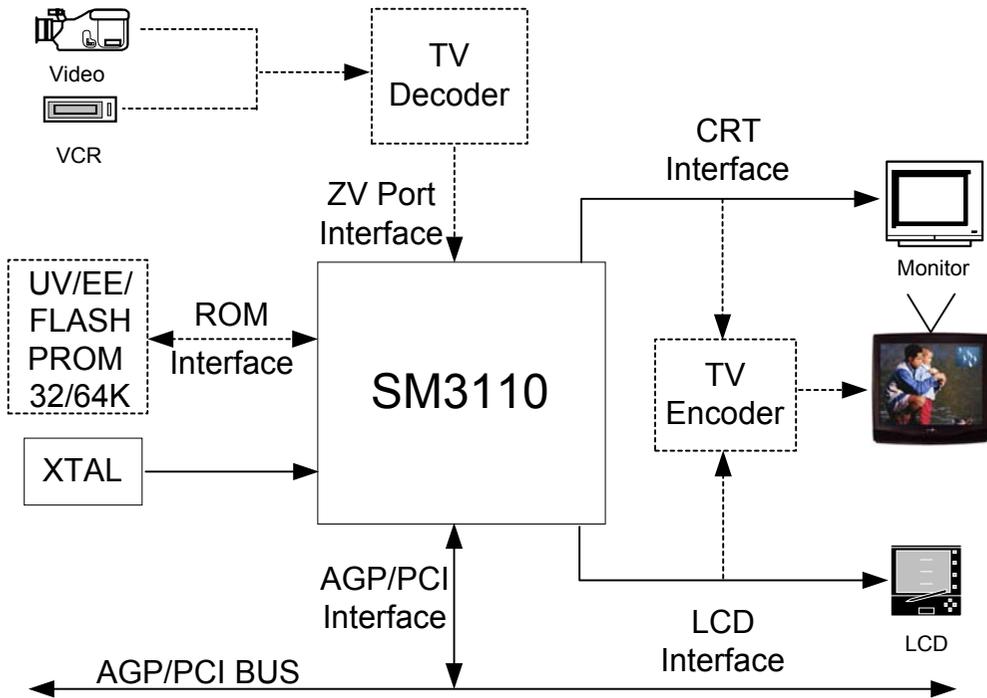


Figure 2-1. Typical SM3110 System Configuration

The SM3110 is an integrated graphics accelerator designed for high performance in a limited space environment, such as a laptop computer. The SM3110 Graphics Accelerator has a number of interfaces to accommodate a variety of configurations and devices.

The host interface is glueless for a direct connection to either AGP or PCI buses. A Zoom Video input port (ZV Port) is provided to accept live video. A single external crystal supplies reference clock timing to three internal independently programmable Phase Locked Loops (PLL). The BIOS ROM port can access a separate graphics ROM or it can be converted for output to a digital TV encoder if the graphics ROM is not used in the system. The display outputs can drive a DSTN or TFT panel and a CRT monitor simultaneously. The data on two displays can be either the same or different, providing a flexible system configuration. Either display port can connect to a TV encoder for TV display. The DAC and clock synthesizers are also built in to minimize the need for external components.

2.2 Internal Configuration

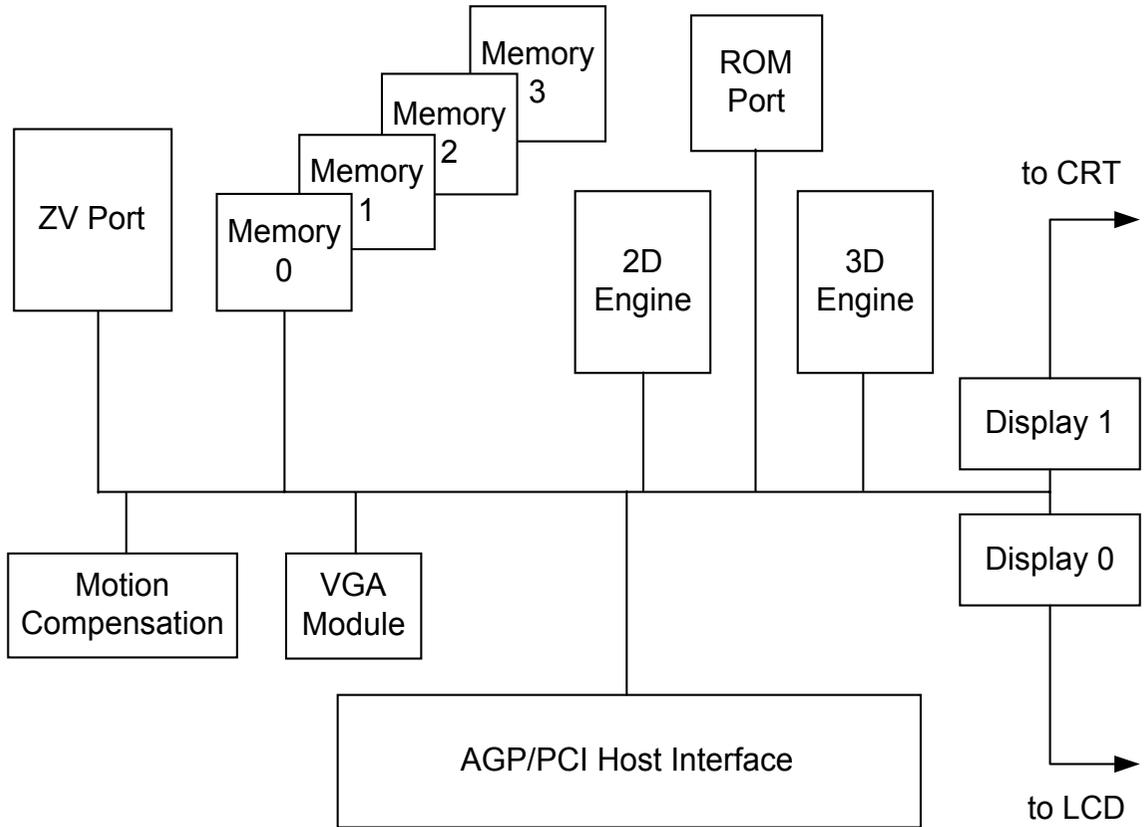


Figure 2-2. SM3110 Internal View

The architecture consists of a host interface, a standard VGA module, a Zoom Video port, a ROM port, four 1MB modules of high performance embedded memory, a 2D engine, a 3D engine, a motion compensation module for accelerating MPEG-2 playback and sophisticated dual display circuitry to support single/simultaneous/dual display on any LCD panel/CRT monitor/TV combination.

2.2.1 Host Interface

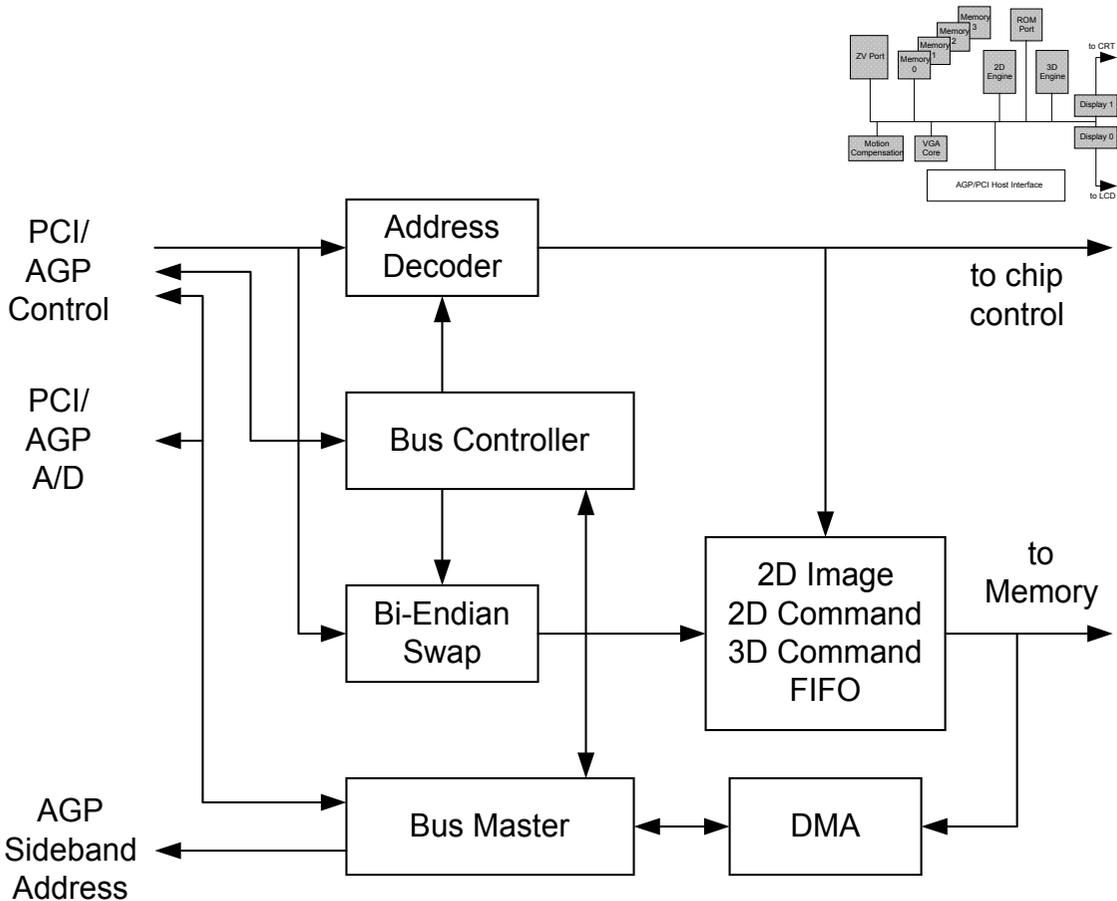


Figure 2-3. Host Interface

The SM3110 has a glueless system interface that is compliant with PCI Local Bus, Revision 2.2 and supports Intel’s Accelerated Graphics Port (AGP), Revision 2.0. The system interface can initiate bus transactions as a master and supports 5V and 3.3V signaling and 66 MHz operation. It supports PCI Bus Power Management Interface for ACPI compliant power management. AGP support includes AGP 1x and 2x with Local Texturing, AGP Pipelining, AGP Sideband Addressing, and AGP Frame Mode.

The host interface supports bi-endian data formats. It has 2D and 3D command/data buffers to improve bus throughput. It handles DMA in bus master mode to improve data transfer while leaving the CPU free for other tasks. In AGP configuration the host interface also supports sideband addressing for higher data throughput.

2.2.2 Embedded Memory

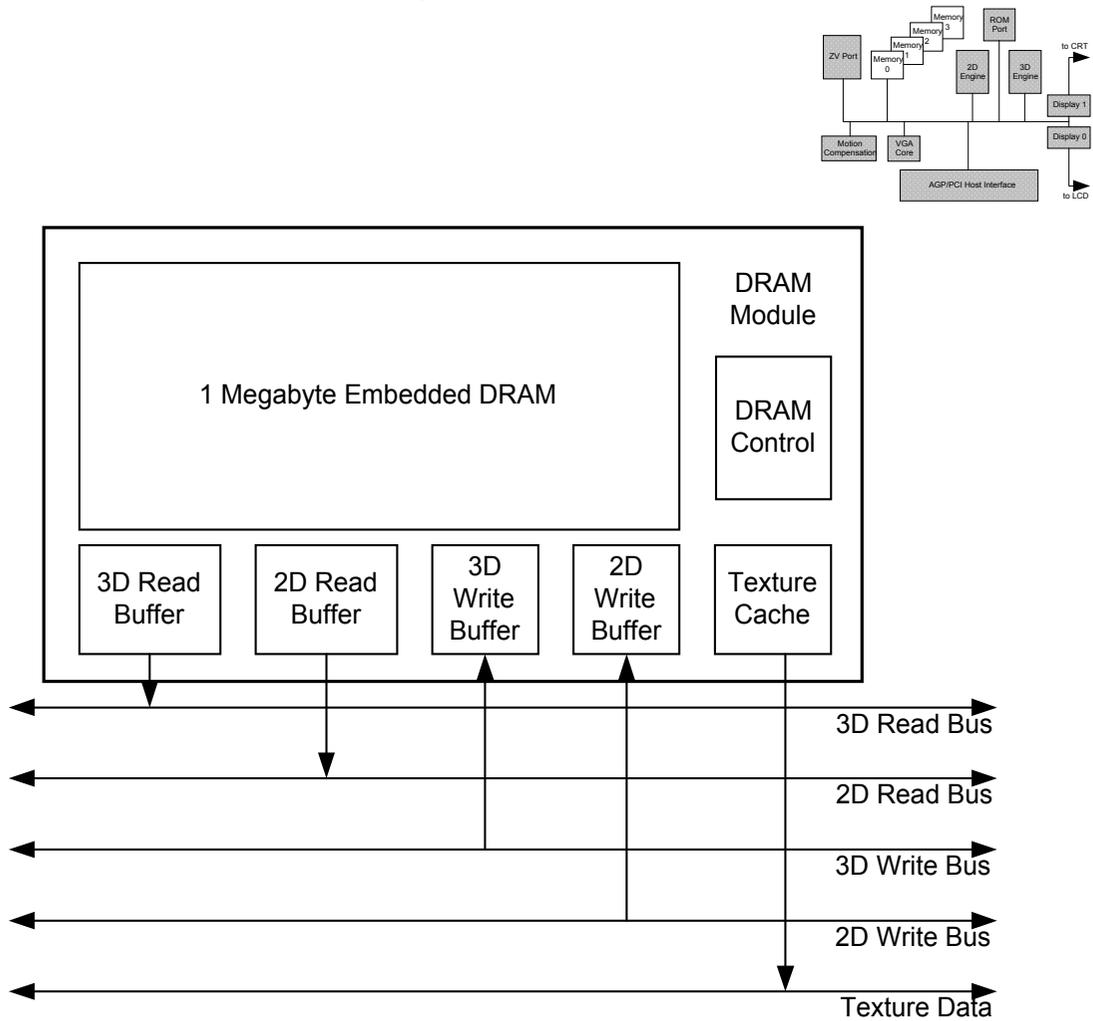


Figure 2-4. Memory Module Internal Path

The SM3110 includes 4 megabytes of embedded memory. Four memory modules of 1 megabyte each comprise the internal memory. The modules have separate read and write buffers for the 2D and 3D data buses and one buffer for texture cache. When combined, the interface to internal memory provides over 6.4 gigabytes/second peak memory bandwidth.

The SM3110 claims 128MB of memory space through AGP/PCI configuration. This 128 megabytes is divided into 4 contiguous 32MB address ranges. The lowest 32MB is for little endian data. The next 32MB is reserved. The swap byte within doubleword big endian copy occupies the next 32MB and the swap bytes within word big endian copy occupies the highest 32MB. The last 256KB of each 32MB address range is reserved for 2D and 3D control accesses.

Table 2-1. Local Address Space Allocation

<i>Range</i>	<i>Size</i>	Little Endian <i>Base</i>	Reserved <i>Base</i>	Big Endian (Double- Word Swap) <i>Base</i>	Big Endian (Word Swap) <i>Base</i>
Display Memory	31.75MB	00000000H	02000000H	04000000H	06000000H
Control	0.25MB	01FC0000H	03FC0000H	05FC0000H	07FC0000H
Total	32.00MB				

Besides the memory address mentioned above, SM3110 also provides access to register I/O and 128KB memory at 000A0000H address space for compatibility with standard VGA. Larger physical memory available in super VGA configuration is made accessible via an I/O addressed bank offset register.

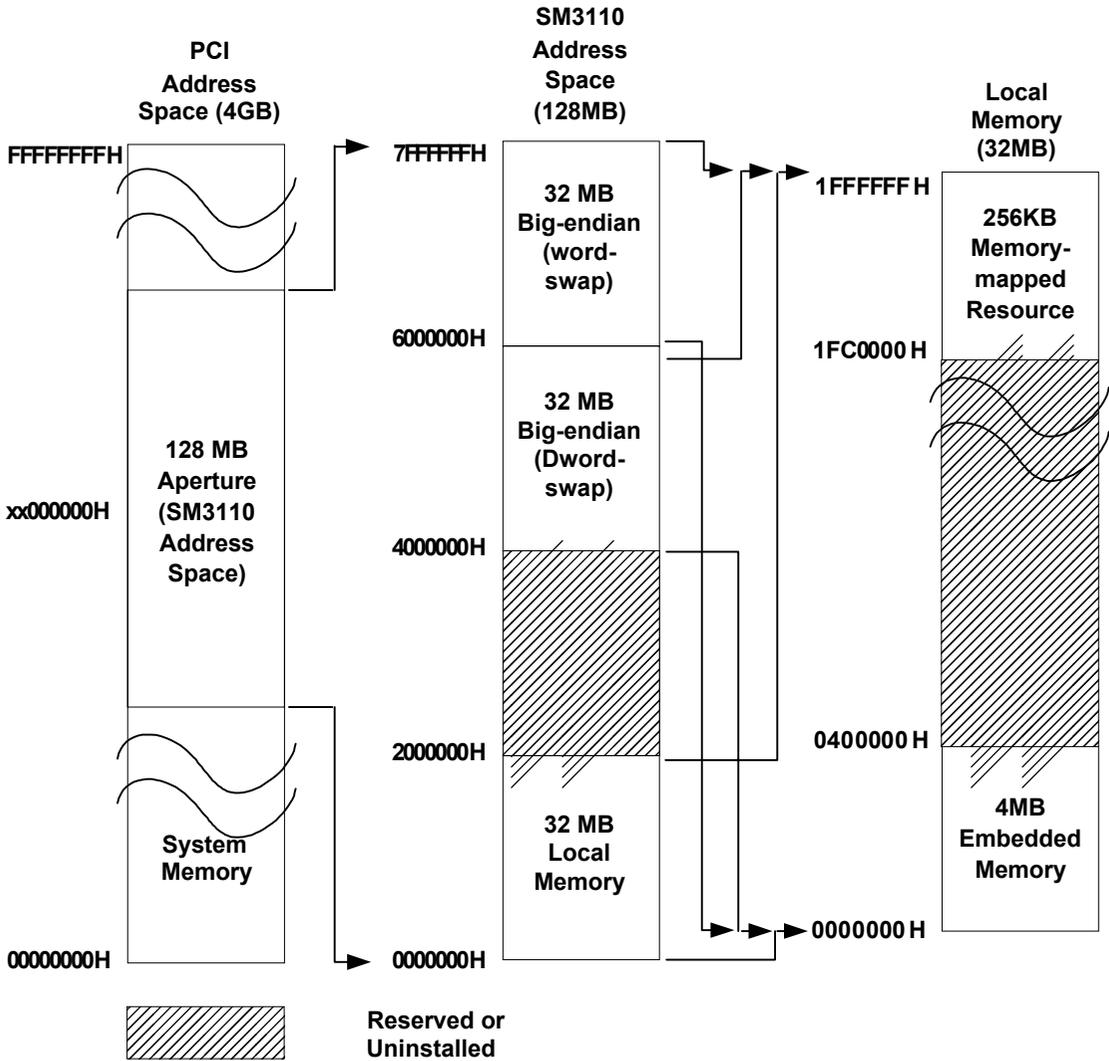


Figure 2-5. Address Space Allocation

2.2.3 VGA Module

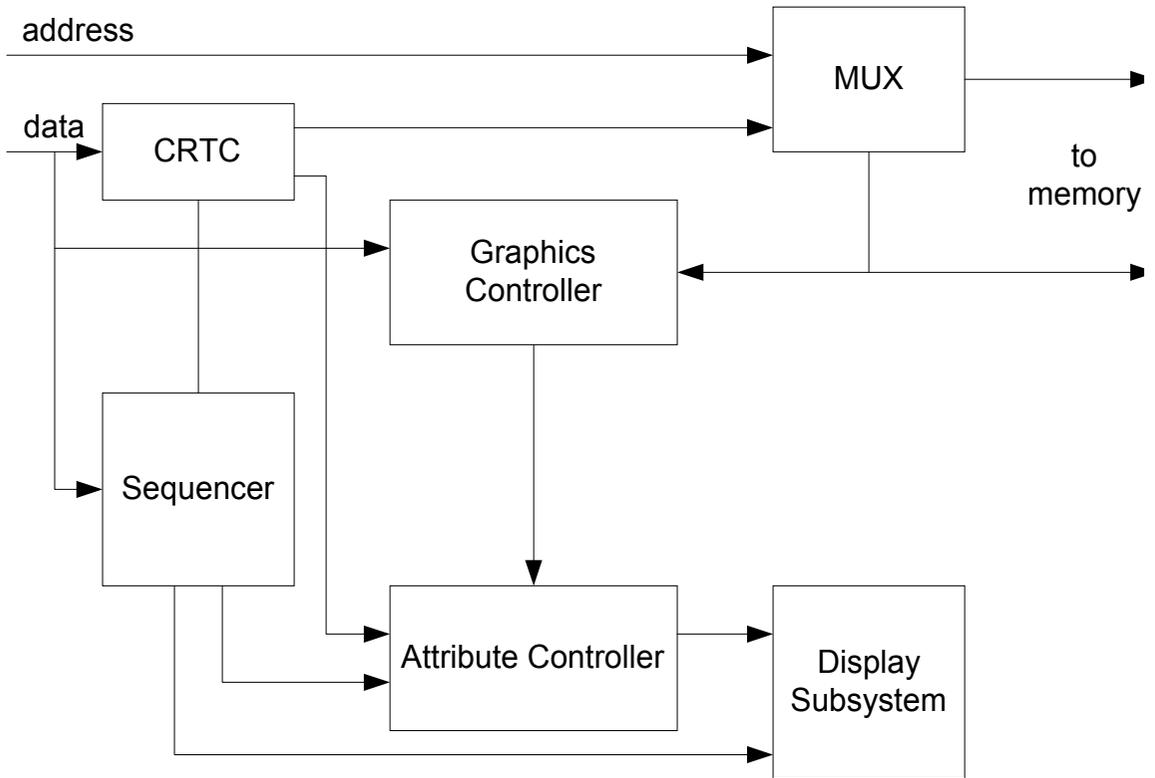
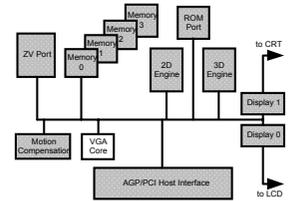


Figure 2-6. VGA Module

SM3110 has a VGA core fully compatible with the industry-standard IBM VGA adapter. This module provides direct access logic to the frame buffer and supports all functions performed by the Sequencer and VGA Graphics Controller registers. Standard VESA and VGA display timings are also provided by this module.

2.2.4 2D Rendering Engine

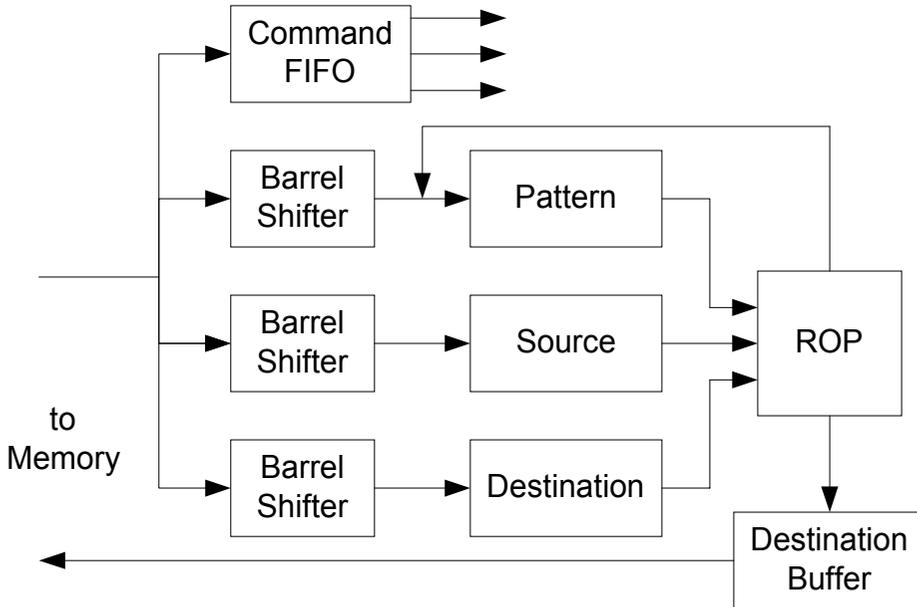
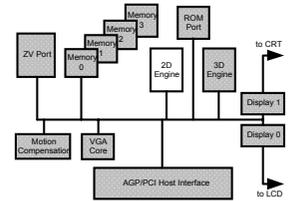


Figure 2-7. 2D BitBLT Engine Internal Data Path

The SM3110 has a high performance 2D bitBLT engine. The 2D engine has a three-operand raster operations (ROP) engine supporting linear and rectangular addressing which is fully Microsoft Windows compatible. It also supports screen-to-screen or memory-to-screen bitBLT with color expansion and transparency control.

The 2D command FIFO receives and decodes commands buffered for the 2D engine. Separate source data, destination data and pattern buffers have their own shifters to align data before they are fed into the ROP unit. A destination buffer stores the results before writing back to the frame buffer for optimized performance in both the 2D engine and the memory interface. This engine also provides transparency, color expansion and color keying during the bitBLT operation.

2.2.5 3D Rendering Engine

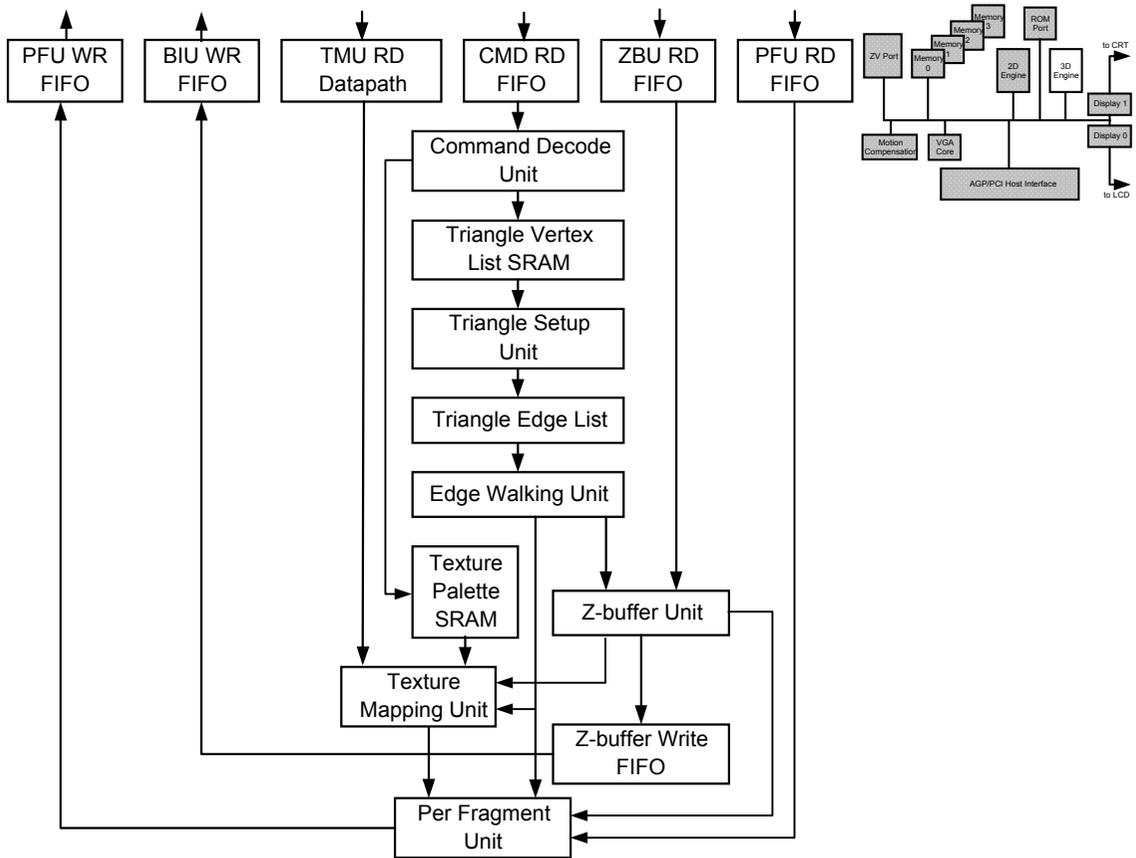


Figure 2-8. 3D Engine Internal Data Path

The SM3110 has a multi-stage 3D rendering engine. It is a one-cycle, bilinear, perspective corrected and two-cycle, trilinear, texture-mapped, perspective corrected MIPmap texture calculation engine. The major components of this rendering engine include a command unit, a triangle setup engine, a texture mapping unit, a texture cache buffer, a texture palette, a per fragment unit and a 16-bit Z-buffer.

The command unit provides command decoding and control of the engine. The triangle setup engine speeds up calculations in the setup process by getting setup information from the host and performing the necessary calculations, freeing the host CPU for other tasks. The texture unit calculates correct texels for each pixel with a bilinear or trilinear MIPmapped filter and perspective correction. The texture palette converts indexed color into 16-bit color. The Z-buffer unit calculates Z value of each new pixel and compares against the Z value in the frame buffer for accurate hidden surface removal. The rendering engine renders each pixel using lighting, fogging, specular highlighting and texturing functions to generate realistic images at a high frame rate.

2.2.6 Display Pipeline

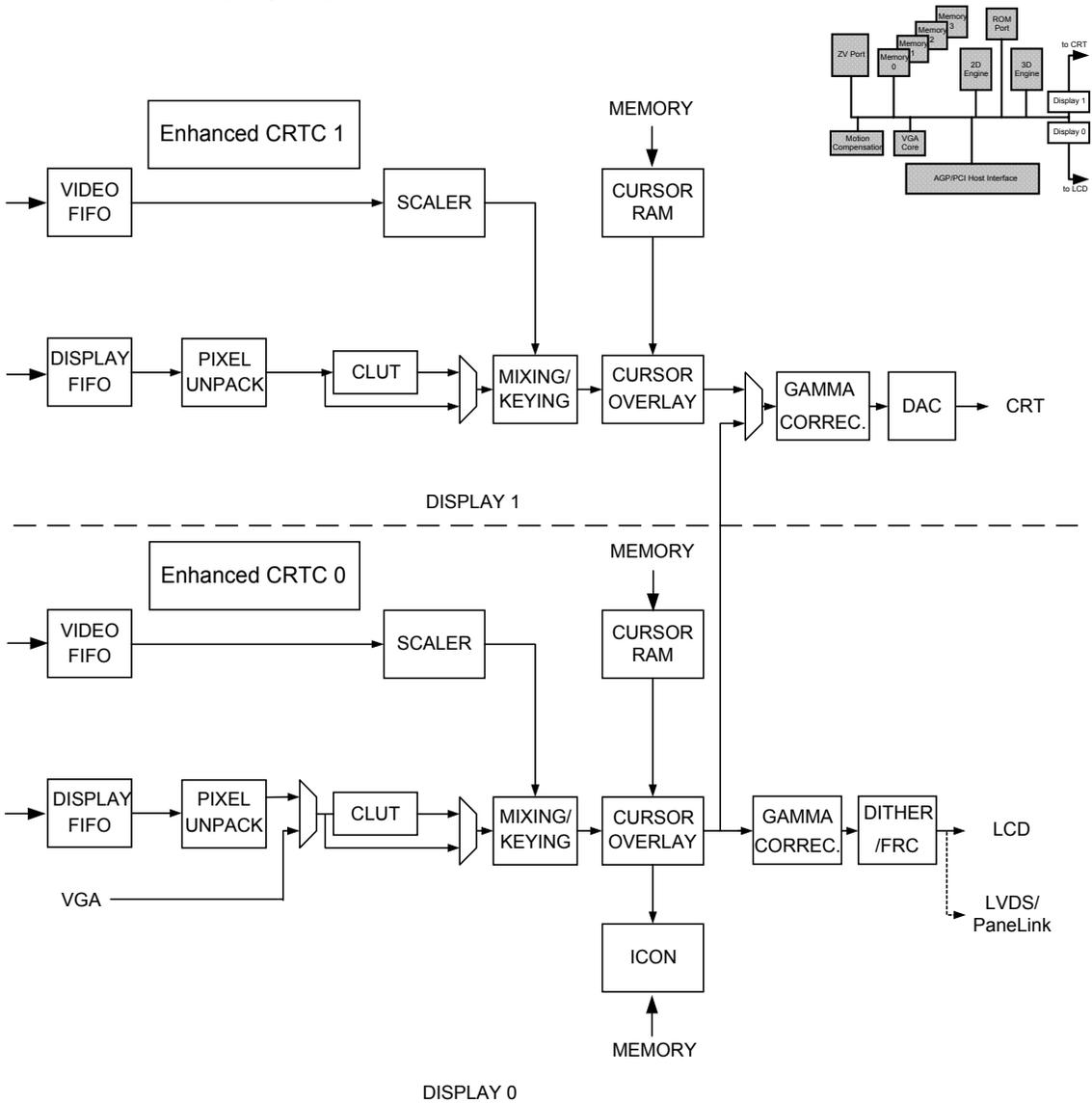


Figure 2-9. Display Pipeline

The display subsystem has two independent display pipelines. Each pipeline has its own enhanced CRT controller (CRTC), display FIFO, video FIFO, pixel unpacking circuit, color lookup table (CLUT), color keying circuit and scaler. Display 0 supports all standard VGA functions including all standard VESA display modes. Both displays support extended VESA modes for higher visual quality. The two enhanced CRT controllers can be set to display different data with different resolutions and refresh rates. Display 1 can accept display 0 data through a multiplexing circuit.

The last stages of the two pipelines differ somewhat. Display pipeline 0 is optimized for flat panel display with gamma correction. It has a software selectable dithering and Frame Rate Control (FRC) circuit to improve display quality on LCD panels. The same interface can also provide digital data to drive an external LVDS/Panelink transmitter. It supports 8, 16, 24 and 36 pin interface DSTN panels with dithering and FRC. It also supports a 9, 12, 18, 24 and 36 pin interface TFT panels with dithering, and supports scaling and centering if the display resolution is lower than panel resolution. SM3110 also supports panning if the display resolution is larger than the panel resolution.

Display pipeline 1 integrates a true color 230 MHz palette DAC with gamma correction. It can display 4, 8, 15, 16, 24 and 32 bits per pixel (bpp) with standard VGA, VESA and programmable enhanced display formats up to 1600 by 1200 resolution at 85Hz. VESA Display Power Management Signaling 1.0 (DPMS 1.0) is supported for power management. The SM3110 also supports VESA Display Data Channel 2.0/2.0B (DDC2/DDC2B) for monitor Plug & Play support. The CRT can have display timing and control independent of the flat panel for dual independent displays.

The display subsystem supports two hardware cursors and a hardware icon. The size of both the cursors and icon is 32 pixels by 32 pixels in 8bpp color mode or 256 pixels by 256 pixels in monochrome mode. The cursors can be on both display pipelines, while the icon can only be on the display 0 pipeline.

2.2.7 Video Input

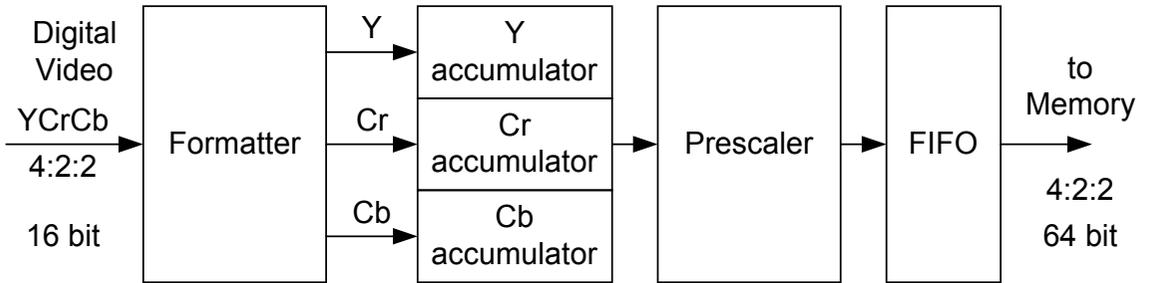
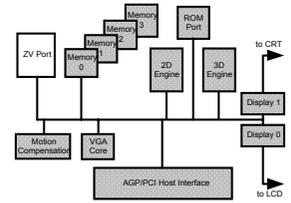


Figure 2-10. Video Input

The SM3110 supports PCMCIA Zoomed Video (ZV) Port and direct video input of ITU-R BT. 601 compliant YCrCb 4:2:2 digital video data.

The video data from this port is first pixel-aligned by a formatter and put into an accumulator. When two complete pixels are accumulated, the decimator circuit prescales the data and writes the scaled result into the frame buffer through its own FIFO for video capture.

2.2.8 Motion Compensation

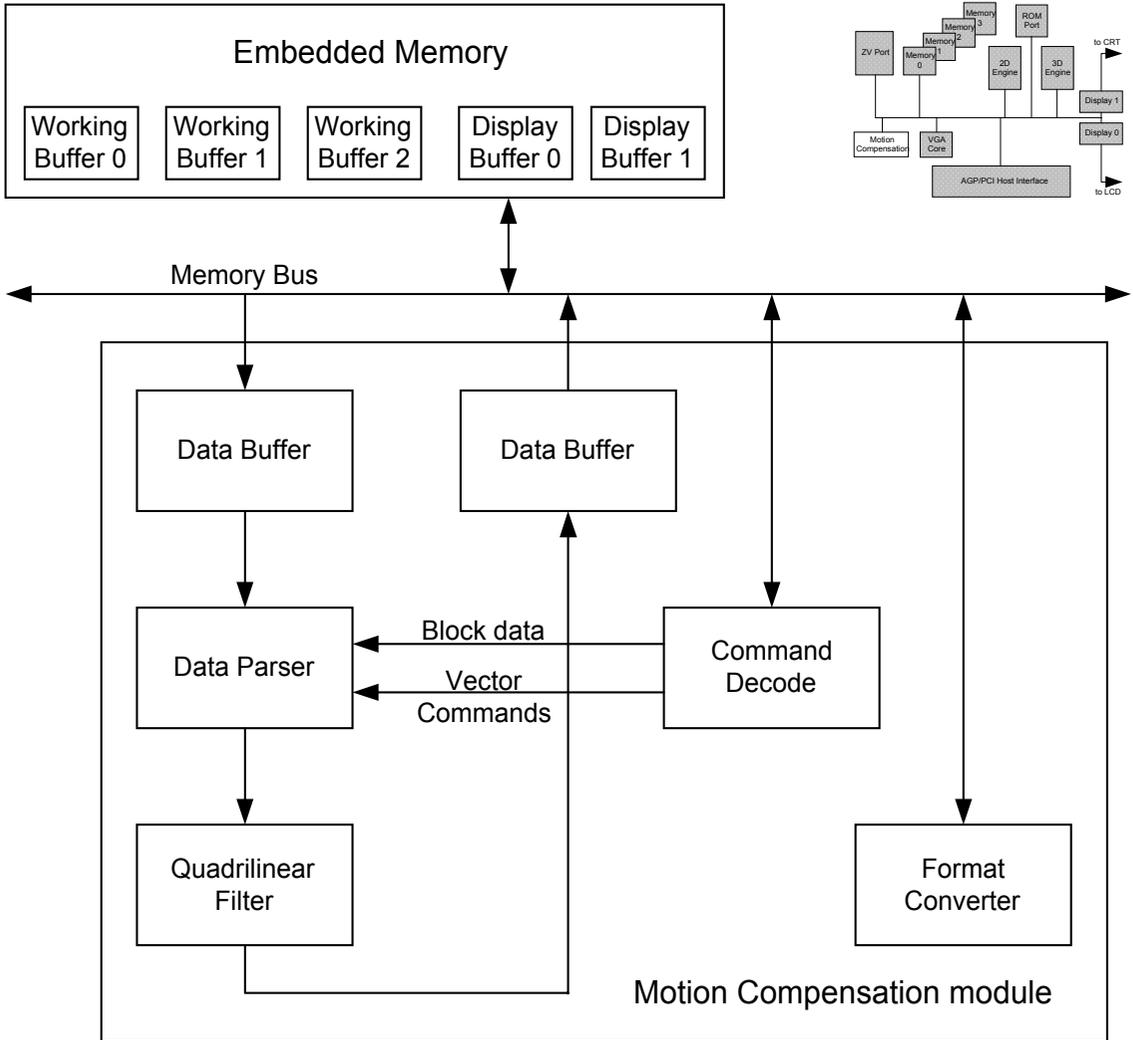


Figure 2-11. Motion Compensation Subsystem

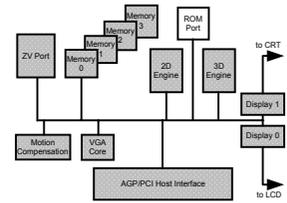
The motion compensation subsystem consists of the motion compensation module and five buffers residing in internal memory. The motion compensation module accepts enhanced MPEG-2 video data, parses the macroblock data and applies the quadrilinear filter. The resulting picture is assembled in the working buffers and then converted on the fly from the input 4:2:0 YCrCb format to output 4:2:2 YCrCb format and written to the display buffer.

2.2.9 ROM Port

The SM3110 supports external display BIOS ROM in standard VGA BIOS address 0xC000:0000. The ROM can be EPROM, EEPROM, or flash ROM. The size is either 32KB or 64KB and is set by power-on strapping.

If display BIOS is incorporated in system BIOS, this port can be converted to drive an external digital TV encoder. The image can be programmed to be the same as either the CRT or the panel display.

The ROM port can also be used for any 8-bit peripheral I/O function through the peripheral port at address offset +1FE000H.



2.3 Additional Features

2.3.1 Power Management

The SM3110 supports PCI Bus Power Management Interface for ACPI compliant power management, VESA Display Power Management Signaling 1.0 (DPMS 1.0) and VESA Display Data Channel 2.0/2.0B (DDC2/DDC2B) for monitor Plug & Play support.

The panel interface, DAC, 2D engine, 3D engine, both scalers and the entire display 1 (CRT) pipeline can be turned off individually to tailor the chip configuration for the lowest power consumption. The SM3110 allows the lowest possible power use dynamically through driver/register-level control for the task at hand.

2.3.2 Board-level Test Functions

The SM3110 has build-in test functions to perform open/short connection testing to help in detecting manufacturing problems. This simple, effective method allows quick detection of any shorted connections or failed soldering. SM3110 also has an isolation test function which allows the board to be tested without response from the SM3110.

2.3.3 Clocks

The SM3110 has 3 internal clock synthesizers which provide a wide range of frequencies to the memory clock and the two display clocks. The inputs of the synthesizers are from a single clock source, which can be an oscillator or a crystal. The typical frequency is 14.31818 MHz.

2.3.4 Inter-IC (I²C) Interface

The SM3110 has 8 programmable bi-directional pins to support I²C and other control functions. Two sets of pins are provided to support DDC2B monitor control and a TV decoder/encoder for video in/TV out functions. More channels can be provided if the need arises.

3 Data Sheet



3.1 Pin Descriptions

3.1.1 AGP/PCI Interface

Note: For detailed pin description, please refer to AGP Spec. 2.0 and PCI Spec. 2.2.

Signal Name	Pin #	*Input/ Output	Description
AD_STB[1:0]	AC11, AC20	I5/O2	AD bus strobe
AD[31:0]	AC8, Y9, AC9, AA9, AB10, Y10, AC10, Y11, AB12, Y13, AC12, AA13, AC13, Y14, AB14, Y15, Y18, AC17, Y19, AB18, AA19, AC18, Y20, AB19, AC21, Y21, AC22, Y23, AB22, AA23, AC23, AB23	I5/O2	AD bus
C/BE[3:0]#	AA12, AC14, AC16, AA20	I5/O2	Bus Command and Byte Enable
CLKRUN#	AB2	I5/O2	Clock start request
DEVSEL#	AB16	I5/O2	Device Select
FRAME#	AA15	I5/O2	Cycle Frame (PCI only)
GNT#	Y5	I5/O2	Bus request Granted
IDSEL	AA11	I5/O2	Initialization Device Select (PCI only)
INTA#	AC1	I5/O2	Interrupt request A
IRDY#	AC15	I5/O2	Initiator Ready
TRDY#	Y16	I5/O2	Target Ready
STOP#	AA16	I5/O2	Stop
M66EN	AC19	I5/O2	66MHz Enable
PAR	AA17	I5/O2	Parity
RBF#	AC4	I5/O2	Read Buffer Full
REQ#	AC2	I5/O2	Bus Request
RST#	AA4	I5/O2	System Reset
SB_STB	AC6	I5/O2	SideBand Address Strobe
SBA[7:0]	AA8, AB8, Y8, AC7, Y7, AB6, Y6, AC5	I5/O2	SideBand Address Bus
CLK	Y4	I5/O2	System Clock
ST[2:0]	AB4, AA5, AC3	I5/O2	Status Bus
AGPCNTL#	AA1	I2	AGP Pad Power Control. This is not part of AGP bus signal.
AGPVref	Y22	AI	AGP Pad voltage reference input. This is part of Silicon Magic's AGP power management.

*See Pad Description, Section 3.1.12

3.1.2 Panel Interface

Signal Name	Pin #	I/O*	Alternate Function	Description
FPCLKO	D13	I6/O5		Clock Output to Flat Panel
FP	A14	I6/O5		Frame Pulse for Flat Panel Timing Control
LP	A13	I6/O5		Line Pulse for Flat Panel Timing Control
FPD[35:29]	E1, E2, E3, E4, D1, D2, C1	I6/O5		Flat Panel Data[35:29]
FPD[28:25]	C2, B1, A1, D4	I6/O5	TEST[3:0]	Flat Panel Data[28:25] or Test Selection pins 3-0
FPD[24]	A2			Flat Panel Data[24]
FPD[23:0]	C4, D5, A3, B4, A4, A5, A6, C6, D6, A7, B7, D7, A8, D8, A9, C9, D9, A10, B10, D10, A11, D11, A12, C12	I6/O5	VDO[23:0]	Flat Panel Data[23:0] or digital data for TV/TMDS/LVDS
FPDE	D14	I6/O5		Flat Panel Display Enable
FPVCC	B13	I6/O3		Flat Panel VCC Control
FPVEE	D12	I6/O3		Flat Panel VEE Control
FPBL	A15	I6/O5		Flat Panel Back Light Control

*See Pad Description, Section 3.1.12

Note: Switching pin function to Alternate Function is controlled by register, except for FPD[28:25], which are switched to test function by TEST# (pin C15).

3.1.3 ROM Interface

Signal Name	Pin #	I/O ³	POWER-UP Configuration	Alternate Function	Description
ROMA [7:6]	E20, C22	I7/O4		VDO[7:6]	ROM Address [7:6] ¹
ROMA [5:0]	B23, A23, A22, D20, A21, B21	I6/O3		VDO[5:0]	ROM Address [5:0] ¹
ROMA8	C23	I7/O4	CONFIG[0]	VDO8	ROM Address [8] ^{1,2}
ROMA9	D22	I7/O4	CONFIG[1]	VDO9	ROM Address [9] ^{1,2}
ROMA10	E21	I7/O4	CONFIG[2]	VDO10	ROM Address [10] ^{1,2}
ROMA11	F20	I7/O4	CONFIG[3]	VDO11	ROM Address [11] ^{1,2}
ROMA12	D23	I7/O4	32K/64K ROM	VDO12	ROM Address [12] ^{1,2}
ROMA13	E22	I7/O4	INT/EXT PLL	VDO13	ROM Address [13] ^{1,2}
ROMA14	F21	I7/O4	INT/EXT SCLK	VDO14	ROM Address [14] ^{1,2}
ROMA15	E23	I7/O4	AGP/PCI	VDO15	ROM Address [15] ^{1,2}
ROMD[7:4]	C20, A20, D19, A19	I6/O3			ROM Data [7:4] ¹
ROMD3	D18	I6/O3		HSYNC2	ROM Data [3] ¹
ROMD2	A18	I6/O3		VSYNC2	ROM Data [2] ¹
ROMD1	D17	I6/O3		DCLK2	ROM Data [1] ¹
ROMD0	A17	I6/O3		BLANK2#	ROM Data [0] ¹
ROMFW#	D15	I6/O3			Flash ROM Flash Write Command
ROMOE#	D16	I6/O3			ROM Output Enable
ROMXW#	A16	I6/O3			Peripheral I/O Write Command
ROMXR#	B16	I6/O3			Peripheral I/O Read Command

- Note:
1. These pins have alternate functions as shown in the Alternate Pad Functions section below.
 2. These pins have configuration functions at power-up. See the Alternate Pad Functions section below.
 3. See Pad Description, Section 3.1.12

3.1.4 ZV Port

Signal Name	Pin #	I/O*	Description
ZVPCLK	L21	I1	ZV Port Pixel Clock
ZVHREF	T21	I7/O4	ZV Port Horizontal Refresh (Sync)
ZVVSYNC	T20	I7/O4	ZV Port Vertical Sync
ZVUV[7:0]	N20, N21, N23, M20, M21, M22, M23, L20	I7/O4	ZV Port UV Data [7:0]
ZVY[7:0]	T23, R20, R21, R22, R23, P20, P22, P23	I7/O4	ZV Port Y Data [7:0]
ZVCREF	U23	I7/O4	For Philips 7110/7111 Chips

*See Pad Description, Section 3.1.12

3.1.5 CRT Interface

Signal Name	Pin #	I/O*	Description
RED	V4	AO	Analog RED Output
GREEN	V3	AO	Analog GREEN Output
BLUE	W1	AO	Analog BLUE Output
RSET	U4	AI	DAC Full-scale Adjustment Resistor
COMP	V1	AI	DAC Compensation Input
DCLK	T1	I6/O3	Pixel Clock
HSYNC	U2	O3	Horizontal Sync
VSYNC	U1	O3	Vertical Sync
BLANK#	T4	O3	Blank

*See Pad Description, Section 3.1.12

3.1.6 Reference Clock Input

Signal Name	Pin #	I/O*	Description
XCLKI	W21	I4	Crystal/Oscillator Input
XCLKO	W20	O1	Crystal Output

*See Pad Description, Section 3.1.12

3.1.7 Programmable I/O Pins

Signal Name	Pin #	I/O*	Description
CIO7	K23	I6/O3	User Programmable Input/Output [7]
CIO6	K22	I6/O3	User Programmable Input/Output [6]
CIO5	K21	I6/O3	User Programmable Input/Output [5]
CIO4	K20	I6/O3	User Programmable Input/Output [4]
CIO3	L23	I6/O3	User Programmable Input/Output [3]/I ² C Data Line
CIO2	L22	I6/O3	User Programmable Input/Output [2]/I ² C Clock Line
CIO1	R3	I6/O3	User Programmable Input/Output [1]/DDC2B Data Line
CIO0	R4	I6/O3	User Programmable Input/Output [0]/DDC2B Clock Line

*See Pad Description, Section 3.1.12

3.1.8 Testing Pins

Signal Name	Pin #	I/O*	Description
TEST#	C15	I1	Test Mode. Not connected in normal use.
IOTST[8:1]	H20, H22, H23, G20, G21, G22, G23, F23	O5	Eight outputs for I/O testing.

*See Pad Description, Section 3.1.12

Note: These pins are for testing use only.

3.1.9 Reserved Pins

Signal Name	Pin #	Description
Reserved[23:1]	R2, R1, J2, J4, H1, H2, H3, H4, G1, G2, G4, F1, F3, F4, J23, J21, J20, AB21, Y17, Y12, AA7, P4, P3, P1, N4, N2, N1, M4, M3, M1, L1, L2, L4, K1, K3, K4, J1	Reserved. Do not connect.

3.1.10 Digital Power and Ground Pins

Signal Name	Pin #	Description
VDD_IO	B2, B5, B8, B11, B15, B18, B22, F2, F22, J22, K2, P2, AA2, AA22, AB5, AB9, AB13, AB17	18 of 3.3V power pins for I/O pads
VDD_C	M2, N22, T2, T22, AB3, AB7, AB11, AB15, AB20	9 of 2.5V power pins for core circuit
GND	C3, C7, C10, C14, C17, D3, D21, G3, H21, J3, K10, K11, K12, K13, K14, L3, L10, L11, L12, L13, L14, M10, M11, M12, M13, M14, N3, N10, N11, N12, N13, N14, P10, P11, P12, P13, P14, P21, T3, U22, AA10, AA14, AA18, AA21, AA3, AA6, AB1	47 Digital ground pins
VDD_R	B3, B6, B9, B12, B14, B17, B19, B20	8 of 2.5V power pins for internal memory modules
GND_R	C5, C8, C11, C13, C16, C18, C19, C21	8 ground pins for internal memory modules
VDD_P	U20	Digital 2.5V for PLL of MCLK, DCLK, and PCLK
GND_P	U21	Digital ground for PLL of MCLK, DCLK, and PCLK
VDD_S	Y2	Digital 2.5V for system clock DLL
GND_S	Y3	Digital ground for system clock PLL

3.1.11 Analog Power and Ground Pins

Signal Name	Pin #	Description
AVDD_P	W22, V22, V21	3 analog power pins for PLL
AGND_P	W23, V23, V20	3 analog ground pins for PLL
AVDD_D	U3, W2	2 analog power pins for DAC
AGND_D	V2, W3	2 analog ground pins for DAC
AVDD_S	W4	Analog power for system clock DLL
AGND_S	Y1	Analog ground for system clock PLL

3.1.12 Pad Descriptions

I/O Type	I/O	Internal Pull Up	Tri-State	Current (mA)	Capacitance (pF)	Note
I1	I	--	--	--	7.337	Clock Pad
I2	I	--	--	--	7.346	
I3	I	--	--	--	7.346	Schmitt-triggered
I4/O1	I/O	N	N	--		OSC Pad
I5/O2	I/O	N	N	24		
I6/O3	I/O	N	Y	4	7.754	
I7/O4	I/O	Y	Y	4	7.755	
I7/O4	I/O	Y	Y	4	7.754	
I6/O5	I/O	N	Y	8	7.387	Limited Slew Rate
I6/O3	I/O	N	Y	4	7.754	
I6/O5	I/O	N	Y	8	7.387	
O6	O	N	Y	2	7.755	
O3	O	N	Y	4	7.754	
AI	Ana	--	--	--		Analog Signal Pad, Input
AO	Ana	--	--	--		Analog Signal Pad, Output
--	Pwr	--	--	--	--	Digital Power, 2.5V
--	Pwr	--	--	--	--	Digital Power, 3.3V
--	APwr	--	--	--	--	Analog Power
--	Gnd	--	--	--	--	Digital Ground, 2.5V
--	Gnd	--	--	--	--	Digital Ground, 3.3V
--	AGnd	--	--	--	--	Analog Ground

3.1.13 Alternate Pad Function

Signal Name	Pin #	I/O ³	Alternate Function	Description
ROMA [7:6]	E20, C22	I7/O4	VDO[7:6]	Digital Data[7:6] for TV/TMDS/LVDS Encoder ¹
ROMA [5:0]	B23, A23, A22, D20, A21, B21	I6/O3	VDO[5:0]	Digital Data[5:0] for TV/TMDS/LVDS Encoder ¹
ROMA8	C23	I7/O4	VDO8	Digital Data[8] for TV/TMDS/LVDS Encoder ¹
ROMA9	D22	I7/O4	VDO9	Digital Data[9] for TV/TMDS/LVDS Encoder ¹
ROMA10	E21	I7/O4	VDO10	Digital Data[10] for TV/TMDS/LVDS Encoder ¹
ROMA11	F20	I7/O4	VDO11	Digital Data[11] for TV/TMDS/LVDS Encoder ¹
ROMA12	D23	I7/O4	VDO12	Digital Data[12] for TV/TMDS/LVDS Encoder ¹
ROMA13	E22	I7/O4	VDO13	Digital Data[13] for TV/TMDS/LVDS Encoder ¹
ROMA14	F21	I7/O4	VDO14	Digital Data[14] for TV/TMDS/LVDS Encoder ¹
ROMA15	E23	I7/O4	VDO15	Digital Data[15] for TV/TMDS/LVDS Encoder ¹
ROMD3	D18	I6/O3	HSYNC2	HSYNC for TV/TMDS/LVDS Encoder ¹
ROMD2	A18	I6/O3	VSYNC2	VSYNC for TV/TMDS/LVDS Encoder ¹
ROMD1	D17	I6/O3	DCLK2	DCLK for TV/TMDS/LVDS Encoder ¹
ROMD0	A17	I6/O3	BLANK2#	BLANK# for TV/TMDS/LVDS Encoder ¹
FPD[28:25]	C2, B1, A1, D4	I6/O5	TEST[3:0]	Test Selection pins 3-0
FPD[23:0]	C4, D5, A3, B4, A4, A5, A6, C6, D6, A7, B7, D7, A8, D8, A9, C9, D9, A10, B10, D10, A11, D11, A12, C12	I6/O5	VDO[23:0]	Digital data for TV/TMDS/LVDS Encoder ¹
ROMA8	C23	I7/O4	CONFIG[0]	Power-up configuration pin ²
ROMA9	D22	I7/O4	CONFIG[1]	Power-up configuration pin ²
ROMA10	E21	I7/O4	CONFIG[2]	Power-up configuration pin ²
ROMA11	F20	I7/O4	CONFIG[3]	Power-up configuration pin ²
ROMA12	D23	I7/O4	32K/64K ROM	Power-up configuration pin ²
ROMA13	E22	I7/O4	INT/EXT PLL	Power-up configuration pin ²
ROMA14	F21	I7/O4	INT/EXT SCLK	Power-up configuration pin ²
ROMA15	E23	I7/O4	AGP/PCI	Power-up configuration pin ²

- Note:
1. TMDS : Transition Minimized Differential Signaling
LVDS : Low Voltage Differential Signaling
 2. Used for power-on configuration strapping and in TV encoding. For information about enabling the alternate function see the Application Note for each function.
 3. See Pad Description, Section 3.1.12

3.1.14 Pin Assignment (by pin number)

Pin	Signal Name
A1	FPD26
A2	FPD24
A3	FPD21
A4	FPD19
A5	FPD18
A6	FPD17
A7	FPD14
A8	FPD11
A9	FPD9
A10	FPD6
A11	FPD3
A12	FPD1
A13	LP
A14	FP
A15	FPBL
A16	ROMXW#
A17	ROMD0
A18	ROMD2
A19	ROMD4
A20	ROMD6
A21	ROMA1
A22	ROMA3
A23	ROMA4
B1	FPD27
B2	VDD_IO
B3	VDD_R
B4	FPD20
B5	VDD_IO
B6	VDD_R
B7	FPD13
B8	VDD_IO
B9	VDD_R
B10	FPD5
B11	VDD_IO
B12	VDD_R
B13	FPVCC
B14	VDD_R
B15	VDD_IO

Pin	Signal Name
B16	ROMXR#
B17	VDD_R
B18	VDD_IO
B19	VDD_R
B20	VDD_R
B21	ROMA0
B22	VDD_IO
B23	ROMA5
C1	FPD29
C2	FPD28
C3	GND
C4	FPD23
C5	GND_R
C6	FPD16
C7	GND
C8	GND_R
C9	FPD8
C10	GND
C11	GND_R
C12	FPD0
C13	GND_R
C14	GND
C15	TEST#
C16	GND_R
C17	GND
C18	GND_R
C19	GND_R
C20	ROMD7
C21	GND_R
C22	ROMA6
C23	ROMA8
D1	FPD31
D2	FPD30
D3	GND
D4	FPD25
D5	FPD22
D6	FPD15
D7	FPD12
D8	FPD10
D9	FPD7
D10	FPD4

Pin	Signal Name
D11	FPD2
D12	FPVEE
D13	FPCLKO
D14	FPDE
D15	ROMFW#
D16	ROMOE#
D17	ROMD1
D18	ROMD3
D19	ROMD5
D20	ROMA2
D21	GND
D22	ROMA9
D23	ROMA12
E1	FPD35
E2	FPD34
E3	FPD33
E4	FPD32
E20	ROMA7
E21	ROMA10
E22	ROMA13
E23	ROMA15
F1	Reserved26
F2	VDD_IO
F3	Reserved25
F4	Reserved24
F20	ROMA11
F21	ROMA14
F22	VDD_IO
F23	IOTST1
G1	Reserved29
G2	Reserved28
G3	GND
G4	Reserved27
G20	IOTST5
G21	IOTST4
G22	IOTST3
G23	IOTST2
H1	Reserved33
H2	Reserved32
H3	Reserved31
H4	Reserved30

Pin	Signal Name
H20	IOTST8
H21	GND
H22	IOTST7
H23	IOTST6
J1	Reserved1
J2	Reserved35
J3	GND
J4	Reserved34
J20	Reserved21
J21	Reserved22
J22	VDD_IO
J23	Reserved23
K1	Reserved4
K2	VDD_IO
K3	Reserved3
K4	Reserved2
K10	GND
K11	GND
K12	GND
K13	GND
K14	GND
K20	CIO4
K21	CIO5
K22	CIO6
K23	CIO7
L1	Reserved7
L2	Reserved6
L3	GND
L4	Reserved5
L10	GND
L11	GND
L12	GND
L13	GND
L14	GND
L20	ZVUV0
L21	ZVPCLK
L22	CIO2
L23	CIO3
M1	Reserved8
M2	VDD_C

Pin	Signal Name
M3	Reserved9
M4	Reserved10
M10	GND
M11	GND
M12	GND
M13	GND
M14	GND
M20	ZVUV4
M21	ZVUV3
M22	ZVUV2
M23	ZVUV1
N1	Reserved11
N2	Reserved12
N3	GND
N4	Reserved13
N10	GND
N11	GND
N12	GND
N13	GND
N14	GND
N20	ZVUV7
N21	ZVUV6
N22	VDD_C
N23	ZVUV5
P1	Reserved14
P2	VDD_IO
P3	Reserved15
P4	Reserved16
P10	GND
P11	GND
P12	GND
P13	GND
P14	GND
P20	ZVY2
P21	GND
P22	ZVY1
P23	ZVY0
R1	Reserved36
R2	Reserved37
R3	CIO1

Pin	Signal Name
R4	CIO0
R20	ZVY6
R21	ZVY5
R22	ZVY4
R23	ZVY3
T1	DCLK
T2	VDD_C
T3	GND
T4	BLANK#
T20	ZVVSYNC
T21	ZVHREF
T22	VDD_C
T23	ZVY7
U1	VSYNC
U2	HSYNC
U3	AVDD_D
U4	RSET
U20	VDD_P
U21	GND_P
U22	GND
U23	ZVCREF
V1	COMP
V2	AGND_D
V3	GREEN
V4	RED
V20	AGND_P
V21	AVDD_P
V22	AVDD_P
V23	AGND_P
W1	BLUE
W2	AVDD_D
W3	AGND_D
W4	AVDD_S
W20	XCCLKO
W21	XCCLKI
W22	AVDD_P
W23	AGND_P
Y1	AGND_S
Y2	VDD_S
Y3	GND_S

Pin	Signal Name
Y4	CLK
Y5	GNT#
Y6	SBA1
Y7	SBA3
Y8	SBA5
Y9	AD30
Y10	AD26
Y11	AD24
Y12	Reserved18
Y13	AD22
Y14	AD18
Y15	AD16
Y16	TRDY#
Y17	Reserved19
Y18	AD15
Y19	AD13
Y20	AD9
Y21	AD6
Y22	AGPVref
Y23	AD4
AA1	AGPCNTL#
AA2	VDD_IO
AA3	GND
AA4	RST#
AA5	ST1
AA6	GND
AA7	Reserved17
AA8	SBA7
AA9	AD28
AA10	GND
AA11	IDSEL
AA12	C/BE3#
AA13	AD20
AA14	GND
AA15	FRAME#
AA16	STOP#
AA17	PAR
AA18	GND
AA19	AD11
AA20	C/BE0#

Pin	Signal Name
AA21	GND
AA22	VDD_IO
AA23	AD2
AB1	GND
AB2	CLKRUN#
AB3	VDD_C
AB4	ST2
AB5	VDD_IO
AB6	SBA2
AB7	VDD_C
AB8	SBA6
AB9	VDD_IO
AB10	AD27
AB11	VDD_C
AB12	AD23
AB13	VDD_IO
AB14	AD17
AB15	VDD_C
AB16	DEVSEL#
AB17	VDD_IO
AB18	AD12
AB19	AD8
AB20	VDD_C
AB21	Reserved20
AB22	AD3
AB23	AD0
AC1	INTA#
AC2	REQ#
AC3	ST0
AC4	RBF#
AC5	SBA0
AC6	SB_STB
AC7	SBA4
AC8	AD31
AC9	AD29
AC10	AD25
AC11	AD_STB1
AC12	AD21
AC13	AD19
AC14	C/BE2#

Pin	Signal Name
AC15	IRDY#
AC16	C/BE1#
AC17	AD14
AC18	AD10
AC19	M66EN
AC20	AD_STB0
AC21	AD7
AC22	AD5
AC23	AD1

3.1.15 Pin Assignment (by signal name)

Pin	Signal Name
AC20	AD_STB0
AC11	AD_STB1
AB23	AD0
AC23	AD1
AA23	AD2
AB22	AD3
Y23	AD4
AC22	AD5
Y21	AD6
AC21	AD7
AB19	AD8
Y20	AD9
AC18	AD10
AA19	AD11
AB18	AD12
Y19	AD13
AC17	AD14
Y18	AD15
Y15	AD16
AB14	AD17
Y14	AD18
AC13	AD19
AA13	AD20
AC12	AD21
Y13	AD22
AB12	AD23
Y11	AD24
AC10	AD25
Y10	AD26
AB10	AD27
AA9	AD28
AC9	AD29
Y9	AD30
AC8	AD31
V2	AGND_D
W3	AGND_D
V20	AGND_P
V23	AGND_P
W23	AGND_P
Y1	AGND_S

Pin	Signal Name
AA1	AGPCNTL#
Y22	AGPVref
U3	AVDD_D
W2	AVDD_D
V21	AVDD_P
V22	AVDD_P
W22	AVDD_P
W4	AVDD_S
T4	BLANK#
W1	BLUE
AA20	C/BE0#
AC16	C/BE1#
AC14	C/BE2#
AA12	C/BE3#
R4	CIO0
R3	CIO1
L22	CIO2
L23	CIO3
K20	CIO4
K21	CIO5
K22	CIO6
K23	CIO7
Y4	CLK
AB2	CLKRUN#
V1	COMP
T1	DCLK
AB16	DEVSEL#
A14	FP
A15	FPBL
D13	FPCLKO
C12	FPD0
A12	FPD1
D11	FPD2
A11	FPD3
D10	FPD4
B10	FPD5
A10	FPD6
D9	FPD7
C9	FPD8
A9	FPD9
D8	FPD10
A8	FPD11

Pin	Signal Name
D7	FPD12
B7	FPD13
A7	FPD14
D6	FPD15
C6	FPD16
A6	FPD17
A5	FPD18
A4	FPD19
B4	FPD20
A3	FPD21
D5	FPD22
C4	FPD23
A2	FPD24
D4	FPD25
A1	FPD26
B1	FPD27
C2	FPD28
C1	FPD29
D2	FPD30
D1	FPD31
E4	FPD32
E3	FPD33
E2	FPD34
E1	FPD35
D14	FPDE
B13	FPVCC
D12	FPVEE
AA15	FRAME#
C3	GND
C7	GND
C10	GND
C14	GND
C17	GND
D3	GND
D21	GND
G3	GND
H21	GND
J3	GND
K10	GND
K11	GND
K12	GND
K13	GND

Pin	Signal Name
K14	GND
L3	GND
L10	GND
L11	GND
L12	GND
L13	GND
L14	GND
M10	GND
M11	GND
M12	GND
M13	GND
M14	GND
N3	GND
N10	GND
N11	GND
N12	GND
N13	GND
N14	GND
P10	GND
P11	GND
P12	GND
P13	GND
P14	GND
P21	GND
T3	GND
U22	GND
AA3	GND
AA6	GND
AA10	GND
AA14	GND
AA18	GND
AA21	GND
AB1	GND
U21	GND_P
C5	GND_R
C8	GND_R
C11	GND_R
C13	GND_R
C16	GND_R
C18	GND_R
C19	GND_R
C21	GND_R

Pin	Signal Name
Y3	GND S
Y5	GNT#
V3	GREEN
U2	HSYNC
AA11	IDSEL
AC1	INTA#
F23	IOTST1
G23	IOTST2
G22	IOTST3
G21	IOTST4
G20	IOTST5
H23	IOTST6
H22	IOTST7
H20	IOTST8
AC15	IRDY#
A13	LP
AC19	M66EN
AA17	PAR
AC4	RBF#
V4	RED
AC2	REQ#
J1	Reserved1
K4	Reserved2
K3	Reserved3
K1	Reserved4
L4	Reserved5
L2	Reserved6
L1	Reserved7
M1	Reserved8
M3	Reserved9
M4	Reserved10
N1	Reserved11
N2	Reserved12
N4	Reserved13
P1	Reserved14
P3	Reserved15
P4	Reserved16
AA7	Reserved17
Y12	Reserved18
Y17	Reserved19
AB21	Reserved20
J20	Reserved21

Pin	Signal Name
J21	Reserved22
J23	Reserved23
F4	Reserved24
F3	Reserved25
F1	Reserved26
G4	Reserved27
G2	Reserved28
G1	Reserved29
H4	Reserved30
H3	Reserved31
H2	Reserved32
H1	Reserved33
J4	Reserved34
J2	Reserved35
R1	Reserved36
R2	Reserved37
B21	ROMA0
A21	ROMA1
D20	ROMA2
A22	ROMA3
A23	ROMA4
B23	ROMA5
C22	ROMA6
E20	ROMA7
C23	ROMA8
D22	ROMA9
E21	ROMA10
F20	ROMA11
D23	ROMA12
E22	ROMA13
F21	ROMA14
E23	ROMA15
A17	ROMD0
D17	ROMD1
A18	ROMD2
D18	ROMD3
A19	ROMD4
D19	ROMD5
A20	ROMD6
C20	ROMD7
D15	ROMFW#
D16	ROMOE#

Pin	Signal Name
B16	ROMXR#
A16	ROMXW#
U4	RSET
AA4	RST#
AC6	SB_STB
AC5	SBA0
Y6	SBA1
AB6	SBA2
Y7	SBA3
AC7	SBA4
Y8	SBA5
AB8	SBA6
AA8	SBA7
AC3	ST0
AA5	ST1
AB4	ST2
AA16	STOP#
C15	TEST#
Y16	TRDY#
M2	VDD_C
N22	VDD_C
T2	VDD_C
T22	VDD_C
AB3	VDD_C
AB7	VDD_C
AB11	VDD_C
AB15	VDD_C
AB20	VDD_C
B2	VDD_IO
B5	VDD_IO
B8	VDD_IO
B11	VDD_IO
B15	VDD_IO
B18	VDD_IO
B22	VDD_IO
F2	VDD_IO
F22	VDD_IO
J22	VDD_IO
K2	VDD_IO
P2	VDD_IO
AA2	VDD_IO
AA22	VDD_IO

Pin	Signal Name
AB5	VDD_IO
AB9	VDD_IO
AB13	VDD_IO
AB17	VDD_IO
U20	VDD_P
B3	VDD_R
B6	VDD_R
B9	VDD_R
B12	VDD_R
B14	VDD_R
B17	VDD_R
B19	VDD_R
B20	VDD_R
Y2	VDD_S
U1	VSYNCR
W21	XCLKI
W20	XCLKO
U23	ZVCREFR
T21	ZVHREF
L21	ZVPCCLK
L20	ZVUV0
M23	ZVUV1
M22	ZVUV2
M21	ZVUV3
M20	ZVUV4
N23	ZVUV5
N21	ZVUV6
N20	ZVUV7
T20	ZVVSYNCR
P23	ZVY0
P22	ZVY1
P20	ZVY2
R23	ZVY3
R22	ZVY4
R21	ZVY5
R20	ZVY6
T23	ZVY7

Figure 3-1. Top View

	1	2	3	4	5	6	7	8	9	10	11
A	301 FPD26	298 FPD24	294 FPD21	291 FPD19	289 FPD18	288 FPD17	283 FPD14	279 FPD11	276 FPD9	271 FPD6	267 FPD3
B	303 FPD27	302 VDD_IO	299 VDD_R	292 FPD20	290 VDD_IO	287 VDD_R	282 FPD13	278 VDD_IO	275 VDD_R	270 FPD5	266 VDD_IO
C	1 FPD29	304 FPD28	296 GND	297 FPD23	293 GND_R	286 FPD16	284 GND	281 GND_R	274 FPD8	272 GND	269 GND_R
D	4 FPD31	3 FPD30	2 GND	300 FPD25	295 FPD22	285 FPD15	280 FPD12	277 FPD10	273 FPD7	268 FPD4	265 FPD2
E	8 FPD35	7 FPD34	6 FPD33	5 FPD32							
F	11 Reserved26	12 VDD_IO	10 Reserved25	9 Reserved24							
G	15 Reserved29	14 Reserved28	16 GND	13 Reserved27							
H	20 Reserved33	19 Reserved32	18 Reserved31	17 Reserved30							
J	24 Reserved1	23 Reserved35	22 GND	21 Reserved34							
K	28 Reserved4	26 VDD_IO	27 Reserved3	25 Reserved2							
L	32 Reserved7	30 Reserved6	31 GND	29 Reserved5							
M	33 Reserved8	35 VDD_C	34 Reserved9	36 Reserved10							
N	37 Reserved11	38 Reserved12	39 GND	40 Reserved13							
P	41 Reserved14	44 VDD_IO	42 Reserved15	43 Reserved16							
R	45 Reserved36	46 Reserved37	47 CIO1	48 CIO0							
T	50 DCLK	49 VDD_C	52 GND	51 BLANK#							
U	53 VSYNC	54 HSYNC	55 AVDD_D	56 RSET							
V	57 COMP	58 AGND_D	60 GREEN	59 RED							
W	61 BLUE	62 AVDD_D	63 AGND_D	64 AVDD_S							
Y	65 AGND_S	67 VDD_S	68 GND_S	71 CLK	79 GNT#	85 SBA1	89 SBA3	93 SBA5	97 AD30	101 AD26	105 AD24
AA	66 AGPCNTL#	70 VDD_IO	73 GND	76 RST#	83 ST1	87 GND	91 Reserved17	94 SBA7	99 AD28	104 GND	106 IDSEL
AB	69 GND	74 CLKRUN#	77 VDD_C	80 ST2	81 VDD_IO	86 SBA2	90 VDD_C	95 SBA6	98 VDD_IO	102 AD27	107 VDD_C
AC	72 INTA#	75 REQ#	78 ST0	82 RBF#	84 SBA0	88 SB_STB	92 SBA4	96 AD31	100 AD29	103 AD25	108 AD_STB1
	1	2	3	4	5	6	7	8	9	10	11

GND	GND

12	13	14	15	16	17	18	19	20	21	22	23	
263	258	253	250	246	241	238	235	232	227	224	223	A
FPD1	LP	FP	FPBL	ROMXW#	ROMD0	ROMD2	ROMD4	ROMD6	ROMA1	ROMA3	ROMA4	
264	256	257	254	244	245	242	236	231	229	228	222	B
VDD_R	FPVCC	VDD_R	VDD_IO	ROMXR#	VDD_R	VDD_IO	VDD_R	VDD_R	ROMA0	VDD_IO	ROMA5	
261	262	260	249	251	248	239	234	230	225	221	218	C
FPD0	GND_R	GND	TEST#	GND_R	GND	GND_R	GND_R	ROMD7	GND_R	ROMA6	ROMA8	
259	255	252	247	243	240	237	233	226	219	217	214	D
FPVEE	FPCLKO	FPDE	ROMFW#	ROMOE#	ROMD1	ROMD3	ROMD5	ROMA2	GND	ROMA9	ROMA12	
								220	216	213	210	E
								ROMA7	ROMA10	ROMA13	ROMA15	
								215	211	212	209	F
								ROMA11	ROMA14	VDD_IO	IOTST1	
								205	208	207	208	G
								IOTST5	IOTST4	IOTST3	IOTST2	
								201	203	202	204	H
								IOTST8	GND	IOTST7	IOTST6	
								197	199	198	200	J
								Reserved21	Reserved22	VDD_IO	Reserved23	
								193	194	195	196	K
								CIO4	CIO5	CIO6	CIO7	
								189	190	191	192	L
								ZVUV0	ZVPCLK	CIO2	CIO3	
								185	186	187	188	M
								ZVUV4	ZVUV3	ZVUV2	ZVUV1	
								181	183	182	184	N
								ZVUV7	ZVUV6	VDD_C	ZVUV5	
								177	178	179	180	P
								ZVY2	GND	ZVY1	ZVY0	
								173	174	175	176	R
								ZVY6	ZVY5	ZVY4	ZVY3	
								169	170	171	172	T
								ZVVSYN	ZVHREF	VDD_C	ZVY7	
								165	166	167	168	U
								VDD_P	GND_P	GND	ZVCREF	
								163	164	162	161	V
								AGND_P	AVDD_P	AVDD_P	AGND_P	
								160	159	158	157	W
								XCLKO	XCLKI	AVDD_P	AGND_P	
109	116	120	123	128	134	137	142	150	155	156	154	Y
Reserved18	AD22	AD18	AD16	TRDY#	Reserved19	AD15	AD13	AD9	AD6	AGPvref	AD4	
110	115	119	122	127	131	135	141	146	147	152	152	AA
C/BE3#	AD20	GND	FRAME#	STOP#	PAR	GND	AD11	C/BE0#	GND	VDD_IO	AD2	
111	114	118	125	126	132	133	139	143	145	149	151	AB
AD23	VDD_IO	AD17	VDD_C	DEVSEL#	VDD_IO	AD12	AD8	VDD_C	Reserved20	AD3	AD0	
112	113	117	121	124	129	130	136	138	140	144	148	AC
AD21	AD19	C/BE2#	IRDY#	C/BE1#	AD14	AD10	M66EN	AD_STB0	AD7	AD5	AD1	
12	13	14	15	16	17	18	19	20	21	22	23	

GND	GND	GND

3.2 Electrical Specifications

3.2.1 Maximum DC Specifications

Table 3-1. Absolute Maximum Specifications

Symbol	Parameter	Minimum	Maximum	Unit	Notes
VDD_IO	Power Supply, I/O		5.5	V	1
VDD_C, VDD_S, VDD_P, VDD_R, AVDD_S, ADD_D, AVDD_P	Power Supply, all other		5.5	V	1
V _{PIN}	Voltage on all signal pins	-0.5	VDD_IO+0.5	V	1
I _{LU}	Injection Current (latch-up)		100	mA	1
P _{AM}	Operating Power Dissipation		4.0	W	1
T _{STG}	Storage Temperature	-65	150	°C	1

Notes:

1. Exposure at or above the absolute maximum conditions may damage devices and cause system malfunction.

3.2.2 Normal Operating DC Specifications

Table 3-2. Normal Operating Specifications

Symbol	Parameter	Minimum	Typical	Maximum	Unit
VDD_IO	Supply Voltage, I/O	3.15	3.3	3.45	V
VDD_C, VDD_R, VDD_S, VDD_P, AVDD_D, AVDD_P, AVDD_S	Supply Voltage, all other	2.375	2.5	2.625	V
AGPVREF	AGP Bus Reference Voltage		1.4		V
C _{IN}	Input Capacitance		10		pF
C _{OUT}	Output Capacitance		10		pF
T _A	Ambient Temperature	0	25	70	°C

3.2.3 DC Characteristics
Table 3-3. DC Characteristics

Symbol	Parameter	Minimum	Typical	Maximum	Unit
V _{IH}	Input High Voltage	0.65xVDD_IO			V
V _{IL}	Input Low Voltage		0	0.35xVDD_IO	V
I _{IH}	Input High Current		4		μA
I _{IL}	Input Low Current		4		μA
I _{OZ}	Tri-state Output Leakage Current		4		μA
V _{OH}	Output High Voltage	2.4			V
V _{OL}	Output Low Voltage			0.4	V
IDDO3.3V	Supply Current - Operating		35		mA
IDDO2.5V	Supply Current - Operating		650		mA
IDDC3.3V	Supply Current - CRT only		25		mA
IDDC2.5V	Supply Current - CRT only		500		mA
IDDP3.3V	Supply Current - Flat Panel only		35		mA
IDDP2.5V	Supply Current - Flat Panel only		500		mA
IDDS3.3V	Supply Current - Suspend Mode		10		mA
IDDS2.5V	Supply Current - Suspend Mode		10		mA

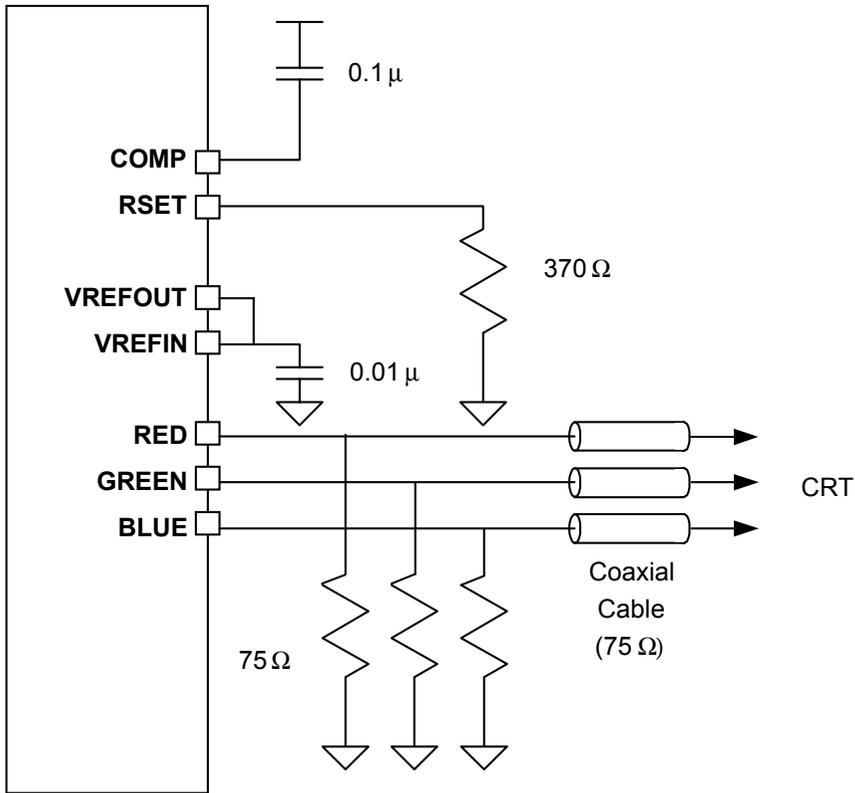


Figure 3-2. DAC Test Set-up

Table 3-4. DAC Characteristics

Symbol	Parameter	Minimum	Typical	Maximum	Unit	Condition
	DAC Output Speed			230	MHz	
IOmax	Full Scale Current		16		mA	
V(IO)	Output Voltage Range		1.0		V	
	DAC Resolution		8		bits	
INL	Integral non-linearity error		0.5	1	LSB	AVDD_D = 2.5V
DNL	Differential non-linearity error		0.5	1	LSB	AVDD_D = 2.5V
	Gain error			5	%	
	DAC to DAC matching		1	2.5	%	

Note: AVDD_D = 2.5V; RL = 37.5Ω, CL = 10pF; RSET = 370Ω; TEMP = 75°C

3.2.4 AC Timing

3.2.4.1 AGP Timing

AGP timings are specified by two sets of parameters, one corresponding to the AGP 1x operation and the second corresponding to the optional AGP 2x transfer mode operation. The AGP 2x specifications are in addition to the AGP 1x specifications. The AGP 1x specifications still apply to all outer loop control signals during the AGP 2x operations. See below for the 1x and 2x timing diagrams.

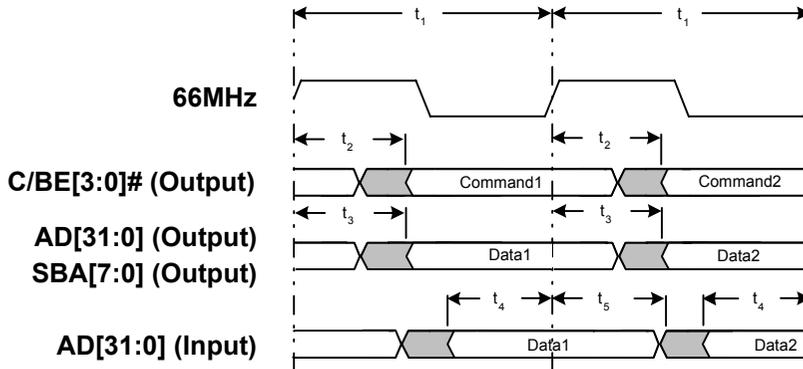


Figure 3-3. AGP 1x Timing

Table 3-5. AGP 1x AC Timing Parameters

Symbol	Parameter	Min	Max	Units	Notes
Clock:					
t_1	CLK cycle time	15	30	ns	
Output Signals:					
t_2	CLK to control signal valid delay	10	5.5	ns	1
t_3	CLK to data valid delay	10	6.0	ns	1
	Output slew rate	1.5	4	V/ns	3
Input Signals:					
t_4	Data setup time to CLK	5.5		ns	
t_5	Data hold time to CLK	0		ns	

Notes:

1. Test Conditions: Output delays into a capacitive load of 10 pF.
2. In AGP 2x mode, t_3 , t_4 and t_5 values for the AD, C/BE# and SBA signals are superseded by the A.G.P. parameters t_1 , t_2 , t_5 and t_6 in Figure 3.
3. The output slew rate applies to all output signals.

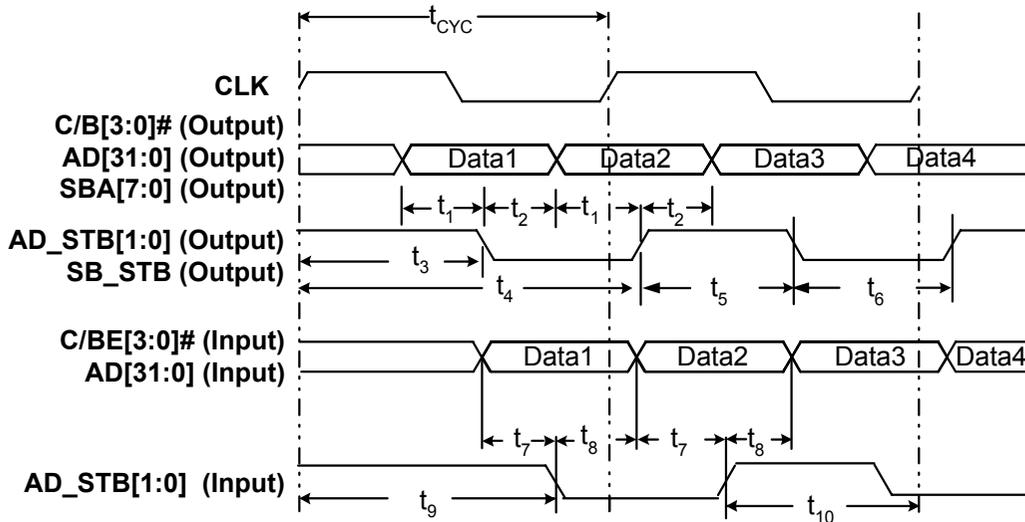


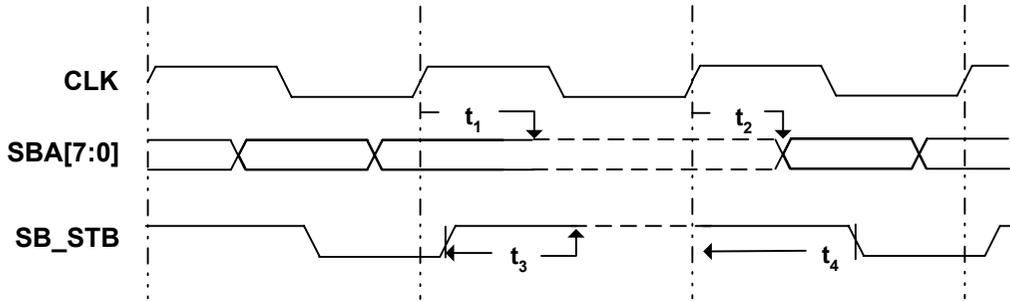
Figure 3-4. AGP 2x Timing

Table 3-7. AGP 2x Timing Parameters

Symbol	Parameter	Min	Max	Units	Notes
Output Signals:					
t_1	Output Data valid before strobe	1.7		ns	1,2,3
t_2	Output Data valid after strobe	1.9		ns	1,2,3
t_3	CLK to output strobe falling	2	12	ns	
t_4	CLK to output strobe rising		20	ns	
t_5	Strobe width high	5.0		ns	
t_6	Strobe width low	5.0		ns	
t_7	Input Data to strobe setup time	1		ns	1,2
t_8	Strobe to input data hold time	1		ns	1,2
t_9	Input strobe hold time from CLK	1		ns	
t_{10}	Input strobe setup time to CLK	6		ns	
t_{cyc}	CLK cycle time	15	30	ns	

Notes:

1. For C/BE[3:2]# and AD[31:16], use AS_STB1 as the reference signal for timing.
2. For C/BE[1:0]# and AD[15:0], use AS_STB0 as the reference signal for timing.
3. For SBA[7:0], use SB_STB as the reference signal for timing.


Figure 3-5. Strobe/Data Turnaround Timing
Table 3-8. Strobe/Data Turnaround Timing Parameters

Symbol	Parameter	Min	Max	Units	Notes
t_1	Active to Float Delay	1	12	ns	
t_2	Float to Active Delay	-1	9	ns	
t_3	Strobe rising edge to strobe float delay	6	10	ns	
t_4	Strobe active to strobe falling edge setup	6	10	ns	

3.2.4.2 PCI Timing

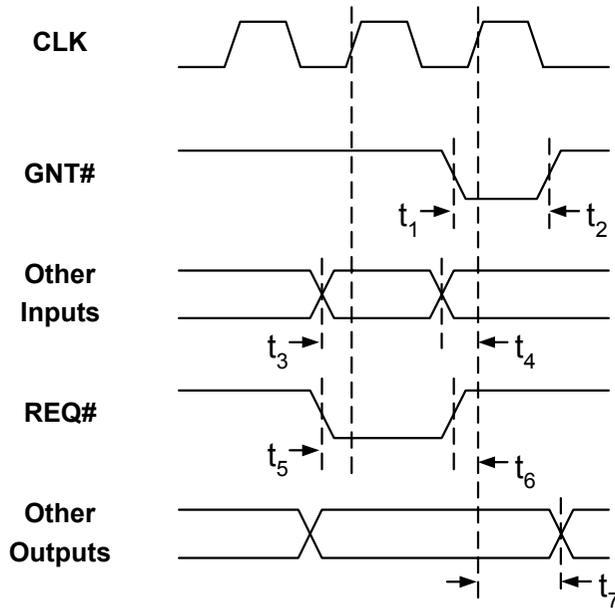
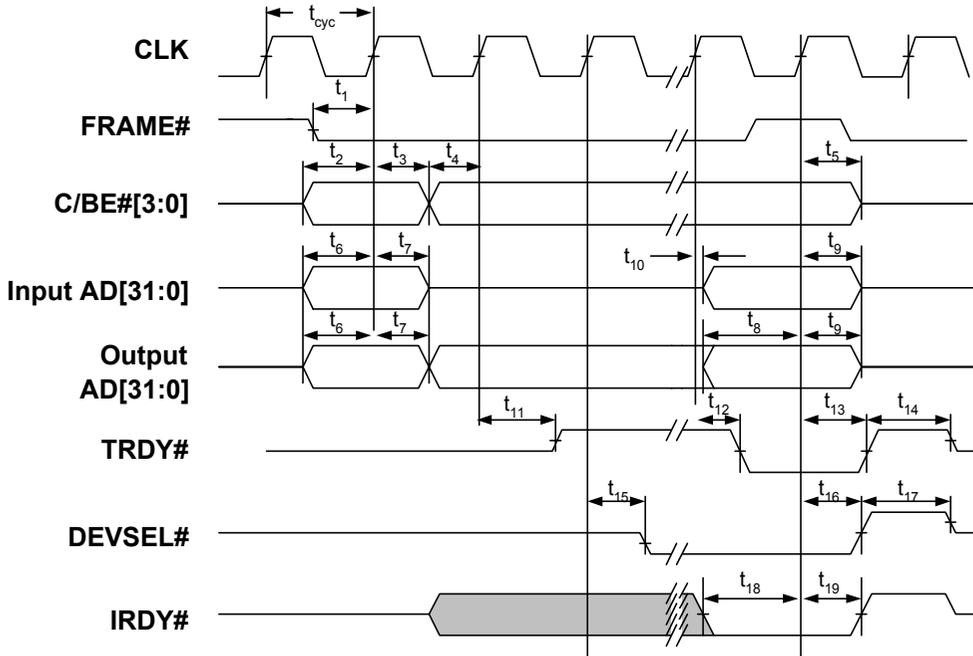


Figure 3-6. PCI Timing

Table 3-9. PCI Timing

Symbol	Parameter	Minimum	Maximum	Units
t_1	Setup time of GNT# to CLK	10		ns
t_2	Hold time of GNT# to CLK	0		ns
t_3	Setup time Other Inputs	7		ns
t_4	Hold time of other Inputs	0		ns
t_5	Setup time REQ#	12		ns
t_6	Hold time REQ#	0		ns
t_7	Setup time Other Output	2		ns


Figure 3-7. PCI Timing
Table 3-10. PCI Timing

Symbol	Parameter	Minimum	Maximum	Units
t_1	FRAME# Setup to CLK	7		ns
t_2	C/BE#[3:0] (Bus CMD) setup to CLK	7		ns
t_3	C/BE#[31:0] (Bus CMD) hold from CLK	0		ns
t_4	C/BE#[3:0] (Byte Enable) setup to CLK	7		ns
t_5	C/BE#[3:0] (Byte Enable) hold from CLK	0		ns
t_6	AD[31:0] (Address) setup to CLK	7		ns
t_7	AD[31:0] (Address) hold from CLK	0		ns
t_8	AD[31:0] (Data) setup to CLK	7		ns
t_9	AD[31:0] (Data) hold from CLK	0		ns
t_{10}	AD[31:0] (Data) valid from CLK	2	11	ns
t_{11}	TRDY# High Z to high from CLK	1		t_{cyc}
t_{12}	TRDY# active from CLK	2	11	ns
t_{13}	TRDY# inactive from CLK	2	11	ns
t_{14}	TRDY# high from before High Z	1		t_{cyc}
t_{15}	DEVSEL# active from CLK	2		ns
t_{16}	DEVSEL# inactive from CLK	2	11	ns
t_{17}	DEVSEL# high before High Z	1		t_{cyc}
t_{18}	IRDY# setup to CLK	7		ns
t_{19}	IRDY# hold from CLK	0		ns

3.2.4.3 PCI Clock Timing

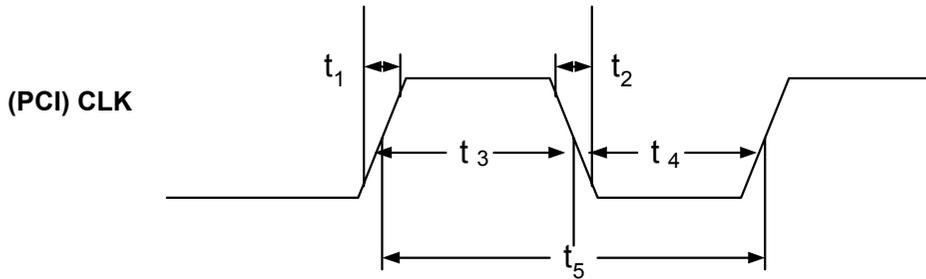
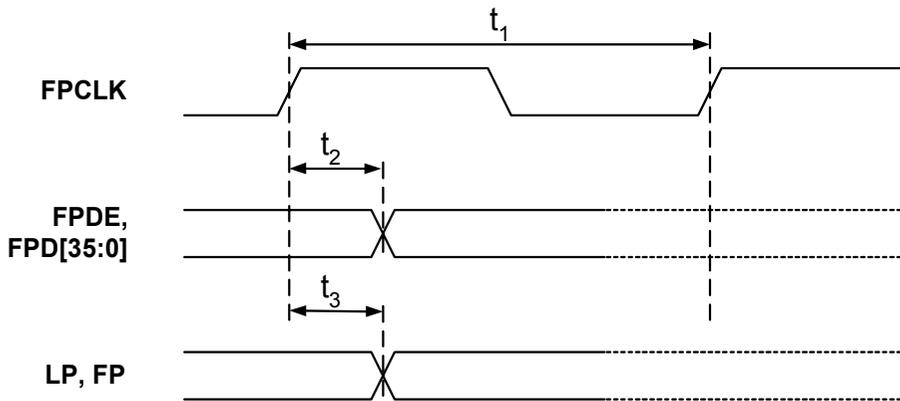


Figure 3-8. PCI Clock Timing

Table 3-11. PCI Clock Timing

Symbol	Parameter	Minimum	Maximum	Units
t_1	Rise time (0.3 VDD_IO to 0.5VDD_IO)	0.16	0.69	ns
t_2	Fall time (90% to 10%)	0.16	0.69	ns
t_3	High pulse width	11		ns
t_4	Low pulse width	11		ns
t_5	Period	30		ns

3.2.4.4 Panel Interface

Figure 3-9. Timing for Flat Panels
Table 3-12. Timing Values for Flat Panels

Symbol	Parameters	Min	Max	Units
t_1	FPCLK cycle time	7	100	MHz
t_2	FPDE and FPD[35:0] output valid delay	5	10	ns
t_3	LP and FP output valid delay	5	10	ns

Note: Polarity of FPCLK, LP, FP and FPDATA are all register programmable. This figure shows them all in normal (non-inverted) setting.

