www.DataSheet4U.com

Vision868

Multimedia

Accelerator

April 1995

S3 Incorporated
2770 San Tomas Expressway
Santa Clara, CA 95051-0968

www.DataSheet4U.com

DB015-A

www.DataSheet4U.com

NOTATIONAL CONVENTIONS

The following notational conventions are used in this data book:

Signal names are shown in all uppercase letters. For example, XD.

A bar over a signal name indicates an active low signal. For example, $\overline{OE}$.

n-m indicates a bit field from bit n to bit m. For example, 7-0 specifies bits 7 through 0, inclusive.

n:m indicates a signal (pin) range from n to m. For example D[7:0] specifies data lines 7 through 0, inclusive

Use of a trailing letter H indicates a hexadecimal number. For example, 7AH is a hexadecimal number.

Use of a trailing letter b indicates a binary number. For example, 010b is a binary number.

When numerical modifiers such as K or M are used, they refer to binary rather than decimal form. Thus, for example, 1 KByte would be equivalent to 1024, not 1,000 bytes.

NOTICES

© Copyright 1995 S3 Incorporated. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written consent of S3 Incorporated, 2770 San Tomas Expwy., Santa Clara CA 95051-0968. The S3 Corporate Logo, S3 on Board, S3 on Board design, Vision64, Vision864, Vision868, Vision964, Vision968, Trio, Trio32, Trio64, MIC, Galileo, SDAC, Native-MPEG, No Compromise Integration, No Compromise Acceleration and Innovations in Acceleration are trademarks of S3 Incorporated and S3 and True Acceleration are registered trademarks of S3 Incorporated. Other trademarks referenced in this document are owned by their respective companies. The material in this document is for information only and is subject to change without notice. S3 Incorporated reserves the right to make changes in the product design without reservation and without notice to its users.

Additional information may be obtained from:

S3 Incorporated, Literature Department, 2770 San Tomas Expressway, Santa Clara, CA 95051-0968. www.DataSheet4U.com

Telephone: 408-980-5400, Fax: 408-980-5444

www.DataSheet4U.com

Table of Contents

www.DataSheet4U.com

www.DataSheet4U.com

www.DataSheet4U.com

www.DataSheet4U.com

List of Figures

www.DataSheet4U.com

www.DataSheet4U.com

List of Tables

www.DataSheet4U.com

www.DataSheet4U.com

Section 1: Introduction

1.1 OVERVIEW OF THIS DATA BOOKLET

This data booklet largely describes the areas in which the S3® Vision868™ accelerator (hereinafter referred to as the Vision868) is different from the S3 Vision864™ (hereinafter referred to as the Vision864).

Although enhanced mode programming largely remains unchanged from the Vision864 data book description, the entire section is included in this data booklet because it has been extensively revised. Similarly, the entire electrical data section is included. For information on other common features between the Vision868 and Vision864, including most of the registers and the graphics acceleration functionality, see the *Vision864 Graphics Accelerator Data Book*. In particular, note that the pinout and pin descriptions are the same for both chips.

1.2 PRODUCT OVERVIEW

In general, the Vision868 provides all the features found in the Vision864. The exception is that backward compatibility support for CGA, MDA and HGC is no longer provided in hardware.

The Vision868 provides a number of enhanced capabilities not found in the Vision864. These are summarized below and are described in detail in subsequent sections.

1.3 ENHANCED DRAM SUPPORT

The Vision868 supports burst mode DRAMs and single cycle EDO operation at speeds up to 50 MHz. These are capable of sequential read or write data transfers at rates approaching twice that of convention fast page mode DRAMs.

1.4 PACKED PIXEL ACCELERATION

The Vision868 provides packed 24 bits/pixel (true color) acceleration. This means that each pixel in 24 bits/pixel modes now occupies 3 bytes of memory instead of 4 for the Vision864. In addition, the graphics engine can process more pixels in a given number of clock cycles than for the Vision864.

1.5 ENHANCED MEMORY-MAPPED I/O IMPLEMENTATION

The Vision864 provides memory-mapped I/O (MMIO) access to certain enhanced command registers at fixed memory addresses. The Vision868 provides the following enhancements to this scheme:

- The base address for MMIO is now the same as for linear addressing and is relocatable for plug and play support
- The new MMIO is enabled at power-on reset for PCI systems, allowing the PCI subsystem to configure the Vision868 without accessing the VGA registers through the I/O space
- All registers are now memory mapped

www.DataSheet4U.com

www.DataSheet4U.com

- Related 16-bit Enhanced command registers are packed into 32-bit registers to reduce the number of CPU cycles required to access them
- There are two apertures into the MMIO address space. One provides little endian (Intel©-style) access and the other provides big endian (Motorola©/ PowerPC™-style) access

1.6 NEW DRAWING COMMANDS

The following drawing commands have been added to those available with the Vision864:

- polyline/2-point line
- polygon with solid or patterned fills
- trapezoid with solid or patterned fills (Bresenham parameters in hardware)
- trapezoid with solid or patterned fills (Bresenham parameters specified by programmer)
- ROPBLT - Support is provided for the full set of 256 triadic raster operations for BitBLTs (as defined by Microsoft©)

1.7 IMPROVED COPROCESSOR SUPPORT

During shared frame buffer operation, the coprocessor can now be guaranteed a specified amount of the memory bandwidth without being forced to return control of the memory bus.

1.8 PCI SUPPORT ENHANCEMENTS

The following have been added to improve operation in PCI systems:

- New registers for interrupt handling
- Complete RAMDAC snooping capability
- PCI disconnect based on the command FIFO status

1.9 VIDEO ENGINE

The Video Engine integrated into the Vision868 can convert a YUV data stream to an RGB data stream. The output of the color space converter is then sent through the scaling engine for scaling up or down. Additionally, a dithering engine converts 24 bits/pixel images to a 16- or 8- bit format (and 16-bit images to an 8-bit format) with little quality impact. To support image compression, the Video Engine can scale images down to the size required to meet disk drive bandwidth requirements.

1.10 RESOLUTIONS SUPPORTED

Supported resolutions are shown in the following table. Extended VGA text modes up to 132 columns by 43 rows are possible as well.

Table 1-1. Video Resolutions Supported

Resolution	1 MB DRAM	2 MBs DRAM	4 MBs DRAM
640x480x4	✔	✔	✔
640x480x8	✔	✔	✔
640x480x16	✔	✔	✔
640x480x24	✔	✔	✔
640x480x32		✔	✔
800x600x4	✔	✔	✔
800x600x8	✔	✔	✔
800x600x16	✔	✔	✔
800x600x24/32		✔	✔
1024x768x4	✔	✔	✔
1024x768x8	✔	✔	✔
1024x768x16		✔	✔
1024x768x24			✔
1152x864x8	✔	✔	✔
1280x1024x4	✔	✔	✔
1280x1024x8		✔	✔
1600x1200x4	✔	✔	✔
1600x1200x8		✔	✔

www.DataSheet4U.com

www.DataSheet4U.com

Section 2: Register Changes

2.1 Vision868 REGISTER CHANGES

Most Vision868 registers operate exactly as their Vision864 counterparts. The exceptions are shown in Table 2-1. New Vision868 registers are described in Sections 3.2 through 3.6.

Table 2-1. Register Differences Between the Vision868 and Vision864

Vision864 Register	Bit(s)	Vision868 Description
CR30	7-0	This is hardwired to E1H
CR32	3-2	Reserved
CR36	3-2	00 = One cycle EDO DRAM (new) 01 = Burst mode DRAM (new) 10 = EDO DRAM 11 = Fast page mode DRAM
CR40	3	0 = Normal SRDY operation 1 = SRDY delayed by one cycle (VL-Bus only)
CR50	1	0 = DACRD, DACWR active pulse = 2 SCLKs 1 = DACRD, DACWR active pulse = 4 SCLKs
CR50	5-4	00 = 4/8 bits/pixel 01 = 16 bits/pixel 10 = 24 bits/pixel (new) 11 = 32 bits/pixel
CR53	0	Reserved (write per bit no longer supported)
CR53		The power-on default depends on the system bus type. See the description for bits 4-3 below
CR53	2-1	Big endian data byte swap (linear addressing and Video Engine transfers only) 00 = No swap 01 = Swap bytes within each word 10 = Swap all bytes in doubleword (bytes reversed) 11 = Reserved

www.DataSheet4U.com

www.DataSheet4U.com

Table 2-1. Register Differences Between the Vision864 and Vision868 (Continued)