3.2.4.5 BIOS ROM Interface

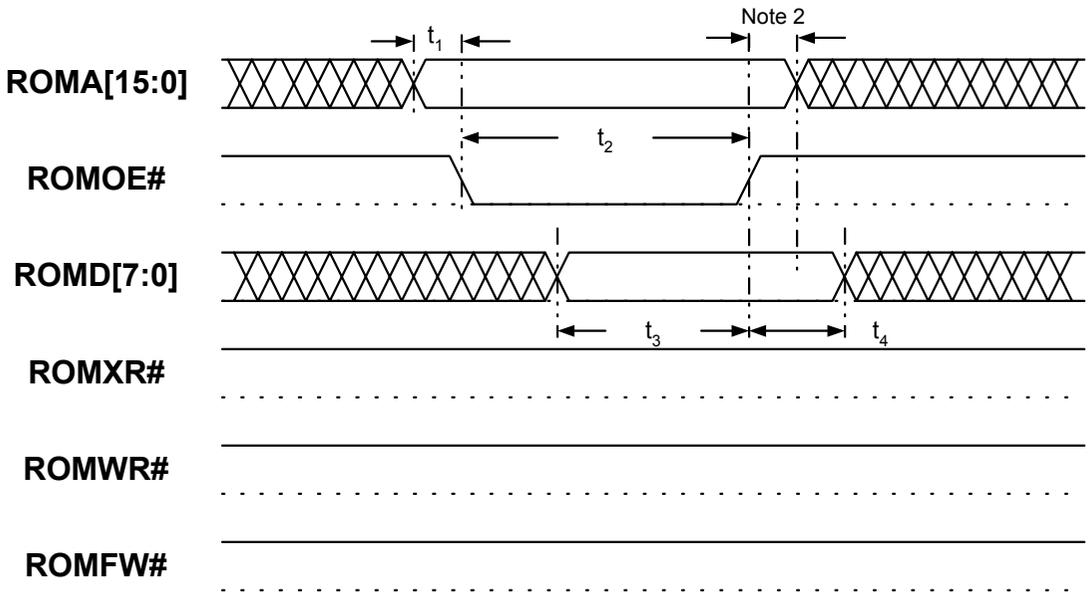
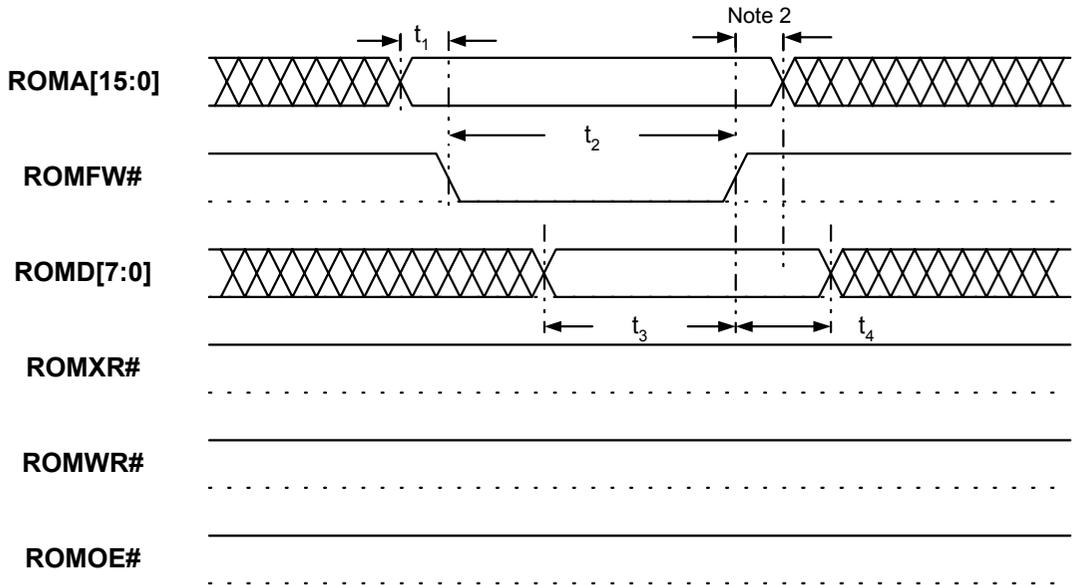


Figure 3-10. BIOS ROM Read Cycle Timing

Table 3-13. BIOS ROM Read Cycle Timing Parameters

Symbol	Parameters	Min	Max	Units	Note
t_1	Address setup time	1	2	AGP/PCI CLK	1
t_2	Output Enable Width	5	12	AGP/PCI CLK	1
t_3	Data to Output Enable Setup Time	10		ns	
t_4	Data to Output Enable Hold Time	10		ns	

- Note: 1. Values are register-programmable.
 2. ROMA to ROMOE# hold time is fixed in one AGP/PCI bus clock cycle


Figure 3-11. BIOS ROM Write Cycle Timing
Table 3-14. BIOS ROM Write Cycle Timing Parameters

Symbol	Parameters	Min	Max	Units	Note
t_1	Address setup time	1	2	AGP/PCI CLK	1
t_2	Flash Write Width	5	12	AGP/PCI CLK	1
t_3	Data to Flash Write Setup Time	5		ns	
t_4	Data to Flash Write Hold Time	10		ns	

- Note:
1. Values are register-programmable.
 2. ROMA to ROMFW# hold time is fixed in one AGP/PCI bus clock cycle

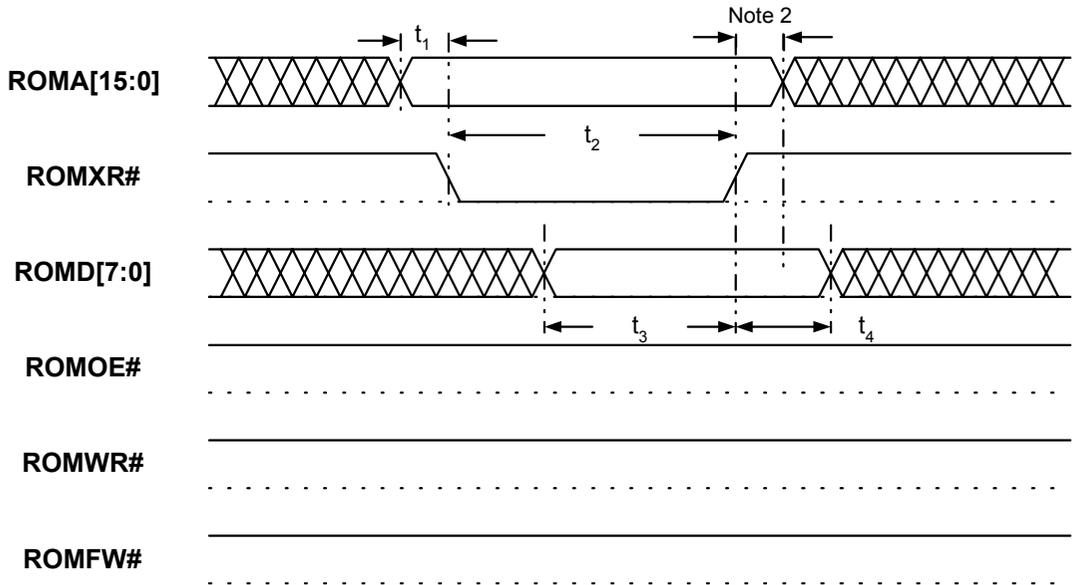
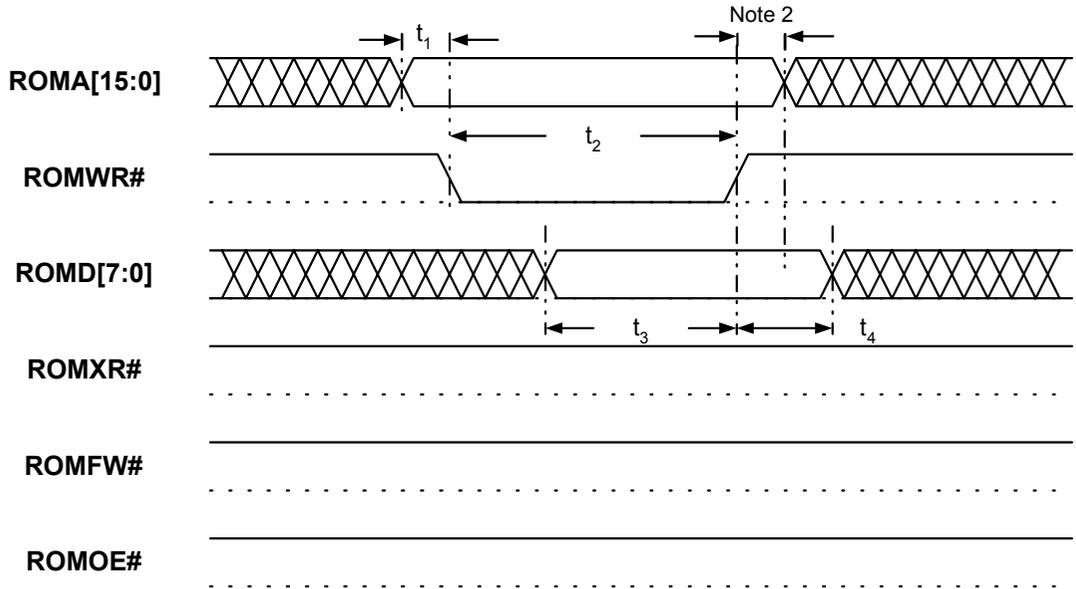


Figure 3-12. Peripheral I/O Read Cycle Timing

Table 3-15. Peripheral I/O Read Cycle Timing Parameters

Symbol	Parameters	Min	Max	Units	Note
t_1	Address setup width	1	2	AGP/PCI CLK	1
t_2	Read Command Width	5	12	AGP/PCI CLK	1
t_3	Data to Read Command Setup Time	10		ns	
t_4	Data to Read Command Hold Time	5		ns	

- Note:
1. Values are register-programmable.
 2. ROMXR# to ROMD[7:0] hold time is fixed in one AGP/PCI bus clock cycle


Figure 3-13. Peripheral I/O Write Cycle Timing
Table 3-16. Peripheral I/O Write Cycle Timing Parameters

Symbol	Parameters	Min	Max	Units	Note
t_1	Address setup time	1	2	AGP/PCI CLK	1
t_2	Write Command Width	5	12	AGP/PCI CLK	1
t_3	Data to Write Command Setup Time	5		AGP/PCI CLK	
t_4	Data to Write Command Hold Time	10		AGP/PCI CLK	

Note: 1.Values are register-programmable.
 2.ROMA to ROMWR# hold time is fixed in one AGP/PCI bus clock cycle

3.2.4.6 ZV Port

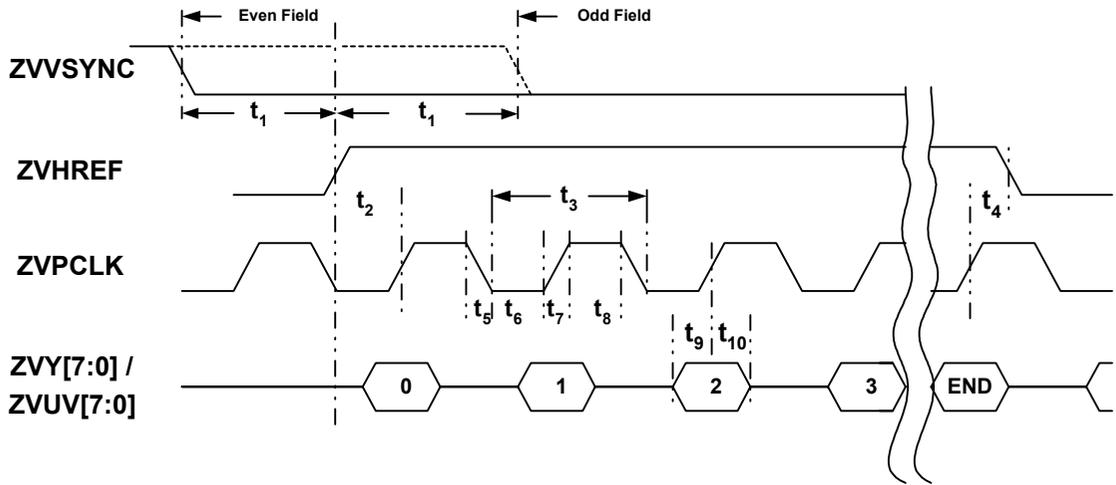
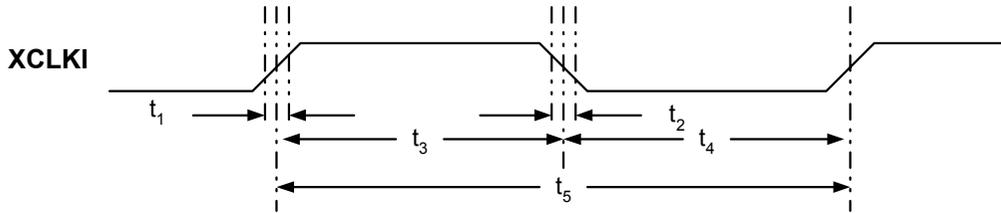


Figure 3-14. ZV Port Timing

Table 3-17. ZV Port Timing Parameters

Symbol	Parameter	Minimum	Maximum	Units
t_1	VSYNC setup time / hold time to HREF	1.5		ns
t_2	HREF setup time	1.5	2.0	ns
t_3	ZVPCLK cycle time	33.0		ns
t_4	HREF hold time	1.5	2.0	ns
t_5	ZVPCLK fall time			ns
t_6	ZVPCLK low time			ns
t_7	ZVPCLK rise time			ns
t_8	ZVPCLK high time			ns
t_9	Y[7::0] / UV[7::0]/HREF setup time	2.0	3.0	ns
t_{10}	Y[7::0] / UV[7::0]/HREF hold time	2.0	3.0	ns

3.2.4.7 Reference Clock

Figure 3-15. PLL Reference Clock
Table 3-18. PLL Reference Clock

Symbol	Parameter	Minimum	Maximum	Units
t_1	Rise Time	0.1	0.5	ns
t_2	Fall Time	0.1	0.5	ns
t_3	High period	45		% of t_5
t_4	Low period	45		% of t_5
t_5	Period	25	500	ns

Note: The sum of the pull-in time and the locking time of the PLL is 0.5 ms max.

3.2.4.8 Reset

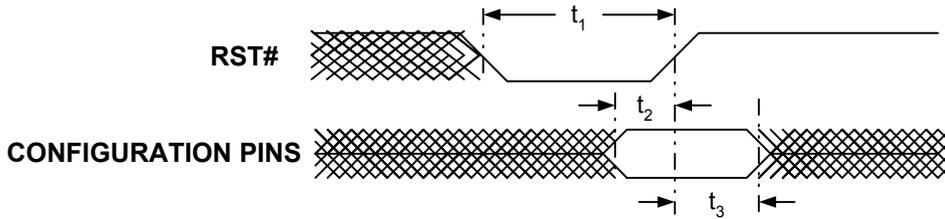
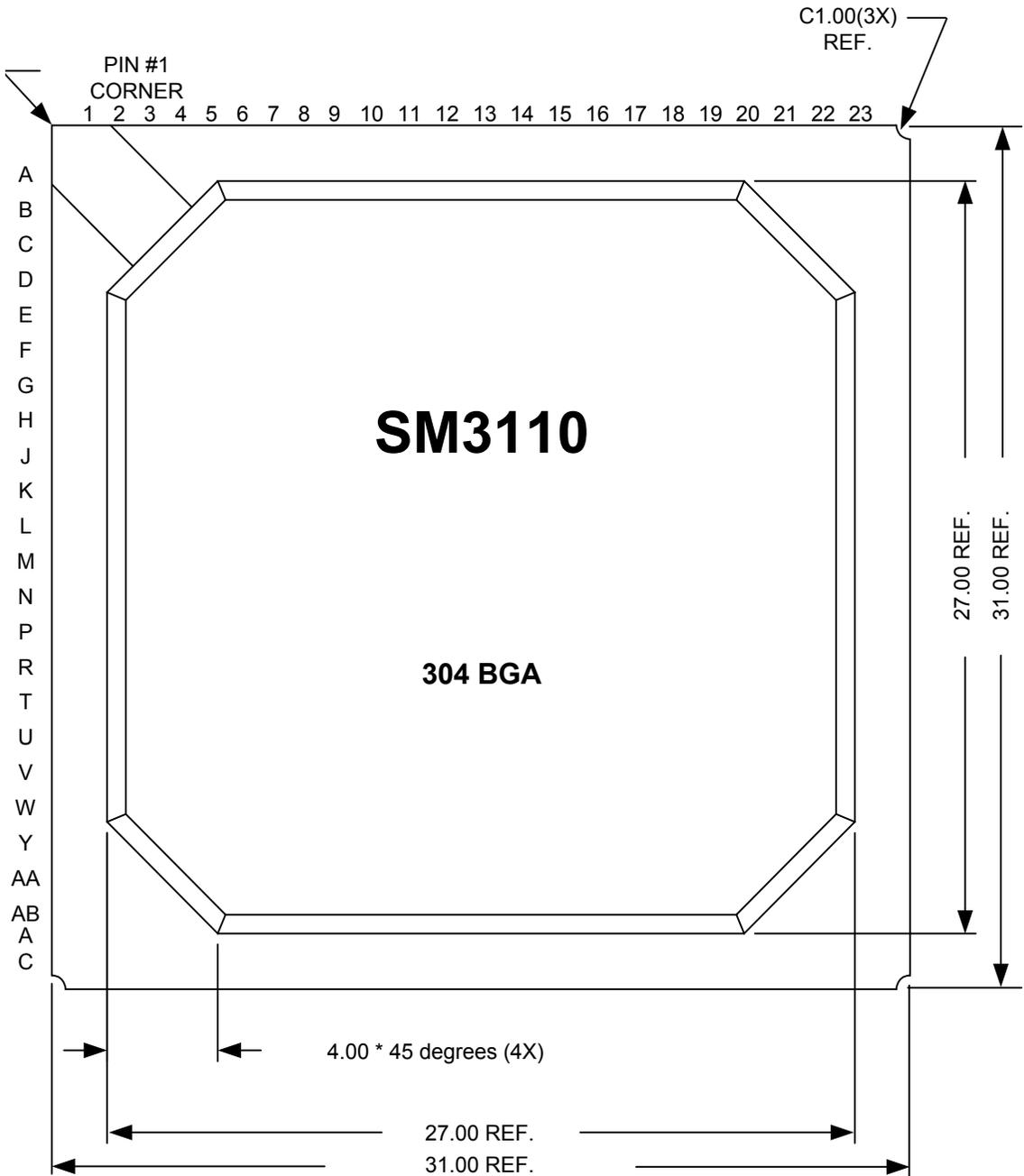


Figure 3-16. Reset Timing

Table 3-19. Reset Timing

Symbol	Parameter	Minimum	Maximum	Units
t_1	RST# active	60		MCLK
t_2	Configuration data setup time	10		ns
t_3	Configuration data hold time	10		ns

3.3 Mechanical Specifications



All dimensions in millimeters (mm) unless otherwise noted.

Figure 3-17. SM3110 Top View

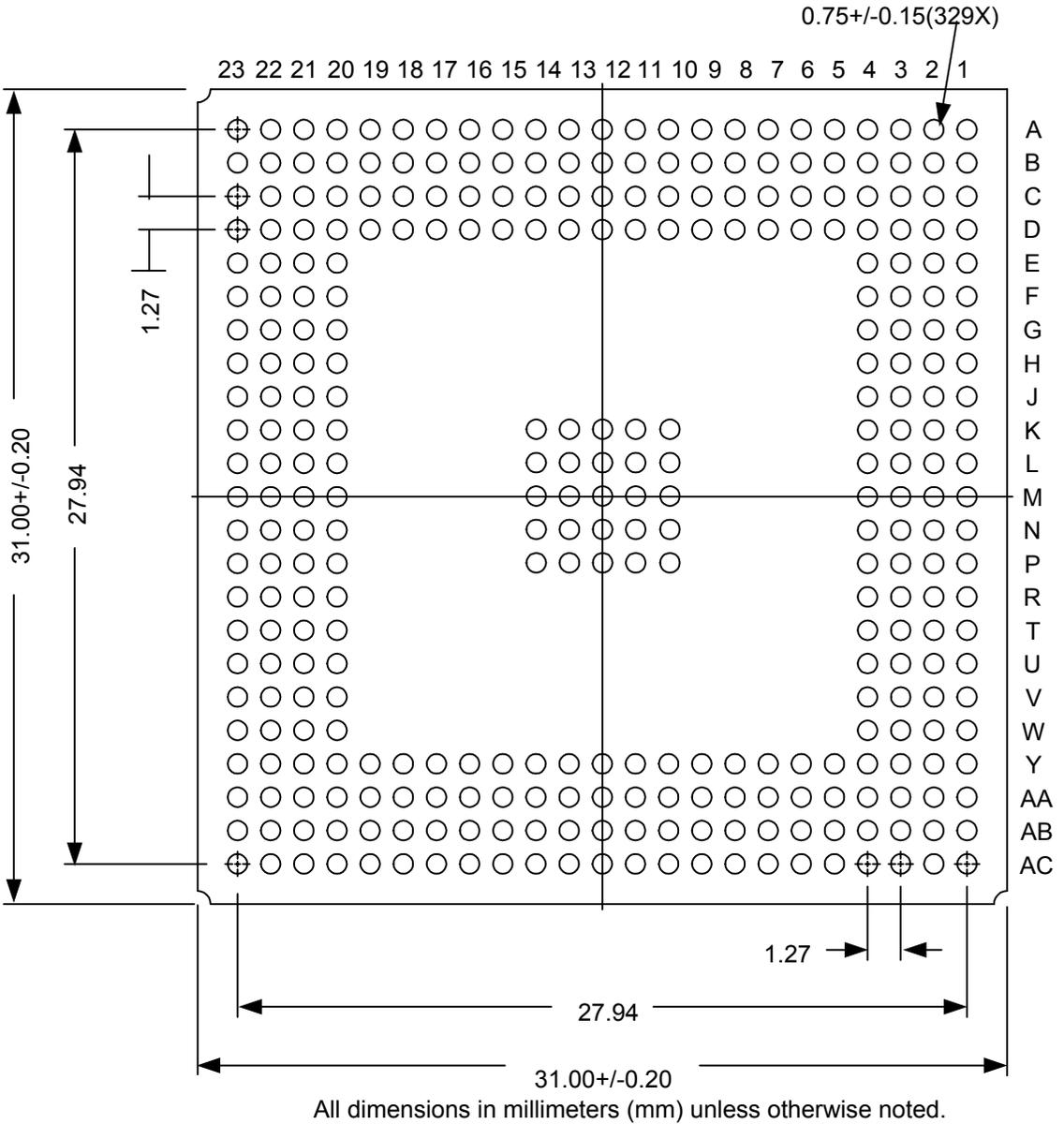


Figure 3-18. SM3110 Bottom View

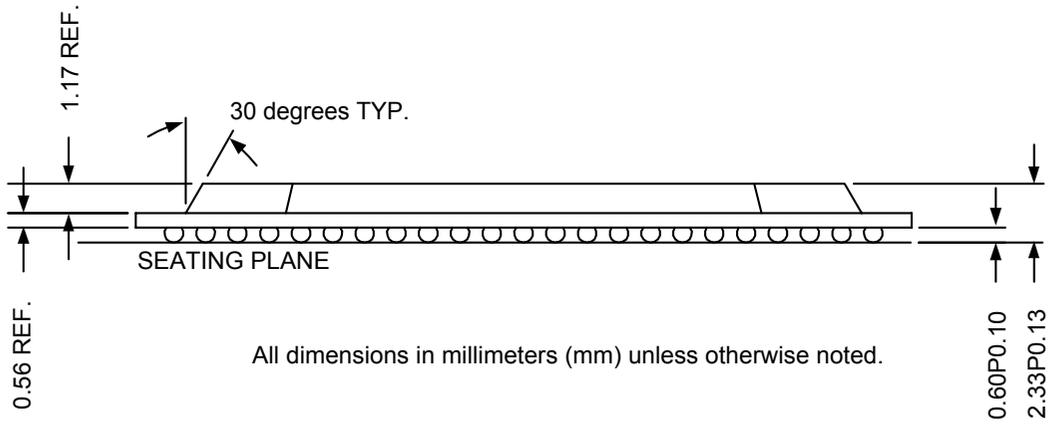
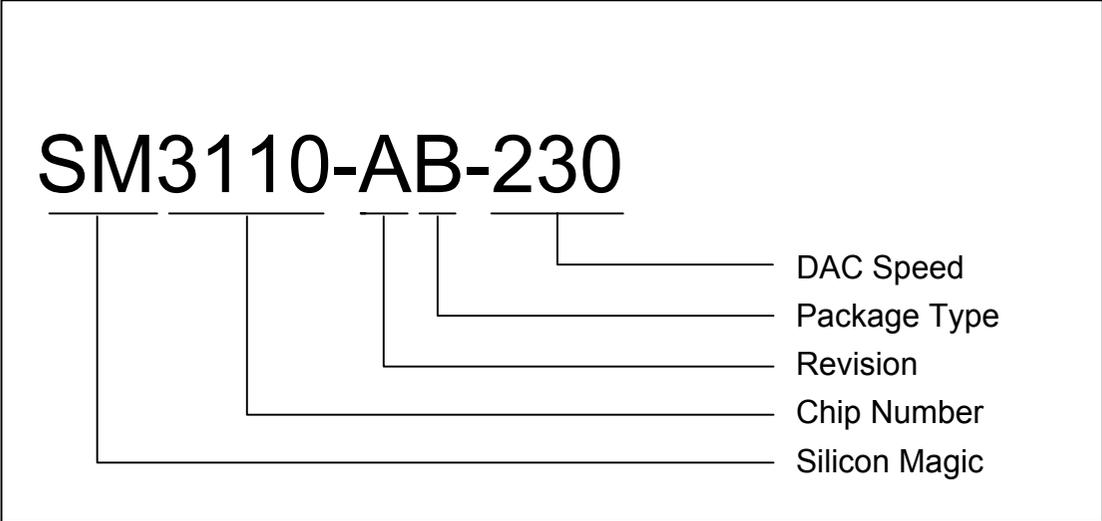


Figure 3-19. SM3110 Side View

3.4 Ordering Information



4 Register Summary



4.1 Control Address Space Overview

All of the functions and data areas of the SM3110 are accessible through memory-mapped registers. This chapter describes each of the registers defined for setting up, writing and reading the SM3110 graphics chip.

All registers associated with the SM3110 are accessible in one or more address spaces. As shown in Figure 4-1, the 32MB Local Memory contains a 256KB region for memory-mapped resources. This is further divided into four(4) 64KB regions, the upper-most of which is the Control Address Space. The registers discussed in this section are all mapped within the Control Address Space. (For more information on address mapping, see Section 6, Programmer's Reference.)

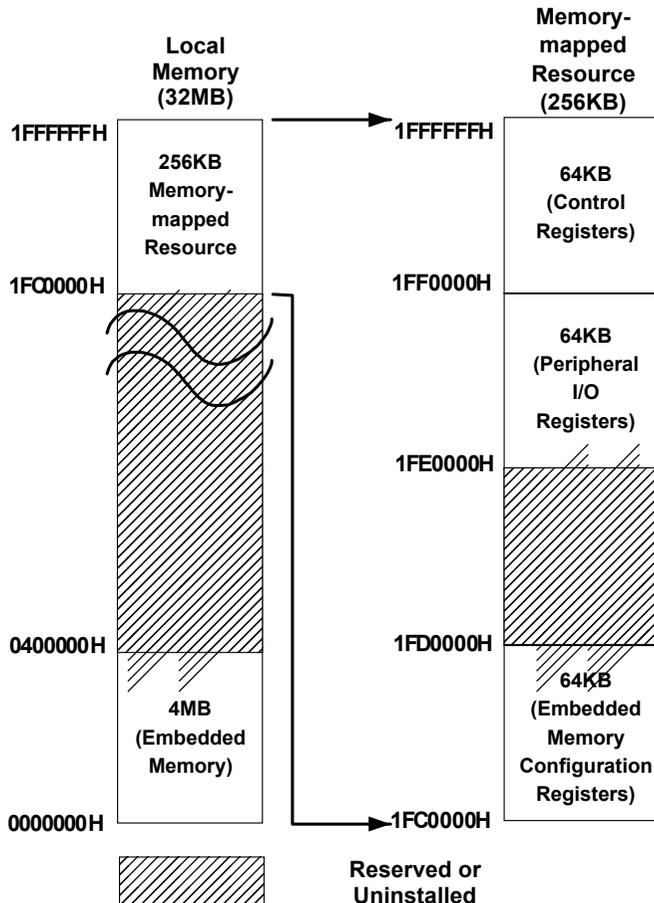


Figure 4-1. Address Space Allocation

Because the SM3110 is a PCI- and VGA-compatible device, certain of its registers and physical addresses are defined by those standards' specifications. The required registers and addresses are available at their standard locations; in addition, they are memory-mapped within the Control Address Space. The next section describes the memory-mapped registers, while the last sub-section describes the standard VGA registers.

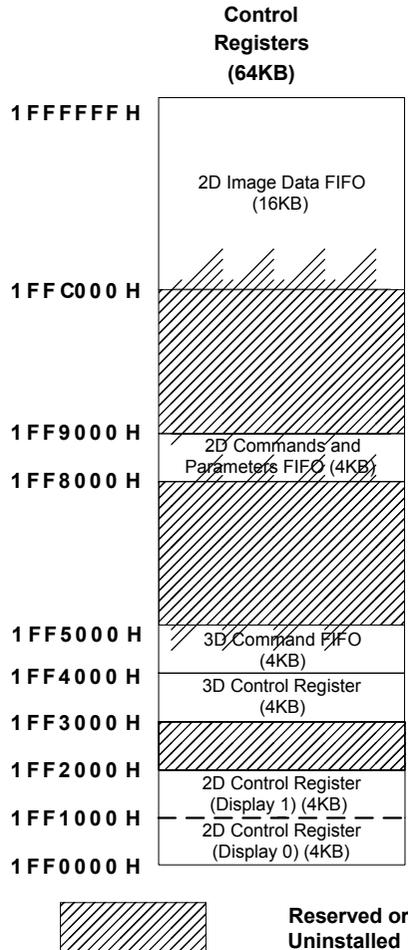


Figure 4-2. Control Address Space

The Control Address Space is divided into several areas containing registers, as shown in Figure 4-2. These areas are:

- 1) 2D Control Registers, Display 0: a 4KB region for Display 0. Base address is 1FF0000H.
- 2) 2D Control Registers, Display 1: a 4KB region for Display 1. Base address is 1FF1000H.
- 3) 3D Control Registers: a 4KB region. Base address is 1FF3000H.
- 4) 3D Command and Parameter FIFO: a 4KB region. Base address is 1FF4000H.
- 5) 2D Command and Parameter FIFO: a 4KB region. Base address is 1FF8000H.
- 6) 2D Image Data FIFO: a 16KB region. Base address is 1FFC000H

Register addresses are shown as a 12-bit offset designated (+xxxH) within the 4KB enhanced control register address space. The actual physical address is formed by adding the offset to 0FF0000H (big-endian) or 1FF0000H (little-endian).

4.2 Register Listing for Control Address Space

This section describes all of the SM3110 registers, grouped according to their major classifications within the Control Address Space. Table 4-1 shows each of these six register regions and its base address for both little- and big-endian access. The six regions are each discussed in the order they occur within the Control Address Space, from lowest to highest address. Within each region, the registers are also presented in order, from lowest to highest address offset.

	<i>Little-Endian</i>	<i>Big-Endian (DWS)</i>	<i>Big-Endian (WS)</i>	<i>Size</i>
2D Image Data FIFO	1FFC000H	5FFC000H	7FFC000H	16KB
<i>Reserved</i>	1FF9000H	5FF9000H	7FF9000H	12KB
2D Cmd/Param FIFO	1FF8000H	5FF8000H	7FF8000H	4KB
3D Cmd/Param FIFO	1FF4000H	5FF4000H	7FF4000H	4KB
3D Control	1FF3000H	5FF3000H	7FF3000H	4KB
<i>Reserved</i>	1FF2000H	5FF2000H	7FF2000H	4KB
2D Control (Display 1)	1FF1000H	5FF1000H	7FF1000H	4KB
2D Control (Display 0)	1FF0000H	5FF0000H	7FF0000H	4KB

Table 4-1. Control Address Space

4.2.1 2D Control (Display 0, Display 1) (1FF0000H)

The 2D control registers occupy a 4KB region within the control address space, as shown in Table 4-1. and are organized as 16 sets of 256B each. This is shown in Table 4-2. All are accessed with byte resolution.

Register addresses below are shown as a 12-bit offset designated “+xxxH” within the 4KB 2D control register address space. Where a register is common to both Display 0 and Display 1, the register will be shown with two addresses: “+0xxxH” and “+1xxxH”. When it is unique to Display 0, it is shown as “+xxxH”; unique to Display 1, as “+1xxxH”.

(Note that the VGA control registers are memory-mapped at offset +3xxH and the PCI configuration registers are memory-mapped at offset +FxxH.)

Register Type	Offset	Size	Access	Comments
Status/Control	000H	256B	any	Status/control, interrupts, power management, strapping, clocks
<i>Reserved</i>	100H	256B	none	
Bus Master	200H	256B	any	Address and size registers
VGA	300H	256B	byte	
Peripheral I/O	400H	256B	byte	ROM, VAFC/Phillips video decoder
LCD	500H	256B	any	LCD Control
<i>Reserved</i>	600H	256B	none	
<i>Reserved</i>	700H	256B	none	
2D Rendering Engine	800H	256B	any	2D setup
Memory Controller	900H	256B	any	Memory Timing
Surface Descriptors	A00H	256B	any	Address, stride, pixel size
Channel Descriptors	B00H	256B	any	Surface base index, offsets, pixel format
Buffers Descriptors	C00H	256B	any	Address, count
Display Control	D00H	256B	any	Surface scaling, formats, filtering, mix/blend control
Display Timing	E00H	256B	any	Display timing, syncs, viewports
PCI Configuration	F00H	256B	any	

Table 4-2. 2D Control Register Allocation

4.2.1.1 Status/Control (+000H)

4.2.1.1.1 Memory Clock Synthesizer

The memory clock synthesizer is controlled by directly accessing four byte registers. Two bytes specify the three (reference, loop/feedback and output) dividers and one controls the output and the PLL/VCO powerdown. One byte controls PLL characteristics and test features and should not be modified during normal operation.

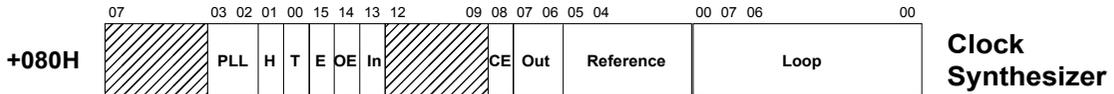


Figure 4-3. Memory Clock Synthesizer Registers

The synthesizer divider and control values must be changed only when the PLL circuitry is disabled. The sequence is to disable the PLL, modify the divider values, changing the register with the enable bit last. The registers controlling the PLL characteristics are not intended to be changed by BIOS (only by POST and/or diagnostic programs).

4.2.1.1.2 Frequency Control

Offset	Field	Access	Function
+0080H	15:00	R/W	MCLK Loop Control
	Bits	Field	
	15:14	Output Divider	
		Value	Semantics
		0	No division
		1	Divide by 2
		2	Divide by 2
		3	Divide by 4
	Bits	Field	
	13:09	Reference Divider	
		Value	Semantics
		0	Reserved
		1-31	Divide ratio is this number, plus two.
	Bits	Field	
	08:00	Loop Divider MSB	
		Value	Semantics
		0	Reserved.
		0-511	Divide ratio.

4.2.1.1.3 VCO Control

Offset	Field	Access	Function
+082H	07:00	R/W	MCLK VCO Control
	Bits	Field	
	07	PLL Bypass.	
		Value	Semantics
		0	Disabled.
		1	Enabled.
	Bits	Field	
	06	PLL power down mode	
		Value	Semantics
		0	Disabled.
		1	Enabled.
	Bits	Field	
	05	External Clock Source Select	
		Value	Semantics
		0	External Clock.
		1	Internal PLL Clock. Set by level of strapping input from ROMA1. <i>Default.</i> Input pulled up to V _{DD} .
	Bits	Field	
	04	Load PLL Counter to Working Register	
		Value	Semantics
		0	Normal Operation. <i>Default.</i>
		1	Load Register. Automatically returns to 0 after loading working register.
	Bits	Field	
	03:02	Reserved	
	01	PLL output enable	
		Value	Semantics
		0	Enabled
		1	Disabled
	Bits	Field	
	00	Clock Enable	
		Value	Semantics
		0	Disabled. Output forced to '0'.
		1	Enabled. Output cycles.

4.2.1.1.4 Dot/Panel Clock Synthesizer

The synthesizer divider and control values must be changed only when the PLL circuitry is disabled. The sequence is to disable the PLL, modify the divider values, changing the register with the enable bit last.

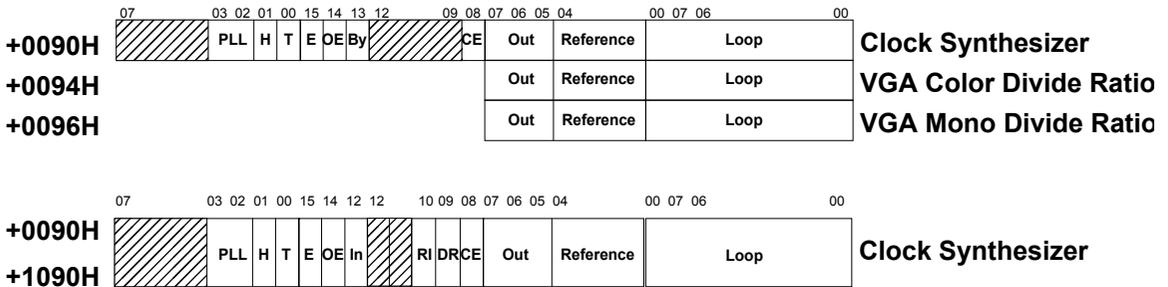


Figure 4-4. Dot Clock Synthesizer Registers

The registers controlling the PLL characteristics are not intended to be changed by BIOS (only by POST and/or diagnostic programs).

4.2.1.1.5 Programmable Clock Ratios

Offset	Field	Access	Function
+0090H	15:00	R/W	DCLK Loop Control
+1090H	15:00	R/W	DCLK Loop Control

Bits	Field	Value	Semantics
15:14	Output Divider	0	No division
		1	Divide by 2
		2	Divide by 2
		3	Divide by 4

Bits	Field	Value	Semantics
13:09	Reference Divider	0	Reserved
		1-31	Divide ratio is this number, plus two.

Bits	Field	Value	Semantics
08:00	Loop Divider MSB	0	Reserved.
		0-511	Divide ratio.

4.2.1.1.6 VCO Control

Offset	Field	Access	Function
+0092H	07:00	R/W	DCLK VCO Control (CRT)
+1092H	07:00	R/W	DCLK VCO Control (LCD)
<i>Bits Field</i>			
07	PLL Bypass.		
<i>Value Semantics</i>			
0	Disabled.		
1	Enabled. Minimum power consumption		
<i>Bits Field</i>			
06	Power Down.		
<i>Value Semantics</i>			
0	Disabled.		
1	Enabled.		
<i>Bits Field</i>			
05	External Clock Source Select.		
<i>Value Semantics</i>			
0	External Clock.		
1	Internal PLL set by level of strapping input from ROMA. <i>Default</i> , input pulled up to V _{DD} .		
<i>Bits Field</i>			
04	Load PLL Counter to Working Register		
<i>Value Semantics</i>			
0	Normal Operation. <i>Default</i> .		
1	Load Register. Automatically returns to 0 after loading working register.		
<i>Bits Field</i>			
03	PLL Output Enable.		
<i>Value Semantics</i>			
0	Enabled.		
<i>Bits Field</i>			
02	Reference Internal Oscillator Enable (Power Down)		
<i>Value Semantics</i>			
0	Enabled. <i>Default</i> .		
1	Disabled. shuts down oscillator to reduce power dissipation in sleep mode.		
<i>Bits Field</i>			
01	Divide Ratio Select		
<i>Value Semantics</i>			
0	VGA DCLK Frequencies, VGA Mode. <i>Default</i> . Use divide ratio specified in either 94 95H or 96 97H selected by 3CCH[3:2].		
1	Programmable DCLK Frequencies, Enhanced Mode. Use divide ratio specified in F0 F1H.		
<i>Bits Field</i>			
00	Clock Enable		
<i>Value Semantics</i>			
0	Disabled. Output forced to '0'.		
1	Enabled. Output toggles.		

4.2.1.1.7 VGA DCLK Presets

The following registers contain the divide ratios for the DCLK clock synthesizer for VGA graphics and text modes. These values are loaded into the DCLK clock synthesizer divide ratio registers as selected by the contents of VGA register 3CCH[3:2]

Mode	3CCH bits[3:2]	Registers offset 0xxH	Wid pixels	Hgt lines	Vert Hz	Fref MHz	Dref	Fref2 MHz	Dfclk	Fvco MHz	Dout	Fout MHz	Dmux	Fpixel MHz	Fdot MHz	Error
VGA Graphics	00B	94 95H	640	480	60	14.318	7	2.045	37	75.68	3	25.23	1	25.23	25.175	0.21
VGA Text	01B	96 97H	720	400	70	14.318	8	1.790	63	112.76	4	28.19	1	28.19	28.322	-0.47
Enhanced	11B	92 93H	640	480	75	14.318	5	2.864	33	94.50	3	31.50	1	31.50	31.500	0.00

Offset	Field	Access	Function
+094H	07:00	R/W	DCLK Loop Control (color)
+096H	07:00	R/W	DCLK Loop Control (mono)

Bits Field

07:00	Loop Divider LSB
-------	------------------

Offset	Field	Access	Function
+095H	07:00	R/W	DCLK Input/Output Control (color)
+097H	07:00	R/W	DCLK Input/Output Control (mono)

Bits Field

07:06	Output Divider
-------	----------------

Value Semantics

0	No division
1	Divide by 2
2	Divide by 2
3	Divide by 4

Bits Field

05:01	Reference Divider
-------	-------------------

Value Semantics

0	Reserved
1-31	Divide ratio

Bits Field

00	Loop divider, MSB
----	-------------------

4.2.1.1.8 System Clock DLL Control

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0A2H	07:00	R/W	SCLK DLL Control
	<i>Bits</i>	<i>Field</i>	
	07:02	Reserved.	
	01	DLL	
		<i>Value</i>	<i>Semantics</i>
		0	Disabled.
		1	Enabled. <i>Default.</i>
	<i>bits</i>	<i>field</i>	
	00	Reserved.	

4.2.1.1.9 Power Management Control

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0D0H	31:00	R/W	Power Management Control
	<i>Bits</i>	<i>Field</i>	
	31:21	Reserved.	
	20	CLKRUN mode	
		<i>Value</i>	<i>Semantics</i>
		0	SCLK nonstop. <i>Default.</i>
		1	Monitor internal SCLK activities.
	<i>Bits</i>	<i>Field</i>	
	19	CLKRUN enable	
		<i>Value</i>	<i>Semantics</i>
		0	Disable. <i>Default.</i>
		1	Enable.
	<i>Bits</i>	<i>Field</i>	
	18:00	Reserved.	

Power Management Control (continued)

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0D4H	31:00	R/W	Gated clock control for MCLK domain
	<i>Bits</i>	<i>Field</i>	
	31:07	<i>Reserved.</i>	
	06	3D	
	05	2D	
	04	<i>Reserved</i>	
	03	Display 2	
	02	Scaler 1	
	01	Scaler 0	
	00	MIU Address Translator FIFO	
		<i>Value</i>	<i>Semantics</i>
		0	Enable. <i>Default.</i>
		1	Disable. Clock is forced to be zero.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0D8H	31:00	R/W	Gated clock control for DCLK domain
	<i>Bits</i>	<i>Field</i>	
	31:04	<i>Reserved.</i>	
	03	Scaler 0	
	02	Scaler 1	
	01	disp1	
	00	disp 0	
		<i>Value</i>	<i>Semantics</i>
		0	Enable. <i>Default.</i>
		1	Disable. Clock is forced to be zero.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0E0H	31:00	R/W	Software Reset
	<i>Bits</i>	<i>Field</i>	
	31:04	<i>Reserved.</i>	
	03	Reset DCLK Domain logic	
	02	Reset MCLK Domain logic	
	01	Reset SCLK PCI configuration registers	
	00	Reset SCLK Domain logic but preserve PCI configuration registers	
		<i>Value</i>	<i>Semantics</i>
		0	Disable. <i>Default</i>
		1	Reset. The output of this register bit will be ORed with hardware reset signal. Software need to write '1' to enable the reset logic and then write '0' to disable the reset logic.

4.2.1.1.10 Status/Control Registers

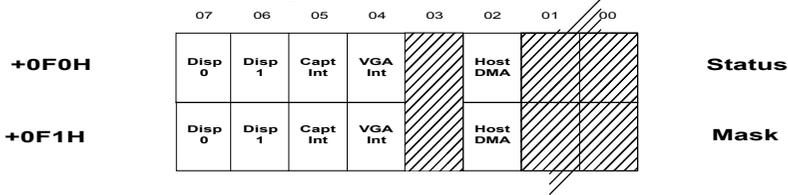


Figure 4-5. Status/Control Registers

The status bits reflected the combined (ORed) interrupt flags from each respective functional unit. The Mask suppresses reporting of interrupts but does not clear the individual interrupt flags. Note that the PCI clock (SCLK) does not have to be running in order to report interrupts. This allows asynchronous external interrupts to be reported to a “sleeping” system to wake it up.

Status

Offset	Field	Access	Function
+0F0H	07:00	R/O	Status
	Bits	Field	
	07	Display 0 VSYNC	
	06	Display 1 VSYNC	
	05	Video Capture Interrupt	
	04	VGA interrupt	
	03	Reserved	
	02	Host DMA	
	01:00	Reserved	
	Value	Semantics	
	0	Event has not occurred	
	1	Event has occurred	

Interrupt Mask

Offset	Field	Access	Function
+0F1H	07:00	R/W	Global Interrupt Mask. <i>Default</i> is 00H.
	Bits	Field	
	07	Display 0 VSYNC (see +0DF0H for details)	
	06	Display 1 VSYNC (see +1DF0H for details)	
	05	Video Capture Interrupt (see +xDF0H for details)	
	04	VGA interrupt (see VGA Register section)	
	03	Reserved	
	02	Host DMA	
	01:00	Reserved	
	Value	Semantics	
	0	Interrupt Disabled. <i>Default</i> .	
	1	Interrupt Enabled	

4.2.1.2 Reserved (+100H)

4.2.1.3 Host Bus Master (+200H)

The host bus master transfers data from buffers in system memory to predefined internal embedded memory addresses, the interface to an external audio DAC. The buffers are managed by hardware and the host driver and are configured as circular queues.

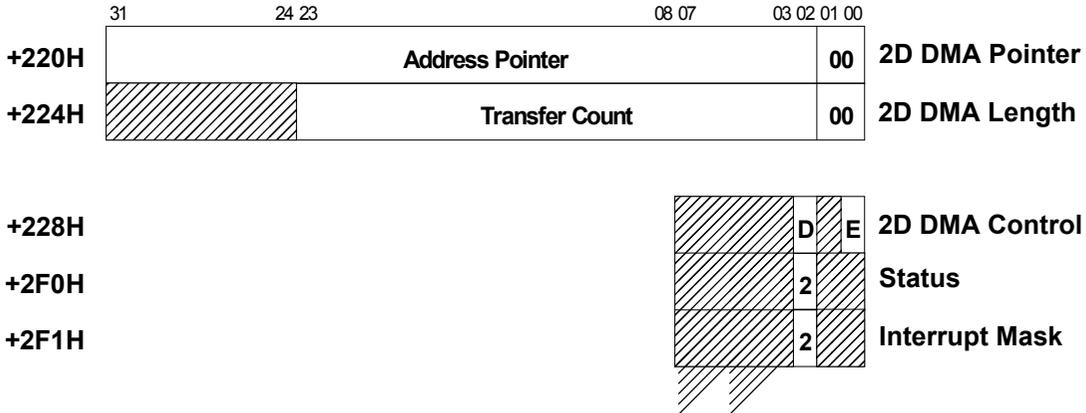


Figure 4-6. Host Bus Master

4.2.1.3.1 Descriptor Address

Offset	Field	Access	Function
+220H	31:00	R/W	2D DMA Address - system to/from local memory
	Bits	Field	
	31:00	Address Pointer to System Memory. Must be a doubleword address, bits [01:00] must be zero.	

4.2.1.3.2 Descriptor Count

Offset	Field	Access	Function
+224H	31:00	R/W	2D DMA Count - system to/from local memory
	Bits	Field	
	31:24	<i>Reserved.</i>	
	23:00	Transfer Count, number of bytes -4. Must be in multiples of doublewords, bits [01:00] must be zero.	

4.2.1.3.3 Descriptor Control

Offset	Field	Access	Function
+228H	07:00	R/W	2D DMA Control - system to/from local memory
	Bits	Field	
	07:03	Reserved.	
	02	Transfer Direction.	
		Value	Semantics
		0	Local to System.
		1	System to Local.
	bits	field	
	01	Reserved.	
	00	Transfer Enable. <i>Note that this bit is not cleared automatically by the DMA upon completion of the last transfer. The driver disables the DMA while channel is being (re)initialized.</i>	
		Value	Semantics
		0	Disable. <i>Default.</i>
		1	Enable.

4.2.1.3.4 Interrupts Status

Offset	Field	Access	Function
+2F0H	07:00	R/W	2D DMA Status/Interrupt Flag
	Bits	Field	
	07:03	Reserved.	
	02	Host DMA transfer complete - system to/from local memory.	
	01:00	Reserved.	
		Value	Semantics
		0	Read. Event has not occurred.
		1	Read. Event occurred.
		0	Write. No change.
		1	Write. Reset Flag.

4.2.1.3.5 Interrupts Mask

Offset	Field	Access	Function
+2F1H	07:00	R/W	2D DMA Interrupt Mask
	Bits	Field	
	07:03	Reserved.	
	02	Host DMA - system to/from local memory	
	01:00	Reserved.	
		Value	Semantics
		0	Interrupt Disabled. (masked)
		1	Interrupt Enabled.

4.2.1.4 VGA Registers (+300H)

This area is a memory-mapped equivalent of the standard VGA control register address space. See Section 4.3, Standard VGA Registers, for a description of the VGA registers.

Accessing VGA registers through memory-mapped addressing can only be done with byte-addressing.

4.2.1.5 Peripheral Ports (+400H)

There are several peripheral I/O ports that are controlled through memory-mapped registers.

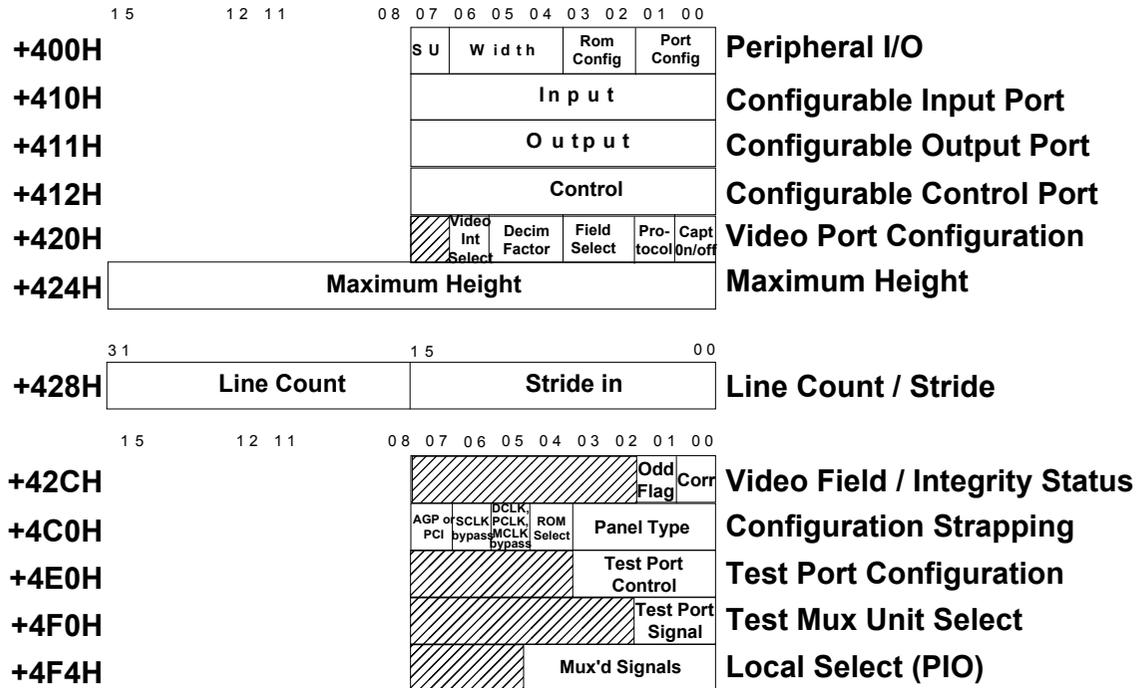


Figure 4-7. Peripheral Ports

4.2.1.5.1 Port Timing

This specifies the transfer timing for the BIOS ROM and peripheral I/O device accesses. Once the BIOS has been shadowed (copied) in system memory, the timing may be adjusted to satisfy the slowest device that is connected to the peripheral I/O bus. Timing is specified in units of the PCI or AGP bus clocks. The timing defaults to the slowest settings (360ns strobe width with a 33MHz PCI clock). The address hold time is fixed at 1 clock.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+400H	07:00	R/W	Peripheral I/O Port Configuration
	<i>Bits</i>	<i>Field</i>	
	07	Address Setup.	
		<i>Value</i>	<i>Semantics</i>
		0	2 clocks. <i>Default.</i>
		1	1 clock.
	<i>Bits</i>	<i>Field</i>	
	06:04	Strobe Width.	
		<i>Value</i>	<i>Semantics</i>
		0	12 clocks. <i>Default.</i>
		1-7	<i>value</i> +4 clocks.
	<i>Bits</i>	<i>Field</i>	
	03:02	ROM port configuration	
		<i>Value</i>	<i>Semantics</i>
		11	Display 1
		10	Display 0
		01	Peripheral I/O address for fresh ROM update.
		00	ROM
	<i>Bits</i>	<i>Field</i>	
	01:00	LCD Port Configuration.	
		<i>Value</i>	<i>Semantics</i>
		11	Display 1
		10	Display 0
		01	<i>Reserved.</i>
		00	LCD

4.2.1.5.2 Configurable Data Port

A configurable data port is provided to access up to 8 bits of data that can be used for:

- A software implementation of I²C or DDC protocols over two (mutually exclusive) pairs of pins.
- Any other programmable input or output.

The interface consists of three bytes allowing complete and independent control of 8 pins:

- The first byte is input only, returning the value at the input pins.
- The second byte is output only, the value written is driven on the output pins if enabled.
- The third byte independently controls the output enable of each bit of the data output port.

Configurable I/O Input

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+410H	07:00	R/O	Configurable Input Port
	<i>Bits</i>	<i>Field</i>	
	07:00	Input.	
		<i>Value</i>	<i>Semantics</i>
		0	Level at pin input is low.
		1	Level at pin input is high.

Configurable I/O Output

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+411H	07:00	R/W	Configurable Output Port
	<i>Bits</i>	<i>Field</i>	
	07:00	Output. Write	
		<i>Value</i>	<i>Semantics</i>
		0	Level to pin output is low.
		1	Level to pin output is high.
	<i>bits</i>	<i>field</i>	
	07:00	Output. Read	
		<i>Value</i>	<i>Semantics</i>
		0	Level last written to pin output is low.
		1	Level last written to pin output is high.

Configurable I/O Control

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+412H	07:00	R/W	Configurable Control Port
	<i>Bits</i>	<i>Field</i>	
	07:00	Enable. <i>Default</i> is FFH, all bits disabled.	
		<i>Value</i>	<i>Semantics</i>
		0	Output is enabled and pulled low.
		1	Output is disabled, it will be pulled high by an external pullup resistor.

4.2.1.5.3 Video Input Port

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+420H	07:00	R/W	Video Port Configuration
	<i>Bits</i>	<i>Field</i>	
	07	<i>Reserved.</i>	
	06	Video Port Interrupt Select	
		0	Interrupt at end of frame. <i>Default</i>
		1	Interrupt at end of field
	<i>Bits</i>	<i>Field</i>	
	05:04	Prescaler Factor. Data is assumed to be YUV/YC _B C _R only.	
		<i>Value</i>	<i>Semantics</i>
		0	Full scale. <i>Default</i>
		1	1/2. Interpolated
		2	1/4. Interpolated
		3	1/8. Interpolated
	<i>Bits</i>	<i>Field</i>	
	03:02	Field Select	
		<i>Value</i>	<i>Semantics</i>
		0	Interleaved (keeps odd and even fields in same surface). <i>Default</i>
		1	Capture odd field only.
		2	Capture even field only.
		3	Capture both odd and even field (bob) in seperate surfaces.
	<i>Bits</i>	<i>Field</i>	
	01	Protocol.	
		<i>Value</i>	<i>Semantics</i>
		0	Phillips SAA7111. <i>Default</i>
		1	ZV protocol.
	<i>Bits</i>	<i>Field</i>	
	00	Video Capture on/off. Program this bit last after all other Vidcap registers.	
		<i>Value</i>	<i>Semantics</i>
		0	Video Capture block off. <i>Default.</i>
		1	Video Capture block on.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+424H	15:00	R/W	Maximum Height
	<i>Bits</i>	<i>Field</i>	
	15:00	Maximum Height (Maximum number of lines to capture)	
		<i>Value</i>	<i>Semantics</i>
		0-FFFFH	Maximum number of lines to capture (0-64K). Note: this number must be even. If an odd number is entered, the last bit will be treated as a zero.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+428H	31:00	R/O	Video Input Dimensions
	<i>Bits</i>	<i>Field</i>	
	31:16	Line count of external video input	
		<i>Value</i>	<i>Semantics</i>
		0-FFFFH	Maximum number of lines captured (0-64K)
	<i>Bits</i>	<i>Field</i>	
	15:00	Stride of external video input.	
		<i>Value</i>	<i>Semantics</i>
		0-FFFFH	Number of pixels/line captured

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+42CH	7:00	R/O	Video Field and Integrity Status
	<i>Bits</i>	<i>Field</i>	
	07:02	<i>Reserved</i>	
	01	Odd Field Flag	
		<i>Value</i>	<i>Semantics</i>
		1	Odd field
		0	Even field
	<i>Bits</i>	<i>Field</i>	
	00	External Video Data Corruption (1=corruption, 0=no corruption)	
		<i>Value</i>	<i>Semantics</i>
		1	Video corruption
		0	No corruption

4.2.1.5.4 Configuration Strapping

This read-only register contains the values sampled from the configuration pins at the trailing edge of RESET. These determine certain operational modes of the chip that must be specified initially before any BIOS or other initialization routines can access any other configuration registers in the chip. In particular, these modes relate to connections and configuration at the board level (such as bus type and I/O) that cannot be determined by initialization/configuration BIOS software. A weak internal pullup to V_{DD} is provided on each sampled input; for a configuration bit to be sampled as a '0', an external pulldown to V_{SS} is required.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+4C0H	7:00	R/O	Configuration Strapping
	<i>Bits</i>	<i>Field</i>	
	07	AGP or PCI Select.	
		<i>Value</i>	<i>Semantics</i>
		0	PCI. <i>Default.</i>
		1	AGP
	<i>Bits</i>	<i>Field</i>	
	06	SCLK bypass	
		<i>Value</i>	<i>Semantics</i>
		0	Bypassed
		1	Not bypassed. <i>Default.</i>
	<i>Bits</i>	<i>Field</i>	
	05	DCLK, PCLK and MCLK Bypass.	
		<i>Value</i>	<i>Semantics</i>
		0	Bypassed
		1	Not bypassed. <i>Default.</i>
	<i>Bits</i>	<i>Field</i>	
	04	ROM Select (32K or 64K).	
		<i>Value</i>	<i>Semantics</i>
		0	32K
		1	64K
	<i>Bits</i>	<i>Field</i>	
	03:00	Panel Type	
		<i>Value</i>	<i>Semantics</i>
			Software defined

4.2.1.6 LCD Panel (+500H)

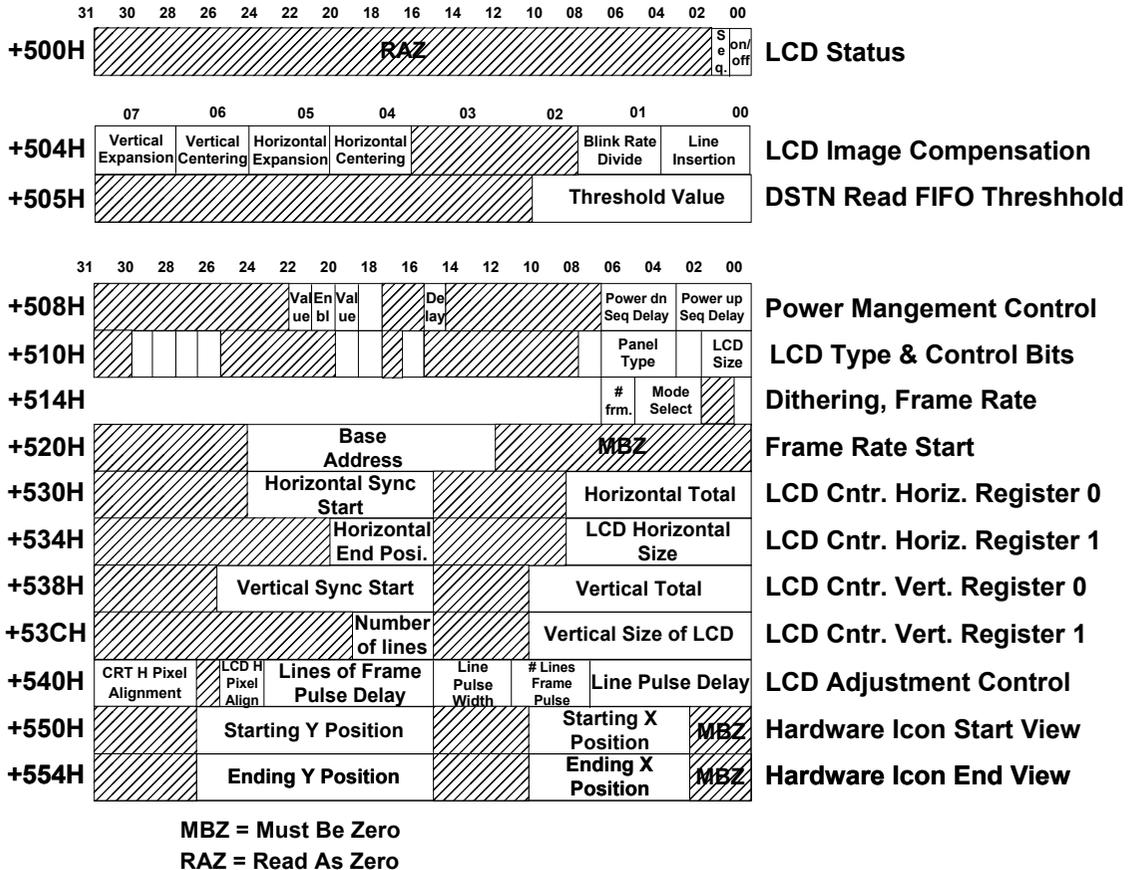


Figure 4-8. LCD Panel Registers

4.2.1.6.1 LCD Status

Offset	Field	Access	Function
+500H	31:00	R/O	LCD Status
	Bits	Field	
	31:02		Reserved. Read as zero.
	01		LCD power sequencing in progress
		Value	Semantics
		0	False
		1	True
	Bits	Field	
	00		LCD
		Value	Semantics
		0	Off
		1	On

4.2.1.6.2 LCD Image Compensation

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+504H	07:00	R/W	LCD Image Compensation
<i>Bits</i>		<i>Field</i>	
07	Enable vertical expansion for standard VGA modes		
<i>Value</i>		<i>Semantics</i>	
0	Disable. <i>Default.</i>		
1	Enable.		
<i>Bits</i>		<i>Field</i>	
06	Reserved		
05	Enable horizontal expansion for standard VGA modes		
<i>Value</i>		<i>Semantics</i>	
0	Disable. <i>Default.</i>		
1	Enable.		
<i>Bits</i>		<i>Field</i>	
04:02	<i>Reserved.</i>		
01	Text foreground color blinking rate and cursor blinking rate divided by half		
<i>Value</i>		<i>Semantics</i>	
0	Disable. <i>Default.</i>		
1	Enable.		
<i>Bits</i>		<i>Field</i>	
00	Enable line insertion function in text mode for 640x480 panel		
<i>Value</i>		<i>Semantics</i>	
0	Disable. <i>Default.</i>		
1	Enable.		

4.2.1.6.3 DSTN Read FIFO Control

Offset	Field	Access	Function
+505H	07:00	R/W	DSTN Read FIFO Threshold
	Bits	Field	
	07:02	Reserved.	
	01:00	Threshold value = 1 (default = 0)	

4.2.1.6.4 LCD Power Management Control

Offset	Field	Access	Function
+508H	31:00	R/W	Power management control
	Bits	Field	
	31:23	Reserved.	
	22	Software override fpV _{ee} value. Default is 0	
	21	Software override fpV _{ee} enable. Default is 0	
	20	Software override fpbklt value. Default is 0	
	19	Software override fpbklt enable. Default is 0	
	18:16	Reserved.	
	15	Power down delay	
		Value	Semantics
		0	Normal power down delay. <i>Default.</i>
		1	fast power down delay
	Bits	Field	
	14:08	Reserved.	
	07:04	Power down sequence delay. <i>Default is: 0010</i>	
	03:00	Power up sequence delay. <i>Default is: 0010</i>	

4.2.1.6.5 LCD Type and Control Bits

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+510H	31:00	R/W	LCD Type and Control Bits
	<i>Bits</i>	<i>Field</i>	
	31:30	<i>Reserved.</i>	
	29	FPVDCLK Polarity	
		<i>Value</i>	<i>Semantics</i>
		0	Normal. <i>Default.</i>
		1	Inverted.
	<i>Bits</i>	<i>Field</i>	
	28	FPVDCLK Clock Enable	
		<i>Value</i>	<i>Semantics</i>
		0	Free running. <i>Default.</i>
		1	Gated by display enable.
	<i>Bits</i>	<i>Field</i>	
	27	Line pulse Enable	
		<i>Value</i>	<i>Semantics</i>
		0	Free running. <i>Default.</i>
		1	Gated by display enable.
	<i>Bits</i>	<i>Field</i>	
	26	HDE	
		<i>Value</i>	<i>Semantics</i>
		0	ANDed with VDE. <i>Default.</i>
		1	Not ANDed with VDE.
	<i>Bits</i>	<i>Field</i>	
	25:20	<i>Reserved.</i>	
	<i>Bits</i>	<i>Field</i>	
	19	Line Pulse Phase	
		<i>Value</i>	<i>Semantics</i>
		0	Normal. <i>Default.</i>
		1	Inverted.
	<i>Bits</i>	<i>Field</i>	
	18	LFS Phase	
		<i>Value</i>	<i>Semantics</i>
		0	Normal. <i>Default.</i>
		1	Inverted.
	<i>Bits</i>	<i>Field</i>	
	17	DSTN frame rate	
		<i>Value</i>	<i>Semantics</i>
		0	Normal. <i>Default.</i>
		1	Doubled
	<i>Bits</i>	<i>Field</i>	
	16	Pixel data polarity	
		<i>Value</i>	<i>Semantics</i>
		0	Normal. <i>Default.</i>
		1	Inverted.

<i>Bits</i>	<i>Field</i>
15:10	<i>Reserved.</i>
09	Synchronize CRT with LCD
	<i>Value Semantics</i>
	0 <i>Disable. Default.</i>
	1 <i>Enable.</i>
<i>Bits</i>	<i>Field</i>
08	LCD Display Subsystem
	<i>Value Semantics</i>
	0 <i>Disable. Default.</i>
	1 <i>Enable.</i>
<i>Bits</i>	<i>Field</i>
07:04	Panel Type
	<i>Value Semantics</i>
	0 <i>DSTN-8. Default.</i>
	1 <i>DSTN-16</i>
	2 <i>DSTN-24</i>
	3-7 <i>Reserved</i>
	8 <i>TFT-9</i>
	9 <i>TFT-12</i>
	10 <i>Reserved</i>
	11 <i>TFT-18</i>
	12 <i>TFT-24</i>
	13-15 <i>Reserved</i>
<i>Bits</i>	<i>Field</i>
03	TFT Pixel to Clock ratio
	<i>Value Semantics</i>
	0 <i>1 pixel/clock. Default.</i>
	1 <i>2 pixel/clock.</i>
<i>Bits</i>	<i>Field</i>
02	VGA Line Graphics extension by duplication of last bit.
	<i>Value Semantics</i>
	0 <i>Disabled. Default.</i>
	1 <i>Enabled for all ASCII codes.</i>
<i>Bits</i>	<i>Field</i>
01:00	LCD Panel Stretch Ratio
	<i>Value Semantics</i>
	0 <i>640x480. Default.</i>
	1 <i>800x600</i>
	2 <i>1024x768 or higher.</i>
	3 <i>Reserved.</i>

4.2.1.6.6 LCD Dithering and Frame Rate Modulation

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+514H	07:00	R/W	Dithering and Frame-Rate-Modulation
<i>Bits</i>		<i>Field</i>	
07:06	FRC frame		
<i>Value</i>		<i>Semantics</i>	
0	8 frame FRC.		
1	16 frame FRC. <i>Default.</i>		
2	256 frame FRC (EEDS pat. pending)		
3	<i>Reserved</i>		
<i>Bits</i>		<i>Field</i>	
05:03	TFT dithering mode selection		
<i>Value</i>		<i>Semantics</i>	
0	5bit to 4bit. <i>Default.</i>		
1	5bit to 3bit		
2	6bit to 4bit		
3	6bit to 3bit		
4	8bit to 6bit		
5	8bit to 4bit		
6-7	<i>Reserved.</i>		
<i>Bits</i>		<i>Field</i>	
02:01	<i>Reserved.</i>		
00	Enable Dithering		
<i>Value</i>		<i>Semantics</i>	
0	Disable. <i>Default.</i>		
1	Enable.		

4.2.1.6.7 DSTN Frame Rate Accelerator Starting Address

Offset	Field	Access	Function
+520H	31:00	R/W	DSTN frame-rate-accelerator starting address
	Bits	Field	
	31:24	Reserved.	
	23:12	Base Address, in units of 4KB.	
	11:00	Must be zero to align to a natural 4KB boundary.	

4.2.1.6.8 LCD Controller CRTC Registers

These registers are fully programmable only in Enhanced VGA Modes. In standard VGA modes only +538H[08:00] and +53CH[10:00] are programmable; the remaining values are managed by the hardware.

Offset	Field	Access	Function
+530H	31:00	R/W	LCD Controller Horizontal Register 0
	Bits	Field	
	31:25	Reserved.	
	24:16	Mirror of VGA CRTC/04; horizontal sync start register	
	15:09	Reserved.	
	08:00	Mirror of VGA CRTC/00; horizontal total register: real total = value + 5, i.e. 63h+5h = 68h	

Offset	Field	Access	Function
+534H	31:00	R/W	LCD Controller Horizontal Register 1
	Bits	Field	
	31:21	Reserved.	
	20:16	Mirror of VGA CRTC/05; horizontal sync end position register	
	15:09	Reserved.	
	08:00	number of character clocks is programmed; LCD horizontal size: real size = value + 1	

Offset	Field	Access	Function
+538H	31:00	R/W	LCD Controller Vertical Register 0
	Bits	Field	
	31:27	Reserved.	
	26:16	Mirror of VGA CRTC/10; number of lines is programmed. Vertical sync start: Width = sync end – sync start	
	15:11	Reserved.	
	10:00	Mirror of CRTC/06; number of lines is programmed. Vertical total: Real total = value + 2	

Offset	Field	Access	Function
+53CH	31:00	R/W	LCD Controller Vertical Register 1
	Bits	Field	
	31:20	Reserved.	
	19:16	Mirror of VGA CRTC/11; number of lines is programmed into this register. Vertical sync end register.	
	15:11	Reserved.	
	10:00	Number of lines is programmed. Vertical size of LCD. Real size = value + 1.	

4.2.1.6.9 LCD Adjustment Control

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+540H	31:00	R/W	LCD Adjustment Control Register
	<i>Bits</i>	<i>Field</i>	
	31:28	CRT vertical pixel alignment.	
		<i>Value</i>	<i>Semantics</i>
		0	No delay.
		1-15	1-15 clock delay.
	<i>Bits</i>	<i>Field</i>	
	27	<i>Reserved.</i>	
	26:24	LCD controller vertical pixel alignment	
		<i>Value</i>	<i>Semantics</i>
		0	No delay.
		1-7	1-7 pixels delay.
	<i>Bits</i>	<i>Field</i>	
	23:16	number of lines is programmed for frame pulse delay control.	
	15:12	number of character clocks is programmed for line pulse width (= value)	
	11:08	number of scan lines is programmed for frame pulse width (= value)	
	07:00	number of character clocks is programmed for line pulse delay control.	

4.2.1.6.10 VGA Hardware ICON Viewport

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+550H	31:00	R/W	Hardware ICON Starting Viewport

<i>Bits</i>	<i>Field</i>
31:27	<i>Reserved.</i>
26:16	Starting Y position
15:11	<i>Reserved.</i>
10:03	Starting X position (8 pixel granularity)
02:00	Must be zero.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+554H	31:00	R/W	Hardware ICON Ending Viewport

<i>Bits</i>	<i>Field</i>
31:27	<i>Reserved.</i>
26:16	Ending Y position
15:11	<i>Reserved.</i>
10:03	Ending X position (8 pixel granularity)
02:00	Must be zero.

4.2.1.7 *Reserved (+600H)*

4.2.1.8 *Reserved (+700H)*

4.2.1.9 2D Rendering Engine (+800H)

There are a maximum of 16 registers controlling the 2D Rendering Engine that are normally loaded via the PCI through the command/parameter FIFO. PCI register indexes are in double-word units and must be multiplied by 4 to generate the corresponding register byte offset that is added to the Rendering Engine offset.

	31	28	27	16	15	12	11	00					
+804H								SRC Stride (Pixels)		Linear BLT Source Stride			
+808H								Y Coordinate	X Coordinate	Destination Y,X Coordinates			
+80CH								Y Extent	X Extent	Destination Y,X Extents			
+810H								Y Coordinate	X Coordinate	Source Y,X Coordinates			
+818H								Y Coordinate	X Coordinate	Clip UL Y,X Coordinates			
+81CH								Y Coordinate	X Coordinate	Clip LR Y,X Coordinates			
+820H			Align	Rt	Up	FT	BT	Pat	Fmt	Op	RasterOp Code	Flags	
+824H						Format			S Index			D Index	Surface Descriptor
+828H								8b CLUT, 15b, 16b or 24b Color			Foreground Color		
+82CH								8b CLUT, 15b, 16b or 24b Color			Background Color		
+830H								8b CLUT, 15b, 16b or 24b Color			Transparent Color		
+838H								Linear BLT Source Byte Address			Linear BLT Source Address		
											07	00	
+8F0H											Busy	State	2D Engine Status
+8F1H	Reset										Single Step	Start/Stop Request	2D Engine Control

Figure 4-9. 2D Rendering Engine

4.2.1.9.1 Linear BLT Source Stride

The source (pixel) stride is specified in the least significant 12 bits of the parameter register (using rectangular format).

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+804H	31:00	R/W	Source Stride
	<i>Bits</i>	<i>Field</i>	
	31:12	<i>Reserved.</i>	
	11:00	Source Stride in (destination) pixels.	

4.2.1.9.2 Source/Destination

Parameter coordinates are specified using rectangular pixel addressing to reduce the path length of driver setup code. The resulting local memory address is computed in bytes. 0,0 is the upper left corner of the screen, X_{\max} , Y_{\max} is the lower right corner of the screen. X and Y coordinates are assumed to be 12-bit two's complement. X and Y extents are 12-bit unsigned values (allowing the extent to range from 1 to 2048; an extent of 0 causes nothing to happen). Both are stored right-justified in a 16 bits field.

Note: Extents specify the actual number of pixels (not decremented by 1).

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+808H	31:00	R/W	Destination Y, X Coordinates
	<i>Bits</i>	<i>Field</i>	
	31:28	<i>Reserved.</i>	
	27:16	Y Coordinate, 12b 2's complement	
	15:12	<i>Reserved.</i>	
	11:00	X Coordinate, 12b 2's complement	

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+80CH	31:00	R/W	Destination Y, X Extents
	<i>Bits</i>	<i>Field</i>	
	31:27	<i>Reserved.</i>	
	26:16	Y Extent, 12b unsigned.	
	15:11	<i>Reserved.</i>	
	10:00	X Extent, 12b unsigned.	

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+810H	31:00	R/W	Source Y, X coordinates
	<i>Bits</i>	<i>Field</i>	
	31:28	<i>Reserved.</i>	
	27:16	Y Coordinate, 12b 2's complement	
	15:12	<i>Reserved.</i>	
	11:00	X Coordinate, 12b 2's complement	

4.2.1.9.3 Clip Rectangle

Offset	Field	Access	Function
+818H	31:00	R/W	Clip Rectangle, Upper left corner Y, X pixel coordinates
	Bits	Field	
	31:28	Reserved.	
	27:16	Y Coordinate, 12b unsigned	
	15:12	Reserved.	
	11:00	X Coordinate, 12b unsigned	
Offset	Field	Access	Function
+81CH	31:00	R/W	Clip Rectangle, Lower right corner Y, X pixel coordinates
	Bits	Field	
	31:28	Reserved.	
	27:16	Y Coordinate, 12b unsigned	
	15:12	Reserved.	
	11:00	X Coordinate, 12b unsigned	

4.2.1.9.4 Flags

The command FIFO loads the following registers as 24 bits.

Alignment, Direction (X,Y)

Offset	Field	Access	Function
+820H	31:00	R/W	Flags
	Bits	Field	
	31:24	Reserved. Must be zero.	
	Bits	Field	
	23:22	Inter-Scan Alignment	
		Value	Semantics
		0	Byte
		1	Word (16b)
		2	DoubleWord (32b)
		3	Bit (valid only for mono source)
	Bits	Field	
	21	X Direction, Right	
		Value	Semantics
		0	Left, left to right, increasing X value.
		1	Right, right to left, decreasing X value.
	Bits	Field	
	20	Y Direction, Up	
		Value	Semantics
		0	Down, top to bottom, increasing Y value.
		1	Up, bottom to top, decreasing Y value.
	Bits	Field	
	19	Reserved. Must be zero.	

Mono Source BLT Transparency

Controls write of the ROP/BLT operation with a mono source to the destination as summarized in the table below:

<i>Bits</i>	<i>Field</i>
18	Pattern Transparency.
	<i>Value Semantics</i>
	0 Opaque.
	1 Transparent, the pattern is compared against the value in the transparent color register.
<i>Bits</i>	<i>Field</i>
17	BLT Foreground Transparency.
	<i>Value Semantics</i>
	0 Opaque.
	1 Transparent.
<i>Bits</i>	<i>Field</i>
16	BLT Background Transparency.
	<i>Value Semantics</i>
	0 Opaque.
	1 Transparent.

Note: Definitions of these bits differs significantly from the same bits in Color Source BLT Transparency Control.

<i>[17]</i>	<i>[16]</i>	<i>Data=1</i>	<i>Data=0</i>	<i>Function</i>	<i>Comments</i>
0	0	Foreground	Background	normal	
0	1	Foreground	don't change	object over pattern	"weatherman in front of a map"
1	0	don't change	Background	pattern thru background	
1	1	Background	Foreground	reverse	

Color Source BLT Transparency Control

For the transparent BLT operation, this field controls the write of the ROP/BLT operation to the destination by comparing the comparand (source, destination or pattern) with the transparent color register.

<i>Bits</i>	<i>Field</i>
18	Transparent BLT Sense.
	<i>Value Semantics</i>
0	Normal. Writes are disabled where the comparand and transparent color register matches.
1	Inverted. Writes are enabled where the comparand and transparent color register matches.
<i>Bits</i>	<i>Field</i>
17:16	BLT Foreground Transparency Comparand Source.
	<i>Value Semantics</i>
0	Opaque. <i>Do not compare.</i>
1	Source.
2	Destination.
3	Pattern.

Note: Definitions of these bits differs significantly from the same bits in Monochrome BLT Transparency Control.

Source Control

<i>Bits</i>	<i>Field</i>
15	Clipping
	<i>Value Semantics</i>
0	Disable Clipping.
1	Enable Clipping.
14	Pattern.
	<i>Value Semantics</i>
0	Use pattern buffer.
1	Use background register instead of pattern buffer.
13	Source
	<i>Value Semantics</i>
0	Local Memory (screen to screen).
1	Image Data (copied from system memory by the CPU).
12	Source Format
	<i>Value Semantics</i>
0	Mono.
1	Color.

Operation

A four bit field specifies up to 16 different operations. A secondary function field specifies the raster Op code that is used when one of the raster operations is specified.

<i>Bits</i>	<i>Field</i>
11:08	Operation.
	<i>Value Semantics</i>
0-1H	<i>Reserved.</i>
2H	BitBLT with Logical ROP. Function specified in ROP code field.
3H	<i>Reserved.</i>
4-5H	<i>Reserved.</i>
6H	LinBLT with Logical ROP. Function specified in ROP code field.
7H	<i>Reserved.</i>
8H	Signal3D
9H	Wait3D
A-FH	<i>Reserved.</i>
<i>Bits</i>	<i>Field</i>
07:00	Raster Op Code
	<i>Value Semantics</i>
00-FFH	Raster Operation.

4.2.1.9.5 Surface Descriptors

The base address register pointers (point to an actual surface base address register) for source and destination. The pixel size (1, 2 or 3 bytes per pixel) is specified for both source and destination.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+824H	31:00	R/W	Surface Descriptor
	<i>Bits</i>	<i>Field</i>	
	31:18	<i>Reserved.</i>	
	17:16	Source/Destination Surface Pixel Size.	
		<i>Value</i>	<i>Semantics</i>
		0	8bpp.
		1	16bpp.
		2	24bpp.
		3	<i>Reserved.</i>
	<i>Bits</i>	<i>Field</i>	
	15:12	<i>Reserved.</i>	
	11:08	Source Surface Base Register Index. specifies which surface base register should be used.	
	07:04	<i>Reserved.</i>	
	03:00	Destination Surface Base Register Index. specifies which surface base register should be used.	
		<i>Value</i>	<i>Semantics</i>
		0-FH	Surface base register index

4.2.1.9.6 Color Setting

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+828H	31:00	R/W	Foreground Color
	<i>Bits</i>	<i>Field</i>	
	23:00	24b pixel, right justified, bits [31:24] are <i>Reserved</i> .	
	15:00	15/16b pixel, right justified, bits [31:16] are <i>Reserved</i> .	
	07:00	8b pixel, right justified, bits [31:08] are <i>Reserved</i> .	
<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+82CH	31:00	R/W	Background Color
	<i>Bits</i>	<i>Field</i>	
	23:00	24b pixel, right justified, bits [31:24] are <i>Reserved</i> .	
	15:00	15/16b pixel, right justified, bits [31:16] are <i>Reserved</i> .	
	07:00	8b pixel, right justified, bits [31:08] are <i>Reserved</i> .	
<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+830H	31:00	R/W	Transparent Color
	<i>Bits</i>	<i>Field</i>	
	23:00	24b pixel, right justified, bits [31:24] are <i>Reserved</i> .	
	15:00	15/16b pixel, right justified, bits [31:16] are <i>Reserved</i> .	
	07:00	8b pixel, right justified, bits [31:08] are <i>Reserved</i> .	

4.2.1.9.7 Linear BLT Source Address

The linear (byte) address is specified in the least significant 24 bits of the parameter register (using linear format). Maximum address space is 16 megabytes.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+838H	31:00	R/W	Linear Source Address
	<i>Bits</i>	<i>Field</i>	
	31:24	<i>Reserved</i> .	
	23:00	Linear Address in bytes.	

4.2.1.9.8 2D Engine Status

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+8F0H	07:00	R/O	2D Rendering Engine Status
	<i>Bits</i>	<i>Field</i>	
	07:02	<i>Reserved.</i>	
	01	<i>Busy.</i>	
		<i>Value</i>	<i>Semantics</i>
		0	Idle, completed all commands and no pending memory accesses. <i>Default.</i>
		1	Busy, either executing a command or pending completion of a memory access).
	<i>Bits</i>	<i>Field</i>	
	00	<i>Running.</i>	
		<i>Value</i>	<i>Semantics</i>
		0	Stopped (State), in response to the run bit being deasserted or after reset. <i>Default.</i>
		1	Running (State), either idle awaiting commands or busy processing a command.

Note: A false Idle may be read if the engine has not yet begun to execute the programmed instruction. Wait several instructions before polling this register to be sure the instruction at least has begun executing.

4.2.1.9.9 2D Engine Control

There is a single byte register that controls the run/step/stop state of the 2D rendering engine. The Run (request) control bit and running status bit are used together to control execution of the rendering engine. After deasserting the Start bit, the Running bit should be polled to determine when the engine has actually stopped. The deassertion of the Running bit will always lag deassertion of the Start bit by one or more clock cycles. The Reset bit can be asserted to cause the engine to be reset; it must then be deasserted to allow the engine to restart execution when the Start bit is subsequently asserted.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+8F1H	07:00	R/W	2D Rendering Engine Control
	<i>Bits</i>	<i>Field</i>	
	07	Reset.	
		<i>Value</i>	<i>Semantics</i>
		0	Normal Operation.
		1	Reset, terminates current operation in progress and resets the engine. When deasserted it will be left in a Stopped state. Any pending memory requests will be completed.
	<i>Bits</i>	<i>Field</i>	
	06:02	Reserved.	
	01	Execute/Single Step. The command specified in the flags register is executed. The bit is reset upon completion of one command. This is ignored if the Run is set.	
		<i>Value</i>	<i>Semantics</i>
		0	Idle.
		1	Single Execution.
	<i>Bits</i>	<i>Field</i>	
	00	Start. This bit is sampled only between command execution boundaries. If it is deasserted, then reasserted while a command is executing it will not cause any change.	
		<i>Value</i>	<i>Semantics</i>
		0	Stop (Request). Stop after the end of the current command. <i>Default</i> . This can also be reset by a command issued through the rendering engine FIFO.
		1	Start (Request). Start execution (if any commands are pending in the FIFO).

4.2.1.10 Memory Controller (+900H)

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+910H	15:00	R/W	Internal Refresh Counter Divide Ratio
	<i>Bits</i>	<i>Field</i>	
	15:10	<i>Reserved.</i>	
	09:00	Internal Refresh Rate Divide Ratio.	
		<i>Value</i>	<i>Semantics</i>
		0	Disabled. <i>Default.</i>
		1-1023	$n + 1$, range of 2...1024.
<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+920H	15:00	R/W	External Refresh Counter Divide Ratio (<i>Reserved</i>)
	<i>Bits</i>	<i>Field</i>	
	15:00	<i>Reserved.</i>	
<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+930H	15:00	R/W	Bus Interface Unit Output Clock Delay
	<i>Bits</i>	<i>Field</i>	
	15:06	<i>Reserved.</i>	
	05:04	Bus Interface Unit Output Clock Delay Value.	
		<i>Value</i>	<i>Semantics</i>
		0	No delay. <i>Default.</i>
		3	Largest delay.
	<i>Bits</i>	<i>Field</i>	
	03:00	<i>Reserved.</i>	

Bits [09:00] in +910H and +920H specify the ratio that the input reference clock is divided by, nominally 14.318MHz down to 56KHz, respectively, to generate the refresh requests.

Reference	14.31818	MHz
DRAM Macro	SMC/4Mb	
Rows	256	
Accessed Blocks/Bank	1	
Total Blocks/Bank	16	
Number of Banks/Macro	1	
Number of Macros	8	
Total Rows/Interval	4096	refresh cycles
Relative Peak Refresh Power	8	x 1 block
Minimum Refresh Interval	16	mS
Typical Refresh Interval	32	mS
Maximum Refresh Interval	64	mS
Minimum Request Rate	64	KHz
Maximum Request Rate	256	KHz
Maximum Divide Ratio	224	
Minimum Divide Ratio	56	
Lowest Refresh Rate	56	KHz

4.2.1.11 Surface Descriptors (+A00H)

The following provide a common definition of the base addresses and strides (where appropriate) of surfaces. These surfaces are required for:

- Graphics display surfaces
- Video display surfaces
- Cursor surfaces
- Icon surface
- Graphics rendering surfaces
- Video capture surfaces
- Spares

Sixteen surface descriptors (0-FH) are provided to allow double buffering for control, cursor or other surfaces.

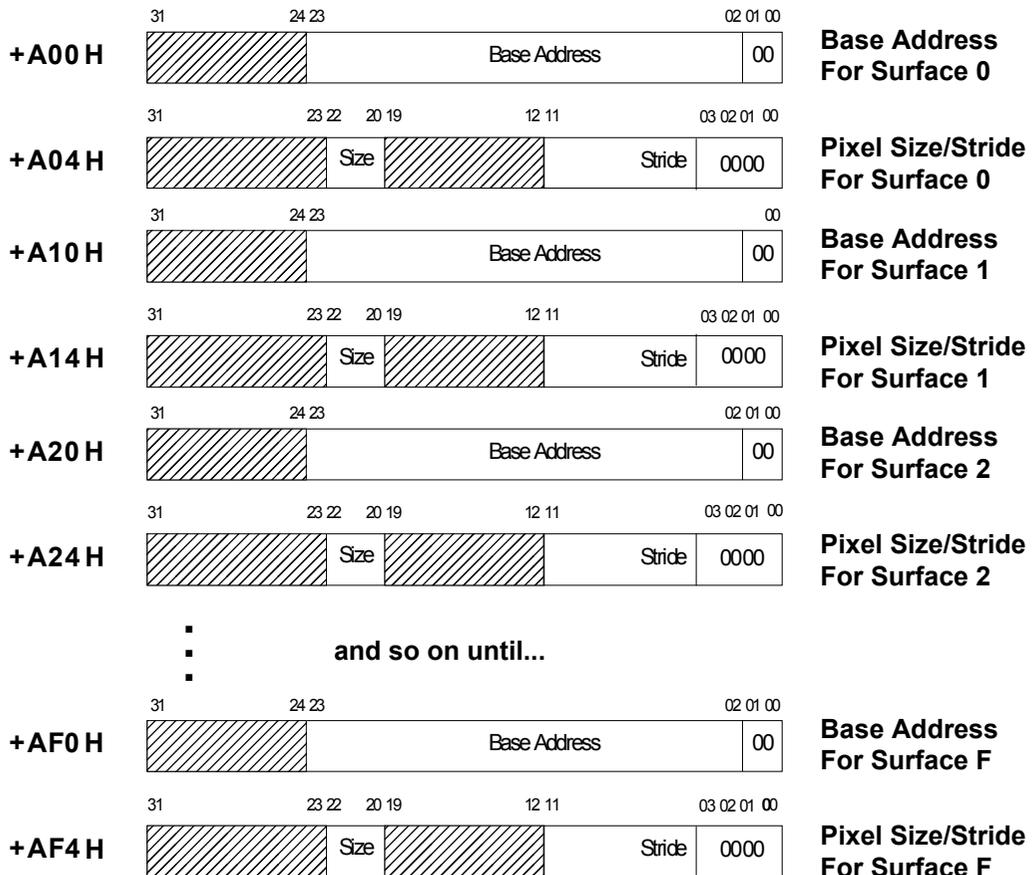


Figure 4-10. Surface Descriptors

4.2.1.11.1 Base Address Descriptor

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+A00H ...+AF0H	31:00	R/W	Surface 0... F Descriptors. Base Address
	<i>Bits</i>	<i>Field</i>	
	31:24	<i>Reserved.</i>	
	23:00	Surface Base Address. Linear Byte address. Surfaces must be aligned to a natural 32b/4B boundary. bits[01:00] must be zero.	

4.2.1.11.2 Pixel size/Stride Descriptor

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+A04H ...+AF4H	31:00	R/W	Surface 0... F Descriptors. Stride/Pixel Size
	<i>Bits</i>	<i>Field</i>	
	31:23	<i>Reserved.</i>	
	22:20	Surface Pixel Size	
		<i>value</i>	<i>semantics</i>
		0	<i>Reserved.</i>
		1	1bpp. Surface stride granularity of 2B. The mono cursor surface stride granularity must be 64B.
		2	2bpp. Surface stride granularity of 4B.
		3	4bpp. Surface stride granularity of 8B.
		4	8bpp. Surface stride granularity of 16B.
		5	16bpp. Surface stride granularity of 32B.
		6	24bpp. Surface stride granularity of 48B.
		7	32bpp. Surface stride granularity of 64B.
	<i>Bits</i>	<i>Field</i>	
	19:12	<i>Reserved.</i>	
	11:00	Stride in pixels. Range from 0 to 4080 in multiples of 16. Bits[03:00] must be zero.	

4.2.1.12 Channel Descriptors (+B00H)

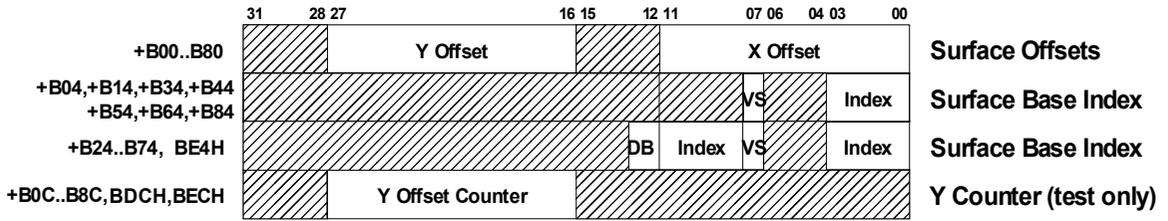


Figure 4-11. Channel Offset/Index Descriptors

4.2.1.12.1 Surface Offsets

Offset	Field	Access	Function
+B00H	31:00	R/W	Surface Offset for Cursor 0 Channel
+B10H	31:00	R/W	Surface Offset for Icon Channel
+B20H	31:00	R/W	Surface Offset for Scaler 0 Channel
+B30H	31:00	R/W	Surface Offset for Display 0 Channel
+B60H	31:00	R/W	Surface Offset for Cursor 1 Channel
+B70H	31:00	R/W	Surface Offset for Scaler 1 Channel
+B80H	31:00	R/W	Surface Offset for Display 1 Channel

Bits	Field
31:28	Reserved.
27:16	Y Offset in pixel/line units.
15:12	Reserved.
11:00	X Offset in pixel units.

4.2.1.12.2 Channel Descriptor Index

This set of registers specifies the index into the surface descriptor array for each channel.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+B04H	31:00	R/W	Cursor 0 Channel/Surface Descriptor Index
+B14H	31:00	R/W	Icon Channel/Surface Descriptor Index
+B34H	31:00	R/W	Display 0 Channel/Surface Descriptor Index
+B64H	31:00	R/W	Cursor 1 Channel/Surface Descriptor Index
+B84H	31:00	R/W	Display 1 Channel/Surface Descriptor Index

<i>Bits</i>	<i>Field</i>
31:08	<i>Reserved.</i>
07	Vsync Selection (Only applied to back end display channel descriptors)
	<i>Value Semantics</i>
0	Display 0 Vsync. <i>Default.</i>
1	Display 1 Vsync

<i>Bits</i>	<i>Field</i>
06:04	<i>Reserved.</i>
03:00	Index.
	<i>Value Semantics</i>
0-FH	Index to surface base register array.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+B24H	31:00	R/W	Scaler 0 Channel/Surface Descriptor Index
+B74H	31:00	R/W	Scaler 1 Channel/Surface Descriptor Index
+BE4H	31:00	R/W	Video Capture Channel Surface Descriptor Index

<i>Bits</i>	<i>Field</i>
31:14	<i>Reserved.</i>
13:12	Double buffering modes for video capture
	<i>Value Semantics</i>
0	Always use buffer 0 index for display/capture
1	Always use buffer 1 index for display/capture
2	Alternating buffer 0 and buffer 1 indices for display/capture
3	<i>Reserved.</i>

<i>Bits</i>	<i>Field</i>
11:08	Buffer 2 Index
	<i>Value Semantics</i>
0-FH	Index to surface base register array.

<i>Bits</i>	<i>Field</i>
07	Vsync Selection (Only applied to back end display channel descriptors)
	<i>value semantics</i>
0	Display 0 Vsync. <i>Default.</i>
1	Display 1 Vsync

<i>Bits</i>	<i>Field</i>
06:04	<i>Reserved.</i>
03:00	Buffer 1 Index
	<i>Value Semantics</i>
0-FH	Index to surface base register array.

4.2.1.12.3 Y Offset Counters

This is the current value of the surface Y offset counter that is incremented every horizontal line and reinitialized at the beginning of every frame.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+B0CH	31:00	R/O	Surface Offset Counter for Cursor 0 Channel
+B1CH	31:00	R/O	Surface Offset Counter for Icon Channel
+B2CH	31:00	R/O	Surface Offset Counter for Scaler 0 Channel
+B3CH	31:00	R/O	Surface Offset Counter for Display 0 Channel
+B6CH	31:00	R/O	Surface Offset Counter for Cursor 1 Channel
+B7CH	31:00	R/O	Surface Offset Counter for Scaler 1 Channel
+B8CH	31:00	R/O	Surface Offset Counter for Display 1 Channel
+BDCH	31:00	R/O	Surface Offset Counter for Linear BLT
+BECH	31:00	R/O	Surface Offset Counter for Video Capture

<i>Bits</i>	<i>Field</i>
31:28	<i>Reserved.</i>
27:16	Y Offset Counter in pixel/line units.
15:00	<i>Reserved.</i>

4.2.1.13 Buffer Descriptors (+C00H)

The following registers provide a common definition of the base addresses, sizes and pointers of the DMA and FIFO buffers.

The base address of each buffer must be initialized before the rendering engine can start execution. The base address must be an even multiple of the buffer size.

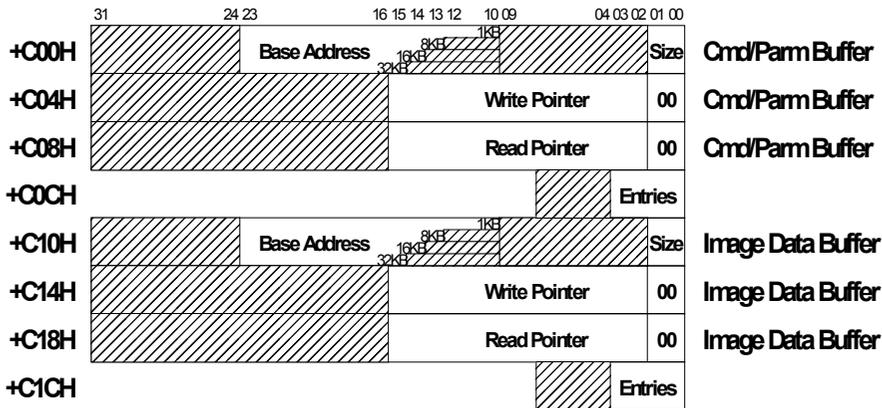


Figure 4-12. Buffer Descriptors

4.2.1.13.1 Buffer Base Address

Offset	Field	Access	Function
+C00H	31:00	R/W	Command/Parameter Buffer Base Address
+C10H	31:00	R/W	Image Data Buffer Base Address
	Bits	Field	
	31:24	Reserved.	
	23:10	Base Address. bits [23:10] of the Base Address in bytes.	
	09:02	Reserved.	
	01:00	Buffer Size.	
	Value	Semantics	
	0	1KB.	
	1	8KB.	
	2	16KB.	
	3	32KB.	

The base addresses of each buffer must be set before writing any parameters/data or enabling the Rendering Engine. The read and write pointers should also be initialized to the same value (normally 0, the start of the buffer) to indicate that the FIFO is empty. After initialization, they should normally not be accessed, except for context switching or error recovery.

4.2.1.13.2 Buffer Pointers

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+C04H	31:00	R/W	Command/Parameter Buffer Write Pointer
+C08H	31:00	R/W	Command/Parameter Buffer Read Pointer
+C14H	31:00	R/W	Image Data Buffer Write Pointer
+C18H	31:00	R/W	Image Data Buffer Read Pointer

<i>Bits</i>	<i>Field</i>
31:16	<i>Reserved.</i>
15:00	Address Pointer. In bytes, must be aligned to a natural doubleword boundary. bits[01:00] must be zero.

The relationship between buffer size and valid bits of the base address and read/write pointers is shown in the table below:

<i>[01:00]</i>	<i>Size</i>	<i>Valid Bits</i>			
		<i>Base Address</i>	<i>Pointer</i>	<i>1/8</i>	<i>Status Subtract</i>
0	1KB	23:10	09:02	128	09:07
1	8KB	23:13	12:02	1K	12:10
2	16KB	23:14	13:02	2K	13:11
3	32KB	23:15	14:02	4K	14:12

Table 4-3. Cmd/Parm, Image Data Buffer

4.2.1.13.3 Engine FIFO Status

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+C0CH	07:00	R/O	Command/Parameter FIFO
+C1CH	07:00	R/O	Image Data FIFO

<i>Bits</i>	<i>Field</i>
07:04	<i>Reserved.</i>
03:00	FIFO Entries Available, in eighths of the total allocated FIFO size (1K, 8K, 16K, 32K).

<i>Value</i>	<i>Semantics</i>
8-0	Eighths (128B, 1KB, 2KB, 4KB corresponding to the above FIFO sizes). The value ranges from 1/8 to 8/8.
15-9	<i>Reserved.</i>

4.2.1.13.4 Cursor/Display Parameter Record/Replay Control

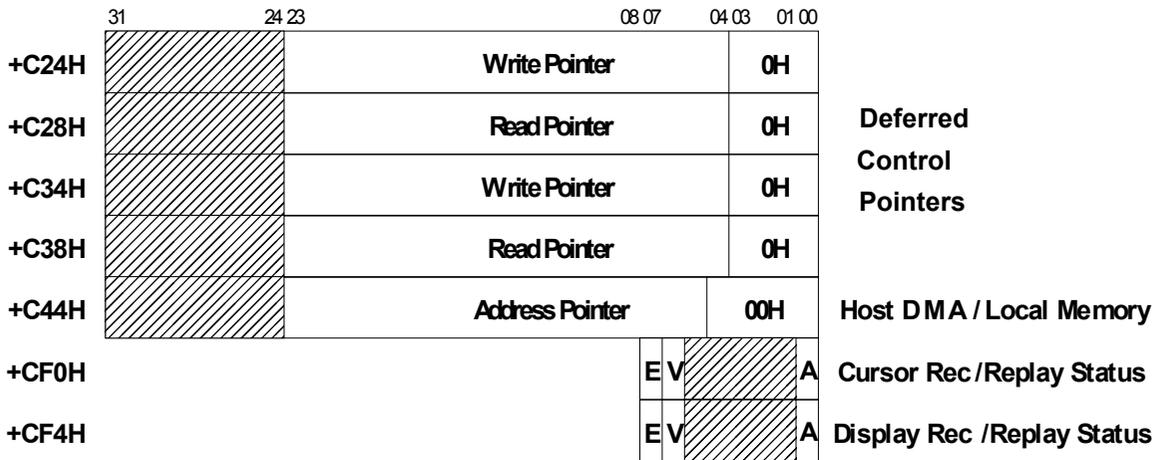


Figure 4-13. Record/Replay Pointers

4.2.1.13.5 Deferred Control Pointers

Offset	Field	Access	Function
+C24H	31:00	R/W	Deferred Control (Write) Pointer 0
+C28H	31:00	R/W	Deferred Control (Read) Pointer 0 Triggered by the leading edge of VBI.
+C34H	31:00	R/W	Deferred Control (Write) Pointer 1
+C38H	31:00	R/W	Deferred Control (Read) Pointer 1 Triggered by the leading edge of VBI after the Deferred Control 0 replay operation.
	<i>Bits</i>	<i>Field</i>	
	31:24	<i>Reserved.</i>	
	23:00	Address. in bytes, must be aligned to a natural quadword boundary. bits[03:00] must be zero.	

4.2.1.13.6 Host DMA Bus Master Local Pointer

This is the local pointer for the Host DMA Bus Master. See registers at +200H.

Offset	Field	Access	Function
+C44H	31:00	R/W	Host DMA Local Memory Address Pointer
	<i>Bits</i>	<i>Field</i>	
	31:24	<i>Reserved.</i>	
	23:00	Address. in bytes, must be aligned to a natural 32 byte boundary. bits[04:00] must be zero.	

4.2.1.13.7 Deferred Control Status

Two byte registers are provided to control the operation of recording (write) and replay (read) of the cursor and display parameters. There are two read and two write pointers associated with the DMA that must also be initialized whenever the record operation is restarted.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+CF0H	07:00	R/W	Deferred Control 0 Status
+CF4H	07:00	R/W	Deferred Control 1 Status
	<i>Bits</i>	<i>Field</i>	
	07	Read.	Replay Enabled. A replay operation is pending. <i>Reset when read.</i>
		<i>Value</i>	<i>Semantics</i>
		0	The last posted Replay has been completed. The read and write pointers must be reinitialized before new parameters can be recorded.
		1	The last posted Replay is still pending. The read and write pointers must not be changed.
	<i>Bits</i>	<i>Field</i>	
	07	Write.	Enable Replay. A replay operation is pending.
		<i>Value</i>	<i>Semantics</i>
		0	Do not load parameters.
		1	Enable loading of parameters at the beginning of the next VBI.
	<i>Bits</i>	<i>Field</i>	
	06	VBI Selection	
		<i>Value</i>	<i>Semantics</i>
		0	Triggered by Display 0 VBI. <i>Default.</i>
		1	Triggered by Display 1 VBI.
	<i>Bits</i>	<i>Field</i>	
	05:01	<i>Reserved.</i>	
	00	Read/Only.	Replay Active.
		<i>Value</i>	<i>Semantics</i>
		0	Host access permitted. Either the posted parameter load operation has not been executed yet or it has been completed.
		1	Host access restricted. The posted operation is in progress.

4.2.1.14 Display Control (+D00H)

4.2.1.14.1 Channels

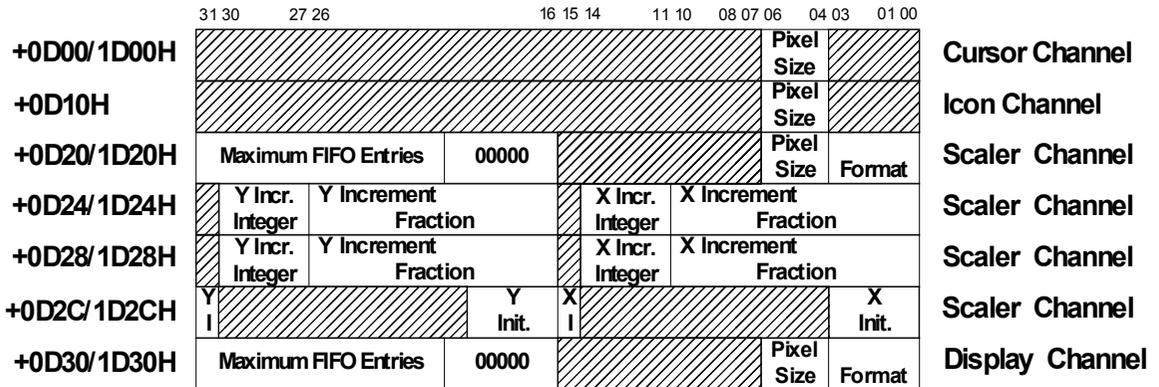


Figure 4-14. Channel Descriptors

Cursor Channels

Offset	Field	Access	Function
+0D00H	31:00	R/W	Cursor 0 Channel
+1D00H	31:00	R/W	Cursor 1 Channel
	Bits	Field	
	31:07	Reserved.	
	06:04	Channel Pixel Size. This should be set to match the pixel size of the surface being processed.	
	Value	Semantics	
	0	Disabled. Takes effect immediately.	
	1	1bpp.	
	2-4	Reserved.	
	5	16bpp.	
	6-7	Reserved.	
	Bits	Field	
	03:00	Reserved.	

Icon Channel

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0D10H	31:00	R/W	Icon Channel
	<i>Bits</i>	<i>Field</i>	
	31:07	<i>Reserved.</i>	
	06:04	Channel Pixel Size. This should be set to match the pixel size of the surface being processed.	
		<i>Value</i>	<i>Semantics</i>
		0	Disabled. Takes effect immediately.
		1	1bpp.
		2-4	<i>Reserved.</i>
		5	16 bpp
		6-7	<i>Reserved.</i>
	<i>Bits</i>	<i>Field</i>	
	03:00	<i>Reserved.</i>	

Scaler Channels

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0D20H	31:00	R/W	Scaler 0 Channel Format
+1D20H	31:00	R/W	Scaler 1 Channel Format
	<i>Bits</i>	<i>Field</i>	
31:21	Maximum number of FIFO requests per-scan line. Each FIFO entry contains 32B data.		
	<i>Value</i>	<i>Semantics</i>	
	0	Disabled. <i>Default.</i>	
	<i>n</i>	<i>n</i> number of FIFO requests. This value is precomputed by software driver based on scaler format, stretch ratio and stride information	
	<i>Bits</i>	<i>Field</i>	
20:16	<i>Reserved.</i> Must be zero.		
15:07	<i>Reserved.</i>		
06:04	Channel Pixel Size. This should be set to match the pixel size of the surface being processed.		
	<i>Value</i>	<i>Semantics</i>	
	0	Disabled. takes effect immediately.	
	1-4	<i>Reserved</i>	
	5	16bpp. for interleaved 16b RGB formats.	
	6	24bpp.	
	7	<i>Reserved</i>	
	<i>Bits</i>	<i>Field</i>	
03:00	Channel Pixel Format. This should be set to match the pixel format of the surface being displayed.		
	<i>Value</i>	<i>Semantics</i>	
16bpp	0H	16bpp RGB 1.5.5.5.	
	1H	16bpp xRGB 5.6.5.	
	2-7H	<i>Reserved.</i>	
16bpp	8H	16bpp YUV/CbCr 4.2.2 Interleaved. YUYV	
	9H	16bpp YUV/CbCr 4.2.2 Interleaved. UYVY	
	A-FH	<i>Reserved.</i>	
24bpp	0H	24bpp RGB 8.8.8.	
	1-FH	<i>Reserved.</i>	

Scaler Channels (continued)

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0D24H	31:00	R/W	1st Scaler control, Y/R Channel Increment Values
+1D24H	31:00	R/W	2nd Scaler control, Y/R Channel Increment Values
+0D28H	31:00	R/W	1st Scaler control, U/V, G/B Channel Increment Values
+1D28H	31:00	R/W	2nd Scaler control, U/V, G/B Channel Increment Values

<i>Bits</i>	<i>Field</i>
31	<i>Reserved.</i>
30:27	Y Increment. Integer part.
26:16	Y Increment. Fractional part.
<i>bits</i>	<i>field</i>
15	<i>Reserved.</i>
14:11	X Increment. Integer part.
10:00	X Increment. Fractional part.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0D2CH	31:00	R/W	1st Scaler control
+1D2CH	31:00	R/W	2nd Scaler control

<i>Bits</i>	<i>Field</i>
31	Y Interpolation Control
	<i>Value Semantics</i>
	0 Previous Neighbor
	1 Bilinear
<i>Bits</i>	<i>Field</i>
30:20	<i>Reserved</i>
19:16	Initial Y value of U/V sub-channels, the sub-(fractional) pixel offset within the source plane.
15	X Interpolation Control
	<i>Value Semantics</i>
	0 Previous Neighbor
	1 Bilinear
<i>Bits</i>	<i>Field</i>
14:04	<i>Reserved</i>
03:00	Initial X value of U/V sub-channels, the sub-(fractional) pixel offset within the source plane.

Display Channels

Offset	Field	Access	Function
+0D30H	31:00	R/W	1st Display Channel Format
+1D30H	31:00	R/W	2nd Display Channel Format

Bits	Field	Value	Semantics
31:21			Maximum number of FIFO requests per-scan line. Each FIFO entry contains 32B data.

0	Disabled. <i>Default.</i>
<i>n</i>	<i>n</i> number of FIFO requests. This value is computed by software driver based on display format and stride information.

Bits	Field	Value	Semantics
20:16			<i>Reserved.</i> Must be zero.
15:07			<i>Reserved.</i>
06:04			Channel Pixel Size. This should be set to match the pixel size of the surface being processed.

Value	Semantics
0	Disabled. Takes effect immediately.
1-2	<i>Reserved.</i>
3	4bpp.
4	8bpp.
5	16bpp.
6	24bpp.
7	32bpp.

Bits	Field	Value	Semantics
03:00			Channel Pixel Format. This should be set to match the pixel format of the surface being displayed.

	Value	Semantics
4bpp	0	4bpp CLUT.
	1-7	<i>Reserved.</i>
8bpp	0	8bpp CLUT.
	1-7	<i>Reserved.</i>
16bpp	0	16bpp xRGB 1.5.5.5.
	1	16bpp RGB 5.6.5.
	2-7	<i>Reserved.</i> f
24bpp	0	24bpp RGB 8.8.8.
	1-7	<i>Reserved.</i>
32bpp	0	32bpp RGB x.8.8.8.
	1-7	<i>Reserved.</i>

4.2.1.14.2 Overlay/Mix Control

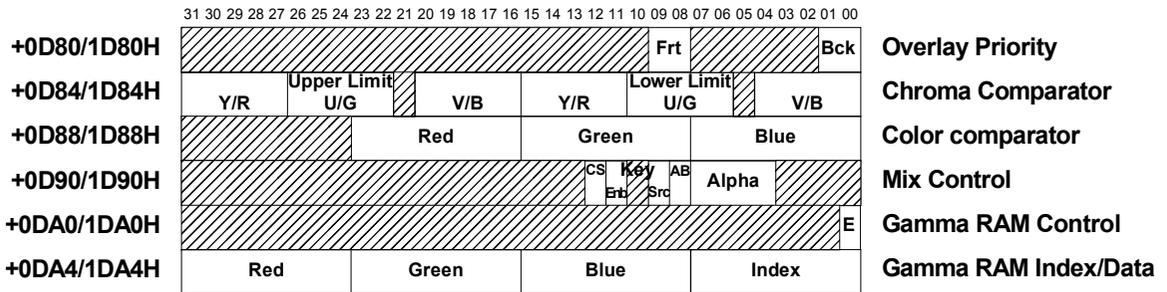


Figure 4-15. Overlay and Mix Controls

Overlay Priority Control

These registers define the priority of overlay when composing the result.

Offset	Field	Access	Function
+0D80H	31:00	R/W	1st Scaler/Display Overlay Priority Specification
+1D80H	31:00	R/W	2nd Scaler/Display Overlay Priority Specification
	<i>Bits</i>	<i>Field</i>	
	31:10	<i>Reserved.</i>	
	09:08	Foreground. Highest Priority Overlay	
		<i>Value</i>	<i>Semantics</i>
		0	Scaler Channel
		1	Display Channel
		2-3	<i>Reserved.</i>
	<i>Bits</i>	<i>Field</i>	
	07:02	<i>Reserved.</i>	
	01:00	Background. Lowest Priority Overlay/Primary Surface.	
		<i>Value</i>	<i>Semantics</i>
		0	Scaler Channel
		1	Display Channel
		2-3	<i>Reserved.</i>

Chroma Comparator

This register specifies the luma/chroma (YUV) values to be compared against the pixel stream for the corresponding surface. Luma/chroma comparison is performed on the output of the scaler (before the color space converter). A magnitude comparison is performed between output of the scaler against the upper and lower limits. Only 5 bits of Y, U and V are compared. If the comparisons of all three components against the set limits are satisfied the compare output is '1' and can be used to control chroma/color keying.

$$K = (P_Y \leq UL_Y) \text{ AND } (P_Y \geq LL_Y) \text{ AND } (P_U \leq UL_U) \text{ AND } (P_U \geq LL_U) \text{ AND } (P_V \leq UL_V) \text{ AND } (P_V \geq LL_V)$$

The comparison of any component can be disabled by setting the upper and lower limits to their maximum and minimum values, respectively, effectively matching all values. Normally the Y component is not compared. Setting the upper limit less than the lower limit for any component effectively disables the entire comparison.

The chroma comparator output is automatically used when keying is enabled for a YUV format surface.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0D84H	31:00	R/W	Scaler 0 Channel Luma/Chroma Comparator
+1D84H	31:00	R/W	Scaler 1 Channel Luma/Chroma Comparator
	<i>Bits</i>	<i>Field</i>	
	31:27	Y/R Upper Limit.	
	26:22	U/G Upper Limit.	
	21	Reserved.	
	20:16	V/B Upper Limit.	
	15:11	Y/R Lower Limit.	
	10:06	U/G Lower Limit.	
	05	Reserved.	
	04:00	V/B Lower Limit.	

Color Comparator

Color comparison is performed using an equality comparator with a generated bit mask that selects the precision of the comparison.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0D88H	31:00	R/W	Display 0 Color Comparator
+1D88H	31:00	R/W	Display 1 Color Comparator
	<i>Bits</i>	<i>Field</i>	
	31:24	Reserved.	
	23:16	Red Component.	
	15:08	Green Component.	
	07:00	Blue Component.	

Mix Control

These registers define the type of operations performed when composing the result. If the source or destination specified for the color comparison is a YUV surface, the chroma comparator result is used in place of the color comparator. *Key control* is valid only where two surfaces (in one mix operation) overlap.

Only one type of control source is allowed for each mix operation: color/chroma keying, control surface keying or blend or default blend or surface selection.

Offset	Field	Access	Function
+0D90H	31:00	R/W	Display 0 Mix Control
+1D90H	31:00	R/W	Display 1 Mix Control
	Bits	Field	
	31:12		Reserved.
	11		Color Key
		Value	Semantics
		0	Disabled. <i>Default.</i>
		1	Enabled.
	Bits	Field	
	10		Reserved.
	09		Key Source
		Value	Semantics
		0	Color Key Compare against Source/Foreground Channel. <i>Default</i>
		1	Color Key Compare against Destination/Background Channel.
	Bits	Field	
	08		Blend using constant Alpha value, bits [07:04]
		Value	Semantics
		0	Disable. <i>Default</i>
		1	Enable.
	Bits	Field	
	07:04		Alpha Blend Constant
		Value	Semantics
		0H	Destination/Background only. <i>Default</i>
		1-7H	7% steps from Destination only
		8H	50% Source plus 50% Destination
		9-EH	7% steps to Source only
		FH	Source/Foreground only
	Bits	Field	
	03:00		Reserved. Must be zero.

Note: If the scaler channel is selected for color key comparison, and the format of the surface in the channel is YUV, the chroma comparator (before the color space converter) is used instead of the foreground or background color comparator.

Gamma Correction

The characteristics of displays and cameras are nonlinear. A small change in voltage when the voltage level is low produces a change in the output display brightness level, but the same small change in voltage at a high voltage level will not produce the same magnitude of change in the brightness output. This effect is known as gamma. The linear RGB data must be processed to compensate for the gamma of the display. Three 256x8 RAMs, one for each color component, is used to implement gamma correction.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0DA0H	31:00	R/W	Display 0 Gamma RAM Control
+1DA0H	31:00	R/W	Display 1 Gamma RAM Control
	<i>Bits</i>	<i>Field</i>	
	31:01	<i>Reserved.</i>	
	00	Gamma Correction Enable	
		<i>Value</i>	<i>Semantics</i>
		0	Disabled/Bypass. <i>Default.</i>
		1	Enabled.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0DA4H	31:00	R/W	Display 0 Gamma RAM Index/Data
+1DA4H	31:00	R/W	Display 1 Gamma RAM Index/Data
	<i>Bits</i>	<i>Field</i>	
	31:24	Red Color Component	
	23:16	Green Color Component	
	15:08	Blue Color Component	
	07:00	Index	

4.2.1.14.3 Display Status/Control

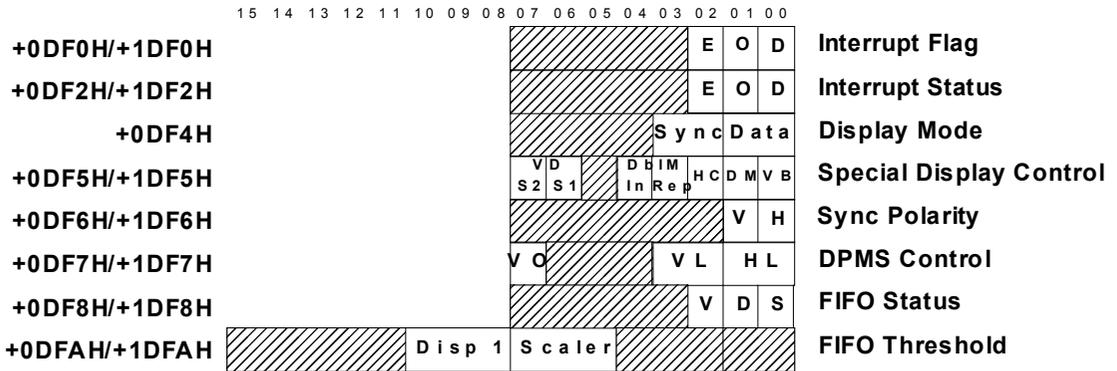


Figure 4-16. Display Control

Display Interrupt/Status

Offset	Field	Access	Function
+0DF0H	07:00	R/W	Display 0 Interrupt Flag
+1DF0H	07:00	R/W	Display 1 Interrupt Flag
+0DF2H	07:00	R/O	Display 0 Status. This returns the actual level of the pin or signal and is provided to allow real-time polling of the state of vertical blanking and the external vertical sync pin.
+1DF2H	07:00	R/O	Display 1 Status. This returns the actual level of the pin or signal and is provided to allow real-time polling of the state of vertical blanking and the external vertical sync pin.

Bits	Field
07:02	Reserved.
01	V- External Video: Frame Vertical Sync derived from VVS pin on Video port, nominally 30Hz rate.
00	D - Display Vertical Enable

	Value	Semantics
+XDF0H	0	Read. Sync/Blank event has not occurred.
+XDF0H	1	Read. Sync/Blank event has occurred.
+XDF0H	0	Write. No change.
+XDF0H	1	Write. Reset Corresponding Flag Bit.

	Value	Semantics
+XDF2H	0	Read. Input is not asserted/Display is inactive.
+XDF2H	1	Read. Input is asserted/Display is active.

Display Mode Set Control

These registers are loaded during mode set prior to enabling the display.

These registers control output to the physical LCD Controller and CRT Controller. The Back End VGA timing and data are inputs to the Display 0 Control, bits 3:2 and 1:0 select whether the output of the Display 0 Control is connected to the VGA or to its own output and timing. They should be set in the same way. Bit 4 selects whether the CRT takes its data and timing from the Display 0 Control or from the Display 1 Control. The Display 0 Control is not connected to the VGA, so the corresponding control bits effectively select between disable and its own output and timing.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0DF4H	07:00	R/W	Display 0 Control. Default is VGA
	<i>Bits</i>	<i>Field</i>	
	07	Reset Display State Machine on VSync	
		<i>Value</i>	<i>Semantics</i>
		0	Reset Display State Machine at Vertical Sync. <i>Default.</i>
		1	Do not Reset Display State Machine.
	<i>Bits</i>	<i>Field</i>	
	06:05	<i>Reserved.</i>	
	04	CRT Display Select	
		<i>Value</i>	<i>Semantics</i>
		0	CRT is connected to VGA/ECRTC0.
		1	CRT is connected to ECRTC1
	<i>Bits</i>	<i>Field</i>	
	03:02	Enhanced Display Timing. Selects the source of display timing signals.	
		<i>Value</i>	<i>Semantics</i>
		0	VGA CRTC. Enhanced CRTC is disabled. <i>Default.</i>
		1	Enhanced CRTC.
		2-3	<i>Reserved.</i>
	<i>Bits</i>	<i>Field</i>	
	01:00	Enhanced Display Output. Selects the source of display data.	
		<i>Value</i>	<i>Semantics</i>
		0	Enable VGA. <i>Default.</i>
		1	Enable Enhanced Display Processor.
		2-3	<i>Reserved.</i>

Offset	Field	Access	Function
+1DF4H	07:00	R/W	Display 1 Control. <i>Default</i> is disabled.
	Bits	Field	
	07		Reset Display State Machine on VSync
		Value	Semantics
		0	Reset Display State Machine at Vertical Sync. <i>Default.</i>
		1	Do not Reset Display State Machine.
	Bits	Field	
	06:04		<i>Reserved.</i>
	03:02		Enhanced Display Timing. Selects the source of display timing signals.
		Value	Semantics
		0	Enhanced CRTC is disabled. <i>Default.</i>
		1	Enhanced CRTC.
		2-3	<i>Reserved.</i>
	Bits	Field	
	01:00		Enhanced Display Output. Selects the source of display data.
		Value	Semantics
		0	Enhanced CRTC is disabled. <i>Default.</i>
		1	Enable Enhanced Display Processor.
		2-3	<i>Reserved.</i>

Special Display Control

Offset	Field	Access	Function
+0DF5H	07:00	R/W	Doublescan Display 0 Control.
+1DF5H	07:00	R/W	Doublescan Display 1 Control.
	Bits	Field	
	07:06		Display Channel Double (vertical) Scan.
		Value	Semantics
		0	Disabled (1×). <i>Default.</i>
		1-2	<i>Reserved.</i>
		3	Enabled (2×).
	Bits	Field	
	05:04		Display Channel Double (horizontal) Scan.
		Value	Semantics
		0	Disabled (1×). <i>Default</i>
		1-2	<i>Reserved.</i>
		3	Enabled (2×).
	Bits	Field	
	03:01		<i>Reserved</i>
	00		Scaler Double (horizontal) with interpolation.

Enhanced Sync Polarity

The polarity of the output sync signals are defined in one register.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0DF6H	07:00	R/W	Display 0 Enhanced Sync Polarity
+1DF6H	07:00	R/W	Display 1 Enhanced Sync Polarity
	<i>Bits</i>	<i>Field</i>	
	07:02	<i>Reserved.</i>	
	01	Enhanced Vertical Sync Polarity.	
	00	Enhanced Horizontal Sync Polarity.	
		<i>Value</i>	<i>Semantics</i>
		0	Positive.
		1	Negative.

Display Power Management Control

These registers contain most of the power management control bits for the display and display sub-system. They override the polarity and value of the sync signals independent of which CRTC/Display Timing Generator is selected. This allows monitor power management to be enabled without affecting the state of either sync generator or the display sub-system.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0DF7H	07:00	R/W	Display 0 Power Management
+1DF7H	07:00	R/W	Display 1 Power Management
	<i>Bits</i>	<i>Field</i>	
	07	Power Down DAC. Saves approximately 57mA for power management.	
		<i>Value</i>	<i>Semantics</i>
		0	Enabled, Normal Operation. <i>Default.</i>
		1	Force DAC outputs to 0. Asserts PD# to DAC.
	<i>Bits</i>	<i>Field</i>	
	06:04	Reserved	
	03:02	Vertical Sync Level.	
	01:00	Horizontal Sync Level.	
		<i>Value</i>	<i>Semantics</i>
		0	Normal Operation. <i>Default</i>
		1	Force to 0.
		2	Force to 1.
		3	<i>Reserved.</i>

Display FIFO Status

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0DF8H	07:00	R/W	Display 0 FIFO Status
	<i>Bits</i>	<i>Field</i>	
	07:03	<i>Reserved.</i>	
	02	Video capture FIFO overflow, Write.	
		<i>Value</i>	<i>Semantics</i>
		0	<i>Reserved.</i>
		1	Clear FIFO overflow status bit.
	<i>Bits</i>	<i>Field</i>	
	02	Video capture FIFO overflow, Read.	
		<i>Value</i>	<i>Semantics</i>
		0	FIFO is operating normally.
		1	FIFO overflow. This bit is set by hardware
	<i>Bits</i>	<i>Field</i>	
	01	Display channel FIFO overflow, Write.	
		<i>Value</i>	<i>Semantics</i>
		0	<i>Reserved.</i>
		1	Clear FIFO overflow status bit.
	<i>Bits</i>	<i>Field</i>	
	01	Display channel FIFO overflow, Read.	
		<i>Value</i>	<i>Semantics</i>
		0	FIFO is operating normally.
		1	FIFO overflow. This bit is set by hardware
	<i>Bits</i>	<i>Field</i>	
	00	Scaler channel FIFO overflow, Write.	
		<i>Value</i>	<i>Semantics</i>
		0	<i>Reserved.</i>
		1	Clear FIFO overflow status bit.
	<i>Bits</i>	<i>Field</i>	
	00	Scaler channel FIFO overflow, Read.	
		<i>Value</i>	<i>Semantics</i>
		0	FIFO is operating normally.
		1	FIFO overflow. This bit is set by hardware

Offset	Field	Access	Function
+1DF8H	07:00	R/W	Display 1 FIFO Status
	Bits	Field	
	07:02	<i>Reserved.</i>	
	01	Display channel FIFO overflow, Write.	
		Value	Semantics
		0	<i>Reserved.</i>
		1	Clear FIFO overflow status bit.
	Bits	Field	
	01	Display channel FIFO overflow, Read.	
		Value	Semantics
		0	FIFO is operating normally.
		1	FIFO overflow. This bit is set by hardware
	Bits	Field	
	00	Scaler channel FIFO overflow, Write.	
		Value	Semantics
		0	<i>Reserved.</i>
		1	Clear FIFO overflow status bit.
	Bits	Field	
	00	Scaler channel FIFO overflow, Read.	
		Value	Semantics
		0	FIFO is operating normally.
		1	FIFO overflow. This bit is set by hardware

Display/Scaler FIFO Control

This specifies the configuration (size and threshold) of the display and scaler FIFOs.

Offset	Field	Access	Function
+0DFAH	15:00	R/W	Display 0 FIFO (lower) Thresholds.
+1DFAH	15:00	R/W	Display 1 FIFO (lower) Thresholds.
	Bits	Field	
	15:11	<i>Reserved.</i>	
	10:08	Display Channel FIFO (lower) Threshold	
		Value	Semantics
		0-7	Entries remaining . <i>Default</i> is 4 entries. Value + 1 = remaining entries.
	Bits	Field	
	07:05	Scaler Y Channel FIFO (lower) Threshold	
		Value	Semantics
		0-4	Entries remaining . <i>Default</i> is 4 entries. Value + 1 = remaining entries.
		5-7	<i>Reserved.</i>
	Bits	Field	
	04:00	<i>Reserved.</i>	

Palette Read/Write State

This register reflects the state of the two-bit (three-state) address counter that sequences through the three color components of each palette entry. This should be saved if an interrupt occurs and the palette is changed due to an interrupt. It can later be restored to properly resume the palette access sequence.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0DFCH	07:00	R/W	Display 0 Palette Read/Write State
+1DFCH	07:00	R/W	Display 1 Palette Read/Write State

<i>Bits</i>	<i>Field</i>
07:02	<i>Reserved.</i>
01:00	<i>Address Cycle.</i>

<i>Value</i>	<i>Semantics</i>
0	Idle, third access (blue) completed.
1	First access (red) completed.
2	Second access (green) completed.
3	<i>Reserved.</i>

4.2.1.15 Display Viewports and Timing (+E00H)

There are two sets of display timing control registers: Enhanced CRTC and Display Viewports.

	31	27	26	16	15	11	10	00	
+0E00/1E00H	Y Start Coordinate			X Start Coordinate					Cursor Viewport Start (UL)
+0E04/1E04H	Y End Coordinate			X End Coordinate					Cursor Viewport End (LR)
+0E10/1E10H	Y Start Coordinate			X Start Coordinate					Icon Viewport Start (UL)
+0E14/1E14H	Y End Coordinate			X End Coordinate					Icon Viewport End (LR)
+0E20/1E20H	Y Start Coordinate			X Start Coordinate					Scaler Viewport Start (UL)
+0E24/1E24H	Y End Coordinate			X End Coordinate					Scaler Viewport End (LR)
+0E30/1E30H	Y Start Coordinate			X Start Coordinate					Display Viewport Start (UL)
+0E34/1E34H	Y End Coordinate			X End Coordinate					Display Viewport End (LR)
+0E80/1E80H	V Display End			H Display End			Display End		Display End
+0E84/1E84H	V Blank Start			H Blank Start			Blank Start		Blank Start
+0E88/1E88H	V Sync Start			H Sync Start			Sync Start		Sync Start
+0E8C/1E8CH	V Sync End			H Sync End			Sync End		Sync End
+0E90/1E90H	V Blank End			H Blank End			Blank End		Blank End
+0E94/1E94H	V Total			H Total			Total		Total
+0E9A/1E9AH	V Counter			VP			S		Counters
+0EE0/1EE0H							S		Rendering Sync
+0EE4/1EE4H							Vertical		Stretch Mode

Figure 4-17. Display Viewports and Timing

4.2.1.15.1 Display Channel Viewports

The display channel viewports are defined by the coordinates of the upper left corner of the position on the screen and the lower right corner. The cursor viewport may have a size of 1,1, i.e., the lower right corner may be one pixel to the right and below the upper left corner. The coordinates may not extend beyond the edge of the active addressable display area. Coordinates may not be less than 0; if they exceed the active addressable display area, they will be clipped to the active display area.

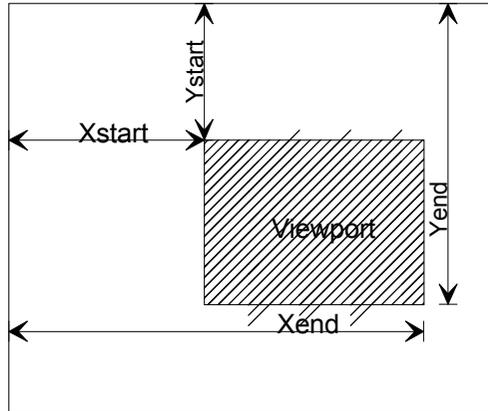


Figure 4-18. Viewport Definition

Start (Upper Left)

The viewport start is inclusive.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0E00H	31:00	R/W	Cursor 0 Channel Viewport Start (Upper Left).
+1E00H	31:00	R/W	Cursor 1 Channel Viewport Start (Upper Left).
+0E10H	31:00	R/W	Icon Channel Viewport Start (Upper Left).
+0E20H	31:00	R/W	Scaler 0 Channel Viewport Start (Upper Left).
+1E20H	31:00	R/W	Scaler 1 Channel Viewport Start (Upper Left).
+0E30H	31:00	R/W	Display 0 Channel Viewport Start (Upper Left).
+1E30H	31:00	R/W	Display 1 Channel Viewport Start (Upper Left).

<i>Bits</i>	<i>Field</i>
31:27	Reserved.
26:16	Upper Y coordinate in pixel/lines. Range from 0 to 2047.
15:11	Reserved.
10:00	Left X coordinate in pixels. Range from 0 to 2047.

End (Lower Right)

The viewport end is exclusive.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0E04H	31:00	R/W	Cursor 0 Channel Viewport End (Lower Right).
+1E04H	31:00	R/W	Cursor 1 Channel Viewport End (Lower Right).
+0E14H	31:00	R/W	Icon Channel Viewport End (Lower Right).
+0E24H	31:00	R/W	Scaler 0 Channel Viewport End (Lower Right).
+1E24H	31:00	R/W	Scaler 1 Channel Viewport End (Lower Right).
+0E34H	31:00	R/W	Display 0 Channel Viewport End (Lower Right).
+1E34H	31:00	R/W	Display 1 Channel Viewport End (Lower Right).

<i>Bits</i>	<i>Field</i>
31:27	<i>Reserved.</i>
26:16	Lower Y coordinate in pixel/lines. Range from 0 to 2047.
15:11	<i>Reserved.</i>
10:00	Right X Coordinate in pixels. Range from 0 to 2047.

Note:

The following restrictions must be followed in specifying the viewport start and end parameters:

1. A viewport (except the cursor) can have a minimum width of 2 characters (16 pixels).
2. The maximum width of a viewport is 255 characters plus 7 pixels = 2047 pixels.
3. The cursor viewport can have a minimum width of 1 pixel.
4. A viewport can have a minimum height of 1 line.
5. The maximum height of a viewport is 2047 lines.

If a viewport has a width or height less than the minimum, the corresponding display channel should be disabled.

4.2.1.15.2 Enhanced CRTC Timing

Vertical Timing

All timing is relative to the start (top; y =0) of the addressable image area.

Offset	Field	Access	Function
+0E82H	15:00	R/W	Display 0 Vertical Display End (of addressable image area)
+1E82H	15:00	R/W	Display 1 Vertical Display End (of addressable image area)
+0E86H	15:00	R/W	Display 0 Vertical Blank Start.(of Front Porch)
+1E86H	15:00	R/W	Display 1 Vertical Blank Start (of Front Porch)
+0E8AH	15:00	R/W	Display 0 Vertical Sync Start
+1E8AH	15:00	R/W	Display 1 Vertical Sync Start
+0E8EH	15:00	R/W	Display 0 Vertical Sync End
+1E8EH	15:00	R/W	Display 1 Vertical Sync End
+0E92H	15:00	R/W	Display 0 Vertical Blank End (of Back Porch)
+1E92H	15:00	R/W	Display 1 Vertical Blank End (of Back Porch)

Bits	Field
15:11	Reserved.
10:00	Delay from start of addressable image area in scan lines.
Value	Semantics
0	Reserved.
1-2047	1...2047 lines

Offset	Field	Access	Function
+0E96H	15:00	R/W	First Vertical Total (of Back Porch).
+1E96H	15:00	R/W	Second Vertical Total (of Back Porch).

Bits	Field
15:11	Reserved.
10:00	Delay from start of addressable image area in scan lines.
Value	Semantics
0	Reserved.
1-2047	$n+1, 2...2048$ lines



Figure 4-19. Display Timing Parameters

Horizontal Timing

The horizontal display timing parameters are defined in character positions counted from the left edge of the active display area (not including the right border). Horizontal Sync Start and End may be specified to pixel resolution to allow fine adjustment of the position of the addressable image area on the display.

The vertical display timing parameters are defined in scan line positions counted from the top edge of the active display area (not including the bottom border).

Restrictions

All timing is relative to the start (left edge) of the addressable image area.

The following restrictions must be observed when setting the enhanced CRTC parameters:

<i>D</i>	<i>sym</i>	<i>Parameter</i>	Actual			Register		<i>comments</i>
			<i>min</i>	<i>max</i>	<i>semantics</i>	<i>min</i>	<i>max</i>	
V	St	VP Start	0	DE	<i>a</i>	0	DE	
V	En	VP End	St+2	DE	<i>a</i>	St+2	DE	
V	DE	Display End		T- <i>n</i>	<i>a</i>			
V	BS	Blank Start	DE+1		<i>a</i>	DE+1		Dot Border is Blank Start- Display End, minimum is 0
V	SS	Sync Start	BS+1		<i>a</i>	BS+1		Front Porch is Sync Start-Blank Start, minimum is 1
V	SE	Sync End	SS+1		<i>a</i>	SS+1		minimum V Sync width is 1
V	BE	Blank End	SE+1		<i>a</i>	SE+1		
V	T	V Total	BE	256	<i>a</i> -1	BE-1	255	Back Porch is Total-Blank End, minimum is 1
H	St	VP Start	0	DE	<i>a</i>	0	DE	
H	En	VP End	St+2	DE	<i>a</i>	St+2	DE	
H	DE	Display End		T- <i>n</i>	<i>a</i>			
H	BS	Blank Start	DE+1		<i>a</i>	DE+1		Right Border is Blank Start-Display End, minimum is 0
H	SS	Sync Start	BS+1		<i>a</i>	BS+1		Front Porch is Sync Start-Blank Start, minimum is 1
H	SE	Sync End	SS+4		<i>a</i>	SS+4		minimum H Sync width is 4
H	BE	Blank End	SE+1		<i>a</i>	SE+1		
H	T	V Total	BE	256	<i>a</i> -1	BE-1	255	Back Porch is Total-Blank End, minimum is 1

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0E80H	15:00	R/W	Display 0 Horizontal Display End (of addressable image area)
+1E80H	15:00	R/W	Display 1 Horizontal Display End (of addressable image area)
+0E84H	15:00	R/W	Display 0 Horizontal Blank Start (of Front Porch)
+1E84H	15:00	R/W	Display 1 Horizontal Blank Start (of Front Porch)
+0E90H	15:00	R/W	Display 0 Horizontal Blank End (of Back Porch)
+1E90H	15:00	R/W	Display 1 Horizontal Blank End (of Back Porch)

<i>Bits</i>	<i>Field</i>
15:11	<i>Reserved.</i>
10:03	Delay from start of addressable image area in character units, (multiples of 8 pixels)
	<i>Value Semantics</i>
	0 <i>Reserved</i>
	1-255 1-255 character times.

<i>Bits</i>	<i>Field</i>
02:00	<i>Reserved.</i> Must be zero.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0E88H	15:00	R/W	Display 0 Horizontal Sync Start
+1E88H	15:00	R/W	Display 1 Horizontal Sync Start
+0E8CH	15:00	R/W	Display 0 Horizontal Sync End Must be at least 4 greater than Sync Start,
+1E8CH	15:00	R/W	Display 1 Horizontal Sync End Must be at least 4 greater than Sync Start,

<i>Bits</i>	<i>Field</i>
15:11	<i>Reserved.</i>
10:03	Delay from start of addressable image area in character units (multiples of 8 pixels)
	<i>Value Semantics</i>
	0 <i>Reserved.</i>
	1-255 1..255 characters

<i>Bits</i>	<i>Field</i>
02:00	Pixel adjustment, additional horizontal delay in pixel units. Allows fine positioning of the display relative to the screen.
	<i>Value Semantics</i>
	0-7 0..7 pixel time additional delay.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0E94H	15:00	R/W	1st Horizontal Total
+1E94H	15:00	R/W	2nd Horizontal Total

<i>Bits</i>	<i>Field</i>
15:11	<i>Reserved.</i>
10:03	Delay from start of addressable image area in character units -1, (multiples of 8 pixels)
	<i>Value Semantics</i>
	0 <i>Reserved.</i>
	1-255 n+1, 2...256 character times.

<i>Bits</i>	<i>Field</i>
02:00	<i>Reserved.</i> Must be zero.

Vertical Counter

This register provides access to the display vertical (scan line) counter for software display synchronization. The display driver may read the current scan line to determine when and what portion of the displayed surfaces may be updated to avoid tearing artifacts.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0E9AH	15:00	R/O	Display 0 Vertical Counter
+1E9AH	15:00	R/O	Display 1 Vertical Counter
	<i>Bits</i>	<i>Field</i>	
	15:11	Reserved.	
	10:00	Count of scan line currently being displayed, incremented at the beginning of each scan line.	
		<i>Value</i>	<i>Semantics</i>
		0	First line of active display.
		1-2047	Line number of display, this continues to increment during the blanking interval and is reset to 0 at the end of vertical blanking.

Rendering Synchronization Selection

This register specifies the source of synchronization request to the 2D rendering engine. This may be the starting or ending line of any of the viewpoints.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0EE0H	07:00	R/W	Display 0 Rendering Sync Selector
+1EE0H	07:00	R/W	Display 1 Rendering Sync Selector
	<i>Bits</i>	<i>Field</i>	
	07:04	Reserved.	
	03	Start/End Select.	
		<i>Value</i>	<i>Semantics</i>
		0H	Start.
		1H	End.
	<i>Bits</i>	<i>Field</i>	
	02:00	Viewport Select	
		<i>Value</i>	<i>Semantics</i>
		0H	Always Enabled. <i>Default.</i> The 2D rendering engine will not wait for synchronization.
		1H	Reserved.
		2H	Scaler.
		3H	Display 1.
		4-7H	Reserved.

Stretch Mode

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0EE4H	07:00	R/W	Display 0 Display Stretch Mode
+1EE4H	07:00	R/W	Display 1 Display Stretch Mode
	<i>Bits</i>	<i>Field</i>	
	07:04	Vertical Stretch Mode	
	03:00	Horizontal Stretch Mode	
		<i>Value</i>	<i>Semantics</i>
		0H	No stretch. <i>Default.</i>
		1H	1 to 2 stretch
		2H	3 to 4 stretch
		3H	4 to 5 stretch
		4H	5 to 8 stretch
		5H	Reserved
		6H	16 to 25 stretch
		7H	25 to 32 stretch
		8H	75 to 128 stretch
		9H-FH	<i>Reserved.</i>

X and Y can independently stretch. No support for 400x300 to 1024x768 or 400x300 to 1280x1024.

Original Format	Stretched Format	xEE4
400x300	640x480	44H
400x300	800x600	11H
512x384	640x480	33H
512x384	800x600	66H
512x384	1024x768	11H
640x480	800x600	33H
640x480	1024x768	44H
640x480	1280x1024	51H
800x600	1024x768	77H
800x600	1280x1024	84H
1024x768	1280x1024	23H

Table 4-4. Stretch Programming

4.2.1.16 PCI/AGP Configuration Registers (+F00H)

The 256-byte PCI Configuration Space is accessible using PCI configuration cycles and is also memory-mapped into the control register space where it can be read directly by the display driver at an offset to the PCI Configuration base of F00H. The first 40H bytes are pre-allocated/defined in the PCI configuration specification. The remaining C0H bytes are vendor/device specific and a few registers are used.

AGP uses a linked list structure within the remaining C0H bytes of the vendor specific register space. This is pointed to by an additional register defined in the standard PCI space and enabled by an additional bit in the status register.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00					
+F00H	Device ID																Vendor ID											Device/Vendor I.D.									
+F04H	PE	SE	MA	RA	SA	Dev Sel	Status				Command											Status/Command															
+F08H	Class Code								Programming Interface				Revision ID				Class Code/Rev. I.D.																				
+F0CH	BIST				Header Type				Latency Timer				00H				Test/Control																				
+F10H	Base Address								0								00	0	RAM Base Address																		
+F2CH	Subsystem Vendor ID											Subsystem ID											Subsystem/Vendor I.D.														
+F30H	ROM Base Address											E				ROM Base Address																					
+F34H	Offset (80H)											Capability List Pointer																									
+F3CH	Max_Lat				Min_Gnt				Interrupt Pin				Interrupt Line				Latency/Grant/Interrupt																				
+F40H	DS											Variant ID				Device Variant																					
+																																					
+F80H	Major											Minor				Pointer to Next				Capability ID				Capability Identifier													
+F84H	Request Depth											S B A				4 G S				R F W S				Rate													
+F88H	Request Depth											S B A				4 G S				R F W S				Data Rate													
+																																					
+FA0H	PME Support				D 2		D 1		In it				A P M				Version				Pointer to Next				Capability ID				Power Management Capability								
+FA4H	Power Consumption Report Data								BP CC en.				Bz Bz				St at us				Data Scale				Data Select				En ab le				Power State				Power Management Control

Figure 4-20. PCI Configuration Registers

Device/Vendor ID

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+F00H	15:00	R/O	Device/Vendor ID
	<i>Bits</i>	<i>Field</i>	
	31:16	Device ID	
		<i>Value</i>	<i>Semantics</i>
		4831H	ASCII for 'H1', SM3110
	<i>Bits</i>	<i>Field</i>	
	15:00	Vendor ID	
		<i>Value</i>	<i>Semantics</i>
		8888H	Silicon Magic.

Command

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+F04H	15:00	R/W	Command
	<i>Bits</i>	<i>Field</i>	
	15:10	<i>Reserved.</i> Read as '0'	
	09	Fast Back to Back Enable. Read as '0'	
	08	SERR#, Read as '0'	
	07	Address Stepping, Read as '0'	
	06	Parity Check, default to '0'	
	05	Palette Snooping, Read as '0'	
	04	Memory Write and Invalidate, Read as '0'	
	03	Special Cycle Support, Read as '0'	
	02	PCI Bus Master, default to '0'	
	01	Memory Space, default to '0'	
	00	I/O Space, default to '0'	

Status

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+F06H	15:00	R/W	Status
	<i>Bits</i>	<i>Field</i>	
	15	Detected Parity Error. <i>default to '0'</i>	
	14	Signaled SERR#. <i>default to '0'</i>	
	13	Received Master Abort. <i>default to '0'</i>	
	12	Received Target Abort. <i>default to '0'</i>	
	11	Signaled Target Abort. <i>default to '0'</i>	
10:09	Read/Only. DEVSEL# timing. Slowest that any DEVSEL response may take.		
		<i>Value</i>	<i>Semantics</i>
		1	Medium - during second clock after FRAME# asserted.
	<i>bits</i>	<i>field</i>	
	08	Data Parity Detected. <i>default to '0'</i>	
	07	Fast Back to Back Capable. Read as '0'	
	06	User Defined Features. Read as '0'.	
	05	66MHz Capable. Read as '1'	
	04	Capability List. <i>Extended Capabilities</i>	
		<i>Value</i>	<i>Semantics</i>
		1	AGP/ACPI Capable. Register 34H points to Extended Capabilities Data Structure
	<i>bits</i>	<i>field</i>	
03:00	<i>Reserved.</i> Read as '0'		

Class/Revision

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+F08H	31:00	RO	Class Code/Revision ID
	<i>Bits</i>	<i>Field</i>	
	31:24	Base Class	
		<i>Value</i>	<i>Semantics</i>
		03H	Display Controller
	<i>Bits</i>	<i>Field</i>	
	23:16	Sub-Class	
		<i>Value</i>	<i>Semantics</i>
		00H	VGA Compatible
	<i>Bits</i>	<i>Field</i>	
	15:08	Programming Interface	
		<i>Value</i>	<i>Semantics</i>
		00H	Not applicable
	<i>Bits</i>	<i>Field</i>	
	07:00	Revision ID	
		<i>Value</i>	<i>Semantics</i>
		0003H	Revision Code (a,b,c...)

Test

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+F0CH	31:00	R/W	Test/Control
	<i>Bits</i>	<i>Field</i>	
	31:24	BIST Capability	
		<i>Value</i>	<i>Semantics</i>
		00H	Not capable.
	<i>bits</i>	<i>field</i>	
	23:16	Header Type, default to 00H (Read-only).	
	15:08	Latency Timer, default to 00H.	
	07:00	<i>Reserved.</i> Read as '0'.	

RAM/ROM Base

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+F10H	31:00	R/W	RAM Base Address
	<i>Bits</i>	<i>Field</i>	
	31:27	Base Address (128MB space)	
	26:04	<i>Reserved.</i> Read as '0'.	
	03	Prefetchable. Read as zero.	
		<i>Value</i>	<i>Semantics</i>
		0	Not Prefetchable.
	<i>Bits</i>	<i>Field</i>	
	02:01	Type. Read as zero.	
		<i>Value</i>	<i>Semantics</i>
		0	Locate anywhere in 32 bit address space.
	<i>Bits</i>	<i>Field</i>	
	00	Base Address Type. Read as zero.	
		<i>Value</i>	<i>Semantics</i>
		0	Memory Space Indicator.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+F14H ...+F28H	31:00	R/O	Reserved

Subsystem ID

The Subsystem ID and System Vendor ID are writable in the extended register space, but read-only in the PCI configuration space. At Reset, they are loaded from ROM addresses 0034, 0035, 0036 AND 0037. The data are then written to the registers below (F2C, F2D, F2E and F2F, respectively).

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+F2CH	15:00	R/W	Subsystem ID
	<i>Bits</i>	<i>Field</i>	
	15:00	ID, default to 0000H.	
<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+F2EH	15:00	R/W	Subsystem Vendor ID
	<i>bits</i>	<i>field</i>	
	15:00	ID, default to 0000H.	

ROM Base

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+F30H	31:00	R/W	ROM Base Address
	<i>Bits</i>	<i>Field</i>	
	31:15	ROM Base Address	
	14:01	<i>Reserved. Must be zero</i>	
	00	ROM Enable	
		<i>Value</i>	<i>Semantics</i>
		0	Disabled. <i>Default.</i>
		1	Enabled.
<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+F38H	31:00	R/O	<i>Reserved. Read as zero.</i>

Capability List Pointer

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+F34H	31:00	R/O	CAP PTR, the Capability List Pointer
	<i>Bits</i>	<i>Field</i>	
	31:08	<i>Reserved.</i>	
	07:00	Capability List Pointer, byte offset into PCI configuration space where the capability list starts. Read as 80H.	

Bus Arbitration and Interrupt

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+F3CH	31:00	R/W	Bus Arbitration and Interrupt
	<i>Bits</i>	<i>Field</i>	
	31:24	Maximum Latency. Specified in units of 0.25µS, assuming the system is operating at 33MHz.	
		<i>Value</i>	<i>Semantics</i>
		1	Read/Only. 0.25µS.
	<i>Bits</i>	<i>Field</i>	
	23:16	Minimum Grant. Specified in units of 0.25µS, assuming the system is operating at 33MHz.	
		<i>Value</i>	<i>Semantics</i>
		4	Read/Only. 1.00µS.
	<i>Bits</i>	<i>Field</i>	
	15:08	Interrupt Pin	
		<i>Value</i>	<i>Semantics</i>
		1	Read/Only. Interrupt A.
	<i>Bits</i>	<i>Field</i>	
	07:00	Interrupt Line	
		<i>Value</i>	<i>Semantics</i>
		1	Read/Write. Specifies to which interrupt line/level (0-FH) the interrupt will be routed.

Device Variant

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+F40H	07:00	R/O	Variant Descriptor
	<i>Bits</i>	<i>Field</i>	
	07:00	Revision ID.	
		<i>Value</i>	<i>Semantics</i>
		0	<i>Default.</i>
<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+F41H	07:00	R/W	Transfer Characteristics
	<i>Bits</i>	<i>Field</i>	
	07	3C6h, 3C7h, 3CEh, 3CFh access in enhanced mode	
		<i>Value</i>	<i>Semantics</i>
		0	Disabled. <i>Default.</i>
		1	Enabled.
	06:03	<i>Reserved.</i>	
	02	Burst Transfers: All transactions.	
	01	Burst Transfers: Transactions to internal memory.	
		<i>Value</i>	<i>Semantics</i>
		0	Burst Allowed. <i>Default.</i>
		1	Burst Not Allowed, Single Transfers only.
	<i>Bits</i>	<i>Field</i>	

00	DEVSEL Timing for linear (memory mapped) address space.
	<i>Value Semantics</i>
0	Medium. 1 wait-state. <i>Default.</i>
1	Fast. 0 wait-states.

AGP Extensions

As defined in AGP Spec Revision 2.0, December 10, 1997

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+F80H	31:00	R/O	Capability Identifier Register
	<i>Bits</i>	<i>Field</i>	
	31:24		<i>Reserved.</i>
	23:20		MAJOR. Revision Number. Read as 2.
	19:16		MINOR. Revision Number. Read as 0.
	15:08		NEXT_PTR. Pointer to next item in capabilities list. Read as A0H.
	07:00		CAP_ID. Read as 02H.
<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+F84H	31:00	R/O	AGP Status Register
	<i>Bits</i>	<i>Field</i>	
	31:24		RQ_DEPTH. Maximum number of AGP command requests this device can generate.
	23:10		<i>Reserved.</i> Read as 0.
	09		SBA. Side Band Addressing. Read as 1.
	08:06		<i>Reserved.</i> Read as '000'.
	05		4 GB Support. Read As 0.
	04		Fast Write Support. Read As 0.
	03		<i>Reserved.</i> Read as 0.
	02:00		RATE. Read as 011B.
<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+F88H	31:00	R/W	AGP Command Register. <i>Default to 0.</i>
	<i>Bits</i>	<i>Field</i>	
	31:24		RQ_DEPTH. Maximum number of AGP command requests this device is allowed to generate.
	23:10		<i>Reserved.</i> Read as 0.
	09		SBA_ENABLE. Side Band Addressing. Default as 0
	08		AGP_ENABLE. Side Band Addressing. Default as 0.
	07:06		<i>Reserved.</i> Read as '00'.
	05		4 GB Support. Default As 0.
	04		Fast Write Support. Default As 0.
	03		<i>Reserved.</i> Read as 0.
	02:00		DATA_RATE. Default as '000'.

Power Management Register Block

As defined in PCI Bus Power Management Interface Specification Revision 1.0, March 18, 1996.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+FA0H	31:00	R/O	Power Management Capability Register
	<i>Bits</i>	<i>Field</i>	
	31:27	PME_Support.	Read as '01111'
	26	D2_Support.	Read as 1.
	25	D1_Support.	Read as 1.
	24:22	AUXILARY POWER SOURCE.	Read as 000.
	21	Device Specific Initialization.	Read as 0.
	20	RESERVED.	Read as 0.
	19	PME_CLOCK.	Read as 1.
	18:16	VERSION	Revision Number. Read as 001.
	15:08	NEXT_PTR.	Pointer to next item in capabilities list. Read as 00H. Final Item.
	07:00	CAP_ID.	Read as 01H.
	<i>Offset</i>	<i>Field</i>	<i>Access</i> <i>Function</i>
+FA4H	31:00	R/W	Power Management Control/Status Register
	<i>Bits</i>	<i>Field</i>	
	31:24	Power consumption/dissipation reporting data register.	Read as '00000000'
	23	BPCC_Enable.	Read as 0.
	22	B2_B3# (B2/B3 support for D3 hot).	Read as 0.
	21:16	Reserved.	Read as "000000".
	15	PME Status for D3 cold.	Default as 0.
	14:13	Data_Scale.	Read as '00'.
	12:09	Data_Select.	Default '0000'.
	08	PME_Enable from D3 cold.	
		<i>Value</i>	<i>Semantics</i>
		0	Disable. <i>Default</i>
		1	Enable
	<i>Bits</i>	<i>Field</i>	
	07:02	Reserved.	Read as '000000'
	01:00	Power State	
		<i>Value</i>	<i>Semantics</i>
		0	D0. <i>Default</i>
		1	D1
		2	D2
		3	D3

4.2.2 3D Control (1FF3000H)

The 3D control registers occupy a 4KB region within the control address space, as shown in Table 4-1. They are organized into 10 sub-regions of 64B each, according to their major functions. Beyond these control registers are 6 sub-regions defined for local storage of data used in 3D operations. All of these sub-regions are shown in the Table 4-5, below. All are accessed with byte resolution.

Register addresses below are shown as a 12-bit offset designated “+xxxH” within the 4KB 3D control register address space.

Register Type	Offset	Size	Access	Comments
Command Decode Unit (CDU)	000H	64B	byte	
Triangle Setup Unit (TSU)	040H	64B	byte	
Edge-Walking Unit (EWU)	080H	64B	byte	
Texture Memory Unit (TMU)	0C0H	64B	byte	
Z-Buffer Unit (ZBU)	100H	64B	byte	
Per-Fragment Unit (PFU)	140H	64B	byte	
Memory Interface Unit (MIU)	180H	64B	byte	
Direct Memory Access (DMA)	1C0H	64B	byte	
<i>Reserved</i>	200H	256B	byte	
Motion Compensation (MC)	300H	64B	byte	
<i>Reserved</i>	340H	192B	byte	
Stipple Mask Array	400H	128B	byte	
Texture Map LOD Array	580H	44B	byte	
Temp. Array (Triangle Setup)	5C0H	64B	byte	
Vertex List Array	600H	512B	byte	
Edge List Array	800H	1024B	byte	
Texture Palette Array	C00H	1024B	byte	

Table 4-5. 3D Control Register Allocation

Another way to visualize the 3D register addresses is shown in Figure 4-21 below

	CDU	TSU	EWU	TMU	ZBU	PFU	MIU	DMA	MC
+3CH	Diag		Diag	Diag	Diag	Diag	Diag	Diag	MC Busy
+38H		Ctr	Control	Control	Control		Int Control		Slot Stride
+34H		Mask		Wrap	Transp	PixFmt			Slot Scanline
+30H		Status			Stride	Stride			Fast Status
+2CH									Tex End
+28H									Tex Start
+24H	S/W Flags			Base	Index		Logical Ops	Mem/Bank	Tex Load
+20H	S/W Flags			Size	Format	Stipple	Dither	Relocation	Tex Load
+1CH							αBlend		Fill Data
+18H							Fog Color	Cmd FIFO	Dst WrPtr
+14H									Src RdPtr
+10H	Perf Cntr						Tex Blend		Transfer
+0CH									
+08H								Z-Buffer	FIFO End
+04H		Synd		High Color	X Limits Right				FIFO RdPtr
+00H	Vertex		Y Limits	Low Color	X Limits Left		Bk Buffer	FIFO WrPtr	
	000H	040H	080H	0C0H	100H	140H	180H	1C0H	300H

Figure 4-21. 3D Control Register Map

4.2.2.1 Command Decode Unit (+000H)

Vertex Format

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+000H	15:00	R/W	Vertex Information (amount of information)
	<i>Bits</i>	<i>Field</i>	
	15:08		<i>Reserved.</i>
	07:06		Vertex Texture Type.
		<i>Value</i>	<i>Semantics</i>
		0	None. <i>Default.</i>
		1	UV.
		2	UVW.
		3	UVWMIP.
	<i>Bits</i>	<i>Field</i>	
	05		Vertex Coordinate Format.
		<i>Value</i>	<i>Semantics</i>
		0	Floating Point, IEEE Single Precision 1.8.23.
		1	Fixed Point, 0.11.21.
	<i>Bits</i>	<i>Field</i>	
	04		Vertex Fog.
		<i>Value</i>	<i>Semantics</i>
		0	Disable. <i>Default.</i>
		1	Enable.
	<i>Bits</i>	<i>Field</i>	
	03		Vertex Specular.
		<i>Value</i>	<i>Semantics</i>
		0	Disable. <i>Default.</i>
		1	Enable.
	<i>Bits</i>	<i>Field</i>	
	02		Vertex Coordinate.
		<i>Value</i>	<i>Semantics</i>
		0	XY.
		1	XYZ.
	<i>Bits</i>	<i>Field</i>	
	01:00		Vertex Color.
		<i>Value</i>	<i>Semantics</i>
		0	No Color.
		1	RGB
		2	ARGB.
		3	<i>Reserved.</i>

Intertriangle synchronization

The SM3110 3D pipeline does not detect read-after-write (RAW) hazards, which can cause stale data to be read unless the read is stalled or a bypass path is available. The SM3110 implements a stall, but using triangle bounding boxes, in the Triangle Setup Unit. This relies on the very long processing time for a triangle (80+ cycles), to prevent more than two triangles from being processed simultaneously by the pipeline, which allows the bounding box test to be applied only to the previous triangle.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+004H	07:00	R/W	Intertriangle synchronization
	<i>Bits</i>	<i>Field</i>	
	07:02	<i>Reserved. Read as zero.</i>	
	01	Per Fragment Unit.	
	00	Z-Buffering Unit.	
		<i>Value</i>	<i>Semantics</i>
		0	Do not stall. <i>Default.</i>
		1	Stall until unit idle.

Performance Counter

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+010H	31:00	R/W	Performance counter. increments whenever an input clock pulse is received.
	<i>Bits</i>	<i>Field</i>	
	31:00	Counter. used for performance analysis, visibility control. Range is 51secs at 83MHz.	

Driver Flags

Offset	Field	Access	Function
+024H	15:00	R/W	Driver Flags
+026H	15:00	R/W	Driver Flags
	Bits	Field	
	15:01		Reserved.
	00		Flag.

Status

Multiple status flags and/or values are accessible through this single register.

Offset	Field	Access	Function
+030H	15:00	R/O	Status
	Bits	Field	
	15		Driver Flag @ Register 026H bit [00].
	14		Driver Flag @ Register 024H bit [00].
	13		2D Signal @ Register 022H bit [00].
	12		2D Synchronization @ Register 020H bit [00].
	11		Format Conversion.
	10		Motion Compensation.
	09		3D Engine Busy.
		Value	Semantics
		0	Idle, completed all commands and no pending memory accesses. <i>Default.</i>
		1	Busy, either executing a command or pending completion of a memory access).
			This is an OR of all of the unit status bits.
	Bits	Field	
	08		Running.
		Value	Semantics
		0	Stopped (State), in response to the run bit being deasserted or after reset. <i>Default.</i>
		1	Running (State), either idle awaiting commands or busy processing a command.
	Bits	Field	
	07		Direct Memory Access.
	06		3D Command FIFO.
	05		Per Fragment Unit.
	04		Z-Buffering Unit.
	03		Texture Mapping Unit.
	02		Edge Walking Unit.
	01		Triangle Setup Unit.
	00		Command Decode Unit.
		Value	Semantics
		0	Idle, completed all commands and no pending memory accesses.
		1	Busy, either executing a command or pending completion of a memory access).

Control

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+038H	07:00	R/W	3D Rendering Engine
	<i>Bits</i>	<i>Field</i>	
	07		Reset.
		<i>Value</i>	<i>Semantics</i>
		0	Normal Operation.
		1	Reset, terminates current operation in progress and resets the engine. <i>Default.</i> When deasserted it will left in a Stopped state.
	<i>Bits</i>	<i>Field</i>	
	06:03		<i>Reserved.</i> Read as zero.
	02		Pixel zero fill
		<i>Value</i>	<i>Semantics</i>
		0	Disabled. <i>Default.</i>
		1	Enabled. (When X or Y are less than 0.5, force to zero)
	<i>Bits</i>	<i>Field</i>	
	01		Execute/Single Step. The command specified in the flags register is executed. The bit is reset upon completion of one command. This is ignored if the Run bit is set.
		<i>Value</i>	<i>Semantics</i>
		0	Normal Execution. <i>Default.</i>
		1	Execute Single Command.
	<i>Bits</i>	<i>Field</i>	
	00		Start. This bit is sampled only between command execution boundaries. If it is deasserted, then reasserted while a command is executing it will not cause any change.
		<i>Value</i>	<i>Semantics</i>
		0	Stop (Request). Stop after the end of the current command. <i>Default.</i> This can also be reset by a command issued through the engine FIFO.
		1	Start (Request). Start execution (if any commands are pending in the FIFO).

3D Engine Diagnostic Port Control

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+03EH	15:00	R/W	Diagnostic Port Control.
	<i>Bits</i>	<i>Field</i>	
	15:12	<i>Reserved.</i> Read as zero.	
	11	Performance Counter Enable.	
		<i>Value</i>	<i>Semantics</i>
		0	Disabled. <i>Default.</i>
		1	Enabled. Resets to Zero when enabled.
	<i>Bits</i>	<i>Field</i>	
	10:08	Select Performance Counter Source.	
		<i>Value</i>	<i>Semantics</i>
		0-7	Selects bit of Diagnostic Port to be used as count enable for Performance Counter.
	<i>Bits</i>	<i>Field</i>	
	07:04	Selects diagnostic information to be accessed from Triangle Setup Unit.	
		<i>Value</i>	<i>Semantics</i>
		0	Bypass output of previous unit. <i>Default.</i>
		1-FH	Select diagnostic signals.
	<i>Bits</i>	<i>Field</i>	
	03:00	Selects diagnostic information to be accessed from CDU.	
		<i>Value</i>	<i>Semantics</i>
		0	Bypass output of previous unit. <i>Default.</i>
		1-FH	Select diagnostic signals.

4.2.2.2 Triangle Setup Unit (+040H)

4.2.2.3 Edge-Walking Unit (+080H)

Vertical Viewport Limits

Offset	Field	Access	Function
+080H	15:00	R/W	Ymin - Viewport Top Edge
+082H	15:00	R/W	Ymax - Viewport Bottom Edge
	Bits	Field	
	15:11		Reserved.
	10:00		Coordinate. specifies the actual pixel at the edge of the viewport (inclusive).

Control

Offset	Field	Access	Function
+0B8H	15:00	R/W	Control
	Bits	Field	
	15:08		Reserved.
	07		Scissors. (shadowed in Z-Buffering Unit).
	06		Texture. (shadowed in Triangle Setup Unit, Z-Buffering Unit, Per Fragment Unit).
	05		Perspective Correction Enable (shadowed in Triangle Setup Unit).
	04		Mipmapping. (shadowed in Triangle Setup Unit).
	03		Subpixel Displacement (shadowed in Triangle Setup Unit).
	02		Specular (shadowed in Triangle Setup Unit, Per Fragment Unit).
	01		Fog (shadowed in Triangle Setup Unit, Per Fragment Unit).
		Value	Semantics
		0	Disabled. <i>Default.</i>
		1	Enabled.
	Bits	Field	
	00		Flat Shading (shadowed in Triangle Setup Unit).
		Value	Semantics
		0	Smooth Shading. <i>Default.</i>
		1	Flat Shading.

Diagnostic Port Control

Offset	Field	Access	Function
+0BEH	15:00	R/W	Diagnostic Port Control
	Bits	Field	
	15:08		Reserved.
	07:04		Read as zero.
	03:00		Selects diagnostic information to be accessed.
		Value	Semantics
		0	Bypass output of previous unit. <i>Default.</i>
		1-FH	Select diagnostic signals.

4.2.2.4 Texture Mapping Unit (+0C0H)

Texture Transparent Color

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0C0H	31:00	R/W	Lower Bound
+0C4H	31:00	R/W	Upper Bound
	<i>Bits</i>	<i>Field</i>	
	31:24	<i>Reserved.</i> Read as zero.	
	23:16	Red.	
	15:08	Green.	
	07:00	Blue/Index. This byte contains the CLUT index if the comparison is done against the palettized texture.	

Texture Format

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0E0H	15:00	R/W	Texel Format , shadowed in Per Fragment Unit
	<i>Bits</i>	<i>Field</i>	
	15:04	<i>Reserved.</i> Read as zero.	
	03:00	Texel Format.	
		<i>Value</i>	<i>Semantics</i>
		0	CLUT 1 (2 color palette)
		1	CLUT 2 (4 color palette)
		2	CLUT 4 (16 color palette)
		3	CLUT 8 (256 color palette)
		4	<i>Reserved</i>
		5	RGB x.5.5.5
		6	RGB 5.6.5
		7	<i>Reserved.</i>
		8-AH	<i>Reserved.</i>
		BH	<i>Reserved.</i>
		CH	ARGB 4.4.4.4
		DH	ARGB 1.5.5.5
		EH	<i>Reserved.</i>
		FH	<i>Reserved.</i>

Texture Map Size

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0E2H	15:00	R/W	Texture Map Size
	<i>Bits</i>	<i>Field</i>	
	15:12	<i>Reserved.</i>	
	11:08	U - Texture Map Size	
		<i>Value</i>	<i>Semantics</i>
	0-10	Ceiling of $\text{Log}_2(\text{Texture_Map_Size})$	
	11-15	<i>Reserved.</i>	
	<i>Bits</i>	<i>Field</i>	
	07:04	<i>Reserved.</i>	
	03:00	V - Texture Map Size	
		<i>Value</i>	<i>Semantics</i>
	0-10	Ceiling of $\text{Log}_2(\text{Texture_Map_Size})$	
	11-15	<i>Reserved.</i>	

Texture Palette Index

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0E4H	15:00	R/W	Palette Index Offset
	<i>Bits</i>	<i>Field</i>	
	15:08	<i>Reserved.</i>	
	07:00	Offset into Palette Array.	

Texture Base Address

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0E6H	15:00	R/W	Texel Base Page Address
	<i>Bits</i>	<i>Field</i>	
	15:14	<i>Reserved.</i>	
	13:03	Page (2KB) Address of the first texel of the current texture map or mip-map.	
	02:00	<i>Reserved.</i>	

Texture Wrap

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0F4H	15:00	R/W	Texture Wrap
	<i>Bits</i>	<i>Field</i>	
	15:10	<i>Reserved.</i> Read as zero.	
	07:02	<i>Reserved.</i> Read as zero.	
	09:08	U Wrap Mode.	
	01:00	V Wrap Mode.	
		<i>Value</i>	<i>Semantics</i>
		0	Clamp.
		1	Repeat.
		2	Mirror.
		3	<i>Reserved.</i>

Texture Control

To properly sample a texture, you need to align the texture samples with the midpoint between the coordinate grid to pixels which are sampled at the midpoint (center) of the coordinate grid.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0F8H	07:00	R/W	Texture Interpolation Control
	<i>Bits</i>	<i>Field</i>	
	07	Alpha Test	
		<i>Value</i>	<i>Semantics</i>
		0	No alpha test. <i>Default.</i>
		1	Perform alpha test (if texel alpha is zero, make pixel transparent).
	<i>Bits</i>	<i>Field</i>	
	06	Texel Sampling.	
		<i>Value</i>	<i>Semantics</i>
		0	Textures are sampled at upper left corner. <i>Default.</i>
		1	Textures are sampled at center.
	<i>Bits</i>	<i>Field</i>	
	05:04	Transparent Mode.	
		<i>Value</i>	<i>Semantics</i>
		0	Nearest Index.
		1	Nearest Color.
		2	Filtered Color.
		3	<i>Reserved.</i>
	<i>Bits</i>	<i>Field</i>	
	03:02	<i>Reserved.</i>	
	01	Interpolation between mipmapped levels.	
		<i>Value</i>	<i>Semantics</i>
		0	Nearest Neighbor.
		1	Linear.
	<i>Bits</i>	<i>Field</i>	
	00	Interpolation in UV space.	
		<i>Value</i>	<i>Semantics</i>
		0	Nearest Neighbor.
		1	Bilinear.

Diagnostic Port Control

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+0FEH	15:00	R/W	Diagnostic Port Control
	<i>Bits</i>	<i>Field</i>	
	15:04	<i>Reserved.</i>	Read as zero.
	03:00	Selects diagnostic information to be accessed.	
		<i>Value</i>	<i>Semantics</i>
		0	Bypass output of previous unit. <i>Default.</i>
		1-FH	Select diagnostic signals.

4.2.2.5 Z Buffer Unit (+100H)

Horizontal Viewport Limits

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+100H	15:00	R/W	Xmin - Viewport Left Edge
+102H	15:00	R/W	Xmax - Viewport Right Edge
	<i>Bits</i>	<i>Field</i>	
	15:11	<i>Reserved.</i>	
	10:00	Coordinate which specifies the actual pixel at the edge of the viewport (inclusive).	

Stipple

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+120H	15:00	R/W	Stipple
	<i>Bits</i>	<i>Field</i>	
	15	Enable	
		<i>Value</i>	<i>Semantics</i>
		0	Disabled. <i>Default.</i>
		1	Enabled.
	<i>Bits</i>	<i>Field</i>	
	14	<i>Reserved.</i> Read as zero.	
	13	<i>Reserved.</i>	
	12:08	Y Stipple Pattern Offset	
	07:06	<i>Reserved.</i> Read as zero.	
	05	<i>Reserved.</i>	
	04:00	X Stipple Pattern Offset	

Z Buffer Stride

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+130H	15:00	R/W	Z Buffer Stride
	<i>Bits</i>	<i>Field</i>	
	15	Z Tiling Enable.	
		<i>Value</i>	<i>Semantics</i>
		0	Disabled. <i>Default.</i>
		1	Enabled.
	<i>Bits</i>	<i>Field</i>	
	14:11	<i>Reserved.</i> Read as zero.	
	10:04	Screen stride in multiples of 16 pixels.	
	03:00	<i>Reserved.</i> Read as zero.	

Texture Transparency

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+134H	15:00	R/W	Texture Transparency Enable
	<i>Bits</i>	<i>Field</i>	
	15:01	<i>Reserved.</i>	
	00	Enable.	
		<i>Value</i>	<i>Semantics</i>
		0	Disabled. <i>Default.</i>
		1	Enabled.

Control

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+138H	15:00	R/W	Z Buffer Control
	<i>Bits</i>	<i>Field</i>	
	15:05	<i>Reserved.</i> Read as zero.	
	04	Z Buffer Write Enable.	
		<i>Value</i>	<i>Semantics</i>
		0	Disabled. <i>Default.</i>
		1	Enabled.
	<i>Bits</i>	<i>Field</i>	
	03	Enable Z Test.	
		<i>Value</i>	<i>Semantics</i>
		0	Disabled. <i>Default.</i>
		1	Enabled.
	<i>Bits</i>	<i>Field</i>	
	02:00	Z Test Function.	
		<i>Value</i>	<i>Semantics</i>
		0	Never Pass
		1	Z ≤ destination Pixel Z
		2	Z < destination Pixel Z
		3	Z > destination Pixel Z
		4	Z ≥ destination Pixel Z
		5	Z == destination Pixel Z
		6	Z ≠ destination Pixel Z
		7	Always Pass

Diagnostic Port Control

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+13EH	15:00	R/W	Diagnostic Port Control
	<i>Bits</i>	<i>Field</i>	
	15:04	<i>Reserved.</i>	
	03:00	Selects diagnostic information to be accessed.	
		<i>Value</i>	<i>Semantics</i>
		0	Bypass output of previous unit. <i>Default.</i>
		1-FH	Select diagnostic signals.

4.2.2.6 Per Fragment Unit (+140H)

Texture Blend

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+150H	15:00	R/W	Texture Blend
	<i>Bits</i>	<i>Field</i>	
	15:09	<i>Reserved.</i> Read as zero.	
	08	Texture Blend. Shadowed in Z-Buffering Unit.	
		<i>Value</i>	<i>Semantics</i>
		0	Disabled. <i>Default.</i>
		1	Enable.
	<i>Bits</i>	<i>Field</i>	
	07:06	<i>Reserved.</i>	
	05:04	Texture Blend Alpha Mode.	
		<i>Value</i>	<i>Semantics</i>
		0	Copy.
		1	Modulate.
		2	Off.
		3	One.
	<i>Bits</i>	<i>Field</i>	
	03	<i>Reserved.</i>	
	02:00	Texture Blend Color Mode.	
		<i>Value</i>	<i>Semantics</i>
		0	Decal.
		1	Modulate.
		2	Decal, Alpha.
		3	<i>Reserved.</i>
		4	Decal, Mask.
		5	Modulate, Mask.
		6	Add, Clamp.
		7	<i>Reserved.</i>

Fog Color

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+158H	31:00	R/W	Fog Color
	<i>Bits</i>	<i>Field</i>	
	31:24	<i>Reserved.</i> Read as zero.	
	23:16	Red.	
	15:08	Green.	
	07:00	Blue.	

Alpha Blend

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+15CH	15:00	R/W	Alpha Blend Control
	<i>Bits</i>	<i>Field</i>	
	15:10	<i>Reserved.</i>	
	09	Alpha Blend.	
	08	Blend Alpha Component. Shadowed in Edge Walking Unit.	
		<i>Value</i>	<i>Semantics</i>
		0	Disabled. <i>Default.</i>
		1	Enabled.
	<i>Bits</i>	<i>Field</i>	
	07	<i>Reserved.</i> Read as zero.	
	06:04	Destination Blend Control.	
		<i>Value</i>	<i>Semantics</i>
		0	Zero
		1	One
		2	Source, Color
		3	Invert Source, Color
		4	Source, Alpha
		5	Invert Source, Alpha
		6	Destination, Alpha
		7	Invert Destination, Alpha
	<i>Bits</i>	<i>Field</i>	
	03	<i>Reserved.</i> Read as zero.	
	02:00	Source Blend Control.	
		<i>Value</i>	<i>Semantics</i>
		0	Zero
		1	One
		2	Destination, Color
		3	Invert Destination, Color
		4	Source, Alpha
		5	Invert Source, Alpha
		6	Destination, Alpha
		7	Invert Destination, Alpha

Dither

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+160H	15:00	R/W	Control
	<i>Bits</i>	<i>Field</i>	
	15:09	<i>Reserved.</i>	
	08	<i>Dither.</i>	
		<i>Value</i>	<i>Semantics</i>
		0	<i>Disabled. Default.</i>
		1	<i>Enabled.</i>
	<i>Bits</i>	<i>Field</i>	
	07:00	<i>Reserved. Read as zero.</i>	

Logical Op

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+164H	15:00	R/W	Logical Operation
	<i>Bits</i>	<i>Field</i>	
	15:09	<i>Reserved.</i>	
	08	<i>Logical Operation.</i>	
		<i>Value</i>	<i>Semantics</i>
		0	<i>Disabled. Default.</i>
		1	<i>Enabled.</i>
	<i>Bits</i>	<i>Field</i>	
	07:04	<i>Reserved. Read as zero.</i>	
	03:00	<i>Logical Operation.</i>	
		<i>Value</i>	<i>Logical Op</i>
			<i>Semantics</i>
		0H	<i>Zero. Zero. Black.</i>
		1H	<i>!(Src OR Dest) NotMergePen</i>
		2H	<i>Dest AND !Src MaskNotPen</i>
		3H	<i>!Src NotCopyPen</i>
		4H	<i>Src AND !Dest MaskPenNot</i>
		5H	<i>!Dest Not</i>
		6H	<i>Dest XOR Src XORPen</i>
		7H	<i>!(Dest AND Src) NotMaskPen</i>
		8H	<i>Dest AND Src MaskPen</i>
		9H	<i>!(Dest XOR Src) NotXORPen</i>
		AH	<i>Dest NOP</i>
		BH	<i>Dest OR !Src MergeNotPen</i>
		CH	<i>Src CopyPen</i>
		DH	<i>Src OR !Dest MergePenNot.</i>
		EH	<i>Src OR Dest MergePen.</i>
		FH	<i>Ones White.</i>

Rendering (Back) Buffer Stride

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+170H	15:00	R/W	Rendering (Back) Stride
	<i>Bits</i>	<i>Field</i>	
	15	Back Buffer Write Enable.	
		<i>Value</i>	<i>Semantics</i>
		0	Write Disabled. <i>Default.</i>
		1	Write Enabled.
	<i>Bits</i>	<i>Field</i>	
	14:11	<i>Reserved.</i> Read as zero.	
	10:04	Screen stride in multiples of 16 pixels.	
	03:00	<i>Reserved.</i> Read as zero.	

Pixel Format

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+174H	15:00	R/W	Pixel Format shadowed in Edge Walking Unit
	<i>Bits</i>	<i>Field</i>	
	15:05	<i>Reserved.</i> Read as zero.	
	04:03	Pixel Size.	
		<i>Value</i>	<i>Semantics</i>
		0	8bpp.
		1	16bpp.
		2	24bpp.
		3	32bpp.
	<i>Bits</i>	<i>Field</i>	
	02	Pixel Format. Alpha enabled.	
	01:00	Pixel Format. Depends on Pixel Size.	
		<i>value</i>	<i>8bpp</i> <i>16bpp</i> <i>24bpp</i> <i>32bpp</i> <i>semantics</i>
		0	<i>Reserved</i> 4.4.4.4 <i>Reserved</i> <i>Reserved</i> 4b/color.
		1	<i>Reserved</i> 1.5.5.5 <i>Reserved</i> <i>Reserved</i> 5b/color.
		2	<i>Reserved</i> 5.6.5 <i>Reserved</i> <i>Reserved</i> 6b/color.
		3	<i>Reserved</i> <i>Reserved</i> 8.8.8 8.8.8.8 8b/color.
	<i>Bits</i>	<i>Field</i>	
	04:00	Pixel Format. Encoded format, illegal or unsupported formats indicated.	
		<i>Value</i>	<i>Semantics</i>
		0-8	<i>Reserved.</i>
		09H	16bpp, RGB x.5.5.5
		0AH	16bpp, RGB 5.6.5
		0CH	16bpp, ARGB 4.4.4.4
		0DH	16bpp, ARGB 1.5.5.5
		0E-12H	<i>Reserved.</i>
		13H	24bpp, RGB 8.8.8 (packed 24b).
		14-1EH	<i>Reserved.</i>
		1FH	32bpp, ARGB 8.8.8.8.

Diagnostic Port Control

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+17EH	15:00	R/W	Diagnostic Port Control
	<i>Bits</i>	<i>Field</i>	
	15:04	<i>Reserved.</i> Read as zero.	
	03:00	Selects diagnostic information to be accessed.	
		<i>Value</i>	<i>Semantics</i>
		0	Bypass output of previous unit. <i>Default.</i>
		1-FH	Select diagnostic signals.

4.2.2.7 Memory Interface Unit (+180H)

Back Buffer

Specifies the starting address of the buffer used for the back (rendering) buffer.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+180H	31:00	R/W	Back (Rendering) Buffer Base
	<i>Bits</i>	<i>Field</i>	
	31:24	<i>Reserved.</i>	
	23:12	Base Address, in units of 4KB (2×page size).	
	11:00	Must be zero, each buffer must be aligned to a natural 4KB boundary.	

Z Buffer

Specifies the starting address of the buffer used for the Z-buffer.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+188H	31:00	R/W	Z Buffer Base
	<i>Bits</i>	<i>Field</i>	
	31:24	<i>Reserved.</i>	
	23:12	Base Address, in units of 4KB (2×page size).	
	11:00	Must be zero, each buffer must be aligned to a natural 4KB boundary.	

Command/Parameter FIFO

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+198H	31:00	R/W	Command/Parameter FIFO Base
	<i>Bits</i>	<i>Field</i>	
	31:24	<i>Reserved.</i>	
	23:14	Base Address, in units of 16KB.	
	13:00	Must be zero, each buffer must be aligned to a natural 16KB boundary.	

Memory/Bank Relocation

There are 8 one-byte registers that allow remapping of the first 16MB of the local address space on 2.0MB increments. Since there are 8 registers, a maximum of 16MB of memory may be assigned. This can be extended in the future. The SM3110 has 4MB of internal DRAM.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+1A0H	07:00	R/W	Memory Relocation. 0xxxxxH-ExxxxxH, in 2MB increments
	<i>Bits</i>	<i>Field</i>	
	07:06	<i>Reserved.</i>	
	05:04	Chip/Array Select	
		<i>Value</i>	<i>Semantics</i>
		0	CS[0]. for Internal DRAM.
		1-3	<i>Reserved for External SDRAM.</i>
	<i>Bits</i>	<i>Field</i>	
	03:02	<i>Reserved.</i>	
	01	Address bit [21] or block/bank selects.	
	00	<i>Reserved.</i>	

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+1A1H	07:00	R/W	Memory Relocation. 0xxxxxH-ExxxxxH, in 2MB increments
	<i>Bits</i>	<i>Field</i>	
	07:06	<i>Reserved.</i>	
	05:04	Chip/Array Select	
		<i>Value</i>	<i>Semantics</i>
		0	CS[0]. for Internal DRAM.
		1-3	<i>Reserved for External SDRAM.</i>
	<i>Bits</i>	<i>Field</i>	
	03:02	<i>Reserved.</i>	
	01	Address bit [21] or block/bank selects.	
	00	<i>Reserved.</i>	

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+1A2H	07:00	R/W	Reserved
...+1A7H			

Memory Access Control

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+1B8H	07:00	R/W	Reserved
+1B9H	07:00	R/W	Internal DRAM Control (reserved)
+1BAH	15:00	R/W	Internal Memory Array Timing (reserved)
+1BCH	15:00	R/W	Internal Memory Array Timing (reserved)

Diagnostic Port Control

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+1BEH	15:00	R/W	Diagnostic Port Control
	<i>Bits</i>	<i>Field</i>	
	15:05	<i>Reserved.</i> Read as zero.	
	04:00	Selects diagnostic information to be accessed.	
		<i>Value</i>	<i>Semantics</i>
		0	Bypass output of previous unit. <i>Default.</i>
		1-1FH	Select diagnostic signals.

4.2.2.8 DMA (+1C0H)

3D Command/Parameter FIFO: Write Ptr, Read Ptr, End/Size

The 3D Command FIFO is located in internal memory. Typically it will be placed at the top of internal memory. The read and write pointers and end/size values are relative to the command FIFO base address specified by the +CxxH registers in the 2D Control section (4.2.1).

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+1C0H	31:00	R/W	Write Pointer
	<i>Bits</i>	<i>Field</i>	
	31:20	<i>Reserved.</i> Write zero to this bit.	
	19:02	Pointer. in double word (32b) units.	
	01:00	<i>Reserved.</i> Write zero to this bit.	
<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+1C4H	31:00	R/W	Read Pointer
	<i>Bits</i>	<i>Field</i>	
	31:20	<i>Reserved.</i> Write zero to this bit.	
	19:02	Pointer. in double word (32b) units.	
	01:00	<i>Reserved.</i> Write zero to this bit.	
<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+1C8H	31:00	R/W	End/Size
	<i>Bits</i>	<i>Field</i>	
	31:20	<i>Reserved.</i> Write zero to this bit.	
	19:14	End relative to base of 0, offset by FIFO base in memory interface unit. Size in number of 16KB increments.	
	13:00	<i>Reserved.</i> Write zero to this bit.	

3D DMA Transfer Data

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+1D0H	31:00	R/W	Transfer Length and Flags
	<i>Bits</i>	<i>Field</i>	
	31:28	<i>Reserved.</i>	
	27:24	Source Flags.	
	23:20	Destination Flags.	
		<i>Value</i>	<i>Semantics</i>
		0H	Non-aligned Interval.
		1H	Aligned to natural 16B boundary.
		2H	State Registers except for Palette. <i>Destination only.</i>
		3H	Palette. Used to load palette RAM, 18b per clock. <i>Destination only.</i>
		4H	AGP/PCI Linear.
		5H	<i>Reserved.</i>
		6H	32b Fill Register. <i>Source only.</i> Used to fill memory
		7H	Internal. Translate (texture) addresses. <i>Destination only.</i>
		8-FH	<i>Reserved.</i>
	<i>Bits</i>	<i>Field</i>	
	19	Transfer Request. This bit is automatically reset by hardware upon completion of a transfer	
		<i>Value</i>	<i>Semantics</i>
		0	No Request Outstanding.
		1	Request Outstanding.
	<i>Bits</i>	<i>Field</i>	
	18:00	Length minus one in doublewords.	
<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+1D4H	31:00	R/W	Source (Read) Pointer
+1D8H	31:00	R/W	Destination (Write) Pointer
	<i>Bits</i>	<i>Field</i>	
	31:00	Pointer. Bits [01:00] must be zero, the pointers are always in double word (32b) units.	
<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+1DCH	31:00	R/W	Source Data
	<i>Bits</i>	<i>Field</i>	
	31:00	Fill Data, primarily used for initializing the Z buffer. It can also be used to fill any accessible memory array. 16b Z-value should be replicated in both upper and lower 16b words.	

When the palette is the destination, bits [09:02] of the destination address specify the palette index. Bits [01:00] specify the bytes within the palette and should be set to 0; byte 0 is Blue, byte 1 is Green and byte 2 is Red (byte 3 is not used). The source (internal or AGP/PCI) and destination palette addresses must be doubleword aligned. The matrix of source and destination address spaces is shown in the table below. The palette RAM cannot be the destination addressed through the State Register space, it must be accessed only through its own dedicated space. Remember, the transfer rate is 32b/cycle when non-aligned but 128b/cycle when aligned.

Destination	Source			
	0-Internal, Non-aligned	1-Internal Aligned	4-External AGP/PCI	6-Fill Register
0 - Internal Non-aligned	Yes	No	Yes	Yes
1 - Internal Aligned	No	Yes	No	Yes
2 - State Registers	Yes	No	No	No
3 - Palette RAM	Yes	No	No	No
4 - AGP/PCI	No	No	No	No
7 - Internal Texture	Yes	No	Yes	No

Table 4-6. DMA Transfer Source/Destination Matrix

Texture Load

These fields specify the parameters needed to perform the address translation required to load textures into texture memory. The texture map size register specifies the ($\log_2(\text{size})$) dimensions of the largest texture and is loaded once per texture map. The translation parameters are loaded for each mipmap level of detail.

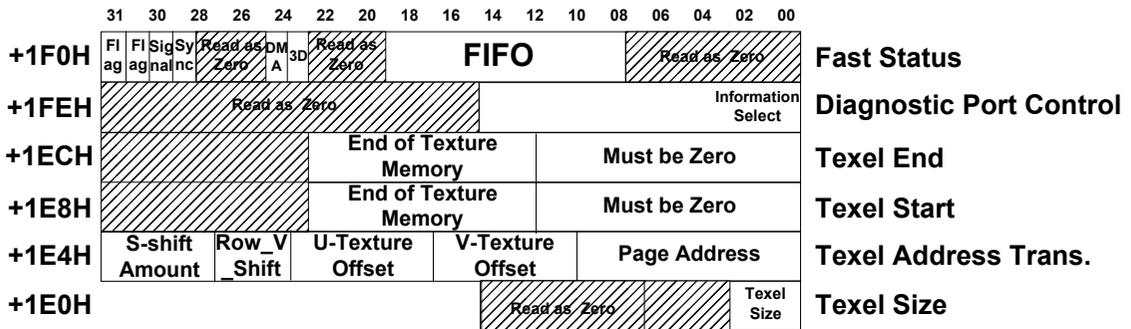


Figure 4-22. Texture Map Size Register

Texture Format

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+1E0H	15:00	R/W	Texel Size used during Texture loads.
	<i>Bits</i>	<i>Field</i>	
	15:08	<i>Reserved.</i>	Read as zero.
	07:03	<i>Reserved.</i>	
	02:00		Texel Size.
		<i>Value</i>	<i>Semantics</i>
		0	1b - CLUT 1 (2 color palette)
		1	2b - CLUT 2 (4 color palette)
		2	4b - CLUT 4 (16 color palette)
		3	8b - CLUT 8 (256 color palette)
		4	16b – RGB x.5.5.5 and 5.6.5.
		5-7	<i>Reserved.</i>

Texture Address Translation Parameters

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+1E4H	31:00	R/W	Texture Address Translation Parameters
	<i>Bits</i>	<i>Field</i>	
	31:28	S - Shift Amount.	$S = \max(0, \log_2 u - \text{lodint} - \text{bsxe})$
	27:25	Row_V_Shift.	
	24:18	U - Texture Offset from Page Address (bits [06:00]).	
	17:11	V - Texture Offset from Page Address (bits [06:00]).	
	10:00	Page Address relative to Texel Base Page Register.	

Texture Base/End

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+1E8H	31:00	R/W	Texture Base Indicates where to start performing texture address translation during loads.
+1ECH	31:00	R/W	Texture End Indicates where to stop performing texture address translation during loads.
	<i>Bits</i>	<i>Field</i>	
	31:23	<i>Reserved.</i>	
	22:12	Address after end of Texture Memory, in units of 4KB (2×page size). <i>Default</i> to 0.	
	11:00	Must be zero, each buffer must be aligned to a natural 4KB boundary.	

Fast Status

Multiple bits and values are combined into one register for quick reading.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+1F0H	31:00	R/O	Fast Status
	<i>Bits</i>	<i>Field</i>	
	31	Driver Flag @ Register 026H bit [00].	
	30	Driver Flag @ Register 024H bit [00].	
	29	2D Signal @ Register 022H bit [00].	
	28	2D Synchronization @ Register 020H bit [00].	
	27-26	<i>Reserved.</i> Read as zero.	
	25	DMA Busy	
	24	3D Engine Busy	
	23:20	<i>Reserved.</i> Read as zero.	
	19:08	FIFO Status. Indicates how full the FIFO is in units of 256 bytes.	
	07:00	<i>Reserved.</i> Read as zero.	

Diagnostic Port Control

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+1FEH	15:00	R/W	Diagnostic Port Control
	<i>Bits</i>	<i>Field</i>	
	15:08	<i>Reserved.</i> Read as zero.	
	07:04	<i>Reserved.</i>	
	03:00	Selects diagnostic information to be accessed.	
		<i>Value</i>	<i>Semantics</i>
		0	Bypass output of previous unit. <i>Default.</i>
		1-FH	Select diagnostic signals.

4.2.2.9 Motion Compensation (+300H)

These fields specify the parameters needed to perform MPEG motion compensation and format conversion of displayable MPEG frames from 4:2:0 planar format to 4:2:2 interleaved format.

Motion Compensation Processing Registers

Offset	Field	Access	Function
+340H	31:00	R/W	Slot 0 Base Address
+360H	31:00	R/W	Slot 1 Base Address
+380H	31:00	R/W	Slot 2 Base Address
+3A0H	31:00	R/W	Slot 3 Base Address
	Bits	Field	
	31:24	Reserved.	
	23:00	Slot Base Address: byte address identifying the location of slot in memory. Located on a dword boundary (at minimum), so 2 lsb must be 0 .	
Offset	Field	Access	Function
+344H	31:00	R/W	Slot 0 Plane 1/ Plane 3 Offset
+364H	31:00	R/W	Slot 1 Plane 1/ Plane 3 Offset
+384H	31:00	R/W	Slot 2 Plane 1/ Plane 3 Offset
+3A4H	31:00	R/W	Slot 3 Plane 1/ Plane 3 Offset
	Bits	Field	
	31:25	Reserved.	
	24	Plane 1/ Plane 3 Identifier	
		Value	Semantics
		0	Offset for Plane 1. <i>Default</i> .
		1	Offset for Plane 3.
	Bits	Field	
	23:19	Reserved.	
	18:00	Plane 1 (first plane) or Plane 3 (interleaved chroma plane) offset from the start of the slot, in bytes. 6 lsb must be 0 .	
Offset	Field	Access	Function
+348H	31:00	R/W	Slot 0 Plane 2 Offset
+368H	31:00	R/W	Slot 1 Plane 2 Offset
+388H	31:00	R/W	Slot 2 Plane 2 Offset
+3A8H	31:00	R/W	Slot 3 Plane 2 Offset
	Bits	Field	
	31:19	Reserved.	
	18:00	Plane 2 (second chroma plane) offset from the start of the slot, in bytes. 6 lsb must be 0 . Not valid for interleaved chroma.	
Offset	Field	Access	Function
+34CH	31:00	R/W	Slot Scanlines
	Bits	Field	
	31:12	Reserved.	
	11:00	Number of scanlines in the luma plane (11:01 = number of scanlines divided by two).	

Slot Stride

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+36CH	31:00	R/W	Slot Stride
	<i>Bits</i>	<i>Field</i>	
	31:10	<i>Reserved.</i>	
	09:00	Luma (plane 0) stride, in bytes. 4 lsb must be 0.	
	(09:01)	Chroma (planes 1 and 2) stride, in bytes. (= luma stride/2) 3 lsb are guaranteed to be 0	
	(09:00)	Chroma (plane 3) stride, in bytes. (= luma stride) 3 lsb must be 0	

Motion Comp Busy

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+350H	7:00	R/O	MC Busy
	<i>Bits</i>	<i>Field</i>	
	7:02	<i>Reserved.</i>	
	01	Motion Compensation Busy Flag. Set by hardware.	
		<i>Value</i>	<i>Semantics</i>
		0	<i>Motion Compensation Idle.</i>
		1	<i>Motion Compensation in Progress.</i>
	<i>Bits</i>	<i>Field</i>	
	00	Format Conversion Busy Flag. Set by hardware	
		<i>Value</i>	<i>Semantics</i>
		0	<i>Format Conversion Idle.</i>
		1	<i>Format Conversion in Progress.</i>

4.2.2.10 Stipple Mask Array (+400H)

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+400H	31:00	R/W	Stipple Mask. 32×32 bit.
...+47CH			Occupies the first 32 doublewords of the address space.
	<i>Bits</i>	<i>Field</i>	
	31:00	Stipple Mask	

4.2.2.11 Texture Map LOD Table Array (+580H)

Eleven(11) entries. 28 of the 32 bits per entry are defined as described in the Texture Mapping Unit section.

The LOD Table is an 11× 32 bit array containing the parameters used to translate the U, V texture address to linear memory address.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+580H -5A8H	31:00	R/W	LOD Table Entry
	<i>Bits</i>	<i>Field</i>	
	31:28	<i>Reserved.</i>	
	27:25	Row_V_Shift. = max_0_Upge_M_L – pbye + 6	
	24:18	U - Texture Offset from Page Address (bits [06:00]).	
	17:11	V - Texture Offset from Page Address (bits [06:00]).	
	10:00	Page Address relative to Texel Base Page Register.	

Note: The least significant 28 bits are identical in format to the Texture Load Register in the host interface unit. Equations are: $\log_2 u = \text{ceiling}(\log_2(u))$.

4.2.2.12 Temporary Array for Triangle Setup Unit (+5C0H)

Sixteen(16) entries. These registers are provided for debugging purposes. It is a requirement that the 3D engine be idle when these registers are accessed. If the 3D Engine is not idle, corruption of internal data may result, and the read data is not a reliable indication of their contents.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+5C0H ...+5FCH	31:00	R/O	Triangle Setup Unit Temp Registers
	<i>Bits</i>	<i>Field</i>	
	31:24	<i>Reserved. Read as zero.</i>	
	23:00	Temporary Data	

Vertex List Array (+600H)

128 entries. These registers are provided for debugging purposes. It is imperative that the 3D Engine be idle when these registers are accessed. If the 3D Engine is not idle, corruption of internal data may result, and the read data is not a reliable indication of their contents.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+600H	31:00	R/O	Vertex List
...+7FCH			
	<i>Bits</i>	<i>Field</i>	
	31:24	<i>Reserved.</i> Read as zero.	
	23:00	Vertex Data	

Edge List Array (+800H)

256 entries. These registers are provided for debugging purposes. It is a requirement that the 3D Engine be idle when these registers are accessed. If the 3D Engine is not idle, corruption of internal data may result, and the read data is not a reliable indication of their contents.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+800H	31:00	R/O	Edge List
...+BFCH			
	<i>Bits</i>	<i>Field</i>	
	31:24	<i>Reserved.</i> Read as zero.	
	23:00	Edge List Data	

Texture Palette Array (+C00H)

256 entries.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+C00H	31:00	R/W	Texture Palette
...+FFCH			
	<i>Bits</i>	<i>Field</i>	
	31:24	<i>Reserved.</i> Read as zero.	
	23:18	Red Palette Data	
	17:16	<i>Reserved.</i> Read as zero.	
	15:10	Green Palette Data	
	09:08	<i>Reserved.</i> Read as zero.	
	07:02	Blue Palette Data	
	01:00	<i>Reserved.</i> Read as zero.	

4.2.3 3D Command/Parameter FIFO (1FF4000H)

This area is for the 3D command/parameter FIFO. No user access is required.

4.2.4 2D Command/Parameter FIFO (1FF8000H)

This area is for the 2D command/parameter FIFO. No user access is required.

4.2.5 2D Image Data FIFO (1FFC000H)

This area is for the 2D image data FIFO. No user access is required.

4.3 Standard VGA Registers

The standard VGA registers are I/O mapped for compatibility and accessed through pairs of index and data registers. They are also memory-mapped (unindexed) into a 256B address space for accessibility by processors without an I/O address space or direct access by x86 processors operating in protected mode. Standard VGA Control Registers with a 10-bit index register I/O are abbreviated **CRxx** (standard VGA compatible) or **ERxx** (enhanced).

The newer enhanced VGA registers are defined as offsets to the Graphics Controller Register (GRC). These new registers include:

- SM3110** Enhanced register access control
- VESA bank offset control
- Scratch registers

4.3.1 Register Locations

There are six sets of registers (grouped functionally) in the standard ISA (10-bit) I/O address space defined by the VGA specification. These are either individual registers or index/data register pairs to access the VGA control and configuration registers. All memory-mapped accesses have an address prefix (bits [31:16]) of *aaFFH*.

<i>Registers Group</i>	<i>VGA</i>	<i>MDA Emulation</i>	<i>CGA Emulation</i>	<i>Comments</i>
General	3C2,3C3,3CA,3CC	3BA	3DA	
CRTC		3B4/3B5	3D4/3D5	
Sequencer	3C4/3C5			Index/data
Graphics Controller	3CE/3CF			Index/data
DAC	3C6-3C9			Index/data
Attribute	3C0/3C1			Index/data

In the VGA address space, most registers employ an index/data scheme to reduce the number of I/O addresses required.

	07	00 07	00	
+C2H	Video Enable	Rd: Input Status Wr: Misc Output		General Control
+CAH		Feature Control		General Control
+CCH		Rd: Misc Output		General Control
+BA/DAH		Rd: Input Status Wr: Feat Control		General Control (MDA/CGA)
+B4/D4H	Data	Index		CRTC (MDA/CGA), indexed
+C4H	Data	Index		Sequencer, indexed
+CEH	Data	Index		Graphics Controller, indexed
+C6H	Read Index/Status	Mask		DAC
+C8H	Data	Write Index		
+C0H	Data	Write Index		Attributes
	+1	+0		

Figure 4-23. VGA Control Registers

4.3.2 General Registers

The following section describes the standard VGA control registers arranged by index value. A register at I/O address *aaaa*, index *xx* is designated *aaaaCRxx*.

<i>Register</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
3C3H	07:00	R/W	Video Subsystem Enable
	<i>Bits</i>	<i>Field</i>	
	07:01	<i>Reserved.</i>	
	00	Access Enable	
		<i>Value</i>	<i>Semantics</i>
		0	Disables VGA subsystem (sleep mde).
		1	Enables VGA subsystem (wake mode).
<i>Register</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
3C2H	07:00	W/O	Miscellaneous Output
3CCH	07:00	R/O	Miscellaneous Output
	<i>Bits</i>	<i>Field</i>	
	07	Vertical Sync Polarity	
	06	Horizontal Sync Polarity	
		<i>Value</i>	<i>Semantics</i>
		0	Positive.
		1	Negative.
	<i>Bits</i>	<i>Field</i>	
	07:06	Vertical Size. Bits 7 and 6 are determined by the vertical size in VGA modes	
		<i>Value</i>	<i>Semantics</i>
		0	<i>Reserved.</i>
		1	400 Lines.
		2	350 Lines.
		3	480 Lines.
	<i>Bits</i>	<i>Field</i>	
	05	Odd/Even Page Bit. Selects between two 64KB pages of local memory in Odd/Even modes.	
		<i>Value</i>	<i>Semantics</i>
		0	Even Page.
		1	Odd Page.
	<i>Bits</i>	<i>Field</i>	
	04	<i>Reserved.</i> Read as zero.	
	03:02	Clock Select	
		<i>Value</i>	<i>Semantics</i>
		0	640 horizontal pixels. (Color 25.175MHz dot clock).
		1	720 horizontal pixels. (Mono 28.322MHz dot clock).
		2	<i>Reserved.</i>
		3	Dot clock selected in mode control register (CR42), bits [3:0].

<i>Bits</i>	<i>Field</i>
01	Enable Memory Access to 000A0000-000BFFFFH.
<i>Value Semantics</i>	
0	Disables host memory access.
1	Enables host memory access.
<i>Bits</i>	<i>Field</i>
00	I/O Address Select.
<i>Value Semantics</i>	
0	MDA emulation. CRTIC at 3B4/3B5H, Input Status 1 at 3BAH
1	CGA emulation. CRTIC at 3D4/3D5H, Input Status 1 at 3DAH

<i>Register</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
3C2H	07:00	R/O	Input Status 0
<i>Bits Field</i>			
	07	Vertical Retrace Interrupt	
<i>Value Semantics</i>			
	0	Interrupt not pending or cleared.	
	1	Interrupt pending.	
<i>Bits Field</i>			
	06:05	<i>Reserved.</i>	
	04	Monitor Sense Input Status	
<i>Value Semantics</i>			
	0	Sense input is 0.	
	1	Sense input is 1.	
	03:00	<i>Reserved.</i>	

<i>Register</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
3BAH	07:00	R/O	Input Status 1 (mono emulation)
3DAH	07:00	R/O	Input Status 1 (color emulation)
<i>Bits Field</i>			
	07:04	<i>Reserved.</i> Read as zero	
	03	Vertical Retrace	
<i>Value Semantics</i>			
	0	Vertical display enabled.	
	1	Vertical Retrace Interval.	
<i>Bits Field</i>			
	02:01	<i>Reserved.</i> Read as zero.	
	00	Display Enable Status (inverted/active low)	
<i>Value Semantics</i>			
	0	Display Enabled.	
	1	Vertical or Horizontal Retrace Interval.	

<i>Register</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
3BAH	07:00	W/O	Feature Control (mono emulation). Must be zero.
3DAH	07:00	W/O	Feature Control (color emulation). Must be zero.
3CAH	07:00	R/O	Feature Control. Read as zero.

4.3.3 CRT Controller (CRTC)

The horizontal and vertical timing of the CRT is specified by several standard VGA and extended control registers located at I/O address 3B4H (MDA emulation) or 3D4H (CGA emulation). Selection of B or D as address bits [07:04] is determined by bit [0] of the Miscellaneous Output register at 3C2H.

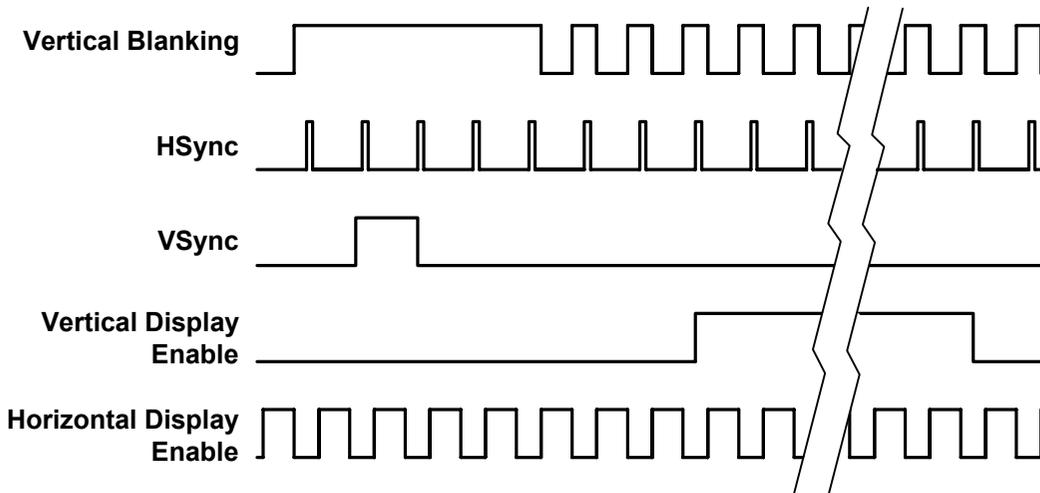


Figure 4-24. VGA CRTC Timing

4.3.3.1 VGA CRTC Resolution Limitations

The VGA compatible CRTC definition allows a maximum resolution of 256 characters (2048 pixels with 8 pixels/character) by 1024 scan lines. This is the maximum display resolution including blanking intervals.

4.3.3.2 CRTC Index/Data Registers

The CRTC index register is at 03B4 (MDA emulation) or 03D4H (CGA emulation) and the data port is at 3B5 or 03D5H.

<i>Register</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
3B4/3D4H	07:00	R/W	Index
	<i>Bits</i>	<i>Field</i>	
	07:05	<i>Reserved.</i> Must be zero.	
	04:00	CRTC Register Index	

<i>Register</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
3B5/3D5H	07:00	R/W	Data
	<i>Index</i>	<i>Field</i>	<i>Access</i>
	00H	07:00	R/W
	01H	07:00	R/W
	02H	07:00	R/W
	03H	07:00	R/W
	04H	07:00	R/W
	05H	07:00	R/W
	06H	07:00	R/W
	07H	07:00	R/W
	08H	07:00	R/W
	09H	07:00	R/W

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
0AH	07:00	R/W	Cursor Start Scan Line
	<i>Bits</i>	<i>Field</i>	
	07:06	<i>Reserved.</i> Read as zero.	
	05	Text Cursor Enable	
		<i>Value</i>	<i>Semantics</i>
		0	Disables the text cursor.
		1	Enables the text cursor.
	<i>Bits</i>	<i>Field</i>	
	04:00	Cursor Start Scan Line	
		<i>Value</i>	<i>Semantics</i>
		<i>n</i>	Scan line within a character row on which the cursor starts. If greater than the cursor end line (CR0B) then no cursor is displayed.

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
0BH	07:00	R/W	Cursor End Scan Line
	<i>Bits</i>	<i>Field</i>	
	07	<i>Reserved.</i> Read as zero.	
	06:05	Cursor Skew	
		<i>Value</i>	<i>Semantics</i>
		<i>n</i>	Specifies the delay in character positions of the cursor (to the right).
	<i>Bits</i>	<i>Field</i>	
	04:00	Cursor End Scan Line	
		<i>Value</i>	<i>Semantics</i>
		<i>n</i>	Scan line within a character row on which the cursor ends. If less than the cursor start line (CR0A) then no cursor is displayed.

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
0CH	07:00	R/W	Start Address (upper), <i>Default</i> is undefined.
0DH	07:00	R/W	Start Address (lower), <i>Default</i> is undefined.
0EH	07:00	R/W	Cursor Location (upper), <i>Default</i> is undefined.
0FH	07:00	R/W	Cursor Location (lower), <i>Default</i> is undefined.
10H	07:00	R/W	Vertical Sync Start
11H	07:00	R/W	Vertical Sync End

Bits *Field*

07	Lock writes to CR00-07H. Set to 1 by the BIOS during a mode set, reset or power on.
----	-------------------------------------------------------------------------------------

Value *Semantics*

0	Enable writes to all CRTC registers. <i>Default</i> .
1	Disable writes to CRTC registers CR00-07H, except CR07[4].

Bits *Field*

06	Memory Refresh Select Cleared to 0 by the BIOS during a mode set, reset or power on.
----	-----------------------------------------------------------------------------------------

Value *Semantics*

0	Three memory refresh cycles per horizontal scan line. <i>Default</i> .
1	Five memory refresh cycles per horizontal scan line.

Bits *Field*

05	Vertical Interrupt Disable Cleared to 0 by the BIOS during a mode set, reset or power on.
----	----------------------------------------------------------------------------------------------

Value *Semantics*

0	Vertical Interrupt enabled. <i>Default</i> .
1	Vertical Interrupt disabled.

Bits *Field*

04	Clear Vertical Interrupt Cleared to 0 by the BIOS during a mode set, reset or power on.
----	--------------------------------------------------------------------------------------------

Value *Semantics*

0	Vertical Interrupt cleared. <i>Default</i> .
1	Vertical Interrupt enabled. Other bits in this register should not be changed while writing to this bit. This allows the interrupt flip-flop to be set at the start of a vertical sync pulse.

Bits *Field*

03:00	Vertical Sync End
-------	-------------------

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
12H	07:00	R/W	Vertical Display End
13H	07:00	R/W	Screen Offset (Pitch) , <i>Default</i> is undefined. Specifies the difference between the starting byte address of two consecutive scan lines. The value is multiplied by 2 (byte), 4 (word) or 8 (doubleword mode) specified by bit 6 of CR14 and bit 3 of CR17.

Bits Field

07:00	Logical Screen Width, maximum of 512 (byte), 1024 (word) or 2048 (doubleword mode) bytes.
-------	-------------------------------------------------------------------------------------------

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
14H	07:00	R/W	Underline Location

Bits Field

07	<i>Reserved</i> . Read as zero.
06	Doubleword Mode

Value Semantics

0	Byte or Word Memory Addressing. <i>Default</i> .
1	Doubleword Memory Addressing.

Bits Field

05	Count by 4 Mode
----	-----------------

Value Semantics

0	The memory address counter operation depends on CR17, bit 3(count by 2). <i>Default</i> .
1	Memory Address Counter is incremented every four character clocks.

Bits Field

04:00	Underline Location within a character row.
-------	--------------------------------------------

Value Semantics

<i>n-1</i>	The scan line -1 (counted from the top) on which the underline is displayed. <i>Default</i> .
------------	--------------------------------------------------------------------------------------------------

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
15H	07:00	R/W	Vertical Blank Start
16H	07:00	R/W	Vertical Blank End
17H	07:00	R/W	CRTC Mode Control

<i>Bits</i>	<i>Field</i>
07	Hardware Reset

Value Semantics

0	Vertical and Horizontal Sync pulses disabled. <i>Default.</i>
1	Vertical and Horizontal Sync pulses enabled.

Bits Field

06	Byte Mode
----	-----------

Value Semantics

0	Word Mode. <i>Default.</i> The memory address counter bits are shifted up (towards the MSB) one bit position.
1	Byte Address Mode.

Bits Field

05:04	<i>Reserved. Default is 0.</i>
-------	--------------------------------

Value Semantics

0	Disabled. <i>Default.</i>
1	Enabled.

Bits Field

02	Vertical Total Double Mode. Specifies the source of the clock/enable for the vertical timing counter.
----	-------------------------------------------------------------------------------------------------------

Value Semantics

0	Horizontal Sync. <i>Default.</i>
1	Horizontal Sync $\div 2$. Double vertical resolution.

Bits Field

01:00	<i>Reserved. Default is 0.</i>
-------	--------------------------------

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
18H	07:00	R/W	Line Compare
22H	07:00	R/O	CPU Data Latch
26H	07:00	R/W	Attribute
2FH	07:00	R/W	CRTC Test Functions
<i>Bits</i>		<i>Field</i>	
07	Enable Test Clock to upper 5 bits of Vertical Counter.		
06	Enable Test Clock to upper 4 bits of Vertical Counter.		
05	Enable Test Clock to upper 5 bits of Horizontal Counter.		
04	Enable Test Clock to upper 5 bits of Horizontal Counter.		
<i>Semantics</i>			
Any I/O Read will cause the enabled section of the vertical or horizontal counter to be clocked.			
<i>Bits</i>		<i>Field</i>	
03:01	<i>Reserved.</i>		
00	Force Sync Polarity HIGH		
<i>Value</i>		<i>Semantics</i>	
0	Normal Sync Polarity (obey Misc. Output Register settings). <i>Default.</i>		
1	Force Sync Polarity HIGH (override of Misc. Output Register settings).		

4.3.4 Sequencer

Index/Data Port

<i>Register</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
3C4H	07:00	R/W	SEQ Index, Default is undefined.
	<i>Bits</i>	<i>Field</i>	
	07:04	<i>Reserved.</i> Read as zero.	
	03:00	Sequencer Index	
<i>register</i>	<i>field</i>	<i>access</i>	<i>function</i>
3C5H	07:00	R/W	SEQ Data, Default is undefined.

Reset

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
0H	07:00	R/W	Reset. These bits have no effect on sequencer or VGA operation.
	<i>Bits</i>	<i>Field</i>	
	07:02	<i>Reserved.</i> Must be zero.	
	01	Synchronous Reset	
	00	Asynchronous Reset	
	<i>Value</i>	<i>Semantics</i>	
	0	Reset asserted.	
	1	Reset not asserted.	

Clocking Mode

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
1H	07:00	R/W	Clocking Mode
	<i>Bits</i>	<i>Field</i>	
	07:06	<i>Reserved.</i> Must be zero.	
	05	Screen Off.	
	04	Shift 4.	
	03	Dot Clock	
	02	Shift Load	
	01	<i>Reserved.</i> Must be zero.	
	00	8/9 Pixel Characters	
	<i>Value</i>	<i>Semantics</i>	
	0	9 Pixel Characters. For text modes only.	
	1	8 Pixel Characters. For text or graphics modes.	

Map Mark

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
2H	07:00	R/W	Map Mask. All maps should be enabled in Chain 4 or Chain 8 modes.
	<i>Bits</i>	<i>Field</i>	
	07:04	<i>Reserved.</i> Must be zero.	
	03	Map Mask. Plane 3 (byte 3 of a doubleword access).	
	02	Map Mask. Plane 2 (byte 2 of a doubleword access).	
	01	Map Mask. Plane 1 (byte 1 of a doubleword access).	
	00	Map Mask. Plane 0 (byte 0 of a doubleword access).	
		<i>Value</i>	<i>Semantics</i>
		0	Disables host access to the corresponding map.
		1	Enables host access to the corresponding map.

Character Map Select

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
3H	07:00	R/W	Character Map Select
	<i>Bits</i>	<i>Field</i>	
	07:06	<i>Reserved.</i> Must be zero.	
	05	Character Map Select High Bit A.	
	04	Character Map Select High Bit B.	
	03:02	Character Map Select A.	
	01:00	Character Map Select B.	
		<i>Value</i>	<i>Semantics</i>
		0	Disables host access to the corresponding map.
		1	Enables host access to the corresponding map.

Memory Mode

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
4H	07:00	R/W	Memory Mode
	<i>Bits</i>	<i>Field</i>	
	07:04	Reserved. Must be zero.	
	03	Chain 4.	
		<i>Value</i>	<i>Semantics</i>
		0	Data is sequentially accessed within the bit map according to the Map Mask register.
		1	Address [01:00] select the map used, A[01:00] = 0 selects map 0, A[01:00] selects map 3.
	<i>Bits</i>	<i>Field</i>	
	02	Odd/Even	
		<i>Value</i>	<i>Semantics</i>
		0	Odd/Even plane addressing. Accesses to even addresses are to planes 0 and 2, odd addresses are to planes 1 and 3.
		1	Data is sequentially accessed within the bit map according to the Map Mask register.
	<i>Bits</i>	<i>Field</i>	
	01	Extended Memory.	
		<i>Value</i>	<i>Semantics</i>
		0	Only 64KB of memory is available.
		1	More than 64KB of memory is available.
	<i>Bits</i>	<i>Field</i>	
	00	Reserved. Must be zero.	

Reset VGA

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Register</i>
07H	07:00	W/O	VGA Reset
	<i>Bits</i>	<i>Field</i>	
	07:00	Reset VGA backend (attribute and CRT controllers).	

4.3.5 Graphics Controller

Index/Data Port

<i>Register</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
3CEH	07:00	R/W	GRC Index <i>Default is undefined.</i>
	<i>Bits</i>	<i>Field</i>	
	07:04	<i>Reserved.</i>	Read as zero.
	03:00	Graphics Controller Index	
<i>Register</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
3CFH	07:00	R/W	GRC Data, <i>Default is undefined.</i>

Set/Reset Data

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
00H	07:00	R/W	Set/Reset Data Contains the color data written to each byte of the memory plane during a host write in write modes 0 and 3.
	<i>Bits</i>	<i>Field</i>	
	07:04	<i>Reserved.</i>	Read as zero.
	03:00	Set Reset Data	

Enable Set/Reset Data

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
01H	07:00	R/W	Enable Set/Reset Data
	<i>Bits</i>	<i>Field</i>	
	07:04	<i>Reserved.</i>	Read as zero.
	03:00	Enable Set Reset Data	

Color Compare

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
02H	07:00	R/W	Color Compare <i>Default</i> is undefined. In read mode 1, the data at the address specified by the host is XNORed with the color compare register contents and the result placed in the color compare register. A ‘1’ indicates a match in the corresponding bit position.
	<i>Bits</i>	<i>Field</i>	
	07:04		<i>Reserved.</i> Read as zero.
	03:00		Color Compare Data
		<i>Value</i>	<i>Semantics</i>
		0	Write: data to be compared. Read: No match.
		1	Write: data to be compared. Read: Match

Raster Op/Rotate Count

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
03H	07:00	R/W	Raster Operation/Rotate Count <i>Default</i> is undefined.
	<i>Bits</i>	<i>Field</i>	
	07:05		<i>Reserved.</i> Read as zero.
	04:03		Raster OP.
		<i>Value</i>	<i>Semantics</i>
		0	No operation.
		1	Logical AND with data in processor latches.
		2	Logical OR with data in processor latches.
		3	Logical XOR with data in processor latches.
	<i>Bits</i>	<i>Field</i>	
	02:00		Rotate Count.
		<i>Value</i>	<i>Semantics</i>
		0	No rotation.
		1-7	Number of positions to right rotate.

Read Plane Select

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
04H	07:00	R/W	Read Plane Select <i>Default</i> is undefined. This specifies the memory plane that will be accessed by host reads in read mode 1. It has no effect on the color compare, read mode 1. In odd/even mode, bit 0 is ignored.
	<i>Bits</i>	<i>Field</i>	
	07:02		<i>Reserved.</i> Read as zero.
	01:00		Select
		<i>Value</i>	<i>Semantics</i>
	0-3		Plane 0 to 3, respectively.
	2		Write mode 2. Memory planes 3-0 are written with 8 bits of the host write data bits [3:0], respectively.
	3		Write mode 3. All four memory planes are written with 8 bits of the color value contained in the set/reset register for that plane. The enable set/reset register has no effect. Rotated host write data is ANDed with the bit mask register to generate an 8 bit value that performs the same function as the bit mask register in write modes 0 and 2.

Graphics Controller Mode

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
05H	07:00	R/W	Graphics Controller Mode <i>Default is undefined.</i>
	<i>Bits</i>	<i>Field</i>	
	07	<i>Reserved.</i> Read as zero.	
	06	256 Color Shift Mode	
		<i>Value</i>	<i>Semantics</i>
		<i>e</i>	
	0	Disable	
	1	Enable	
	<i>Bits</i>	<i>Field</i>	
	05	Shift Mode	
		<i>Value</i>	<i>Semantics</i>
		<i>e</i>	
	0	Disable	
	1	Enable	
	<i>Bits</i>	<i>Field</i>	
	04	Odd/Even Addressing	
		<i>Value</i>	<i>Semantics</i>
		<i>e</i>	
	0	Disable	
	1	Enable	
	<i>Bits</i>	<i>Field</i>	
	03	Read Mode Select	
		<i>Value</i>	<i>Semantics</i>
		<i>e</i>	
	0	Read mode 0. Read data is accessed by the host selected by the Plane Select register.	
	1	Read mode 1. Color compare using the Color Compare and Color Don't Care registers.	
	<i>Bits</i>	<i>Field</i>	
	02	<i>Reserved.</i> Read as zero.	
	01:00	Write Mode Select	
		<i>Value</i>	<i>Semantics</i>
		<i>e</i>	
	0	Write mode 0. Direct host access.	
	1	Write mode 1. All four memory planes are written with data from the processor latches. Raster OPs, rotate count, set/reset data/enable and bit mask registers do not affect the write operation.	

Memory Map Mode

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
06H	07:00	R/W	Memory Map Mode
	<i>Bits</i>	<i>Field</i>	
	07:04	<i>Reserved.</i> Read as zero.	
	03:02	Memory Map specifies the size and starting address of the VGA aperture that is mapped to the local memory array starting at the address specified in the VESA bank register.	
		<i>Value</i>	<i>Semantics</i>
		0	000A0000-BFFFFH (128KB).
		1	000A0000-AFFFFH (64KB).
		2	000B0000-B7FFFFH (32KB).
		3	000B8000-BFFFFH (32KB).
	<i>Bits</i>	<i>Field</i>	
	01	Chain Odd/Even Planes	
		<i>Value</i>	<i>Semantics</i>
		0	Address bit [00] is unchanged.
		1	Host address bit A[00] is replaced by a higher order address bit: A[00] then specifies which plane is selected. A[00]=0 selects the even planes (0/2), A[00]=1 selects the odd planes (1/3).
	<i>Bits</i>	<i>Field</i>	
	00	Text/Graphics Mode	
		<i>Value</i>	<i>Semantics</i>
		0	Text Mode display addressing.
		1	Graphics Mode display addressing. The character generator address latches are disabled.

Color Don't Care

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
07H	07:00	R/W	Color Don't Care <i>Default</i> is undefined. In read mode 1 this performs comparisons.
	<i>Bits</i>	<i>Field</i>	
	07:04		<i>Reserved.</i> Read as zero.
	03:00		Color Plane Select
		<i>Value</i>	<i>Semantics</i>
		0	Ignore the data from the corresponding color plane.
		1	Use the data from the corresponding color plane when performing a color compare operation.
<i>index</i>	<i>field</i>	<i>access</i>	<i>function</i>
08H	07:00	R/W	Bit Mask

DAC Mask

<i>register</i>	<i>field</i>	<i>access</i>	<i>function</i>
3C6H	07:00	R/W	DAC Mask <i>Default</i> is undefined.

Palette State

Reads and writes to the Palette are performed as a sequence of three byte accesses. An internal 3-state sequencer generates the selection of the appropriate byte of the palette to be accessed. If this sequence is interrupted, and another software process needs to change the palette, the state of the access sequence must be saved along with the palette contents. The sequencer state is accessible in this register.

<i>Register</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
3C7H	07:00	Write	DAC Read Index <i>Default</i> is undefined.
		Read	DAC Status <i>Default</i> is undefined.
	<i>Bits</i>	<i>Field</i>	
	07:02		<i>Reserved.</i> Read as zero.
	01:00		Cycle Status
		<i>Value</i>	<i>Semantics</i>
		0	Write Cycle.
		3	Read Cycle.
<i>register</i>	<i>field</i>	<i>access</i>	<i>function</i>
3C8H	07:00	R/W	DAC Write Index <i>Default</i> is undefined.
3C9H	07:00	R/W	Palette Data <i>Default</i> is undefined.

4.3.6 Attribute Controller

Index/Data Port

<i>Register</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
3C0H	07:00	R/W	ATC Index <i>Default is undefined.</i>
	<i>Bits</i>	<i>Field</i>	
	07:06	<i>Reserved.</i>	
	05	Palette Address Source	
		<i>Value</i>	<i>Semantics</i>
		0	Disable display access to VGA palette registers at index [0...FH]. <i>Default.</i> This enables host access to the palette registers.
		1	Enable access to VGA palette registers for normal display operation. The host cannot access the palette registers.
	<i>Bits</i>	<i>Field</i>	
	04:00	Attribute Controller Index	
<i>Register</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
3C1H	07:00	R/W	ATC Data <i>Default is undefined.</i>
<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
00-0FH	05:00	R/W	Palette
	<i>Bits</i>	<i>Field</i>	
	07:06	<i>Reserved. Must be zero.</i>	
	05:03	Secondary Color, R, G, B, respectively	
	02:00	Primary Color, R, G, B, respectively	

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
10H	07:00	R/W	Attribute Mode Control
	<i>Bits</i>	<i>Field</i>	
	07	Video bits [5:4] Mode	
	06	256 Color Mode	
		<i>Value</i>	<i>Semantics</i>
		0	4b pixels are translated to 6b through the VGA palette. <i>Default.</i>
		1	Two consecutive 4b pixels are assembled into an 8b pixel at half the internal DCLK.
	<i>Bits</i>	<i>Field</i>	
	05	PEL Panning Force to Zero	
		<i>Value</i>	<i>Semantics</i>
		0	No effect on panning register. <i>Default.</i>
		1	A successful line-compare operation forces the output of the PEL Panning register to zero until VSYNC occurs, then returns the output to the programmed value.
	<i>Bits</i>	<i>Field</i>	
	04	<i>Reserved.</i> Read as zero.	
	03	Blink	
		<i>Value</i>	<i>Semantics</i>
		0	Use background intensity for text attribute. <i>Default.</i>
		1	Use blink attribute for text mode.
	<i>Bits</i>	<i>Field</i>	
	02	Line Graphics	
		<i>Value</i>	<i>Semantics</i>
		0	The 9th pixel of a text character is same as the background. <i>Default.</i>
		1	Special line graphics character codes are enabled. The 9th pixel is the same as the 8th if the if the character code is C0-DFH.
	<i>Bits</i>	<i>Field</i>	
	01	Mono	
		<i>Value</i>	<i>Semantics</i>
		0	Use color display text attributes. <i>Default.</i>
		1	Use mono display text attributes.
	<i>Bits</i>	<i>Field</i>	
	00	Graphics	
		<i>Value</i>	<i>Semantics</i>
		0	Text attribute control mode. <i>Default.</i>
		1	Graphics control mode.

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
11H	07:00	R/W	Border Color
12H	07:00	R/W	Color Plane Enable

<i>Bits</i>	<i>Field</i>
07:06	<i>Reserved.</i> Read as zero.
05:04	Video Test. <i>Default</i> is 0.
03	Display Plane 3 Enable. <i>Default</i> is 0.
02	Display Plane 2 Enable. <i>Default</i> is 0.
01	Display Plane 1 Enable. <i>Default</i> is 0.
00	Display Plane 0 Enable. <i>Default</i> is 0.

<i>Value</i>	<i>Semantics</i>
0	Forces the output of the corresponding color plane to '0' before accessing the VGA palette. <i>Default.</i>
1	Enables the output of the corresponding color plane.

<i>index</i>	<i>field</i>	<i>access</i>	<i>function</i>
13H	03:00	R/W	Horizontal Pixel Panning

<i>Bits</i>	<i>Field</i>
07:04	<i>Reserved.</i> Read as zero.
03:00	Pan Shift Amount. <i>Default</i> is 0. The number of pixels panned depends on the mode.

<i>index</i>	<i>field</i>	<i>access</i>	<i>function</i>
14H	03:00	R/W	Pixel Padding

<i>Bits</i>	<i>Field</i>
07:04	<i>Reserved.</i> Read as zero.
03:02	Pixel Padding, the two MSBs of the 8b color output for all except 256 color mode. <i>Default</i> is 0.
01:00	Pixel Padding, V5/V4. <i>Default</i> is 0.

4.3.7 Enhanced VGA Registers

The small number of enhanced VGA registers are accessed thru the GRC index/data port of 3CE/3CFH. These registers are located at offsets +80-8FH of the GRC port (see Section 4.3.5, Graphics Controller).

4.3.7.1 Enhanced Register Access Enable

To prevent inadvertent access to the enhanced I/O mapped registers, a specific sequence must be written through the VGA Graphics Controller index/data port to enable access. Disabling is accomplished by writing a single byte to the same index/data port. Determination of the Locked/Unlocked state can be made by reading the port. The sequence of unlock writes must not be interrupted by any other accesses to the Graphics Controller index/data port: either the code sequence should be executed with interrupts disabled or the sequence should be performed interruptible and the lock state read to determine if the unlock occurred. If not it should be repeated until the lock state indicates unlocked. Reading the register returns the last value written.

Notes: The unlock sequence is triggered by writing the correct data to the data port only. If intervening accesses to other index/data ports are performed, the unlock state machine is not affected.

The unlock sequence can be performed by writing index(80H)/data("S"), index(80H)/data("M") or index(80H)/data("S")/data("M").

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
80H	07:00	R/W	Enhanced Register Access Enable (LOCK).
	<i>Bits</i>	<i>Field</i>	
	07:00	Key	
		<i>Value</i>	<i>Semantics</i>
		53H	ASCII 'S', upper case, first unlock key in sequence.
		4DH	ASCII 'M', upper case, second unlock key in sequence.
		<i>All other</i>	Lock. Any value except the correct value in the unlock sequence resets the state machine.
		4CH	ASCII 'L', upper case to Lock.
<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
81H	07:00	R/O	Enhanced VGA Register Access State.
	<i>Bits</i>	<i>Field</i>	
	07:01	<i>Reserved.</i>	
	00	Lock State	
		<i>Value</i>	<i>Semantics</i>
		0	Unlocked.
		1	Locked.

4.3.7.2 Enhanced Control Access

Access to the enhanced control (register) space from real mode is controlled by one register. When enhanced register access is enabled, local memory cannot be accessed through 000A0000-000BFFFFH. The 64KB control space is mapped from 000A0000-000AFFFFH. The size and base specified for normal VGA accesses in 3CFH, index 06H, bits [3:2] is ignored. *This works only for little-endian access to registers; big-endian access can only be performed directly through the linear aperture.*

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
82H	07:00	R/W	Control Access Bank.
	<i>Bits</i>	<i>Field</i>	
	07	Memory mapped Enhanced Register access	
		<i>Value</i>	<i>Semantics</i>
		0	Disabled. <i>Default.</i> Access to this range returns the memory contents.
		1	Enabled.
	<i>Bits</i>	<i>Field</i>	
	06:03	<i>Reserved.</i>	
	02:00	Control Space Access Bank Offset. This selects the 64K bank within the 512KB control space at the top of the first 8MB local address space. The immediate control register space is at <i>bbFFxxxxH</i> , the offset should be set to 7, for convenience bits[03:00] may be set to FH, i.e, bit[03] is ignored.	

4.3.7.3 Bank Offset

In S/VGA modes the amount of memory that is addressable must be increased to support higher resolution displays. A switching of a 64KB bank with a 32KB granularity allows continuous access to the entire physical address space. This 64KB segment is accessed at the standard VGA address space of 000A0000H. The most significant bit allows specification of either VGA-mode access or banked, packed-pixel access to local memory. The latter is useful for VESA graphics modes or packed-pixel access to local memory for debug.

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
84H	07:00	R/W	Bank Offset. For accesses to the VGA space. The size and start address are determined by 3CFH, index 06H, bits [3:2].
	<i>Bits</i>	<i>Field</i>	
	07	VGA Aperture Local Memory Access.	
		<i>Value</i>	<i>Semantics</i>
		0	Normal, banked, VGA defined mode. <i>Default.</i>
		1	Banked, packed pixel access to Local Memory, bypass VGA GRC.
	<i>Bits</i>	<i>Field</i>	
	06:00	Address Offset in units of 32KB. <i>Default</i> is 00H. These seven bits are <i>added</i> to host address A[21:15]. The resulting 7 bits are concatenated with A[14:00] of the host address to form a 22b linear address [21:00] allowing access anywhere within the 4MB maximum local physical memory address space.	

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
85H	07:00	R/W	Dual Palette RAM Selection
	<i>Bits</i>	<i>Field</i>	
	07	Palette Ram Selection.	
		<i>Value</i>	<i>Semantics</i>
		0	Select display 0 palette RAM. <i>Default.</i>
		1	Select display 1 palette RAM.
	<i>Bits</i>	<i>Field</i>	
	06:02	<i>Reserved</i>	
	01:00	Address offsets in units of 4MB. <i>Default</i> is 00B.	

The inputs to the address offset adder are defined below:

<i>Value</i>	<i>Size</i>	<i>Base</i>	<i>Adder A Input</i>	<i>AdderB Input</i>	<i>Comments</i>
00	128KB	000A0000H	ER[84][5:0]	000B A[16:15]	
01	64KB	000A0000H	ER[84][5:0]	0000B A[15]	
10	32KB	000B0000H	ER[84][5:0]	00000B	No carry will propagate
11	32KB	000B8000H	ER[84][5:0]	00000B	No carry will propagate

4.3.7.4 User-Defined Scratch Registers

There are 8 bytes of user-defined scratch registers provided for BIOS or display driver use. The definitions of these registers are given in Section 5, BIOS Specifications

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
88-8FH	07:00	R/W	User Defined.

Note: **0x8Eh** must contain the current SM3110 video mode number. If the video mode is changed using BIOS routines, then the BIOS will update the number automatically. If the video mode is changed by programming registers, then this register must be updated by the programmer at that time.

4.3.7.5 CRTC Test

Additional facilities are provided for testing that are enabled only when the VGA enhanced registers are unlocked. This is located in the VGA CRTC I/O space at **3B4/3D4H**.

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
2FH	07:00	R/W	CRTC Test Functions

Bits *Field*

07	Enable Test Clock to upper 5 bits of Vertical Counter.
06	Enable Test Clock to upper 4 bits of Vertical Counter.
05	Enable Test Clock to upper 5 bits of Horizontal Counter.
04	Enable Test Clock to upper 5 bits of Horizontal Counter.

Semantics

Any I/O Read will cause the enabled section of the vertical or horizontal counter to be clocked.

Bits *Field*

03:01	<i>Reserved.</i>
00	Force Sync Polarity HIGH

Value *Semantics*

0	Normal Sync Polarity (obey Misc. Output Register settings). <i>Default.</i>
1	Force Sync Polarity HIGH (override of Misc. Output Register settings).

5 BIOS Specifications



5.1 Overview

The SM3110 BIOS is fully compatible with standard VGA, VBE/PM version 1.0, DDC Level 2B, and VESA VBE version 2.0. The SM3110 BIOS also supports single or multiple display adapter configurations, as well as simultaneous and dual view configurations. The SM3110 BIOS requirements, scratchpad register usage and a detailed description of the VGA BIOS, VESA BIOS extensions and SM3110 BIOS extensions are also described here.

Software and Hardware Environments

The SM3110 BIOS operates in the standard IBM PC/AT (X-86) software environment. It operates in conjunction with the Microsoft MS-DOS or equivalent operating systems. The SM3110 BIOS is designed to support the MS Windows 95/98 and MS Windows NT operating systems, implementing standard VGA, SVGA, and VBE modes.

The SM3110 BIOS operates in either the PCI or AGP bus environments. The SM3110 BIOS fits into either 32KB or 64KB ROM and is designed to be easily customized to fit customers' needs.

Support Capabilities

The SM3110 supports multiple display adapters, bus environments and video modes.

Multiple Display Adapter Support

Although the SM3110 is used primarily in systems without other display adapters, the SM3110 BIOS is compatible with multiple adapter configurations supported by Windows 98 and Windows NT.

System Bus Support

The SM3110 BIOS supports two different bus environments: AGP and PCI. The PCI header is implemented in the ROM image. The PCI System BIOS reads and writes PCI configuration registers.

BIOS Modes Supported

The following modes are supported by the SM3110 BIOS. The BIOS provides mode switching for all of the video modes listed in Table 5-1.

Table 5-1. Supported BIOS Modes

Type	VESA Mode No.	SM3110 Mode No.	Width	Height	Text / bpp	Colors	Refresh Rates
VGA	0		40	25	40X25	64, 16 gray	70
VGA	1	0, 1	40	25	40X25	64, 16/8 color	70
VGA	2		80	25	80X25	64, 16 gray	70
VGA	3	2, 3	80	25	80X25	64, 16/8 color	70
VGA	4		320	200		4(256)	70
VGA	5	4, 5	320	200		4, gray	70
VGA	6	6	640	200		2, gray	70
VGA	7	7	80	25		2, mono	70
VGA	D	D	320	200		16	70
VGA	E	E	640	200		16 planar	70
VGA	F	F	640	350		mono	70
VGA	10	10	640	350		16, 64	70
VGA	11	11	640	480	80X25	2	60
VGA	12	12	640	480	80X25	16 planar	60
VGA	13	13	320	200	40X25	256 linear	60
VESA	100	5E	640	400	8	256	60, 85
VESA	101	5F	640	480	8	256	60, 72, 75, 85
VESA	102	58, 6A	800	600	4	16	56, 60, 72, 75, 85
VESA	103	5C	800	600	8	256	56, 60, 72, 75, 85
VESA	104	5D	1024	768	4	16	60, 70, 75, 85
VESA	105	60	1024	768	8	256	60, 70, 75, 85
VESA	106	6C	1280	1024	4	16	60, 75, 85
VESA	107	62	1280	1024	8	256	60, 75, 85
VESA	109	55	132	25	132X25	16	70
VESA	10A	54	132	43	132X43	16	70
VESA	10E	61	320	200	16	65536	60, 85

Table 5-1. Supported BIOS Modes (continued)

Type	VESA Mode No.	SM3110 Mode No.	Width	Height	Text / bpp	Colors	Refresh Rates
VESA	110	66	640	480	15	32768	60, 72, 75, 85
VESA	111	64	640	480	16	65536	60, 72, 75, 85
VESA	112	71	640	480	24	16.8M	60, 72, 75, 85
VESA	113	67	800	600	15	32768	56, 60, 72, 75, 85
VESA	114	65	800	600	16	65536	56, 60, 72, 75, 85
VESA	115	68	800	600	24	16.8M	56, 60, 72, 75, 85
VESA	116	72	1024	768	15	32768	60, 70, 75, 85
VESA	117	73	1024	768	16	65536	60, 70, 75, 85
VESA	118	74	1024	768	24	16.8M	60, 70, 75, 85
VESA	119	75	1280	1024	15	32768	60, 75, 85
VESA	11A	76	1280	1024	16	65536	60, 75, 85
VESA	11B	77	1280	1024	24	16.8M	60, 75, 85
VESA	11C	63	640	400	16	65536	60, 85
VESA	11D	78	640	480	32	16.8M	60, 72, 75, 85
VESA	11E	79	800	600	32	16.8M	56, 60, 72, 75, 85
VESA	11F	7A	1024	768	32	16.8M	60, 70, 75, 85
VESA	120	7B	1600	1200	8	256	60, 65, 70, 75, 85
SM3110		50	320	200	8	256	60, 85
SM3110		51	320	240	8	256	60, 72, 75, 85
SM3110		52	320	240	16	65536	60, 72, 75, 85
SM3110		53	400	300	8	256	56, 60, 72, 75, 85
SM3110		56	400	300	16	65536	56, 60, 72, 75, 85
SM3110		57	512	384	8	256	60, 70, 75, 85
SM3110		59	512	384	16	65536	60, 70, 75, 85
SM3110		5A	720	480	8	256	60, 72, 75, 85
SM3110		5B	720	480	16	65536	60, 72, 75, 85
SM3110		7C	1600	1200	16	65536	60, 65, 70, 75, 85

5.1.1 SM3110 Specific Extended BIOS Functions

The SM3110 BIOS also provides some BIOS functions specifically designed for the SM3110 device, including some special functions which support Dual View. Dual View is the capability to display totally independent contents on LCD and CRT monitor. All of these extended BIOS functions use INT-10, with a function call value of 5Fh. They are listed in Table 5-2.

Table 5-2. SM3110 Specific Extended BIOS Functions

BIOS	Functions
SM3110	INT-10 (5F, 00, Query BIOS Version)
SM3110	INT-10 (5F, 01, Query Super VGA Mode Support)
SM3110	INT-10 (5F, 02, Set Super VGA Mode for Second View)
SM3110	INT-10 (5F, 03, Enable Dual View Mode)
SM3110	INT-10 (5F, 04, Disable Dual View Mode)

5.1.2 Standard VGA BIOS Functions

The SM3110 BIOS implements the standard VGA functions listed in Table 5-3. All of these use INT-10h, with function call values 00h to 1Ch.

Table 5-3. Standard VGA BIOS Functions

BIOS	Function
VGA BIOS	INT-10h (00, Set Video Mode)
VGA BIOS	INT-10h (01, Set Cursor Type)
VGA BIOS	INT-10h (02, Set Cursor Position)
VGA BIOS	INT-10h (03, Get Cursor Position)
VGA BIOS	INT-10h (04 N/A)
VGA BIOS	INT-10h (05, Set Active Display Page)
VGA BIOS	INT-10h (06, Windows Scroll Up)
VGA BIOS	INT-10h (07, Windows Scroll Down)
VGA BIOS	INT-10h (08, Read Character/Attributes at Cursor)
VGA BIOS	INT-10h (09, Write Character/Attributes at Cursor)
VGA BIOS	INT-10h (0A, Write Character at Cursor)
VGA BIOS	INT-10h (0B,0, Set Background, Border Color)
VGA BIOS	INT-10h (0B,1, Select Palette Set)
VGA BIOS	INT-10h (0C, Write Dot (pixel))
VGA BIOS	INT-10h (0D, Read Dot (pixel))
VGA BIOS	INT-10h (0E, Write TTY Character to Active Page)
VGA BIOS	INT-10h (0F, Get Video State)
VGA BIOS	INT-10h (10, 00, Set Palette Register)
VGA BIOS	INT-10h (10, 01, Set Overscan (border) Register)
VGA BIOS	INT-10h (10, 02, Set All Palette Registers and Overscan Register)
VGA BIOS	INT-10h (10, 03, Intensity/Blinking)
VGA BIOS	INT-10h (10, 04...6, Reserved)
VGA BIOS	INT-10h (10, 07, Read Individual Palette Register)
VGA BIOS	INT-10h (10, 08, Read Overscan (border) Register)
VGA BIOS	INT-10h (10, 09, Read All Palette Registers and Overscan Register)
VGA BIOS	INT-10h (10, 0A...F, Reserved)
VGA BIOS	INT-10h (10, 10, Set Individual Color Register)
VGA BIOS	INT-10h (10, 11, Reserved)
VGA BIOS	INT-10h (10, 12, Set Block of Color Registers)
VGA BIOS	INT-10h (10, 13, Select Color Page)
VGA BIOS	INT-10h (10, 14, Reserved)
VGA BIOS	INT-10h (10, 15, Read Individual Color Register)
VGA BIOS	INT-10h (10, 16, Reserved)
VGA BIOS	INT-10h (10, 17, Read Block of Color Registers)

Table 5-3. Standard VGA BIOS Functions (continued)

BIOS	Function
VGA BIOS	INT-10h (10, 18, Set DAC Mask)
VGA BIOS	INT-10h (10, 19, Get DAC Mask)
VGA BIOS	INT-10h (10, 1A, Read Current State of Color Page)
VGA BIOS	INT-10h (10, 1B, Sum Color Values to Gray Shades)
VGA BIOS	INT-10h (11, 0, Load User Text Font)
VGA BIOS	INT-10h (11, 01, Load 8x14 ROM Text Font)
VGA BIOS	INT-10h (11, 02, Load 8x8 ROM Text Font)
VGA BIOS	INT-10h (11, 03, Select Block Specifier)
VGA BIOS	INT-10h (11, 04, Load 8x16 ROM Text Font)
VGA BIOS	INT-10h (11, 10, Load User Text Font and Reprogram Controller)
VGA BIOS	INT-10h (11, 11, Load 8x14 ROM Text Font and Reprogram Controller)
VGA BIOS	INT-10h (11, 12, Load 8x8 ROM Text Font and Reprogram Controller)
VGA BIOS	INT-10h (11, 14, Load 8x16 ROM Text Font and Reprogram Controller)
VGA BIOS	INT-10h (11, 20, Set Pointer of User Graphics Font Table to INT-1F)
VGA BIOS	INT-10h (11, 21, Set Pointer of User Graphics Font Table to INT-43)
VGA BIOS	INT-10h (11, 22, Set Pointer of 8x14 ROM Graphics Font Table to INT-43)
VGA BIOS	INT-10h (11, 23, Set Pointer of 8x8 ROM Graphics Font Table to INT-43)
VGA BIOS	INT-10h (11, 26, Set Pointer of 8x16 ROM Graphics Font Table to INT-43)
VGA BIOS	INT-10h (11, 30, Get Font Information)
VGA BIOS	INT-10h (12, 10, Get Configuration Information)
VGA BIOS	INT-10h (12, 20, Select Alternate Print-screen Routine)
VGA BIOS	INT-10h (12, 30, Select Scanlines (Alphanumeric Mode))
VGA BIOS	INT-10h (12, 31, Enable/Disable Default Palette Loading)
VGA BIOS	INT-10h (12, 32, Enable/Disable Video)
VGA BIOS	INT-10h (12, 33, Enable/Disable Greyscale Summing)
VGA BIOS	INT-10h (12, 34, Enable/Disable Cursor Emulation)
VGA BIOS	INT-10h (12, 35, Switch Active Display)
VGA BIOS	INT-10h (12, 36, Enable/Disable Screen Refresh)
VGA BIOS	INT-10h (13, Write String in TTY)
VGA BIOS	INT-10h (1A, Get/Set Display Combination Code)
VGA BIOS	INT-10h (1B, Get Functionality/State Information)
VGA BIOS	INT-10h (1C, Save/Restore Video State)
VGA BIOS	INT-10h (1D...FF, Reserved)

5.1.3 VESA BIOS Extended Functions

The VESA Video BIOS Extensions (VBE version 2.0), Power Management (PM version 1.0), DDC2B (version 1.0), and Flat Panel (FP version 1.0) functions listed in Table 5-4 are implemented in the SM3110 BIOS. All of these use INT-10, with a function call value of 4Fh.

Table 5-4. VESA BIOS Extended Functions

BIOS	Function
VBE	INT-10 (4F, 00, Return Super VGA Information)
VBE	INT-10 (4F, 01, Return Super VGA Mode Information)
VBE	INT-10 (4F, 02, Set Super VGA Mode)
VBE	INT-10 (4F, 03, Return Current Video Mode)
VBE	INT-10 (4F, 04, Save/Restore Super VGA Video State)
VBE	INT-10 (4F, 05, CPU Video Memory Window Control)
VBE	INT-10 (4F, 06, Set/Get Logical Scanline Length)
VBE	INT-10 (4F, 07, Set/Get Display Start)
VBE	INT-10 (4F, 08, Set/Get DAC Palette Control)
VBE	INT-10 (4F, 09, Set/Get Palette Data) (VBE 2.0)
VBE	INT-10 (4F, 0A, Return VBE Protected Mode Interface) (VBE 2.0)
PM	INT-10 (4F, 10, 00, Report VBE/PM Capabilities)
PM	INT-10 (4F, 10, 01, Set Display Power State)
PM	INT-10 (4F, 10, 02, Get Display Power State)
DDC	INT-10 (4F, 15, 00, Report VBE/DDC Capabilities)
DDC	INT-10 (4F, 15, 01, Read EDID)
FP	INT-10 (4F, 11, 00, Return Flat Panel Extensions Support Information)
FP	INT-10 (4F, 11, 01, Return Flat Panel Information)
FP	INT-10 (4F, 11, 03, Return/Select Flat Panel Shading Options)
FP	INT-10 (4F, 11, 06, Return/Select Vertical and Horizontal Positioning)
FP	INT-10 (4F, 11, 07, Return/Select Vertical and Horizontal Expansion)

5.1.4 SCRATCHPAD Register Usage

There are eight scratchpad registers which can be accessed through the GRC index/data port of 3CE/FH (see Section 4.3.7, Enhanced VGA Registers). These are defined for the SM3110 as shown here.

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
88	07:00	R/W	Refresh rate control
	<i>Bits</i>	<i>Field</i>	
	07:04	Refresh rate for 640 x 400.	
	03:00	Refresh rate for 320 x 200.	
		<i>Value</i>	<i>Semantics</i>
		0H	<i>Reserved.</i>
		1H	60 Hz.
		2H	85 Hz.
		3H-FH	<i>Reserved.</i>

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
89	07:00	R/W	Refresh rate control
	<i>Bits</i>	<i>Field</i>	
	07:04	Refresh rate for 640 x 480	
	03:00	Refresh rate for 320 x 240.	
		<i>Value</i>	<i>Semantics</i>
		0H	<i>Reserved.</i>
		1H	60 Hz.
		2H	72 Hz.
		3H	75 Hz.
		4H	85 Hz.
		5H-FH	<i>Reserved.</i>

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
8A	07:00	R/W	Refresh rate control
	<i>Bits</i>	<i>Field</i>	
	07:04	Refresh rate for 800 x 600.	
	03:00	Refresh rate for 400 x 300.	
		<i>Value</i>	<i>Semantics</i>
		0H	<i>Reserved.</i>
		1H	56 Hz.
		2H	60 Hz.
		3H	72 Hz.
		4H	75 Hz.
		5H	85 Hz.
		6H-FH	<i>Reserved.</i>

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
8B	07:00	R/W	Refresh rate control
	<i>Bits</i>	<i>Field</i>	
	07:04	Refresh rate for 1024 x 768	
	03:00	Refresh rate for 512 x 384.	
		<i>Value</i>	<i>Semantics</i>
		0H	<i>Reserved.</i>
		1H	60 Hz.
		2H	70 Hz.
		3H	75 Hz.
		4H	85 Hz.
		5H-FH	<i>Reserved.</i>

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
8C	07:00	R/W	Refresh rate control

<i>Bits</i>	<i>Field</i>
07:04	Refresh rate for 1600 x 1200.

<i>Value</i>	<i>Semantics</i>
0H	<i>Reserved.</i>
1H	60 Hz.
2H	65 Hz.
3H	70 Hz.
4H	75 Hz.
5H	85 Hz.
6H-FH	<i>Reserved.</i>

03:00	Refresh rate for 1280 x 1024.
-------	-------------------------------

<i>Value</i>	<i>Semantics</i>
0H	<i>Reserved.</i>
1H	60 Hz.
2H	75 Hz.
3H	85 Hz.
4H-FH	<i>Reserved.</i>

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
8B	07:00	R/W	Refresh rate control

<i>Bits</i>	<i>Field</i>
07:04	<i>Reserved.</i>

03:00	Refresh rate for 720 x 480
-------	----------------------------

<i>Value</i>	<i>Semantics</i>
0H	<i>Reserved.</i>
1H	60 Hz.
2H	72 Hz.
3H	75 Hz.
4H	85 Hz.
5H-FH	<i>Reserved.</i>

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
8E	07:00	R/W	Current Video Mode Number

<i>Bits</i>	<i>Field</i>
07:00	Current video mode number.

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
8F	07:00	R/W	Video Memory Size

<i>Bits</i>	<i>Field</i>
07:00	Total video memory size available, in multiples of 64K units.

5.2 VGA BIOS Functions

The SM3110 VGA BIOS is a high-performance firmware product optimized to take full advantage of the SM3110 VGA controllers. The SM3110 BIOS is based on proven BIOS technology, and is fully compatible with the IBM VGA BIOS INT 10h interface and the VESA/VBE 2.0 interface. The BIOS is designed to provide a well-defined interface between MS-DOS, application software, and special OEM utility programs. In addition, it provides an extended set of functions to support the SM3110 VGA controller, including high-resolution extended video mode support, direct-color operation, high-performance adapter or system board implementation and system BIOS integration and customization.

Extended Video Mode Support

The SM3110 VGA BIOS provides full support for all extended high-resolution video modes by INT 10h function calls. In addition, the SM3110 BIOS supports a variety of extended functions, such as extended VGA mode inquiry and mode switching for dual view.

Direct-Color Operation

The SM3110 BIOS supports Direct-color and True-color video modes. These modes allow the SM3110 to display 32,768 colors, 65,636 colors, or 16.8 million colors at resolutions of up to 1280x1024.

High Performance

The SM3110 BIOS is optimized to provide maximum performance in adapter or motherboard implementations. The SM3110 local bus, display memory interface, memory clock, and dot clock configurations are configurable using the VGA BIOS.

In addition, time-critical routines such as TTY output and scroll are designed to provide maximum throughput in both text and graphics modes.

Customization

The default SM3110 BIOS is designed for use, without modification, in almost all environments. However, the SM3110 BIOS can be easily customized for a specific system environment with the SM3110 BIOS Utility program. Such modifications do not require the SM3110 BIOS source code. Many of the BIOS parameters and features including the sign-on message or logo, video mode parameter tables, font tables and LCD panel parameter tables can be modified.

Compatibility

The SM3110 BIOS is fully compatible with the IBM VGA BIOS and supports BIOS-level compatibility for an adapter card or integrated VGA on the system board. In addition, the SM3110 BIOS complies fully with the video modes and specifications issued by the Video Electronics Standards Association (VESA).

5.2.1 SM3110 Extended BIOS Functions (5Fh)

The SM3110 BIOS provides a set of function calls to control operation of the extended features of the SM3110 Flat Panel/CRT VGA Controller. These function calls are implemented as sub-functions under the extended VGA control function (5Fh).

5.2.1.1 **Subfunction: 00h - Query BIOS Version**

This function returns the BIOS version number.

Input:	AH	= 5Fh	
	AL	= 00h	Query BIOS version
Output:	AX	= 005Fh	Success
		= 015Fh	Function not supported
	BH	=	Version, major
	BL	=	Version, minor

All other registers are preserved.

5.2.1.2 **Subfunction: 01h - Query VBE Mode Support**

This function returns whether the BIOS supports a specific VBE display mode.

Input:	AH	= 5Fh	
	AL	= 01h	Query VBE mode support
	CX	=	Mode number
Output:	AX	= 005Fh	Mode supported
		= 015Fh	Mode not supported
	BX	=	Mode number
	CX	=	Maximum Refresh Rate if success, else it is FFh

5.2.1.3 **Subfunction: 02h - Set VBE Mode for Second View**

This function sets the desired video mode to the second display channel.

Input:	AH	= 5Fh	
	AL	= 02h	Set VBE mode support
	BX	=	Desired mode number
Output:	CX	=	starting address
	AX	= 005Fh	Mode set successfully
		= Other	Fail

All other registers are preserved.

NOTES:

- 1) This function can only be called to set to VESA enhanced modes or SM3110 modes. Calling this function to set to standard VGA modes will fail.
- 2) This function will not clear video memory if bit 15 of BX is set.

5.2.1.4 **Subfunction: 03h - Enable Second View Mode**

This function enables Second Display and switch to Dual View Mode

Input:	AH	= 5Fh	
	AL	= 03h	Enable Dual View Mode
Output:	AX	= 005Fh	Success
		= 015Fh	Fail

5.2.1.5 **Subfunction: 04h - Disable Second View Mode**

This function will turn off the second display and switch to simultaneous view.

This function disables Dual View Mode.

Input	AH	= 5Fh	
	AL	= 04h	Disable Dual View Mode
Output:	AX	= 005Fh	Success
		= 015Fh	Fail

5.2.2 Standard VGA BIOS Functions

5.2.2.1 *Function: 00h - Set Video Mode*

Input: AH = 00h
AL = Video mode (see below)

Output: None

NOTES:

- 1) Refer to the table of standard VGA video modes.
- 2) If bit 7 of AL is set, the display buffer is not cleared. Otherwise, the display buffer is cleared during mode setting (EGA, VGA only) (Clear Screen).
- 3) No hardware cursor in graphics modes.
- 4) Default mode during POST: mode = 3h for Color Monitor, and mode = 07h for Monochrome Monitor.
- 5) There is no difference between modes 00h and 01h, 02h and 03h, or 05h and 06h on EGA/VGA. These mode are different only on a CGA which supports composite video displays.
- 6) The default settings of each video mode can be overridden by several subfunctions in Function 12h or by the supplied user's video service table, whose address is stored in BIOS data area 0040:00A8h.
- 7) See Table 5-1, "Supported BIOS Modes," for a list of Mode numbers.

5.2.2.2 Function: 01h - Set Cursor Type

Input: AH = 01h
 CH = Start scanline of cursor (0 base)
 CL = End scanline of cursor (0 base)

Output: None

NOTES:

- 1) This function is only available in text modes. The value of cursor types are stored at [40:60].
- 2) The definition of value in register CH:

Register CH Value Definition

Bit	Definition
7:6	Reserved = 0
5	1 = No cursor display 0 = Normal blinking cursor
4:0	Start scanline (0 base)

- 3) The definition of value in register CL:

Register CL Value Definition

Bit	Definition
7	Reserved = 0
6:5	Number of character skew
4:0	End scanline (0 base)

- 4) Default setting:

Default Settings

Font Size	Start	End
8 · 8	6	7
8 · 14	11	12
8 · 16	13	14

- 5) To allow cursor displaying as the values set in the function call, turn off cursor emulation. The Cursor Emulation Flag is located in bit 0 of [40:87]. Turn it on/off with Subfunction 34h of Function 12h call.

5.2.2.3 *Function: 02h - Set Cursor Position*

Input: AH = 02h
 BH = Display page (0 base)
 DH = Row number of cursor location start (0 base)
 DL = Column number of cursor location end (0 base)

Output: None

NOTES:

- 1) This function is available for both text and graphics modes.
- 2) If register DL is specified greater than the width of screen in the display area, this causes the cursor to wrap to the next row. If register DH is specified greater than the height of screen in the display area, this causes the cursor to disappear.
- 3) Default setting for each mode: Cursor Location at 0000h.
- 4) BIOS maintains one cursor location for each page and supports up to eight pages. These values are recorded at [40:50] and occupy eight words (one word for each location).

5.2.2.4 *Function: 03h - Get Cursor Position*

Input: AH = 03h
 BH = Display page (0 base)

Output: CH = Start scanline of cursor (0 base)
 CL = End scanline of cursor (0 base)
 DH = Row number of cursor start location (0 base)
 DL = Column number of cursor end location (0 base)

NOTES:

- 1) Cursor type is same for all pages. The cursor location of each page is maintained separately.

5.2.2.5 **Function: 04h - Light Pen -- (Not Supported)**

5.2.2.6 **Function: 05h - Select Active Display Page**

Input: AH = 05h
 AL = Display page (0 base)
Output: None

NOTES:

- 1) The contents of each page are not altered by changing to other pages.
- 2) Please refer to the video mode table of Function 00h.

5.2.2.7 *Function: 06h - Window Scroll Up*

Input: AH = 06h
 AL = Number of rows to be scrolled up
 (0 = scroll up and clear entire window)
 BH = Attribute to be used when inserting blank lines
 CH = Y coordinate of top left corner of window (0 base)
 CL = X coordinate of top left corner of window (0 base)
 DH = Y coordinate of bottom right corner of window (0 base)
 DL = X coordinate of bottom right corner of window (0 base)
Output: None

NOTES:

- 1) When it encounters the number of rows of window equal to the value in register AL or when AL = 0, this function clears the entire window.
- 2) The image outside the window is not changed and the cursor is not updated.
- 3) Whenever an old line at the top of window is scrolled out of window, a new blank line, with attribute value specified in BH, is inserted from the bottom of window.
- 4) This function is available for both text and graphics modes.

5.2.2.8 *Function: 07h - Window Scroll Down*

Input: AH = 07h
 AL = Number of rows to be scrolled down
 (0 = scroll down and clear entire window)
 BH = Attribute to be used in inserting blank lines
 CH = Y coordinate of top left corner of window (0 base)
 CL = X coordinate of top left corner of window (0 base)
 DH = Y coordinate of bottom right corner of window (0 base)
 DL = X coordinate of bottom right corner of window (0 base)
Output: None

NOTES:

- 1) This function clears entire window when it encounters the number of rows of window equal to the value in register AL or when AL = 0.
- 2) The image outside the window is not changed and the cursor is not updated.
- 3) Whenever an old line at the bottom of window is scrolled out of window a new blank line, with attribute value specified in BH, is inserted from the top of window.
- 4) This function is available for both text and graphics modes.

5.2.2.9 **Function: 08h - Read Character/Attribute at Cursor Position**

Input: AH = 08h
BH = Display page (0 base)

Output: AH = Attribute (Valid on text modes)
AL = ASCII character code

NOTES:

- 1) This function is able to read data from other valid inactive pages in multiple page modes at any time.
- 2) After reading a character from the screen, the cursor is not updated and it must be moved manually.
- 3) No control characters such as, LF, CR, BACKSPACE, and BELL, are recognized.
- 4) In Graphics modes 04h–06h of CGA adapter, the first half of character font (Code 00h–7Fh) is only maintained in system ROM. To support the second half of character font (Code 80h–FFh), the Interrupt Vector 1Fh at 0000:007Ch, must be initialized to point to the second half of character font.
- 5) Graphics modes only return the Character Code. Three characters, 00h/20h/ FFh, cannot be distinguished, and the function always reads them back as Character Code 00h.
- 6) When they are written with the same color in the background color in graphics modes, the character codes are read back as Character Code 00h.

5.2.2.10 Function: 09h - Write Character/Attribute at Cursor Position

Input: AH = 09h
AL = ASCII character code
BH = Display page (0 base)
or
BL = Attribute (text modes)
Display color (graphics modes)
CX = Repeat character count

Output: None

NOTES:

- 1) This function is able to write data to other valid inactive pages in multiple page modes at any time.
- 2) After reading a character from the screen, the cursor is not updated and it must be moved manually.
- 3) No control characters such as, LF, CR, BACKSPACE, and BELL, are recognized.
- 4) Graphics modes 04h–06h of CGA adapter, the first half of character font (Code 00h–7Fh) is maintained in system ROM. To support the second half of character font (Code 80h–FFh), the Interrupt Vector 1Fh at 0000:007Ch, must be initialized to point to the second half of character font.
- 5) In graphics modes, the color (attribute) is treated as pixel color to generate an ASCII character pattern. The color value is masked according to the number of colors in the video modes.
- 6) When they are written with the same color in the background color in graphics modes, the character codes are displayed as blank.
- 7) The characters written to the screen, specified in CX, should not extend to the next row in graphics modes or invalid results are generated.
- 8) If bit 7 of register BL is set, the function takes the color value X'OR with the value in display memory (valid in all graphics modes except mode 13h). Use this feature in fast character/dot erasing.

5.2.2.11 **Function: 0Ah - Write Character at Cursor Position**

Input: AH = 0Ah
AL = ASCII character code
BH = Display page (0 base)
or
BL = Foreground color (graphics modes does not)
CX = Repeat character count

Output: None

NOTES:

- 1) This function is able to write data to other valid inactive pages in multiple page modes at any time.
- 2) After reading a character from the screen, the cursor is not updated and it must be moved manually.
- 3) No control characters such as, LF, CR, BACKSPACE, and BELL, are recognized.
- 4) In Graphics modes 04h–06h of CGA adapter, the first half of character font (Code 00h–7Fh) is only maintained in system ROM. To support the second half of character font (Code 80h–FFh), the Interrupt Vector 1Fh at 0000:007Ch, must be initialized to point to the second half of character font.
- 5) In graphics modes, the color (attribute) is treated as pixel color to generate an ASCII character pattern. The color value is masked according to the number of colors in the video modes.
- 6) When they are written with the color same as background color in graphics modes, the character codes are displayed as blank.
- 7) The characters written to screen, specified in CX, should not extend to the next row in graphics modes or invalid results are generated.
- 8) If bit 7 of register BL is set, the function will take the color value XOR'ed with the value in display memory (valid in all graphics modes except mode 13h). This feature is applicable in fast character/dot erasing.

5.2.2.13 **Function: 0Ch - Write Dot (Pixel)**

Input: AH = 0Ch
AL = Color value for pixel (bit 7 is XOR'ed flag)
BH = Display page (0 base)
CX = X coordinate, column number (0 base)
DX = Y coordinate, row number (0 base)

Output: None

NOTES:

- 1) For coordinate range, refer to the resolution field of video mode for Function 00h.
- 2) If bit 7 of AL register is set, it causes the requesting color value X'ORed with memory color value.

5.2.2.14 **Function: 0Dh - Read Dot (Pixel)**

Input: AH = 0Dh
BH = Display page (0 base)
CX = X coordinate, column number (0 base)
DX = Y coordinate, row number (0 base)

Output: AL = Dot (Pixel) color

NOTES:

- 1) For coordinate range, please refer to the resolution field of video mode for Function 00h.

5.2.2.15 *Function: 0Eh - Write Character to Active RAM in TTY Mode*

Input: AH = 0Eh
 AL = ASCII character
 BL = Foreground color in graphics modes
Output: None

NOTES:

- 1) Control characters such as LF, CR, Backspace, and BELL are recognized. (ASCII Codes: LF = 0AH, CR = 0DH, Backspace = 08H, and BELL = 07H).
- 2) Line wrapping and screen scrolling are supported.
- 3) After writing a character to the screen, the cursor is moved to the next position.
- 4) If PC BIOS Version is 10/19/81 or earlier; set register BH to a '0'.
- 5) If bit 7 of the register is set in graphics modes. the color value in register BL is XOR'ed with the content of display memory.
- 6) In text modes, the attribute of a character written to a new line is taken from the attribute of the last character in previous line. To control the attribute for a character, use function 09h with blank character/attribute before the function issued.

5.2.2.16 *Function: 0Fh - Get Video State*

Input: AH = 0Fh
Output: AH = Number of displayable columns (1 base)
 AL = Current video mode
 BH = Current active page (0 base)

5.2.2.17 Function: 10h - Palette Register (Subfunctions)

Subfunction: 00h—Set Individual Palette Register (Internal Palette Register)

Input: AH = 10h
 AL = 00h (Subfunction)
 BH = Color value
 BL = Palette register (0–0Fh)
Output: None

NOTES:

- 1) Color value in the Internal Palette register serves as a pointer that points to one of the external registers (RAMDAC).
- 2) In mode 13h, the color would not be changed by this function

Subfunction: 01h — Set Overscan (Border) Register

Input: AH = 10h
 AL = 01h (Subfunction)
 BH = Color value (00h–FFh)
Output: None

NOTES:

- 1) Border color is driven by one of 256 external registers.

Subfunction: 02h — Set All Palette Registers and OverScan Register

Input: AH = 10h
 AL = 02h (Subfunction)
 ES: DX = Points to a 17-byte buffer
Output: None

NOTES:

- 1) The first 16 bytes in the buffer store the values for 16 Internal Palette registers. The last byte is the value for Overscan register.
- 2) The display color is not affected, except for the Overscan register in mode 13h.

Subfunction: 03h — Toggle Intensify/Blinking Bit

Input: AH = 10h
 AL = 03h (Subfunction)
 BL = 00h–Intensify
 01h–Blinking

Output: None

NOTES:

- 1) Bit 7 of Attribute Byte is interpreted according to the setting state by this function. This function can provide 16 background colors (in intensify state) of 16-color text modes.
- 2) This function also supports Monochrome modes (07h, 0Fh).

Subfunction: 04h - 06h — Reserved

Subfunction: 07h — Read Individual Palette Register (Internal Palette Register)

Input: AH = 10h
 AL = 07h (Subfunction)
 BL = Palette register (0–0Fh)
Output: BH = Color value

NOTES:

- 1) Color Value in Internal Palette register is served as a pointer that points to one of the external registers (RAMDAC).

Subfunction: 08h — Read Overscan (Border) Register

Input: AH = 10h
 AL = 08h (Subfunction)
Output: BH = Color value

NOTES:

- 1) Border color is from 00h to FFh.

Subfunction: 09h — Read All Palette Registers and OverScan Register

Input: AH = 10h
 AL = 09h (Subfunction)
 ES: DX = Points to a 17-byte buffer
 (The first 16 bytes for returning values from 16 palette registers
 respectively and the last byte for Overscan register.)
Output: ES: DX = Points to the same buffer that was provided to the function call.

Subfunction: 0A-0Fh — Reserved

Subfunction: 10h — Set Individual Color Register (RAMDAC/External Palette Registers)

Input: AH = 10h
 AL = 10h (Subfunction)
 BX = Color register (00h–FFh)
 DH = Red color
 CH = Green color
 CL = Blue color
Output: None

NOTES:

- 1) Each color only has 6 significant bits. Three colors (RED, GREEN, and BLUE) are formed into a 18-bit datum stored in the Color register.
- 2) The maximum number of displayable colors is 256 out of 262,000 colors.
- 3) In standard VGA, mode 13h uses all 256 Color registers to display colors.
- 4) Whenever Function 00h (Set Video mode) is called, the BIOS loads default values into Color registers. This is only true when the default palette loading disable flag is not set (refer to Subfunction 31h of Function 12h).
- 5) With the gray-summing flag set, the value returned for all three colors is the grayshade value.

Subfunction: 11h — Reserved

Subfunction: 12h — Set Block of Color Registers

Input: AH = 10h
 AL = 12h (Subfunction)
 BX = Start Color register (00h–FFh)
 CX = Number of color registers to set
 ES: DX = Points to table of color values
 (each table entry is in RGB format)

Output: None

NOTES:

- 1) Each color only has 6 significant bits. Three colors; RED, GREEN, BLUE, are formed into a 18-bit datum stored in the Color register.
- 2) The number of maximum displayable colors is 256 of 262,144 (2^{18}) possible colors.
- 3) In standard VGA, mode 13h uses all 256 Color registers to display colors.
- 4) Whenever Function 00h (Set Video mode) is called, BIOS loads default values into Color registers. This is only true when the disable flag, of default palette loading, is not set (refer to Subfunction 31h of Function 12h).
- 5) With the gray-summing flag set, the value returned for all three colors is the grayshade value.

Subfunction: 13h — Select Color Page (Not valid on Mode 13h)

Input: AH = 10h
 AL = 13h (Subfunction)
 BL = 00h (select paging mode)
 01h (select color page)
 When BL = 00h -
 BH = 00h (select 4 pages of 64-color register page)
 01h (select 16 pages of 16-color register page)
 When BL = 01h -
 BH = Color page number (0 base)

Output: None

NOTES:

- 1) Except for 256-color modes, all video modes are supported by the function.
- 2) This function treats 256-color registers as sets of 16- or 64-color registers. It can be used to display different colors quickly by switching among color sets (pages).
- 3) After Video mode is set, Default setting is Page 0 of 64-color Page mode. Normally, Function 00h (set Video mode) loads the default colors of the first 64-color registers (Page 00h). These are loaded for all standard VGA modes, except mode 13h (248 registers loading).

Subfunction: 14h — Reserved

Subfunction: 15h — Read Individual Color Register (RAMDAC/External Palette Registers)

Input:	AH =	10h
	AL =	15h (Subfunction)
	BX =	Color register (00h–FFh)
Output:	DH =	Red color
	CH =	Green color
	CL =	Blue color

NOTES:

- 1) The maximum number of displayable colors is 256 out of 262,144 colors.
- 2) In standard VGA, mode 13h uses all 256-color registers to display colors.
- 3) With the gray-summing flag set, the value returned for all three color elements of Color register is the grayshade value.

Subfunction: 16h — Reserved

Subfunction: 17h — Read Block of Color Registers

Input:	AH =	10h
	AL =	17h (Subfunction)
	BX =	Start Color register (00h–FFh)
	CX =	Number of Color registers to read
	ES:DX =	Point to user provided buffer for returned color values
Output:	ES:DX =	Points to same buffer from function call entry (buffer is treated as a color table and each entry of the table consists of three bytes in RGB format).

NOTES:

- 1) Each color is a 6-bit value. All three colors form a 18-bit datum.
- 2) The maximum colors displayable are 256 out of 262,144 colors.
- 3) In standard VGA, mode 13h uses all 256-color registers to display colors.
- 4) Whenever Function 00h (set Video mode) is called, BIOS loads default values into Color registers. This is only true when the disable flag of default palette loading is not set (please refer to Subfunction 31h of Function 12h).
- 5) With the gray-summing flag set, the value returned for all three colors is the grayshade value.

Subfunction: 18h — Set DAC Mask

Input: AH = 10h
 AL = 18h (Subfunction)
 BL = DAC mask
Output: None

Subfunction: 19h — Get DAC Mask

Input: AH = 10h
 AL = 19h (Subfunction)
Output: BX= DAC mask

**Subfunction: 1Ah — Read Current State of Color Page
(Not valid on Mode 13h)**

Input: AH = 10h
 AL = 1Ah (Subfunction)
Output: BH = Current page
 (Value depends on paging mode. 00h is the default.)
 BL = Current Paging mode
 (00h = 4 pages of 64-Color registers (default),
 01h = 16 pages of 16-color registers)

NOTES:

- 1) All video modes except 256-color modes are supported by the function.
- 2) This function treats 256-color registers as sets of 16- or 64-color registers. It can be used to display different colors quickly by switching among color sets (pages).
- 3) After Video mode set, default setting is page 00h of 6-color page mode.

Subfunction: 1Bh — Sum Color Values to Grayshades

Input: AH = 10h
 AL = 1Bh (Subfunction)
 BX = Start Color register (00h–FFh)
 CX = Number of Color registers to sum
Output: None

NOTES:

- 1) This function combines the required color registers into grayshade values regardless of the gray-summing flag.

5.2.2.18 Function: 11h - Character Generation (Subfunctions)**Subfunction: 00h — Load User Text Font**

Input: AH = 11h
 AL = 00h (Subfunction)
 BH = Number of bytes per character
 BL = Block to load (00h–07h)
 CX = Number of characters to store
 DX = ASCII character ID of the first character in the font table
 (ES: BP)
 ES:BP = Points to the user-provided font table

Output: None

NOTES:

- 1) This function is only available for text modes. The value in register BH represents the height of each character and can be specified a maximum of 32-bytes-per-character in standard VGA specification.
- 2) In VGA, the character font is loaded into RAM Map 2 (0 base), which can contain up to eight fonts at any time, and is only available for the Character Generator. Consequently, the block value specified in register BL ranges from 00h (default) to 07h.
- 3) Out of eight character fonts, two fonts can be used at any time. This provides 512 simultaneously displayable characters instead of 256 characters. Two fonts may be displayed this way:
 - a) Load the fonts into desired blocks.
 - b) Out of eight font blocks, Subfunction 03h (Select Block Specifier) of Function 11h is called to select two different font blocks; one for a primary font and the other for a secondary font.

Bit 3 of Attribute Byte serves as the Font Block Selector and foreground intensity for the character.

Bit 3 = 0 — Primary font selected and normal display (eight foreground colors) if Subfunction 00h of Function 10h is called with BX = 0712h.

Bit 3 = 1 — Secondary font selected and normal display (eight foreground colors) else Bit 3 = 1 — Secondary font selected and intensity display (16 foreground colors)
- 4) Default setting by BIOS loads a font into Block 0, which is used for both primary font and secondary font (256 displayable characters).
- 5) Since the controller is not reprogrammed, abnormal character display may occur. Consequently, the font loading by Subfunction 00h requires caution. For example, if a font table is loaded to override the font in a block, a double image can result (especially when the loaded font size is smaller than the one previously displayed).

Subfunction: 01h — Load 8 x 14 ROM Font

Input: AH = 11h
 AL = 01h (Subfunction)
 BL = Block to load (00h–07h)

Output: None

NOTES:

- 1) This function is only available for text modes.
- 2) This subfunction actually loads an 8 x 16 font.
- 3) The height of character is 14 bytes but the height of display cell is same as default setting.
- 4) In VGA, the character font is loaded into RAM Map 2 (0 base), which can contain up to eight fonts at any time and is only available for the Character Generator. Consequently, the block value specified in register BL ranges from 00h (default) to 07h.
- 5) Two out of eight character fonts can be used at any time. Instead of 256 characters, this provides 512 simultaneously displayable characters. Two fonts may be displayed this way:
 - a) Load fonts into desired blocks.
 - b) Out of eight font blocks, subfunction 03h (Select Block Specifier) of Function 11h is called to select two different font blocks; one for the primary font and one for the secondary font.
 - c) Bit 3 of Attribute Byte serves as the Font Block Selector and foreground intensity for the character.
Bit 3 = 0 — Primary font selected and normal display (eight foreground colors) if Subfunction 00h of Function 10h is called with BX = 0712h.
Bit 3 = 1 — Secondary font selected and normal display (eight foreground colors)
else
Bit 3 = 1 — Secondary font selected and intensity display (16 foreground colors)
- 6) Default setting by BIOS loads a font into Block 0, which is used for both the primary font and the secondary font (256 displayable character).
- 7) Since the controller is not reprogrammed and abnormal character display can occur, the font loading by Subfunction 00h requires caution. For example, if a font table is loaded to override the font in a block, a double image can result (especially if the loaded font size is smaller than the one previously displayed).

Subfunction: 02h — Load 8 x 8 ROM Font

Input: AH = 11h
 AL = 08h (Subfunction)
 BL = Block to load (00h–07h)
Output: None

NOTES:

- 1) This function is only available for text modes.
- 2) The height of the character is eight bytes, but the height of the display cell is the same as the default setting.
- 3) In VGA, the character font is loaded into RAM Map 2 (0 base), which can contain up to eight fonts at any time and is only available for the Character Generator. Consequently, the block value specified in register BL ranges from 00h (default) to 07h.
- 4) Two out of eight character fonts can be used at any time. Instead of 256 characters, this provides 512 simultaneously displayable characters. Two fonts may be displayed this way:
 - a) Load fonts into desired blocks.
 - b) Subfunction 03h (Select Block Specifier) of Function 11h is called to select two different font blocks out of eight font blocks; one for primary font, the other for secondary font.
 - c) Bit 3 of Attribute Byte serves as the Font Block Selector and foreground intensity for the character.
Bit 3 = 0 — Primary font selected and normal display (eight foreground colors) if Subfunction 00h of Function 10h is called with BX = 0712h.
Bit 3 = 1 — Secondary font selected and normal display (eight foreground colors)
else Bit 3 = 1 — Secondary font selected and intensity display (16 foreground colors)
- 5) Default setting by BIOS loads a font into Block 0, which is used for both primary font and secondary font (256 displayable characters).
- 6) Since the controller is not reprogrammed and abnormal character display can occur, the font loading by Subfunction 00h requires caution. For example: if a font table is loaded to override the font in a block, a double image can result (especially if the loaded font size is smaller than the one previously displayed).

Subfunction: 03h — Select Block Specifier

Input: AH = 11h
 AL = 03h (Subfunction)
 BL = Selection of character generator blocks

Output: None

NOTES:

- 1) The definition of the value in register BL as follows:

Register BL Value

Bits	Font	Blocks
4, 1, 0	Primary Font	Block (00h–07h)
5, 3, 2	Secondary Font	Block (00h–07h)

- 2) For EGA-compatible operation, bits 0-1 are used for primary font, and bits 2-3 for secondary font.
- 3) To retain eight consistent colors during 512-character display, the Subfunction 00h of Function 10h must be called first, in the following setting:
AX = 1000h
BX = 0712h

Subfunction: 04h — Load 8 x 16 ROM Font

Input: AH = 11h
 AL = 04h (Subfunction)
 BL = Block to load (00h–07h)
Output: None

NOTES:

- 1) This function is only available for text modes.
- 2) The height of character is 16 bytes, but the height of the display cell is the same as the default setting.
- 3) In VGA, the character font is loaded into RAM Map 2 (0 base), which can contain up to eight fonts at any time and is only available for the Character Generator. Consequently, the block value specified in register BL ranges from 00h (default) to 07h.
- 4) Two out of eight character fonts can be used at any time. Instead of 256 characters, this provides 512 simultaneously displayable characters. The way to display two different fonts at one time is described in the following procedure:
 - a) Load fonts into desired blocks.
 - b) Subfunction 03h (Select Block Specifier) of Function 11h is called to select two different font blocks out of eight font blocks; one for primary font, the other for secondary font.
 - c) Bit 3 of Attribute Byte serves as the Font Block Selector and foreground intensity for the character.
Bit 3 = 0 — Primary font selected and normal display (eight foreground colors) if Subfunction 00h of Function 10h is called with BX = 0712h.
Bit 3 = 1 — Secondary font selected and normal display (eight foreground colors)
else
Bit 3 = 1 — Secondary font selected and intensity display (16 foreground colors)
- 5) Default setting by BIOS loads a font into Block 0, which is used for both primary font and secondary font (256 displayable characters).
- 6) Since the controller is not reprogrammed and abnormal character display can occur, the font loading by Subfunction 00h requires caution. For example: if a font table is loaded to override the font in a block, a double image can result (especially if the loaded font size is smaller than the one previously displayed).

Subfunction: 10h — Load User Text Font and Reprogram Controller

Input: AH = 11h
 AL = 10h (Subfunction)
 BH = Number of bytes per character
 BL = Block to load (00h–07h)
 CX = Number of characters to store
 DX = ASCII character ID of the first character in the font table
 (ES: BP)
 ES: BP = Points to the user-provided font table

Output: None

NOTES:

- 1) This function is only available for text modes.
- 2) The value in register BH represents the height of each character. It can be specified a maximum of 32 bytes per character in standard VGA specification.
- 3) In VGA, the character font is loaded into RAM Map 2 (0 base), which can contain up to eight fonts at any time, and is only available for the Character Generator. Consequently, the block value specified in register BL is ranged from 00h (default) to 07h.
- 4) Two out of eight character fonts can be used at any time. Instead of 256 characters, this provides 512 simultaneously displayable characters. The way to display two different fonts at one time is described in the following procedure:
 - a) Load fonts into desired blocks.
 - b) Subfunction 03h (Select Block Specifier) of Function 11h is called to select two different font blocks out of eight font blocks; one for primary font and the other for secondary font.
 - c) Bit 3 of Attribute Byte serves as the Font Block Selector and foreground intensity for the character.
Bit 3 = 0 — Primary font selected and normal display (eight foreground colors) if Subfunction 00h of Function 10h is called with BX = 0712h.
Bit 3 = 1 — Secondary font selected and normal display (eight foreground colors)
else
Bit 3 = 1 — Secondary font selected and intensity display (16 foreground colors)
- 5) Default setting by BIOS loads a font into Block 0, which is used for both primary font and secondary font (256 displayable characters).
- 6) Subfunction 10h is almost identical to Subfunction 00h, except for the following differences:

- a) Page 00h must be active.
- b) Character Height (bytes-per-character) is recalculated.
- c) Number of rows (0 base) are recalculated as:
(Scanlines-per-screen/Character Height) minus 1.
- d) The length of display buffer (1 base) is recalculated as
(Total number of rows x Number of columns) x 2.
- e) The CRTC registers are reprogrammed as follows:

Index	Register Name	Change
09h	Maximum scanlines	Character Height minus 1
0Ah	Cursor Start	Character Height minus 2
0Bh	Cursor End	Character Height minus 1
12h	Vertical Display Enable End:	
	For 350 or 400 scanline modes	(Rows/screen x Character Height) - 1
	For 200 scanline modes	((Rows/screen x Character Height) x 2) - 1
14h	Underline Location	Character Height minus 1 (mode 07h only)
- f) It must be called immediately after Function 00h call (Set Video mode), otherwise the result is unpredictable.

Subfunction: 11h — Load 8 x 14 ROM Font & Reprogram Controller

Input: AH = 11h
 AL = 11h (Subfunction)
 BL = Block to load (00h–08h)

Output: None

NOTES:

- 1) This function is only available for text modes.
- 2) Note that this subfunction actually loads an 8 X 16 font.
- 3) The character and display cells are both 14 bytes high (scanlines).
- 4) In VGA, the character font is loaded into RAM Map 2 (0 base), which can contain up to eight fonts at any time and is only available for the Character Generator. Consequently, the block value specified in register BL is ranged from 00h (default) to 07h.
- 5) Two out of eight character fonts can be used at any time. Instead of 256 characters, this provides 512 simultaneously displayable characters. The way to display two different fonts at one time is described in the following procedure:
 - a) Load fonts into desired blocks.
 - b) Subfunction 03h (Select Block Specifier) of Function 11h is called to select two different font blocks out of eight font blocks; one for primary font, the other for secondary font.
 - c) Bit 3 of Attribute Byte serves as the Font Block Selector and foreground intensity for the character.
 - Bit 3 = 0 — Primary font selected and normal display (eight foreground colors) if Subfunction 00h of Function 10h is called with BX = 0712h.
 - Bit 3 = 1 — Secondary font selected and normal display (eight foreground colors)
 - else
 - Bit 3 = 1 — Secondary font selected and intensity display (16 foreground colors)
- 6) Default setting by BIOS loads a font into Block 0, which is used for both primary font and secondary font (256 displayable characters).
- 7) Subfunction 11h is almost identical to Subfunction 01h, except for the following differences:
 - a) Page 00h must be active.
 - b) Character Height = 14
Number of rows (0 base) are recalculated as:
(Scanlines-per-screen/Character Height) minus 1.

d) The length of display buffer (1 base) is recalculated as
(Total number of rows X Total number of columns) x 2 .

e) The CRTC registers are reprogrammed as follows:

Index	Register Name	Change
09h	Maximum scanlines	13 (0Dh)
0Ah	Cursor Start	13 (0Dh)
0Bh	Cursor End	13 (0Dh)
12h	Vertical Display Enable End	(Rows/screen X 14) - 1
14h	Underline Location	13 (0Dh) (mode 07h only)

f) It must be called immediately after Function 00h call (set Video mode) or the result is unpredictable.

Subfunction: 12h — Load 8 x 8 ROM Font & Reprogram Controller

Input: AH = 11h
 AL = 12h (Subfunction)
 BL = Block to load (00h–07h)

Output: None

NOTES:

- 1) This function is only available for text modes.
- 2) The height of character and display cell are all eight bytes (scan-lines).
- 3) In VGA, the character font is loaded into RAM Map 2 (0 base), which can contain up to eight fonts at any time and is only available for the Character Generator. Consequently, the block value specified in register BL is ranged from 00h (default) to 07h.
- 4) Two out of eight character fonts can be used at any time. This provides 512 simultaneously displayable characters, instead of 256 characters. The way to display two different fonts at one time is described in the procedure below.
 - a) Load fonts into desired blocks.
 - b) Subfunction 03h (Select Block Specifier) of Function 11h is called to select two different font blocks out of eight font blocks; one for primary font, the other for secondary font.
 - c) Bit 3 of Attribute Byte serves as the Font Block Selector and foreground intensity for the character.

Bit 3 = 0 — Primary font selected and normal display (eight foreground colors) if Subfunction 00h of Function 10h is called with BX = 0712h.

Bit 3 = 1 — Secondary font selected and normal display (eight foreground colors)

else

Bit 3 = 1 — Secondary font selected and intensity display (16 foreground colors)
- 5) Default setting by BIOS loads a font into Block 0, which is used for both primary font and secondary font (256 displayable characters).
- 6) Subfunction 12h is almost identical to Subfunction 02h, except for the following differences:
 - a) Page 00h must be active.
 - b) Character Height = 8.
 - c) Number of rows (0 base) are recalculated as:

- (Scanlines-per-screen/Character Height) minus 1.
- d) The length of display buffer (1 base) is recalculated as (Total number of rows X Number of columns) X 2.
- e) The CRTC registers are reprogrammed as follows:

Index	Register Name	Change
09h	Maximum scanlines	7 (07h)
0Ah	Cursor Start	6 (06h)
0Bh	Cursor End	7 (07h)
12h	Vertical Display Enable End	(Rows/screen · 8) - 1
14h	Underline Location	7 (07h)

(mode 07h only)

- f) It must be called immediately after Function 00h call (set Video mode) or the result is unpredictable.

Subfunction: 14h — Load 8 x 16 ROM Font & Reprogram Controller

Input: AH = 11h
 AL = 14h (Subfunction)
 BL = Block to load (00h–07h)

Output: None

NOTES:

- 1) This function is only available for text modes.
- 2) The height of character and display cell are all 16 bytes (scanlines).
- 3) In VGA, the character font is loaded into RAM Map 2 (0 base), which can contain up to eight fonts at any time and is only available for the Character Generator. Consequently, the block value specified in register BL is ranged from 00h (default) to 07h.
- 4) Two out of eight character fonts can be used at any time. This provides 512 simultaneously displayable characters, instead of 256 characters. The way to display two different fonts at one time is described in the procedure below.
 - a) Load fonts into desired blocks.
 - b) Subfunction 03h (Select Block Specifier) of Function 11h is called to select two different font blocks out of eight font blocks; one for primary font, the other for secondary font.
 - c) Bit 3 of Attribute Byte serves as the Font Block Selector and foreground intensity for the character.

Bit 3 = 0 — Primary font selected and normal display (eight foreground colors) if Subfunction 00h of Function 10h is called with BX = 0712h.

Bit 3 = 1 — Secondary font selected and normal display (eight foreground colors)

else

Bit 3 = 1 — Secondary font selected and intensity display (16 foreground colors)
- 5) Default setting by BIOS loads a font into Block 0, which is used for both primary font and secondary font (256 displayable characters).
- 6) Subfunction 14h is almost identical to Subfunction 04h except the following differences:
 - a) Page 00h must be active.
 - b) Character Height = 16.
 - c) Number of rows (0 base) are recalculated as:
(Scanlines-per-screen/Character Height) minus 1.

- d) The length of display buffer(1 base) is recalculated as
(Total number of rows X Number of columns) X 2.
- e) The CRTC registers are reprogrammed as follows:

Index	Register Name	Change
09h	Maximum scanlines	15 (0Fh)
0Ah	Cursor Start	14 (0Eh)
0Bh	Cursor End	15 (0Fh)
12h	Vertical Display Enable End (Rows/screen-6) - 1	
14h	Underline Location	15 (0Fh) (mode 07h only)
- g) It must be called immediately after Function 00h call (set Video mode) or the result is unpredictable.

Subfunction: 20h — Set Pointer of User's Graphics Font Table to INT 1Fh

Input: AH = 11h
 AL = 20h (Subfunction)
 ES: BP = Points to the user's graphics font table

Output: None

NOTES:

- 1) The value in this Interrupt Vector serves as a pointer that points to Graphics Font for character codes (80h–FFh) in modes 04h, 05h, and 06h.
- 2) In the CGA adapter, the planar BIOS only provides 128 character codes (00h– 7Fh). The user can supply the other half of character codes (80h–FFh), or use GRAFTABL in DOS to load these character codes.
- 3) This function must be called immediately after setting Video mode.

Subfunction: 21h — Set Pointer of User's Graphics Font Table to INT 43h

Input: AH = 11h
 AL = 21h (Subfunction)
 BL = Character rows specifier
 00h = Value in register DL (the number of displayable rows
 specified by user)
 01h = 14 (0Eh) character rows
 02h = 25 (19h) character rows
 03h = 43 (2Bh) character rows
 CX = Bytes per character
 DL = Number of character rows (if register BL = 00h)
 ES: BP = Points to the user's graphics font table

Output: None

NOTES:

- 1) The value in this Interrupt Vector serves as a pointer that points to Graphics Font for character codes (00h–7fh) in modes 04h, 05h, and 06h. The vector also handles Graphics Font for character codes (00h–FFh) in all other graphics modes.
- 2) This function should only be called immediately after setting Video mode.
- 3) The portion of character rows above displayable rows are not displayed unless it is scrolled up.
- 4) The overlapping screen may occur on video modes that use all Display Memory addresses, such as mode 13h.

Subfunction: 22h — Set Pointer of ROM 8x14 Graphics Font Table to INT 43h

Input:	AH =	11h
	AL =	22h (Subfunction)
	BL =	The specifier of character rows on screen 00h = Value in register DL (the number of displayable rows specified by user) 01h = 14 (0Eh) character rows 02h = 25 (19h) character rows 03h = 43 (2Bh) character rows
	DL =	Number of character rows to display (if register BL = 00h)
	Output:	None

NOTES:

- 1) This actually sets the pointer of an 8 X 16 font table.
- 2) The value in this Interrupt Vector serves as a pointer that points to Graphics Font for character codes (00h–7fh) in modes 04h, 05h, and 06h. The vector also handles Graphics Font for character codes (00h–FFh) in all other graphics modes.
- 3) This function should only be called immediately after setting Video mode.
- 4) The portion of character rows above displayable rows are not displayed unless it is scrolled up.
- 5) The overlapping screen may occur on video modes that use all Display Memory addresses, such as mode 13h.

Subfunction: 23h — Set Pointer of ROM 8 x 8 Graphics Font Table to INT 43h

Input: AH = 11h
 AL = 23h (Subfunction)
 BL = Specifier of character rows on screen
 00h = Value in register DL (the number of displayable rows
 specified by user)
 01h = 14 (0Eh) character rows
 02h = 25 (19h) character rows
 03h = 43 (2Bh) character rows
 DL = Number of character rows to display (if register BL = 00h)

Output: None

NOTES:

- 1) The value in this Interrupt Vector serves as a pointer that points to Graphics Font for character codes (00h–7Fh) in modes 04h, 05h, and 06h. The vector also handles Graphics Font for character codes (00h–FFh) in all other graphics modes.
- 2) This function should only be called immediately after setting Video mode.
- 3) The portion of character rows above displayable rows are not displayed unless it is scrolled up.
- 4) The overlapping screen may occur on video modes that use all Display Memory addresses, such as mode 13h.

Subfunction: 24h — Set Pointer of ROM 8 x 16 Graphics Font Table to INT 43h

Input: AH = 11h
 AL = 24h (Subfunction)
 BL = Specifier of character rows on screen
 00h = Value in register DL (the number of displayable rows
 specified by user)
 01h = 14 (0Eh) character rows
 02h = 25 (19h) character rows
 03h = 43 (2Bh) character rows
 DL = Number of character rows to display (if register BL = 00h)

Output: None

NOTES:

- 1) The value in this Interrupt Vector serves as a pointer that points to Graphics Font for character codes (00h–7fh) in modes 04h, 05h, and 06h. The vector also handles Graphics Font for character codes (00h–FFh) in all other graphics modes.
- 2) This function should only be called immediately after setting Video mode.
- 3) The portion of character rows above displayable rows are not displayed unless it is scrolled up.
- 4) The overlapping screen may occur on video modes that use all Display Memory addresses, such as mode 13h.

Subfunction: 30h — Get Pointer Information of Fonts

Input: AH = 11h
 AL = 30h (Subfunction)
 BH = Pointer information of fonts
 00h–Current Font Pointer stored in Interrupt Vector 1Fh
 01h–Current Font Pointer stored in Interrupt Vector 43h
 02h–Font Pointer of ROM 8 X 16 Font Table
 03h–Font Pointer of ROM 8 X 8 Font Table
 (Character Codes 00h–7fh)
 04h–Font Pointer of ROM 8 X 8 Font Table
 (Character Codes 80h–FFh)
 05h–Font Pointer of ROM 8 X 16 Alternate Font Table
 06h–Font Pointer of ROM 8 X 16 Font Table
 07h–Font Pointer of ROM 9 X 16 Alternate Font Table

Output: CX = Current character height (bytes per character)
 DL = Number of rows of current video mode (0 base)
 ES: BP = Points to information of desired font

5.2.2.19 Function: 12h - Alternate Select (Subfunctions)

Subfunction: 10h — Get Current Video Configuration

Input: AH = 12h
 BL = 10h (Subfunction)

Output: BH = 00h—Color mode (3Dx)
 01h—Monochrome mode (3Bx)
 BL = Video Memory Size
 00h = N/A
 01h = N/A
 02h = N/A
 03h = 256 Kbytes
 CH= Feature Bits

Feature Control Output	Input Status	Bit Setting (Register 3C2h)
0	0	bit 5
1	0	bit 6
2	1	bit 5
3	1	bit 6
4–7		Reserved

CL=Switch Settings	Description
bit 0	Configuration Switch-1
bit 1	Configuration Switch-2
bit 2	Configuration Switch-3
bit 3	Configuration Switch-4
bit 4–bit 7	Reserved

NOTES:

- 1) This function is valid for backwards compatibility in EGA mode only.

Subfunction: 20h — Alternate PrintScreen Handler

Input: AH = 12h
 BL = 20h (Subfunction)

Output: None

NOTES:

- 1) This function call replaces original PrintScreen Interrupt Handler (INT 05h) to support the modes whose displayable rows on screen are over 25 rows.

Subfunction: 30h — Select Scanlines for Text Modes

Input: AH = 12h
 BL = 30h (Subfunction)
 AL = Specifier of scanlines
 00h = 200 scanlines
 01h = 350 scanlines
 02h = 400 scanlines
Output: AL = 12h (function supported)

NOTES:

- 1) The selected scanlines will take effect on next mode setting.
- 2) Mode 07h is only supported by 350/400 scanlines. Modes 00h–03h are supported by three scanlines.
- 3) The modes, 200 scanlines, are double scanned.

Subfunction: 31h — Enable/Disable Default Palette Loading

Input: AH = 12h
 BL = 31h (Subfunction)
 AL = 00h–Enable default palette loading
 01h–Disable default palette loading
Output: AL = 12h (function supported)

NOTES:

- 1) This function will take effect on next mode setting.
- 2) All Internal/External Palette registers are affected.

Subfunction: 32h — Enable/Disable Video

Input: AH = 12h
 BL = 32h (Subfunction)
 AL = 00h–Enable Video
 01h–Disable Video
Output: AL = 12h (function supported)

NOTES:

- 1) Video subsystem does not respond to any I/O or Video Memory Addressing.

Subfunction: 33h — Enable/Disable Summing-to-Grayshades

Input: AH = 12h
 BL = 33h (Subfunction)
 AL = 00h—Enable summing-to-grayshades
 01h—Disable summing-to-grayshades
Output: AL = 12h (function supported)

NOTES:

- 1) This function will take effect on a subsequent mode setting or internal/external palette setting.

Subfunction: 34h — Enable/Disable Cursor Emulation

Input: AH = 12h
 BL = 34h (Subfunction)
 AL = 00h—Enable cursor emulation
 01h—Disable cursor emulation
Output: AL = 12h (function supported)

NOTES:

- 1) This function will take effect on a subsequent mode setting or Function 01h call (set cursor type).
- 2) Bit 0 of Address [40:87] Emulation Flag is affected.

Subfunction: 35h — Switch Video Display

Input: AH = 12h
 BL = 35h (Subfunction)
 AL = 00h—Initial video adapter turned off
 (ES:DX must point to a 128-byte
 buffer for switching state save area)
 01h—System board video turned on
 02h—Active video turned off (ES:DX must
 point to a buffer for switching state save area)
 03h—Inactive video turned on (ES:DX must point
 to a buffer which saves switching state previously)
 ES:DX = Buffer for switching state (valid when AL = 00h, 02h, or 03h)

Output: AL = 12h (function supported)

NOTES:

- 1) There are several requirements that have to be met before using this function. These requirements are:
 - a) Two video subsystems coexisting: system board video and video adapter.
 - b) The two video systems have a conflict over the use of video resources.
 - c) Video adapter is primary video; system board is secondary video.
 - d) This function must be supported by system board video and video adapter.
- 2) If the first time switching from video adapter to system board video:
Call the function with register AL = 00h
Call the function with register AL = 01h
else
Call the function with register AL = 02h
Call the function with register AL = 03h

Subfunction: 36h — Enable/Disable Screen Display

Input: AH = 12h
 BL = 36h (Subfunction)
 AL = 00h–Enable screen display
 01h–Disable screen display
Output: AL = 12h (function supported)

NOTES:

- 1) This function can be used for fast video memory updating without losing synchronization.

5.2.2.20 Function: 13h - Write Teletype String

Input: AH = 13h
AL = Write function specifier
00h—Write character string without updating cursor (BL = Attribute)
01h—Write character string with updating cursor (BL = Attribute)
02h—Write character/attribute string without updating cursor
03h—Write character/attribute string with updating cursor
BH = Display page (0 base)
BL = Attribute (valid when AL = 00h or 01h)
CX = String length
Dh = Start Y coordinate of string displayed on screen
DL = Start X coordinate of string displayed on screen
ES: BP = Start address of string (in Segment: Offset Format)

Output: None

NOTES:

- 1) Control Characters: LF, CR, Backspace, and BELL are recognized. (ASCII Codes: LF = 0Ah, CR = 0Dh, Backspace = 08h, BELL = 07h).
- 2) String can be written to any page regardless of active state.
- 3) Line wrapping and screen scrolling are supported. Screen scrolling is only supported on active page.
- 4) The color value in register BL will do X'OR with the content of display memory, if bit 7 of the register is set in graphics modes.

5.2.2.21 Function: 1Ah - Get/Set Display Combination Code

Subfunction: 00h — Get Display Combination Code (DCC)

Input: AH = 1Ah
AL = 00h (Subfunction)

Output: If function supported:
AL = 1Ah
BH = Alternate display code
BL = Active display code

NOTES:

- 1) Display Combination Code Definition:

Code	Definition
00h	No Display
01h	N/A
02h	N/A
03h	Reserved
04h	N/A
05h	N/A
06h	N/A
07h	Video Graphics Array with Analog Monochrome Monitor (MVGA)
08h	Video Graphics Array with Analog Color Monitor (VGA)

- 2) The user is responsible for providing the correct DCC. There is no physical device checking.

Subfunction: 01h — Set Display Combination Code (DCC)

Input: AH = 1Ah
 AL = 01h (Subfunction)
 BH = Alternate display code
 BL = Active display code
Output: If function supported:
 AL = 1Ah

NOTES:

- 1) Display Combination Code Definition:

Code	Definition
00h	No Display
01h	N/A
02h	N/A
03h	Reserved
04h	N/A
05h	N/A
06h	N/A
07h	Video Graphics Array with Analog Monochrome Monitor (MVGA)
08h	Video Graphics Array with Analog Color Monitor (VGA)

- 2) The user is responsible for providing the correct DCC. There is no physical device checking.

5.2.2.22 **Function: 1Bh - Collection of Video Information**

Input: AH = 1Bh
 BX = 00h
 ES: DI = Points to 128-byte buffer

Output: If function supported:
 AL = 1Bh

NOTES:

1) Video information in 128-byte buffer:

Offset	Size	Definition
00h	2 Words	Pointer points to collection of static functionality information
04h	Byte	Current Video mode
05h	Word	Number of columns (1 base)
07h	Word	Refresh Buffer Length (unit: byte)
09h	Word	The starting address of Refresh Buffer (Offset value relates to start of video memory default = 0000h)
0Bh	8 Words	Cursor Position for each page (maximum eight pages supported)
1Bh	Word	Current Cursor Type (High Byte = start scanline, Low Byte = end scanline)
1Dh	Byte	Active Video Page
1Eh	Word	Base Port Address of CRT Controller (CRTC) (Monochrome = 3Bxh, Color = 3Dxh)
20h	Byte	Current setting of 3B8h or 3D8h (Mode Control register)
21h	Byte	Current setting of 3B9h or 3D9h
22h	Byte	Number of rows (1 base)
23h	Word	Character Height (1 base; unit: scanline)
25h	Byte	Active Display Code
26h	Byte	Alternate Display Code
27h	Word	Number of displayable colors (1 base monochrome = 0000h)
29h	Byte	Number of Pages (1 base)
2Ah	Byte	Specifier of vertical resolution 00h = 200 scanlines 01h = 350 scanlines 02h = 400 scanlines

		03h = 480 scanlines
		04h–FFh = Reserved
2Bh	Byte	Primary Font Block (00h–07h)
2Ch	Byte	Secondary Font Block (00h–07h)
2Dh	Byte	Flags of Video State:
		Bit Definition
		7-6 Reserved
		0 = Background intensity
		1 = Blinking (Default)
		4 0 = Cursor Emulation Disable
		1 = Cursor Emulation Enable
		3 0 = Default Palette
		Loading Enable
		1 = Default Palette Loading
		Disable
		2 0 = Color Monitor Attached
		1 = Monochrome Monitor
		Attached
		1 0 = Summing-to-grayshades
		Disable
		1 = Summing-to-grayshades
		Enable
		0 1 = All modes are active
		on all displays
2E–30h		Reserved
31h	Byte	Specifier of total video RAM
		00h = N/A
		01h = N/A
		02h = N/A
		03h = 256 Kbytes
		04h–FFh = Reserved
32h	Byte	Save pointer state information:
		Bit Definition
		7-6 Reserved
		5 1 = Extension of Display
		Combination
		Code Active
		4 1 = Palette Override Active
		3 1 = Graphics Font
		Override Active
		2 1 = Alpha Font Override Active

		1	1 = Dynamic Save Area Active
		0	1 = 512-character Set Active
33–3Fh		Reserved	
2	Collection of static functionality information:		
	Offset	Size	Definition
	00h	Byte	Available video modes if bit set:
			Bit Video Mode
			0 00h
			1 01h
			2 02h
			3 03h
			4 04h
			5 05h
			6 06h
			7 07h
	01h	Byte	Available video modes if bit set:
			Bit Video Mode
			0 08h
			1 09h
			2 0Ah
			3 0Bh
			4 0Ch
			5 0Dh
			6 0Eh
			7 0Fh
	02h	Byte	Available video modes if bit set:
			Bit Video Mode
			0 10h
			1 11h
			2 12h
			3 13h
			4–7 Reserved
	03–06h	Reserved	
	07h	Byte	Number of scanlines available in text modes: (Subfunction 30h, Function 12h)
			Bit Scanlines (if bit = 1)
			0 200
			1 350
			2 400
			3–7 Reserved
	08h	Byte	Number of active character blocks available in text modes
	09h	Byte	Maximum number of character blocks available in text modes

0Ah	Byte	Supported functions (No. 1): Bit Function (if bit = 1) 0 All modes on All Displays 1 Summing to grayshades 2 Character fonts Loading 3 Default Palette Loading 4 Cursor Emulation 5 EGA Palettes (Internal Palettes) 6 Color Palettes (External Palettes/RAMDAC) 7 Color Paging
0Bh	Byte	Supported functions (No. 2): Bit Function (if bit = 1) 0 Reserved 1 Save/Restore Video State 2 Background Intensity/Blinking Control 3 Set Display Combination Code 4–7 Reserved
0C–0Dh 0Eh	Reserved	Save Pointer Functions: Bit Function (if bit = 1) 0 512-character Set 1 Dynamic Save Area 2 Alpha Font Override 3 Graphics Font Override 4 Palette Override 5 Extension of Display Combination Code 6–7 Reserved
0Fh	Reserved	

5.2.2.23 **Function: 1Ch - Video State Information**

Subfunction: 00h — Get Buffer Size for Video State

Input: AH = 1Ch
 AL = 00h (Subfunction)
 CX = Requested Video State:

Bit	Video State
0	Hardware State
1	BIOS Data Area
2	Color registers (External Palettes/RAMDAC)
3–15	Reserved

Output: If function supported:
 AL = 1Ch
 BX = Blocks/Buffer (Unit: 64 Byte/Block)

NOTES:

- 1) This function will report the required size of buffer to save video state. To guarantee Subfunction 01h and 02h are performed successfully, call this subfunction first.

Subfunction: 01h — Saving Video State

Input: AH = 1Ch
 AL = 01h (Subfunction)
 CX = Requested Video States:
 Bit **Video States**
 0 Hardware State
 1 BIOS Data Area
 2 Color registers (External Palettes/RAMDAC)
 3–15 Reserved
 ES:BX = Points to buffer (Segment: Offset format)

Output: If function supported:
 AL = 1Ch
 ES: BX = State information saved in a user-supplied buffer

Subfunction: 02h — Restore Video State

Input: AH = 1Ch
 AL = 02h (Subfunction)
 CX = Requested Video States:
 Bit **Video States**
 0 Hardware State
 1 BIOS Data Area
 2 Color registers (External Palettes/RAMDAC)
 3–15 Reserved
 ES: BX = Points to a previously saved buffer (Segment: Offset format)

Output: If function supported:
 AL = 1Ch

5.3 VESA VBE BIOS Functions

5.3.1 Overview

The VESA VBE 2.0 standard defines a set of functions used to obtain information about the capabilities of a specific Super VGA implementation. Additionally, these functions control hardware while initializing the video mode and accessing the video memory. They are intended as extensions of the VGA BIOS video services and are accessed through interrupt 10h.

For more information about the VESA standard, contact VESA at the address below.

Video Electronics Standards Association
920 Hillview Court
Suite 140, Milpitas, CA 95035
TEL: (408) 435-0333 FAX:(408) 435-8225

Silicon Magic's SM3110 VESA BIOS extensions are VESA/VBE 2.0 compliant.

5.3.2 DDC Implementation

The SM3110 BIOS supports the DDC2B display type, as specified in paragraph 2.4.2 of the VESA DDC Standard version 1.0 revision 0. Also, the SM3110 BIOS implements a Host System Type of DDC2B, which is specified in paragraph 2.5.4 of the VESA DDC Standard version 1.0 revision 0. During DDC implementation, the following functions are integrated into the SM3110 BIOS:

- The data is transmitted and received on the DDC2 bi-directional data channel based on the I²C protocol.
- The BIOS does NOT implement the ACCESS bus protocol.
- The host requests EDID information and returns it to the application.
- The BIOS supports configurations two and five in Appendix A of the VESA DDC specification.

5.3.2.1 VBE Mode Numbers

Standard VGA mode numbers are 7 bits wide and presently range from 00h to 13h. OEMs have defined extended display modes in the range 14h to 7Fh. Values from 80h to FFh cannot be used, since VGA BIOS Function 00h (Set Video mode) interprets bit 7 as a flag to clear or preserve display memory.

Due to the limitations of 7-bit mode numbers, the optional VBE mode numbers are 14 bits wide. To initialize a VBE mode, the mode number is passed in the BX register to VBE Function 02h (Set VBE mode).

The format of VBE mode numbers is as follows:

D0-D8=	Mode number	
		If D8 == 0, this is not a VESA-defined mode
		If D8 == 1, this is a VESA-defined mode
D9-D13=	Reserved by VESA for future expansion (= 0)	
D14 =	Linear/Flat Frame Buffer Select	
		If D14 == 0, Use VGA Frame Buffer
		If D14 == 1, Use Linear/Flat Frame Buffer
D15 =	Preserve Display Memory Select	
		If D15 == 0, Clear display memory
		If D15 == 1, Preserve display memory

Thus, VBE mode numbers begin at 100h. This mode numbering scheme implements standard 7-bit mode numbers for OEM-defined modes. Standard VGA modes may be initialized through VBE Function 02h (Set VBE mode) simply by placing the mode number in BL and clearing the upper byte (BH). 7-bit OEM-defined display modes may be initialized in the same way. Note that modes may only be set if the mode exists in the Video Mode List returned in Function 0.

NOTE: Starting with VBE version 2.0, VESA will no longer define new VESA mode numbers and it will no longer be mandatory to support these old mode numbers. However, it is highly recommended that BIOS implementations continue to support these mode numbers for compatibility with older software.

5.3.2.2 VBE Functions Call

This section describes in detail each of the functions defined by the VBE standard. VBE functions are called using the INT 10h interrupt vector, passing arguments in the 80X86 registers. The INT 10h interrupt handler first determines if a VBE function was requested, and if so, processes that request. Otherwise, control is passed to the standard VGA BIOS for completion.

All VBE functions are called with the AH register set to 4Fh to distinguish them from the standard VGA BIOS functions. The AL register is used to indicate which VBE function is to be performed. For supplemental or extended functionality, the BL register is used when appropriate to indicate a specific subfunction.

Functions 00h-0Fh have been reserved for Standard VBE function numbers; Functions 10h- FFh are reserved for VBE Supplemental Specifications.

5.3.2.3 VBE Return Status

The AX register is used to indicate the completion status upon return from VBE functions. If VBE support for the specified function is available, the 4Fh value passed in the AH register on entry is returned in the AL register. If the VBE function completed successfully, 00h is returned in the AH register. Otherwise, the AH register is set to indicate the nature of the failure.

VBE RETURN STATUS

AL ==	4Fh:	Function is supported
AL !=	4Fh:	Function is not supported
AH ==	00h:	Function call successful
AH ==	01h:	Function call failed
AH ==	02h:	Function is not supported in the current hardware configuration
AH ==	03h:	Function call invalid in current video mode

NOTE: Applications should treat any non-zero value in the AH register as a general failure condition as later versions of the VBE may define additional error codes.

5.3.3 VESA BIOS Extended Functions (4Fh)

The sections below list the VESA/VBE functions and their descriptions.

5.3.3.1 Subfunction: 00h - Return VBE Controller Information

This required function returns the capabilities of the display controller, the revision level of the VBE implementation, and vendor specific information to assist in supporting all display controllers in the field.

The purpose of this function is to provide information to the calling program about the general capabilities of the installed the VBE software and hardware. This function fills an information block structure at the address specified by the caller. The VbeInfoBlock information block size is 256 bytes for VBE 1.x, and 512 bytes for VBE 2.0.

Input:	AH	= 4Fh	
	AL	= 00h	Return VBE Controller Information
	ES:DI	=	Pointer to buffer in which to place VbeInfoBlock structure (VbeSignature should be set to 'VBE2' when function is called to indicate VBE 2.0 information is desired and the information block is 512 bytes in size.)
Output:	AX	=	VBE Return Status

All other registers are preserved.

The information block has the following structure:

```

VbeInfoBlock struc
VbeSignature      db    'VESA'          ; VBE Signature
VbeVersion        dw    0200h          ; VBE Version
OemStringPtr      dd    ?              ; Pointer to OEM String
Capabilities       db    4 dup (?)     ; Capabilities of graphics controller
VideoModePtr      dd    ?              ; Pointer to Video Mode List
TotalMemory       dw    ?              ; Number of 64 Kbytes memory blocks
                  ; added for VBE 2.0
OemSoftwareRev    dw    ?              ; VBE implementation Software Rev.
OemVendorNamePtr  dd    ?              ; Pointer to Vendor Name String
OemProductNamePtr dd    ?              ; Pointer to Product Name String
OemProductRevPtr  dd    ?              ; Pointer to Product Revision String
Reserved          db    222 dup (?)    ; Reserved for VBE implementation
                  ; scratch area
OemData           db    256 dup (?)    ; Data Area for OEM Strings
VbeInfoBlock ends
    
```

NOTE: All data in this structure is subject to change by the VBE implementation when VBE Function 00h is called. Therefore, it should not be used by the application to store data of any kind.

Description of the VbeInfoBlock Structure Fields

The **VbeSignature** field is filled with the ASCII characters 'VESA' by the VBE implementation. VBE 2.0 applications should preset this field with the ASCII characters 'VBE2' to indicate to the VBE implementation that the VBE 2.0 extended information is desired, and the VbeInfoBlock is 512 bytes in size. Upon return from VBE Function 00h, this field should always be set to 'VESA' by the VBE implementation.

The **VbeVersion** is a BCD value which specifies what level of the VBE standard is implemented in the software. The higher byte specifies the major version number. The lower byte specifies the minor version number.

NOTE: The BCD value for VBE 2.0 is 0200h, and the BCD value for VBE 1.2 is 0102h. In the past there have been some applications misinterpreting these BCD values. For example, BCD 0102h was interpreted as 1.02, which is incorrect.

The **OemStringPtr** is a Real Mode far pointer to a null terminated OEM-defined string. This string may be used to identify the graphics controller device or OEM product family for hardware specific display drivers. There are no restrictions on the format of the string. This pointer may point into either ROM or RAM, depending on the specific implementation. VBE 2.0 BIOS implementations must place this string in the OemData area within the VbeInfoBlock if 'VBE2' is preset in the VbeSignature field on entry to Function 00h. This makes it possible to convert the Real Mode address to an offset within the VbeInfoBlock for Protected mode applications.

NOTE: The length of the OEMString is not defined, but for space considerations, it is recommend that a string length of less than 256 bytes is used.

The **Capabilities** field indicates the support of specific features in the graphics environment. The bits are defined as follows:

D0	= 0	DAC is fixed width, with 6 bits per primary color
	= 1	DAC width is switchable to 8 bits per primary color
D1	= 0	Controller is VGA compatible
	= 1	Controller is not VGA compatible
D2	= 0	Normal RAMDAC operation
	= 1	When programming large blocks of information to the RAMDAC, use the blank bit in Function 09h.
D3-31	=	Reserved

NOTE: BIOS Implementation -The DAC must always be restored to 6 bits per primary as default upon a mode set. If the DAC was switched to 8 bits per primary, the mode set must restore the DAC to 6 bits per primary to ensure the application programmer that he does not have to reset it.

NOTE: Application Programmer -If a DAC is switchable, the programmer can assume that the DAC is restored to 6 bits per primary upon a mode set. For an application to use a DAC, the application program is responsible for setting the DAC to 8 bits per primary mode using Function 08h.

VGA compatibility is defined as supporting all standard IBM VGA modes, fonts and I/O ports. However, VGA compatibility does not guarantee that all modes which can be set are VGA compatible, or that the 8x14 font is available.

The need for D2 = 1 “program the RAMDAC using the blank bit in Function 09h” is for older style RAMDACs, where programming the RAM values during display time causes a “snow-like” effect on the screen. Newer style RAMDACs do not have this limitation and can easily be programmed at any time. Older RAMDACs require that they be blanked so as not to display the snow while values change during display time. This bit informs the software that it should make the function call with BL=80h, rather than BL=00h, to ensure the minimization of the “snow-like” effect.

The **VideoModePtr** points to a list of mode numbers for all display modes supported by the VBE implementation. Each mode number occupies one word (16 bits). The list of mode numbers is terminated by a -1 (0FFFFh). The mode numbers in this list represent all of the potentially supported modes by the display controller. VBE 2.0 BIOS implementations must place this mode list in the Reserved area of the VbeInfoBlock or have it statically stored within the VBE implementation if 'VBE2' is preset in the VbeSignature field on entry to Function 00h.

NOTE: It is the application's responsibility to verify the actual availability of any mode returned by this function through the Return VBE Mode Information (VBE Function 01h) call. Some of the returned modes may not be available due to the actual amount of memory physically installed on the display board, or due to the capabilities of the attached monitor.

If a **VideoModeList** is found to contain no entries (starts with 0FFFFh), it can be assumed that the VBE implementation is a “stub” implementation where only Function 00h is supported for diagnostic or “Plug and Play” reasons. These stub implementations are not VBE 2.0 compliant and should be implemented only in cases where no space is available to implement the entire VBE.

The **TotalMemory** field indicates the maximum amount of memory physically installed and available to the frame buffer in 64KB units (for example, 256KB = 4, 512KB = 8). Not all video modes can address all this memory, see the ModelInfoBlock for detailed information about the addressable memory for a given mode.

The **OemSoftwareRev** field is a BCD value which specifies the OEM revision level of the VBE software. The higher byte specifies the major version number. The lower byte specifies the minor version number. This field can be used to identify the OEM's VBE software release. This field is only filled in when 'VBE2' is preset in the VbeSignature field on entry to Function 00h.

The **OemVendorNamePtr** is a pointer to a null-terminated string containing the name of the vendor which produced the display controller board product. (This string may be contained in the VbeInfoBlock or the VBE implementation.) This field is only filled in when 'VBE2' is preset in the VbeSignature field on entry to Function 00h. (Note that the length of the strings OemProductRev, OemProductName and OemVendorName (including terminators) summed, must fit within a 256 byte buffer; this is to allow for return in the OemData field if necessary.)

The **OemProductNamePtr** is a pointer to a null-terminated string containing the product name of the display controller board (this string may be contained in the VbeInfoBlock or the VBE imple-

mentation). This field is only filled in when 'VBE2' is preset in the VbeSignature field on entry to Function 00h (note that the length of the strings OemProductRev, OemProductName and OemVendorName (including terminators) summed, must fit within a 256 byte buffer; this is to allow for return in the OemData field if necessary).

The **OemProductRevPtr** is a pointer to a null-terminated string containing the revision or manufacturing level of the display controller board product (this string may be contained in the VbeInfoBlock or the VBE implementation). This field can be used to determine which production revision of the display controller board is installed. This field is only filled in when 'VBE2' is preset in the VbeSignature field on entry to Function 00h (note that the length of the strings OemProductRev, OemProductName and OemVendorName (including terminators) summed, must fit within a 256-byte buffer; this is to allow for return in the OemData field if necessary).

The **Reserved** field is a space reserved for dynamically building the VideoModeList, if necessary, if the VideoModeList is not statically stored within the VBE implementation. This field should not be used for anything else, and may be reassigned in the future. Application software should not assume that information in this field is valid.

The **OemData** field is a 256-byte data area that is used to return OEM information returned by VBE Function 00h when 'VBE2' is preset in the VbeSignature field. The OemVendorName string, OemProductName string and OemProductRev string are copied into this area by the VBE implementation. This area is only used by VBE implementations 2.0 and above when 'VBE2' is preset in the VbeSignature field.

5.3.3.2 Subfunction: 01h - Return VBE Mode Information

This required function returns extended information about a specific VBE display mode from the mode list returned by VBE Function 00h. This function fills the mode information block, ModeInfoBlock, structure with technical details on the requested mode. The ModeInfoBlock structure is provided by the application with a fixed size of 256 bytes.

Information can be obtained for all listed modes in the Video Mode List returned in Function 00h. If the requested mode cannot be used or is unavailable, a bit is set in the ModeAttributes field to indicate that the mode is not supported in the current configuration.

Input:	AH	=	4Fh	
	AL	=	01h	Return VBE mode information
	CX	=		Mode number
	ES:DI	=		Pointer to ModeInfoBlock structure
Output:	AX	=		VBE Return Status

All other registers are preserved.

The mode information block has the following structure:

ModeInfoBlock	struc				; Mandatory information for all VBE ; revisions
ModeAttributes	dw	?			; Mode attributes
WinAAttributes	db	?			; Window A attributes
WinBAttributes	db	?			; Window B attributes
WinGranularity	dw	?			; Window granularity
WinSize	dw	?			; Window size
WinASegment	dw	?			; Window A start segment
WinBSegment	dw	?			; Window B start segment
WinFuncPtr	dd	?			; Pointer to window function
BytesPerScanLine	dw	?			; Bytes per scanline
					; Mandatory information for VBE 1.2 ; and above
XResolution	dw	?			; Horizontal resolution in pixels ; or characters (pixels in ; graphics modes, ; characters in text modes.)
YResolution	dw	?			; Vertical resolution in pixels or ; characters
XCharSize	db	?			; Character cell width in pixels
YCharSize	db	?			; Character cell height in pixels
NumberOfPlanes	db	?			; Number of memory planes
BitsPerPixel	db	?			; Bits-per-pixel
NumberOfBanks	db	?			; Number of banks
MemoryModel	db	?			; Memory model type
BankSize	db	?			; Bank size in Kbytes
NumberOfImagePages	db	?			; Number of images
Reserved	db	1			; Reserved for page function ; Direct Color fields (required for direct/6 ; and YUV/7 memory models)
RedMaskSize	db	?			; Size of direct color red mask in ; bits
RedFieldPosition	db	?			; Bit position of lsb of red mask
GreenMaskSize	db	?			; Size of direct color green mask ; in bits
GreenFieldPosition	db	?			; Bit position of lsb of green ; mask
BlueMaskSize	db	?			; Size of direct color blue mask in ; bits
BlueFieldPosition	db	?			; Bit position of lsb of blue mask
RsvdMaskSize	db	?			; Size of direct color reserved ; mask in bits
RsvdFieldPosition	db	?			; Bit position of lsb of reserved ; mask
DirectColorModeInfo	db	?			; Direct color mode attributes ; Mandatory information for VBE 2.0 and

PhysBasePtr	dd	?	; above ; Physical address for flat ; memory frame buffer
OffScreenMemOffset	dd	?	; Pointer to start off screen ; memory
OffScreenMemSize	dw	?	; Amount of off screen memory in ; 1k units
Reserved	db	206 dup (?)	; Remainder of ModeInfoBlock ; ModeInfoBlock ends

The **ModeAttributes** field is defined as follows:

D0 =	Mode supported by hardware configuration 0 = Mode not supported in hardware 1 = Mode supported in hardware
D1 =	1 (Reserved)
D2 =	TTY Output functions supported by BIOS 0 = TTY Output functions not supported by BIOS 1 = TTY Output functions supported by BIOS
D3 =	Monochrome/color mode (see note below) 0 = Monochrome mode 1 = Color mode
D4 =	Mode type 0 = Text mode 1 = Graphics mode
D5 =	VGA compatible mode 0 = Yes 1 = No
D6 =	VGA compatible windowed memory mode is available 0 = Yes 1 = No
D7 =	Linear frame buffer mode is available 0 = No 1 = Yes
D8-D15 =	Reserved

Bit D0 is set to indicate that this mode can be initialized in the present hardware configuration. This bit is reset to indicate the unavailability of a graphics mode if it requires a certain monitor type, more memory than is physically installed, and so on.

Bit D1 was used by VBE 1.0 and 1.1 to indicate that the optional information following the Bytes-PerScanLine field were present in the data structure. This information became mandatory with VBE version 1.2 and above, so D1 is no longer used and should be set to 1. The Direct Color fields are only valid if the MemoryModel field is set to a 6 (Direct Color) or 7 (YUV).

Bit D2 indicates whether the video BIOS has support for output functions like TTY output, scroll, and so on, in this mode. TTY support is recommended, but not required, for all extended text and graphic modes. If bit D2 is set to 1, then the INT 10h BIOS must support all of the standard output functions listed below.

01	Set Cursor Size
02	Set Cursor Position
06	Scroll TTY window up or Blank Window
07	Scroll TTY window down or Blank Window
09	Write character and attribute at cursor position
0A	Write character only at cursor position
0E	Write character and advance cursor

Bit D3 is set to indicate color modes, and cleared for monochrome modes.

Bit D4 is set to indicate graphics modes, and cleared for text modes.

NOTE: Monochrome modes map their CRTC address at 3B4h. Color modes map their CRTC address at 3D4h. Monochrome modes have attributes in which only bit 3 (video) and bit 4 (intensity) of the attribute controller output are significant. Therefore, monochrome text modes have attributes of off, video, high intensity, blink, and so on. Monochrome graphics modes are two plane graphics modes and have attributes of off, video, high intensity, and blink. Extended two color modes that have their CRTC address at 3D4h are color modes with one bit-per-pixel and one plane. The standard VGA modes 06h and 11h would be classified as color modes, while the standard VGA modes 07h and 0Fh would be classified as monochrome modes.

Bit D5 is used to indicate if the mode is compatible with the VGA hardware registers and I/O ports. If this bit is set, then the mode is NOT VGA compatible and no assumptions should be made about the availability of any VGA registers. If clear, then the standard VGA I/O ports and frame buffer address, defined in WinASegment and/or WinBSegment can be assumed.

Bit D6 is used to indicate if the mode provides Windowing or Banking of the frame buffer into the frame buffer memory region specified by WinASegment and WinBSegment. If set, then Windowing of the frame buffer is NOT possible. If clear, then the device is capable of mapping the frame buffer into the segment specified in WinASegment and/or WinBSegment (this bit is used in conjunction with bit D7, see table following D7 for usage).

Bit D7 indicates the presence of a Linear Frame Buffer memory model. If this bit is set, the display controller can be put into a flat memory model by setting the mode (VBE Function 02h) with the Flat Memory Model bit set (this bit is used in conjunction with bit D6, see following table for usage).

Bit 7 and Bit 6 Usage

	D7	D6
Windowed frame buffer only	0	0
n/a	0	1
Both Windowed and Linear*		0
Linear frame buffer only	1	1

*Use D14 of the Mode Number to select the Linear Buffer on mode set (Function 02h).

The **BytesPerScanLine** field specifies how many full bytes are in each logical scanline. The logical scanline could be equal to or larger than the displayed scanline.

The **WinAAttributes** and **WinBAttributes** describe the characteristics of the CPU windowing scheme such as whether the windows exist and are read/writable, as follows:

- D0 = Relocatable window(s) supported
 - 0 = Single non-relocatable window only
 - 1 = Relocatable window(s) are supported
- D1 = Window readable
 - 0 = Window is not readable
 - 1 = Window is readable
- D2 = Window writable
 - 0 = Window is not writable
 - 1 = Window is writable
- D3-D7= Reserved

Even if windowing is not supported (bit D0 = 0 for both Window A and Window B), then an application can assume that the display memory buffer resides at the location specified by WinASegment and/or WinBSegment.

WinGranularity specifies the smallest boundary, in Kbytes, on which the window can be placed in the frame buffer memory. The value of this field is undefined if Bit D0, of the appropriate WinAttributes field, is not set.

WinSize specifies the size of the window in Kbytes.

WinASegment and **WinBSegment** address specify the segment addresses where the windows are located in the CPU address space.

WinFuncPtr specifies the segment:offset of the VBE memory windowing function. The windowing function can be invoked either through VBE Function 05h, or by calling the function directly. A direct call will provide faster access to the hardware paging registers than using VBE Function 05h, and is intended to be used by high performance applications. If this field is NULL, then VBE Function 05h must be used to set the memory window when paging is supported. This direct call method uses the same parameters as VBE Function 05h, including AX, and for VBE 2.0 implementations will return the correct Return Status. VBE 1.2 implementations and earlier did not require the Return Status information to be returned. For more information on the direct call method, see the notes in VBE Function 05h.

The **XResolution** and **YResolution** specify the width and height in pixel elements or characters for this display mode. In graphics modes, these fields indicate the number of horizontal and vertical pixels that may be displayed. In text modes, these fields indicate the number of horizontal and vertical character positions. The number of pixel positions for text modes may be calculated by multiplying the returned XResolution and YResolution values by the character cell width and height indicated in the XCharSize and YCharSize fields described below.

The **XCharSize** and **YCharSize** specify the size of the character cell in pixels. This value is not zero based. For example, XCharSize for Mode 3, using the 9 point font, will have a value of 9.

The **NumberOfPlanes** field specifies the number of memory planes available to software in that mode. For standard 16-color VGA graphics, the field would be set to 4. For standard packed pixel modes, the field would be set to 1. For 256-color non-chain-4 modes, where the programmer needs to do banking to address all pixels, this value should be set to the number of banks required to get to all the pixels (typically this would be 4 or 8).

The **BitsPerPixel** field specifies the total number of bits allocated to one pixel. For example, a standard VGA 4 Plane 16-color graphics mode would have a 4 in this field and a packed pixel 256-color graphics mode would specify 8 in this field. The number of bits-per-pixel per plane can normally be derived by dividing the BitsPerPixel field by the NumberOfPlanes field.

The **MemoryModel** field specifies the general type of memory organization used in this mode. The following models have been defined:

00h	=	Text mode
01h	=	CGA graphics
02h	=	N/A
03h	=	Planar
04h	=	Packed pixel
05h	=	Non-chain 4, 256 colors
06h	=	Direct Color
07h	=	YUV
08h-0Fh	=	Reserved, to be defined by VESA
10h-FFh	=	To be defined by OEM

VBE Version 1.1 and earlier defined Direct Color graphics modes with pixel formats 1:5:5:5, 8:8:8, and 8:8:8:8 as a Packed Pixel model with 16, 24, and 32 bpp, respectively. In VBE Version 1.2 and later, the Direct Color modes use the Direct Color memory model and use the MaskSize and Field-Position fields of the ModeInfoBlock to describe the pixel format. BitsPerPixel is always defined to be the total memory size of the pixel, in bits.

NumberOfBanks. This is the number of banks in which the scanlines are grouped. The quotient from dividing the scanline number by the number of banks is the bank that contains the scanline and the remainder is the scanline number within the bank. For example, CGA graphics modes have two banks and Hercules graphics mode has four banks. For modes that do not have scanline banks (such as VGA modes 0Dh-13h), this field should be set to 1.

The **BankSize** field specifies the size of a bank (group of scanlines) in units of 1 Kbyte. For CGA and Hercules graphics modes this is 8, as each bank is 8192 bytes in length. For modes that do not have scanline banks (such as VGA modes 0Dh-13h), this field should be set to 0.

The **NumberOfImagePages** field specifies the “total number minus one (-1)” of complete display images that will fit into the frame buffer memory. The application may load more than one image into the frame buffer memory if this field is non-zero, and move the display window within each of those pages. This should be used only to determine the additional display pages that are available to the application. To determine the available off screen memory, use the OffScreenMemOffset and OffScreenMemSize information.

NOTE: If the **ModeInfoBlock** is for an IBM Standard VGA mode and the NumberOfImagePages field contains more pages than would be found in a 256 Kbytes implementation, the TTY support described in the ModeAttributes must be accurate. That is, if the TTY functions are claimed to be supported, they must be supported in all pages, not just the pages normally found in the 256 Kbytes implementation.

The **Reserved** field is defined to support a future VBE feature and will always be set to one in this version.

The **RedMaskSize**, **GreenMaskSize**, **BlueMaskSize**, and **RsvdMaskSize** fields define the size, in bits, of the red, green, and blue components of a direct color pixel. A bitmask can be constructed from the MaskSize fields using simple shift arithmetic. For example, the MaskSize values for a Direct Color 5:6:5 mode would be 5, 6, 5, and 0, for the red, green, blue, and reserved fields, respectively. Note that in the YUV MemoryModel, the red field is used for V, the green field is used for Y, and the blue field is used for U. The MaskSize fields should be set to 0 in modes using a memory model that does not have pixels with component fields.

The **RedFieldPosition**, **GreenFieldPosition**, **BlueFieldPosition**, and **RsvdFieldPosition** fields define the bit position within the direct color pixel or YUV pixel of the least significant bit of the respective color component. A color value can be aligned with its pixel field by shifting the value left by the FieldPosition. For example, the FieldPosition values for a Direct Color 5:6:5 mode would be 11, 5, 0, and 0, for the red, green, blue, and reserved fields, respectively. Note that in the YUV MemoryModel, the red field is used for V, the green field is used for Y, and the blue field is used for U. The FieldPosition fields should be set to 0 in modes using a memory model that does not have pixels with component fields.

The **DirectColorModelInfo** field describes important characteristics of direct color modes. Bit D0 specifies whether the color ramp of the DAC is fixed or programmable. If the color ramp is fixed, then it can not be changed. If the color ramp is programmable, it is assumed that the red, green, and blue lookup tables can be loaded by using VBE Function 09h. Bit D1 specifies whether the bits in the Rsvd field of the direct color pixel can be used by the application or are reserved, and thus unusable.

D0 =	Color ramp is fixed/programmable
0 =	Color ramp is fixed
1 =	Color ramp is programmable
D1 =	Bits in Rsvd field are usable/reserved
0 =	Bits in Rsvd field are reserved
1 =	Bits in Rsvd field are usable by the application

The **PhysBasePtr** is a 32-bit physical address of the start of frame buffer memory when the controller is in flat frame buffer memory mode. If this mode is not available, then this field is zero.

The **OffScreenMemOffset** is a 32-bit offset from the start of the frame buffer memory. Extra off-screen memory that is needed by the controller may be located either before or after this off-screen memory. The programmer should be sure to check OffScreenMemSize to determine the amount of off-screen memory which is available to the application.

The **OffScreenMemSize** contains the amount of available, contiguous off-screen memory in 1 Kbyte units which can be used by the application.

NOTE: VBE version 1.1 and later will zero out all unused fields in the Mode Information Block, always returning exactly 256 bytes. This facilitates upward compatibility with future versions of the standard, as any newly added fields are designed such that values of zero will indicate nominal defaults or non-implementation of optional features (for example, a field containing a bitmask of extended capabilities would reflect the absence of all such capabilities). Applications that wish to

be backwards compatible to VBE version 1.0 should pre-initialize the 256 byte buffer before calling the Return VBE Mode Information function.

5.3.3.3 **Subfunction: 02h - Set VBE Mode**

This required function initializes the controller and sets a VBE mode. The format of VESA VBE mode numbers is described earlier in this chapter. If the mode cannot be set, the BIOS should leave the graphics environment unchanged and return a failure error code.

Input:	AH	= 4Fh	
	AL	= 02h	Set VBE Mode
	BX	=	Desired Mode to set
	D0-D8	=	Mode number
	D9-D13	=	Reserved (must be 0)
	D14	= 0	Use windowed frame buffer model
		= 1	Use linear/flat frame buffer model
	D15	= 0	Clear display memory
		= 1	Don't clear display memory
Output:	AX	=	VBE Return Status

All other registers are preserved.

If the requested mode number is not available the call will fail, returning AH=01h to indicate the failure to the application.

If bit D14 is set, the mode is initialized for use with a flat frame buffer model. The base address of the frame buffer can be determined from the extended mode information returned by VBE Function 01h. If D14 is set and a linear frame buffer model is not available, then the call will fail, returning AH=01h to the application.

If bit D15 is not set, all reported image pages based on Function 00h returned information NumberOfImagePages are cleared to 00h in graphics mode and 20 07 in text mode. Memory over and above the reported image pages are not changed. If bit D15 is set, then the contents of the frame buffer after the mode change is undefined. Note that the 1-byte mode numbers used in Function 00h of an IBM VGA compatible BIOS use D7 to signify the same thing as D15 does in this function. If D7 is set for an IBM compatible mode set using this Function, 02h, this mode set will fail. VBE aware applications must use the memory clear bit in D15.

NOTE: VBE BIOS 2.0 implementations should also update the BIOS Data Area 40:87 memory clear bit so that VBE Function 03h can return this flag. VBE BIOS 1.2 and earlier implementations ignore the memory clear bit.

NOTE: This call should not set modes not listed in the list of supported modes. In addition, all modes (including IBM standard VGA modes), if listed as supported, must have ModeInfoBlock structures associated with them.

5.3.3.4 Subfunction: 03h - Return current VBE Mode

This required function returns the current VBE mode. The format of VBE mode numbers is described earlier in this chapter.

Input:	AH	= 4Fh	
	AL	= 03h	Return current VBE mode
Output:	AX	=	VBE Return Status
	BX	=	Current VBE mode
	D0-D13	=	Mode number
	D14	= 0	Windowed frame buffer model
		= 1	Linear/flat frame buffer model
	D15	= 0	Memory cleared at last mode set
		= 1	Memory not cleared at last mode set

NOTES:

- 1) All other registers are preserved.
- 2) Version 1.x - In a standard VGA BIOS, Function 0Fh (Read current video state) returns the current graphics mode in the AL register. In D7 of AL, it also returns the status of the memory clear bit (D7 of 40:87). This bit is set if the mode was set without clearing memory. In this VBE function, the memory clear bit are not returned in BX because the purpose of the function is only to return the video mode. If an application wants to obtain the memory clear bit, it should call the standard VGA BIOS Function 0Fh.
- 3) Version 2.x - Unlike version 1.x VBE implementations, the memory clear flag is returned. The application should NOT call the standard VGA BIOS Function 0Fh if the mode was set with VBE Function 02h.
- 4) The mode number returned must be the same mode number used in the VBE Function 02h mode set.
- 5) This function is not guaranteed to return an accurate mode value if the mode set was not done with VBE Function 02h.

5.3.3.5 Subfunction: 04h - Save/Restore State

This required function provides a complete mechanism to save and restore the display controller hardware state. The functions are a superset of the three subfunctions under the standard VGA BIOS Function 1Ch (Save/restore state) which does not guarantee that the extended registers of the video device are saved or restored. The complete hardware state (except frame buffer memory) should be saveable/restorable by setting the requested states mask (in the CX register) to 000Fh.

Input:	AH	=	4Fh	
	AL	=	04h	Save/Restore State
	DL	=	00h	Return Save/Restore State buffer size
		=	01h	Save state
		=	02h	Restore state
	CX	=		Requested states
	D0	=		Save/Restore controller hardware state
	D1	=		Save/Restore BIOS data state
	D2	=		Save/Restore DAC state
	D3	=		Save/Restore Register state
ES:BX		=	Pointer to buffer (if DL <> 00h)	
Output:	AX	=		VBE Return Status
	BX	=		Number of 64-byte blocks to hold the state buffer (if DL=00h)

NOTES:

- 1) All other registers are preserved.

5.3.3.6 Subfunction: 05h - Display Window Control

This required function sets or gets the position of the specified display window or page in the frame buffer memory by adjusting the necessary hardware paging registers. To use this function properly, the software should first use VBE Function 01h (Return VBE Mode information) to determine the size, location and granularity of the windows.

For performance reasons, it may be more efficient to call this function directly, without incurring the INT 10h overhead. VBE Function 01h returns the segment:offset of this windowing function that may be called directly for this reason. Note that a different entry point may be returned based upon the selected mode. Therefore, it is necessary to retrieve this segment:offset specifically for each desired mode.

Input:	AH	=	4Fh	
	AL	=	05h	VBE Display Window Control
	BH	=	00h	Set memory window
		=	01h	Get memory window
	BL	=		Window number
		=	00h	Window A
		=	01h	Window B
	DX	=		Window number in video memory in window granularity units (Set Memory Window only)
Output:	AX	=		VBE Return Status
	DX	=		Window number in window granularity units (Get Memory Window only)

NOTES:

- 1) In VBE 1.2 implementations, the direct far call version returns no Return Status information to the application. Also, in the far call version, the AX and DX registers is destroyed. Therefore, if AX and/or DX must be preserved, the application must do so prior to making the far call. The application must still load the input arguments in BH, BL, and DX (for Set Window). In VBE 2.0 implementations, the BIOS will return the correct Return Status, and therefore the application must assume that AX and DX are destroyed.
- 2) Application Programmers - This function is not intended for use in a linear frame buffer mode. If this function is requested, the function call will fail with the VBE Completion code AH=03h.
- 3) VBE BIOS Implementation - If this function is called while in a linear frame buffer memory model, this function must fail with completion code AH=03h.

5.3.3.7 Subfunction: 06h - Set/Get Logical Scanline Length

This required function sets or gets the length of a logical scanline. This allows an application to set up a logical display memory buffer that is wider than the displayed area. VBE Function 07h (Set/Get Display Start) then allows the application to set the starting position that is to be displayed.

Input:	AH	=	4Fh	
	AL	=	06h	VBE Set/Get Logical Scanline Length
	BL	=	00h	Set Scanline Length in Pixels
		=	01h	Get Scanline Length
		=	02h	Set Scanline Length in Bytes
		=	03h	Get Maximum Scanline Length
	CX	=		If BL=00h Desired Width in Pixels If BL=02h Desired Width in Bytes (Ignored for Get Functions)
Output:	AX	=		VBE Return Status
	BX	=		Bytes Per Scanline
	CX	=		Actual Pixels Per Scanline (truncated to nearest complete pixel)
	DX	=		Maximum Number of Scanlines

NOTES:

- 1) The desired width in pixels or bytes may not be achievable because of hardware considerations. The next larger value is selected that will accommodate the desired number of pixels or bytes, and the actual number of pixels is returned in CX. BX returns a value that, when added to a pointer into display memory will point to the next scanline. For example, in VGA mode 13h this would be 320, but in mode 12h this would be 80. DX returns the number of logical scanlines based upon the new scanline length and the total memory installed and usable in this display mode.
- 2) This function is also valid in VBE supported text modes. In VBE supported text modes the application should convert the character line length to pixel line length by getting the current character cell width through the XCharSize field returned in ModeInfoBlock, multiplying that times the desired number of characters per line and passing that value in the CX register. In addition, this function will only work if the line length is specified in character granularity. That is, only in 8 dot modes will multiples of 8 work. Any value which is not in character granularity will result in a function call failure.
- 3) On a failure to set scanline length by setting a CX value too large, the function will fail with error code 02h.
- 4) The value returned when BL=03h is the lesser of either the maximum line length that the hardware can support, or the longest scanline length that would support the number of lines in the current video mode.

5.3.3.8 Subfunction: 07h - Set/Get Display Start

This required function selects the pixel to be displayed in the upper left corner of the display. This function can be used to pan and scroll around logical screens that are larger than the displayed screen. This function can also be used to rapidly switch between two different displayed screens for double buffered animation effects.

Input:	AH	= 4Fh	
	AL	= 07h	VBE Set/Get Display Start Control
	BH	= 00h	Reserved and must be 00h
	BL	= 00h	Set Display Start
		= 01h	Get Display Start
CX	= 80h	Set Display Start during Vertical Retrace	
	=	First Displayed Pixel In Scanline (Set Display Start only)	
	=	First Displayed Scanline (Set Display Start only)	
Output:	AX	=	VBE Return Status
	BH	=	00h Reserved and is 0 (Get Display Start only)
	CX	=	First Displayed Pixel In Scanline (Get Display Start only)
	DX	=	First Displayed Scanline (Get Display Start only)

NOTES:

This function is also valid in text modes. To use this function in text mode, the application should convert the character coordinates to pixel coordinates by using XCharSize and YCharSize returned in the ModeInfoBlock. If the requested Display Start coordinates do not allow for a full page of video memory or the hardware does not support memory wrapping, the Function call should fail and no changes should be made. As a general case, if a requested Display Start is not available, fail the Function call and make no changes.

5.3.3.9 Subfunction: 08h - Set/Get DAC Palette Format

This required function manipulates the operating mode or format of the DAC palette. Some DACs are configurable to provide 6 bits, 8 bits, or more of color definition per red, green, and blue primary colors. The DAC palette width is assumed to be reset to the standard VGA value of 6 bits per primary color during any mode set.

Input:	AH	=	4Fh	
	AL	=	08h	VBE Set/Get Palette Format
	BL	=	00h	Set DAC Palette Format
		=	01h	Get DAC Palette Format
	BH	=		Desired bits of color per primary (Set DAC Palette Format only)
Output:	AX	=		VBE Return Status
	BH	=		Current number of bits of color per primary

An application can determine if DAC switching is available by querying Bit D0 of the Capabilities field of the VbeInfoBlock structure returned by VBE Function 00h (Return Controller Information). The application can then attempt to set the DAC palette width to the desired value. If the display controller hardware is not capable of selecting the requested palette width, then the next lower value that the hardware is capable of is selected. The resulting palette width is returned.

This function will return failure code AH=03h if called in a direct color or YUV mode.

5.3.3.10 Subfunction: 09h - Set/Get Palette Data

This required function is very important for RAMDACs which are larger than a standard VGA RAM-DAC. The standard INT 10h BIOS Palette function calls assume standard VGA ports and VGA palette widths. This function offers a palette interface that is independent of the VGA assumptions.

Input:	AH	= 4Fh	
	AL	= 09h	VBE Load/Unload Palette Data
	BL	= 00h	Set Palette Data
		= 01h	Get Palette Data
		= 02h	Set Secondary Palette Data
		= 03h	Get Secondary Palette Data
		= 80h	Set Palette Data during Vertical Retrace with Blank Bit on
	CX	=	Number of palette registers to update (to a maximum of 255)
	DX	=	First of the palette registers to update (start)
ES:DI		=	Table of palette values (see below for format)
Output:	AX	=	VBE Return Status

Format of Palette Values: Alignment byte, Red byte, Green byte, Blue byte

NOTES:

- 1) The need for BL= 80h is for older style RAMDACs where programming the RAM values during display time causes a "snow-like" effect on the screen. Newer style RAMDACs do not have this limitation and can easily be programmed at any time. Older RAMDACs require that they only be programmed during a non-display time to stop the "snow-like" effect seen when changing the DAC values. When this is requested, the VBE implementation programs the DAC with blanking on. Check D2 of the Capabilities field returned by VBE Function 00h to determine if 80h should be used instead of 00h.
- 2) The need for the secondary palette is for anticipated future palette extensions. If a secondary palette does not exist in an implementation and these calls are made, the VBE implementation will return error code 02h.
- 3) When in 6-bit mode, the format of the 6 bits is LSB. This is done for speed reasons, as the application can typically shift the data faster than the BIOS can.
- 4) All applications should assume the DAC is defaulted to 6-bit mode. The application is responsible for switching the DAC to higher color modes using Function 08h.
- 5) Query VBE Function 08h to determine the RAMDAC width before loading a new palette.

5.3.3.11 Subfunction: 0Ah - VBE Protected Mode Interface

This required function call returns a pointer to a table that contains code for a 32-bit protected mode interface that can either be copied into local 32-bit memory space or can be executed from ROM, providing the calling application sets all required selectors and I/O access correctly. This function returns a pointer (in real mode space) with offsets to the code fragments, and additionally returns an offset to a table which contains non-VGA port and memory locations to which an application may have to have I/O access.

Input:	AH	= 4Fh	
	AL	= 0Ah	VBE 2.0 Protected Mode Interface
	BL	= 00h	Return protected mode table
Output:	AX	=	Status
	ES	=	Real Mode Segment of Table
	DI	=	Offset of Table
	CX	=	Length of Table including protected mode code in bytes (for copying purposes)

The format of the table is as follows:

ES:DI	= 00h	Word Offset in table of Protected mode code for Function 5 for Set Window Call
ES:DI	= 02h	Word Offset in table of Protected mode code for Function 7 for set Display Start
ES:DI	= 04h	Word Offset in table of Protected mode code for Function 9 for set Primary Palette data
ES:DI	=06h	Word Offset in table of Ports and Memory Locations for which the application may need I/O privilege. (Optional: if unsupported this must be 0000h) (See Sub-table for format)
ES:DI	=?	Variable remainder of Table including Code

The format of the Sub-Table (Ports and Memory locations)

Port, Port, ... , Port, Terminate Port List with FF FF, Memory locations (4 bytes), Length (2 bytes), Terminate Memory List with FF FF.

Example 1. For Port/Index combination 3DE/Fh and Memory locations DE800-DEA00h (length = 200h) the table would look like:

DE 03 DF 03 FF FF 00 E8 0D 00 00 02 FF FF

Example 2. For only the ports, it would look like:

DE 03 DF 03 FF FF FF FF

Example 3. For only the memory locations, it would look like:

FF FF 00 E8 0D 00 00 02 FF FF

NOTES:

1) All protected mode functions should end with a near RET (as opposed to FAR RET) to allow the application software to CALL the code from within the ROM. The Port and Memory location Sub-table does not include the Frame Buffer Memory location. The Frame Buffer Memory location is contained within the ModelInfoBlock returned by VBE Function 01h.

2) The protected mode code is assembled for a 32-bit code segment. When copying it, the application must copy the code to a 32-bit code segment. It is the application's responsibility to ensure that the selectors and segments are set up correctly. If the memory location is zero, then only I/O mapped ports will be used so the application does not need to do anything special. This should be the default case for ALL cards that have I/O mapped registers because it provides the best performance.

If the memory location is nonzero (there can be only one), the application will need to create a new 32-bit selector with the base address that points to the "physical" location specified with the specified limit.

When the application needs to call the 32-bit bank switch function, it must then load the ES selector with the value of the new selector that has been created. The bank switching code can then directly access its memory mapped registers as absolute offsets into the ES selector (i.e., `mov [es:10],eax` to put a value into the register at `base+10`).

It is up to the application code to save and restore the previous state of the ES selector if this is necessary (for example in flat model code).

3) Currently undefined registers may be destroyed with the exception of ESI, EBP, DS and SS.

4) Applications must use the same registers for the Function 05h and Function 09h protected mode interface that it would use in a real mode call. This includes the AX register.

5) Function 07h protected mode calls have a different format.

AH	=	4Fh	
AL	=	07h	
BL	=	00h	Set Display CRTC Start
		80h	Set Display CRTC Start during Vertical Retrace
CX	=		Bits 0-15 of display start address
DX	=		Bits 16-31 of display start address

The protected mode application must keep track of the color depth and scanline length to calculate the new start address. If a value that is out of range is programmed, unpredictable results will occur.

5.3.4 VBE Display Power Management Subfunctions(10h)

The VESA VBE Subfunction 10h is used to implement the VBE/Power Management (PM) services. The VBE/PM services are defined below and are not included in the VBE Standard documentation.

5.3.4.1 Subfunction: 00h - Report VBE/PM Capabilities

Input:	AH =	4Fh	VESA Extension
	AL =	10h	VBE/PM Services
	BL =	00h	Report VBE/PM Capabilities
	CX =	00h	Controller unit number (00 = primary controller)
	ES:DI		Null pointer, must be 0000:0000h in version 1.0. Reserved for future use.
Output:	AX =	Status	
	BH =	Power saving state signals supported by the controller(Note 1). 1 = supported, 0 = not supported.	
	bit 0	STAND BY	
	bit 1	SUSPEND	
	bit 2	OFF	
	bit 3	REDUCED ON (Note 2)	
	bits 4-7	Reserved for future power control of the display controller or other related circuits.	
	BL =	VBE/PM Version number (0001 0000b for this version)	
	bits 0-3	Minor Version number	
	bits 4-7	Major Version number	
	CX=	Unchanged	
	ES:DI	Unchanged	

All other registers may be destroyed.

NOTES:

- 1) 1) The attached display may not support all the power states that can be signaled by the controller. It is the responsibility of the power management program to determine the power saving states that are offered by the controller. If the controller has a means of determining which power saving state that is implemented in the attached display device, this function reports the power saving states that are supported by both the controller and the display.
- 2) 2) REDUCED ON is not supported by DPMS 1.0 displays and is intended for use by Flat Panel Displays.

5.3.4.2 Subfunction: 01h - Set Display Power State

Input:	AH =	4Fh	VESA Extension
	AL =	10h	VBE/PM Services
	BL =	01h	Set Display Power state
	BH =		Requested Power state
		00h	ON
		01h	STAND BY
		02h	SUSPEND
		04h	OFF
		08h	REDUCED ON (Note 2)
			All other BH values are currently undefined and are reserved for future power control of the display controller.
Output:	CX =	00h	Controller unit number (00 = primary controller)
	AX =		Status (Note 1)
	BH =		Unchanged
	CX =		Unchanged

All other registers may be destroyed.

NOTES:

- 1) If the requested state is not available, this function will return AX=014Fh to indicate that the function is supported but the call failed. The BH register and Display Power State shall be left unchanged in this case.
- 2) REDUCED ON is not supported by DPMS 1.0 displays and is intended for use by Flat Panel Displays.

5.3.4.3 Subfunction: 02h - Get Display Power State

Input:	AH =	4Fh	VESA Extension
	AL =	10h	VBE/PM Services
	BL =	02h	Get Display Power state
	CX =	00h	Controller unit number (00 = primary controller)
Output:	AX =		Status (Note 1)
	BH =		Power state currently requested by the controller
		00h	ON
		01h	STAND BY
		02h	SUSPEND
		04h	OFF

08h REDUCED ON (Note 2)
 All other BH values are reserved and may be used to signal other power saving states in future revisions of VBE/PM. To ensure future compatibility, applications written for VBE/PM 1.0 should ignore the value of bits 4-7.

CX = Unchanged
 All other registers may be destroyed.

NOTES:

- 1) If this function is not supported by the controller hardware, AL=01 should be returned in the status register. This function should always read the controller hardware to determine the current power state. This function should not rely on any form of information stored in memory about what power state was last initiated through a VBE/PM call. The power state signaling can be initiated through many different hardware and software interfaces, making any stored information potentially invalid.
- 2) REDUCED ON is not supported by DPMS 1.0 displays and is intended for use by Flat Panel Displays.

5.3.5 VBE Display Identification (DDC) Subfunctions(15h)

The VESA VBE subfunction 15h is used to implement the VBE/DDC services. The VBE/DDC interface allows the operating system and application software to retrieve the display identity information without specific hardware knowledge or direct hardware access. The hardware protocol for retrieving the display identity is described in the VESA standard DDC 1.0. The VBE/ DDC services are defined below.

5.3.5.1 Subfunction: 00h - Report VBE/DDC Capabilities

Input:	AH =	4Fh	VESA Extension
	AL =	15h	VBE/DDC Services
	BL =	00h	Report DDC Capabilities
	CX =	00h	Controller unit number (00 = primary controller)
	ES:DI		Null pointer, must be 0:0 in version 1.0. Reserved for future use.
Output:	AX =		Status
	BH =		Approximate time in seconds, rounded up, to transfer one EDID block (128 bytes).
	BL =		DDC level supported (Note 1)
	bit 0	= 0	DDC1 not supported
		= 1	DDC1 supported
	bit 1	= 0	DDC2 not supported
		= 1	DDC2 supported
	bit 2	= 0	Screen not blanked during data transfer (Note 2)
		= 1	Screen blanked during data transfer
	CX =		Unchanged
	ES:DI		Unchanged
	All other registers may be destroyed.		

NOTES:

- 1) DDC level supported by both the display and the controller.
- 2) This refers to the behavior of the controller and the VBE/DDC software.

5.3.5.2 Subfunction: 01h - Read EDID

Input:	AH =	4Fh	VESA Extension
	AL =	15h	VBE/DDC Services
	BL =	01h	Read EDID
	CX =	00h	Controller unit number (00 = primary controller)
	DX =	00h	EDID block number. Zero is only valid value in version 1.0.
	ES:DI		Pointer to area in which the EDID block (128 bytes) shall be returned.
Output:	AX =		Status
	BH =		Unchanged
	CX =		Unchanged
	ES:DI =		Pointer to area in which the EDID block is returned.
			All other registers may be destroyed.

5.3.6 VBE / Flat Panel Subfunctions (11h)

5.3.6.1 Flat Panel Interface Functions

The following functions are defined by the VBE/FP standard. VBE/FP functions are called using the INT 10h interrupt vector, passing arguments in the 80x86 registers. The INT 10h interrupt handler first determines if a VBE/FP function has been requested, and if so, processes that request; otherwise control is passed down the chain to the next INT 10h handler.

All VBE/FP functions are called with the AH register set to 4Fh and the AL register set to 11h. The AH register is used to determine that it is a VBE call, the AL register is used to determine that the Subfunction is a Flat Panel Extension.

For future revisions of the specification, VBE/FP assumes that other registers in a function call will be destroyed; the exception to this is SI, BP, DS, SS for obvious reasons and ES:DI when not used to return information. This assumption allows for the use of other currently undefined registers in each function, if necessary.

Reserved values should always be set to the value zero (0).

5.3.6.2 VBE/FP Completion Codes

The AX register is used to indicate the completion status upon return from VBE functions. If VBE support for the specified function is available, the 4Fh value passed in the AH register on entry is returned in the AL register. If the VBE function completed successfully, 00h is returned in the AH register. Otherwise the AH register is set to indicate the nature of the failure.

VBE RETURN STATUS

AL	== 4Fh:	Function is supported
AL	!= 4Fh:	Function is not supported
AH	== 00h:	Function call successful
AH	== 01h:	Function call failed
AH	== 02h:	Function is not supported in the current hardware configuration
AH	== 03h:	Function call invalid in current video mode

Note: Applications should treat any non-zero value in the AH register as a general failure condition as later versions of the VBE may define additional error codes.

5.3.6.3 Subfunction 00h - Panel Extensions Support

This function is designed to allow application developers to get information about the Flat Panel Extensions. This function is required by VBE 2.0 as a Supplemental Specification. For more details on the decisions for this function, please refer to the VESA BIOS Extensions Standard version 2.0 Chapter 5.

Input:	AH	= 4F	
	AL	= 11h	VBE Flat Panel Extension request
	BL	= 00h	Return Flat Panel Extensions Support Information
	ES:DI	=	Pointer to 256 byte buffer in which to place SupVbeInfoBlock structure
Output:	AX	=	VBE Return Status

(Note: Currently undefined registers may be destroyed with the exception of SI,BP,DS and SS)

Description of the SupVbeInfoBlock structure

The information block has the following structure:

```

SupVbeInfoBlock struc
SupVbeSignature      db    'VBE/FP',0FFh  ; Supplemental VBE Signature
SupVbeVersion        dw    0100h          ; Supplemental VBE Version
SupVbeSubFunc        db    03h,7 dup (0)  ; Bitfield of supported subfunctions
OemSoftwareRev       dw    ?              ; OEM Software revision
OemVendorNamePtr     dd    ?              ; Pointer to Vendor Name String
OemProductNamePtr    dd    ?              ; Pointer to Product Name String
OemProductRevPtr     dd    ?              ; Pointer to Product Revision String
OemStringPtr         dd    ?              ; Pointer to OEM String
Reserved             db    221 dup (?)    ; Reserved for description strings and
                                           ; future expansion
SupVbeInfoBlock ends
    
```

Note: All data in this structure is subject to change by the VBE implementation when any VBE Subfunction 00h is called. Therefore it should not be used by the application to store data of any kind; if the information is required at a later time, a local copy of the data should be made.

The **SupVbeSignature** field is filled with the ASCII characters 'VBE/FP'. The 7th character must be filled with FFh.

The **SupVbeVersion** is a BCD value which specifies what version of VBE/FP is implemented in the software. The higher byte specifies the major version number. The lower byte specifies the minor version number. The current value is: 0100h.

The **SupVbeSubFunc** is a bitfield that represents the subfunctions available in this implementation of VBE/FP. If the bit representing a particular subfunction is set to 1, then that subfunction is supported. Subfunction '00h' is represented by the LSB of the first byte, bit 1 corresponds to subfunction 01h and the other subfunctions follow suit. The LSB of byte 2 corresponds to subfunction 09h etc. Only bits for subfunctions defined in the specification can be set to 1, other bits should be set to 0. The required function set for VBE/FP is 0000 001, the other bits must be set to 1 if the corresponding subfunctions are implemented. In addition, any subfunctions not supported should turn the corresponding bits to 0 and must return AL!=4Fh as a completion code.

The **OemSoftwareRev** field is a BCD value which specifies the OEM revision level of VBE/FP. The higher byte specifies the major version number. The lower byte specifies the minor version number. This field can be used to identify the OEM's software release, so that upgrades can be identified.

The **OemVendorNamePtr** is a Real Mode far pointer to a null-terminated OEM-defined string containing the name of the vendor who produced the display controller board product. Other supplemental functions may point to the same location.

The **OemProductNamePtr** is a Real Mode far pointer to a null-terminated OEM-defined string containing the product name of the display controller board. Other supplemental functions may point to the same location.

The **OemProductRevPtr** is a Real Mode far pointer to a null-terminated OEM-defined string containing the revision or manufacturing level of the display controller board product. This field can be used to determine which production revision of the display controller board is installed. Other supplemental functions may point to the same location.

The **OemStringPtr** is a Real Mode far pointer to a null-terminated OEM-defined string. This string may be used to identify the graphics controller chip or OEM product family for hardware specific display drivers. There are no restrictions on the format of the string. This pointer may point into either ROM or RAM, depending on the specific implementation. Other supplemental functions may point to the same location.

5.3.6.4 Subfunction 01h - Return Flat Panel Info

This subfunction allows application software to determine information about the currently attached flat panel.

Input:	AH	= 4F	
	AL	= 11h	VBE Flat Panel Extension request
	BL	= 01h	Return Flat Panel Information
	ES:DI	=	Pointer to 32 byte buffer in which to place FPIInfo structure
Output:	AX=		VBE Return Status

(Note: Currently undefined registers may be destroyed with the exception of SI, BP, DS and SS)

FPIInfo struc			
HSize	dw	?	; Horizontal Size in Pixels
VSize	dw	?	; Vertical Size in Lines
FPTYPE	dw	?	; Technology (TSTN LCD, TFT LCD, Plasma, EL), Single/Dual, Color/Mono
RedBPP	db	?	; Red Bits Per Primary
GreenBPP	db	?	; Green Bits Per Primary
BlueBPP	db	?	; Blue Bits Per Primary
ReservedBPP	db	0	; Reserved Bits Per Primary
RsvdOffScrnMemSize	dd	?	; Size in KB of Offscreen Memory
			; required frame buffer
RsvdOffScrnMemPtr	dd	?	; Pointer to reserved offscreen memory
Reserved	db	14 dup(0)	; remainder of FPIInfo
FPIInfo ends			

The **HSize** field is an integer value representing the horizontal size of the flat panel in pixels. A typical 640x480 flat panel will have an HSize = 640 (280h).

The **VSize** field is an integer value representing the vertical size of the flat panel in pixels. A typical 640x480 flat panel will have an VSize = 480 (1E0h).

The **FPTYPE** describes the type of the flat panel as follows:

D0	= 0	Monochrome
	= 1	Color
D1	= 0	Single panel construction
	= 1	Dual (Split) panel construction
D2-D7	= 0000 00	STN (Passive Matrix)
	= 0000 01	TFT (Active Matrix)
	= 0000 10	Other LCD
	= 0000 11	EL
	= 0001 00	Plasma
D8-D15	=	Reserved

The **RedBPP**, **GreenBPP**, and **BlueBPP** represent the color (mono) capabilities of the panel. The number of Bits Per Primary (BPP) is a description of the color depth supported by the panel. The GreenBPP field should be used for monochrome panels such that an 8-bit per pixel monochrome panel should be described as RedBPP:GreenBPP:BlueBPP = 0:8:0. Note that this field describes the capability of the panel and not the color palette or frame rate control capabilities of the graphics controller.

The **ReservedBPP** is a reserved field indicating BitsPerPrimary in the event that color depths increase beyond 24-bit. This is the same method that VBE 1.2 uses to reserve extra information for color depth. This field shall be reserved as 0 until needed.

The **RsvdOffScrnMemSize** is the size of the unavailable Off Screen Memory that is required for flat panel related issues such as dual drive panel frame buffers. This size should be subtracted from the Total Memory installed that is reported by VBE Core functions.

The **RsvdOffScrnMemPtr** is a pointer to the location of the unavailable memory used for the full or half frame buffers.

The remainder of the FPInfo structure is **Reserved** for future use.

5.3.6.5 Subfunction 03h - Panel Shading Options

This subfunction allows an end user the ability to select between a number of different shading options supported by the current video controller. This becomes most useful when displaying color modes on a monochrome flat panel. Examples of such shading may be: NTSC; green gun only; or various combinations of R,G,B weighting. The above shading examples are not required by this specification, this subfunction simply supplies the end user a means of selecting from the OEM supplied options.

An application that may use this function could be a control panel which after querying VBE/FP for the number of options, gives the user a cyclic method to select one of the different gray scale methods. Because gray scale methods are subjective and different OEMs will supply different options, the user would only need to know which option they prefer for their implementation. To keep this function as flexible as possible it was decided that assigning specific names to options should not be done. (This allows OEM1 to supply different options from OEM2.)

Get Request Sequence:

Input:	AH	= 4F	
	AL	= 11h	VBE Flat Panel Extension request
	BL	= 03h	Return/Select Shading Options
	BH	= 00h	Get number of shading options request
Output:	AX	=	VBE Return Status
	CL	=	Number of shading options
	CH	=	Current shading option

Set Request Sequence:

Input:	AH	= 4F	
	AL	= 11h	VBE Flat Panel Extension request
	BL	= 03h	Return/Select Shading Options
	BH	= 01h	Select option request
	CH	=	Shading option number to set
Output:	AX	=	VBE Return Status

Notes:

- 1) Currently undefined registers may be destroyed with the exception of SI,BP,DS,SS,ES and DI)
- 2) This subfunction may only be supported on monochrome panels

5.3.6.6 Subfunction 06h - Vertical & Horizontal Position

This subfunction allows an application the ability to position the displayed portion of a mode. The need for this subfunction arises from the fact that today's flat panel devices are of fixed dimensions, and the displayed portions of some modes are smaller than these fixed dimensions. With the ability to position the displayed portion of a mode, an application can produce a better look.

This subfunction represents a global hardware setting. Other factors within the current operation of the controller may limit whether or not this setting actually occurs (e.g., controller may not allow centering of a 350-line image but will center a 400-line image). The result is that the hardware request for centering is actually set; however its effectiveness may be mode dependent.

VBE/FP Implementations of this function should return actually hardware values that have been set, not mode dependent information.

Set Request Sequence:

Input:	AH	=	4F	
	AL	=	11h	VESA VBE Flat Panel Extension request
	BL	=	06h	Return/Select Vertical and Horizontal Positioning
	BH	=	00h	Return request
Output:	AX	=		VESA VBE Return Status
	BL	=		Available Horizontal Position Settings B0=Left B1=Center B2=Right B3-B7=Reserved, must be 0
	BH	=		Available Vertical Position Settings B0=Top B1=Center B2=Bottom B3-B7=Reserved, must be 0
	CL	=		Current Horizontal Position 0=Left 1=Center 2=Right all other values are reserved
	CH	=		Current Vertical Position 0=Top 1=Center 2=Bottom

			all other values are reserved
Select Request Sequence:			
Input:	AH	= 4F	
	AL	= 11h	VESA VBE Flat Panel Extension request
	BL	= 06h	Return/Select Vertical and Horizontal Positioning
	BH	= 01h	Select request
	CH	=	Vertical Position to set
			0=Top
			1=Center
			2=Bottom
			all other values are reserved
	CL	=	Horizontal Position to set
			0=Left
			1=Center
			2=Right
			all other values are reserved
Output:	AX	=	VESA VBE Return Status

Notes:

- 1). Currently undefined registers may be destroyed with the exception of SI,BP,DS,SS,ES and DI)
- 2). If Vertical Positioning is not available, Current Vertical Position shall return zero (BL=00h), Available Vertical Position Settings shall return zero (BH=00h), and any attempt to set a position shall be ignored. If Horizontal Positioning is not available, Current Horizontal Position shall return zero (CL=00h), Available Horizontal Position Settings shall return zero (CH=00h), and any attempt to set a position shall be ignored.

5.3.6.7 Subfunction 07h - Vertical & Horizontal Expansion

This subfunction allows an application the ability to expand the displayed portion of a mode. The need for this subfunction arises from the fact that today's flat panel devices are of fixed dimensions, and the displayed portions of some modes are smaller than these fixed dimensions. With the ability to expand the displayed portion of a mode, an application can produce a better look.

This subfunction represents a global hardware setting. Other factors within the current operation of the controller may limit whether or not this setting actually occurs. For example, a controller may not allow expansion of a 14-point font, however it will expand a 16-point font. The result is that the hardware request for expansion is actually set; however its effectiveness may be mode dependent.

VBE/FP Implementations of this function should return actually hardware values that have been set, not mode dependent information.

Return Request Sequence:

Input:	AH	=	4F	
	AL	=	11h	VESA VBE Flat Panel Extension request
	BL	=	07h	Return/Select Vertical and Horizontal Expansion
	BH	=	00h	Return request
Output:	AX	=	VESA VBE Return Status	
	BL	=	Available Horizontal Expansion Settings	
			B0=Text Expansion	
			0 = Not Available, 1= Available	
			B1=Graphics Expansion	
			0 = Not Available, 1= Available	
			B2-B7=Reserved, must be 0	
	BH	=	Available Vertical Expansion Settings	
			B0=Text Expansion	
			0 = Not Available, 1= Available	
			B1=Graphics Expansion	
			0 = Not Available, 1= Available	
			B2-B7=Reserved, must be 0	
	CL	=	Current Horizontal Expansion	
			B0=Text Expansion	
			0 = Disabled, 1 = Enabled	
			B1=Graphics Expansion	
			0 = Disabled, 1 = Enabled	
			B2-B7=Reserved, must be 0	
	CH	=	Current Vertical Expansion	
			B0=Text Expansion	
			0 = Disabled, 1 = Enabled	
			B1=Graphics Expansion	
			0 = Disabled, 1 = Enabled	
			B2-B7=Reserved, must be 0	

Select Request Sequence:

Input:	AH	= 4F	
	AL	= 11h	VESA VBE Flat Panel Extension request
	BL	= 07h	Return/Select Vertical and Horizontal Expansion
	BH	= 01h	Select request
	CH	=	Vertical Expansion B0=Text Expansion 0 = Disabled, 1 = Enabled B1=Graphics Expansion 0 = Disabled, 1 = Enabled B2-B7=Reserved, must be 0
	CL	=	Horizontal Expansion B0=Text Expansion 0 = Disabled, 1 = Enabled B1=Graphics Expansion 0 = Disabled, 1 = Enabled B2-B7=Reserved, must be 0
Output:	AX	=	VBE Return Status

Note:

- 1) Currently undefined registers may be destroyed with the exception of SI, BP, DS, SS, ES and DI.
- 2) If Vertical Expansion is not available, Current Vertical Expansion shall return zero (BL=00h), Available Vertical Expansion Settings shall return zero (BH=00h), and any attempt to set vertical expansion shall be ignored. If Horizontal Expansion is not available, Current Horizontal Expansion shall return zero (BL=00h), Available Horizontal Expansion Settings shall return zero (BH=00h), and any attempt to set horizontal expansion shall be ignored.

5.4 BIOS Data Area Assignments

Note: **Bold typeface indicates CGA/MDA/EGA- and VGA-datamap.**
Normal typeface is for system datamap.

0040:0000 WORD	COM1 port base address
0040:0002 WORD	COM2 port base address
0040:0004 WORD	COM3 port base address
0040:0006 WORD	COM4 port base address
0040:0008 WORD	Printer 1 port base address
0040:000A WORD	Printer 2 port base address
0040:000C WORD	Printer 3 port base address
0040:000E WORD	Printer 4 port base address

0040:0010 WORD EQUIPMENT_FLAG

Bit	Definition															
D15, D14	No. of printer adapters															
D13, D12	Reserved															
D11, D10, D9	No. of RS232-C															
D8	Reserved															
D7, D6	No. of diskette drives															
D5, D4	Identify the current primary display device:															
	<table border="0"> <thead> <tr> <th>D5</th> <th>D4</th> <th>Adapter</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>EGA (or none)</td> </tr> <tr> <td>0</td> <td>1</td> <td>CGA 40 X 25</td> </tr> <tr> <td>1</td> <td>0</td> <td>CGA 80 X 25</td> </tr> <tr> <td>1</td> <td>1</td> <td>MDA</td> </tr> </tbody> </table>	D5	D4	Adapter	0	0	EGA (or none)	0	1	CGA 40 X 25	1	0	CGA 80 X 25	1	1	MDA
D5	D4	Adapter														
0	0	EGA (or none)														
0	1	CGA 40 X 25														
1	0	CGA 80 X 25														
1	1	MDA														
D3, D2	Reserved															
D1	Math coprocessor															
D0	IPL diskette															

0040:0012 BYTE	Reserved
0040:0013 WORD	USABLE_RAM Usable memory size in kilobytes
0040:0015 BYTE	Reserved
0040:0016 BYTE	Reserved
0040:0017 WORD	KBD_CNTRL Stores status of special keys
0040:0019 BYTE	ALT_KBD Alternate keypad entry
0040:001A WORD	KBD_BUF_HD Points to head of keyboard buffer
0040:001C WORD	KBD_BUF_TL Points to tail of keyboard buffer
0040:001E 16 WORDS	KBD_BUFFER Circular keyboard buffer
0040:003E BYTE	Diskette drive re-calibrate status
0040:003F BYTE	Diskette drive motor status
0040:0040 BYTE	Diskette drive motor off counter

0040:0041 BYTE		Last diskette driver operation status
0040:0042 7 BYTES		Diskette driver controller status
0040:0049 BYTE	VIDEO_MODE	Current BIOS Video mode
0040:004A WORD	COLUMNS	Number of text columns
0040:004C WORD	PAGE_LENGTH	Length of each page in bytes
0040:004E WORD	START_ADDR	Start Address register value for page
0040:0050 8 WORDS	CURSOR_POS	Cursor positions for all eight pages. The high byte of each word contains the character row, the low byte the column
0040:0060 WORD	CURSOR_TYPE	Start and ending lines for text cursor. High byte has start line.
0040:0062 BYTE	ACTIVE_PAGE	Currently displayed page number
0040:0063 WORD	ADDR_CRTC	I/O port address of 6845/CRTC address register (3B4 monochrome; 3D4 color)
0040:0065 BYTE	CRT_MODE_SET	Current value for Mode Control register (3B8 MDA; 3D8 CGA). The EGA and VGA values emulate the MDA/CGA values
0040:0066 BYTE	CRT_PALETTE	Current value for the CGA color select register (3D9); emulated by EGA/VGA
0040:0067 DWORD		Pointer to MCA PS/2 reset code
0040:006B BYTE		Reserved
0040:006C DWORD		Timer counter
0040:0070 BYTE		Timer overflow
0040:0071 BYTE		Break key state
0040:0072 WORD		RESET flag
0040:0074 BYTE		Last hard disk drive operation status
0040:0075 BYTE		No. of hard disk drives attached
0040:0076 BYTE		PC XT hard disk drive control
0040:0077 BYTE		PC XT hard disk drive controller port
0040:0078 BYTE		Printer 1 Time-out value
0040:0079 BYTE		Printer 2 Time-out value
0040:007A BYTE		Printer 3 Time-out value
0040:007B BYTE		Printer 4 Time-out value
0040:007C BYTE		COM1 Time-out value
0040:007D BYTE		COM2 Time-out value
0040:007E BYTE		COM3 Time-out value
0040:007F BYTE		COM4 Time-out value
0040:0080 WORD		Keyboard Buffer Start Offset pointer
0040:0082 WORD		Keyboard Buffer End Offset pointer
0040:0084 BYTE		ROWSNumber of text rows minus 1

0040:0085 WORD
0040:0087 BYTE

CHAR_HEIGHT
INFO_1

Bytes-per-character

Bit

Description

D7	Equals bit D7 from AL register on most recent mode select. (A one indicates display memory was not cleared by mode select.)
D6, D5	Display memory size (00=N/A, 01=N/A, 10=N/A, 11=256K).
D4	Reserved
D3	A zero indicates EGA is the primary display.
D2	A one will force the BIOS to wait for Vertical Retrace before memory write.
D1	A one indicates that EGA is in Monochrome mode.
D0	A zero means that CGA cursor emulation is enabled. The cursor shape is modified if enhanced text is used.

0040:0088 BYTE

INFO_3

D4-D7	Feature Control bits (from Feature Control register)
D0-D3	EGA Configuration Switch settings

0040:0089 BYTE

FLAGS

Miscellaneous flags

D7	Alphanumeric Scanlines (with bit 4):															
	<table border="0"> <tr> <td>Bit 7</td> <td>Bit 4</td> <td></td> </tr> <tr> <td>0</td> <td>0</td> <td>350-line mode</td> </tr> <tr> <td>0</td> <td>1</td> <td>400-line mode</td> </tr> <tr> <td>1</td> <td>0</td> <td>200-line mode</td> </tr> <tr> <td>1</td> <td>1</td> <td>(reserved)</td> </tr> </table>	Bit 7	Bit 4		0	0	350-line mode	0	1	400-line mode	1	0	200-line mode	1	1	(reserved)
Bit 7	Bit 4															
0	0	350-line mode														
0	1	400-line mode														
1	0	200-line mode														
1	1	(reserved)														
D6	1 – display switching is enabled 0 – display switching is disabled															
D5	Reserved															
D4	(see bit 7)															
D3	1 – default palette loading is disabled 0 – default palette loading is enabled															
D2	1 – using monochrome monitor 0 – using color monitor															
D1	1 – grayscale summing is enabled 0 – grayscale summing is disabled															
D0	1 – VGA active 0 – VGA not active															

0040:008A BYTE		Reserved
0040:008B BYTE		Media control
0040:008C BYTE		Hard disk drive controller status
0040:008D BYTE		Hard disk drive error status
0040:008E BYTE		Hard disk drive interrupt control
0040:008F BYTE		Reserved
0040:0090 BYTE		Drive 0 Media state
0040:0091 BYTE		Drive 1 Media state
0040:0092 BYTE		Reserved
0040:0093 BYTE		Reserved
0040:0094 BYTE		Drive 0 Current cylinder
0040:0095 BYTE		Drive 1 Current cylinder
0040:0096 BYTE		Keyboard mode State and Type flags
0040:0097 BYTE		Keyboard LED Flags
0040:0098 WORD		Address offset to User Wait Complete flag
0040:009A WORD		Segment address to User Wait Complete flag
0040:009C WORD		User wait count – Low word (mseconds)
0040:009E WORD		User wait count – High word (mseconds)
0040:00A0 BYTE		Wait active flag
0040:00A1 BYTE		Reserved
0040:00A2 BYTE		Reserved
0040:00A3 BYTE		Reserved
0040:00A4 BYTE		Reserved
0040:00A5 BYTE		Reserved
0040:00A6 BYTE		Reserved
0040:00A7 BYTE		Reserved
0040:00A8 DWORD	SAVE_PTR	Pointer to BIOS Save Pointer Table

NOTE: The next 84 bytes from 0040:00A1 to 0040:00FF are reserved.

I/O Port Assignment for PC XT and AT Computers

Port Usage for PC XT	I/O Address	Port Usage for AT
DMA Controller	000–01F	DMA Controller, Note 1
Interrupt Controller	020–03F	Interrupt Controller, Note 1
Timer	040–04F	Coprocessor access, Timer
	050–05F	Timer
PPI	060–063	
(system configuration)	060–06F	Keyboard
Reserved	070–07F	Real-time Clock
DMA Page register	080–09F	DMA Page register
NMI Mask register	0A0–0AF	
	0A0–0BF	Interrupt Controller, Note 2
Reserved	0B0–0FF	
	0C0–0DF	DMA Controller, Note 2
	0F0–0FF	Math coprocessor
Unusable	100–13F	Reserved
Unusable	140–14F	Token Ring Adapter, Note 2
Unusable	150–15F	Advanced Color Graphics Display
Unusable	160–16F	Advanced Monochrome Graphics Display
Unusable	170–177	Fixed-disk Adapter, Note 2
Unusable	1C0–1CF	Token Ring Adapter, Note 1
Unusable	1E8–1EF	Streaming Tape Drive Adapter
Unusable	1F0–1F7	Fixed-Disk Adapter, Note 1
Unusable	1F8–1FF	Reserved
Game I/O	200–20F	Game I/O
Expansion Unit	210–217	
Multifunction Card,		
Note 1	218–21F	Multifunction Card, Note 1
Reserved	220–24F	
	278–27F	Parallel port 2
	2B0	GD5462 Host Control register for VESA VL-Bus Clock Calendar, Clock Calendar, Note 1
Note 1	2C0–2CF	3278/79 Emulation Adapter, Clock/calendar, Note 1
	2D0–2DF	
Serial port 4, Note 1	2E0–2E7	
Serial port 3 or 4,		
Note 1	2E8–2EF	
Reserved	2F0–2F7	Interrupt Sharing
Serial port 2	2F8–2FF	Serial port 2
Prototype Card	300–31F	Prototype Card
Fixed Disk	320–32F	
	360–36F	PC Network
Parallel port 1	378–37F	Parallel port 1
SDLC	380–38F	SDLC, Bisync 2
Bisync	3A0–3AF	Bisync 1

6 Programmer's Reference



6.1 Overview

This section of the Technical Reference Manual provides information of particular interest to programmers of the SM3110 graphics chip.

A key component of programming for the SM3110 is the ability to access all of the registers, FIFO and data areas of the graphics hardware and memory regions. This section begins with by detailing the addressing spaces and access modes of the SM3110.

Following the address space introduction are the subsections describing the setup and use of the graphics chip itself. The topics presented are arranged from initialization by the BIOS and driver, through the 2D and 3D functions. Then the remaining capabilities of the device are described. Finally, some equates and macros are defined that make the coding easier.

6.2 Address Mapping

Programming for the SM3110 requires access to four(4) address spaces:

- (1) VGA addresses for direct I/O port access at 3CxH, 3BxH, 3DxH
- (2) VGA addresses for VGA frame buffer access at A0000H (128KB);
(larger memory for Super VGA frame buffer data is made accessible via an I/O addressed bank/offset register)
- (3) PCI configuration space (256B)
- (4) SM3110 "Local Memory" addresses, encompassing all the embedded memory and memory-mapped registers (128MB)

In addition to having their standard addresses, address spaces (1) and (3) have memory-mapped addresses within the SM3110 Local Memory range.

6.2.1 Local Memory Address Space

The term "local memory" refers to the 32MB linear address space within which the SM3110 maps all of its available embedded memory, as well as its internal registers and memory-mappings of VGA and PCI registers. In order to support all variations of little-endian/big-endian and word/double word byte-swapping architectures, the SM3110 requires a total address space of 128MB (out of the 4GB available) address space. The four(4) individual 32MB address spaces are multiply-mapped to the one 32MB local memory area required. This is shown in Figure 6-1.

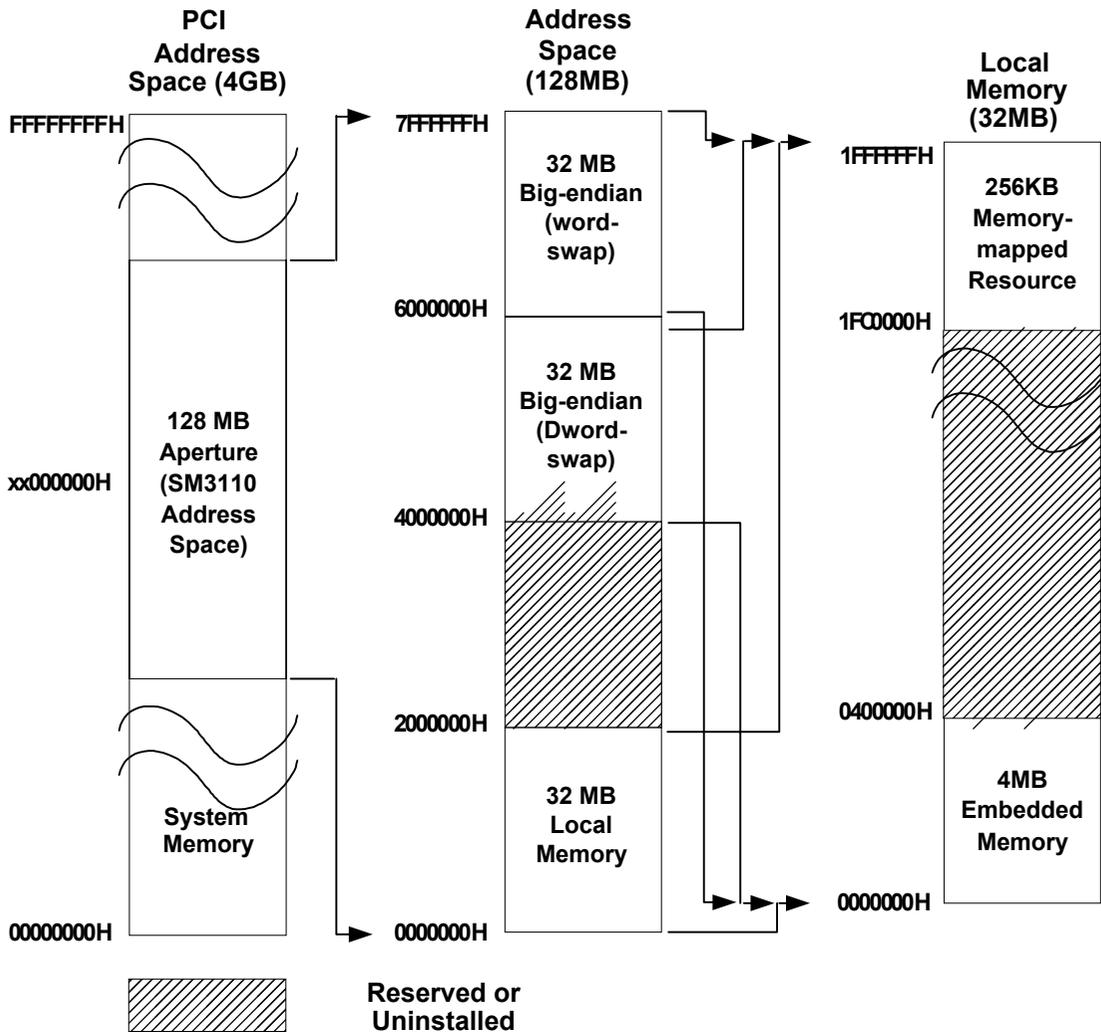


Figure 6-1. SM3110 Address Space Allocation

Range	Size	Little Endian Base	Little Endian (Reserved) Base	Big Endian (Dword Swap) Base	Big Endian (word Swap) Base
Display Memory	31.75MB	00000000H	02000000H	04000000H	06000000H
Memory-mapped Resources	0.25MB	01FC0000H	03FC0000H	05FC0000H	07FC0000H
Total	32.00MB				

Table 6-1. Local Memory Space Allocation

Also shown in Figure 6-1 is a further division of the 32MB local memory addresses into two areas:

- (1) 256KB at the top of the address range, for memory-mapping SM3110 and other system resources
- (2) 31.75MB at the bottom of the address range, for addressing the SM3110 display memory (of which of 4MB is used for embedded memory on the SM3110)

The embedded 4MB display memory is dynamically partitioned by the software to provide data buffers, frame buffers, FIFO's, etc. These divisions and areas are discussed later in this section.

6.2.2 Memory-Mapped Resources Address Space

The 256KB region (at the top of local memory) for memory-mapped resources is further divided into four areas:

- (1) Control addresses (64KB)
- (2) Peripheral I/O addresses (64KB)
- (3) Reserved (64KB)
- (4) Embedded memory configuration registers

as shown in Figure 6-2.

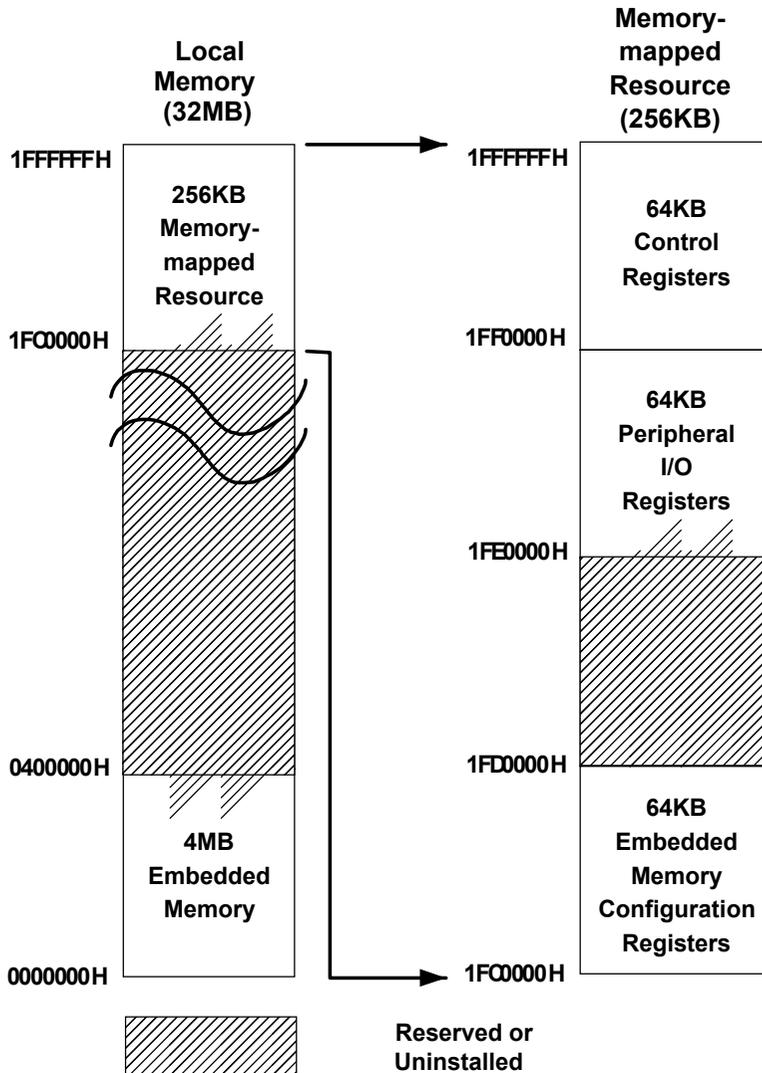


Figure 6-2. Memory-mapped Resources

6.2.3 Control Address Space Memory Mapping

The 64KB Control Address space is divided into five regions as shown in Figure 6-3. Each of these regions is discussed briefly in the following sections. The registers which access these regions are defined in Section 4, Register Summary.

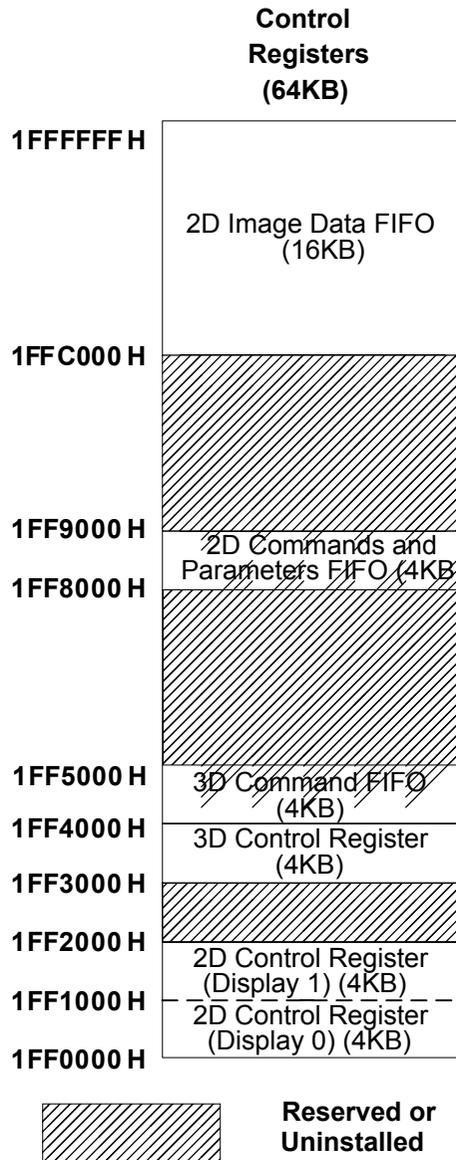


Figure 6-3. Control Address Space

6.2.3.1 2D Control Registers (Display 0/Display 1)

The 2D control registers occupy two 4KB (8KB total) memory-mapped address spaces. Registers that are common to both Display 0 and Display 1 are at the same offset within their respective 4KB address spaces. The layout of this region is shown in Figure 6-4 and described in detail in Section 4, Register Summary.

+F00H	PCI Config		
+E00H	LCD Timing	CRT Timing	
+D00H	LCD Control	CRT Control	
+C00H	Buffer Descriptors		
+B00H	Channel Descriptors		
+A00H	Surface Descriptors		
+900H	Ext Mem Control		
+800H	2D Engine		
+500H	LCD Panel Control		
+400H	Peripheral I/O		
+200H	Host Bus Master		
+000H	Status/Control		Status/Control
	+0xxxH		+1xxxH

Figure 6-4. 2D Control Register Space

6.2.3.2 3D Control Registers

The 3D control registers occupy a 4KB memory-mapped address space. The layout of this region is shown in Figure 6-5 and described in detail in Section 4, Register Summary.

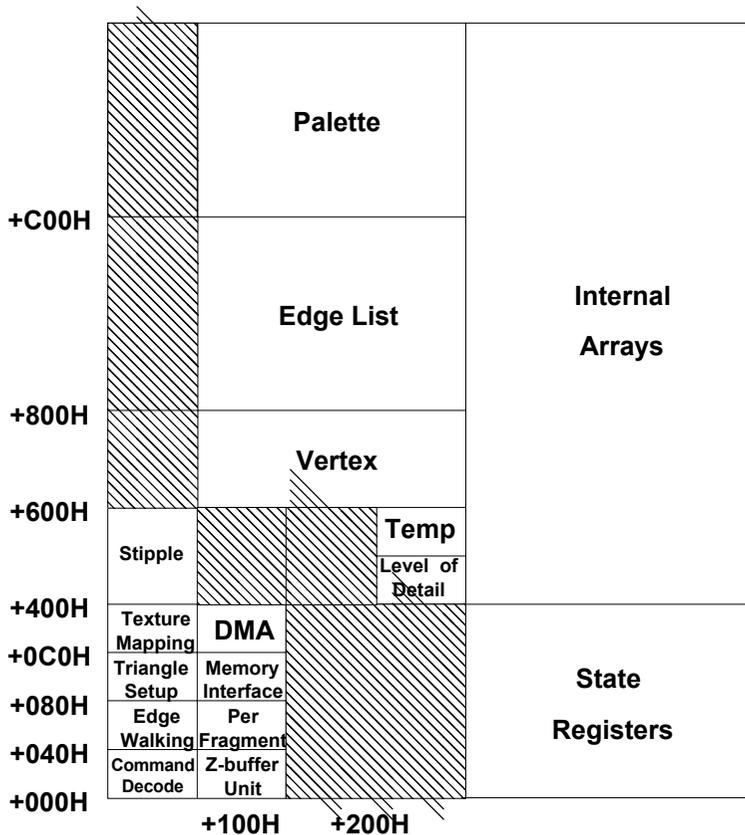


Figure 6-5. 3D Control Register Space

Access to 3D state registers is provided through a 4KB memory-mapped address space. Each register is directly accessible (via PCI) with byte granularity. State registers loaded via the command/parameter stream are, however, only writable with a granularity of 16 bits, i.e., all bits within the 16-bit word are modified. For this reason, control fields are grouped so that fields that are typically modified together are located in the same or nearby words. For PCI access, data must be correctly byte-aligned. These registers can be read and written directly for diagnostic and debug purposes.

6.2.3.3 3D Command/Parameter FIFO

A 4KB input (write-only) FIFO port is reserved for command and parameter transfers from the host to the 3D rendering engine. All accesses to anywhere in the window will be written to the location pointed to by the FIFO write pointer. The actual memory region reserved for the FIFO is configurable in 16KB increments up to 1MB on 16KB boundaries. It can be located at any addressable memory region within the 32MB local address space (see 3D Command FIFO)

6.2.3.4 2D Command/Parameter FIFO

A 4KB input (write-only) FIFO port is reserved for command and parameter transfers from the host to the 2D rendering engine. All accesses to anywhere in the window will be written to the location pointed to by the FIFO write pointer. The actual memory region reserved for the FIFO can be configurable to 1KB, 8KB, 16KB, or 32KB in size. It can be located at any addressable memory region addressable within the 32MB local address space.

6.2.3.5 2D Image Data FIFO

A 16KB input (write-only) FIFO port is reserved for 2D image data transfers from the host. All accesses to anywhere in the window will be written to the location pointed to by the FIFO write pointer. The actual memory region reserved for the FIFO can be configurable to 1KB, 8KB, 16KB, or 32KB in size. It can be located at any addressable memory region within the 32MB local address space.

6.2.4 Display Memory Arrays

The following arrays are located in display (internal/embedded DRAM) memory:

6.2.4.1 Surfaces

All 2D operations operate on surfaces defined by surface descriptor registers that specify the base, stride and pixel format of the specific array.

6.2.4.2 Texture Memory

Texture memory consists of multiple bitmaps containing different views of textures to be applied to objects. A base and end register in increments of 4KB define the region that may be used for texture memory.

6.2.4.3 Z-Buffers

The Z-buffer is typically a 16-bit bitmap the same size as the front and back buffers. It is initialized to a constant value representing the largest (farthest) position from the viewer. It is accessed sequentially in short horizontal spans just prior to rendering the span. If the Z value of the pixel being rendered is less than the stored Z value, the pixel is in front of the previously rendered pixel and it is written to the back buffer. The Z-buffer is then updated with the new Z value. The most frequent Z-buffer accesses are short sequential reads followed by short sequential writes (masked by the byte enable).

6.2.4.4 Memory Buffer Allocation

Memory for textures, draw and Z-buffer are stored within the same logical address space. The entire address space, however, need not be contiguous, i.e., it may be composed of several different regions.

The following restrictions apply to the allocation of buffers:

- The command/parameter/data FIFO must be in display memory. This buffer's size is programmable up to one megabyte.
- The Z-buffer must be contiguous.
- Each of the front and back buffers must be contiguous within a region.
- Textures must be in display memory.

6.2.5 Byte Ordering

As previously discussed, the SM3110 supports both little- and big-endian accesses to the entire linear address space, determined by the most significant address bit ([26]). In big-endian mode, two different byte orders are supported, selected by the 2nd most significant bit ([25]) of the address. Because both data registers and memory-mapped enhanced control registers are accessible in this linear address space, it is possible to access both memory and registers in either byte ordering by using different addresses, even on an individual doubleword basis.

A[26:25]	Endian	Swapping	Processor
00	little endian	Bytes within DWORDs	x86, Alpha
01	<i>Reserved</i>		
10	big endian	Bytes within DWORDs	PowerPC: 8, 32bpp
11	big endian	Bytes within WORDs	PowerPC: 16bpp

Table 6-2. Byte Order Selection

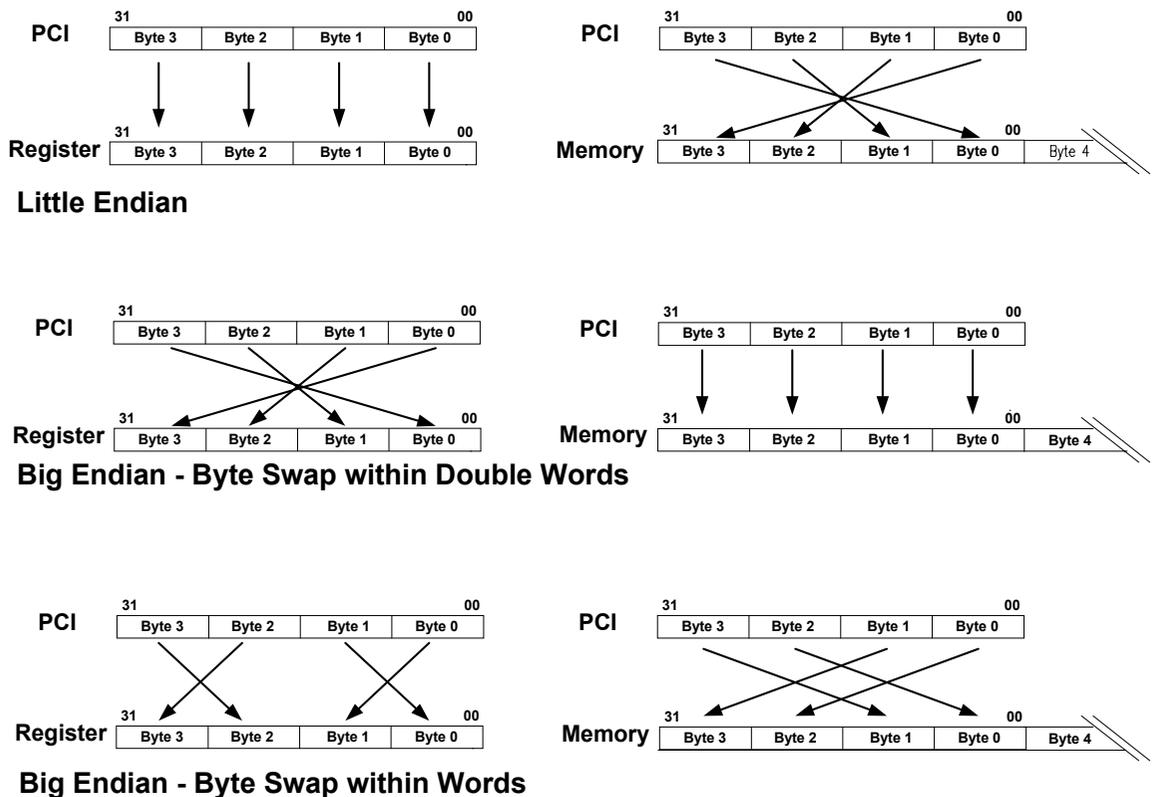


Figure 6-5. Byte Ordering

6.3 BIOS Functions

VGA BIOS (Basic Input and Output Service) provides many low level services to higher level programs. The interface is defined by IBM and VESA committee. Section 5, BIOS Specification, of this document describes in detail all the supported functions. It also defines all the modes supported by the SM3110. This section describes how the BIOS programs the SM3110 in order to achieve those tasks defined in the BIOS Specification.

Among all the services provided by the BIOS, the two most important are initialization and mode setting. The following sections describe the exact sequence the BIOS performs to achieve these two tasks.

6.3.1 BIOS Initialization: Power-On Self Test (POST)

Prior to using any of the functions of the SM3110, several initialization steps are required. These actions are performed as part of the Power-On Self Test (POST) of the BIOS.

POST will be executed during cold boot or warm boot. During cold boot, a hardware-reset signal connected to the chip will be toggled, followed by a call into POST. For warm boot (when the user hits Ctrl-Alt-Del), POST is the only function that gets called. Therefore, POST has the responsibility to initialize the chip back to its reset state. The pseudocode below shows how the BIOS initializes the hardware during the POST function.

```

0x3C3 ? 1 // Turn on video sub-system

0x3CE _ 0x80 // Max locking register
0x3CF _ 0x53 // 'S' - first unlock code
0x3CF _ 0x4D // 'M' - second unlock code, unlock
extended registers

Program 0x80 to 0x82 // Memory clock

// Initialize following registers
0xF1 _ 0 // Disable all interrupts
0x2F1 _ 0 // Disable host DMA interrupt
0x224 _ 0 // Clear host DMA count
0x8F1 _ 0x80 // Reset 2D engine
0x910 _ 0x37 // Internal refresh rate divide ratio
0xDF5 _ 0 // Disable double scan for display 0
0x1DF5 _ 0 // Disable double scan for display 1
0xDF7 _ 0 // Set DAC 0 to normal
0x1DF7 _ 0 // Set DAC 1 to normal
0xDF4 _ 0 // Disable ECRT 0 - set to VGA
0x1DF4 _ 0 // Disable ECRT 1
0xD00 _ 0 // Disable cursor channel 0
0x1D00 _ 0 // Disable cursor channel 1
0xD10 _ 0 // Disable icon channel
0xD20 _ 0 // Disable scaler channel 0
0x1D20 _ 0 // Disable scaler channel 1
0xD30 _ 0 // Disable display channel 0
0x1D30 _ 0 // Disable display channel 1
    
```

```
0xCF0_0 // Disable deferred control 0
0xCF4_0 // Disable deferred control 1
0xF04_7 // PCI command register

// Initialize LCD registers
Disable stretching - 0x504
Power management Control - 0x508
LCD type and misc. - 0x 510
Dithering and frame rate modulation - 0x514
DSTN starting address - 520

Call set mode routine to set to mode 3

Detect CRT // Check whether monitor is attached

Turn on 0x3C4 index 1 bit 5 // Turn off screen

Memory test // Also detect memory size

Test DAC

Exit
```

6.3.2 Set Mode

After POST, setting mode becomes the most essential portion of BIOS. Its responsibility is to set the hardware to a known working state. The pseudocode below shows how the BIOS handles this function.

```

0xDF4 _ 0 // Turn off ECRTC, enable VGA CRTIC
0xDF7 _ 0 // Set DAC to normal
3C4.1 bit 5 _ 1 // Turn off screen
If(Standard VGA Mode)
{
    Set standard VGA registers - 3C4/5, 3D4/5, 3C0/1, 3CE/F, etc. according to
    the mode
}
Program 3C8, 3C9 // Initialize palette if necessary

// Reset following registers
0xD00 _ 0 // Disable cursor channel 0
0xD10 _ 0 // Disable icon channel
0xD20 _ 0 // Disable scaler channel 0
0xD30 _ 0 // Disable display channel 0
0x8F1 _ 0x80 // Reset 2D engine
If(Set to enhanced mode)
{
    // Program following registers
    0xB34 _ 4 // Set display 0 channel to surface 4
    0xB30 _ 0 // Surface offset for display 0 channel
    0xE30 _ 0 // Display 0 view port start
    0xA40 _ 0 // Surface 4 base address
    0xD80 _ 1 // scaler and display priority
    0xD90 _ 0 // Disable color key
    0xDF8 _ 0 // Clear display FIFO status
    0xCF0 _ 0 // Disable deferred 0
    0xCF4 _ 0 // Disable deferred 1
    0xEE0 _ 0 // Display 0 sync selector
    Program E80 - E94 // Timing parameters for ECRTC 0
    Program 0xE34 // View port end
    Program 0xA44 // Surface 4 stride and format
    Program 0xD30 // Display 0 format
    Program 0xDF5 // Double scan control
    0xDF4 _ 5 // Turn on ECRTC
}
Program 0x90, 0x91, 0x92 // Dot clock

Clear screen if needed

0x3C4.1 bit 5 _ 0 // Turn on screen

Exit
    
```

6.4 Driver Functions

The lowest-level driver for the SM3110 is responsible for these functions:

- device detection
- configuration
- memory-mapped control addressing
- memory address allocation
- physical buffers for FIFO's
- linear/segmented mode
- cursor control
- icon programming

Each of these topics is presented in this section.

6.4.1 Device detection

This PCI interrupt routine is called to detect the SM3110 as a PCI/AGP device:

Sample code : Sense presence of SM3110

```

SenseHardware    proc    near
    mov     si, 0                ; Search from 0
SH_FindPCIDevice:
    mov     cx, DEVICEID        ; =4831h, Device ID for SM3110 board
    mov     dx, VENDORID        ; =8888h, Vendor ID for Silicon Magic
    mov     ax, 0B102h          ; Find device command
    int     1Ah                 ; PCI interrupt
    jnc     SH_Found            ; Found

    cmp     ah, 86h             ; Error code in ah - not found
    je     SH_NotSiMagic        ; Search all slots and not found

SH_FindNextSlot:
    inc     si                  ; Search next slot
    cmp     si, 256
    jb     SH_FindPCIDevice

SH_NotSiMagic    ; Search all slots and not found
    xor     ax, ax              ; means fail
    ret

SH_Found:
    mov     wPCIBusAddress, bx  ; bx=bus/device number, save in global
    mov     ax, 1               ; means successful
    ret
SenseHardware    endp

```

6.4.2 Device configuration

Once the SM3110 device is detected, the following routine is used to read the PCI/AGP configuration registers.

Sample code: Read PCI/AGP Configuration register

```
ReadPCIDword    proc    near
;;Entry:        ax=index
;;Return:       eax=double word be read
    mov    bx, wPCIBusAddress    ; get from SenseHardware sub-routine
    mov    di, ax                ; index
    mov    ax, 0B10Ah            ; AH:PCI function ID,AL:Read a dword
    int    1Ah                  ; PCI interrupt, return in ECX
    mov    eax,ecx              ; return in EAX
    ret
ReadPCIDword    endp
```

6.4.3 Memory-mapped control address

The last 64KB of the local memory address space is reserved for memory-mapped 2D/3D control registers.

6.4.3.1 Enable/Disable access

To prevent inadvertent access to the enhanced memory-mapped control registers, a specific sequence must be written through the VGA Graphics Controller index/data port to enable access.

Sample code: Enable/Disable memory-mapped control register access

```
EnableIOMap      proc    near
    mov     dx,3ceh
    mov     ax,5380h      ;write S into reg 80h
    out    dx,ax
    mov     ax,4D80h      ;write M into reg 80h
    out    dx,ax
    ret
EnableIOMap      endp

DisableIOMap     proc    near
    mov     dx,3ceh
    mov     ax,5280h      ;write L into reg 80h
    out    dx,ax
    ret
DisableIOMap     endp
```

Since PCI devices will be allocated with different memory addresses, the enable code is programmed just once for insurance, and the disable code should never be used.

6.4.3.2 2D Command Port selector

Use the same way as above to set a variable to the base address of the memory-mapped control register region. This is also the base address of the 2D control region.

Sample code: Get the 2D command port selector

```
GetPointer_MemoryMapCtl proc      near
    call  GetLocalMemoryBase
    mov   eax, PhyVRAMAddress
    add   eax, 1FF0000             ;32M-64K
    mov   PhyCmdPortAddress,eax   ;save as global
    mov   edx, 10000h             ;size=64K
    call  GetSelector
    mov   CmdPortSelector, ax     ;save as global variable
    ret
GetPointer_MemoryMapCtl endp
```

After calling `GetPointer_MemoryMapCtl` routine during initialization, the following macros can be used to read/write memory-mapped control registers at any offset.

```
ReadMemoryMapCtl Macro offset,data
    mov   es, CmdPortSelector
    mov   di,offset
    mov   data,es:[di]
Endm
```

```
WriteMemoryMapCtl Macro offset,data
    mov   es, CmdPortSelector
    mov   di,offset
    mov   es:[di], data
Endm
```

6.4.3.3 2D Image Data Port selector

The 2D Image Data FIFO is located at C000H from the memory mapped control address. Instead of using the 2D Command port selector with offset C000H, it is easier to understand allocating another global variable (selector) to address the area directly.

Sample code: Get the 2D image data FIFO port selector

```
GetPointer_ImagePort    proc    near
    mov     eax, PhyCmdPortAddress
    add     eax, 0C000h          ;last 16K of command port
    mov     edx, 4000h          ;size=16K

    call    GetSelector
    mov     ImagePortSelector, ax      ;save as global variable
    ret
GetPointer_ImagePort    endp
```

After calling `GetPointer_ImagePort` routine at initialization period, the following macros can be used to write data to 2D image data FIFO.

```
WriteImagePort    Macro offset,data
    mov     es, ImagePortSelector
    mov     di,offset
    mov     es:[di],data
endm
```

6.4.3.4 VGA I/O Ports

Program the VGA I/O port (3Cxh, 3Bxh or 3Dxh) by writing index/data pairs.

Sample Code: Program VGA I/O port

```
VGAIOport    proc    near
    mov     dx, port          ; 3Cxh, 3Bxh or 3Dxh
    mov     al, index
    out     dx, al
    inc     dx
    mov     al, data
    out     dx, al
    ret
VGAIOport    endp
```

6.4.3.5 A0000H-based 128K VGA memory

For 16-bit real mode, use A0000H as the segment. A0000H:0 points to screen at (0,0) and A0000H:C30H points to pixel at (48,3) for 1024x768x256colors mode.

For 16-bit protected mode, allocate a selector, set its base to A0000H and set its limit to dwVRAM-Size. Then use the selector:offset pair to access video memory (see also the following section, Local Memory Address Space).

6.4.3.6 PCI configuration space

In addition to calling subroutine `ReadPCIDword`, the PCI configuration register can also be accessed from the offset 0F00H to the memory-mapped control selector. For example,

```
mov    ax, 30h
call   ReadPCIDword           ;return in eax
```

has the same effect as

```
mov    fs, CmdPortSelector
mov    eax, fs:[0F30h]
```

6.4.4 Memory Address Allocation

Section 6.2 described the address space of the SM3110 and all of its subdivisions. This section describes how each of these is accessed by software.

6.4.4.1 Local memory address space

The SM3110 Address Space occupies four contiguous 32MB address ranges (of the total 4GB PCI address space). The Local Memory Address Space is the lower 32MB of the 128MB range. The PCI address of the local memory address space is located in the BASE 0 index of the PCI configuration register. Use the following routine to get this physical address.

Sample code: Get the physical address of the base of local memory.

```
GetLocalMemoryBase    proc    near
    mov    ax, 10h                ;Base 0 physical address index
    call   ReadPCIDword
    and    eax,0FF000000h         ;align
    mov    PhyVRAMAddress,eax    ;save in global,will be used later
    ret
GetLocalMemoryBase    endp
```

6.4.4.2 Display (video) memory address

The display (video) memory starts from the beginning of the local memory address, with a physical memory size which can be set in a global variable by the following routine.

Sample code: Get display memory size

```
GetVideoMemSize  proc  near
    xor     eax,eax
    mov     dx,3ceh
    mov     al,8fh                ;BIOS save bank size during POST
    out     dx,al
    inc     dx
    in      al,dx                ;al has size in 64k units
    shl     eax,16                ;size in eax
    mov     dwVRAMSize,eax       ;save in global, will be used later
    ret
GetVideoMemSize  endp
```

To access the local memory, the physical address needs to be converted to a linear/logical address:

Sample code: Convert Physical to Linear Address

```
ConvertPhysicalToLinearAddress proc                                near
;;Entry:  eax=physical address, edx=size
;;Return: eax=linear address
    shld   ebx,eax,16
    mov    cx,ax                ;BX:CX has base physical address
    mov    edi,edx
    dec    edi
    shld   esi,edi,16           ;SI:DI has limit
    mov    ax,0800h             ;convert physical to linear address.
    Int    31h                  ;DOS Protected Mode Interface (DPMI)
                                call, return BX:CX = linear address

    shrd   eax,ebx,16
    mov    ax,cx                ;eax= linear address
    ret
ConvertPhysicalToLinearAddress endp
```

For 16-bit programs, data is pointed to by segment:offset pair (in real mode) or by selector:offset pair (in protected mode).

6.4.4.3 16-bit Real Mode Addressing

In 16-bit real mode (i.e., DOS environment), data is pointed to by segment:offset pair. For example, to access memory at B8004h, program as follows:

Sample code: Read/Write data to segment:offset in 16-bit real mode

```

mov     ax,0B000h                ;segment or B800h
mov     es, ax
mov     di, 8004h                ;offset or 4h

```

Then use es:[di] to read/write data.

6.4.4.4 16-bit Protected Mode Addressing

In 16-bit protected mode (i.e., Microsoft Windows environment), data is pointed to by selector:offset pair. The segment needs to be converted to a selector by DOS Protected Mode Interface (DPMI) calls.

Sample code: Get a selector in 16-bit protected mode

```

GetSelector     proc  near
;; Entry:      eax= dwPhysAddr,  edx=dwSize
;; Return:     ax=selector assigned
    mov     dwSize, edx          ;save in stack or global variable
    mov     dwPhysAddr,eax      ;save in stack or global variable

    xor     ax,ax
    mov     cx,1                ;count
    int     31h                 ;DPMI, allocate an LDT selector in ax
    mov     wSelecotr,ax        ;save selector in stack or global variable

    mov     eax, dwPhysAddr
    mov     edx, dwSize
    call    ConvertPhysicalToLinearAddress    ;return eax=linear addr

    shld   ecx,eax,16
    mov    dx,ax                ;CX:DX = linear address
    mov    bx,wSelector        ;BX=selector
    mov    ax,7
    int    31h                 ;DPMI, set selector base.

    mov    edx,dwSize
    dec   edx
    shld   ecx,edx,16          ;CX:DX = limit
    mov    ax,8
    int    31h                 ;DPMI, set selector limit

    mov    ax, wSelector      ;return value in ax
    ret

GetSelector     endp

```

6.4.4.5 32-bit Mode Addressing

For 32-bit code, since all memory is within the 4GB logical space, the linear address points to the proper location of the memory.

By combining with 16-bit protected code, use the following routine to get display (video) memory pointer:

Sample code: Get the Video Memory pointer

```
GetPointer_VideoMemory proc near
    call    GetLocalMemoryBase
    call    GetVideoMemSize
    mov     eax, PhyVRAMAddress
    mov     edx, dwVRAMSize
#ifdef USE_32BIT
    call    ConvertPhysicalToLinearAddress
    mov     dwVRAMLinearAddress,eax           ;save as global variable
else ;;USE_16BIT protected mode
    call    GetSelector
    mov     wVRAMSelector,ax                ;save as global variable
#endif
    ret
GetPointer_VideoMemory endp
```

After calling the `GetPointer_VideoMemory` routine at initialization time, the following macros can be used to read/write display (video) memory:

Sample code: Read/Write Video Memory

```
IFDEF USE_32BIT

ReadVideoMemory      Macro      offset,data
    mov    edi,dwLinearAddress_VideoMemory
    add    edi,offset
    mov    data,[edi]
Endm

WriteVideoMemory     Macro      offset,data
    mov    edi,dwLinearAddress_VideoMemory
    add    edi,offset
    mov    [edi],data
Endm

Else                  ;;USE_16BIT protected mode

ReadVideoMemory      Macro      offset,data
    mov    es,wVRAMSelector
    mov    di,offset
    mov    data,es:[di]
Endm

WriteVideoMemory     Macro      offset,data
    mov    es,wVRAMSelector
    mov    di,offset
    mov    es:[di],data
Endm

ENDIF
```

Note: For simplicity, only 16-bit protected mode code will be used for the remainder of this document.

6.4.5 Physical Buffers for FIFOs

When programming or addressing data in the 3D command, 2D command and 2D image data FIFO areas, the SM3110 chip hardware will first decode the address and transfer data to the physically allocated buffer and then execute the command or retrieve data in a FIFO order. These buffers must be physically allocated somewhere in the 4MB display memory. By convention: use the last 128KB for these buffers

- 3D command FIFO buffer: 64KB
- 2D image data FIFO buffer: 32KB
- 2D command FIFO buffer: 4KB
- Cursor Deferred control buffer: 1KB
- Display Deferred control buffer: 1KB
- Cursor buffer: 2KB
- Icon: 2KB
- others

For example, consider the 2D command FIFO. The 2D command FIFO can only be accessed within a 4KB range: offsets +8000H to +9000H within the Control Address Space. However, the SM3110 can actually access up to 32KB by using addresses in this range repeatedly. The following code shows how this is done (in this example only 16KB is accessed).

Sample code: Set buffer locations for FIFOs

```
GetFIFOBufferLocation  proc  near
    call  GetLocalMemoryBase
    call  GetVideoMemSize
    mov   eax, PhyVRAMAddress
    add   eax, dwVRAMSize
    sub   eax, 20000h           ;reserved last 128K for FIFO buffer
    mov   dwPrivatePhyAddr, eax;save as global
    ret
GetFIFOBufferLocation  endp
```

Please refer to “Equates” section. Here is code to set buffer addresses:

```
SetBufferBase_2DCmdFIFO  proc  near
    call  GetFIFOBufferLocation
    add   eax, CMDFIFO_OFFSET
    or    eax, ECMDFIFOSIZE
    mov   fs, CmdPortSelector
    mov   fs:[CMDBufferBaseAddress],eax
    ret
SetBufferBase_2DCmdFIFO  endp

SetBufferBase_2DImageFIFO  proc  near
    call  GetFIFOBufferLocation
    add   eax, IMGFIFO_OFFSET
    or    eax, EIMGFIFOSIZE
    mov   fs, CmdPortSelector
    mov   fs:[IMGBufferBaseAddress],eax
    ret
SetBufferBase_2DImageFIFO  endp
```

6.4.6 Linear/Segmented mode

Display memory can be configured as a large contiguous address space (linear mode) up to 32MB, or as multiple 64KB segments (segmented mode). For linear mode, it is easy to access the memory by 32-bit linear pointer or selector:[32-bit offset]. Usually in 16-bit programs with 16-bit offsets such as BX,SI or DI, the maximum value is 64KB. To access memory beyond 64KB requires setting the memory bank.

Sample Code: Program VGA Memory bank

```

GetVGAMemoryBank proc    near
    mov     dx, 3CEh
    mov     al, 84h        ;bank offset register index
    out     dx, al
    inc     dx
    in      al, dx        ;bit 6:1=bank in 64K unit
    and     ax,7Fh        ;bit 7=1 for packed pixel mode banking mode
    shr     ax,1
    ret                                ;return ax=bank #
GetVGAMemoryBank endp

SetVGAMemoryBank proc    near    ;ax=bank #
    add     al, al        ;bit 6:1=bank in 64K unit
    or      al, 80h        ;bit 7=1 for packed pixel mode banking mode
    mov     dx, 3CEh
    mov     ah, 84h        ;bank offset register index
    out     dx, ax
    ret
SetVGAMemoryBank endp

```

Thus, after programming

```

mov     ax,1
call    SetVGAMemoryBank

```

wVRAMSelector:[0] points to the pixel at (0,64) instead of (0,0) in 1024x768x256color mode. Obviously, accessing memory in linear mode is easier, simpler and recommended.

6.4.7 Cursor control

The SM3110 supports both monochrome and color cursors. The cursors can be turned on and off. The turning on and the positioning of the cursor can be synchronized with VBLANK.

6.4.7.1 Cursor Format

The monochrome cursor can be up to 256 x 256 pixels in size. It is defined as two 1-bit-per-pixel (bpp) masks: the AND mask and the XOR mask. The cursor display logic is described below.

AND Mask XOR Mask Result

AND Mask	XOR Mask	Result
0	0	Background Color (BLACK)
0	1	Foreground Color (WHITE)
1	0	Screen Pixel
1	1	NOT Screen Pixel

The color cursor can be up to 32 x 32 pixels. It is defined as a one-bit-per-pixel AND mask and a 16-bit-per-pixel color bitmap which is the XOR bitmap. The cursor display logic is described below.

AND Mask XOR Bitmap Result

AND Mask	XOR Bitmap	Result
0	Cursor Pixel	Cursor Pixel
1	Cursor Pixel	Screen Pixel XOR Cursor Pixel

6.4.7.2 Cursor data buffer and surface

The cursor pattern is stored in the offscreen memory. Cursor buffer allocation is mentioned in Section 6.4.5, Physical Buffers for FIFOs. A surface descriptor has the physical base address of the cursor in the offscreen memory. This example shows how to prepare buffer before setting cursor.

Sample code: Set buffer location for cursor and deferred control

```
;fs= CmdPortSelector: selector points to the register base address
SetBuffer_Cursor proc near
    mov     eax, dwPrivatePhyAddr      ;get from GetFIFOBufferLocation
    addeax, CURSOR_OFFSET
    sub     eax, PhyVRAMAddress       ;relative to local memory base
    mov     dwCursorBufferOffset,eax  offscreen to load the cursor pattern,
                                     save as global
    mov     fs:[CURSOR0SURFACE],eax   ;surface descriptor base
    mov     fs:[ChCurOffsets],0      ;offsets into cursor surface, maybe !=0
                                     if clipped
    mov     fs:[ChCurSIndex],CURSOR0SURFACEINDEX ;set cursor surface
                                     descriptor index
    ret
SetBuffer_Cursor endp
```

6.4.7.3 Set Monochrome Cursor data

For monochrome cursor, the cursor source (i.e., from Windows) is specified as two 1 BPP bitmaps, the AND and XOR masks. SM3110 defines the cursor surface as 1 BPP format, and sets the stride to 2 times the cursor width. These two masks are written to the cursor surface buffer as interleaved words with the AND mask followed by the XOR mask.

Following example sets a monochrome cursor with size of 256 x 256 (the maximum possible size). Note that the cursor surface stride is fixed at 256 pixels (2 BPP).

Sample code: Set Monochrome cursor

```

;fs= CmdPortSelector: selector points to the register base address
;ds:[esi] points to the cursor pattern, 512 bytes of AND mask followed by 512
                        bytes of XOR.

SetMonochromeCursor    proc    near
    mov     es, wVRAMSelector
    mov     edi, dwCursorBufferOffset        ;es:[edi] points to the offscreen
                                                to load the cursor pattern.

    mov     fs:[CursorFormat],CURSOROFF      ;turn off cursor
    mov     gdwCursorFormat, CURSOR1        ;global variable to turn on cur-
                                                sor later

    mov     eax,(256*2)                      ;stride*2 - only for mono cursor
    or     eax,BPP1 shl 20;surface is 1 BPP
    mov     fs:[CURSOR0SURFACE+4],eax       ;set stride and format

    mov     ecx,64                          ;count of rows to handle
loop_nextrow:
    mov     bx,4
loop_inlrow:
    mov     ax,ds:[esi+512];load XOR mask
    shl     eax,16                          ;transfer to high bits
    loadw   ;load AND mask ls 16 bits
    stosd  ;save AND & XOR masks in off
                                                screen memory

    dec     bx
    jnz    loop_inlrow
    add     edi,256*2/8 - 16                 ;offset into next row
    loop   loop_nextrow
    ret
SetMonochromeCursor    endp

```

Note: The cursor is now set and needs to be turned on and positioned. The monochrome cursor can be a maximum of 256 x 256 pixels. If the given cursor size is smaller, the cursor pattern is loaded into the top left corner of the cursor surface and the viewport adjusted to display the cursor. The CursorPositionStarts and CursorPositionEnds are set based on the cursor width and height; the desired portion of the cursor surface is displayed.

6.4.7.4 Set Color Cursor data

For color cursor, the cursor source (i.e., from Windows) is specified as a 1 BPP AND mask and a color bitmap which can be 8, 16 or 24 BPP. SM3110 defines the cursor surface as a 16 BPP format with the desired stride in pixel units. The color pixels and the AND mask are combined into 16-bit pixels with the most significant bit containing the AND mask and the remaining 15 bits containing RGB value with a 5-5-5 format. The resulting 16-bit pixels are written into the cursor surface.

The cursor size is 32x32 for this example, the maximum possible size for color cursors. Assume the current mode is 16 BPP, RGB 5:6:5 mode. The given cursor definition has 128 bytes of monochrome bitmap (the AND mask) and 2KB of color bitmap (the XOR component). The color cursor surface stride has to be 32 pixels (16 BPP).

Sample code: Set Color cursor

```

;fs= CmdPortSelector: selector points to the register base address
;ds:[esi] points to the cursor pattern,128 bytes of AND mask followed by 2048
                        bytes of color bitmap.

SetColorCursor    proc        near
    mov     es, wVRAMSelector
    mov     edi, dwCursorBufferOffset ;es:[edi] points to the offscreen to
                                        load the cursor pattern.

    mov     fs:[CursorFormat],CURSOROFF ;turn off cursor
    mov     gdwCursorFormat, CURSOR16 ;global variable to turn On cursor later
    mov     eax,32                      ;stride
    or     eax,BPP16 shl 20             ;surface is 16 BPP
    mov     fs:[ CURSOR0SURFACE+4],eax ;set stride and format

    mov     ecx,32*2                    ;count of words to handle
    mov     ebx,128                     ;init. offset to color bitmap
OLOOP:
    mov     ax,ds:[esi]                 ;load AND mask
    rol     eax,16                      ;and mask in MS 16 bits
    mov     dx,16                       ;inner loop count
@@:
    mov     ax,ds:[esi+ebx]             ;load color component
    rcr     ax,6                        ;convert 565 to 555, discard LS Green
                                        bit
    rol     ax,5                        ;prepare to load AND mask in MS bit
    rcl     eax,1                       ;get AND mask bit into carry
    rcr     ax,1                        ;rotate into MS bit with 555 color
    add     ebx,2                       ;offset to next pixel of color bitmap
    stosw                                ;store AND mask and color component into
                                        offscreen
    dec     dx                          ;process 16 pixels
    jnz     @B
    add     esi,2                       ;point to next word of AND mask
    loop   OLOOP
    ret
SetColorCursor    endp

```

Note: The cursor is now set and needs to be turned on and positioned. The color cursor size can be a maximum of 32 x 32 pixels. If the given cursor size is smaller, the cursor pattern is loaded into the top left corner of the cursor bitmap and the viewport adjusted to display the cursor. The CursorPositionStarts and CursorPositionEnds are set based on the cursor width and height; the desired portion of the cursor surface is displayed. The 24 BPP, RGB 888 bitmaps are converted to a 16-bit, 555 format by packing the most significant 5 bits of the red, green and blue components. If the color depth is 8 BPP, the RGB values for each color index (pixel) are converted to an RGB 555 word. No conversion is necessary for 16-bit RGB 555 bitmaps.

6.4.7.5 Cursor Deferred control

To avoid the tearing or jumping of the cursor image, all writes to the cursor registers are recorded into the Deferred Cursor Buffer and rewritten during VBLANK. The immediate register address, with encoded byte enables followed by register data, is recorded. The immediate registers are updated with the recorded data during VBLANK. Macros LDAB, LDAH and LDAD are used to load the deferred offsets with encoded byte enables. These macros are defined in the “Equates” section.

Sample code: Set buffer location for cursor and deferred control

```

;fs= CmdPortSelector: selector points to the register base address

SetBuffer_CursorDeferredCtl    proc    near
    mov     eax, dwPrivatePhyAddr    ;get from GetFIFOBufferLocation
    add     eax, CURDEFERRED_OFFSET
    sub     eax, PhyVRAMAddress      ;relative to local memory base
    mov     dwCurDeferBufferOffset,eax ;offscreen to load the cursor deferred
                                         (addr,data) pairs

    ret
SetBuffer_CursorDeferredCtl    endp

```

Usually SM3110 uses the “Deferred control pointer 0” as the cursor deferred control pointers. To use the deferred control, first check if the deferred control status is replaying. When idle, reset the read pointer to the buffer allocated above, write the (command address, value) pairs to the buffer, then update the write pointer to the address of the last pair. Finally, enable the replay bit in the status register. During the next VBLANK period, the SM3110 will get the data from the deferred read pointer and execute the ‘command’ with the following ‘value’. The read pointer will be automatically updated by the hardware after the data was read. It will not stop until the read pointer matches the write pointer. An example is shown in the next section of “Set Cursor Position”.

6.4.7.6 Set Cursor Position and Turn Cursor On

The cursor has been previously turned off and a new cursor shape has been set. The cursor display and position parameters have to be set during VBLANK to avoid tearing or jumping of the cursor. This is accomplished by writing to the Deferred buffer and enabling replay; and the real register updates occur during VBLANK. The cursor position can be set independently if the cursor was already on.

The cursor position and display size can be controlled, along with offsets which enable partial cursors to be displayed at the left and top edges of the screen. The cursor surface descriptor is set up for the base address, format and stride. The (x,y) offsets are set along with the (x,y) position of the top left corner of the cursor. The (x,y) offsets are set to (0,0) if the cursor is fully visible. Note that the cursor position (x,y) can only be positive numbers. To enable partial cursor to be visible at the left or top edges of the screen, the cursor position x or y is set to 0. The corresponding x or y offset, the portion of the cursor not visible in pixel units, is set in the offset register. The cursor format has to be set for the display portion along with the extents.

Sample code: Set Cursor Position and Turn Cursor On

```

;fs= CmdPortSelector: selector points to the register base address
;Cursor height and width are assumed 32x32
;Deferred Control buffer in the offscreen memory, selector es points to the
                        screen base .
;variables gCursorX/Y is cursor location, gCursorHotX/Y is hot spot inside
                        cursor,
;variables ScreenWidth/Height is current screen dimension.

MoveCursor      proc      near
;read Deferred control status - disarms Deferred control replay
@@:
    mov     al,fs:[CursorReplayStsReg]    ;wait while replay active
    test   al,REPLAYACTIVE;
    jnz    @B                            ;skip to load cursor if inactive

    mov     edi, dwCurDeferBufferOffset  ;physical buffer reserved for
                                        deferred control
    mov     fs:[CursorReplayPtr],edi     ;Reset replay pointer
    mov     es, wVRAMSelector            ;es:[edi]=offscreen for deferred control

    mov     eax, gdwCursorFormat; CURSOR1/16
    LDAD   ebx,CursorFormat;
    mov     es:[edi],ebx                 ;record reg address and data into
                                        deferred buffer
    mov     es:[edi+4],eax                ;turn on mono or color cursor
    add     edi,8
;;clip cursor at Y axis
    xor     bx,bx                        ;assume cursor yoffset = 0
    mov     cx,32                        ;assume extent = 32
    mov     ax,gCursorY                  ;ax = screen coordinate of cursor
    sub     ax,gCursorHotY                ;correct for hot spot
    jge    clipcursorY_bottom            ;Is adjusted coordinate negative?
    add     cx,ax                          ;yes, shorten extent
    sub     bx,ax                          ;advance yoffset
    xor     ax,ax                          ;set screen coordinate to zero.
    jmp     clipcursorY_done

```

```

clipcursorY_bottom:
    mov    dx,ax                ;dx = top of cursor
    add    dx,cx                ;dx = bottom+1 of cursor
    sub    dx,ScreenHeight     ;dx = # pixels cursor is clipped on bottom
    jle    clipcursorY_done    ;If 0 or neg, cursor is completely visible
    sub    cx,dx                ;adjust extent
clipcursorY_done:

    shl   eax,16                ;accumulate y position
    shl   ebx,16                ;offset
    shl   ecx,16                ;and y end in upper 16 bits
;;clip cursor at X axis
    xor   bx,bx                ;assume cursor xoffset = 0
    mov   cx,32                ;assume extent = 32
    mov   ax,gCursorX          ;ax = screen coordinate of cursor
    sub   ax,gCursorHotX       ;correct for hot spot
    jge   clipcursorX_right    ;Is adjusted coordinate negative?
    add   cx,ax                ;yes, shorten extent
    sub   bx,ax                ;advance xoffset
    xor   ax,ax                ;set screen coordinate to zero.
    jmp   clipcursorX_done
clipcursorX_right:
    mov   dx,ax                ;dx = lhs of cursor
    add   dx,cx                ;dx = rhs+1 of cursor
    sub   dx,ScreenWidth       ;dx = # pixels cursor is clipped on right side
    jle   clipcursorX_done    ;If 0 or neg, cursor is completely visible
    sub   cx,dx                ;adjust extent
clipcursorX_done:
    add   ecx,eax;cx = (right,bottom)
;
    mov   es:[edi],edx         ;record reg address and data into deferred
                                buffer
    mov   es:[edi+4],eax       ; set cursor position top left x,y
    add   edi,8

    LDAD  edx,ChCurOffsets;
    mov   es:[edi],edx         ;record reg address and data into deferred
                                buffer
    mov   es:[edi+4], ebx     ;set cursor offsets
    add   edi,8

    LDAD  edx,CursorPositionEnds;
    mov   es:[edi],edx         ;record reg address and data into deferred
                                buffer
    mov   es:[edi+4], ecx     ;set bottom right position
    add   edi,8

    sub   edi,4                ;go back to the last valid address
    mov   fs:[CursorRecordPtr],edi ;set cursor write base pointer
    mov   fs:[CursorReplayCtlReg],ENABLEREPLAY ;Enable replay at next VBLANK
    ret
MoveCursor      endp

```

The cursor will be displayed at the next VBLANK.

6.4.8 Icon

The SM3110 supports a hardware icon.

Sample code: Set icon buffer

```
;fs= CmdPortSelector: selector points to the register base address
SetBuffer_Icon  proc  near
    mov     eax, dwPrivatePhyAddr      ;get from GetFIFOBufferLocation
    add    eax, ICON_OFFSET
    sub    eax, PhyVRAMAddress        ;relative to local memory base
    mov    dwIconBufferOffset,eax     ;offscreen to load the icon pattern,
                                        save as global
    mov    fs:[ICONSURFACE],eax      ;surface descriptor base
    mov    fs:[ChIconOffsets],0      ;offsets into icon surface
    mov    fs:[ChIconSIndex],ICONSURFACEINDEX ;set icon surface descrip-
                                        tor index
    ret
SetBuffer_Icon  endp
```

Programming for the icon is similar to the cursor functions in the previous section. Follow those routines to set Monochrome/Color Icon to the allocated icon buffer, set icon position and turn on icon, except use “IconChFormat/IconChPositionStarts/ChIconOffsets/IconChPositionEnds” instead of “CursorFormat /CursorPositionStarts/ChCurOffsets/CursorPositionEnds” respectively.

6.5 2D Functions

6.5.1 Display Operations

The SM3110 controller provides a very powerful and flexible mechanism for defining surfaces and displaying multiple surfaces via channels. Multiple surfaces can be displayed with alpha blending and color keying.

The display memory can be logically viewed as surfaces. The surface descriptors define surfaces. Each descriptor has the base address of the surface (in physical memory), the stride (in pixels) and the pixel size (in bytes). Surface stride has to be a multiple of 16 pixels, and the base address has to start on a 4-byte boundary. The SM3110 allows 16 descriptors, 00H through 0FH. The 2D engine or the CPU can render into any surface. It is possible to perform inter-surface BitBLTs. Surfaces are displayed by assigning them to one of seven(7) channels: two cursors, one icon, two displays and two scalers. Surface descriptors 00 and 0Fh are reserved for cursor channels 0 and 1, descriptor 01h is reserved for the icon channel and descriptors 04h and 0Eh are reserved for display channels 0 and 1 by BIOS and the display driver.

There are two sets of cursor/scaler/display: for output channel 0 and output channel 1. The cursor/scaler/display set for output channel 0 is associated with the LCD output, and the cursor/scaler/display set for output channel 1 is associated with the CRT. A surface can be attached to a display channel. The attached surface is displayed on the screen through a viewport located at a specified position. The screen resolution is based on the mode currently set. The pixel format and size have to be specified for each channel. The viewport location and its extents (size) can be specified. It is thus possible to display any portion of a surface on the viewport, clipped at the left and top edges. The X,Y offsets become the origin into the surface bitmap that is displayed in the viewport. With the X,Y offsets set to 0,0, the bitmap is displayed with the top left corner coincident with the top left corner of the viewport. Figure 6-6 below shows an example of a surface being displayed by a display channel.

The icon channel is used to display ICON bitmap when LCD is powered down. The cursor channel is provided for hardware cursor support which is described in the previous section. The surface associated with the scaler channel can be of type RGB or YUV, and can be stretched or shrunk. Only RGB surfaces can be attached to display channels. The scaler channel can be used as a blend or as key control for the display channels. Certain modes can display only two channels due to bandwidth and/or memory limitations.

By convention, when an enhanced mode is set, surface 4 is allocated by the BIOS to describe the bitmap for that mode. This surface, the primary surface, is attached to a display channel. The surface 0 is reserved for the cursor channel. The base addresses for the primary surface and the cursor surface can be obtained from making extended BIOS calls. During mode setting, the cursor and scaler channels are turned off by programming the channel format to 0. The channels can be turned on later by programming with a legal format.

The surfaces attached to scaler and display channels can be displayed on the screen simultaneously. The overlay priority control register can specify the display order of the surfaces. The scaler channel can be assigned an order as foreground for display, or as a background surface which can be blended or colorkeyed with the foreground display channel. The blending can be constant defined by the Mix control register. The keying can be on the source (foreground channel) or the destination (background channel). If source keying is selected, foreground channel pixels having the key color display pixels from the background channel; the foreground channel

pixels not having the key color are displayed everywhere else. If destination keying is selected, foreground channel pixels corresponding to key color pixels in the background channel are displayed. The background channel pixels are displayed everywhere else in the overlapping area. If a control channel is specified for keying, it takes precedence over the key control registers. Figure 6-7 shows the overlay and mix control. In certain modes, it is possible to have only two channels displayed due to memory and or bandwidth limitations.

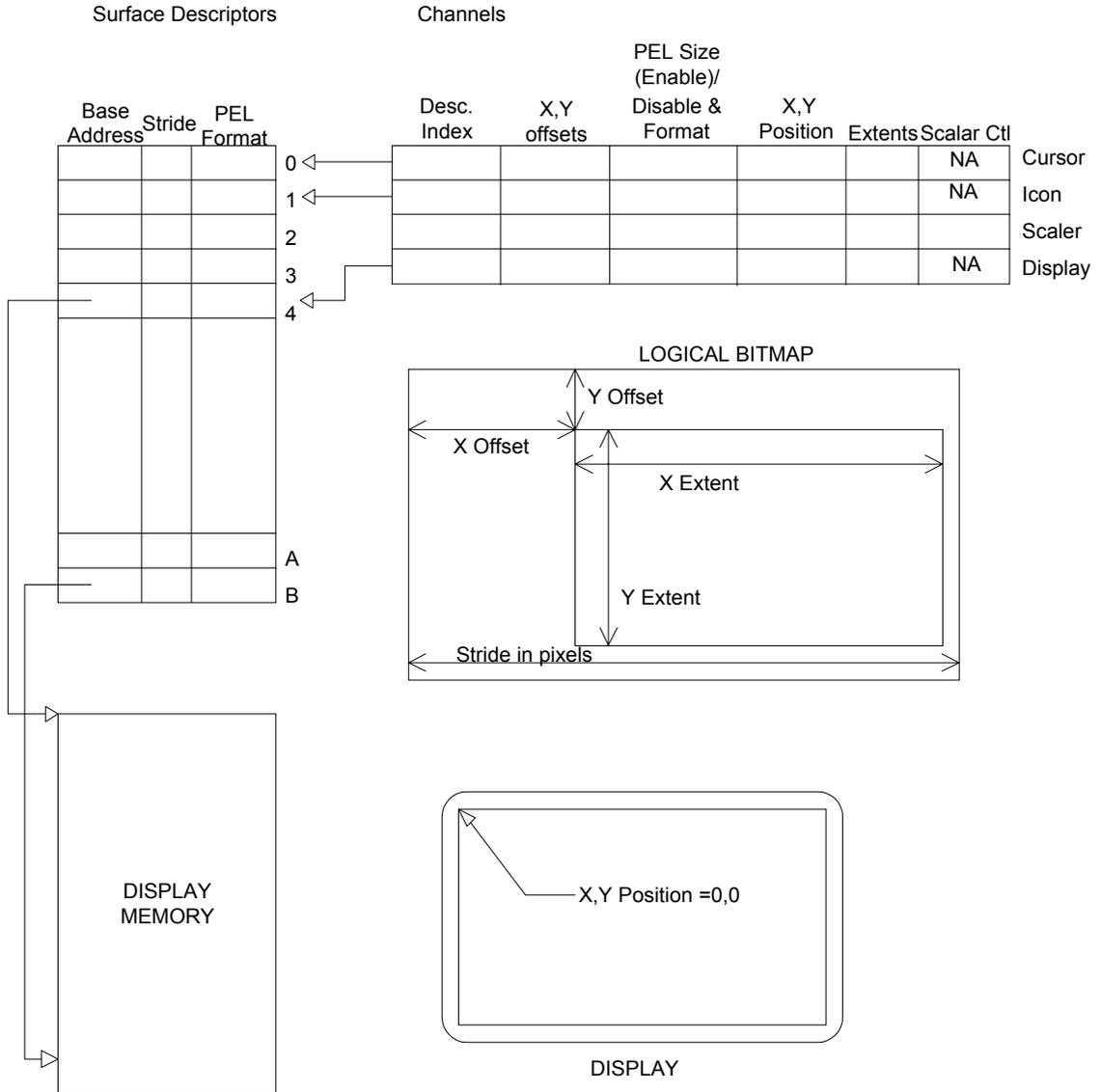


Figure 6-6. Surface Definition

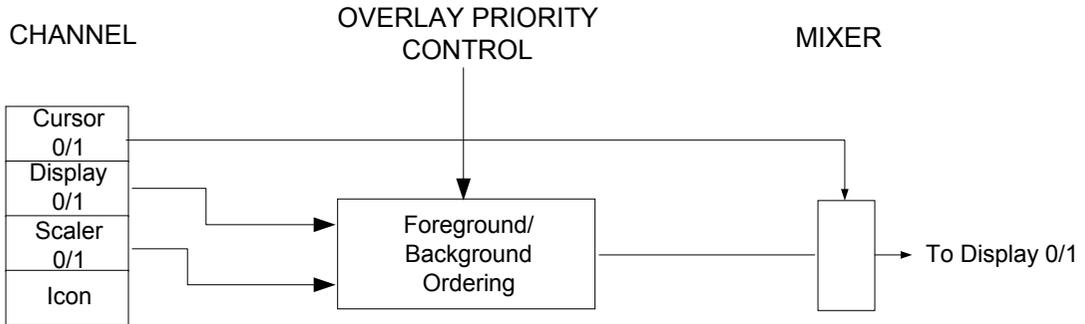


Figure 6-7. Overlay Priority and Mixing

This section provides examples to perform common display operations. These are explained as specific examples for clarity, but could be combined in real applications.

- Define surfaces.
- Channel controls - assign surface
- Channel controls - set location, size, scale factors etc.
- Enable/disable a channel.
- Set Overlay priority.
- Set Back/Fore Mix control (blend and overlay).
- Query the scanline being displayed.

In all the examples, the writes to the display control registers are deferred and written during VBLANK. Writes to these registers are recorded into the Deferred Control Buffer and replayed during VBLANK. The immediate register address with encoded byte-enables, followed by register data, are recorded. Macros LDAB, LDAH and LDAD are used to load the deferred offsets with encoded byte-enables (these macros are defined in Section 6.14.2, Macros). This eliminates the undesirable tearing effects. A surface buffer attached to a channel can be swapped with a newly rendered surface without visual artifacts. It is appropriate to modify the controls via the immediate registers if the channel is disabled for display.

6.5.1.1 Define a Surface

This routine defines a surface given the base address, the bits per pixel and the stride in pixel units. The base address should be on a doubleword boundary, and the stride should be a multiple of 16-pixel units. The surface is uninitialized. Make sure that the surface entry being defined is not already assigned to any surface.

Sample code: Define a Surface

```

;fs selector points to the register base address
;the variable bitspixel has the BPP with the stride in surfacestride. The sur-
    face number is in surfacenum (0..F) and
    surface
;base address is in the variable surfacebase.

DefineSurface    proc    near
    mov    ax,NEXTSURFACE        ;calculate the surface
    mul    ax,surfacenum        ;register address
    mov    bx,ax                ;
    mov    eax,surfacebase      ;load surface base address
    mov    fs:[bx+SurfaceBaseAddress],eax    ;set base address for given
    surface
    mov    ax,bitspixel        ;load bits per pixel
    shl    eax,20              ;move to upper bits
    mov    ax,surfacestride    ;load surface stride
    mov    fs:[bx+SurfaceStrideFormat],eax    ;set stride and format
    ret
DefineSurface    endp

```

6.5.1.2 Channel Controls - assign surface

The given surface is assigned to the channel 0,1,2 or the control channel. In a typical operation, one surface assigned to a channel is being displayed, and a second surface is being rendered into or composed. The surfaces are then flipped to display the newly rendered surface. The surface flip needs to occur during VBLANK to avoid undesirable screen artifacts. This is accomplished by recording the writes to the Display Control registers into the Deferred Control Buffer and replaying the register writes into the registers during VBLANK.

Sample code: Display Deferred control

```

;fs selector points to the register base address
;the variable surfacenum has the Surface descriptor index of surface to be
;attached. The variable channelnum has the channel
;number, one of (CHANNELCTL, CHANNEL0,CHANNEL1, CHANNEL2).
;Variable PhyCmdPortAddress has the immediate register physical base
;address, add offsets to yield register addresses
;displayreplaybase has the base address offset of Display Deferred
;Control;buffer in the offscreen memory.
;gs selector points to the physical screen base.

```

```

DisplayDeferredControl    proc    near
    mov    edi,fs:[DisplayRecordPtr]        ;load edi display write pointer
;read Display control status - disarms Deferred Display replay
    mov    al,fs:[DisplayReplayStsReg]      ;read status, disarm replay
    and    al,REPLAYPENDING+REPLAYACTIVE    ;
    cmp    al,REPLAYPENDING                ;skip to add to Control buffer
    jz     RECORDLOADS                    ;if replay pending

@@:
    mov    al,fs:[DisplayReplayStsReg]      ;else wait while replay active
    test   al,REPLAYACTIVE                 ;
    jnz    @B                               ;skip to record writes if inactive
    mov    edi,displayreplaybase            ;set read and write pointers
    mov    fs:[DisplayRecordPtr],edi        ;set display write base pointer
    mov    fs:[DisplayReplayPtr],edi        ;set display read base pointer

RECORDLOADS:
    sub    edi,PhyCmdPortAddress            ;edi has offset into buffer
    mov    ax,NEXTCHANNEL                   ;calculate
    mul    ax,channelnum                    ;register address offset for
                                                channel
    movzx  ebx,ax                           ;in ebx
    LDAB  edx,ChannelIndex                  ;
    add    edx,ebx                           ;add to base
    or     edx,PhyCmdPortAddress            ;record reg address and data into
                                                deferred buffer
    mov    gs:[edi],edx                     ;
    mov    al,surfacenum                    ;load surface base number, dis-
                                                played after next VBLANK
    mov    gs:[edi+4],al                    ;set surface number
    add    edi,(8-4)                         ;compute write pointer
    add    edi,PhyCmdPortAddress            ;
    mov    fs:[DisplayRecordPtr],edi        ;set display write pointer
    mov    fs:[DisplayReplayCtlReg],ENABLEREPLAY;Enable replay at next VBLANK
    ret

DisplayDeferredControl    endp

```

6.5.1.3 Channel Controls - Offsets & viewports location

The offsets, viewport location and the extents are set for a *display* channel. It is assumed that a surface has been previously assigned to this channel and is being displayed. Setting the offsets establishes the new origin into the surface being displayed. The viewport X,Y sets the location of the top left corner of the display viewport, and the extents sets the width and height of the display viewport. If the channel is being displayed, these parameters need to be set during VBLANK to avoid undesirable screen artifacts. This is accomplished by recording the writes to the Display Control registers into the Deferred Control Buffer and replaying the register writes during VBLANK. A *scaler* channel can be set in a similar manner along with the location and extents. This controls the stretch factor in the x and y directions.

Sample code: Channel control

```

;fs selector points to the register base address
;The variable channelnum has the channel number, one of (CHANNELCTL,
CHANNEL0,CHANNEL1, CHANNEL2).
;Variable PhyCmdPortAddress has the immediate register physical base address,
add offsets to yield register addresses
;displayreplaybase has the base address offset of Display Deferred Control
buffer in the offscreen memory.
;gs selector points to the physical screen base.
;Variables offsetx, offsety, viewx, viewy, and viewendx, viewendy contain the
offsets, viewport location starts and ends.
;The view location parameters are clipped to screen coordinates. The variable
sstride contains the stride of the surface
;sheight has the surface height.

```

```

ChannelControl  proc  near
    mov     edi,fs:[DisplayRecordPtr]      ;load edi display write pointer
;read Display control status - disarms Deferred Display replay
    mov     al,fs:[DisplayReplayStsReg]   ;read status, disarm replay
    and     al,REPLAYPENDING+REPLAYACTIVE ;
    cmp     al,REPLAYPENDING              ;skip to add to Control buffer
    jz      RECORDLOADS                   ;if replay pending

@@:
    mov     al,fs:[DisplayReplayStsReg]   ;else wait while replay active
    test    al,REPLAYACTIVE                ;
    jnz     @B                             ;skip to record writes if inactive

    mov     edi,displayreplaybase          ;set read and write pointers
    mov     fs:[DisplayRecordPtr],edi     ;set display write base pointer
    mov     fs:[DisplayReplayPtr],edi     ;set display read base pointer

RECORDLOADS:
    sub     edi,PhyCmdPortAddress          ; edi has offset into buffer
    mov     ax,NEXTCHANNEL                  ;calculate
    mul     ax,channelnum                   ;register address offset for channel
    movzx   ebx,ax                          ;
    LDAD    edx,ChViewPositionStarts      ;load Channel View base

```

```

add    edx,ebx                ;offset into desired channel
or     edx,PhyCmdPortAddress ;record reg address and data into
                                deferred buffer

mov    gs:[edi],edx          ;
mov    ax,viewy              ;load viewport x,y
shl    eax,16                ;
mov    ax,viewx              ;
mov    gs:[edi+4],eax        ;set viewport x,y

LDAD   edx,ChViewPositionEnds ;load Channel Extent Base
add    edx,ebx                ;offset into desired channel
or     edx,PhyCmdPortAddress ;record reg address and data into
                                deferred buffer

mov    gs:[edi+8],edx        ;
mov    ax,viewendy           ;load x,y ends
shl    eax,16                ;
mov    ax,viewendx          ;
mov    gs:[edi+12],eax       ;set x,y ends

LDAD   edx,ChannelOffsets    ;load Channel offset Base
add    edx,ebx                ;offset into desired channel
or     edx,PhyCmdPortAddress ;record reg address and data into
                                deferred buffer

mov    gs:[edi+16],edx       ;
mov    ax,offsety            ;load x,y offsets into the surface
                                from top left corner

shl    eax,16                ;
mov    ax,offsetx           ;
mov    gs:[edi+20],eax       ;set x,y offsets

if (Surface type YUV) {
LDAD   edx,Ch0UOffsets       ;load Channel U offsets
or     edx,PhyCmdPortAddress ;record reg address and data into
                                deferred buffer

mov    gs:[edi+24],edx       ;
mov    ax,offsety            ;load U offsety = offsety/2+sheight
shr    ax,1                  ;
add    ax,sheight           ;
shl    eax,16                ;
mov    ax,offsetx           ;U offsetx = offsetx/2
shr    ax,1                  ;
mov    gs:[edi+28],eax       ;set U offsets

LDAD   edx,Ch0VOffsets       ;load Channel V offsets
or     edx,PhyCmdPortAddress ;record reg address and data into
                                deferred buffer

mov    gs:[edi+32],edx       ;
mov    ax,offsety            ;load V offsety = offsety/2+sheight
shr    ax,1                  ;
add    ax,sheight           ;
shl    eax,16                ;

```

```
mov    ax,offsetx                ;V offsetx = (offsetx+stride)/2
add    ax,sstride                ;
shr    ax,1                      ;
mov    gs:[edi+36],eax           ;set V offsets
add    edi,(40-4)                ;compute write pointer
} else {
add    edi,(24-4)                ;compute write pointer
}
add    edi,PhyCmdPortAddress     ;
mov    fs:[DisplayRecordPtr],edi ;set display write pointer
mov    fs:[DisplayReplayCtlReg],ENABLEREPLAY ;Enable replay at next
                                           VBLANK
ret
ChannelControl    endp
```

6.5.1.4 Enable/Disable Channel Display

When a display channel is either enabled or disabled, the attached surface is either displayed or turned off. The disabling can be effected immediately by writing to the immediate registers. The enable needs to occur during VBLANK to avoid undesirable screen artifacts. This is accomplished by recording the writes to the Display Control registers into the Deferred Control Buffer and replaying the register writes during VBLANK. On enabling/disabling channels, the display FIFO controls have to be set.

Sample code: Enable/Disable Channel Display

```

;fs selector points to the register base address
;Variable physicalbase has the immediate register physical base address, add
    offsets to yield register addresses
;displayreplaybase has the base address offset of Display Deferred Control
    buffer in the offscreen memory.
;gs selector points to the physical screen base.
;The variable channelnum has the channel number, one of (CHANNELCTL,
    CHANNEL0,CHANNEL1, CHANNEL2).
;displayformat contains the format and bitdepth of the surface being dis-
    played, e.g. 16 BPP YUV 4.2.2.
;If the display channel is to be disabled, displayformat contains 0

```

```

EnableChannelDisplay    proc    near
    mov     bx,NEXTCHANNEL        ;calculate
    mul     ax,channelnum        ;register address offset for channel
    movzx  ebx,ax ;
    mov     cl,displayformat;
    test   cl,0ffh              ;turn on display
    jnz    @F                   ;yes: skip to turn on display
    mov     fs:[bx+ChannelFormat],cl ;no turn off channel display
    jmp    DONE

@@:
    mov     edi,fs:[DisplayRecordPtr] ;load edi display write pointer
    ;read Display control status - disarms
    ;Deferred Display replay
    mov     al,fs:[DisplayReplayStsReg];read status, disarm replay
    and     al,REPLAYPENDING+REPLAYACTIVE;
    cmp     al,REPLAYPENDING      ;skip to add to Control buffer
    jz     RECORDLOADS          ;if replay pending

@@:
    mov     al,fs:[DisplayReplayStsReg];else wait while replay active
    test   al,REPLAYACTIVE;
    jnz    @B                   ;skip to record writes if inactive

    mov     edi,displayreplaybase ;set read and write pointers
    mov     fs:[DisplayRecordPtr],edi ;set display write base pointer
    mov     fs:[DisplayReplayPtr],edi ;set display read base pointer

```

```

RECORDLOADS:
    sub    edi,physicalbase        ; edi has offset into buffer
LDAB     edx,ChannelFormat        ;load Channel format Base
    add    edx,ebx                ;offset into desired channel
    or     edx,physicalbase        ;record reg address and data into
                                deferred buffer

    mov    gs:[edi],edx           ;
    mov    gs:[edi+4],cl          ;enable channel

    add    edi,(8-4)              ;compute write pointer
    add    edi,physicalbase        ;
    mov    fs:[DisplayRecordPtr],edi ;set display write pointer
    mov    fs:[DisplayReplayCtlReg],ENABLEREPLAY ;Enable replay at next
                                VBLANK DONE:

    ret
EnableChannelDisplay    endp

```

6.5.1.5 Set Display FIFO Controls

The display FIFO has 10 entries and needs to be programmed for entry allocation and thresholds for the control channel as well as display channels 1 and 2. The FIFO control register has to be reprogrammed on enabling or disabling display channels.

```

;fs selector points to the register base address
;Variable physicalbase has the immediate register physical base address, add
                                offsets to yield register addresses
;displayreplaybase has the base address offset of Display Deferred Control
                                buffer in the offscreen memory.
;gs selector points to the physical screen base.
;The variable channelnum has the channel number, one of (CHANNELCTL,
                                CHANNEL0,CHANNEL1, CHANNEL2).
;displayformat contains the format and bitdepth of the surface being dis-
                                played, e.g. 16 BPP YUV 4.2.2.
;If the display channel is to be disabled, displayformat contains 0

```

6.5.1.6 Set Channel 0 X,Y Scale Factors

Sets up X and Y scale factor registers for for channel 0. The scale factors are based on the source and destination image sizes. The scaling interpolation can be bilinear or nearest-neighbor. It is not possible to scale down below an X or a Y factor of 1/16; the scaling upper bound is 8 for YUV and 16 for RGB. The MPEG uses a bilinear interpolation scheme if the X scale factor is greater than 0.25; nearest-neighbor is used otherwise. For non-MPEG applications, bilinear interpolation is used with pre-filtering if the X scale factor is less than or equal to 0.5.

```

/* The following variables are given:
/*
Given
int srcXstride, srcYstride
int dstXstride, dstYstride, dstXoffset, dstYoffset, dstWidth, dstHeight, dstX-
left, dstYtop
(Refer to Figure 6-6 for definition of above arguments.)
Boolean MPEG (1 if source is from SM3110's MPEG decoder, otherwise 0)
Boolean YUV422(1 if source is in interleave 4:2:2 format, 0 if planar format
4:2:0)
Boolean YUV (1 if source is in YUV format, 0 if RGB)
*/

#define IncFracBits          11
#define INTG(x)              ((x)>>IncFracBits)
#define FRAC(x)              ((x)&((1<<IncFracBits)-1))

/* Clamp scaling factors to supported range */

xfactor = (double)dstXstride/srcXstride;

/* clamp scale Factor in x down to 1/16 or up to 8 for YUV or 16 for RGB */
if (xFactor < 0.0625) {
    xFactor = 0.0625;
    dstXstride = srcXstride * xFactor + 0.5;
} else {
    if ((RGB && (xFactor > 16.0)) {
        xFactor = 16.0;
        dstXstride = srcXstride * xFactor + 0.5;
    } else {
        if (YUV && (xFactor > 8.0)){
            xFactor = 16.0;
            dstXstride = srcXstride * xFactor + 0.5;
        }
    }
}

yFactor = (double)dstYstride/srcYstride;

/* clamp scale Factor in y down to 1/16 or up to 8 for YUV or 16 for RGB */
if (yFactor < 0.0625) {

```

```

yFactor = 0.0625;
dstYstride = srcYstride * yFactor + 0.5;
} else {
  if ((RGB && (yFactor > 16.0)) {
    xFactor = 16.0;
    dstYstride = srcYstride * yFactor + 0.5;
  } else {
    if (YUV && (yFactor > 8.0)){
      yFactor = 16.0;
      dstYstride = srcYstride * yFactor + 0.5;
    }
  }
}

/* Adjust the source x stride if pre-scaling by hardware is to be performed.
*/

prescale = 1;
if (!MPEG && (xfactor<=.5)) {
  while (xfactor<=.5) {
    xfactor *= 2.;
    prescale <<= 1;
  }
}

/* We need to pre-filter in X if prefilter > 1. Create a new source bitmap,
average every 2, 4 or 8 pixels in the x direction based on prescale of 2,
4 or 8 respectively. If p0,p1,p2,p3,p4,p5,p6,p7,p8..pn are pixels in a
row. If variable prescale == 2, then the new source pixels will be pn0,
pn1, pn2... where pn0=(p0+p1)>>1, pn1=(p2+p3)>>1... If variable prefilter
== 4, then the new source pixels will be pn1, pn2, pn3... where
pn0=(p0+p1+p2+p3)>>2, pn1=(p4+p5+p6+p7)>>2. Note: If srcWidth, the width
of the bitmap is not an even multiple of prefilter, the last pixel in the
row pixel is the average of the trailing remainder of pixels (less than
prescale) . */

if (prefilter > 1) {
  srcXstride = dstXstride/xfactor+.5;
  prefilter the Bitmap
}

/* Determine interpolation mode */
if (MPEG && (xFactor<=0.25))
  interpolationMode = XNEAREST+YNEAREST;
else
  interpolationMode = XBILINEAR+YBILINEAR;

/* Compute x and y increments for Y component */
xIncY = (int)(((double)(srcXstride-1)/(dstXstride-1))*(1<<IncFracBits)+.5);
yIncY = (int)(((double)(srcYstride-1)/(dstYstride-1))*(1<<IncFracBits)+.5);

```

```

/* Find (xOffsetY, yOffsetY) from (dstXoffset, dstYoffset) */
if (dstXleft < 0 && dstXoffset < abs(dstXleft)) {
    dstXoffset = -dstXleft;
    dstWidth = dstWidth + dstXleft - dstXoffset;
}
srcXstart = dstXoffset*xIncY;          /* srcXstart in 0.21.11 format */
xOffsetY = INTG(srcXstart);           /* xOffsetY = x offset for Y component
                                        */
initialXfracUV = FRAC(srcXstart>>1)>>(IncFracBits-4);          /* 0.0.4 for-
                                        mat */
..... (Compute initialYfracUV in a similar way.)

/* Compute x and y increments for UV components */
xOffsetUV = xOffsetY>>1;
srcXstrideUV = srcXstride>>1;
if (!YUV420) {
    srcYstrideUV = srcYstride;
    yOffsetUV = yOffsetY;
} else {
    srcYstrideUV = srcYstride>>1;
    yOffsetUV = yOffsetY>>1;
}
xIncUV = (int)(((double)(srcXstrideUV-xOffsetUV-1-initialXfracUV*.0625) /
    (dstXstride-dstXoffset-1))*(1<<IncFracBits)+.5);
yIncUV = (int)(((double)(srcYstrideUV-yOffsetUV-1-initialYfracUV*.0625) /
    (dstYstride-dstYoffset-1))*(1<<IncFracBits)+.5);

viewX = dstXleft + dstXoffset;
viewY = dstYtop + dstYoffset;
interpolationMode = (initialYfracUV << 16) + initialXfracUV;

```

Sample code: Set Scaler Factor

```

;fs selector points to the register base address
;Variable physicalbase has the immediate register physical base address, add
                                offsets to yield register addresses
;displayreplaybase has the base address offset of Display Deferred Control
                                buffer in the offscreen memory.
;gs selector points to the physical screen base.

```

```

Set Scaler Factor      proc    near
    mov     edi,fs:[DisplayRecordPtr]          ;load edi display write pointer
;read Display control status - disarm Deferred Display replay
    mov     al,fs:[DisplayReplayStsReg]       ;read status, disarm replay
    and     al,REPLAYPENDING+REPLAYACTIVE     ;
    cmp     al,REPLAYPENDING                  ;skip to add to Control buffer
    jz     RECORDLOADS                        ;if replay pending

```

```

@@:
mov     al,fs:[DisplayReplayStsReg]      ;else wait while replay active
test   al,REPLAYACTIVE                  ;
jnz    @B                                ;skip to record writes if inactive

mov     edi,displayreplaybase           ;set read and write pointers
mov     fs:[DisplayRecordPtr],edi       ;set display write base pointer
mov     fs:[DisplayReplayPtr],edi       ;set display read base pointer
    
```

RECORDLOADS:

```

sub     edi,physicalbase                 ; edi has offset into buffer
LDAD   edx,Ch0Controls                  ;load Scaler controls, interpolation
                                           mode with initial fractions
or     edx,physicalbase                 ;record reg address and data into
                                           deferred buffer

mov     gs:[edi],edx                    ;
mov     eax,interpolationMode           ;
mov     gs:[edi+4],eax                  ;

LDAD   edx,Ch0YRControls                 ;load Scaler controls, Y/R increaments
or     edx,physicalbase                 ;record reg address and data into
                                           deferred buffer

mov     gs:[edi+8],edx                  ;
mov     ax,yIncY                        ;
shl    eax,16                           ;
mov     ax,xIncY                        ;
mov     gs:[edi+12],eax                 ;

LDAD   edx,Ch0UVGBControls              ;load Scaler controls, UV/GB increaments
or     edx,physicalbase                 ;record reg address and data into
                                           deferred buffer

mov     gs:[edi+16],edx                 ;
mov     ax,yIncUV                       ;
shl    eax,16                           ;
mov     ax,xIncUV                       ;
mov     gs:[edi+20],eax                 ;

LDAD   edx,Ch0Offsets                   ;load Scaler controls, X,Y offsets
or     edx,physicalbase                 ;record reg address and data into
                                           deferred buffer

mov     gs:[edi+24],edx                 ;
mov     ax, yOffsetY                    ;
shl    eax,16 `                          ;
mov     ax, xOffsetY                    ;
mov     gs:[edi+28],eax                 ;

LDAD   edx,Ch0UOffsets                  ;load Scaler controls, X,Y offsets for U
                                           plane
or     edx,physicalbase                 ;record reg address and data into
                                           deferred buffer

mov     gs:[edi+32],edx                 ;
    
```

```

mov     ax, yOffsetY      ;
shr     ax,1              ;
add     ax,srcYstride ;
shl     eax,16           ;
mov     ax, xOffsetY     ;
shr     ax,1             ;
mov     gs:[edi+36],eax   ;

LDAD    edx,Ch0VOffsets   ;load Scaler controls, X,Y offsets for V plane
or      edx,physicalbase ;record reg address and data into deferred
                               buffer

mov     gs:[edi+40],edx   ;
mov     ax, yOffsetY     ;
shr     ax,1             ;
add     ax,srcYstride    ;
shl     eax,16           ;
mov     ax,xOffsetY     ;
add     ax,srcXstride    ;
shr     ax,1             ;
mov     gs:[edi+44],eax   ;

LDAD    edx,ChViewPositionStarts ;load Channel View base
add     edx,ebx          ;offset into desired channel
or      edx,physicalbase ;record reg address and data into
                               deferred buffer

mov     gs:[edi+48],edx   ;
mov     ax,viewY         ;load viewport x,y
shl     eax,16           ;
mov     ax,viewX         ;
mov     gs:[edi+52],eax   ;set viewport x,y

LDAD    edx,ChViewPositionEnds ;load Channel Extent Base
add     edx,ebx          ;offset into desired channel
or      edx,physicalbase ;record reg address and data into
                               deferred buffer

mov     gs:[edi+56],edx   ;
mov     ax,viewY         ;load x,y ends
add     ax,dstHeight     ;
shl     eax,16           ;
mov     ax,viewX         ;
add     ax,dstWidth      ;
mov     gs:[edi+60],eax   ;set x,y ends

add     edi,(64-4)        ;compute write pointer
add     edi,physicalbase ;
mov     fs:[DisplayRecordPtr],edi ;set display write pointer
mov     fs:[DisplayReplayCtlReg],ENABLEREPLAY ;Enable replay at next
                               VBLANK

ret

Set Scaler Factor      endp

```

6.5.1.7 Set Overlay Priority

Sets overlay priority for the channel displays. The channel displays are logically designated as background and foreground for composing the resulting display. When no mix or overlays are set, the foreground appears on top of the background image. If only one channel is enabled, it is assigned as the background; the foreground is assigned a channel number that is disabled.

Sample code: Set Overlay Priority

```

;fs selector points to the register base address
;Variable physicalbase has the immediate register physical base address, add
      offsets to yield register addresses
;displayreplaybase has the base address offset of Display Deferred Control
      buffer in the offscreen memory.
;gs selector points to the physical screen base.
;variables forechannel and backchannel have channel numbers , one
;of ( CHANNEL0,CHANNEL1,CHANNEL2 ).

```

```

SetOverlayPriority proc near
    mov     edi,fs:[DisplayRecordPtr];load edi display write pointer
      ;read Display control status - disarms
      Deferred Display replay
    mov     al,fs:[DisplayReplayStsReg]      ;read status, disarm replay
    and     al,REPLAYPENDING+REPLAYACTIVE;
    cmp     al,REPLAYPENDING                ;skip to add to Control buffer
    jz     RECORDLOADS;if replay pending

```

```

@@:
    mov     al,fs:[DisplayReplayStsReg];else wait while replay active
    test    al,REPLAYACTIVE                 ;
    jnz    @B                               ;skip to record writes if inactive

    mov     edi,displayreplaybase           ;set read and write pointers
    mov     fs:[DisplayRecordPtr],edi      ;set display write base pointer
    mov     fs:[DisplayReplayPtr],edi      ;set display read base pointer

```

```

RECORDLOADS:
    sub     edi,physicalbase                ;edi has offset into buffer
    LDAD    edx,OverlayPriority              ;load Overlay priority
    or     edx,physicalbase                 ;record reg address and data into
      deferred buffer

    mov     gs:[edi],edx                    ;
    mov     al,forechannel                  ;load foreground
    sub     al,2                             ;normalize
    shl     eax,8                            ;
    mov     al,backchannel                  ;load background channel
    sub     al,2                             ;normalize
    mov     gs:[edi+4],eax                  ;enable channel

    add     edi,(8-4)                        ;compute write pointer
    add     edi,physicalbase                 ;
    mov     fs:[DisplayRecordPtr],edi      ;set display write pointer

```

```
mov    fs:[DisplayReplayCtlReg],ENBLEREPLAY
        ;Enable replay at next VBLANK
    ret
SetOverlayPriority endp
```

6.5.1.8 Set Mix Controls - Blend

With the priority controls set, the surfaces attached to the background can be blended with the foreground channel in certain modes. The blending can be constant or controlled on a per-pixel basis by defining a control surface and has to be enabled. For constant blend, the blend factor is set in the Mix Control register. This determines the percentage of source and destination used for the blend. With blend enabled, if a control surface (4BPP) is defined and enabled, the blend factor is defined by the control surface bitmap on a per-pixel basis. The constant blend is used for areas that do not overlap the control surface.

Sample code: Set Mix Controls - Blend

```

;fs selector points to the register base address
;Variable physicalbase has the immediate register physical base address, add
offsets to yield register addresses
;displayreplaybase has the base address offset of Display Deferred Control
buffer in the offscreen memory.
;gs selector points to the physical screen base.
;Assume that surfaces have been previously defined and attached to channel 0
and channel 1, and priority has been set.
;This example sets a constant blend for back and fore channels and enables
blending. Blend constant is specified by the
;variable blendfactor (0 < blendfactor < 0fh). If a 4BPP blend surface has
been previously defined and attached to
;the control surface, enables blend on a per pixel basis on the overlapping
areas.

```

```

SetBlending      proc  near
    mov     edi,fs:[DisplayRecordPtr] ;load edi display write pointer
                                           ;read Display control status - disarms
                                           Deferred Display replay
    mov     al,fs:[DisplayReplayStsReg];read status, disarm replay
    and     al,REPLAYPENDING+REPLAYACTIVE;
    cmp     al,REPLAYPENDING           ;skip to add to Control buffer
    jz     RECORDLOADS                ;if replay pending

```

```
@@:
mov     al,fs:[DisplayReplayStsReg];else wait while replay active
test   al,REPLAYACTIVE           ;
jnz    @B                         ;skip to record writes if inactive

mov     edi,displayreplaybase     ;set read and write pointers
mov     fs:[DisplayRecordPtr],edi ;set display write base pointer
mov     fs:[DisplayReplayPtr],edi ;set display read base pointer

RECORDLOADS:
sub     edi,physicalbase          ;edi has offset into buffer
LDAW   edx,Dsp0MixCtl            ;load Mix control register
or     edx,physicalbase          ;record reg address and data into
                                       deferred buffer

mov     gs:[edi],edx              ;
mov     ax,blendfactor           ;load blend constant
or     ax,BLENDENABLE           ;enable blend
mov     gs:[edi+4],ax            ;load back/fore mix data

add     edi,(8-4)                 ;compute write pointer
add     edi,physicalbase         ;
mov     fs:[DisplayRecordPtr],edi ;set display write pointer
mov     fs:[DisplayReplayCtlReg],ENABLEREPLAY
                                       ;Enable replay at next VBLANK

ret
SetBlending     endp
```

6.5.1.9 Set Mix Controls - Color Keying

Color keying allows pixels with a specified color (the key color) on an overlapping surface to become transparent and display the pixels from the underlying surface. The key color is set and the keying enabled. If the surface being compared is a YUV surface, the chroma compare color (Ch0ChromaCmp) is set. The keying operation can be performed on background (RGB) with the foreground channel. If keying is enabled and a 1 BPP control surface is defined and enabled, the control surface specifies the keying. Pixels corresponding to a 1 bit on the control surface allow the foreground to be displayed, while a 0 bit displays pixels from the background surface.

Sample code: Set Mix Controls - Color Keying

```

;fs selector points to the register base address
;Variable physicalbase has the immediate register physical base address, add
  offsets to yield register addresses
;displayreplaybase has the base address offset of Display Deferred Control
  buffer in the offscreen memory.
;gs selector points to the physical screen base.
;Assume that surfaces have been previously defined and attached to channel 0
  and channel 1, and priority has been set.
;This example sets key color given in variable keycolor and enables keying for
  the back channels.

```

```

SetColorKeying    proc        near
    mov     edi,fs:[DisplayRecordPtr]        ;load edi display write pointer
                                                ;read Display control status -
                                                ;disarms Deferred Display replay
    mov     al,fs:[DisplayReplayStsReg]     ;read status, disarm replay
    and     al,REPLAYPENDING+REPLAYACTIVE  ;
    cmp     al,REPLAYPENDING               ;skip to add to Control buffer
    jz     RECORDLOADS                     ;if replay pending

@@:
    mov     al,fs:[DisplayReplayStsReg]     ;else wait while replay active
    test    al,REPLAYACTIVE                ;
    jnz    @B                              ;skip to record writes if inactive

    mov     edi,displayreplaybase          ;set read and write pointers
    mov     fs:[DisplayRecordPtr],edi      ;set display write base pointer
    mov     fs:[DisplayReplayPtr],edi      ;set display read base pointer

```

```

RECORDLOADS:
    sub     edi,physicalbase           ; edi has offset into buffer
    LDAD   edx,Disp0ColCmp           ;load Display0 back/fore key compare
                                           color register
    or     edx,physicalbase           ;record reg address and data into
                                           deferred buffer

    mov    gs:[edi],edx               ;
    mov    eax,keycolor               ;set key color
    mov    gs:[edi+4],eax             ;load Key compare color

    LDAW  edx,Dsp0MixCtl             ;load Mix control register
    or     edx,physicalbase           ;record reg address and data into
                                           deferred buffer

    mov    gs:[edi+8],edx             ;
    mov    ax,ENABLECOLKEY+KEYNORMAL+KEYSRCFORE
    mov    gs:[edi+12],ax             ;load back/fore mix data

    add   edi,(16-4)                  ;compute write pointer
    add   edi,physicalbase            ;
    mov   fs:[DisplayRecordPtr],edi   ;set display write pointer
    mov   fs:[DisplayReplayCtlReg],ENABLEREPLAY ;Enable replay at next
                                           VBLANK

    ret
SetColorKeying   endp

```

6.5.1.10 *Get Displayed Scanline*

Returns the current scanline being displayed. The first line of active display is 0 increasing in Y, resets to 0 at the end of vertical blanking.

```

;fs selector points to the register base address.

mov ax,fs:[VerticalCounter]           ;read scanline currently being displayed

```

6.5.2 2D Command and Data FIFOs

All commands, parameters and operand data for the rendering engine are communicated by the host processor through two FIFOs that are fetched, parsed and executed by the rendering engine command processor. This allows concurrence between the host processor and the rendering engine that significantly improves drawing performance.

The two FIFOs are implemented in a portion of local memory allocated by the display driver. This has several advantages:

1. The size of these buffers may be (relatively) much larger than can be implemented using conventional SRAM or latch array FIFOs. This reduces the amount of overhead that the driver must spend managing a command/data FIFO (to prevent overflows).
2. It allows much larger data images (particularly BLTs and text) to be written into the buffer. This allows a longer sequence of commands to be posted by the driver so that it can return earlier or perform another task.
3. It allows the state of a command and its operand data to be retained in local memory if some task switch occurs with minimal state-saving overhead.

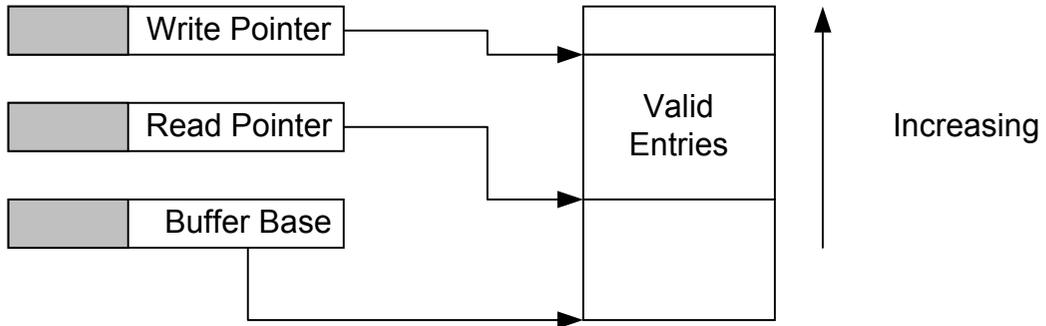
The control/parameter and image/data FIFO are implemented as circular buffers. The base addresses, write pointers and read pointers are stored memory controller registers and must be initialized by the display driver. The base address and size of each circular buffer are configurable and should be set up by writing directly to the registers before the engine is used the first time. They should not have to be changed again unless the buffer needs to be moved or resized.

The write pointer is normally updated by hardware when control information is written to the rendering engine aperture. The read pointer is incremented when this information is read by the rendering engine command processor. If the read pointer reaches the write pointer, the circular buffer is empty and fetching of control information stops pending further writes. If the write pointer reaches the read pointer, the circular buffer is full and further data will not be accepted from the host. The host bus interface may abort (if possible) to prevent locking out other independent transfers.

The driver should make every attempt to avoid having a transfer stall due to a filled condition by checking the status of the buffer if a large transfer is to be performed.

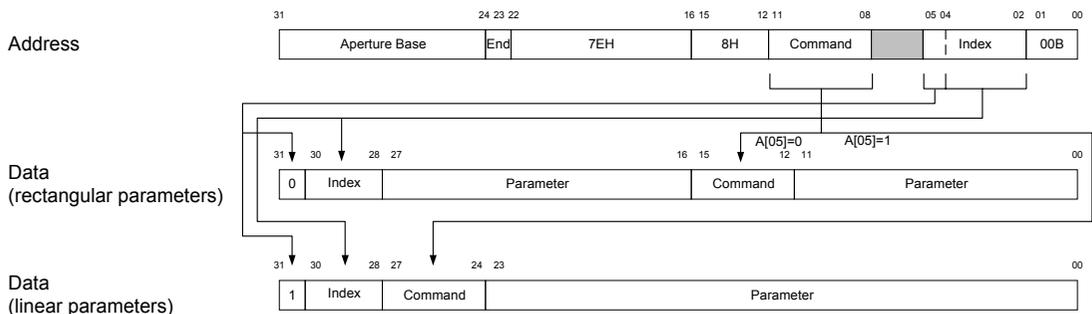
6.5.2.1 Image/Data Control Port

The 2D image data is also passed through buffers in local memory. Transfer of data images must be broken up into blocks of 64KB or less, constrained by the size of the buffer allocated. The buffer base address and buffer size must be initialized prior to use. The read and write pointers must also be initialized to zero, indicating an empty buffer. These should not have to be accessed again during normal operation.



6.5.2.2 2D Command Port

The command port is a 64KB port through which control and parameter information can be written in local memory. Additional control information is encoded in the address to reduce the number of data transfers required across the host bus. The least significant 16 bits of address are extracted, parsed and inserted into bits [31:28] and [15:12] of each parameter doubleword before it is written into the rendering buffer list. The actual address used bears no relationship to the address to which the parameter is written to local memory. If the command is other than a NOP, a doubleword entry is automatically stored in the buffer.



6.5.2.3 FIFO Status

The FIFO pointers will be updated whenever data is written by the host through the command/parameter or image data ports or read by the engine, which must have been started.

The read and write pointers may be directly read by the host to determine how full or empty a FIFO is. Note that if the host is writing to the respective FIFO or the engine is operating, the value read may be inaccurate for one of several reasons:

1. The exact value of each register may be off due to the latency in performing the read operation.
2. The accuracy of the value read may be wrong due to the latency of reading more than one byte, which may take several clock cycles. If one of the counters is updated while a read occurs, the value read from each byte may occur at slightly different times, yielding an inconsistent sample of counter contents. For example, the read from the first byte samples the contents of the entire counter as 03F0H but returns only the least significant byte as F0H; then the read of the second byte samples the contents of the entire counter as 0400H but returns the most significant byte as 04H. The bus interface will assemble these two bytes and return a value of 04F0H, which is incorrect. To eliminate this problem, it is recommended that only the most significant byte be read unless the engine is stopped. This will guarantee a maximum inaccuracy of 100H bytes, because the content of the least significant byte is not known.
3. A single bit returns the status of each FIFO indicating whether less than 1/8th of the FIFO remains unfilled. The driver may check this single bit to determine whether the exact status of the FIFO should be read to determine whether additional command should be written. Note that in normal circumstances the large size of the command/parameter FIFO means it should never get completely filled.

An additional flag byte is provided to indicate that the FIFOs are completely empty. This is useful for the image data FIFO, since this indicates the maximum amount of data that can be written.

6.5.3 2D Rendering Engine Commands

The commands and parameters are programmed via the 64KB memory-mapped control port, and any image data (for operations involving a source bitmap or pattern) is copied to a 32KB image port within 16KB address range (C000h to FFFFh). The parameter registers are at specific addresses, and alternate parameter addresses have additional control information encoded. Parameters written to these specific addresses trigger execution of commands. This mechanism eliminates the need to explicitly program a command.

The parameter addresses and the commands are defined in Section 6.14.1, Equates. The 2D programming examples use Intel assembly language style constructs. All examples assume:

```
fs = CmdPortSelector: selector points to the register base address
es = ImagePortSelector: selector points to the image data buffer base address
```

The 2D commands include:

- Wait Engine Idle, Check FIFO Size, Set Clip Rectangle
- Load Mono Pattern, Load Mono Color, Load General Pattern
- Block fill, Transparent_Text and Opaque Text
- BitBLT with 256 ROPs
- Transparent BitBLT with 256 ROPs
- Line segments

It is assumed that the controller has been set to the desired mode, i.e., the resolution and the color depth have been selected. Note that these commands are operational only in 8BPP, 16BPP or 24BPP modes. The programming is consistent for all color depths. The color information is significant in the lower order bits.

6.5.3.1 Wait Engine Idle

To ensure data integrity, direct read/write to display memory may need to wait for all rendering engine operations to finish.

Sample code: Wait Engine Idle

```
WaitEngineIdle          macro                wSelector, reg
Local    _loop
_loop:
    Mov    reg, wSelector&:[Status2D]
    test   reg, BUSY2D
    jnz   _loop
endm
```

6.5.3.2 Check FIFO size

The 2D engine commands written into the Control Port addresses are stored in a command FIFO. The image data written into the Image Port addresses are stored in the image FIFO. The engine executes these commands and consumes the image data. The application can query the FIFO entries available in eighths of the total allocated size. Both the command FIFO and image FIFO size can go up to 32KB maximum. In Section 6.4.5, Physical Buffer for FIFOs, command FIFO is allocated 16KB as an example. One eighth of allocated buffer is the minimum size to be checked.

Sample code: Check FIFO size

```
WaitCMDFIFO      macro      wSelector, Eighths
Local    _loop
_loop:
    cmp    byte ptr wSelector&:[CmdFIFOStatus], Eighths
    jl    _loop
    endm

WaitIMGFIFO      macro      wSelector, Eighths
Local    _loop
_loop:
    cmp    byte ptr wSelector&:[ImgDataFIFOStatus], Eighths
    jl    _loop
    endm
```

Thus, to wait for 2D command FIFO 4KB (1/4 of 16KB) empty, use this code.

```
mov    fs, CmdPortSelector
WaitCMDFIFO fs, FIFO1_4TH
```

To wait for 2D image FIFO 4KB (1/8 of 32KB) empty, use this code.

```
WaitIMGFIFO    fs, FIFO1_8TH
```

6.5.3.3 Clip Rectangle

SM3110 supports hardware clipping to eliminate the software checking whether the rendering destination's dimension is out of the defined boundary. Note that all bounding coordinates are inclusive.

Sample code: Set Clip rectangle

```
;fs= CmdPortSelector: selector points to the register base address
; lpClipRect = clip rectangle

ClipRectangle    proc    near
    mov     gClipFlag,0                ;global flag, default 0
    cmp     word ptr lpClipRect+2, 0   ;see if clip rect exist?
    je     CR_done                    ;no, exit
    lds     si,lpClipRect              ;ds:si-->clip rect

    WaitCMDFIFO  fs, FIFO1_8TH
    mov     eax,[si]                  ;left and top are inclusive
    mov     edx,[si][4]               ;right and bottom are exclusive
    sub     edx,10001h                ;make them inclusive
    mov     fs:[ClipULXY], eax        ;Load h/w clip registers
    mov     fs:[ClipLRXY], edx
    mov     gClipFlag,F_CLIP          ;save for later use
CR_done:
    ret
Cliprectangle    endp
```

After calling this routine, remember to combine the global clipping flag 'gClipFlag' when setting the parameter "FLAGS".

6.5.3.4 Load Mono Pattern

Loads an 8x8 mono pattern into the internal pattern register. The pattern is stored internally as a color pattern with a 1 in the mono pattern replaced by the color from the Foreground color register and a 0 replaced by the color from the Background color register. This pattern will be used in subsequent Pattern Copy or BitBLT commands that involve a mono pattern.

Sample code: Load Mono Pattern

```

;fs= CmdPortSelector: selector points to the register base address
;es=ImagePortSelector: selector points to the image data buffer base address.
;ds:[esi] points to a 8 byte buffer that contains the 8x8 mono pattern
;eax has the foreground color COLOR0 and ebx has the background color COLOR1

LoadMonoPattern proc      near
    WaitCMDFIFO    fs, FIF01_8TH
    mov     fs:[BGColor],ebx           ;load background color and load
    mov     fs:[FGColor+EXEC_LOAD_MPATTERN],eax   ;foreground color and exe-
                                                cute command
    WaitIMGFIFO    fs, FIF01_8TH
    xor     edi,edi                   ;index into image buffer == 0
    movsd                    ;load 8 bytes of mono pattern
    movsd
    ret
LoadMonoPattern endp

```

6.5.3.5 Load Color Pattern

Loads an 8x8 color pattern into the internal pattern register. This pattern will be used in subsequent Pattern Copy or BitBLT commands that involve a color pattern.

Sample code: Load Color Pattern

```

;fs= CmdPortSelector: selector points to the register base address
;es=ImagePortSelector: selector points to the image data buffer base address.
;ds:[esi] points to a 8x8 color bitmap buffer that contains the mono pattern

LoadColorPattern proc  near
    WaitCMDFIFO    fs, FIF01_8TH
    mov     fs:[EXEC_LOAD_CPATTERN+NOLOAD],0     ;execute command **
    mov     ecx, dwBpp                          ;8/16/24 bpp
    add     ecx, ecx                            ;8*8*bpp/8 /4 for DWORD count=bpp*2
    WaitIMGFIFO    fs, FIF01_8TH
    xor     edi,edi                             ;index into image buffer == 0
    rep     movsd                               ;load the color pattern
    ret
LoadColorPattern endp

```

NOTE: The destination X,Y for the succeeding Pattern Copy or BitBLT can be set while executing the load color pattern command.

6.5.3.6 Load General pattern

For the general case, a Load Pattern routine can be as follows:

Sample code: Load Pattern

```
;lpPattern points to a 8x8 color/mono pattern

LoadPattern      proc      near
    cmp    word ptr lpPattern+2, 0    ;see if pattern exist?
    je     LP_done                    ;no, exit
    lds    si, lpPattern              ;ds:si-->clip rect
    cmp    ds:[si].BitsPerPixel,1    ;check if color or mono
    je     LP_mono
LP_color:
    lea    esi,ds:[si].Color          ;points to start of color pattern
    call   LoadColorPattern
    jmp    LP_done
LP_mono:
    lea    esi,ds:[si].Mono           ;points to start of mono pattern
    call   LoadMonoPattern
LP_done:
    ret
LoadPattern      endp
```

6.5.3.7 Block fill (Opaque Rectangle)

Draws a solid rectangle filled with COLOR1. The rectangle top left corner is located at X1,Y1 with height of HEIGHT1 and width WIDTH1. This command can also be used to do the BitBLT operations with ROP set to WHITENESS (255) or BLACKNESS (0).

Sample code: Block fill (Opaque Rectangle)

```

;fs= CmdPortSelector: selector points to the register base address

OpaqueRectangle proc          near
    WaitCMDFIFO fs, FIFO1_8TH
    Set2WordsToReg32WIDTH1,HEIGHT1,ax                ;eax=(W1,H1), W1 in low
                                                    word
    Set2WordsToReg32X1,Y1,bx                        ;ebx=(X1,Y1), X1 in low
                                                    word
    mov     edx,COLOR1                               ;the color for the fill
                                                    rectangle
    mov     fs:[XYExtents],eax                       ;set height and width
    mov     fs:[DestXY],ebx                          ;load top left x,y
    mov     fs:[BGColor+EXEC_OPAQUE_RECT],edx       ;load color and execute
                                                    command
    ret
OpaqueRectangle endp

```

Note: Throughout this document, the following macro is used to move the source and destination rectangle dimensions into registers.

Sample code: Set 2 WORDs to a 32-bit register

```

Set2WordsToReg32 macro Lo_word, Hi_Word, reg16
    mov     reg16,Hi_Word
    shl     e&reg,16
    mov     reg16,Lo_Word                ;upper-left X,Y coordinate of the rect-
                                        angle(X in low word)
endm

```

6.5.3.8 Transparent Text

The monochrome bitmap that defines the text character is rendered. The image is color-expanded with pixels corresponding to a 1 colored with COLOR1, set in the foreground color register. Pixels corresponding to a 0 are unmodified. The character is rendered with the top left corner at X1,Y1. The character width and height are WIDTH1 and HEIGHT1, respectively. Each new row of the bit-map defining the text begins on the next bit. The last double word written to the image port may have to be padded .

Sample code: Transparent Text

```

;fs= CmdPortSelector: selector points to the register base address
;es=ImagePortSelector: selector points to the image data buffer base address.
;ds:[esi] points to the monochrome character data
;ecx has count double words in the monochrome image = (( (width +7)/8) *
                    height) + 3) / 4 dwords

```

```

TransparentText  proc  near
    WaitCMDFIFO  fs, FIFO1_8TH
    Set2WordsToReg32WIDTH1,HEIGHT1,ax          ;eax=(W1,H1), W1 in low word
    Set2WordsToReg32X1,Y1,bx                  ;ebx=(X1,Y1), X1 in low word
    mov  edx,COLOR1                           ;the foreground color
    mov  fs:[FGColor],edx                     ;set foreground color
    mov  fs:[XYExtents],eax                   ;set height and width
    mov  fs:[DestXY+ EXEC_TRANS_TEXT],ebx     ;load top left x,y & execute cmd
    WaitIMGFIFO  fs, FIFO1_8TH
    xor  edi,edi                               ;index into image buffer == 0
    rep  movsd                                 ;copy mono bitmap
    ret
TransparentText  endp

```

6.5.3.9 Opaque Text

The monochrome bitmap that defines the text character is rendered. The image is color-expanded with pixels corresponding to a 1 colored with COLOR1, the foreground color. Pixels corresponding to a 0 are colored with COLOR0, the background color.

Sample code: Opaque Text

```

;fs= CmdPortSelector: selector points to the register base address
;es=ImagePortSelector: selector points to the image data buffer base address.
;ds:[esi] points to the monochrome character data
;ecx has count double words in the monochrome image = (( (width +7)/8) *
                                     height) + 3) / 4 dwords

OpaqueText proc    near
    WaitCMDFIFO    fs, FIFO1_8TH
    Set2WordsToReg32WIDTH1,HEIGHT1,ax          ;eax=(W1,H1), W1 in low word
    Set2WordsToReg32X1,Y1,bx                  ;ebx=(X1,Y1), X1 in low word
    mov     edx,COLOR1                        ;the foreground color
    mov     edi,COLOR0                        ;the background color
    mov     fs:[BGColor],edi                  ;set background color
    mov     fs:[FGColor],edx                  ;set foreground color
    mov     fs:[XYExtents],eax                ;set height and width
    mov     fs:[DestXY+ EXEC_OPAQUE_TEXT],ebx ;load top left x,y & execute command

    WaitIMGFIFO    fs, FIFO1_8TH
    xor     edi,edi                            ;index into image buffer == 0
    rep     movsd                               ;copy mono bitmap
    ret
OpaqueText endp

```

6.5.3.10 BitBLT

The BitBLT operation involves movement of rectangular blocks of data from the source to the destination on the screen. The source can be on the screen or in a system memory bitmap. If the source is from the system memory, the bitmap can be monochrome or color. The monochrome bitmaps are color-converted based on the foreground and the background colors. If the source bitmap bit is a 1, the foreground color is used to render, and if the bitmap bit is a 0, the background color is used. An 8x8 pattern can be involved in the BitBLT operation. The three operands pattern, source and destination can be combined based on the ternary raster operation specified (ROP). The ternary ROP specifies a combination of AND, OR, XOR and NOT operations between the three operands. The resulting data is rendered into the destination.

BitBLT with ROPs that do not contain source, pattern or destination operands can be rendered using the BitBLT command. The BitBLT operations with only the destination operand are similar to the BitBLT operation with pattern operand described below. All BitBLT operations that contain a pattern operand must first load the monochrome or the color pattern. The load mono pattern or the load color pattern command is used for this operation. BitBLT operations with source operand in local memory (screen) must set up drawing directions in X and Y, if the source and destination rectangles overlap. Default drawing directions for all other BitBLTs are increasing in the X and Y directions.

6.5.3.11 Pattern BLT Without Source Operand

Performs a BitBLT to the specified destination rectangle. The ROP defined for this operation contains only the pattern operand or a pattern and destination combination. The top left of the rectangle is at X1,Y1 and the rectangle has a height HEIGHT1 and a width WIDTH1. The monochrome or color pattern is previously loaded with a pattern load command. The rendered pattern is aligned with the origin 0,0. If only a destination operand is present in the ROP, no pattern is loaded.

Sample code: Pattern BLT

```

;fs= CmdPortSelector: selector points to the register base address
;variable RopValue = ROP function, contains only the pattern, or pattern and
                        destination operands

PatternBlt          proc   near
    call  ClipRectangle          ;set clip rectangle if necessary
    call  LoadPattern            ;Load the pattern to internal buffer
    WaitCMDFIFO  fs, FIFO1_8TH
    Set2WordsToReg32WIDTH1,HEIGHT1,ax          ;eax=(W1,H1), W1 in low word
    Set2WordsToReg32X1,Y1,bx                  ;ebx=(X1,Y1), X1 in low word
    mov   fs:[XYExtents],eax                  ;set height and width
    mov   fs:[DestXY],ebx                    ;load top left x,y & execute command
    mov   ecx,F_XP+F_YP+CMD_BITBLT          ;draw in X left to right, Y top to bot-
                                                tom, blt command
    or    ecx,gClipflag                    ;set clip flag
    mov   cl,RopValue                      ;move ROP into low byte
    mov   fs:[ FLAGS + EXEC_CMD],ecx         ;set flags register & execute
                                                command
    ret
PatternBlt          endp

```

For the pure **pattern copy** (ROP=0F0h) case without clipping rectangle, the last 5 command lines can be combined as following:

```

    mov   fs:[ DestXY + EXEC_PAT_COPY],ebx          ;load top left x,y & exe-
                                                cute command

```

6.5.3.12 Display Memory Source BLT with/without Pattern

Performs a BitBLT to the specified destination rectangle. The ROP defined for this operation contains a source operand or a source and destination combination. There can be an associated pattern operand in each case. The top left of the destination rectangle is at X1,Y1 and the rectangle has a height HEIGHT1 and a width WIDTH1. The source is in local display memory and the top left of the source rectangle is at X0,Y0. If a pattern is present, the monochrome or a color pattern is previously loaded with a pattern load command. The rendered pattern is aligned with the origin at 0,0. If source and destination rectangles overlap, then BLT directions need to be determined in parameter 'FLAGS'.

Sample code: Screen To Screen BLT

```

;fs= CmdPortSelector: selector points to the register base address
;variable RopValue = ROP function, contains only source, or source and destination operands

ScreenToScreenBlt    proc    near
    call    ClipRectangle;set clip rectangle if necessary
    call    LoadPattern;Load the pattern to internal buffer

    WaitCMDFIFO    fs, FIFO1_8TH
    mov     ecx,F_XP+F_YP+CMD_BITBLT+F_COLOR            ;draw X left to right, Y
                                                    top to bottom, blt
                                                    cmd,color src

;;set blt directions
    mov     ax,Y1
    cmp     ax,Y0
    jlsbd_done                ;Y1<Y0, positive direction
    jg     sbd_set_neg        ;Y1>Y0, negative direction
    mov     ax,X1
    cmp     ax,X0
    jlesbd_done              ;Y1=Y0, X1<=X0, positive direction
sbd_set_neg:                ;set negative direction
    orecx,F_XN+ F_YN        ;decreasing x,y
    mov     ax,HEIGHT1
    mov     dx,WIDTH1
    dec     ax                ;HEIGHT1-1
    dec     dx                ;WIDTH1- 1
    add     Y1,ax            ;Y1 = Y1+HEIGHT1-1
    add     Y0,ax            ;Y0 = Y0+HEIGHT1-1
    add     X1,dx            ;X1 = X1+WIDTH1- 1
    add     X0,dx            ;X0 = X0+WIDTH1- 1
sbd_done:
    or     ecx,gClipflag    ;set clip flag
    mov     cl,RopValue     ;move ROP into low byte
    mov     fs:[FLAGS],ecx  ;set flags register

    Set2WordsToReg    32WIDTH1,HEIGHT1,ax    ;eax=(W1,H1), W1 in low word
    Set2WordsToReg    32X1,Y1,bx            ;ebx=(X1,Y1), X1 in low word
    Set2WordsToReg    32X0,Y0,dx            ;edx=(X0,Y0), X0 in low word
    mov     fs:[XYExtents],eax            ;set height and width
    mov     fs:[SrcXY],edx                ;load source x,y and destination
    mov     fs:[DestXY+EXEC_CMD],ebx      ;x,y & execute command
    ret
ScreenToScreenBlt    endp

```

Note:

A screen-to-screen BLT does not necessarily mean from visible area to visible area. The source and/or destination may be from offscreen memory. There are two ways to perform this type of BLT. The first is to specify a Surface Descriptor for the offscreen surface. If the surface to be BLT'ed is not the current one, set the base address, (Ax0h), pixel size (Ax4h) and surface stride (Ax4h), then set the surface descriptor (824h) to that surface index. Do a normal screen-to-screen BLT and the operation will use the specified surface.

In the second BLT method, set the source stride register (804h) to the source's stride in offscreen memory, then set the linear source address register (838h) to the source's offset in offscreen memory. Set the destination using surface descriptors as described above and the BLT using the LinBLT operation (820h[11:08]=6H) instead of the standard BitBLT

6.5.3.13 System Memory Source BLT with/without Pattern

Performs a BitBLT to the specified destination rectangle. The source is image data in system (CPU) memory. The ROP defined for this operation contains a source operand or a source and destination combination. There can be an associated pattern operand in each case. If a pattern is present, the monochrome or color pattern is previously loaded with a pattern load command. The rendered pattern is aligned with the origin at 0,0. The top left of the destination rectangle is at X1,Y1 and the rectangle has a height HEIGHT1 and a width WIDTH1. The source data can be monochrome or color. The image data is moved in one of three(3) ways: as an opaque BLT, as a monochrome transparent BLT, or as a color transparent BLT. Each of these is shown below.

Opaque BitBLT

Usually opaque operation is used with ternary ROP. With a monochrome source, the 1BPP image data is color expanded. Pixels corresponding to a 1 in the source bitmap are colored with the foreground color, and pixels corresponding to a 0 are colored with the background color.

Sample code: Image Data Source BLT

```

;fs= CmdPortSelector: selector points to the register base address
;es=ImagePortSelector: selector points to the image data buffer base address.
;variable RopValue = ROP function, contains only source, or source and destination operands

ImageDataSrcBlt proc near
    call ClipRectangle           ;set clip rectangle if necessary
    call LoadPattern            ;Load the pattern to internal buffer
    WaitCMDFIFO fs, FIF01_8TH

    ;;set command flags portion - begin
    mov ecx,F_IMGDATA+ F_DWORD+CMD_BITBLT ;CPU to screen Blt, mono source
    lds esi,lpSrcDevice         ;point to source device
    cmp ds:[esi].BitsPerPixel,1 ;check if color or mono?
    Jne SB_src_color

SB_src_mono:
    Mov eax,COLOR1             ;load foreground color
    
```

```

Mov     ebx,COLOR0                ;load background color
mov     fs:[FGColor],eax
mov     fs:[BGColor],ebx
jmp     SB_check_src_done
SB_src_color:
or      ecx,F_COLOR+ F_TCOPAQUE   ;source is color bitmap, no transparency
SB_check_src_done:
or      ecx,gClipflag             ;set clip flag
mov     cl,RopValue               ;move ROP into low byte
mov     fs:[FLAGS],ecx           ;set flags register
;;set command flags portion - en

Set2WordsToReg32WIDTH1,HEIGHT1,ax ;eax=(W1,H1), W1 in low word
Set2WordsToReg32    X1,Y1,bx      ;ebx=(X1,Y1), X1 in low word
mov     fs:[XYExtents],eax        ;set height and width and destination
mov     fs:[DestXY+EXEC_CMD],ebx  ;top left x,y & execute command

;calculate the number of dwords to be transferred= ((width*bpp + 31) / 32) *
;                                     height dwords
movzx  eax,ds:[esi].BitsPerPixel
mul    WIDTH1
add    eax,31
shr    eax,5
;note that each row is padded so new rows begin on a fresh dword.  If the image
;is color then F_COLOR bit of the FLAGS has to be set
;In F_DWORD mode, new rows use the next dword of source data, rows ends may
;have to be padded
;if the image is larger then 64K, the transfer is performed in increments of
;the image buffer size

mov     edx,ds:[esi].WidthBytes   ;src rect's bytes per scan line
lea    esi,ds:[esi].BmpData      ;pointer to image data in CPU
mov     ebx,HEIGHT1
SB_loop:
mov     ecx,eax                   ;transfer dword count
Push   esi
WaitIMGFIFO fs, FIFO1_8TH
xor     edi,edi
rep    movsd                       ;transfer the image
pop    esi
add    esi,edx                     ;advance to next line of src data
dec    ebx
jnz    SB_loop
ret
ImageDataSrcBlt  endp

```

Monochrome Transparent BitBLT

To inhibit rendering of pixels corresponding to a 0 bit or a 1 bit in a monochrome source bitmap, set the Mono Source BLT Transparency bits in the FLAGS register to the desired configuration. The following example transfers a mono rectangular image to screen. The monochrome image is color expanded. Pixels corresponding to a 1 in the source bitmap are colored with the foreground color. Pixels corresponding to a 0 are unmodified.

In the “set command flags portion” of sample code above, modify to the following

```
mov     ecx,F_IMGDATA+ F_MBT+F_DWORD+CMD_BITBLT
                                           ;CPU to screen Blt, mono source,
                                           Transparent background

mov     eax,COLOR1
mov     fs:[FGColor],eax                 ;load foreground color
or      ecx,gClipflag                    ;set clip flag
mov     cl,RopValue                      ;move ROP into low byte
mov     fs:[FLAGS],ecx                  ;set flags register
```

Note: By setting the F_MFT bits in the FLAGS register instead of the F_MBT bits, background color is rendered for pixels corresponding to a 0 bit in the source bitmap. The pixels corresponding to the 1 bit in the source bitmap are unmodified. Selecting the F_MNORMAL mode in the FLAGS register performs a straight color expansion, with 0's and 1's colored with background and foreground respectively. The F_MINVERT performs an inverted color expansion, with 1's expanded to background color and 0's expanded to foreground color.

Color Transparent BitBLT

Transparent BitBLT commands are similar to normal BitBLT commands. With a color bitmap as a source, it is possible to inhibit rendering pixels of a given source color. With this configuration selected in the flags register, if source pixels are the same as the compare color set in the background color register, corresponding pixels in the destination are not rendered. The flags can also be set so updates occur only when the color matches. The following example performs a color Transparent BitBLT. The source color bitmap is in the CPU system memory. The compare color is COLOR_COMPARE and has to be loaded into the background color register. The destination is not updated for pixels when source pixel color is equal to COLOR_COMPARE. The destination rectangle is at X1,Y1 with a height HEIGHT1 and width WIDTH1. The ROP is SOURCECOPY.

In the “set command flags portion” of sample code above, modify to the following

```
mov     eax,F_IMGDATA+ F_COLOR+F_DWORD+F_TCSRC+CMD_BITBLT
                                               ;CPU to screen transparent Blt, color
                                               source,Transparent color source

mov     eax,COLOR_COMPARE
mov     fs:[TRANSColor],eax                 ;load compare color
or      ecx,gClipflag                       ;set clip flag
mov     cl,0CCh                             ;move SOURCECOPY ROP into low byte
mov     fs:[FLAGS],ecx                     ;set flags register
```

Note: The transparency function can be applied to the destination by setting F_TCDEST in the flags register instead of the F_TCSRC bits. Destination pixels are unmodified if the destination pixel color equals COLOR_COMPARE the compare color as set in the background color register.

Transparent pattern BLTs can also be performed for BitBLTs involving patterns by selecting F_TCPAT bits in the flags register. The pattern has to be previously loaded. Destination pixels are not updated if the pattern pixel color equals COLOR_COMPARE, the compare color. In summary, color transparent flags can set to F_TCSRC, F_TCDEST or F_TCPAT in the FLAGS register for Source, Destination or Pattern transparency, respectively. Additionally, the F_TBINVERT flag can be set so the sense of the transparency can be inverted. The BitBLT destination is updated only when a color match occurs. Transparent BLTs can also be performed with local display memory (screen to screen) BitBLTs.

6.5.3.14 Line Draw

The SM3110 controller does not have an explicit line draw command. It does provide commands to draw horizontal or vertical intercepts which can be used to draw lines between arbitrary X,Y coordinate pairs. The line drawing can be classified into 8 direction types (eight octants) based on the drawing direction. A single command draws a horizontal line intercept and can position the destination X,Y to the next pixel in the previous or next row, if drawing in X major ($|x_2-x_1| > |y_2-y_1|$). A series of these commands can be used to draw lines which are X major, with X increasing or X decreasing. If the lines are Y major ($|y_2-y_1| > |x_2-x_1|$), the line can be constructed by drawing a series of vertical line intercepts. The destination X,Y after drawing of each vertical intercept can be positioned to the next row (up or down), and to the previous or next column by setting appropriate FLAGS. For efficiency, treat horizontal and vertical lines as special cases.

Draw X Major line

The example below describes a line drawn in the first octant. The start point is X1,Y1 and the end point is X2,Y2 with $|X_2-X_1| > |Y_2-Y_1|$ and $X_2 > X_1$ and $Y_2 < Y_1$.

Algorithm:

```

x = X1;
dx = X2 - X1; dy = Y1 - Y2;
error = 2 dy - dx;
intercept = 0x10000; /* preset y extent = 1, indicates X major */

while (x < X2)
{
    intercept += 1;
    if (error <= 0)
    {
        error += 2 dy;
    } else
    {
        error += 2 * (dy - dx);
        draw_pixels(intercept);
        intercept = 0x10000; /* preset y extent = 1, indicates X major */
    }
    x += 1;
}
if (intercept != 0x10000)
    draw_pixels(intercept);

```

Sample code: Draw X Major line

```

;fs= CmdPortSelector: selector points to the register base address
DwarXMajorLine  proc  near
    call  ClipRectangle                ;set clip rectangle if necessary
    WaitCMDFIFO  fs, FIFO1_8TH
    mov     ecx,F_XP+F_YN+F_CLIP+CMD_LINE    ;Clip on, x positive,
                                           ;y negative, horizontal line

    or     ecx,gClipflag                ;set clip flag
    mov     cl,RopValue                  ;move ROP into low byte
    mov     fs:[FLAGS],ecx                ;set flags register
    mov     eax, COLOR1                  ;the draw color
    Set2WordsToReg32X1,Y1,bx             ;ebx=(X1,Y1), X1 in low word
    mov     fs:[FGColor],eax              ;load color
    mov     fs:[DestXY],ebx               ;set start x,y
    mov     eax,10000h                    ;intercept, preset y extent = 1, indi-
                                           ;cates X major

    mov     si,X1                          ;x = X1
    mov     dx,X2                          ;
    sub     dx,si                           ;dx=X2-X1
    mov     bx,Y1                           ;
    sub     bx,Y2                           ;dy=Y1-Y2
    add     bx,bx                             ;
    mov     di,bx                           ;2*dy
    sub     bx,dx                             ;
    mov     cx,bx                           ;error = 2*dy-dx
    sub     bx,dx                           ;2 * (dy-dx)

DXL_loop:
    cmp     si,X2                          ;while (x < X2)
    jge     DXL_do_remaining                ;{
    inc     eax                               ; intercept += 1;
    cmp     cx,0                             ; if (error <= 0)
    jg     DXL_greter                        ; {
    add     cx, di                             ; error += 2 dy;
    jmp     DXL_compare_done                ; } else
DXL_greter:
    ; {
    add     cx,bx ; error += 2 * (dy - dx);
    mov     fs:[XYExtents+EXEC_CMD],eax      ; draw_pixels(intercept);
    mov     eax,10000h                        ; intercept=0x10000
    ; //reset intercept

DXL_compare_done:
    ; }
    inc     si                               ; x += 1;
    jmp     DXL_loop                          ;}

DXL_do_remaining:
    cmp     eax,10000h
    je     DXL_done                          ; if (intercept != 0x10000)
    mov     fs:[XYExtents+EXEC_CMD],eax      ; draw_pixels(intercept);
DXL_done:
    ret
DwarXMajorLine  endp

```

Note: The last pixel in never drawn. Each new line has to load a start X,Y.

Draw Y Major line

The example below describes a line drawn in the second octant. The start point is X1,Y1 and the end point is X2,Y2 with $|X2-X1| < |Y2-Y1|$ and $X2 > X1$ and $Y2 < Y1$.

Algorithm:

```
y = Y1;
dx = X2 -X1; dy = Y1 - Y2;
error = 2 dx - dy;
intercept = 1;          /* preset x extent = 1, indicates Y major */

while (y > Y2)
{
    intercept += 0x10000;
    if (error <= 0)
    {
        error += 2 dx;
    } else
    {
        error += 2 * (dx - dy);
        draw_pixels(intercept);
        intercept = 1; /* preset x extent = 1, indicates Y major */
    }
    y -= 1;
}
if (intercept <> 1)
    draw_pixels(intercept);
```

Sample code: Draw Y Major line

;fs= CmdPortSelector: selector points to the register base address

```

DwarYMajorLine    proc    near
    call    ClipRectangle                ;set clip rectangle if necessary
    WaitCMDFIFO    fs, FIF01_8TH
    mov     ecx,F_XP+F_YN+ CMD_LINE      ;Clip on, x positive, y negative
    or     ecx,gClipflag                ;set clip flag
    mov     cl,RopValue                  ;move ROP into low byte
    mov     fs:[FLAGS],ecx               ;set flags register
    mov     eax, COLOR1                  ;the draw color
    Set2WordsToReg32X1,Y1,bx            ;ebx=(X1,Y1), X1 in low word
    mov     fs:[FGColor],eax            ;load color
    mov     fs:[DestXY],ebx             ;set start x,y
    mov     eax,1                        ;intercept, preset x extent = 1, indicates Y major
    mov     si,Y1                        ;y = Y1
    mov     dx,si                        ;
    sub     dx,Y2                        ;dy=Y1-Y2
    mov     bx,X2                        ;
    sub     bx,X1                        ;dx=X2-X1
    add     bx,bx                        ;
    mov     di,bx                        ;2*dx
    sub     bx,dx                        ;
    mov     cx,bx                        ;error = 2*dx-dy
    sub     bx,dx                        ;2 * (dx-dy)
DYL_loop:
    cmp     si,Y2                        ;while (y > Y2)
    jle     DYL_do_remaining             ;{
    inc     eax                            ;    intercept += 1;
    cmp     cx,0                          ;    if (error <= 0)
    jg     DYL_greter                     ;    {
    add     cx,di                          ;    error += 2 dx;
    jmp     DYL_compare_done              ;    } else
DYL_greter:
    ;    {
    add     cx,bx                            ;    error += 2 * (dx - dy);
    mov     fs:[XYExtents+EXEC_CMD],eax    ;    draw_pixels(intercept);
    mov     eax,1                            ;
intercept=1                                ;//reset intercept
DYL_compare_done:
    ;    }
    dec     si                              ;    y -= 1;
    jmp     DYL_loop                        ;}
DYL_do_remaining:
    cmp     eax,1
    je     DYL_done                        ; if (intercept != 1)
    mov     fs:[XYExtents+EXEC_CMD],eax    ;
draw_pixels(intercept);
DYL_done:
    ret
DwarYMajorLine    endp

```

Note: The last pixel is never drawn. Each new line has to load start X,Y.

6.6 3D Functions

6.6.1 Overview

A 3D graphics pipeline contains two partitions, the geometry pipeline and the rasterization pipeline (see Figure 6-8, below). The determination of feature set and the trade-off between performance and design cost are made based on the requirements for the target market, and the computational and I/O bandwidth requirement analysis. The analysis assures the load-balancing between a given CPU and the graphic processor, and the performance-balancing between the rendering engine and I/O devices including bus and memory. Based on this analysis, the host CPU performs the geometry portion and the SM3110's 3D engine accelerates the rasterization pipeline as shown in Figure 6-8, below.

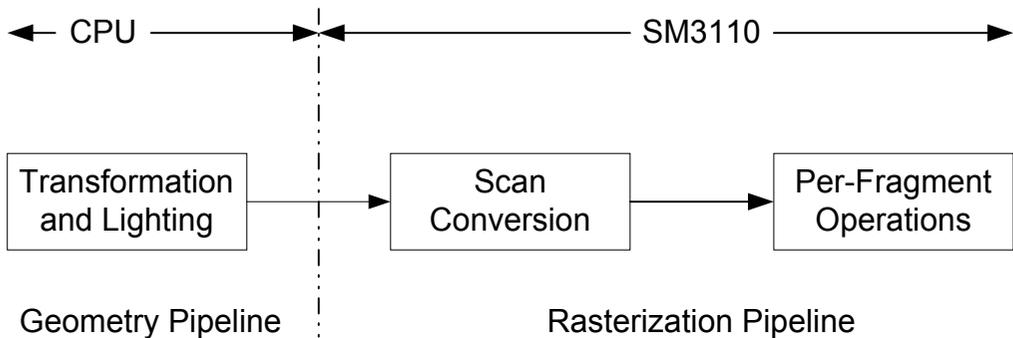


Figure 6-8. 3D Functions Partition

6.6.1.1 Feature Set

The SM3110 3D rendering engine provides high-performance rendering of 3D primitives with a complete feature set of rendering functions:

- Primitives
- Triangle (including setup and sub-pixel displacement)
- Point
- Texturing
- On-chip texture memory
- Perspective correction
- Pixel-level mipmapping
- Filtering: nearest-neighbor, bilinear, trilinear
- Texture transparency
- Texture blending
- Flat and smooth (Gouraud) shading
- Fogging
- Z buffering
- Scissoring
- Stippling
- Alpha blending
- Dithering
- Logic operations
- Double buffering

6.6.1.2 3D Rendering Engine

The 3D driver interfaces with the 3D engine by way of a 3D command FIFO in the frame buffer memory, which allows decoupling of driver and rendering functions. Figure 6-9 shows the block diagram of the 3D engine.

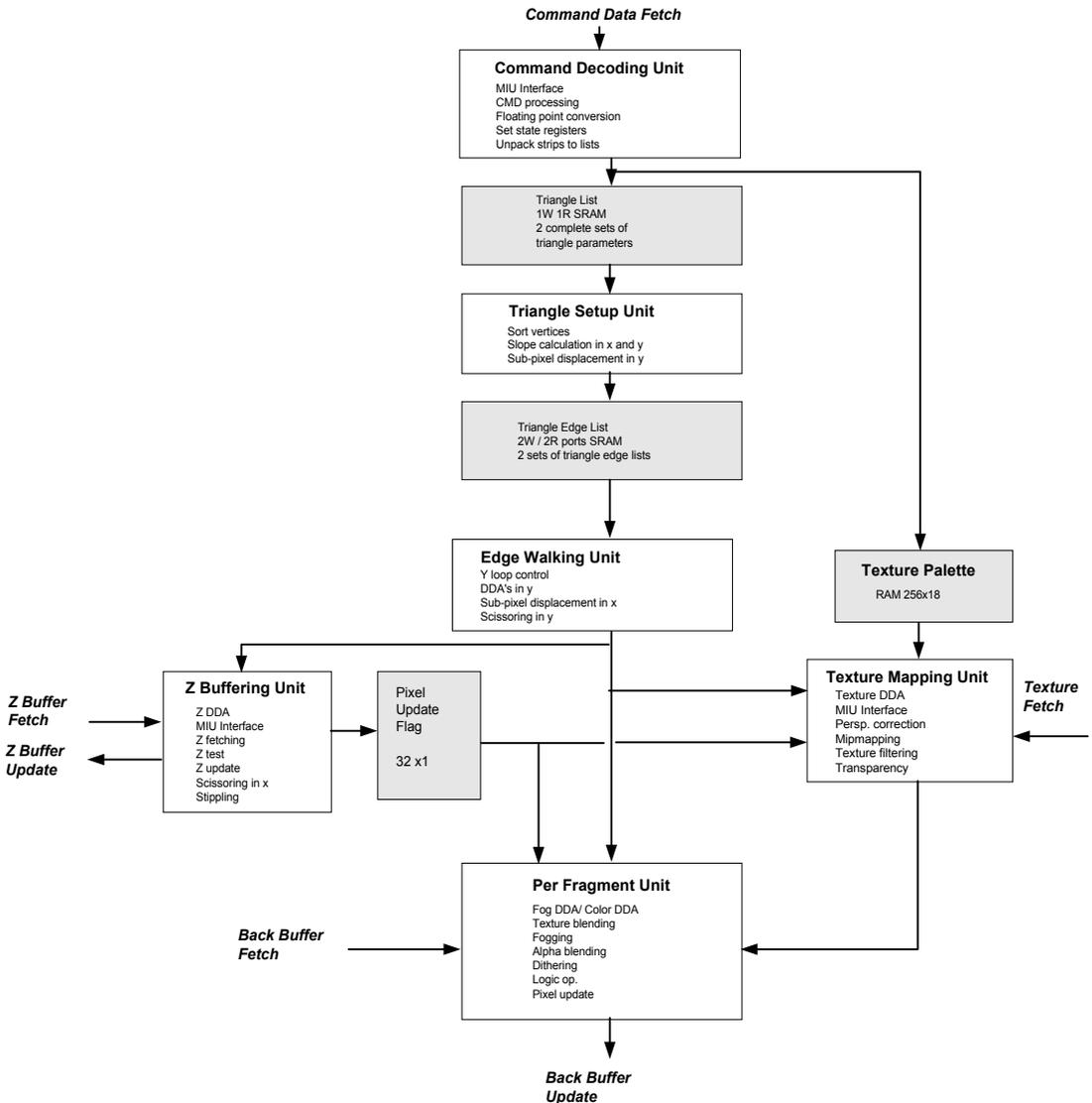


Figure 6-9. 3D Engine block Diagram

6.6.1.3 3D Rasterization Pipeline

The figure below shows functions within the 3D rasterization pipeline.

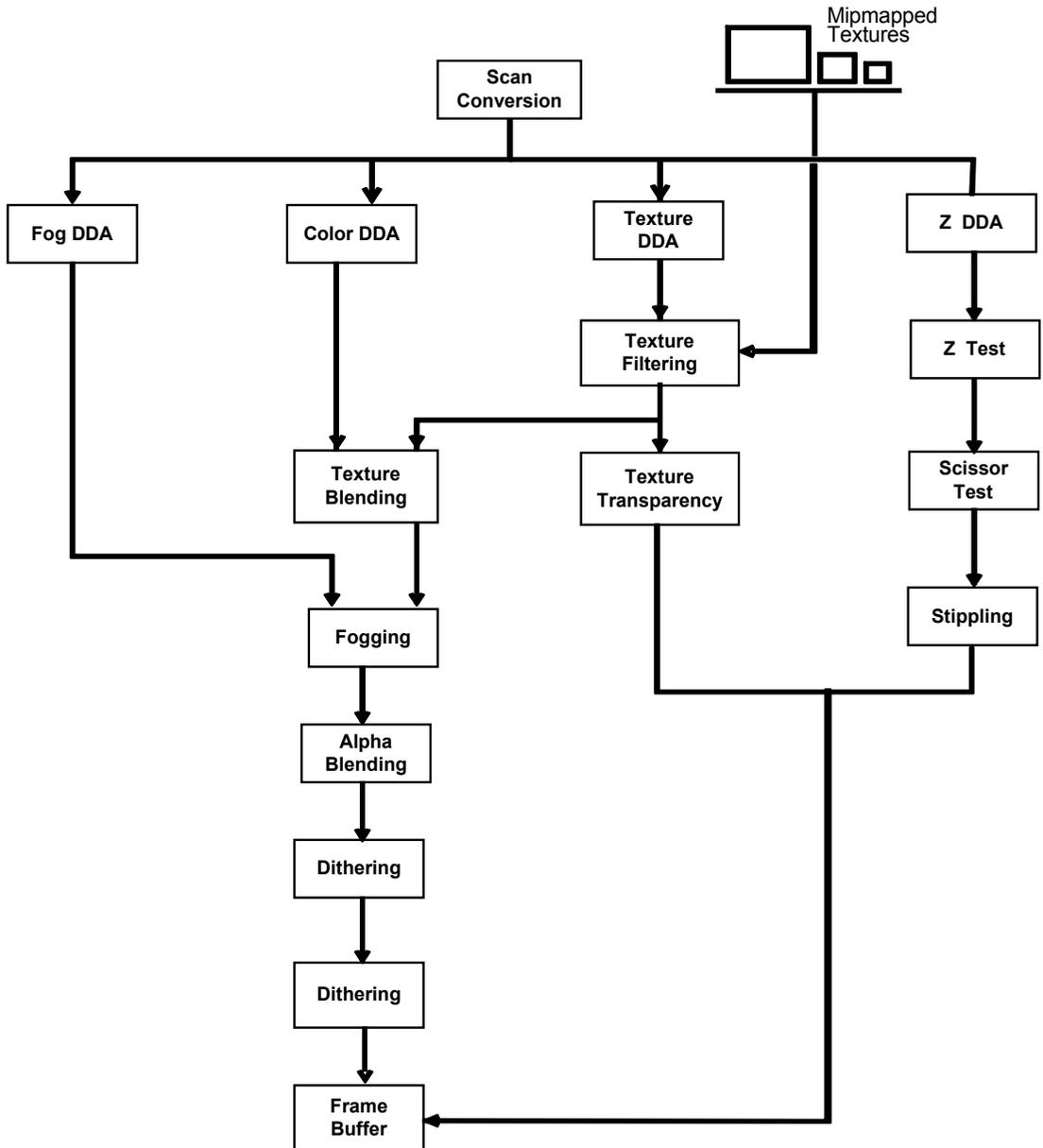


Figure 6-10. 3D Rasterization Pipeline

The order of rendering functions is critical to rendering accuracy and the diagram below represents the 3D pipeline.

<i>scisy</i>	<i>ztst</i>	<i>scisx</i>	<i>stip</i>	<u>tex</u>	<u>tfil</u>	<u>ckey</u>	<u>zwrt</u>	<i>tmod</i>	<i>spec</i>	<i>fog</i>	<i>clmp</i>	<i>blnd</i>	<u>dith</u>	<u>rop</u>
--------------	-------------	--------------	-------------	------------	-------------	-------------	-------------	-------------	-------------	------------	-------------	-------------	-------------	------------

where:

italics – means pixel may be rejected at this step

underline – means only do this step if previous tests pass

stip – stipple test

ztst – z test

tex – texture mapping

tfil – texture filtering

tmod – texture modulation with diffuse color

spec – specular color

fog – apply fog

clmp – clamp results

blnd – blend source with destination

dith – apply dithering pattern

rop – apply raster operation

zwrt – write z value to z-buffer (if enabled)

scisy, *scisx* – scissor in y and x

ckey – color key

(Refer to the rasterization pipeline diagram above.)

1. Scissoring in Y
Failed test = reject pixel (updating interpolants).
2. Perform z-buffer test (only set pass/fail).
3. Scissoring in X (only set pass/fail).
4. Perform stipple test (only set pass/fail).

5. If previous pixel tests pass, do texture mapping, repeat for as many samples as is required by step six.
 - Look up RGB texture value from q, u, v corresponding to x, y.
 - If the texture has an alpha channel, read the A value, otherwise set A to 1.0
6. If previous pixel tests pass, perform bi- or tri-linear filtering operation on RGBA values.
7. If previous pixel tests pass, do alpha related visibility tests.
 - If chroma key transparency is enabled and the RGBA texture color lies inclusively within the chroma key range, set pass/fail for this pixel.
 - If 'alpha test' enabled, set pass/fail for pixel, based on alpha value non-zero/0
8. If previous pixel tests pass, write z value to z-buffer if z-write is enabled.
9. Modulate (see blending in a following step) texture color with diffuse color to get srcColor. If MODULATE_ALPHA is set, do the same for the alpha channel as well. Update color interpolants after this step.
10. Add specular color to srcColor. The alpha channel is not affected by this stage. This stage may produce results greater than 1.0. Update specular color interpolants.
11. Apply RGB fog. This stage may produce results greater than 1.0
12. Clamp colors to range [0.0,1.0).
13. Alpha blend srcColor with destColor to get finalColor.
14. Convert finalColor to destination pixel format, applying dither pattern.
15. Apply raster operation.
newDest = src Rop dest.
16. Write final color to destination at x, y.

6.6.2 3D Engine Operation

6.6.2.1 Initialization

All of the initialization of the SM3110 3D engine is done by the driver. The reason for this is partly because the BIOS doesn't have access to the 3D register space and partly because the driver is more flexible for handling different initialization conditions.

Prior to activation of the 3D driver, the 2D driver has already set up 32-bit linear pointers to the 3D register space, 3D command FIFO port and frame buffer memory. Refer to the 2D section for more information on this procedure. 3D registers are loaded by either using the memory-mapped byte address (minimum 16-bit reads/writes) or via the register load command in the 3D command FIFO (see that section for more details).

On power up, most registers in the SM3110 have a predefined initial state. Refer to the SM3110 register section for more information. The driver can always return the hardware to this initial state by toggling the reset bit in the Command Decode Unit (CDU) control register (0x0038).

Initialization consists of setting up certain of the state registers prior to do 3D rendering. Note that as part of context creation, the driver will initialize virtually all the registers, but the registers below are set up as part of driver initialization, prior to any 3D contexts being created. (Register addresses below are offsets within the 3D control register space of the SM3110 local address space. See the Section 4.2.2, 3D Control, for more details.)

The driver initializes the following registers:

CDU control	+038H	set to stop while initializing other regs, then start
Cmd FIFO base	+198H	internal memory start of command FIFO, 16KB aligned
Cmd FIFO size	+1C8H	size of command FIFO in bytes, 16KB aligned
Cmd FIFO write ptr	+1C0H	set to 0
Cmd FIFO read ptr	+1C4H	set to 0
Texel base	+0E6H	set to 0 (see texture loading for how texture addresses set up)
Palette index offset	+0E4H	set to 0
Texture loading base	+1E8H	set to 0
Texture loading end	+1ECH	set to 0
DRAM control	+1B9H	set according to DRAM parameters (but note that texture cache invalidate bit is used during rendering)
Internal memory timing	+1BAH	set according to DRAM parameters
Address mapping	+1A0H 1A7H	allows memory (in 2MB increments) to be marked as internal/external (n/a for SM3110) or relocated. driver currently direct maps memory.

For context creation, that is, prior to doing 3D rendering, it will be necessary to set up all the registers associated with the desired rendering mode. See the SM3110 register section for more information on the meaning and syntax of these registers. These consist of the following:

Registers applicable to 3D rendering context (offset within 3D Control region)

Modes		
Vertex Format	+000H	defines primitives to be rendered
Control	+0B8H	enables for scissors, texture, perspective correction, mipmapping, subpixel displacement, specular, fog and flat shading
Texture		
Texture Format	+0E0H	format of texture
Texture Map Size	+0E2H	\log_2 of texture height and width
Texture Wrap	+0F4H	texture wrap mode (clamp, repeat, mirror)
Texture Interpolation	+0F8H	alpha test enable, interpolation in uv space (nearest neighbor, bilinear) and between mipmap levels (nearest neighbor, linear)
Texture Transparency Color	+0C0H +0C4H	range of colors for texture transparency
Texture Transparency Enable	+134H	enables texture transparency
Texture LOD Table	+580H - +5A8H	see Texture Loading section for details
Texture Blend	+150H	enable, alpha and color blend modes
Texture Palette	+C00H - +FFCH	256 entry texture palette
Visibility		
Viewport Limits	+080H +082H +100H +102H	viewport top, bottom, left and right edges
Z Buffering	+138H	enable, write enable and z test function
Stipple	+120H	enable and x,y pattern offsets
Stipple Mask	+400H - +47CH	32x32 bit stipple mask
Buffers		
Z Buffer Stride	+130H	stride of z buffer
Back Buffer Stride	+170H	write enable and stride
Pixel Format	+174H	format of back buffer
Back Buffer Base Address	+180H	back buffer address relative to frame buffer start
Z-Buffer Base Address	+188H	Z-buffer address relative to frame buffer start
Pixel Characteristics		
Fog Color	+158H	RGB fog color
Alpha Blend	+15CH	enable, source and destination blend controls
Dither Enable	+160H	enables dither
Logic Op	+164H	enable and logic op

6.6.2.2 3D Command FIFO

The primary interface of the driver with the 3D hardware is through the 3D command FIFO. This includes loading of state registers, primitive commands, synchronization and DMA. Using the command FIFO serializes processing of commands and allows for decoupling of driver and rendering processing. The actual command FIFO is allocated in frame buffer memory (see Initialization), but is written to via a port, which is a fixed offset relative to the Control Address Space of the SM3110. Any commands written to the 4KB 3D command FIFO port are added to the next available location in the 3D command FIFO. Thus, all the FIFO management is handled by the hardware. A diagram of the operation of the 3D command FIFO is shown in the figure below.

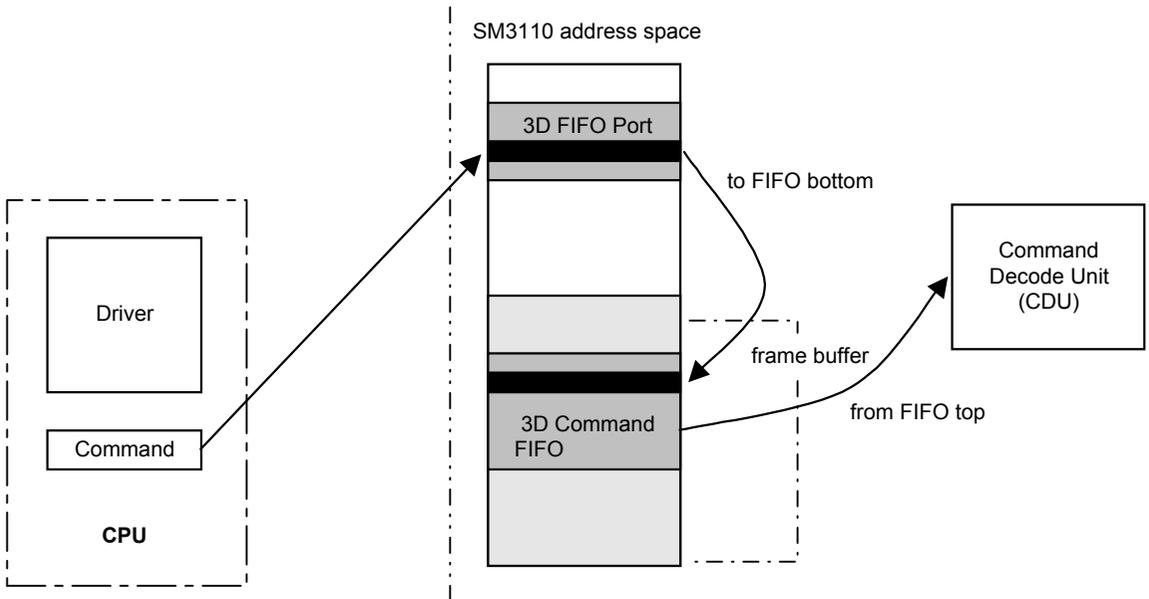


Figure 6-11. 3D Command FIFO

The driver only needs to ensure that sufficient FIFO entries are available prior to writing. This is done as shown in the following code sample:

```
// Read fast status register
regVal = ReadRegister(0x1F0);
// now compute remaining DWORDS in FIFO
remDWORDS = (FIFOSIZE - regVal.fifoStatus + 255)/sizeof(DWORD)
```

where:

FIFOSIZE – total size of 3D command FIFO in bytes

255 – is added because the read is only accurate to 256 byte units

It is recommended that the reads to the FIFO status be minimized by keeping track of the available DWORDS in a local variable and only reading again when this value is depleted.

When writing to the 3D command FIFO port, normal considerations regarding optimal PCI burst lengths should be kept in mind. That is, it may be more efficient to buffer a series of commands in CPU memory and then write them all at once.

It should also be noted that the SM3110 hardware reads the FIFO on 128-bit (4 DWORD) boundaries, which means that alignment must be kept in mind to ensure that the FIFO is able to be emptied, for example, on frame boundaries. In order to do this, 4 NOP command DWORDs should be written to the FIFO at end of frame. This will ensure that no non-NOP commands are left in the FIFO (it doesn't matter if NOP commands are left there).

6.6.3 3D Command Format

The command/parameter format consists of a sequence of 32-bit doublewords:

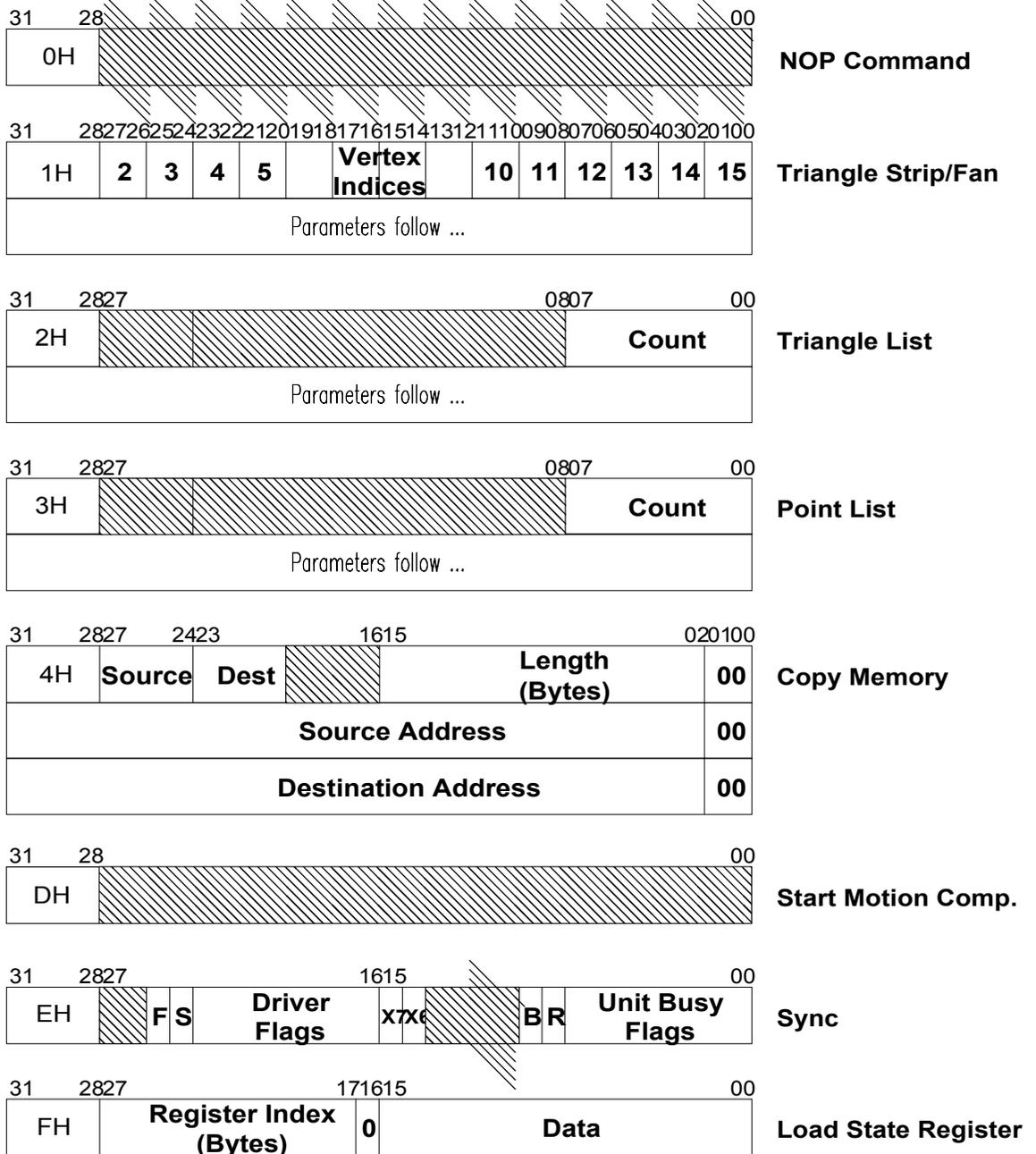


Figure 6-12. 3D Command Formats

31:28 Command	
Value	Semantics
0H	NOP.
1H	Triangle Strip/Fan.
2H	Triangle List.
3H	Point List.
4H	Copy Memory.
5-CH	<i>Reserved.</i>
DH	Start Motion Compensation Command
EH	Control/Sync/control field in immediate data field.
FH	Load State Register.

6.6.4 3D Primitive Commands

The SM3110 handles 3 types of primitives: triangle lists, triangle strips/fans and point lists. Lines are not directly supported but may be constructed in software using triangles. One significant consideration is that the SM3110 requires the extent of primitives to be less than 127 in x and y. Primitives which exceed this limit, must be subdivided in software. Also, backface culling and clipping must be done in software, as this is not done in the SM3110. Vertex order is not important.

All parameters/coordinates are represented in floating point using IEEE single precision format (1.8.23). An optional fixed point (11.21) format is also provided. All color/pixel data is represented as integer 8.8.8.8.

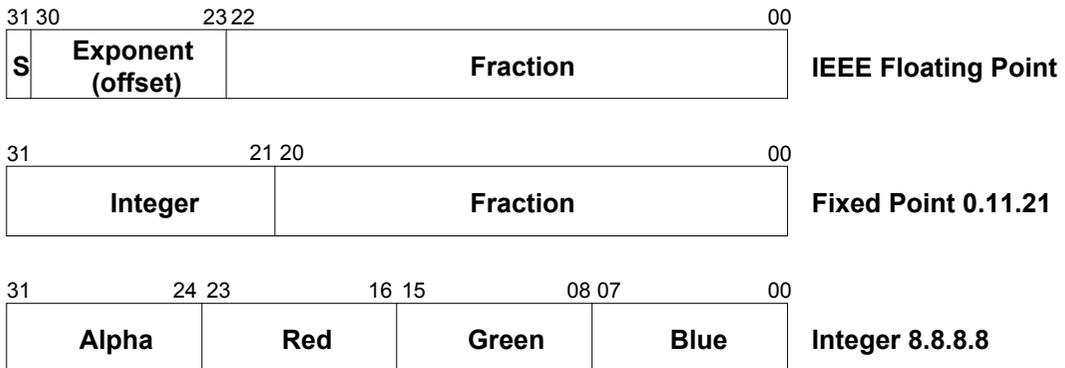


Figure 6-13. Parameter Formats (in Memory)

6.6.4.1 Triangle Strip/Fan

Up to 15 triangles may be specified in one triangle strip/fan list. The last valid vertex is indicated by specifying 0 in the successive vertex number field.

Field	Function
31:28	Command. 1H . The first triangle is implied and specified by the first three vertices.
	Vertex Index. Specifies which vertex should be replaced in a triangle strip or fan for the
27:26	2nd
25:24	3rd
23:22	4th
21:20	5th
19:18	6th
17:16	7th
15:14	8th
13:12	9th
11:10	10th
09:08	11th
07:06	12th
05:04	13th
03:02	14th
01:00	15th triangle .

Value Semantics

1-3	Vertex Number.
0	End.

+4	1st Vertex of 1st triangle.
	2nd Vertex of 1st triangle.
	3rd Vertex of 1st triangle.
	Successive vertices.

The offset of the first parameter of each successive vertex is

$$(3+n-2) \times \text{Size}_{\text{VERTEX_PARAMETER_LIST}} + 4,$$

where n is the triangle number (starting from 1).

6.6.4.2 Triangle List

Field	Function
31:28	Command. 2H .
27:08	Reserved.
07:00	Number of Triangles.

6.6.4.3 Point List

Field	Function
31:28	Command. 3H .
27:08	Reserved.
07:00	Number of Points.

Vertices can contain different numbers of elements based on the setting of the bits in the vertex format register (see 4.2.2.1, Vertex Format). The table below shows valid combinations:

	texture type	fog	specular	coordinate	color
x				xy,xyz	
y				xy,xyz	
z				xyz	
color					rgb,rgb
specular		fog	specular rgb		
u	uv,uvw mip				
v	uv,uvw mip				
u/w	uvw,uvw mip				
v/w	uvw,uvw mip				
rhw	uvw,uvw mip				

Some special considerations should be made with regard to some of the primitive characteristics. These are noted below:

flat shading – SM3110 implements this by assuming that the color and alpha are the same in all 3 vertices and then not interpolating between them.

fog – the fog values in the vertices are treated as though a value of 0x00 is no fog and a value of 0xFF is fog color only

z scaling – the range of z values expected by the hardware is from 0 to 65535.99999.

alpha test – the SM3110 doesn't support a full alpha test, but it does support the alpha != zero case by setting the bit in the texture interpolation control register (+0F8H).

texture uv and wrap – the SM3110 has 10 integer bits for u,v addressing. This includes the texture width/height (which must be a power of 2, but not necessarily square) and any wrap bits. Thus, a 64-textel wide texture can have a wrap of 16 in the u direction (and similarly for v). Note that u,v coordinates supplied with primitives must be pre-multiplied by the texture width/height.

6.6.5 Copy Memory and DMA Command

The SM3110 3D engine has DMA capability in the form of the copy memory command (0x4) for the 3D command FIFO. This consists of a source and destination code and addresses, and a length in doublewords. Some useful ways in which this command can be used include:

- load local memory from PCI or AGP space (e.g., for texture DMA)
- pattern fill local memory from register (does 128-bit operations)
- set driver value in PCI or AGP space (by loading value in fill register, then transferring to local memory and from local memory to PCI or AGP space) for flagging when command FIFO operations are complete

The table below specifies the source and destination addresses and transfer length for the internal DMA. All internal addresses are 23 bits relative to the PCI base address, all external (AGP/PCI) addresses are linear 32 bits.

	<i>field</i>	<i>function</i>
+0	31:28	Command. 4H .
	27:24	Source Flags.
	23:20	Destination Flags.
	<i>value</i>	<i>semantics</i>
	0H	Internal non-aligned. Within SM3110 internal embedded DRAM.
	1H	Internal, aligned to natural 16B boundary.
	2H	State Registers except for Palette. Destination only .
	3H	Palette. Used to load palette RAM, 18b per clock. Destination only .
	4H	AGP/PCI Linear.
	5H	<i>Reserved</i> .
	6H	32b Fill Register. Source only . Used to fill memory.
	7H	Internal. Translate (texture) addresses. Destination only .
	8-FH	<i>Reserved</i> .
	<i>bits</i>	<i>field</i>
	19	Transfer Request. Must Be One for transfer to occur.
	18:00	Length in doublewords-1
+4	31:00	Starting Source Address
+8	31:00	Starting Destination Address.

When the palette is the destination, bits [09:02] of the destination address specify the palette index. Bits [01:00] specify the bytes within the palette and should be set to 0, byte 0 is Blue, byte 1 is Green and byte 2 is Red, byte 3 is not used. The source (internal or external) and destination palette addresses must be doubleword aligned. The matrix of source and destination address spaces is shown below:

Destination	Source			
	<i>0-Internal Non-aligned</i>	<i>1-Internal Aligned</i>	<i>4-External AGP/PCI</i>	<i>6-Fill Register</i>
0 - Internal Non-aligned	Yes	No	Yes	Yes
1 - Internal Aligned	No	Yes	No	Yes
2 - State Registers	Yes	No	No	No
3 - Palette RAM	Yes	No	No	No
4 - AGP/PCI	No	No	No	No
7 - Internal Texture	Yes	No	Yes	No

Table 6-3. DMA Transfer Source/Destination Matrix

Restrictions:

Transfers performed between Internal Aligned source and destination will be done at a 16 bytes per transfer.

Transfers between the 32-bit Fill Register and Internal Aligned destination will be done at 16 bytes per transfer.

The palette RAM cannot be the destination addressed through the State Register space; it must be accessed only through its own dedicated space.

All other transfers are performed at 4 bytes per transfer.

When setting up DMA operations in the 3D command FIFO, care should be taken with regard to other commands in the FIFO. Since there is only one DMA engine, each subsequent DMA command will wait until the previous DMA command is complete; that's not true for other, non-DMA commands. It may be necessary to put a sync command in the FIFO after the DMA command to wait until the DMA engine is idle before processing subsequent commands in the FIFO. For example, this would be the case if a DMA was being used to load a texture and the next command in the FIFO was a primitive which used the texture. See the synchronization command section for information on how this is handled.

Note that when DMAing in PCI space, physical addresses will be needed in the address field. If a transfer larger than a physical page (4Kbyte) is desired, simply use multiple DMA commands. If it is necessary to wait for this chain of DMAs to be done, it is only necessary to put a sync command after the last DMA. This could be used for texture loading, as described in the section on texture loading.

6.6.6 Synchronization Command

Synchronization is performed by the appropriate use of the Sync command. A conditional wait is performed on specific conditions set by internal and external hardware or driver software. When the conditions are met the wait terminates and command interpretation and execution resumes. The driver specifies a set of conditional flags that may be tested. A mask function performs the following operation:

$$\text{RESULT} = \text{OR}_{i=23 \text{ to } 00} [\text{Mask}_i \text{ AND Flag}_i]$$

When the condition becomes “true” the wait terminates.

The wait condition is defined as follows

```
condition = DrvFlg3 & cmdreg[19] |  
           DrvFlg2 & cmdreg[18] |  
           SigI2D & cmdreg[17] |  
           Sync2D & cmdreg[16] |  
           (fc_rbusy | fc_wbusy) & cmdreg[11] |  
           mcmem_busy & cmdreg[10] |  
           busy & cmdreg[9] |  
           bd_pipe_busy_m & cmdreg[7] |  
           fd_pipe_busy_m & cmdreg[5] |  
           zd_pipe_busy_m & cmdreg[4] |  
           (td_ag_pipe_busy_m | td_tp_pipe_busy_m) & cmdreg[3] |  
           ed_pipe_busy_m & cmdreg[2] |  
           sd_pipe_busy_m & cmdreg[1] |  
           cdu_busy & cmdreg[0];
```

```
CDU stall = cmdreg[25] & (cmdreg[24] & ~condition or ~cmdreg[24] & condition)
```

Some of the useful events are covered in the sections on DMA, 2D/3D synchronization and performance issues. In addition to these, it is possible to sync on driver flags which are set in registers 0x024 and 0x026, or on external event pins.

Synchronization Command EH

+0	<i>Field</i>	<i>Function</i>
	31:28	Command. EH .
	27:26	<i>Reserved</i> .
	25	Function.
		<i>Value</i> <i>Semantics</i>
	0	NOP. continue command interpretation.
	1	Sync: wait until condition, then continue command interpretation.
	24	Condition Sense.
		<i>Value</i> <i>Semantics</i>
	0	Not Result.
	1	Result.
	<i>Field</i>	<i>Function</i>
	23:20	<i>Reserved</i> .
	19:18	DrvFlags. Correspond to the Driver Flags in CDU Register bytes 027H - 024H
		<i>Value</i> <i>Semantics</i>
	0	Driver Flag Byte Bit 0 is '0'.
	1	Driver Flag Byte Bit 0 is '1'.
	<i>Field</i>	<i>Function</i>
	17	2D Signal
		<i>Value</i> <i>Semantics</i>
	0	Signal to 2D is low
	1	Signal to 2D is high
	<i>Field</i>	<i>Function</i>
	16	2D Synchronization
		<i>Value</i> <i>Semantics</i>
	0	2D has not signalled 3D
	1	2D has signalled 3D

Synchronization Command EH continued

<i>Field</i>	<i>Function</i>
15:12	<i>Reserved.</i>
11	Format Conversion.
10	Motion Compensation.
09:00	Hardware Status Mask.
09	Busy.
08	Running.
07	Direct Memory Access.
06	<i>Reserved.</i>
05	Per Fragment Unit.
04	Z-Buffering Unit.
03	Texture Mapping Unit.
02	Edge Walking Unit.
01	Triangle Setup Unit.
00	Command Decode Unit.

<i>Value</i>	<i>Semantics</i>
0	Not Busy.
1	Busy.

6.6.7 Load State Register Command

Register loading is supported by the load state register command. Using the 3D command FIFO to load registers is primarily useful because it allows serialization of state with primitives. The format is shown below.

This command is used to load state and control registers, and internal arrays.

	<i>Field</i>	<i>Function</i>
+0	31:28	Command. FH .
	27:16	Register Index. Bit [16] must be 0.
	15:00	Data

6.6.8 Texture Loading

The SM3110 requires textures to be stored in an optimized format in frame buffer memory. A number of texture formats are supported (see 6.6.10, Buffer Issues) and mipmaps are supported, with the restriction that the mipmaps (as well as the start of non-mipmapped textures) begin on 2KB byte boundaries (although sharing of lower level mipmaps within a single 2KB block is possible, as described below).

The basic mechanism for loading textures involves setting up certain registers and then transferring the texture data, either by memory-mapped write to frame buffer memory, or via DMA (see DMA section). This is done using the Load State Register command. The following registers must be set up for texture loading:

texel size	0x1E0	texture format
texture base	0x1E8	base address for texture translation (s/b 0)
texture end	0x1EC	end address for texture translation (s/b 4MB)
tex param (shift)	0x1E4	$\max(0, \log_2 u - bsxe)$ where: $\log_2 u$ is \log_2 of texture width $bsxe$ is based on texture format bit width (see Table 6-5)
tex param (row V shift)	0x1E4	$\max(0, (\log_2 u - \log_2 uoffset)) + rowVShiftOffset$ where: $\log_2 u$ is \log_2 of texture width $\log_2 uoffset$ is based on texture format bit width (see table) $rowVShiftOffset$ is based on texture format bit width (see Table 6-5)
tex param (U page offset)	0x1E4	offset from start of page (normally 0)
tex param (V page offset)	0x1E4	offset from start of page (normally 0)
tex param (page address)	0x1E4	destination address in 2K byte units

Table 6-4. Texture Loading Registers

	1 bit	2 bit	4 bit	8 bit	16 bit
bsxe	5	4	3	2	1
$\log_2 uoffset$	7	7	6	6	5
rowVShiftOffset	0	1	1	2	2

Table 6-5. Texture Loading Register Values

When loading textures, note that the actual destination address goes in the texture parameters' page address, but the destination for memory-mapped write or DMA purposes should always be the start of frame buffer memory. This allows access to the entire 4MB internal memory space in the SM3110. The address translation done by the SM3110 loading will then translate and correctly load anything sent between its base and end addresses (which is why they should be 0 and 4MB, respectively). Normally, textures (or mipmap levels of textures) will start at the beginning of the 2KB page, but in some cases, the driver will share the 2KB page of the lowest level mipmap which doesn't take an entire page. In this case, the u and v page offsets can be used to specify the starting point.

When primitives are rendered using textures, similar information is required to be loaded into the state registers, including texel format (0x0e0) and a similar value to the texture parameter register above, but with an entry for each mipmap level (texture LOD table, 0x580-0x5a8). In addition, a texture size register is required (0xe2), with the \log_2 values of the texture width (u) and height (v).

It is also necessary to be aware of the texture cache invalidate bit (0x1b9), which will need to be toggled if the texture cache contents are no longer valid. This could happen if the contents of an existing texture are reloaded.

Texture palettes are loaded using the load state register 3D command FIFO command. For SM3110, texture palette colors are 6 bits each (R,G,B), so an 18-bit data field is used for this register load command (vs. other commands, which have a 16-bit data field).

6.6.9 2D/3D synchronization

Because the SM3110 has separate FIFOs for 2D and 3D, a mechanism is provided to ensure synchronization between the two. The capability of adding a synchronization command to either FIFO, which waits for a signal from the other is available. The basic mechanism is as follows:

```
// To have 2D wait on 3D, add the following to the 2D command FIFO:
2D command 0xd

// To have 2D signal 3D, add the following to the 2D command FIFO:
2D command 0xc

// To have 3D wait on 2D, add the following to the 3D command FIFO:
3D command 0xe, sync2D = 1, function = 1, sense = 1
load state register 0x20 with 0 // this is to clear 2D signal after rcvd

// To have 3D signal 2D, add the following to the 3D command FIFO:
load state register 0x22 with 1
load state register 0x22 with 0 // this is to clear 3D signal after sent
```

These synchronization commands are useful for events which are anticipated, for example on a once-per-frame basis. If 2D commands were used to clear the z-buffer (actually there are more efficient 3D commands for this, see DMA section), the 2D blt command could be followed by a signal3D command in the 2D command FIFO and a waiton2D in the 3D command FIFO. That way, no 3D rendering commands would be processed until the z-buffer BLT was complete.

More often, it is necessary to ensure that one or the other engine is inactive before proceeding. In this case, it is necessary to wait for the corresponding FIFO to be empty, as shown below:

```
// To wait for 3D engine to be idle and FIFO empty:
flush 3D FIFO // add 4 null commands to FIFO, see 3D command FIFO desc.
poll 3D engine busy bit of register 0x1F0 until clear

// To wait for 2D engine to be idle and FIFO empty:
poll 2D engine busy bit of register 0x8F0 until clear
```

6.6.10 Buffer Issues

6.6.10.1 Buffer Alignment

The following table shows required alignments for 3D buffers on SM3110:

Buffer	Alignment	Pitch Alignment
front buffer/back buffer	4K bytes	16 pixels
z-buffer	4K bytes	16 pixels
3D command FIFO	16K bytes	n/a
textures	2K bytes	n/a

6.6.10.2 Buffer Formats

The following table shows available formats for buffers on SM3110:

Buffer	Formats
front buffer/back buffer	RGB0555, RGB565, ARGB4444, ARGB1555, RGB888, ARGB8888
z-buffer	16 bit
textures	CLUT1, CLUT2, CLUT4, CLUT8, RGB0555, RGB565, ARGB4444, ARGB1555

6.6.11 Performance Issues

A number of the performance issues in the SM3110 3D engine are related to general 'good practice' for 3D applications. This includes minimizing state changes and minimizing texture loading. For state changes, the 3D engine only has one set of state registers, so it is necessary to make sure the 3D pipeline is idle before modifying the state registers. This is done by adding the following command to the 3D command FIFO:

```
// To wait for 3D pipeline idle  
control/sync command with sync bit and bits for Per Fragment Unit, Z-Buffering  
Unit, Texture Mapping Unit, Edge Walking Unit, Triangle Setup Unit and Command  
Decode Unit set
```

Another bottleneck could come from the fact that the 3D engine will stall if it detects that the bounding boxes for 2 triangles overlap, in order to prevent read-after-write consistency errors. This function can be enabled under software control (register 0x004). If it is known that 2 triangles will not overlap (for example in a strip/fan) this function should be turned off.

6.7 LCD Panel Programming

The SM3110 supports an LCD flat panel display. The supported panel sizes are 640x480, 800x600, 1024x768, and 1280x1024. Both DSTN and TFT panels with different color bits are also supported. The following block diagram shows the display subsystem of SM3110.

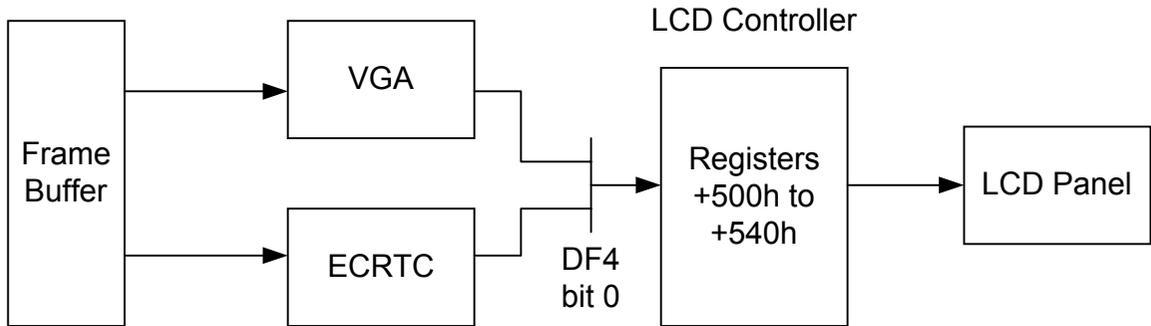


Figure 6-14. LCD Display Subsystem

6.7.1 LCD Flat Panel Registers

The following registers are LCD flat panel related. These are offsets within the 2D Control Address (see Section 4.2.1, 2D Control).

LCD Type and Miscellaneous - +510h

Bits 2-0 is the panel stretch ratio used for standard VGA modes. Values must be found using trial and error. When stretching is not needed, 0 is programmed.

Bits 7-4 define the panel type. The SM3110 family currently supports 7 types of panels.

Other bits need to be programmed according to the data sheet of the particular LCD panel.

LCD Image Compensation – +504h

Bit 7 and bit 5 enable or disable stretching for vertical and horizontal, respectively. This register will take effect only in standard VGA modes.

Dithering and Frame rate Modulation - +514h

Bit 7 and 6 define the number of frames used for frame rate modulation. More frames tend to result in better quality. Bits 5-3 control the dithering and are used if the color depth of the current mode is greater than that of the panel. Again, different values should be tried out to determine the best display quality for each particular panel.

DSTN Buffer Starting Address – +520h

When a DSTN panel is selected, some additional memory in the frame buffer is required to store a half-frame of the panel display data. This register specifies the starting address from the beginning of the frame buffer. The formula for the size required is as follows.

$$\text{Size} = \text{Panel Width} * \text{Panel Height} * 3^1 / 8^2 / 2^3$$

1. DSTN is 3 bits per pixel.
2. There are 8 bits per byte.
3. Half frame.

LCD CRTC Registers – +530h ... +53Ch

These registers contain the same values from VGA CRTC or ECRTC except for register 534 bits 8- 0 and register 53C bits 10-0, which are panel width and height, respectively. Note that the horizontal parameters are counted by characters, that is, they need to be divided by 8.

LCD Adjustment Control – +540h

This register is used to fine-tune the display image size and centering. Each panel has different characteristics and needs to be tuned to get the best display image size and positioning.

ECRTC Stretch Register – +EE4h

This register is used to stretch the display image under enhanced mode. Each value represents a different stretching ratio. There is a small, fixed set of stretch ratios defined for vertical and horizontal stretching. The closest value is picked to achieve the maximum stretch ratio, i.e., the largest possible image for the given panel resolution.

6.7.1.1 Initialization

During power up, only register 510 (LCD Type and Misc. Register) needs to be set according to the panel. If the panel is DSTN, register 520 (Base Address Register) needs to be programmed, too. Sufficient frame buffer memory needs to be reserved for DSTN panel to display properly.

6.7.1.2 Set to Desired Mode

Setting to a desired mode requires programming LCD-related registers as well as VGA CRTC or ECRTC registers. DCLK (dot clock – register 90, 91, and 92) needs to be programmed accordingly, too.

6.7.1.3 Standard VGA Modes

To set to standard VGA modes, the standard VGA CRTC registers are programmed as usual. Registers +530h to +53Ch need to be programmed according to the panel size. Use register 504 to enable or disable stretching. If stretching is disabled, the display image will be centered by default. Also use registers 530 to 540 to adjust the display image to desired position.

6.7.1.4 Enhanced Modes

No matter which enhanced mode, the timing parameters of the panel resolution have to be programmed into ECRTC registers as well as registers 530 to 53C. Refresh rate has to be 60Hz. For example, a panel with size of 1024x768 needs to use the parameters from the 1024x768 60Hz portion of the table. Any mode with resolutions higher than the panel size can't be set.

To center the display image, view port registers (E30 to E34) need to be programmed by the following formula:

$$\text{View port start} = (\text{panel size} - \text{real resolution}) / 2$$

$$\text{View port end} = \text{view port start} + \text{real resolution}$$

To stretch the display image, register 504 has no effect. Instead, register EE4 is used. The following tables list all the possible stretch ratios to be programmed.

Table 6-6. Horizontal Stretch Ratio

Mode Width	Panel Width	Ratio	Result Width	Program Value
640	800	4 -> 5	800	3
640	1024	5 -> 8	1024	4
720	1024	3 -> 4	960	2
800	1024	25 -> 32	1024	7
640	1280	1 -> 2	1280	1
720	1280	75 -> 128	1230	8
800	1280	5 -> 8	1280	4
1024	1280	4 -> 5	1280	3

Table 6-7. Vertical Stretch Ratio

Mode Height	Panel Height	Ratio	Result Height	Program Value
400	600	3 -> 4	533	2
480	600	4 -> 5	600	3
400	768	75 -> 128	682	8
480	768	5 -> 8	768	4
600	768	25 -> 32	768	4
400	1024	15 -> 32	853	5
480	1024	15 -> 32	1024	5
600	1024	75 -> 128	1024	8
768	1024	3 -> 4	1024	2

For some resolutions, it is not possible to stretch to the exact panel size. In this case, the closest ratio is picked. It is still necessary to program viewport registers to center the image after stretching. Again, registers 530 to 540 can be used to fine-tune the display image to the desired position.

6.7.1.5 Set Mode Sequence

The following describes the sequence of set mode:

```
Turn off screen through sequence register - 3C4 index 1 bit 5
If(Standard VGA Mode)
{
  Set standard VGA registers - 3C4/5, 3D4/5, 3C0/1, 3CE/F, etc. according to
  the mode
  Initialize palette if necessary - 3C8, 3C9
  Turn on or off stretching - 504
}
else// Enhanced mode
{
  Disable ECRTC0 - program 0 to DF4
  Define a surface for display - A00...AF0, A04...AF4
  Program channel surface index register - B34
  Program display format - D30
  Program enhanced dot clock - 090, 091, 092
  Put correct timing parameters into ECRTC registers - E80... E94
  Program display view port registers - E30...E34
  Initialize palette if necessary - 3C8, 3C9
  Program stretch register according to Table 6-6 and Table 6-7 - EE4
  Enable ECRTC0 - Program 5 to DF4
}
Program LCD dithering register - 514
Program LCD timing registers - 530...53C
Use LCD Adjustment Control Register to center the display image - 540
```

6.8 Dual View

6.8.1 Overview

The SM3110 supports two display subsystems. They can be set up to display the same content from the same frame buffer location. This is called simultaneous view. They can also be set up to display the same or different portions of frame buffer with different refresh rates and color depths. This is called dual view. (Support for dual view is provided partly by the SM3110 hardware and partly by the host operating system; if the operating system does not allow a single graphics chip to drive two displays, this feature is not available.) The following diagram shows the relationship of these two display subsystems.

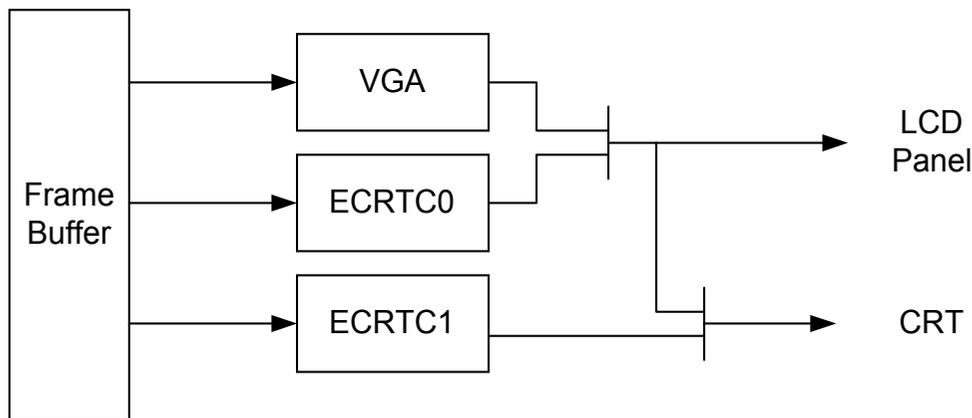


Figure 6-15. Dual View Mechanism

6.8.2 Enabling Second Display

Enabling the second display involves the following registers:

Second Dot Clock – 1090, 1091, 1092

Correct clock rate needs to be set for different modes and refresh rates.

Surface/Channel/Format registers – A00...AF0, B84, 1D30

A different surface memory has to be selected and programmed. Channel index register B84 has to be programmed to the selected surface index. Also register 1D30 (display channel format) needs to be programmed to the correct format according to the mode.

Display 0 and 1 control registers – DF4 and 1DF4

Set bit 4 of DF4 to 1 to hook the second display output to ECRTC1. Also set 1DF4 to 5 to enable ECRTC1.

Double Scan and Sync Polarity Registers – 1DF5, 1DF6

These two registers control both horizontal and vertical double scan and polarity according to each mode.

ECRTC1 View Port and Timing registers – 1E00...1E94

ECRTC1 supports only CRT, so no LCD consideration is involved. These registers can be set to different resolution and refresh rates.

6.8.2.1 *Enable Simultaneous View*

Bit 4 of Display 0 Control Register (DF4) enables dual view. Set it to 0 will enable simultaneous view. Everything displaying on the first display will show up on the second display, too.

6.8.2.2 *Enable Dual View*

The following sequence will enable dual view:

```
Set to simultaneous view - turn off DF4 bit 4
Disable ECRTC1 - program 0 to 1DF4
Define a surface - A00...AF0, A04...AF4
Program channel surface index register - B84
Program second display format - 1D30
Program second dot clock - 1090, 1091, 1092
Put correct timing parameters into ECRTC registers - 1E80... 1E94
Program second display view port registers - 1E30...1E34
Enable ECRTC1 - Program 5 to 1DF4
Enable dual view - turn on DF4 bit 4
```

6.9 Motion Compensation

The SM3110 supports Motion Compensation (MC) hardware acceleration when displaying MPEG-2 video data. While the graphics chip is performing motion compensation, the rest of video decoding (variable length decoding, inverse scan, inverse quantization and Inverse Discrete Cosine Transform (IDCT)) is done by a software client (e.g., SoftDVD®).

6.9.1 Overview

MPEG-2 uses three kinds of picture storage methods. Intra (*I frames*), Predicted (*P frames*) and Bi-directional (*B frames*). Intra frames are frames coded as standalone images, very much like JPEG pictures. They allow random access points within a video stream and create a reference frame from which other frames are built. In general, an I frame will occur around twice a second. Predicted frames contain motion vectors describing the difference from the closest previous I frame or P frame. This forward prediction allows for greater compression than with I frames. Bi-directional frames are like P frames in that they use motion vectors, but with B frames, motion vectors are generated by looking both forward and backward. This forward and backward referencing allow B frames to have the greatest compression ratio. Figure 6-16 shows the data flow during this process.

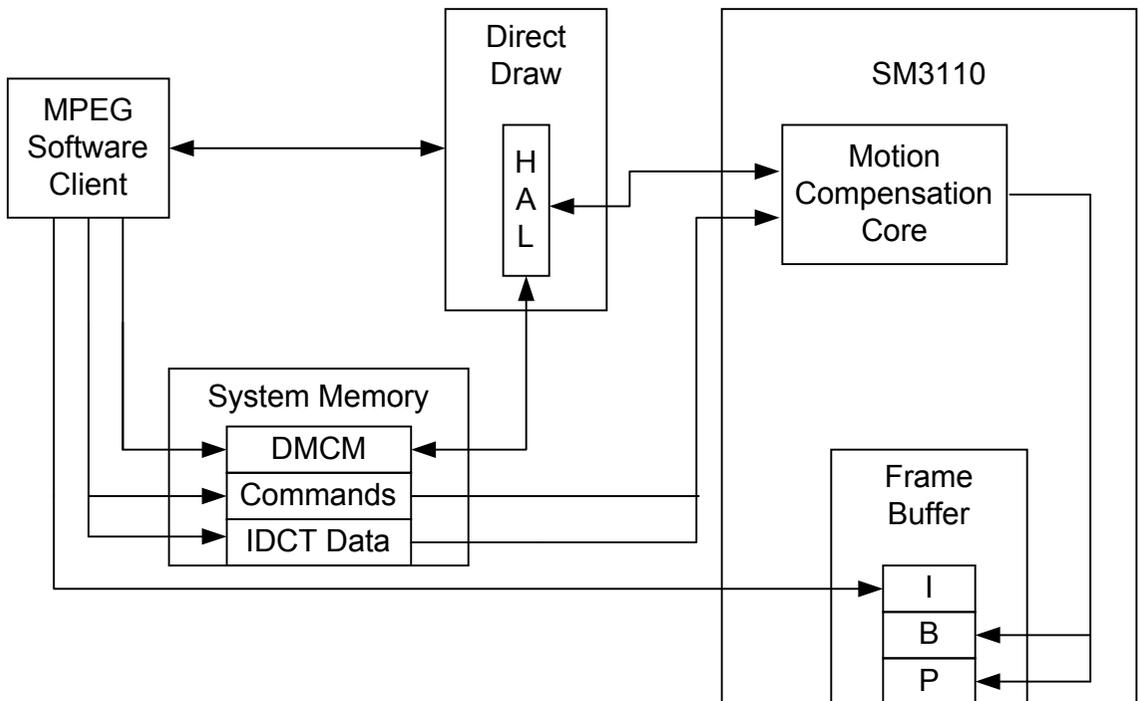


Figure 6-16. Motion Compensation Dataflow

Microsoft's DirectDraw is used as the interface between the MPEG software client and the motion compensation core. The block diagram shown above illustrates the data flow. The client and the driver communicate following the steps shown below.

<u>Client</u>	<u>Driver</u>
1 Direct Show - Tell driver to create surface.	Create Surface
2 Direct Show - Will lock/unlock the surface	Fill data structure
3 Decoder - Tell Driver what slot for decode	
4 Decoder - Lock, Tells what surface to display	Display Slot
5 Decoder - Goto 3	

6.9.2 Block Diagram

To use the integrated MC core, the MC driver first ensures that the MC subsystem is idle by programming the status registers. The driver writes the MC commands and immediate data into the 3D command/parameter FIFO. The MC subsystem is returned to normal processing mode by issuing an End of Stream MC command. These steps are explained in detail in the following sections. Figure 6-17 shows the MC block diagram of the SM3110.

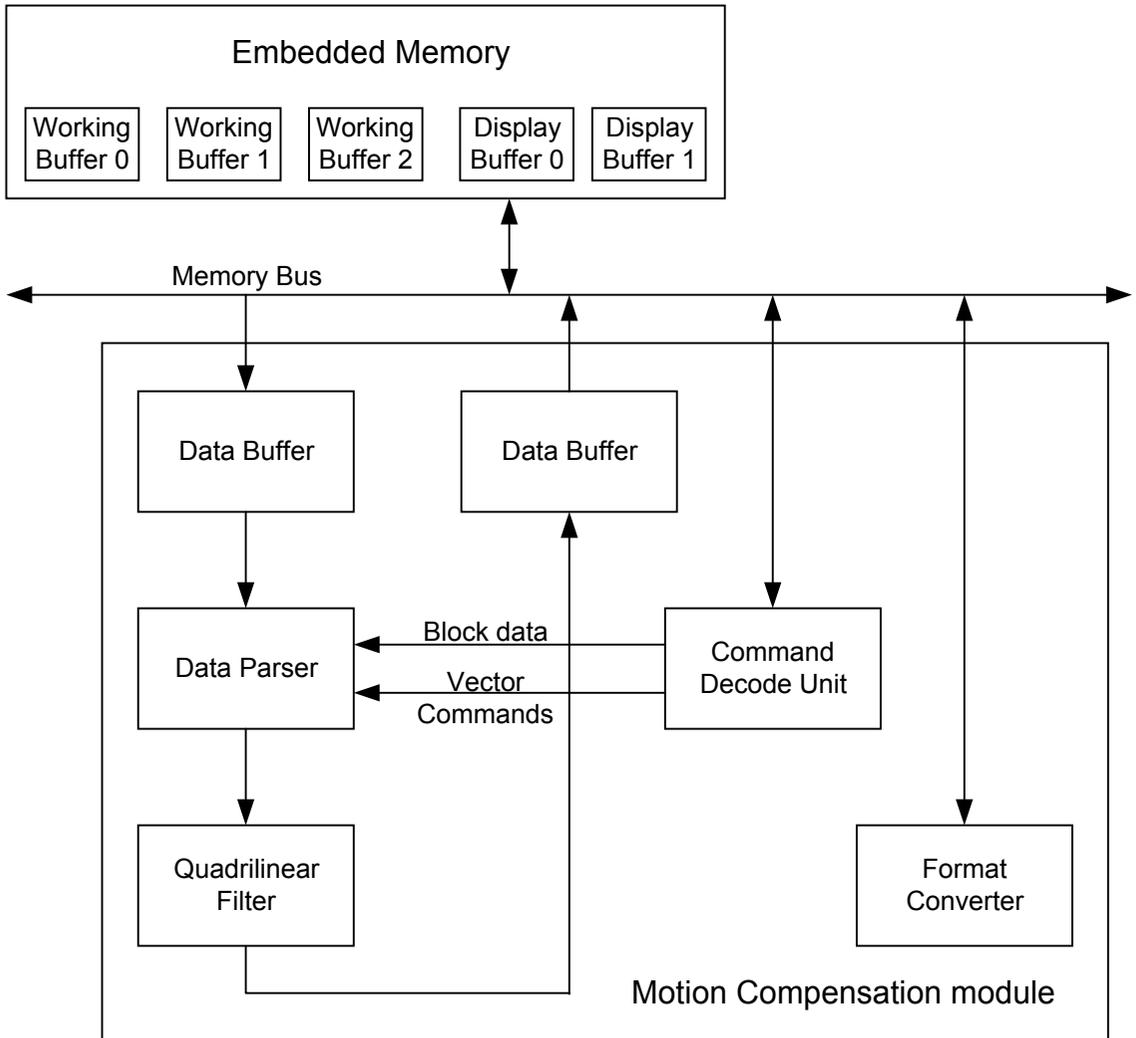


Figure 6-17. Motion Compensation Block Diagram

6.9.3 Motion Compensation Client Commands

Motion Compensation is a “back-end” process which is part of the overall MPEG-2 decoding model. The “front-end” process includes parsing the bitstream and reconstructing IDCT data. The interface between the front-end and the MC back-end will include the command stream. Data can be interleaved in such a way that the command comes first, followed by any IDCT data required by it, followed by the next command, etc.

6.9.3.1 DMCM Commands

DMCM (dwCommand field in the DMCM structure) is a 4CC command code which is set by the client (SoftDVD). It determines which action the DDraw driver will take on Unlock. There are two commands available in this implementation.

```
//----- COMMANDS -----  
#define DMCM_CMD_NONE 0  
#define DMCM_CMD_DISPLAY 2
```

The default command is DMCM_CMD_NONE - do nothing. It is used when client locks surface just to check driver's capabilities, version number or current state.

The other command is: DMCM_CMD_DISPLAY - new frame has to be shown.

These commands are sent to the MC core through the Command Decode Unit (CDU) from the 3D command FIFO.

6.9.3.2 DMCM surface allocation

The DMCM DDraw surface is an interface for hardware-assisted MPEG motion compensation using Silicon Magic's SM3110 graphics chips. The DirectShow client (SoftDVD) passes to the DDraw driver motion vectors, coefficient data (the output of IDCT) and control data on a frame by frame basis.

Memory is allocated for the DirectDraw motion compensation surface (DMCM) structure. The definition for this interface structure is given below.

```
//----- DMCM COMMANDS -----
// Commands for DDraw
#ifdef dmcm_h
#define dmcm_h
#define DMCM_CMD_NONE          0
#define DMCM_CMD_DISPLAY      2

typedef struct MCSiMagicBeginData_TAG
{
    BYTE  PictureStructure;
    BYTE  PictureCodingType;
    BYTE  TopFieldFirst;
    int   ForwardRefSlot;
    int   BackwardRefSlot;
    int   DestinationSlot;
} MCSiMagicBeginData;

typedef struct MCSiMagicMacroBlk_TAG
{
    int   hOffset;           // upper left macroblock coordinates in pels
    int   vOffset;           // upper left macroblock coordinates in scanlines
    BYTE  DCTType;           // Field DCT or Frame DCT
    BYTE  codedBlockPattern; // Coded Block Pattern
    BYTE  overflowCodedBlockPattern; // Overflow CodedBlock Pattern
    BYTE  motionType;
    int   PMV[2][2][2];      // contains motion vectors in half pel units
    int   motionFieldSelect[2][2]; // Motion
    int   macroblockType;
    LPBYTE *lpIDCTData;
    LPBYTE *lpOverflowData;
} MCSiMagicMacroBlk;
```

```

typedef struct
{
    struct
    {
        PBYTE      pIDCTData;    // pointer to IDCT data buffers
        PBYTE      pYBuffer;     // pointer to Y plane for I-frame
        PBYTE      pUBuffer;     // pointer to U plane for I-frame
        PBYTE      pVBuffer;     // pointer to V plane for I-frame
        UINT       uPitch;       // pitch for Y in frame buffer; half of this for UV
        UINT       uWaitTime;    // time to sleep in ms.
        LPVOID     lpfStartDecodeDriverFunction; // Called at start of decode
        LPVOID     lpfEndDecodeDriverFunction;   // Called at end of decode
        LPVOID     lpfProcessMBDriverFunction;   // Process Macroblock Function
        DWORD      dwInstanceData; //value to be passed back in parameter of
                                   the *lpfStartNewDecodeDriverFunction
    }
    DriverData;                // initialized by DDraw upon surface LOCK

    struct
    {
        DWORD      dwCmd;        // DMCM command (display)
        DWORD      dwTaskType;   // DMCM command (display)
        DWORD      dwDisplayTaskType; // DMCM command (display)
        int        dwDecodeBuffIndex;
        DWORD      dwDecodeTaskCount;
        DWORD      dwDataSize;   // Size of IDCT data to transfer
        DWORD      dwFrameType;  // Type of Frame (1:I, 2:P, 3:B)
        DWORD      dwSlotNumber; // Logic slot number
        DWORD      dwDisplayBuffIndex;
        DWORD      dwBobMode;    // 0:off, 1:Top 1'st, 2:Btm 1'st, 3:Only
                                   Top, 4:Only Btm
    }
    ClientData;                // filled by the client before the surface
                                   UNLOCK

} SURFMCSiMagic, *LPSURFMCSiMagic;

typedef LPSURFMCSiMagic (__stdcall *LPSTARTDECODEFUNC_MG)(MCSiMagicBeginData *);
typedef LPSURFMCSiMagic (__stdcall *LPENDECODEFUNC_MG)();
typedef LPSURFMCSiMagic (__stdcall *LPPROCESSMBFUNC_MG)(MCSiMagicMacroBlk *);

#endif

extern LPSURFMCSiMagic __stdcall StartDecodeMotionComp( MCSiMagicBeginData *);
extern LPSURFMCSiMagic __stdcall EndDecodeMotionComp();
extern LPSURFMCSiMagic __stdcall ProcessMacroBlock(MCSiMagicMacroBlk *);

```

6.9.4 Registers

6.9.4.1 Start and End Decode Functions

Two functions are defined in the driver that start and end the MC decode process. The driver passes pointers to these two functions to SoftDVD. The StartDecode function checks the status register to ensure that the 3D/MC subsystem is not busy and then writes the MC command to the command FIFO. SoftDVD passes the size of the data written to the driver during in the EndDecode function using which the driver writes the IDCT data to the command FIFO.

6.9.4.2 CDU Start Motion Compensation Command

The CDU interprets **0xD (13)** as the Motion Compensation command. This command places the CDU into MC mode.

Field	Function
31:28	Command. DH
27:00	Reserved.

6.9.4.3 End of Stream Instruction

In MC mode, the Command Decode Unit interprets 3D command FIFO data as MC core instructions. To leave MC mode, an End of Stream instruction is sent to the command FIFO. This instruction is **20000000H**.

Field	Function
31:00	MC End of Stream. 20000000H

6.9.4.4 Format Conversion Instruction

The MPEG-2 decoded bitstream is converted from the 4:2:0 YCrCb format to the SM3110 supported 4:2:2 format. The Command Decode Unit interprets a command with a leading value of **0xC (12)** in the four msb bits as an MC format conversion command. These commands are not valid between the MC command **0xD (13)** and the End of Stream command **20000000H**.

Field	Function
31:28	Command CH.
27:26	Slot Number (0,1,2,3). Selects which slot should be converted.
25:24	Horizontal Compression Ratio

Value	Semantics
0	1-1 compression (no compression)
1	2-1 compression
2	4-1 compression
3	8-1 compression
23:00	Destination buffer address.

In MC mode, the registers for slot base addresses and plane offsets need to be set by the driver. These are as follows:

Slot Base Addresses:

Offset	Field	Function
+340H	31:00	Slot 0 Base Address
+360H	31:00	Slot 1 Base Address
+380H	31:00	Slot 2 Base Address
+3A0H	31:00	Slot 3 Base Address

Plane 1/ Plane 3 Offsets:

Offset	Field	Function
+344H	31:00	Slot 0 Plane 1/Plane 3 Offset
+364H	31:00	Slot 1 Plane 1/Plane 3 Offset
+384H	31:00	Slot 2 Plane 1/Plane 3 Offset
+3A4H	31:00	Slot 3 Plane 1/Plane 3 Offset

Plane 2 Offsets:

Offset	Field	Function
+348H	31:00	Slot 0 Plane 2 Offset
+368H	31:00	Slot 1 Plane 2 Offset
+388H	31:00	Slot 2 Plane 2 Offset
+3A8H	31:00	Slot 3 Plane 2 Offset

6.9.5 Display Memory Requirements

6.9.5.1 Frame and Format Buffers

MPEG-2 uses a 4:2:0 video format which implies that for each video frame, the number of samples of each chrominance component (Cr and Cb) is one-half of the number of samples of luminance (Y), both horizontally and vertically.

With the 4:2:0 format, each pixel will take an average of 12 bits (eight bits for the luma and four for the chroma) which translates to each pixel requiring 1.5 bytes of storage. This tells us that one frame or picture in NTSC format which is 720 x 480 will require 518,400 bytes of memory while one frame in PAL format which is 720 x 576 will require 622,080 bytes of memory. The motion compensation core requires that 3 buffers of equal size need to be available to store the I, P and B frames that will be shown.

Table 6-8. Memory required for the three frame buffers

NTSC:	$720 \times 480 \times 1.5 = 518,400 \text{ bytes} \times 3 = 1,555,200 \text{ bytes}$
PAL:	$720 \times 576 \times 1.5 = 622,080 \text{ bytes} \times 3 = 1,866,240 \text{ bytes}$

The MPEG-2 decoded bitstream is in the 4:2:0 YCrCb pixel format as mentioned earlier. SM3110 can directly display 4:2:2 YCrCb progressive video. This requires a conversion to the 4:2:2 video format. Two format conversion buffers are allocated for this purpose. The memory requirements are shown below.

Table 6-9. Memory required for the two format conversion buffers

NTSC:	$720 \times 480 \times 2 = 691,200 \text{ bytes} \times 2 = 1,382,400 \text{ bytes}$
PAL:	$720 \times 576 \times 2 = 829,440 \text{ bytes} \times 2 = 1,658,880 \text{ bytes}$

The total memory requirement for NTSC is 1,555,200 bytes + 1,382,400 bytes = 2,937,600 bytes and the total memory required for PAL is 1,866,240 bytes + 1,658,880 bytes = 3,525,120 bytes.

6.9.5.2 Screen Resolutions Supported

Based on the above memory requirements for frame buffers and format conversion buffers, the following screen resolutions are supported.

Table 6-10. Screen Resolutions Supported

Format	Total Memory Available (bytes)	Required for Motion Comp. (bytes)	Memory Available for Display (bytes)	Supported Display Resolution
NTSC	4,194,304	2,937,600	1,256,704	800x600@16bpp (=960,000 bytes) 1024x768@8bpp (=786,432 bytes)
PAL	4,194,304	3,525,120	669,184	800x600@8bpp (=480,000 bytes)

6.9.5.3 System Memory Requirements

Table 6-11 lists the system memory requirement for the DirectDraw surface command buffer. This is the buffer that is used to store the commands and data that are sent to the video card driver (HAL) from SoftDVD

Table 6-11. Command and Data buffer sizes (in bytes) for NTSC and PAL format

ID	Type	Formula	NTSC	PAL
A	Macro Blocks	(pixels)/(16*16)	1,350	1,620
B	# Blocks	6 * A	8,100	9,720
C	Commands	12 * B	97,200	116,640
D	Data	64 * B	518,400	622,080
E	Data with 8 or 16 bit Error	2 * D	1,036,800	1,244,160
	Total Buffer Size	C + E	1,134,000	1,360,600

6.10 Power Management

6.10.1 Overview

The Power Management Function supports Microsoft's OnNow specification and the VESA BIOS Extensions/Power Management (VBE/PM) Standard. The Power Management Function receives commands (function calls) from the Operating System to place the display controller and display monitor(s) into one of several power saving/consuming states. The Power Management Function does not establish any power usage policy; it only supports the policy defined by the operating system.

Power management supports four states: On, Standby, Suspend and Off. These states represent four successive levels of power consumption. The operating system selects the appropriate state based upon user parameters, available system power and keyboard/mouse events.

6.10.2 OnNow Power Management

For Power Management in a Windows operating system environment, Microsoft has specified how power management can be supported "OnNow Power Management and Display Device Class Drivers". This specification describes which functions are expected from a miniVDD.

6.10.3 VESA BIOS Power Management

For BIOS Power Management, the VBE/PM Standard specifies how power management is supported. This standard specifies a set of functions (actually sub-functions) which are invoked by application software (using interrupt 10H) to utilize power saving features of display hardware.

6.10.4 Other Power Management

Other than VESA and OnNow power management, software can perform dynamic power management according to the hardware usage. SM3110 is designed in a way that each individual block can be shut off independently. The remaining issue will be when and what to shut off or turn on at any particular time. Generally speaking, under a normal simultaneous view state, display 1 and scaler 1 can all be turned off, while display 0 should be always on.

6.10.5 Registers for OnNow and VESA DPMS

What follows are details to support power management either in BIOS or a miniVDD using the SM3120. Support for each power consuming/saving state is described as if the state is entered without any dependence upon the previous state.

Note that DCLK register must be the last register to be changed else it can hang the machine. When the device is turned back to a full power state, DCLK must be the first register to be set back and some delay must be added (~100 ms).D0 – On

Register	bit(s)	Description
+00D4H	06:00	Enable MCLK by setting all bits to 0.
+00D8H	03:00	Enable DCLK by setting all bits to 0.
+0DF7H	01:00	Enable Horizontal Sync by setting field to 0.
+1DF7H		
+0DF7H	03:02	Enable Vertical Sync by setting field to 0.
+1DF7H		
+0D30H	07:00	Restore video output by setting field to a nonzero number based on display format and stride information.
+1D30H		
+0DF7H	07	Power up DAC by setting bit to 0.
+1DF7H		

6.10.5.1 D1 – Standby

Register	bit(s)	Description
+00D4H	06:00	Enable MCLK by setting all bits to 0.
+0DF7H	01:00	Disable Horizontal Sync by setting field to the value 1 or 2. This depends upon what the display panel expects as a pulse (a voltage increase pulse or a voltage drop pulse). 0 implies voltage is always low, 1 implies voltage is always high.
+1DF7H		
+0DF7H	03:02	Enable Vertical Sync by setting field to 0.
+1DF7H		
+0DF7H	07	Power down DAC by setting bit field to 1.
+1DF7H		
+0D30H	07:00	Make Video go “Blanked” by setting field to 0.
+1D30H		
+00D8H	03:00	Disable DCLK by setting all bits to 1.

6.10.5.2 D2 – Suspend

Register	bit(s)	Description
+00D4H	06:00	Enable MCLK by setting all bits to 0.
+0DF7H	01:00	Enable Horizontal Sync by setting field to 0.
+1DF7H		
+0DF7H	03:02	Disable Vertical Sync by setting field to 1 or 2. Depends upon what display panel expects as a pulse (a voltage increase pulse or a voltage drop pulse). 0 implies voltage is always low, 1 implies voltage is always high.
+1DF7H		
+0DF7H	07	Power down DAC by setting bit field to 1.
+1DF7H		
+0D30H	07:00	Make Video go “Blanked” by setting field to 0.
+1D30H		
+00D8H	03:00	Disable DCLK by setting all bits to 1.

6.10.5.3 D3 – Off

Register	bit(s)	Description
+0DF7H	01:00	Disable Horizontal Sync by setting field to the value 1 or 2. This depends upon what the display panel expects as a pulse (a voltage increase pulse or a voltage drop pulse). 0 implies voltage is always low, 1 implies voltage is always high.
+1DF7H		
+0DF7H	03:02	Disable Vertical Sync by setting field to 1 or 2. Depends upon what display panel expects as a pulse (a voltage increase pulse or a voltage drop pulse). 0 implies voltage is always low, 1 implies voltage is always high.
+1DF7H		
+0DF7H	07	Power down DAC by setting bit field to 1.
+1DF7H		
+0D30H	07:00	Make Video go “Blanked” by setting field to 0.
+1D30H		
+00D4H	06:00	Disable MCLK by setting all bits to 1.
+00D8H	03:00	Disable DCLK by setting all bits to 1.

6.10.6 Registers for Dynamic Power Management

The registers to control individual blocks are 0xD4 and 0xD8. The following table is a breakdown of their definition.

Block	Registers	Description
Display 0	+00D8H bit 0	Should be always on
Display 1	+00D8H bit 1, +00D4H bit 3	Shut off when using simultaneous view
Scaler 0	+00D8H bit 3, +00D4H bit 1	Turn on during CreateSurface() in DirectDraw Turn off during DestroySurface() in DirectDraw
Scaler 1	+00D8H bit 2, +00D4H bit 2	When dual view is enabled, same condition as Scaler 0
2D Engine	+00D4H bit 5	Turn on during 2D driver enable Turn off during 2D driver disable
3D Engine	+00D4H bit 6	Turn on during context creation in Direct3D Turn off during context destroy in Direct3D
MIU	+00D4H bit 0	Should be always on

6.11 Video Capture

6.11.1 Overview

The SM3110 controller supports YUV4:2:2 and CCIR 656 video input and interrupt mechanisms for video capture. A I²C video decoder must be used to convert the incoming NTSC/PAL/SECAM video signal to digital streams. A configurable input/output data port is provided to access up to 8 bits of data that can be used for I²C device control.

6.11.2 I²C device interface

Pins 2 and 3 of the configurable data port are reserved to control the video decoder I²C device. Pin 2 is for clock (SCL) and pin 3 is for data (SDA).

Sample code: Control CIO port (configurable input/output data port)

```

SCL_BIT    equ    4                ;bit 2 for Clock
SDA_BIT    equ    8                ;bit 3 for Data

InitCIOport    proc    near
    mov    fs, CmdPortSelector    ;mapio selector
    mov    al, 0F3h                ;bit 3:2=00 to enable the mask
    mov    fs:[412h], al          ;CIO control mask
    mov    holding_register, 0FFh ;pull high init.
    ret
InitCIOport    endp

WriteBit    macro    bMask, fHighLow ;set clock/data pin to Low(0) or High(non-0)
    mov    al, fHighLow
    neg    al                        ;entry:0,    non-0
    sbb    al, al                    ;    0,    FF
    and    al, bMask                 ;    0,    SCL_BIT/SDA_BIT
    and    holding_register, NOT bMask
    or     holding_register, al
    mov    al, holding_register
    mov    byte ptr fs:[411h], al    ;CIO output
    endm

ReadBit    macro    bMask                ;read clock or data pin disable mask
    mov    ah, fs:[412h]              ;CIO control mask
    or     ah, bMask                  ;set 1 to disable
    mov    fs:[412h], ah              ;CIO control mask
;Read in holding_register
    mov    al, fs:[410h]              ;CIO input port
    mov    holding_register, al
;enable mask
    and    ah, NOT bMask              ;set 0 to enable

```

```

    mov    fs:[412h],ah          ;CIO control mask
;return in al=0 or non-0
    and    al, bMask            ;data in mask bit
    endm

Delay macro
    push  cx
    mov   cx, delay_time       ;delay time adjust to fit I2C device
                                       duration spec

delay_loop:
    loop  delay_loop
    pop   cx
    endm

```

With macros for reading and writing individual bits to the I²C device, this is an example of how to write one full byte of data to the I²C device.

```

SendByteToI2cDevice proc near    ;entry: bData
    mov    bl,bData
    mov    cx,8                  ;8 bits
    WriteBit    SCL_BIT,0      ;write clock pin to low
sbi2c_loop:
    shl    bl,1
    sbb    bh,bh
    WriteBit    SDA_BIT,bh     ;write data pin

    WriteBit    SCL_BIT,1      ;write clock pin to high
    Delay
    WriteBit    SCL_BIT,0      ;write clock pin to low to create clock pulse
    Delay
    loop    sbi2c_loop         ;loop 8 times for a BYTE

    Delay

    WriteBit    SDA_BIT,1      ;write data pin with HIGH to check for ACK
    Delay
    Delay
    Delay
    WriteBit    SCL_BIT,1      ;write clock pin to HIGH for a clock pulse
    Delay

    ReadBit    SDA_BIT         ;read bit in AL to see if it is pulled low by
                                       I2C device

    Delay

    push  ax                    ;save
    WriteBit    SCL_BIT,0      ;write clock pin to LOW for a clock pulse
    Delay
    pop    ax                   ;restore
    ret                        ;return in AL with 0 (ACK) or non-0 (no ACK)

```

```
SendByteToI2cDevice  endp

ReadByteFromI2cDevice proc near
    mov     cx,8                ;8 bits
    WriteBit SCL_BIT,0        ;write clock pin to low
rbi2c_loop:
    WriteBit SCL_BIT,1        ;write clock pin to high
    Delay

    ReadBit  SDA_BIT          ;write data pin in AL=0 or non-0
    neg     al
    adc     bl,bl              ;carry bit shift to BL

    WriteBit SCL_BIT,0        ;write clock pin to low to create a clock pulse
    Delay
    loop   rbi2c_loop         ;loop 8 times for a BYTE

    Delay

    WriteBit SDA_BIT,0        ;write data pin to send ACK
    Delay
    WriteBit SCL_BIT,1        ;write clock pin to HIGH for a clock pulse
    Delay
    WriteBit SCL_BIT,0        ;write clock pin to LOW for a clock pulse
    Delay

    WriteBit SDA_BIT,1        ;write data pin device
    Delay

    mov     ax,bx              ; return read BYTE data in AL
    ret
ReadByteFromI2cDevice  endp
```

6.11.3 Video port control

As with other graphics data, the SM3110 surface mechanism is used to associate the video input with a buffer. The scaler channels are used to associate the surface to a display. This section provides examples to perform common control operations. These are explained as specific examples for clarity, but could be combined in real applications.

To prepare for video capture function, following steps are required at the initialization period.

1. Define scaler and video capture surfaces.
2. Channel controls - assign surface
3. Set Overlay priority.
4. Set ColorKey

Sample code: Prepare for video capture

```

PrepareVideoCapture      proc   near
    ;;Define a surfaces for video data input
    mov     ax, VIDCAP_SURFACEINDEX      ;surface descriptor used for video cap-
                                         ture

    mov     surfacenum,ax
    mov     eax, VIDCAP_WIDTH
    mov     surfacestride,eax
    mov     edx, VIDCAP_HEIGHT
    mul     edx
    mov     edx,dwPrivatePhyAddr        ;total mem-reserved buffer
    sub     edx,eax                     ;assign surface location at the end of
                                         memory

    mov     surfacebase,edx
    Call    DefineSurface                ;define the surface to accept video data
    mov     fs,CmdPortSelector           ;mapio selector
    mov     ax, VIDCAP_HEIGHT
    mov     fs:[VideoPortMaxHeight],ax   ;set video input height here
    ;;assign the defined for scaler and video capture channel
    mov     eax,surfacenum
    mov     fs:[ChScaler0SIndex],eax     ;assign to scaler channel
    mov     fs:[ChVidCapSIndex],eax     ;assign to video capture channel
    xor     eax,eax
    mov     fs:[ChScaler0Offsets],eax    ;scaler channel default start at (0,0)
    mov     fs:[ChVidCapOffsets],eax    ;video capture channel always start at
                                         (0,0)

    ;;set color key
    mov     eax,color-key
    mov     fs:[Dsp0ColCmp],eax         ;set dest. Color-key color
    mov     eax,200h
    mov     fs:[Dsp0MixCtl],eax        ;enable dest. Color-key
    ;;set overlay control
    mov     eax,10h                     ;BG:scaler channel, FG:display channel
    mov     fs:[Overlay0Priority],eax    ;enable Color-key overlay
    ret
PrepareVideoCapture      endp

```

The method used to shrink or stretch video data from a scaler channel to a display channel was described in display controls, Section 6.5.1: 2D Functions, Display Operations.

To enable or disable video, both the video input port and the scaler channel need to be programmed.

Sample code: Enable or Disable Video

```
EnableVideo proc near
;;enable video input port
    mov    fs,CmdPortSelector      ;mapio selector
    mov    al,prescale             ;prescale factor, 0,1,2,3 to shrink
                                   video input by factor 1,1/2,1/4 and 1/8
    shl   al,4                    ;move to bit 4 and 5
    or    al,1                    ;bit 0 to enable/disable
    mov    fs:[VideoPortConfig],al ;set video input height here
;;enable video to display
    mov    eax,58h
    mov    fs:[ChScaler0Format],eax ;set format to YUY2 to enable
    ret
EnableVideo endp

DisableVideo proc near
;;disable video input
    mov    fs,CmdPortSelector      ;mapio selector
    mov    al,prescale             ;prescale factor, 0,1,2,3 to shrink
                                   video input by factor 1,1/2,1/4 and 1/8
    shl   al,4                    ;move to bit 4 and 5
    mov    fs:[VideoPortConfig],al ;set video input height here
;;disable video to display
    xor    eax,eax
    mov    fs:[ChScaler0Format],eax ;0 to disable
    ret
DisableVideo endp
```

6.11.4 Interrupt for video capture

An interrupt line serves as the hardware interrupt scheme for video capture. The following functions show how to enable, disable and clear the interrupt line.

Sample code: Control interrupt line

```
EnableInterrupt proc near
    mov     fs,CmdPortSelector           ;mapio selector
    mov     al,020h                     ; bit 5 for Video capture interrupt
    or     fs:[ InterruptMask],al      ; 0F1h
    mov     al,02h                       ;bit 1 for External Video Input
    or     fs:[DispIntFlag],al         ; 0DF0h
    ret
EnableInterrupt endp

DisableInterrupt proc near
    mov     fs,CmdPortSelector           ;mapio selector
    mov     al,0fch                     ;bit 1 for External Video Input, bit
                                        ;0:reset flag bit
    and     fs:[DispIntFlag],al         ; 0DF0h
    mov     al,0dfh                     ; bit 5 for Video capture interrupt
    and     fs:[ InterruptMask],al     ; 0F1h
    ret
DisableInterrupt endp

ClearInterrupt proc near
    mov     fs,CmdPortSelector           ;mapio selector
    mov     al,02h                       ;bit 1 for External Video Input
    or     fs:[DispIntFlag],al         ; 0DF0h
    ret
ClearInterrupt endp
```

6.12 TV Out

See Chapter 7, Application Notes in the TV encoder section for details.

6.13 Diagnostics

The SM3110 contains some features useful for diagnostic purposes. The primary one of these is the diagnostic port, which allows predefined signals from different subsystems in the chip to be brought out to 8 output pins. To do this, software must set the appropriate 3-bit code in the diagnostic port register for the desired subsystem (other subsystems must be set to 0 to act as a pass-through).

In addition, a performance counter capability allows counting of any of the 8 diagnostic port bits selected by setting the appropriate counter source and enabling the performance counter in register 0x03E of the Command Decode Unit. The resultant performance count appears in register 0x010.

Also, some of the SM3110 internal SRAM values for the Triangle Setup Unit and Command Decode Unit are visible via registers 0x5C0-0x5FC, 0x600-0x7FC and 0x800-0xBFC.

6.14 Software Definitions

This section provides software definitions that are useful in understanding and programming the functions of the SM3110.

6.14.1 Equates

```

VENDORID          equ    8888H
DEVICEID          equ    4831H
Memory/Register Map
; Offsets to register areas, command ports, image ports, cursor area and
; deferred register write buffer

PERIPHERAL_CTL_BASE    equ    01FE0000H
REGISTER_BASE_OFFSET  equ    01FF0000H
IMAGE_BASE_OFFSET     equ    01FFC000H

; PrivateMemSelector equates
PRIVATEAREA_SIZE      equ    20000H    ;we reserve 128K for all fifos
CMD3DFIFO_SIZE        equ    10000H    ;64K 3d cmd fifo
CMDFIFO_SIZE          equ    4000H     ;16K 2d cmd fifo
IMGFIFO_SIZE          equ    8000H     ;32K image fifo
CURDEFERREDBUFFER_SIZE equ    400H     ;1K cursor defer fifo
DSPDEFERREDBUFFER_SIZE equ    400H     ;1K display defer fifo
CURSORBUFFER_SIZE     equ    0800H     ;2K cursor image buffer
ICONBUFFER_SIZE       equ    0800H     ;2K icon image buffer
ECMDFIFOSIZE          equ    2         ;encoded size - 16K
EIMGFIFOSIZE          equ    3         ;encoded size - 32K

CMD3DFIFO_OFFSET      equ    0
IMGFIFO_OFFSET        equ    CMD3DFIFO_OFFSET+CMD3DFIFO_SIZE
CMDFIFO_OFFSET        equ    IMGFIFO_OFFSET+IMGFIFO_SIZE
CURDEFERRED_OFFSET    equ    CMDFIFO_OFFSET+CMDFIFO_SIZE
DSPDEFERRED_OFFSET    equ    CURDEFERRED_OFFSET+CURDEFERREDBUFFER_SIZE
CURSOR_OFFSET         equ    DSPDEFERRED_OFFSET+DSPDEFERREDBUFFER_SIZE
ICON_OFFSET           equ    CURSOR_OFFSET+CURSORBUFFER_SIZE

; Rendering Engine Control/Status registers
RenderEngineStatus    equ    008F0h    ;rendering engine status
RenderEngineControl    equ    008F1h    ;rendering engine control
Status2D              equ    08F0H     ;2D engine status
BUSY2D                equ    010B     ;2D engine is busy executing
RUNNING2D             equ    001B     ;2D engine is running (not stopped)

Control2D             equ    08F1H     ;2D engine control register
RESET2D              equ    01000000B  ;Reset 2D engine
SINGLESTEP2D          equ    000000010B ;Executes next command and stops
START2D              equ    000000001B ;Starts 2D engine
STOP2D              equ    000000000B  ;Stops 2D

```

```

; Interrupts
InterruptStatus      equ    000F0H      ;Interrupt status
InterruptMask        equ    000F1H      ;Global interrupt mask
I_DISPLAY            equ    10000000B   ;Display 0 Interrupt
I_DISPLAY1           equ    01000000B   ;Display 1 interrupt
I_VideoCapture       equ    00100000B   ;Video Capture interrupt
I_PeripheralIO       equ    00010000B   ;Peripheral IO interrupt
I_HostDMA            equ    00000100B   ;Host DMA interrupt

; Power Management
OutPutEnablePM       equ    000E0H      ;Pwr Mgmt. Output enable
EXTMEMBUS            equ    10000000B   ;Ext. Mem Bus
VAFCPORT             equ    01000000B   ;VAFC port
PERIPHIO             equ    00100000B   ;Peripheral IO port

; Surface Descriptor Registers
NEXTSURFACE          equ    010H        ;multiplier to index into next
                                     surface registers
SurfaceBaseAddress   equ    00A00H      ;Surfaces base address
SurfaceStrideFormat  equ    00A04H      ;Surfaces stride format base
S0BaseAddress        equ    00A00H      ;Surface 0 Base address
S0StrideFormat       equ    00A04H      ;Surface 0 Stride and format
S1BaseAddress        equ    00A10H      ;Surface 1 Base address
S1StrideFormat       equ    00A14H      ;Surface 1 Stride and format
S2BaseAddress        equ    00A20H      ;Surface 2 Base address
S2StrideFormat       equ    00A24H      ;Surface 2 Stride and format
S3BaseAddress        equ    00A30H      ;Surface 3 Base address
S3StrideFormat       equ    00A34H      ;Surface 3 Stride and format
S4BaseAddress        equ    00A40H      ;Surface 4 Base address
S4StrideFormat       equ    00A44H      ;Surface 4 Stride and format
S5BaseAddress        equ    00A50H      ;Surface 5 Base address
S5StrideFormat       equ    00A54H      ;Surface 5 Stride and format
S6BaseAddress        equ    00A60H      ;Surface 6 Base address
S6StrideFormat       equ    00A64H      ;Surface 6 Stride and format
S7BaseAddress        equ    00A70H      ;Surface 7 Base address
S7StrideFormat       equ    00A74H      ;Surface 7 Stride and format
S8BaseAddress        equ    00A80H      ;Surface 8 Base address
S8StrideFormat       equ    00A84H      ;Surface 8 Stride and format
S9BaseAddress        equ    00A90H      ;Surface 9 Base address
S9StrideFormat       equ    00A94H      ;Surface 9 Stride and format
SABaseAddress        equ    00AA0H      ;Surface A Base address
SAStrideFormat       equ    00AA4H      ;Surface A Stride and format
SBBaseAddress        equ    00AB0H      ;Surface B Base address
SBStrideFormat       equ    00AB4H      ;Surface B Stride and format
SCBaseAddress        equ    00AC0H      ;Surface C Base address
SCStrideFormat       equ    00AC4H      ;Surface C Stride and format
SDBaseAddress        equ    00AD0H      ;Surface D Base address
SDStrideFormat       equ    00AD4H      ;Surface D Stride and format
SEBaseAddress        equ    00AE0H      ;Surface E Base address
SEStrideFormat       equ    00AE4H      ;Surface E Stride and format
SFBaseAddress        equ    00AF0H      ;Surface F Base address

```

```

SFStrideFormat          equ    00AF4H          ;Surface F Stride and format

CURSOR0SURFACEINDEX    equ    0
CURSOR1SURFACEINDEX    equ    1
SCALER0SURFACEINDEX    equ    2
SCALER1SURFACEINDEX    equ    3
DISPLAYSURFACEINDEX    equ    4
FIRSTSURFACEINDEX      equ    5
LASTSURFACEINDEX       equ    0Eh
DISPLAY1SURFACEINDEX   equ    0Eh
ICONSURFACEINDEX       equ    0Fh

CURSOR0SURFACE equ SurfaceBaseAddress +(CURSOR0SURFACEINDEX* NEXTSURFACE)
CURSOR1SURFACE equ SurfaceBaseAddress +(CURSOR1SURFACEINDEX* NEXTSURFACE)
SCALER0SURFACE equ SurfaceBaseAddress +(SCALER0SURFACEINDEX* NEXTSURFACE)
SCALER1SURFACE equ SurfaceBaseAddress +(SCALER1SURFACEINDEX* NEXTSURFACE)
DISPLAYSURFACE equ SurfaceBaseAddress +(DISPLAYSURFACEINDEX* NEXTSURFACE)
FIRSTSURFACE   equ SurfaceBaseAddress +(FIRSTSURFACEINDEX* NEXTSURFACE)
LASTSURFACE    equ SurfaceBaseAddress +(LASTSURFACEINDEX* NEXTSURFACE)
DISPLAY1SURFACE equ SurfaceBaseAddress +(DISPLAY1SURFACEINDEX* NEXTSURFACE)
ICONSURFACE    equ SurfaceBaseAddress +(ICONSURFACEINDEX* NEXTSURFACE)

BPP1          equ    1
BPP4          equ    3
BPP8          equ    4
BPP16         equ    5
BPP15         equ    5
BPP24         equ    6

CHANNELCUR    equ    0
CHANNELICON   equ    1
CHANNEL0      equ    2
CHANNEL1      equ    3
CHANNEL2      equ    4

; Display Channel parameters
ChannelOffsets equ 00B00H          ;Offsets register base
ChannelIndex   equ 00B04H          ;Index register base
ChCurOffsets  equ 00B00H          ;Cursor Channel display X,Y offsets
ChCurSIndex   equ 00B04H          ;Cursor Channel attached surface
index

ChIconOffsets  equ 00B10H          ;Control Channel display X,Y offsets
ChIconSIndex   equ 00B14H          ;Control Channel attached surface
index

Ch0Offsets     equ 00B20H          ;Channel 0 display X,Y offsets
Ch0SIndex      equ 00B24H          ;Channel 0 attached surface index
Ch1Offsets     equ 00B30H          ;Channel 1 display X,Y offsets
Ch1SIndex      equ 00B34H          ;Channel 1 attached surface index
Ch2Offsets     equ 00B40H          ;Channel 2 display X,Y offsets
Ch2SIndex      equ 00B44H          ;Channel 3 attached surface index

```

```

Ch0UOffsets      equ    00BB0H      ;Channel 0 U offsets
Ch0VOffsets      equ    00BC0H      ;Channel 0 V offsets
ChAudSIndex      equ    00BA4H      ;Audio Channel Playback Surface Index
ChVidCapSIndex   equ    00BE4H      ;Video Capture Channel Surface Index

```

; Buffer Descriptor Registers

```

CMDBufferBaseAddress  equ    00C00H      ;Command Buffer Base/Size
CMDBufferWritePtr     equ    00C04H      ;Command buffer Write pointer
CMDBufferReadPtr      equ    00C08H      ;Command buffer Read pointer
IMGBufferBaseAddress  equ    00C10H      ;Image data Buffer Base/Size
IMGBufferWritePtr     equ    00C14H      ;Image buffer Write pointer
IMGBufferReadPtr      equ    00C18H      ;Image buffer Read pointer
CmdFIFOStatus         equ    00C0CH      ;Command Parameter FIFO status
ImgDataFIFOStatus     equ    00C1CH      ;Image Data FIFO status
FIFOMASK              equ    0FH        ;
FIFOFULL              equ    0          ;Fifo full
FIFO1_8TH             equ    1          ;Fifo 1/8 empty
FIFO1_4TH             equ    2          ;Fifo 1/4 empty
FIFO3_8TH             equ    3          ;Fifo 3/8 empty
FIFO1_2TH             equ    4          ;Fifo 1/2 empty
FIFO5_8TH             equ    5          ;Fifo 5/8 empty
FIFO3_4TH             equ    6          ;Fifo 3/4 empty
FIFO7_8TH             equ    7          ;Fifo 7/8 empty
FIFOEMPTY             equ    8          ;Fifo empty
CursorRecordPtr       equ    00C24H      ;Cursor Record pointer - Write
                        pointer
CursorReplayPtr       equ    00C28H      ;Cursor Replay pointer - Read pointer
DisplayRecordPtr      equ    00C34H      ;Display Record pointer - Write
                        pointer
DisplayReplayPtr      equ    00C38H      ;Display Replay pointer - Read
                        pointer

DMALocalMemAddPtr     equ    00C44H      ;Host DMA Local Memory Address
                        Pointer
CursorReplayStsReg    equ    00CF0H      ;Cursor Replay status
DisplayReplayStsReg   equ    00CF4H      ;Display Replay status
REPLAYPENDING         equ    01000000B   ;Deferred register update is active
REPLAYACTIVE          equ    01B        ;Cursor Replay control (R/W)
CursorReplayCtlReg    equ    00CF0H      ;Display Replay control (R/W)
DisplayReplayCtlReg   equ    00CF4H      ;Enable replay of deferred regis-
ENABLEREPLAY          equ    00000001B   ter writes

; Display Control
ChannelFormat         equ    00D00H      ;Channel format
CursorFormat          equ    00D00H      ;Cursor Channel Format
CURSOROFF             equ    0          ;
CURSOR1               equ    00010000B   ;enable mono cursor
CURSOR16              equ    01010000B   ;enable color cursor
IconChFormat          equ    00D10H      ;Icon Control Channel Format
Ch0Format             equ    00D20H      ;Scaler channel(0) format
Ch0YRControls         equ    00D24H      ;Scaler channel(0) controls YR

```

Ch0UVGBControls	equ	00D28H	;Scaler channel(0) controls UV/GB
INC_FRAC_BIT	equ	11	;Fractional component, LS bits
Ch0Controls	equ	00D2CH	;Scaler channel(0), Increment values
XNEAREST	equ	00000H	;X - Nearest neighbor
XBILINEAR	equ	08000H	;X - Bilinear
YNEAREST	equ	00000H	;Y - Nearest neighbor
YBILINEAR	equ	080000000H	;Y - Bilinear
Ch1Format	equ	00D30H	;Channel 1 format
Ch2Format	equ	00D40H	;Channel 2 format
CLUT4	equ	(BPP4 shl 4)+1	;4 BPP color lookup
CLUT8	equ	(BPP8 shl 4)+1	;8 BPP color lookup
RGB555	equ	(BPP16 shl 4)	;RGB 5-5-5
RGB565	equ	(BPP16 shl 4)+1	;RGB 5-6-5
RGB888	equ	(BPP24 shl 4)	;RGB 8-8-8
YUV420	equ	(BPP8 shl 4)+8	;8 Bit YUV 4-2-0 Planar MPEG
YUV9	equ	(BPP8 shl 4)+9	;8 Bit YUV 9 - Indeo
YUV422	equ	(BPP16 shl 4)+8	;16 Bit YUV 4-2-2
YVU422	equ	(BPP16 shl 4)+9	;16 Bit YVU 4-2-2
OverlayPriority	equ	00D80H	;Overlay priority control
Ch0ChromaCmp	equ	00D84H	;Channel 0, chroma comparator
Dsp0ColCmp	equ	00D88H	;Display0 color comparator
Dsp1ColCmp	equ	01D88H	;Display1 color comparator
Dsp0MixCtl	equ	00D90H	;Mix control between Display0
Dsp1MixCtl	equ	01D90H	;Mix control between Display1
ENABLECTL	equ	0001000000000000B	;Enable control surface
DISABLECTL	equ	0000000000000000B	;Enable control surface
ENABLECOLKEY	equ	0000100000000000B	;Enable color keying
DISABLECOLKEY	equ	0000000000000000B	;Enable color keying
KEYINVERT	equ	0000010000000000B	;Key sense inverted selects Dest.(Back) channel
KEYNORMAL	equ	0000000000000000B	;Key sense normal selects Src.(Fore) channel
KEYSRCBACK	equ	0000001000000000B	;Key source Dest. (Back) channel
KEYSRCFORE	equ	0000000000000000B	;Key source . (Fore) channel
BLENDENABLE	equ	0000000100000000B	;Enable blending
BLENDDISABLE	equ	0000000000000000B	;Disable blending
AudioChFormat	equ	00DE0H	;Audio Playback channel format
AUDIOPLAY	equ	0000000B	;Audio Play
AUDIOMUTE	equ	0100000B	;Audio mute
AUDIOPLAYENA	equ	0010000B	;Audio enabled at next display HSYNC
AUDIOPLAYDISA	equ	0000000B	;Audio disabled at next display HSYNC
AUDIOBSIZE256	equ	0000000B	;Audio buffer size 256 bytes
AUDIOBSIZE512	equ	0000001B	;Audio buffer size 256 bytes
AudioChClkControlseq	equ	00DE4H	;Serial Audio port clock controls
DispIntFlag	equ	00DF0H	;Display Interrupt Flag
DISPVBLANKINT	equ	001B	;Display Vblank Interrupt occurred
EXTVIDEOSYNCINT	equ	010B	;External Video Sync interrupt occurred

```

DispIntEnable      equ    00DF1H      ;MAXH only, SM3110 merged to F1H
DisplayStatus     equ    00DF2H      ;
ModeCtlReg        equ    00DF4H      ;Mode control Register
TIMEVGACRTC       equ    00000000B   ;Timing - VGA CRTC
TIMEENHCRTC       equ    00000100B   ;Timing - Enh. CRTC
TIMEENHVGA        equ    00001000B   ;Timing - Enh. CRTC genlock VGA CRTC
TIMEENHEXT        equ    00001100B   ;Timing - Enh. CRTC genlock with Exter-
                                     nal Sync
DISPENAVGA        equ    00000000B   ;Display Output - Enable VGA
DISPENAENH        equ    00000001B   ;Display Output - Enable Enhanced
DISPVGAENH        equ    00000010B   ;Display Output - VGA and Enh
DISPDISABLED      equ    00000011B   ;Display Output - disabled
SpDisplayControl  equ    00DF5H      ;Special Display Control
EnhSyncPolarityRegequ 00DF6H      ;Enh Sync Polarity Register
HSYNCPOS          equ    001B        ;
VSYNCPOS          equ    010B        ;
SyncLevelReg      equ    00DF7H      ;Sync levels for power management
SYNCSNORMAL       equ    0           ;
HSYNCSET0         equ    00001B      ;Force Hsync to 0
HSYNCSET1         equ    00010B      ;Force Hsync to 1
VSYNCSET0         equ    00100B      ;Force Vsync to 0
VSYNCSET1         equ    01000B      ;Force Vsync to 1
ScalerFifoSize    equ    00DF8H      ;Display Scaler FIFO Size
ScalerFifoThresholdequ 00DFAH      ;Display Scaler FIFO Threshold
PaletterWState    equ    00DFCH      ;Palette Read/Write status
;
; Display Timing
ChViewPositionStarts equ 00E00H      ;Channel View position starts
ChViewPositionEnds   equ 00E04H      ;Channel View position ends
CursorPositionStarts equ 00E00H      ;Cursor Channel Viewport posi-
                                     tion top left
CursorPositionEnds   equ 00E04H      ;Cursor Channel Viewport extents
                                     bot. right
IconChPositionStarts equ 00E10H      ;Icon Control Channel Viewport
                                     position top left
IconChPositionEnds   equ 00E14H      ;Icon Control Channel Viewport
                                     extents bot right
Ch0ViewPositionStarts equ 00E20H      ;Channel 0 Viewport position top
                                     left
Ch0ViewPositionEnds   equ 00E24H      ;Channel 0 Viewport extents bot
                                     right
Ch1ViewPositionStarts equ 00E30H      ;Channel 1 Viewport position top
                                     left
Ch1ViewPositionEnds   equ 00E34H      ;Channel 1 Viewport extents bot
                                     right
Ch2ViewPositionStarts equ 00E40H      ;Channel 2 Viewport position top
                                     left
Ch2ViewPositionEnds   equ 00E44H      ;Channel 2 Viewport extents bot
                                     right
VerticalCounter      equ    00E9AH      ;Vertical counter - Current scan-
                                     line displayed

```



```

FLAGS equ 00000000000000001000000000100000B ;Flags Register
SURFACEBaseIndex equ 00000000000000001000000000100100B ;Src. Dest. Base
indices
FGColor equ 00000000000000001000000000101000B ;Foreground
Color
BGColor equ 00000000000000001000000000101100B ;Background
Color
TRANSColor equ 00000000000000001000000000110000B ;Transparent
Color

; FLAGS Register Definition
; 10987654321098765432109876543210
F_BYTE equ 00000000000000000000000000000000B ;Image data byte aligned
F_WORD equ 0000000000100000000000000000000000B ;Image data word aligned
F_DWORD equ 0000000001000000000000000000000000B ;Image data dword aligned
F_BIT equ 000000000110000000000000000000000000B ;Image data bit aligned
;- valid with mono data
only
F_XN equ 000000000010000000000000000000000000B ;X decreasing - origin at
top left
F_XP equ 000000000000000000000000000000000000B ;X increasing - left to
right
F_YN equ 000000000000100000000000000000000000B ;Y decreasing
F_YP equ 000000000000000000000000000000000000B ;Y increasing - top to bot-
tom
; If Bit 12 is 0 - Mono
Source
F_MPT equ 000000000000001000000000000000000000B ;Mono Blt - pattern
transparent
F_MNORMAL equ 000000000000000000000000000000000000B ;Mono Blt - color expand
;1==Foreground, 0==Back-
ground
F_MBT equ 000000000000000001000000000000000000B ;Mono Blt Transparency
; - Background transparent
F_MFT equ 000000000000000001000000000000000000B ;Mono Blt Transparency
; - Foreground transparent
F_MINVERT equ 000000000000000110000000000000000000B ;Mono Blt - color expand
;0==Foreground, 1==Back-
ground
; If Bit 12 is 1 - Color
Source
F_TCINVERT equ 000000000000001000000000000000000000B ;Invert color transparency
F_TCOPAQUE equ 000000000000000000000000000000000000B ;No color transparency

```

F_TCSRC	equ	00000000000000 <u>01</u> 000000000000000B;Color Source transparency
F_TCDEST	equ	00000000000000 <u>10</u> 000000000000000B;Color Destination transparency
F_TCPAT	equ	00000000000000 <u>11</u> 000000000000000B;Color Pattern transparency
F_CLIP	equ	00000000000000 <u>10</u> 000000000000000B;Enable clipping
F_NOCLIP	equ	00000000000000 <u>00</u> 000000000000000B;Disable clipping
F_USEPATTERN	equ	00000000000000 <u>00</u> 000000000000000B;Use pattern buffer
F_USEBGCOLOR	equ	00000000000000 <u>01</u> 000000000000000B;Use Background color for pattern
F_LOCALMEM	equ	00000000000000 <u>00</u> 000000000000000B;Blt Src Local memory
F_IMGDATA	equ	00000000000000 <u>01</u> 000000000000000B;Blt Src System memory
F_MONO	equ	00000000000000 <u>00</u> 000000000000000B;Source Format Monochrome
F_COLOR	equ	00000000000000 <u>10</u> 000000000000000B;Source Format Color
CMD_BITBLT	equ	000000000000000000 <u>01</u> 000000000B;Bitblt command
CMD_LINE	equ	000000000000000000 <u>01</u> 000000000B;Line command
CMD_MEMTEST	equ	000000000000000000 <u>10</u> 000000000B;Memory test command
F_ROPMASK	equ	000000000000000000 <u>01111111</u> B;ROP mask

6.14.2 Macros

The following macro is used to generate deferred an address value for its immedate register. The register address must be ORed with the physical screen base address and written to the deferred port data area followed by the register data (both values must be doubleword). These writes are recorded into the deferred buffer and immedate registers are written with this data during VBLANK (replay).

Load Deferred Register Address with DWORD

```
LDAD      MACRO  reg, labl
          mov    reg, PhyCmdPortAddress
          add   reg, labl
          ENDM
```

6.14.3 Clock Synthesizer Programming Guide

There are three Phase-Locked Loop-based programmable clock synthesizers in SM3110 for the two display clocks (DCLK and PCLK) and one memory/2D/3D engine clock (MCLK) from an externally supplied reference frequency. Each of the output frequencies may be programmed to up to 400 MHz. An external, crystal-controlled oscillator generates the reference frequency (typically 14.31818 MHz) that is driven into the SM3110 on pin Y22. Alternatively, an external crystal can be connected between pins Y22 and W23 to generate the reference frequency. All three PLL synthesizers in the SM3110 are the same design.

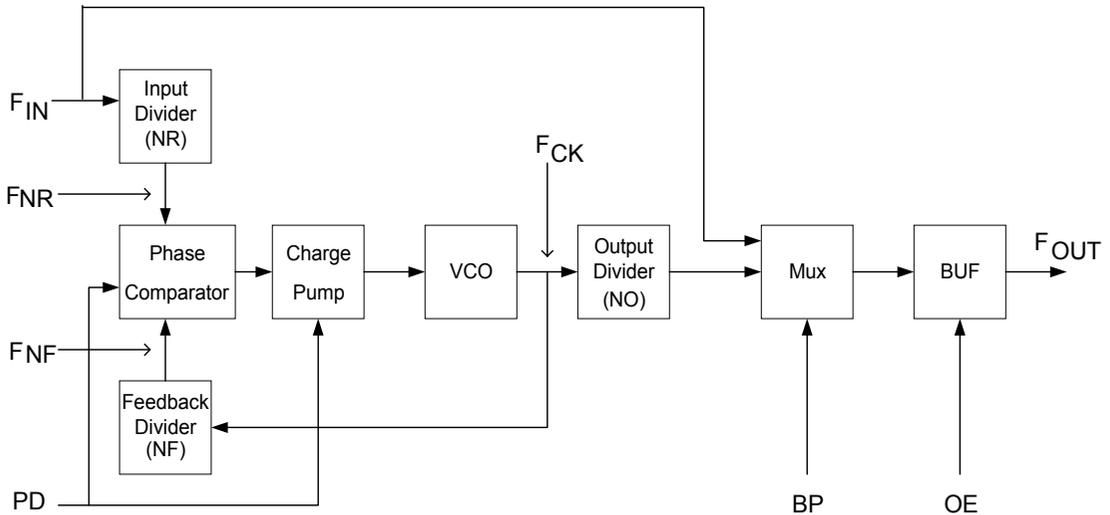


Figure 6-18. PLL Block Diagram

6.14.3.1 Theory of Operation

The Phase Comparator compares the phase difference of the input clocks from Input Divider (or Reference Divider), F_{NR} , and Feedback Divider (or Loop Divider), F_{NF} . If the edge of F_{NR} arrives earlier than the edge of F_{NF} , the output of the comparator adjusts the Charge Pump to provide a higher voltage to VCO, which increases the frequency of F_{CK} and brings the edge of F_{NF} earlier. If F_{NR} is later than F_{NF} , the Charge Pump decreases the voltage supplied to the VCO, which decreases frequency of F_{CK} and delays the edge of F_{NF} . With both edges aligned all the time, their frequencies must be the same, too.

With programmable Input and Feedback Dividers, the F_{NR} and F_{NF} can be at the same frequency while F_{IN} and F_{CK} can be different frequencies. This means the SM3110 can provide a wide range of output frequencies F_{CK} with a fixed input reference frequency F_{IN} .

The Output Divider provides a wider output frequency range for F_{OUT} while it keeps the VCO running at narrower, optimized range. Without this divider, F_{OUT} would have to have the same frequency range as F_{CK} .

These clock synthesizers have built-in power management which is controlled by signal PD. It can turn off the clock circuit to save power when not in use. The output buffer can also be turned off with signal OE.

The BP signal controls a muxing bypass circuit. When the BP signal is set to bypass mode, the output F_{OUT} and input F_{IN} have the same frequency.

6.14.3.2 Registers

Two registers control each clock synthesizer. In the first register, two byte specify the three (reference, feedback loop and output) dividers. In the second register, one byte controls the output and the PLL/VCO power down.

6.14.3.3 Frequency Control for Memory/Dot/Panel Clock

<i>Index</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+080H	15:00	R/W	MCLK Loop Control
+0090H	15:00	R/W	DCLK Loop Control
+1090H	15:00	R/W	DCLK Loop Control
<i>Bits</i>		<i>Field</i>	
	15:14	Output Divider	
<i>Value</i>		<i>Semantics</i>	
	0	NO has no division	
	1	NO is divide by 2	
	2	NO is divide by 2	
	3	NO is divide by 4	
<i>Bits</i>		<i>Field</i>	
	13:09	Reference Divider	
<i>Value</i>		<i>Semantics</i>	
	0	Reserved	
	1-31	Divide ratio (NR) is this value plus two	
<i>Bits</i>		<i>Field</i>	
	08:00	Feedback Loop Divider	
<i>Value</i>		<i>Semantics</i>	
	0	Reserved	
	1-511	Divide ratio (NF) is this value plus two	

The actual divide values for NR and NF are the binary values programmed in this register, bit 13 to 9 for NR and bit 8 to 0 for NF, plus two. For example, if the binary value in bit 13 to 9 is 10011 (decimal 19), the actual dividing value for Input Divider is decimal 21. If the binary value in bit 8 to 0 is 011001010 (decimal 202), the actual dividing value for Feedback Divider is decimal 204. The actual divide value for NO is specified in the register definition.

6.14.3.4 VCO Control for Memory Clock Synthesizer

Index	Field	Access	Function
+082H	07:00	R/W	MCLK VCO Control
	Bits	Field	
	07	PLL Module Internal Bypass Control (BP)	
		Value	Semantics
		0	Disabled. <i>Default</i> , normal operation.
		1	Enabled.
	Bits	Field	
	06	PLL Module Internal power down Control (PD)	
		Value	Semantics
		0	Disabled <i>Default</i> , normal operation.
		1	Enabled. PLL is in power down mode.
	Bits	Field	
	05	Clock Tree Source Select (Set by level of strapping input from ROMA1. Input pulled up to VDD.)	
		Value	Semantics
		0	External Clock
		1	Internal PLL Clock. <i>Default</i> .
	Bits	Field	
	04	Load PLL Counter to Working Register	
		Value	Semantics
		0	Normal Operation. <i>Default</i> .
		1	Load Register. Automatically returns to 0 after loading working register.
	Bits	Field	
	03:02	Reserved	
	01	PLL Module Internal output enable Control (OE)	
		Value	Semantics
		0	Enabled
		1	Disabled
	Bits	Field	
	00	Clock Tree Source Enable	
		Value	Semantics
		0	Disabled. MCLK forced to '0'.
		1	Enabled. MCLK running.

6.14.3.5 VCO Control for Dot/Panel Clock Synthesizer

Index	Field	Access	Function
+0092H	07:00	R/W	DCLK VCO Control (CRT)
+1092H	07:00	R/W	DCLK VCO Control (LCD)
	<i>Bits</i>	<i>Field</i>	
	07		PLL Module Internal Bypass Control (BP)
		<i>Value</i>	<i>Semantics</i>
		0	Disabled. <i>Default</i> , normal operation.
		1	Enabled.
	<i>Bits</i>	<i>Field</i>	
	06		PLL Module Internal power down Control (PD)
		<i>Value</i>	<i>Semantics</i>
		0	Disabled <i>Default</i> , normal operation.
		1	Enabled. PLL is in power down mode.
	<i>Bits</i>	<i>Field</i>	
	05		Clock Tree Source Select (Set by level of strapping input from ROMA1. Input pulled up to VDD.)
		<i>Value</i>	<i>Semantics</i>
		0	External Clock
		1	Internal PLL Clock. <i>Default</i> .
	<i>Bits</i>	<i>Field</i>	
	04		Load PLL Counter to Working Register
		<i>Value</i>	<i>Semantics</i>
		0	Normal Operation. <i>Default</i> .
		1	Load Register. Automatically returns to 0 after loading working register.
	<i>Bits</i>	<i>Field</i>	
	03		PLL Module Internal output enable Control (OE)
		<i>Value</i>	<i>Semantics</i>
		0	Enabled
		1	Disabled

VCO Control for Dot/Panel Clock Synthesizer (continued)

Bits	Field						
02	Reference Internal Oscillator Pad Control (Power Down)						
	<table border="1"> <thead> <tr> <th>Value</th> <th>Semantics</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Enabled. <i>Default.</i></td> </tr> <tr> <td>1</td> <td>Disabled. shut down oscillator to reduce power dissipation in sleep mode.</td> </tr> </tbody> </table>	Value	Semantics	0	Enabled. <i>Default.</i>	1	Disabled. shut down oscillator to reduce power dissipation in sleep mode.
Value	Semantics						
0	Enabled. <i>Default.</i>						
1	Disabled. shut down oscillator to reduce power dissipation in sleep mode.						
01	Divide Ratio Select						
	<table border="1"> <thead> <tr> <th>Value</th> <th>Semantics</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>VGA DCLK Frequencies, VGA Mode. <i>Default.</i> Use divide ratio specified in either 94 95H or 96 97H selected by 3CCH[3:2].</td> </tr> <tr> <td>1</td> <td>Programmable DCLK Frequencies, Enhanced Mode. Use divide ratio specified in F0 F1H.</td> </tr> </tbody> </table>	Value	Semantics	0	VGA DCLK Frequencies, VGA Mode. <i>Default.</i> Use divide ratio specified in either 94 95H or 96 97H selected by 3CCH[3:2].	1	Programmable DCLK Frequencies, Enhanced Mode. Use divide ratio specified in F0 F1H.
Value	Semantics						
0	VGA DCLK Frequencies, VGA Mode. <i>Default.</i> Use divide ratio specified in either 94 95H or 96 97H selected by 3CCH[3:2].						
1	Programmable DCLK Frequencies, Enhanced Mode. Use divide ratio specified in F0 F1H.						
00	Clock Tree Source Enable						
	<table border="1"> <thead> <tr> <th>Value</th> <th>Semantics</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Disabled. DCLK forced to '0'.</td> </tr> <tr> <td>1</td> <td>Enabled. DCLK running.</td> </tr> </tbody> </table>	Value	Semantics	0	Disabled. DCLK forced to '0'.	1	Enabled. DCLK running.
Value	Semantics						
0	Disabled. DCLK forced to '0'.						
1	Enabled. DCLK running.						

6.14.3.6 Selecting Register Value

The output frequency of VCO, F_{CK} , is determined by following equation:

$$F_{CK} = F_{in} * (NF / NR)$$

Where F_{CK} is the output frequency of VCO, F_{in} is the input reference frequency, NR is the input divider value, and NF is the feedback divider value.

The output frequency is determined by following equation:

$$F_{out} = \frac{F_{in} * NF}{NR * NO}$$

Where F_{out} is the output frequency, F_{in} is the input reference frequency, NR is the input divider value, NF is the feedback divider value and NO is the output divider value.

6.14.3.7 Programming Limitations

The synthesizer is very flexible. It doesn't have any special requirement of the programming order. However, VCO output frequency, F_{CK} , should be kept in its mid frequency range, which is about 200 MHz.

Bit 4 of the 082H, 0092H and 1092H registers is used to load the divider value into each PLL. These registers should be the last ones programmed when changing any divider values.

6.14.3.8 Register Values for Common Frequencies

The table below shows most of the frequencies required by VESA defined modes. This table is shown only as a programming example. There may be other register values not shown which can be used to achieve the same frequency. Please consult the equations in the earlier section for programming values. This table is based on reference input clock FIN at 14.31818 MHz.

Resolution	Refresh Rate	NF Register Value	NR Register Value	NO Register Value	VCO Freq.	Actual Freq.	Pixel Clock Freq.	% Error
640x480	60	216	29	3	100.7	25.17	25.175	-0.02
640x480	75	42	3	3	126	31.5	31.5	0
640x480	56	169	15	3	144.04	36.01	36	0.03
800x600	60	188	15	3	160.04	40.01	40	0.02
800x600	75	247	16	3	198.08	49.52	49.5	0.04
800x600	72	431	29	3	200	50	50	0
800x600	85	438	26	3	225	56.25	56.25	0
1024x768	60	343	17	3	260	65	65	0
1024x768	70	218	19	2	150	75	75	0
1024x768	75	31	1	2	157.5	78.75	78.75	0
1024x768	85	64	3	2	189	94.5	94.5	0
1280x1024	60	345	21	2	216.02	108.01	108	0.01
1280x1024	75	130	5	2	270	135	135	0
1280x1024	85	64	1	2	315	157.5	157.5	0
1600x1200	60	179	6	2	323.96	161.97	162	-0.02
1600x1200	65	96	2	2	350.8	175.4	175.5	-0.06
1600x1200	70	130	3	2	378	189	189	0
1600x1200	75	97	5	0	202.5	202.5	202.5	0
1600x1200	85	318	18	0	229.1	229.09	229.5	-0.18

7 **Application Notes**



7.1 Overview

The Application Notes elaborate on issues arising from designing this chip into computer systems. The first one, Board Design, covers board design by pin function group. Each section has a simple pin connection diagram, showing different designs where there is more than one way to use these pins.

The On-Board testing section describes how to use the built-in testing functions in SM3110 to isolate motherboard problems without removing the SM3110 from the board.

The Programmable I/O section shows the use of both types of programmable I/O in SM3110 in customizing system designs with unique functions.

The Power-On Configuration section describes the chip configuration that must be set at power up time for SM3110 to function properly on each system type.

7.2 Board Design

The board is divided into sections, with each functional pin group having a block diagram as an overall view of display subsystem, including the host interface, reference clock, LVDS interface, LCD panel interface, CRT interface, BIOS ROM interface, TV encoder interface and power-on strapping.

7.2.1 Power Supply

The SM3110 uses 3.3V and 2.5V. The 3.3V is only for the I/O pads and should be applied to the power pins marked as VDD_IO.

2.5V power has been divided into several sections. The first one, VDD_C, is for the digital core section. This section requires the highest current in the 2.5V section when the chip is in normal operation. The VDD_R is the 2.5V for embedded memory section. VDD_P and VDD_S are 2.5V for digital section of internal PLLs and DLL section, respectively. AVDD_P, AVDD_S, and AVDD_D are 2.5V for analog portion in PLLs, DLL, and DACs, respectively. PCB layout for these different power supplies must be done carefully to prevent noise generated from one section coupling into any other section. A ground layer in the PCB is recommended for the entire display subsystem section.

7.2.2 Block Diagram

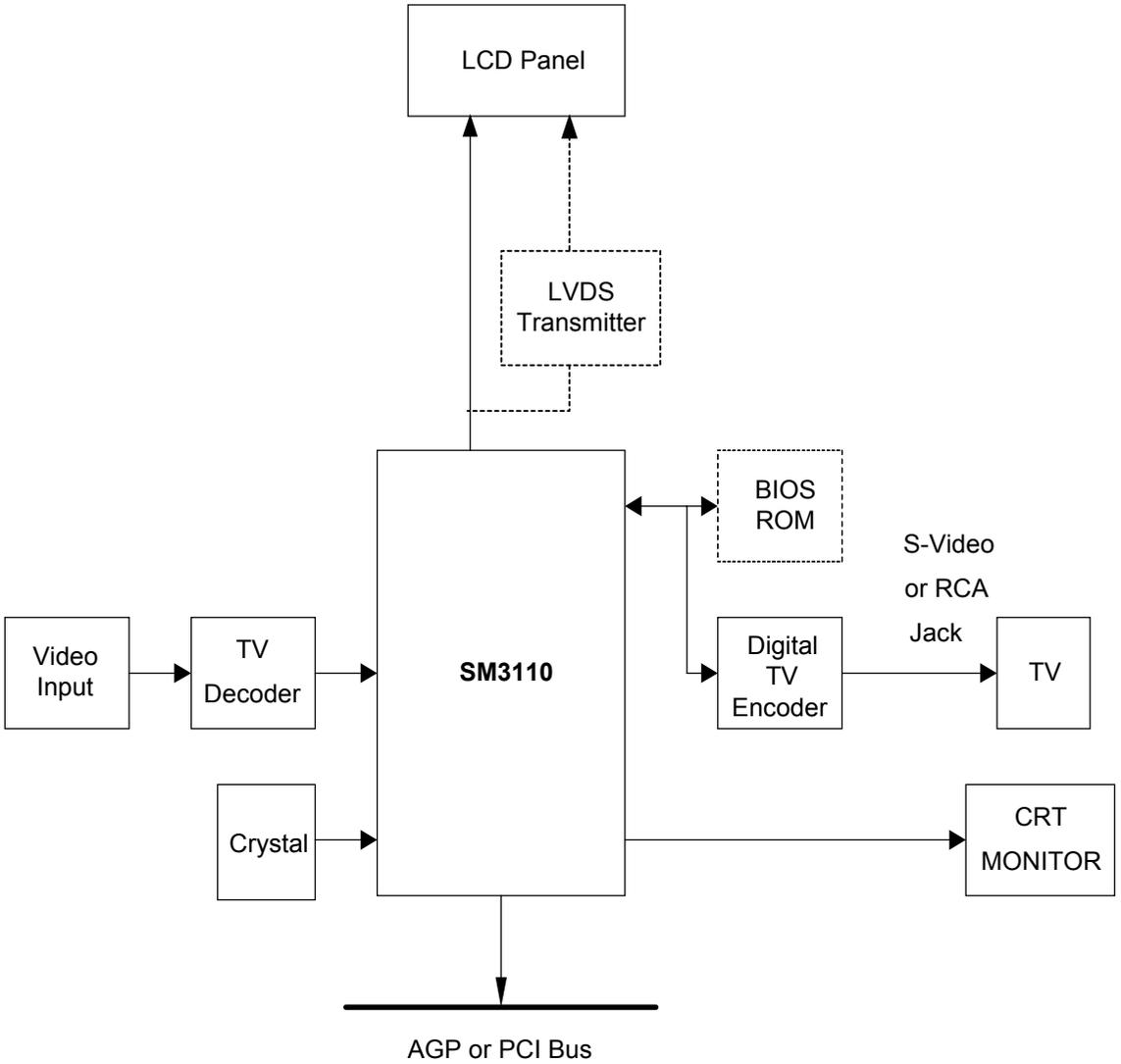


Figure 7-1. SM3110 System Diagram

7.2.3 Host Interface

The SM3110 supports either AGP 1X/2X or PCI 66 MHz interface. A power-on strapping pin determines the interface type and must be set correctly for communicating with the CPU. See the Power-On Configuration section for details of strapping options.

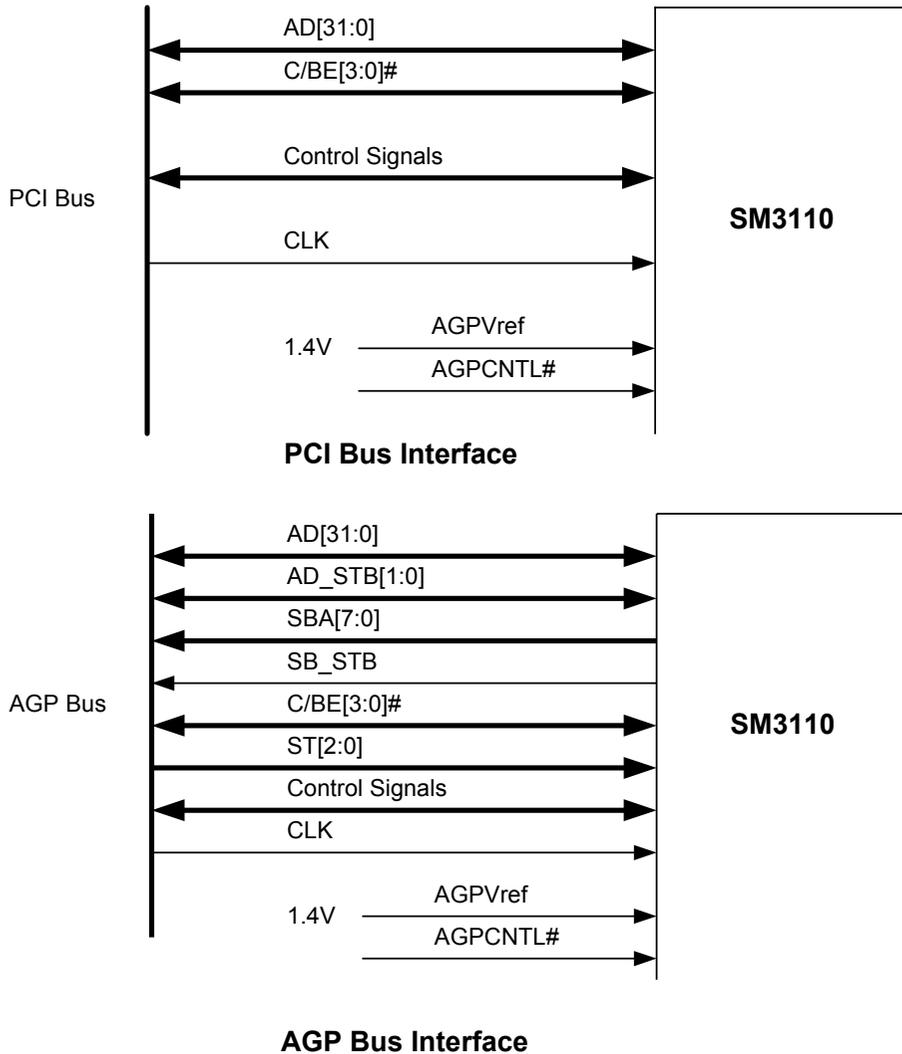


Figure 7-2. SM3110 Interfaces to Host

The 2X AGP bus uses strobing signals on the receiving side to latch input data at the correct time. The layout of these strobing signals and their corresponding data/address lines must be carefully done to ensure the trace lengths are within specification (see AGP Specification version 2.0 and AGP Platform Design Guide for more detailed layout considerations).

7.2.4 Reference Clock

The SM3110 requires a single reference clock input for its internal PLLs. These PLLs can be programmed individually to generate 3 different clocks used by the chip. The typical frequency of this reference clock is 14.31818 MHz. However, it can be any frequency from 2 MHz to 40 MHz. This clock can be from either a crystal (with external resistors and capacitors) or a CMOS level oscillator. Please refer to the drawing below for the usage of these two types of circuit.

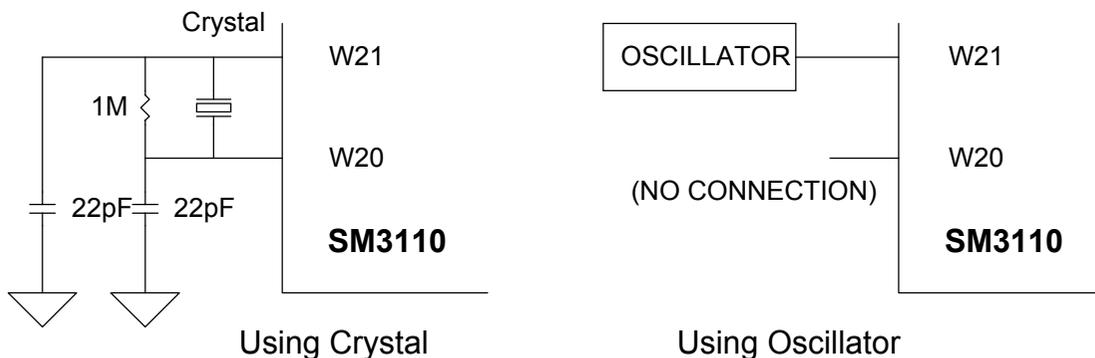


Figure 7-3. Reference Clock Methods

7.2.5 LCD Panel Interface

The SM3110 provides 36 pixel data pins and all other timing and control pins needed to interface with any LCD flat panel. It supports 8-bit, 16-bit, or 12+12-bit DSTN panel. For TFT panel, 9-bit, 12-bit, 18-bit, 24-bit, 9+9-bit, 12+12-bit, and 18+18-bit are supported. The data/pin assignment is listed in Table 7-1.

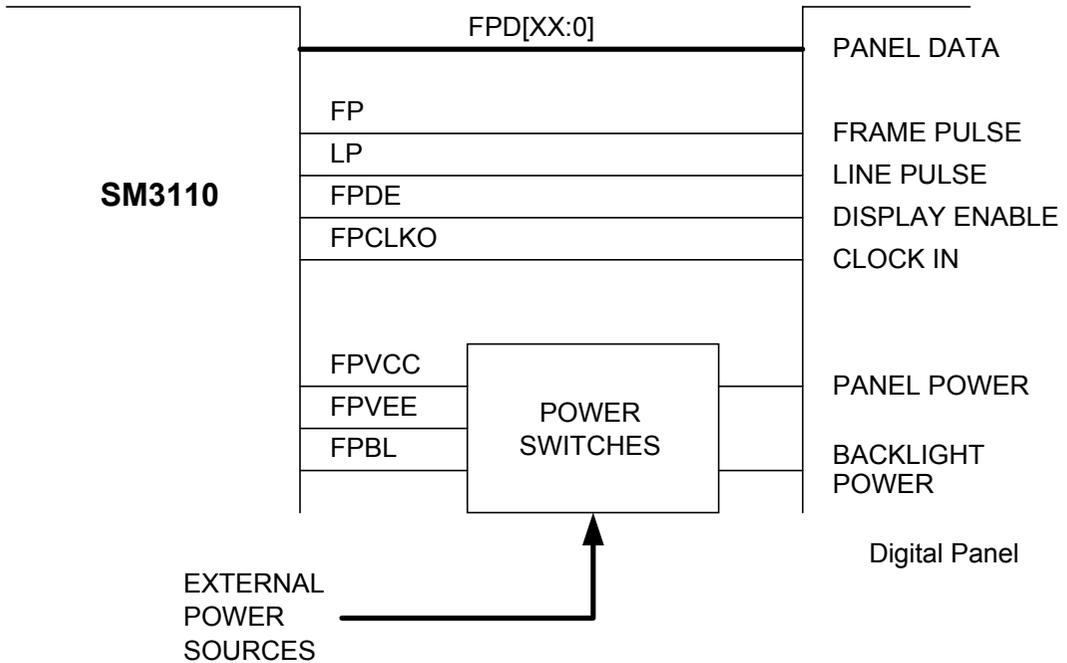


Figure 7-4. Digital Panel Interface Diagram

7.2.6 LVDS Transmitter Interface

SM3110's 36 pixel data, timing and control pins can interface with any LVDS transmitter(s), which can in turn drive an LCD panel with an LVDS interface.

The SM3110 supports 8-bit, 16-bit, or 12+12-bit DSTN panels as well as 9-bit, 12-bit, 18-bit, 24-bit, 9+9-bit, 12+12-bit and 18+18-bit TFT panels. The data/pin assignment is listed in Table 7-1.

To drive a panel through the LVDS interface, an LVDS transmitter chip is required. A block diagram is provided below for reference.

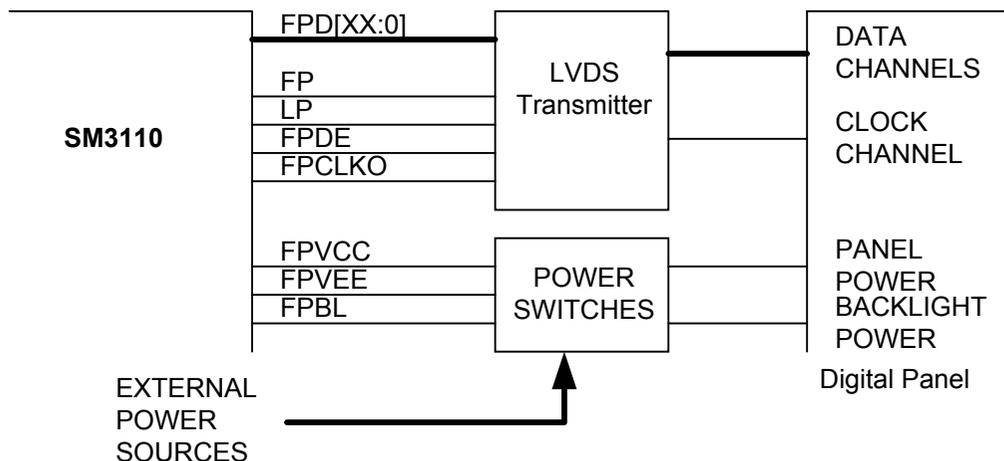


Figure 7-5. LVDS Panel Interface

Table 7-1. FPD[35:0] Different Panel Type Settings

Bit per Pixel	TFT									
	Single Pixel				Double Pixel			DSTN		
	9	12	18	24	9+9	12+12	18+18	4+4	8+8	12+12
Pin Name										
FPD0	B0		B0		FB0		FB0	UR1	UR0	UR0
FPD1	B1		B1		FB1		FB1	UG1	UG0	UG0
FPD2	B2		B2		FB2		FB2	UB1	UB0	UB0
FPD3	B3		B3		FB3		FB3	UR2	UR1	LR0
FPD4	B4		B4		SB0		FB4	LR1	LR0	LG0
FPD5	G0		B5		SB1		FB5	LG1	LG0	LB0
FPD6	G1		B6		SB2		SB0	LB1	LB0	UR1
FPD7	G2		B7		SB3		SB1	LR2	LR1	UG1
FPD8	G3		G0		FG0		SB2	-	UG1	UB1
FPD9	G4		G1		FG1		SB3	-	UB1	LR1
FPD10	G5		G2		FG2		SB4	-	UR2	LG1
FPD11	R0		G3		FG3		SB5	-	UG2	LB1
FPD12	R1		G4		SG0		FG0	-	LG1	UR2
FPD13	R2		G5		SG1		FG1	-	LB1	UG2
FPD14	R3		G6		SG2		FG2	-	LR2	UB2
FPD15	R4		G7		SG3		FG3	-	LG2	LR2
FPD16	-	-	R0		FR0		FG4	-	-	LG2
FPD17	-	-	R1		FR1		FG5	-	-	LB2
FPD18	-	-	R2		FR2		SG0	-	-	UR3
FPD19	-	-	R3		FR3		SG1	-	-	UG3
FPD20	-	-	R4		SR0		SG2	-	-	UB3
FPD21	-	-	R5		SR1		SG3	-	-	LR3
FPD22	-	-	R6		SR2		SG4	-	-	LG3
FPD23	-	-	R7		SR3		SG5	-	-	LB3
FPD24	-	-	-	-	--		FR0	-	-	-
FPD25	-	-	-	-	--		FR1	-	-	-
FPD26	-	-	-	-	--		FR2	-	-	-
FPD27	-	-	-	-	--		FR3	-	-	-
FPD28	-	-	-	-	--		FR4	-	-	-
FPD29	-	-	-	-	--		FR5	-	-	-
FPD30	-	-	-	-	--		SR0	-	-	-
FPD31	-	-	-	-	--		SR1	-	-	-
FPD32	-	-	-	-	--		SR2	-	-	-
FPD33	-	-	-	-	--		SR3	-	-	-
FPD34	-	-	-	-	--		SR4	-	-	-
FPD35	-	-	-	-	--		SR5	-	-	-

7.2.7 CRT Monitor Interface

The SM3110 uses a standard analog RGB interface to drive a CRT monitor. Two customer I/O pins, CIO[1:0], use the standard DDC2B protocol to communicate with monitor. The output of DACs are designed to drive a 37.5 Ohm load. The optimum loading connection is a 75 Ohm resistor to the ground and a 75 Ohm cable load to the VGA connector. The PCB layout for this section needs to be done carefully, since these analog signals are prone to noise.

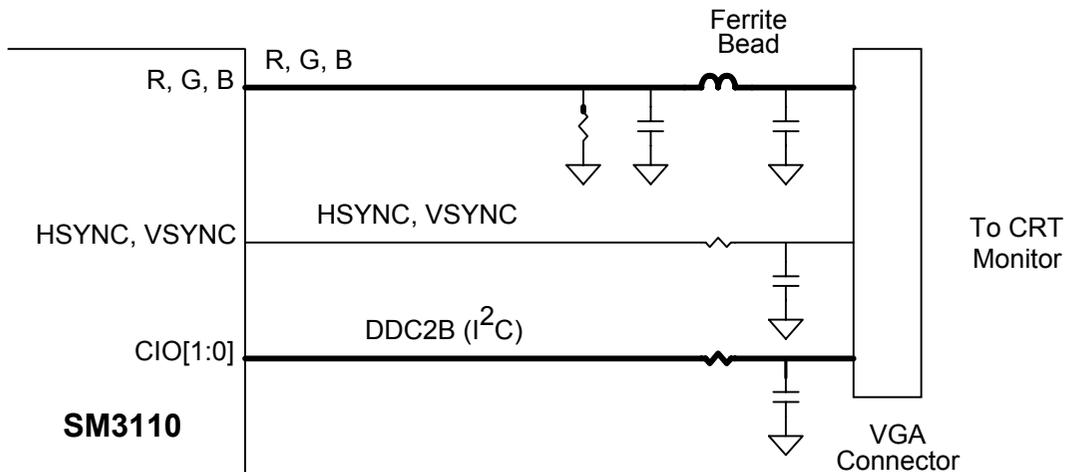


Figure 7-6. CRT Monitor Interface

7.2.8 BIOS ROM Interface

The SM3110 supports external BIOS ROM. The ROM can be EPROM, EEPROM, or flash ROM for easy field update. The ROM size can be either 32 KB or 64 KB, set by a power-on strapping pin.

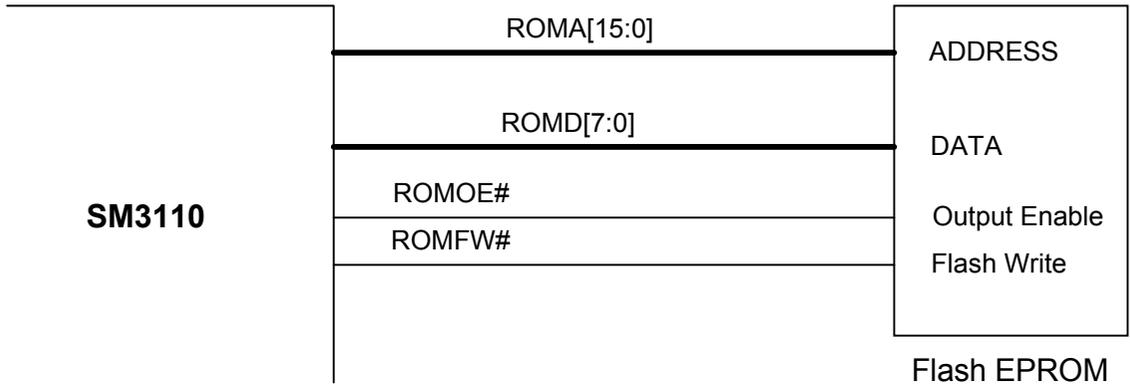


Figure 7-7. BIOS ROM Interface

7.2.9 Analog and Digital TV Encoder Interfaces

The SM3110 supports both analog and digital types of TV encoders. The analog type receives data from RGB lines in a CRT monitor interface. A typical hook-up is shown in Figure 7-8, using Chromtel’s CH7002 analog encoder. The digital type can use either ROM interface or panel interface. For most applications, the ROM interface is where the external TV encoder chip connects. The ROM interface provides 16 bits of RGB data and timing signals (HSYNC, VSYNC and DCLK) for the encoder chip. Any TV encoder chip that can accept 16-bit RGB non-interlace signal can be used. A Chromtel CH7003/7004 is used in the sample design illustrated in Figure 7-9. Please refer to the Alternate Output Function portion of the Datasheet and the Register Summary for detailed signal assignment and register controls.

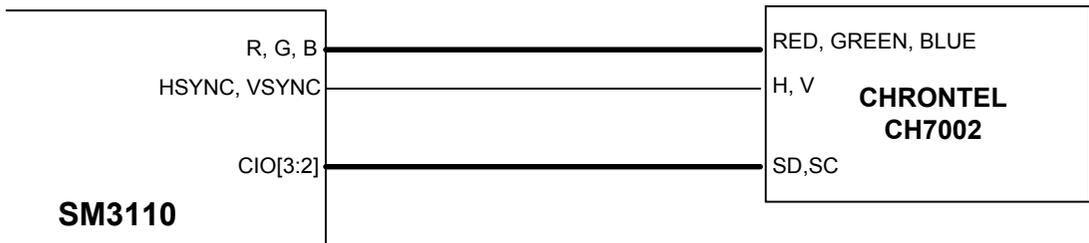


Figure 7-8. Analog TV Encoder Interface

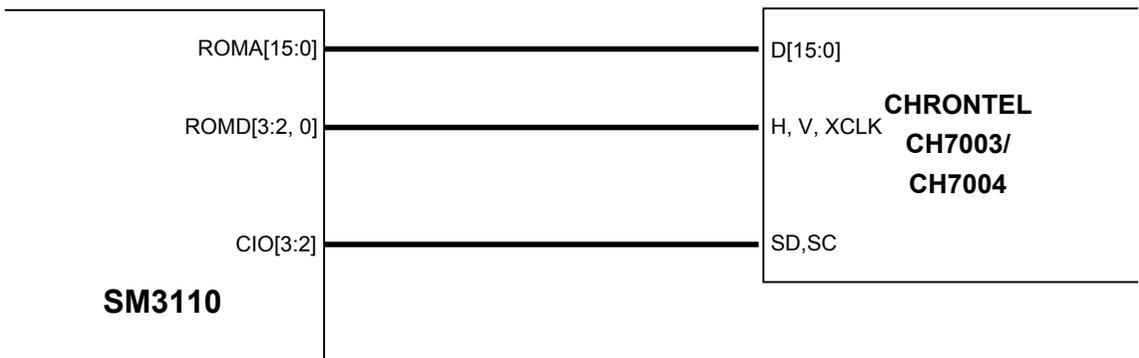


Figure 7-9. Digital TV Encoder Interface

7.2.10 TV Decoder Interface

The SM3110 supports an external TV decoder chip to provide live video window and video capture functions. The interface can connect to either standard PCMCIA ZV Port or Philips' SA7111/7112 (or compatible) TV decoder chip. The connection is straightforward, and the control to the external TV decoder is through the I²C interface.

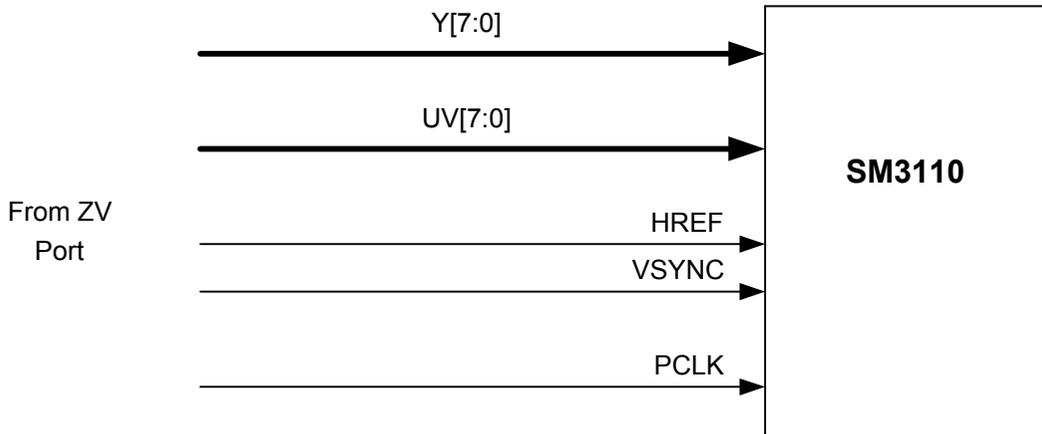


Figure 7-10. ZV Port-TV Decoder Interface

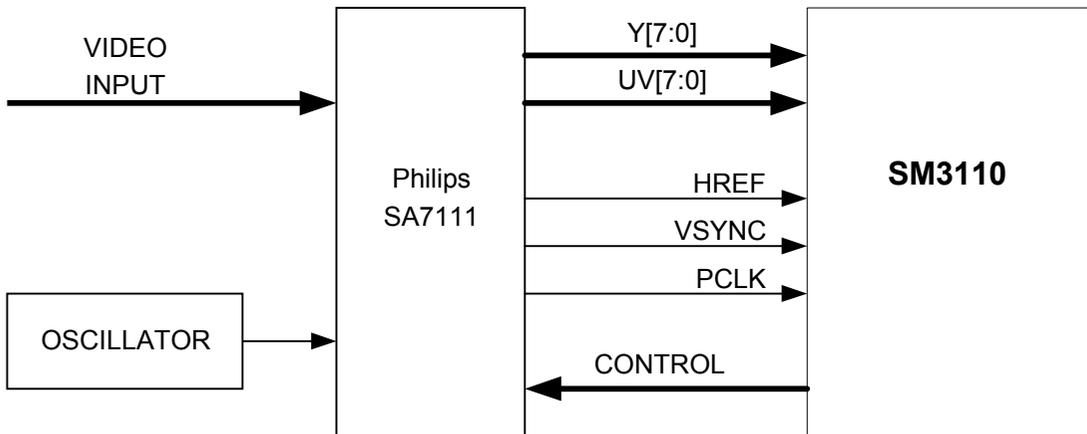


Figure 7-11. Philips TV Decoder Interface

7.3 Power-On Strapping

Eight pins in the ROM interface are also used as Power-On Strapping pins in setting SM3110 configuration within a particular system design. One of these pins is designed to control AGP/PCI interface selection, one is designed to select 32K/64K ROM size, two are for debugging use and four are for customer defined functions, such as panel type. These pins have an internal pull-up resistor and therefore default to “HIGH” without any external component. Refer to the alternate pin function section in the Datasheet.

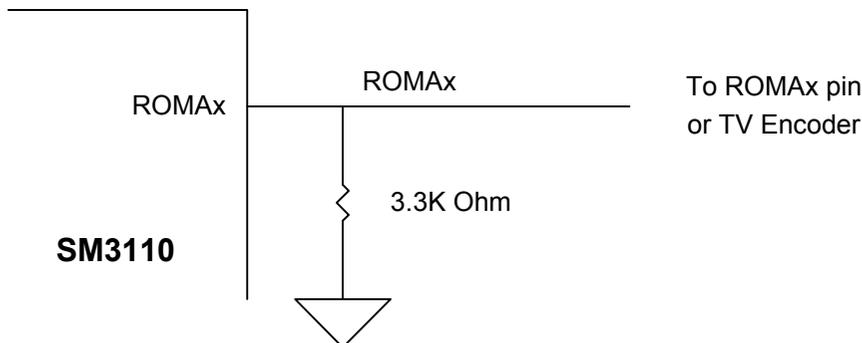


Figure 7-12. Power-on Strapping

7.3.1 Power-on Strapping Pins

Ball No.	Primary Pin Function	Configuration Function		Note
		Default	w/Pull-down Resistor	
C23	ROMA8	CONFIG[0]		User Defined
D22	ROMA9	CONFIG[1]		User Defined
E21	ROMA10	CONFIG[2]		User Defined
F20	ROMA11	CONFIG[3]		User Defined
D23	ROMA12	32K ROM	64K ROM	
E22	ROMA13	Reserved		
F21	ROMA14	Reserved		
E23	ROMA15	AGP Host Interface	PCI Host Interface	

7.3.2 Strapping Method

During reset time, RST# is low and ROMA8 through ROMA15 are forced to be input pins. Each pin has an internal pull-up resistor to set the pin value high, if no external pull-down resistor or other driving force is used to bring the signal level to low. At the trailing edge of reset (RST# changes from low to high), the values of these pins are latched into registers where they can be read by software. Because these functions are set according to the hardware configuration of the board, changes are not allowed. If they are somehow changed, the chip and board may not function properly.

7.3.2.1 External Pull Down Resistor

Each pin has internal pull-up resistor to default the pin value to high. Some pins have inverters between pin and register bit and so the default value in the register is inverted from the default value at the pin. Please refer to the **Registers** section below for the default value of each bit. If default value is the right setup for the bit, no external resistor is required. However, if the other value is needed, a 3.3K Ohm resistor should be added between the corresponding pin and ground.

7.3.2.2 Registers

The register bits that control these configuration functions are listed below.

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+4C0H	7:00	R/O	Configuration Strapping
	<i>Bits</i>		<i>Field</i>
	07		AGP or PCI Select.
			<i>External Pull-down Resistor</i> <i>Value</i> <i>Semantics</i>
			No (default) 0 PCI. <i>Default.</i>
			Yes 1 AGP
	<i>Bits</i>		<i>Field</i>
	06:05		Reserved (must be zero))
	<i>bits</i>		<i>field</i>
	04		ROM Select (32K or 64K).
			<i>External Pull-down Resistor</i> <i>Value</i> <i>Semantics</i>
			No (default) 0 32K
			Yes 1 64K
	<i>Bits</i>		<i>Field</i>
	03:00		User Defined
			<i>External Pull-down Resistor</i> <i>Value</i> <i>Semantics</i>
			Yes 0 User Defined
			No (default) 1 User Defined

7.3.2.3 Strapping Pin Functions

Host Interface

This pin determines what type of host interface the chip is talking to. The default is PCI.

ROM Size

This pin sets the external BIOS ROM size. The default is 32 KB. With a pull-down resistor, the size is set to 64 KB (this is only for the external BIOS ROM that interfaces to the chip through the chip's ROM interface).

User Defined

These pins are for user defined functions, such as panel type. Both the driver and the BIOS can read the register to determine what setting the system requires through any or all of these four pins.

Reserved

There are two reserved power-on strapping pins. These pins must NOT have external pull-down resistors.

7.4 Board-level Testing

The SM3110 currently implements two board-level testing modes: Connectivity Test Mode and Tri-State Mode.

7.4.1 Test Mode Control Pins

There are five pins that control test modes. The TEST# pin should be left unconnected in normal operation, since an internal pull-up sets the chip in normal operation mode. To enter test mode, this pin must be driven to low from outside the chip and maintained at that level during test operation.

The four other pins are used to select different test modes. These four pins have normal functions when TEST# is not connected. The table below lists the pin name (in normal mode and test mode), pin number and setting required for each different test mode.

Table 7-2. Test Mode Control Pins

Pin Name	Pin No.	C15	C2	B1	A1	D4	Test Mode
	Normal	TEST#	FPD28	FPD27	FPD26	FPD25	
	Test	TEST#	TEST3	TEST2	TEST1	TEST0	
Setting		1	-	-	-	-	Normal Operation
		0	0	0	0	0	Reserved
		0	0	0	0	1	Reserved
		0	0	0	1	0	Reserved
		0	0	0	1	1	Reserved
		0	0	1	X	X	Reserved
		0	1	0	X	X	Reserved
		0	1	1	0	X	Reserved
		0	1	1	1	0	IO Connectivity Test
		0	1	1	1	1	Tri-State All Outputs

7.4.2 Test Modes

The I/O Connectivity Test Mode helps in identifying soldering problems on the PCB. In this mode, all digital pins except the TEST# pin and the TEST[3:0] pins are arranged to connect to either input or output of NAND gate array.

To perform this test, set TEST[3:0] to 0xE and apply logical highs to all input pins. All the output pins should then be at logical low. Any output that is not in the correct logic state indicates one of its inputs is either open or shorted to logic low. Next, toggle one input at a time from logic high to logic low and back to logic high; the corresponding output must change from logic low to logic high and back to logic low. All other outputs must stay at logic low. If the corresponding output does not change state, the input pin is open. If another output changes states with the corresponding output, one of its inputs is shorted to this input.

The Tri-State Test Mode isolates the chip from the board during the debugging of other board components. Set TEST[3:0] to 0xF to force all output drivers in output pins and bi-directional pins into tri-state mode.

The connection map is listed below, followed by tables of untested pins and unused pins.

Input		Output	
Pin	Sig. Name	Pin	Sig. Name
Y4	INTA#	H22	IOTST7
AC2	REQ#		
AC4	RBF#		
AC6	SB_STB		
Y6	SBA1	AC1	CLK
AA8	SBA7		
AC8	AD31		
AA9	AD28		
Y8	SBA5	AA4	RST#
AB10	AD27		
Y11	AD24		
Y13	AD22		
AB6	SBA2	AC3	ST0
Y9	AD30		
AC9	AD29		
AA12	C/BE3#		
AB2	CLKRUN#	Y5	GNT#
AC5	SBA0		
Y7	SBA3		
AC7	SBA4		
AB8	SBA6	AB4	ST2
AC10	AD25		
AB12	AD23		
Y15	AD16		
Y10	AD26	AA5	ST1
AC11	AD_STB1		
AA13	AD20		
AC13	AD19		

Input		Output	
Pin	Sig. Name	Pin	Sig. Name
AC12	AD21	AA11	IDSEL
Y14	AD18		
AC14	C/BE2#		
AA16	STOP#		
AB14	AD17	AC19	M66EN
Y16	TRDY#		
AB16	DEVSEL#		
AC18	AD10		
AA15	FRAME#	F23	IOTST1
AC16	C/BE1#		
Y19	AD13		
AB19	AD8		
AC15	IRDY#	G23	IOTST2
AC17	AD14		
AA19	AD11		
Y23	AD4		
AA17	PAR	G22	IOTST3
AA20	C/BE0#		
AC22	AD5		
AA23	AD2		
AB18	AD12	G21	IOTST4
AC21	AD7		
AB23	AD0		
Y18	AD15	G20	IOTST5
AB22	AD3		
Y21	AD6		
AC20	AD_STB0	H23	IOTST6
Y20	AD9		
AC23	AD1		

Application Notes

Input		Output	
Pin	Sig. Name	Pin	Sig. Name
K22	CIO6	H20	IOTST8
K20	CIO4		
R3	CIO1		
U2	HSYNC		
E23	ROMA15	J23	MCLKOUT
E21	ROMA10		
C23	ROMA8		
E22	ROMA13	J21	SCLKOUT
F20	ROMA11		
L21	ZVPCLK		
A23	ROMA4		
F21	ROMA14	J20	PCLKOUT
D23	ROMA12		
B23	ROMA5		
A21	ROMA1		
D22	ROMA9	T20	ZVVSYNC
E20	ROMA7		
A22	ROMA3		
C20	ROMD7		
C22	ROMA6	T21	ZVHREF
D20	ROMA2		
A20	ROMD6		
D18	ROMD3		
B21	ROMA0	T23	ZVY7
D19	ROMD5		
A18	ROMD2		
D16	ROMOE#		
A19	ROMD4	R20	ZVY6
D17	ROMD1		
B16	ROMXR#		

Input		Output	
Pin	Sig. Name	Pin	Sig. Name
D14	FPDE		
A17	ROMD0	R21	ZVY5
D15	ROMFW#		
A15	FPBL		
A13	LP		
A16	ROMWR#	R22	ZVY4
A14	FP		
D12	FPVEE		
A12	FPD1		
B13	FPVCC	R23	ZVY3
D13	FPCLKO		
A11	FPD3		
A9	FPD9		
C12	FPD0	P20	ZVY2
D10	FPD4		
A10	FPD6		
A8	FPD11		
D11	FPD2	P22	ZVY1
C9	FPD8		
D7	FPD12		
A7	FPD14		
B10	FPD5	P23	ZVY0
D8	FPD10		
B7	FPD13		
D6	FPD15		
C6	FPD16	N20	ZVUV7
A4	FPD19		
C4	FPD23		
D1	FPD31		
D9	FPD7	N21	ZVUV6

Input		Output	
Pin	Sig. Name	Pin	Sig. Name
A6	FPD17		
A3	FPD21		
D2	FPD30		
A5	FPD18	N23	ZVUV5
D5	FPD22		
C1	FPD29		
E2	FPD34		
B4	FPD20	M20	ZVUV4
A2	FPD24		
E4	FPD32		
F1	FPD38		
E3	FPD33	M21	ZVUV3
E1	FPD35		
G4	FPD39		
H2	FPD44		
F4	FPD36	M22	ZVUV2
H4	FPD42		
H1	FPD45		
L22	CIO2		
F3	FPD37	M23	ZVUV1
G1	FPD41		
H3	FPD43		
K23	CIO7		
G2	FPD40	L20	ZVUV0
J4	FPD46		
J2	FPD47		
T4	BLANK#		
K21	CIO5	U23	ZVCREF
L23	CIO3		
R4	CIO0		
U1	VSYNC		

Application Notes

Untested Pins	
T1	DCLK
U4	RSET
V1	COMP
V4	RED
V3	GREEN
W1	BLUE
AA1	AGPCNTL#
Y22	AGPVref
W21	XCLK1
W20	XCLK0
C15	TEST#
D4	FPD25
A1	FPD26
B1	FPD27
C2	FPD28

Unused Pins	
AA7	Reserved17
AB21	Reserved21
J1	Reserved1
K1	Reserved4
K3	Reserved3
K4	Reserved2
L1	Reserved7
L2	Reserved6
L4	Reserved5
M1	Reserved8
M3	Reserved9
M4	Reserved10
N1	Reserved11
N2	Reserved12
N4	Reserved13
P1	Reserved14
P3	Reserved15
P4	Reserved16
R1	VREFIN
R2	VREFOUT
Y12	Reserved19
Y17	Reserved20

7.5 Programmable (Peripheral) I/O

There are two types of programmable I/O in the SM3110. The first is through eight 'customer' I/O pins and the second type is peripheral I/O which uses the ROM interface address and data pins plus two command pins.

7.5.1 Customer I/O

The eight customer I/O pins are controlled directly by register bits. These pins can be individually programmed to be either input or output pins. If set to be input, the value at the pin (either one (high) or zero (low)) can be read from corresponding register bit by software through the normal register read methods. If set as output, writing to the corresponding register bit will set the output value at the pin.

7.5.1.1 Customer Pins

Four of the eight pins are used for I²C function to support DDC2B for CRT monitor and to provide TV encoder/decoder control. The remaining four pins are for customer-definable functions. They can be customized for each design and could control a number of different circuits on the system. Please refer to the table below for the pin number and its corresponding signal name.

Table 7-3. Customer I/O Pins

Pin	Signal Name	Description
K23	CIO7	User programmer input/output [7]
K22	CIO6	User programmer input/output [6]
K21	CIO5	User programmer input/output [5]
K20	CIO4	User programmer input/output [4]
L23	CIO3	User programmer input/output [3]/I ² C Data Line
L22	CIO2	User programmer input/output [2]/I ² C Clock Line
R3	CIO1	User programmer input/output [1]/DDC2B Data Line
R4	CIO0	User programmer input/output [0]/DDC2B Clock Line

7.5.1.2 Customer Registers

There are three bytes of register to control these pins. The first one is READ-ONLY and is for getting signal level of pins. The second one is for output level at pin. The third one is to control direction of pin. The offset and function of each register bit is described in following tables.

CIO Input

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+410H	07:00	R/O	Configurable Input Port
	<i>Bits</i>	<i>Field</i>	
	07:00	Input.	
		<i>Value</i>	<i>Semantics</i>
		0	Level at pin input is low.
		1	Level at pin input is high.

CIO Output

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+411H	07:00	R/W	Configurable Output Port
	<i>Bits</i>	<i>Field</i>	
	07:00	Output. Write	
		<i>Value</i>	<i>Semantics</i>
		0	Level to pin output is low.
		1	Level to pin output is high.
	<i>Bits</i>	<i>Field</i>	
	07:00	Output. Read	
		<i>Value</i>	<i>Semantics</i>
		0	Level last written to pin output is low.
		1	Level last written to pin output is high.

CIO Control

<i>Offset</i>	<i>Field</i>	<i>Access</i>	<i>Function</i>
+412H	07:00	R/W	Configurable Control Port
	<i>Bits</i>	<i>Field</i>	
	07:00	Enable. <i>Default</i> is FFH, all bits disabled.	
		<i>Value</i>	<i>Semantics</i>
		0	Output is enabled and pulled low.
		1	Output is disabled.

7.5.2 Peripheral I/O

The peripheral I/O shares ROM interface pins with other functions. Bit 3 and 2 in register offset 400H control the function of these pins. To use this function, both bits should be set to 0 for ROM function. Peripheral I/O functions share the ROMA[15:0] and ROMD[7:0] pins with the ROM interface but use different address range and command pins.

Peripheral I/O occupies a 64KB address range, offset 1FE0000H to 1FEFFFFH, in each 32MB of memory space within total 128MB of memory space reserved. The lower 16-bit of address will output to ROMA[15:0] pins during the command cycle. For read operation, ROMXR# will be asserted and data at ROMD[7:0] pins will be read. For write operation, ROMXW# will be asserted and data will place at ROMD[7:0] pins. Refer to the SM3110 Datasheet for specific timing values.

7.5.2.1 Peripheral I/O Pins

Pin	Signal Name	POWERUP Configuration	Alternate Function	Description
E20, C22, B23, A23, A22, D20, A21, B21	ROMA[7:0]		VDO[7:0]	ROM Address [7:0]/ Digital Data[7:0] for TV/TMDS/LVDS Encoder
C23	ROMA8	CONFIG[0]	VDO8	ROM Address [8]/ Digital Data[8] for TV/TMDS/LVDS Encoder
D22	ROMA9	CONFIG[1]	VDO9	ROM Address [9]/ Digital Data[9] for TV/TMDS/LVDS Encoder
E21	ROMA10	CONFIG[2]	VDO10	ROM Address [10]/ Digital Data[10] for TV/TMDS/LVDS Encoder
F20	ROMA11	CONFIG[3]	VDO11	ROM Address [11]/ Digital Data[11] for TV/TMDS/LVDS Encoder
D23	ROMA12	32K/64K ROM	VDO12	ROM Address [12]/ Digital Data[12] for TV/TMDS/LVDS Encoder
E22	ROMA13	INT/EXT PLL	VDO13	ROM Address [13]/ Digital Data[13] for TV/TMDS/LVDS Encoder
F21	ROMA14	INT/EXT SCLK	VDO14	ROM Address [14]/ Digital Data[14] for TV/TMDS/LVDS Encoder
E23	ROMA15	AGP/PCI	VDO15	ROM Address [15]/ Digital Data[15] for TV/TMDS/LVDS Encoder
C20, A20, D19, A19	ROMD[7:4]			ROM Data [7:4]
D18	ROMD3		HSYNC2	ROM Data [3]/ HSYNC for TV/TMDS/LVDS Encoder
A18	ROMD2		VSYNC2	ROM Data [2]/ VSYNC for TV/TMDS/LVDS Encoder
D17	ROMD1		DCLK2	ROM Data [1]/ DCLK for TV/TMDS/LVDS Encoder
A17	ROMD0		BLANK2#	ROM Data [0]/ BLANK# for TV/TMDS/LVDS Encoder
D15	ROMFW#			Flash ROM Flash Write Command
D16	ROMOE#			ROM Output Enable
A16	ROMWR#			Peripheral I/O Write Command
B16	ROMXR#			Peripheral I/O Read Command

7.5.2.2 Peripheral I/O Registers

There are six register bits which control these pins: bits [3:2] controls pin function, bits [6:4] control the command pulse width and bit 7 controls the address setup time (bits [1:0] are not part of this function). The details are shown below:

Offset	Field	Access	Function
+400H	07:00	R/W	Peripheral I/O Port Configuration
	Bits	Field	
	07	Address Setup.	
		Value	Semantics
		0	2 clocks. <i>Default.</i>
		1	1 clock.
	Bits	Field	
	06:04	Strobe Width.	
		Value	Semantics
		0	12 clocks. <i>Default.</i>
		1-7	<i>value</i> +4 clocks.
	Bits	Field	
	03:02	ROM port configuration	
		Value	Semantics
		11	Display 0
		10	Display 1
		01	<i>Reserved.</i>
		00	ROM
	01:00	LCD Port Configuration.	
		Value	Semantics
		11	Display 0
		10	Display 1
		01	<i>Reserved.</i>
		00	LCD

Register bits [7:4] of offset 400H control the timing. Bit 7 selects address setup time, either at the default of two system clocks (SCLK) or only one system clock. ROMXR# and ROMXW# strobes (commands) pulse width is set by bits [6:4]. The pulse width is the value in these bits plus 4 system clocks (total range from 5 to 12 system clocks), where 000 in these bits is the default and is treated as 8 to get a total of 12 system clocks of pulse width. The address hold time is fixed at one system clock.

Bit value in 6:4	Clock Value of Bits 6:4	Width of address strobe in System Clocks
000 (default)	8	12
001	1	5
010	2	6
011	3	7
100	4	8
101	5	9
110	6	10
111	7	11

A Appendix A



A.1 Pixel Format Information

A.1.1 Enhanced Data Format

Pixels in RGB and ARGB, various YUV/YCbCr formats and CLUT4/8 are accepted for processing and display. Pixels are addressed in ascending order corresponding to a left to right, top to bottom (typical raster scan) of the display. All formats may be packed, i.e., there are no padding bytes between pixels at the end of one line and pixels at the start of the next line. Some facilities do support unpacked organizations of the pixel bit map by defining a stride to allow access or display of rectangular regions within a bitmap that are narrower than the line width.

A.1.2 RGB Pixel Formats

There are four RGB (15b, 16b, 24b, and 32b) pixel formats.

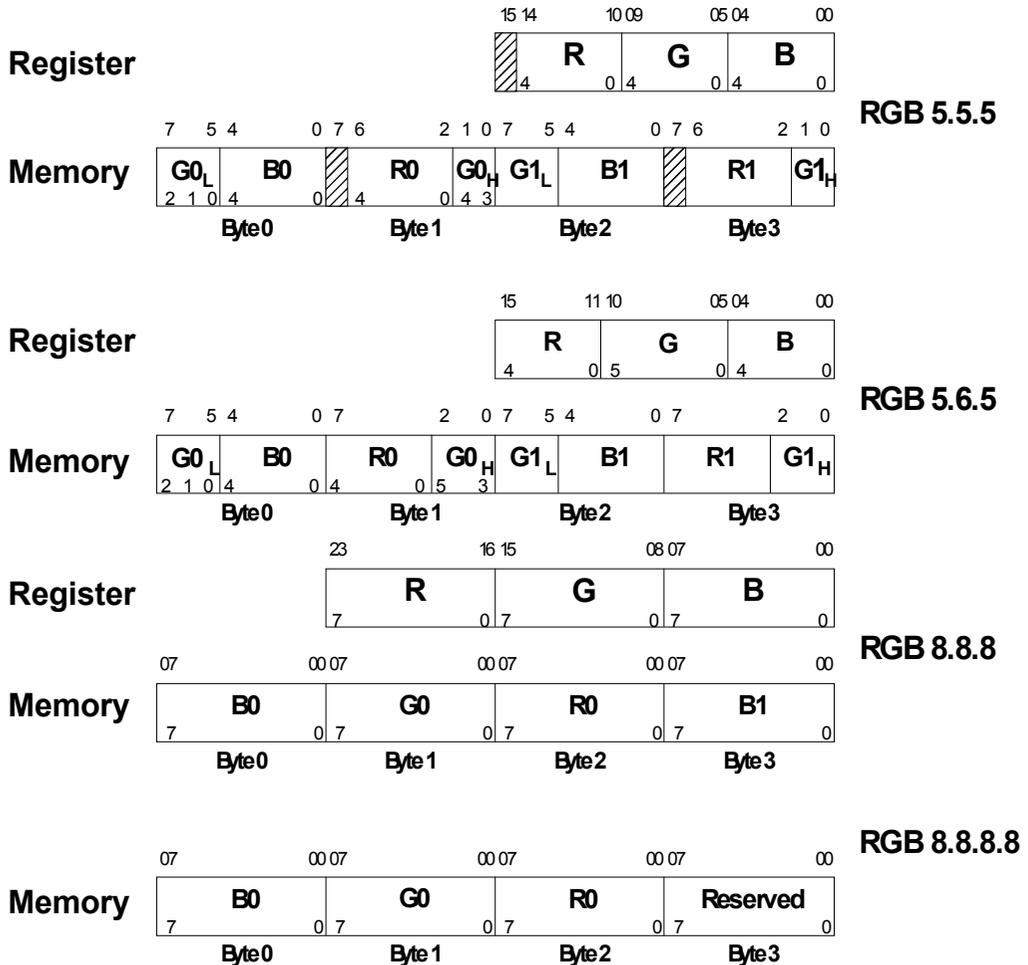


Figure A-1. RGB Pixel Representation

15b- xRGB 1.5.5.5

The most significant bit is unused and ignored during display.

16b - RGB 5.6.5

Note that the Green field pushes the Red field towards the most significant bit. It is not upwards compatible with the 5.5.5 format.

24b - RGB 8.8.8

This is standard packed 24bpp. Each line of a 24bpp bitmap is assumed to start on a double word boundary.

32b - xxRGB 8.8.8.8

The most significant byte is unused and ignored during display. This format is for display only; the 2D engine does not currently support this format.

A.1.3 Palettized/Color Lookup Table
CLUT 8

8 bit color index format is supported.

CLUT 4

4-bit packed color index format is supported for resolutions of 640×480 and above. Note that the 4-bit pixels are arranged left to right within a byte, i.e., the left-most pixel (lower/even) pixel is stored in bits [7:4] of each byte.

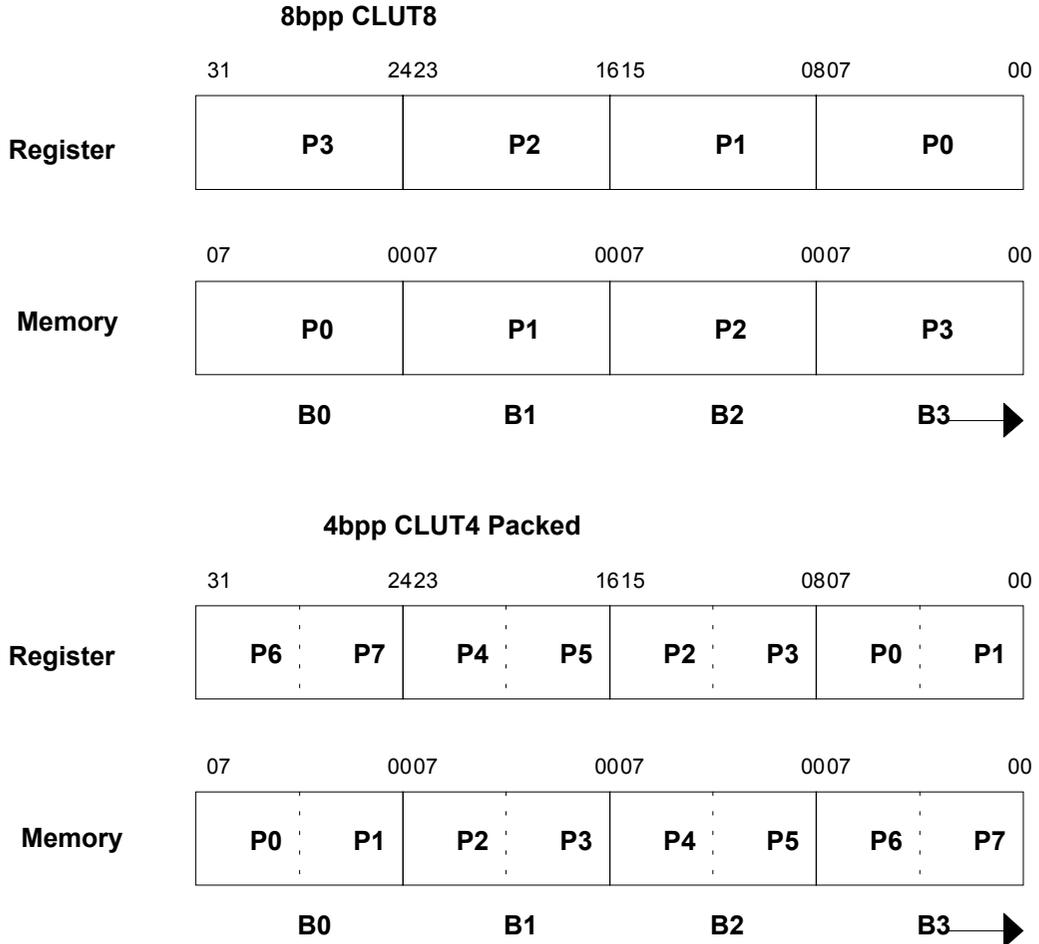


Figure A-2. CLUT 4/8b Pixel Representation

A.1.4 CLUT Data Format

The palette logically consists of 256 entries of 3×6, formatted as 6.6.6 RGB. It is accessed from the CPU as xRGB 8.8.8.8, the most significant byte is not used. CLUT color component data ranges from 0...63; the least significant 6 bits of each byte are physically stored in the CLUT RAM.

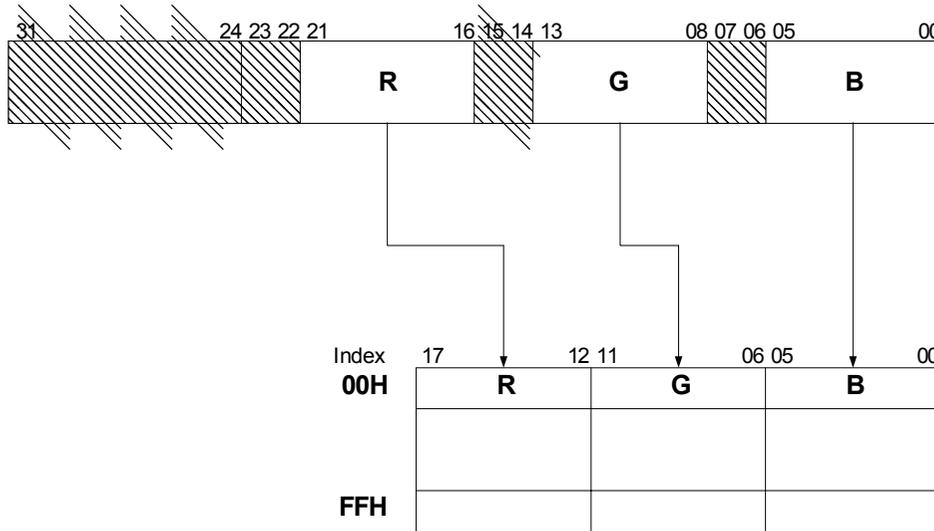


Figure A-3. VGA/2D CLUT Data Format

A.2 3D Data Formats

A.2.1 Coordinates

Single precision floating point 1.8.23
 32-bit fixed point 0.11.21

A.2.2 RGB Pixel Formats

The same four RGB pixel formats (15b, 16b, 24b, and 32b) are supported as those defined for 2D operations (see Section A.1.2.).

A.2.3 ARGB Pixel Formats

There are three additional ARGB formats supported: 1.5.5.5, 4.4.4.4 and 8.8.8.8

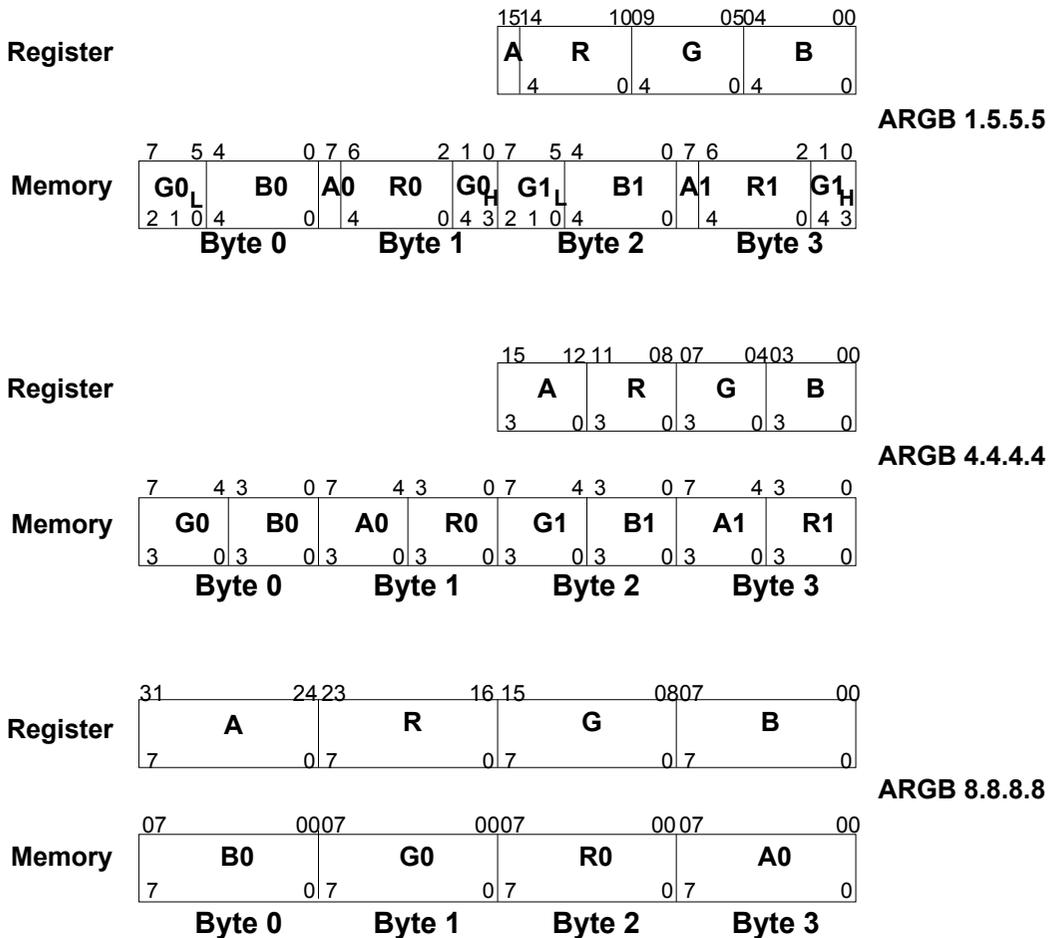


Figure A-4. ARGB Pixel Representation

A.2.4 Texel Formats

Palettized CLUT	1, 2, 4, 8
16b ARGB	1.5.5.5, 4.4.4.4
16b RGB	x.5.5.5, 5.6.5

Palettized/Color Lookup Table

The texture palette logically consists of 256 entries of 3×6, formatted as 6.6.6 RGB. It is accessed from the CPU as xRGB 8.8.8.8, the most significant byte is not used. The most significant 6 bits of each byte are physically implemented and the least significant 2 bits are discarded when writing to and forced to 0 when reading from the palette RAM. Note that this is different from the CLUT format for standard VGAs.

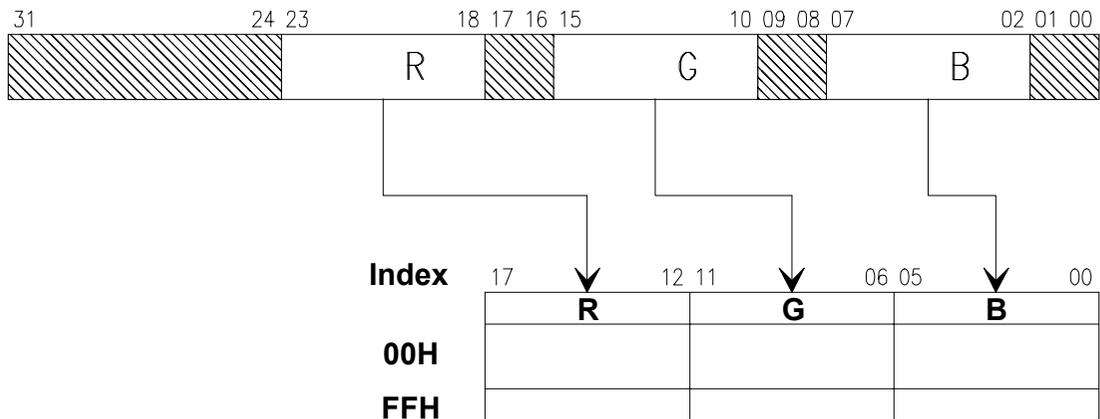


Figure A-5. 3D CLUT Data Format

CLUT 1, 2, 4

1, 2 and 4-bit packed color index formats are supported, with the pixels arranged left to right within a byte, i.e., the left-most (lower/even) pixel is stored starting in bit [7] of each byte.

CLUT 8

8-bit color index format is supported. Because of the little Endian representation, bytes are swapped within a doubleword when registers are written to memory. The relationship of individual pixels (in 1, 2, 4bpp CLUT) is not changed between registers and memory.

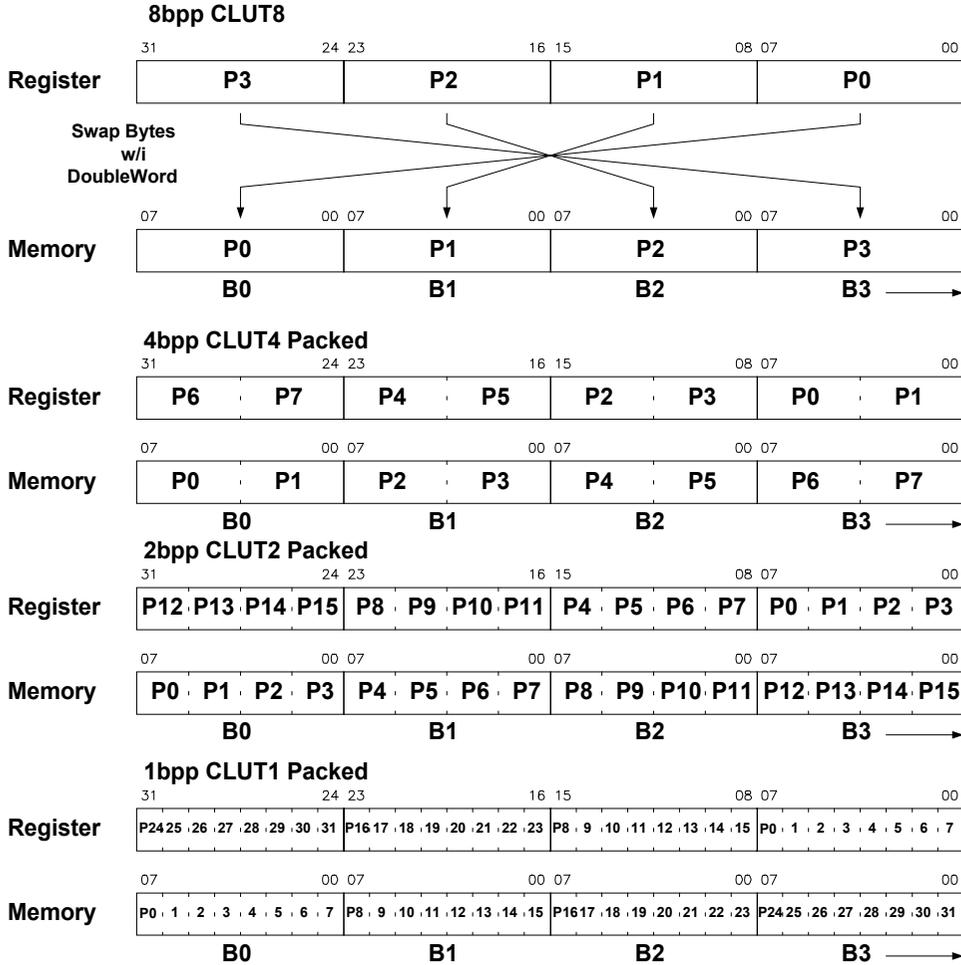


Figure A-6. CLUT 1/2/4/8b Pixel Representation

A.3 Video Formats

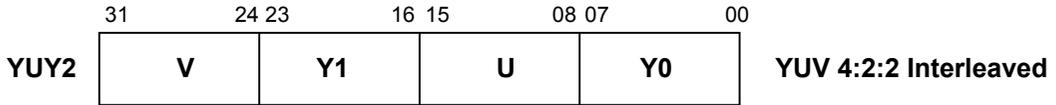


Figure A-7. YUV/YCbCr Pixel Representation

Interleaved YUV 4.2.2 (FourCC = YUY2) 16bpp format is supported.

The numeric format of the Y (luminance) pixels is an integer ranging from 16...235.

The numeric format of the U/V (chroma) pixels is an integer ranging from 16...240.

These are scaled upon display up to 0...255.

A.3.1 Cursor

A 2bpp mono and 16bpp color cursor are supported. The mono cursor format consists of interleaved words (16 pixels) of AND and XOR data. The color cursor consists of sequential 16b words with a 1.5.5.5 format. The most significant bit is the AND plane data and the remaining 15 bits contain the RGB color data.

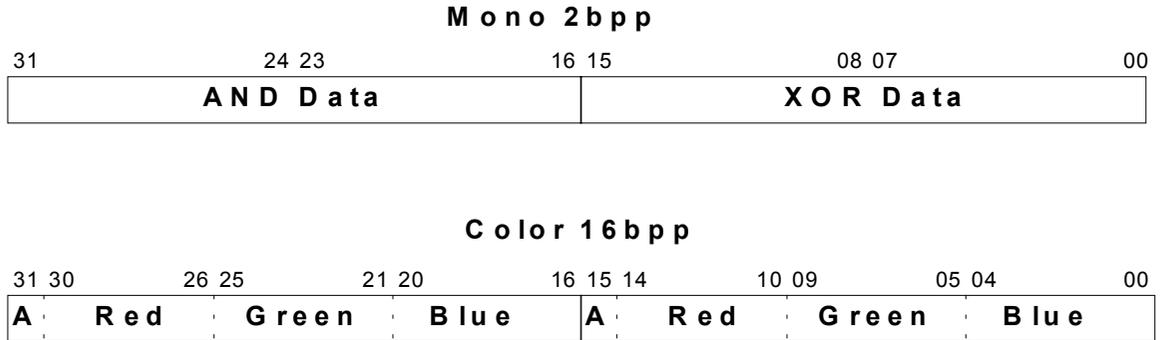


Figure A-8. Cursor Formats

A.3.2 VGA Data Formats

In addition, standard VGA planar and packed pixel (linear) formats are supported for compatibility. To implement extended (higher resolution) SVGA, the address range must be extended beyond the 64KB or 128KB that is addressable by a standard VGA.

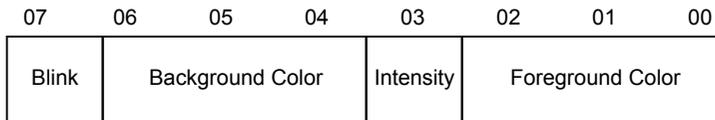
A.3.3 Planar

For pixels of 4 bits or less (16 colors or less), a planar representation is used. Four independently addressable bit planes (I, R, G and B) are accessed in parallel. Each pixel is composed of up to 4 bits from each of the four planes. The amount of memory required depends on the resolution. For a plane greater than 64KB, each 128KB can be addressed as either two segments of 64KB (configuration 2) or a single 128KB segment (configuration 1).

The read map select register in the VGA graphics controller and the map mask register in the VGA sequencer must be configured to control the access to individual planes.

A.3.4 Text

In text modes, data is arranged as a series of 16-bit words storing the character code (even byte) and attribute (odd byte). Character pairs are arranged starting at 0 (offset from the base of B0000H in mono or B8000 in color) with increasing addresses arranged across the screen from left to right, then down the screen. The (byte) stride of characters (from one character line to the next) is two times the number of characters per line because two bytes are used to store each character and attribute. The lowest resolution VGA text display of 40x25 is 1000 characters occupying 2000 bytes; the highest SVGA text resolution of 132x60 is 7920 characters occupying 15840 bytes. The format of a text attribute byte and the encoding of the colors are shown in Figure A-9. VGA Text Attributes.



	Default Color	Mono
0	Black	Black
1	Blue	Underline
2	Green	
3	Cyan	
4	Red	
5	Magenta	
6	Brown	
7	White	Color

Figure A-9. VGA Text Attributes

A.3.5 Packed Pixel/Linear

This format is identical to that used in enhanced modes: 4bpp and 8bpp are supported and each pixel is stored consecutively in ascending order corresponding to a left to right, top to bottom traversal of the display.

B Appendix B



B.1 Graphics Modes Support

B.1.1 Video Display Window Memory Constraints

Presenting television images in one or both of the possible video displays (LCD and/or CRT) is dependent upon the memory required for the size and color depth of the physical displays; the memory required for the dimensions and color depth of the individual television display windows within those physical displays; and the amount of memory required by the motion compensation buffers for the desired video displays. Fortunately, the motion compensation buffers can easily be accommodated by SM3110's embedded 4MB RAM. Motion compensation requires only about three megabytes to display a full-sized NTSC stream and about three and a half megabytes for PAL, leaving plenty of memory for standard display operations.

B.1.2 Definitions

The LCD or CRT will be termed the 'physical display', and the color depth of the physical display will be referred to as the '2D background' (this is also sometimes described as the 'desktop'). A video stream is displayed in a 'video display window'.

Note: The video display window will always have the same color depth as the 2D background, courtesy of the color space conversion built into the video processing stream.

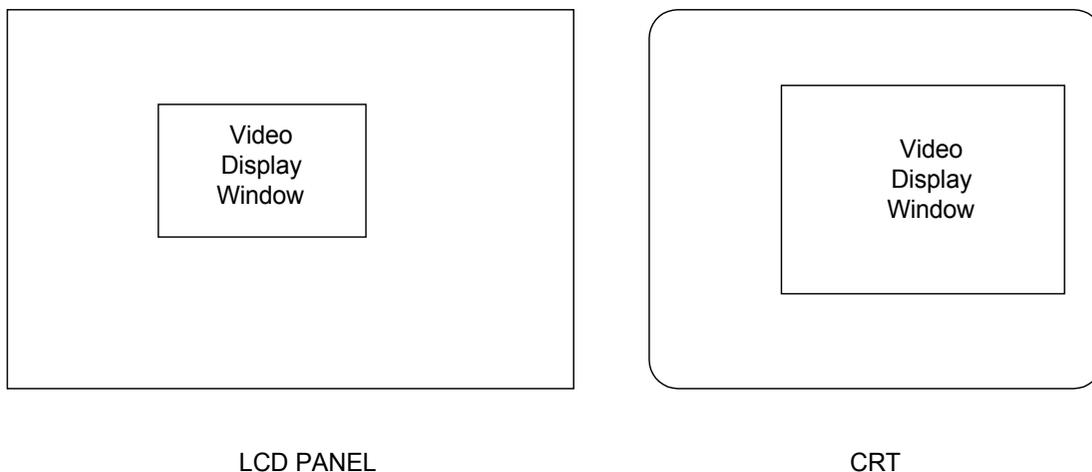


Figure B-1. Video Display Terminology

B.1.3 Motion Compensation Memory Requirements

The motion compensation buffers consist of the video front, video back and processing buffers. The processing buffers require (given that the three-frame MPEG processing model is used)

$$1.5 \text{ bytes/pixel} \times 3 \times (\text{Frame size: } 720 \times 480 \text{ for NTSC, } 720 \times 576 \text{ for PAL}) = \text{bytes}$$

That's 1,555,200 bytes for NTSC and 1,866,240 bytes for PAL.

The video front and video back buffers require

$$2 \text{ bytes/pixel} \times 2 \text{ buffers} \times (\text{video display window frame size}^*) = \text{bytes}$$

That value is 1,382,400 bytes for full-size NTSC and 1,658,880 bytes for full-size PAL.

Note: Use the full frame size in this calculation unless the video display window size will be less than half the input frame size. The pre-scaler will not scale down video (and thereby reduce memory requirements) unless it is asked to reduce the size to less than half the original size.

B.2 Graphics Modes Summary

Parameter	Bytes
Memory Size (4Meg)	4,184,064
System Overhead in Memory	131,072
Icon Overhead	1,024
Memory Available For Display and Video	4,051,968
NTSC Video Source Standard Size (720x480; 16 bpp)	1,382,400
NTSC Video Quarter-Screen Size (Horizontal Predecimation)	691,200
PAL Video Source Standard Size (720x576; 16 bpp)	1,658,880
PAL Video Quarter-Screen Size (Horizontal Predecimation)	829,440
NTSC Motion Comp Memory Size (720x480x3; 12 bpp“plus 720x480x2, 16bpp”)	2,937,600
PAL Motion Comp Memory Size (720x576x3; 12 bpp)“plus 720x576x2, 16bpp”)	3,525,120

Note: Simultaneous video (same size, color depth) on two displays uses the same memory and supports the same modes as single display.

Note on Table Nomenclature:

1. Video is double-buffered
2. LCD: DSTN requires more memory than TFT
3. Single 1/4 video: classic ““small-window” ” video via video capture” allows memory requirement reduction using horizontal predecimation.

B.2.1 Single Display with NTSC Video

Size	Color Depth by Memory Load Configuration (bpp)								
	Display Only		Display + 1/4 NTSC Video		Display + 1 NTSC Video		Display + NTSC DVD		
	TFT/ CRT	DSTN	TFT/ CRT	DSTN	TFT/ CRT	DSTN	TFT/ CRT	DSTN	
320 x 240	32	32	32	32	32	32	32	32	32
640 x 480	32	32	32	32	32	32	24	24	24
800 x 600	32	32	32	32	32	32	16	16	16
1024 x 768	32	32	32	32	24	24	8	8	8
1280 x 1024	24	16	16	16	16	8	--	--	--
1600 x 1200	16	8	8	8	8	8	--	--	--

Note: -- = mode not supported

B.2.2 Single Display with PAL Video

Size	Color Depth by Memory Load Configuration (bpp)								
	Display Only		Display + 1/4 PAL Video		Display + 1 PAL Video		Display + PAL DVD		
	TFT/ CRT	DSTN	TFT/ CRT	DSTN	TFT/ CRT	DSTN	TFT/ CRT	DSTN	
320 x 240	32	32	32	32	32	32	32	32	32
640 x 480	32	32	32	32	32	32	8	8	8
800 x 600	32	32	32	32	32	32	8	--	--
1024 x 768	32	32	32	24	24	16	--	--	--
1280 x 1024	24	16	16	16	8	8	--	--	--
1600 x 1200	16	8	8	8	8	8	--	--	--

Note: -- = mode not supported

B.2.3 MyView™ Dual display with NTSC Video

Size		Color Depth By Memory Load Configuration (bpp)							
First Display TFT or DSTN	Second Display CRT	Displays Only	Displays + 1/4 NTSC Video	Displays + 1 NTSC Video	Displays + 1+1/4 NTSC Videos	Displays + 2 NTSC Videos	Displays + NTSC DVD		
		CRT and TFT DSTN	CRT and TFT DSTN	CRT and TFT DSTN	CRT and TFT DSTN	CRT and TFT DSTN	CRT and TFT DSTN	CRT and TFT DSTN	TFT
320 x 240	320 x 240	32 32	32 32	32 32	32 32	32 32	32 32	32 32	32 32
320 x 240	640 x 480	32 32	32 32	32 32	32 32	32 32	24 24	16 16	16 16
320 x 240	800 x 600	32 32	32 32	32 32	24 24	24 24	16 16	16 8	16 8
320 x 240	1024 x 768	32 32	24 24	24 24	16 16	16 16	8 8	8 8	8 8
320 x 240	1280 x 1024	16 16	16 16	8 8	8 8	-- --	-- --	-- --	-- --
320 x 240	1600 x 1200	16 16	8 8	8 8	-- --	-- --	-- --	-- --	-- --
640 x 480	320 x 240	32 32	32 32	32 32	32 32	24 24	16 16	16 16	16 16
640 x 480	640 x 480	32 32	32 32	32 32	24 24	16 16	8 8	8 8	8 8
640 x 480	800 x 600	32 32	32 32	24 24	16 16	16 16	8 8	8 8	8 8
640 x 480	1024 x 768	24 24	24 24	16 16	8 8	8 8	8 8	8 --	8 --
640 x 480	1280 x 1024	16 16	16 16	8 8	8 8	-- --	-- --	-- --	-- --
640 x 480	1600 x 1200	8 8	8 8	8 8	-- --	-- --	-- --	-- --	-- --
800 x 600	320 x 240	32 32	32 32	32 32	24 24	16 16	16 16	16 8	16 8
800 x 600	640 x 480	32 32	32 32	24 24	16 16	8 8	8 8	8 8	8 8
800 x 600	800 x 600	32 32	24 24	16 16	16 16	8 8	8 --	8 --	8 --
800 x 600	1024 x 768	24 24	16 16	16 16	8 8	8 --	-- --	-- --	-- --
800 x 600	1280 x 1024	16 16	8 8	8 8	8 8	-- --	-- --	-- --	-- --
800 x 600	1600 x 1200	8 8	8 8	8 8	-- --	-- --	-- --	-- --	-- --
1024 x 768	320 x 240	32 32	24 24	24 16	16 16	8 8	8 8	8 8	8 8
1024 x 768	640 x 480	24 24	24 16	16 16	8 8	8 8	8 --	8 --	8 --
1024 x 768	800 x 600	24 24	16 16	16 8	8 8	8 --	-- --	-- --	-- --
1024 x 768	1024 x 768	16 16	16 16	8 8	8 8	-- --	-- --	-- --	-- --
1024 x 768	1280 x 1024	8 8	8 8	8 8	-- --	-- --	-- --	-- --	-- --
1024 x 768	1600 x 1200	8 8	8 8	-- --	-- --	-- --	-- --	-- --	-- --
1280 x 1024	320 x 240	16 16	16 16	8 8	8 8	-- --	-- --	-- --	-- --
1280 x 1024	640 x 480	16 16	16 8	8 8	8 8	-- --	-- --	-- --	-- --
1280 x 1024	800 x 600	16 16	8 8	8 8	8 --	-- --	-- --	-- --	-- --
1280 x 1024	1024 x 768	8 8	8 8	8 8	-- --	-- --	-- --	-- --	-- --
1280 x 1024	1280 x 1024	8 8	8 8	8 --	-- --	-- --	-- --	-- --	-- --
1280 x 1024	1600 x 1200	8 8	8 --	-- --	-- --	-- --	-- --	-- --	-- --

Note: -- = mode not supported

B.2.4 MyView™ Dual display with PAL Video

Size		Color Depth By Memory Load Configuration (bpp)											
First Display TFT or DSTN	Second Display CRT	Displays Only		Displays + 1/4 PAL Video		Displays + 1 PAL Video		Displays + 1+1/4 PAL Videos		Displays + 2 PAL Videos		Displays + PAL DVD	
		CRT and TFT DSTN	CRT and TFT DSTN	CRT and TFT DSTN	CRT and TFT DSTN	CRT and TFT DSTN	CRT and TFT DSTN	CRT and TFT DSTN	CRT and TFT DSTN	CRT and TFT DSTN			
320 x 240	320 x 240	32	32	32	32	32	32	32	32	32	32	24	24
320 x 240	640 x 480	32	32	32	32	32	32	32	32	8	8	8	8
320 x 240	800 x 600	32	32	32	32	32	32	16	16	8	8	--	--
320 x 240	1024 x 768	32	32	24	24	16	16	8	8	--	--	--	--
320 x 240	1280 x 1024	16	16	16	16	8	8	8	8	--	--	--	--
320 x 240	1600 x 1200	16	16	8	8	8	8	--	--	--	--	--	--
640 x 480	320 x 240	32	32	32	32	32	32	32	24	8	8	8	8
640 x 480	640 x 480	32	32	32	32	24	24	16	16	8	8	--	--
640 x 480	800 x 600	32	32	32	32	24	16	8	8	--	--	--	--
640 x 480	1024 x 768	24	24	16	16	16	16	8	8	--	--	--	--
640 x 480	1280 x 1024	16	16	8	8	8	8	--	--	--	--	--	--
640 x 480	1600 x 1200	8	8	8	8	8	8	--	--	--	--	--	--
800 x 600	320 x 240	32	32	32	32	32	32	16	16	8	8	--	--
800 x 600	640 x 480	32	32	32	24	24	16	8	8	--	--	--	--
800 x 600	800 x 600	32	32	24	24	16	16	8	8	--	--	--	--
800 x 600	1024 x 768	24	24	16	16	8	8	8	8	--	--	--	--
800 x 600	1280 x 1024	16	16	8	8	8	8	--	--	--	--	--	--
800 x 600	1600 x 1200	8	8	8	8	--	--	--	--	--	--	--	--
1024 x 768	320 x 240	32	32	24	24	16	16	8	8	--	--	--	--
1024 x 768	640 x 480	24	24	16	16	16	16	8	8	--	--	--	--
1024 x 768	800 x 600	24	24	16	16	8	8	8	8	--	--	--	--
1024 x 768	1024 x 768	16	16	16	8	8	8	--	--	--	--	--	--
1024 x 768	1280 x 1024	8	8	8	8	8	8	--	--	--	--	--	--
1024 x 768	1600 x 1200	8	8	8	8	--	--	--	--	--	--	--	--
1280 x 1024	320 x 240	16	16	16	16	8	8	8	--	--	--	--	--
1280 x 1024	640 x 480	16	16	8	8	8	8	--	--	--	--	--	--
1280 x 1024	800 x 600	16	16	8	8	8	8	--	--	--	--	--	--
1280 x 1024	1024 x 768	8	8	8	8	8	8	--	--	--	--	--	--
1280 x 1024	1280 x 1024	8	8	8	8	--	--	--	--	--	--	--	--
1280 x 1024	1600 x 1200	8	8	--	--	--	--	--	--	--	--	--	--

Note: -- = mode not supported

C Appendix C



C.1 References

Advanced Configuration and Power Interface Specifications
Intel, Microsoft and Toshiba, Revision 1.0.

Advanced Graphics Port (AGP) Interface Specifications
Intel, Revision 2.0.

AMD K6-III Processor Data Sheet
AMD, Feb. 1999.

Display Power Management Signaling (DPMS) Standard
Video Electronic Standards Association, version 2.0

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

Foley, J.D. and Van Dam, A. - Fundamentals of Interactive Computer Graphics
Addison-Wesley, 1992.

International Radio Consultative Committee, Recommendation ITU-R BT601
(formerly Recommendation CCIR601)

Jack, Keith - Video Demystified
High Text Publications, Inc., 1996.

Kliewer, Bradley Dyck. - EGA/VGA - A Programmer's Reference Guide
McGraw-Hill, second edition, 1990.

PC Card Standard

Personal Computer Memory Card International Association, vol. 2, 1997

PCI Local Bus Specification

PCI Special Interest Group, Revision 2.2.

SAA7111A Enhanced Video Input Processor Product
Specification, Philips Semiconductors, May 15, 1998

VESA BIOS Extensions (VBE) Core Functions
Video Electronic Standards Association, Revision 2.0.

VESA DDC Standard version 1.0
Video Electronic Standards Association, Revision 0.