Vision864 Register	Bit(s)	Vision868 Description
CR53	4-3	MMIO select 00 = MMIO disabled 01 = MMIO disabled 10 = Vision864-type MMIO (A0000H window) 11 = New MMIO (relocatable/packed) and Vision864-type MMIO VL-Bus power-on default = 00 PCI bus power-on default = 11
CR53	7	Enable upper byte write protection in 32 bits/pixel mode 0 = CAS3 and CAS7 enabled during 32 bits/pixel memory writes 1 = CAS3 and CAS7 disabled during 32/bits/pixel memory writes
CR54	1-0	Big endian data byte swap (not linear addressing, not image writes and not Video Engine access) 00 = No swap 01 = Swap bytes within each word 10 = Swap all bytes in doubleword (bytes reversed) 11 = Swap according to BE[3:0] (VL-Bus) or C/BE(3:0] (PCI bus) BE[3:0]/C/BE[3:0] 0000 = Swap all bytes in doubleword (bytes reversed) 0011 = Swap bytes within selected word 1100 = Swap bytes within selected word All other values = No swap
CR54	2	This is the high order extension bit (bit 5) for the M parameter (bits 7-3 of this register). This doubles the amount of memory bandwidth that can be allocated for Graphics Engine/CPU access.
CR58		The power-on default depends on the system bus type. See the description for bit 4 below.
CR58	4	VL-Bus power-on default = 0 PCI bus power-on default = 1
CR59, CR5A	15-0	VL-Bus power-on default = 000AH PCI bus power-on default = 7000H
CR61	6-5	Big endian data byte swap (image writes) 00 = No swap 01 = Swap bytes within each word 10 = Swap all bytes in doubleword (bytes reversed) 11 = Reserved
CR66	3	PCI disconnect enable (PCI bus only) 0 = No effect 1 = An attempt to write data with the Command FIFO full or to read data with the Command FIFO not empty generates a PCI bus disconnect cycle Bit 7 of this register must also be set to 1 to enable this feature.
CR67	7-4	0111 = Mode 11: 32-bit color, 2 VCLKs/pixel 1001 = Mode 12. 24-bit packed color, 3 VCLKs/2 pixels

www.DataSheet4U.com

www.DataSheet4U.com

Table 2-1. Register Differences Between the Vision864 and Vision868 (Continued)

Vision864 Register	Bit(s)	Vision868 Description
Enhanced Registers		
82E8H	11-0	For polygons or trapezoids, this is the starting pixel vertical position for the first of two edges to be drawn.
86E8H	11-0	For polygons or trapezoids, this is the starting pixel horizontal position for the first of two edges to be drawn.
8AE8H	11-0	For polylines, this is the ending vertical position for each line segment. For polygons and 4-point trapezoids, this is the ending vertical position for the first of two edges to be drawn For Bresenham parameter trapezoids, this is the axial step constant for the first of two edges to be drawn.
8EE8H	11-0	For polylines, this is the ending horizontal position for each line segment. For polygons and 4-point trapezoids, this is the ending horizontal position for the first of two edges to be drawn For Bresenham parameter trapezoids, this is the diagonal step constant for the first of two edges to be drawn
92E8H	11-0	For Bresenham parameter trapezoids, this is the error term for the first of two edges to be drawn.
96E8H	11-0	For Bresenham parameter trapezoids, this is the major axis length for the first of two edges to be drawn
9AE8H	10-9	Select image write (E2E8H, E2EAH) bus transfer width 00 = 8 bits 01 = 16 bits 10 = 32 bits. All doubleword bits beyond the image rectangle width are discarded Each line starts with a fresh doubleword The current drawing position ends up one pixel below the lower left hand corner of the image rectangle 11 = 32 bits. This setting applies only to image transfers across the plane Only padding bits up to the start of the next byte are discarded. The current drawing position ends up one pixel to the right of the top right corner of the rightmost image rectangle
9AE8H	11, 15-13	Drawing Command 0000 = NOP 0001 = Draw line 0010 = Rectangle fill 0011 = Polygon fill solid (new) 0100 = 4-point trapezoid fill solid (new) 0101 = Bresenham parameter trapezoid fill solid (new) 0110 = BitBLT 0111 = Pattern BLT 1001 = Polyline/2-point line (new) 1011 = Polygon fill pattern (new) 1100 = 4-point trapezoid fill pattern (new) 1101 = Bresenham parameter trapezoid fill pattern (new) 1110 = ROPBLT (new) See the Enhanced Mode Programming section for a description of each of these commands.

www.DataSheet4U.com

www.DataSheet4U.com

Table 2-1. Register Differences Between the Vision864 and Vision868 (Continued)

Vision864 Register	Bit(s)	Vision868 Description
AEE8H	31-0	Read mask for the ROPBLT pattern (same as previous Read Mask)
BEE8H, Index E	6	Reserved
BEE8H, Index E	10	0 = Allow burst mode/1 cycle EDO write operation 1 = Disable burst mode/1 cycle EDO write operation
PCI Registers		
PCI Index 02H	15-0	The device ID is hardwired to 8880H.
PCI Index 04H	7	Hardwired to 1 to indicate that the device always does address/data stepping.
PCI Index 0AH	7-0	Now included in new Class Code register.

www.DataSheet4U.com

www.DataSheet4U.com

2.2 NEW Vision868 SR REGISTERS

The SR registers described in this section have been added for the Vision868 and are not found in the Vision864.

Extended Sequencer Register 9 (SR9)

Read/Write
Power-On Default: 00H

Address: 3C5H, Index 09H

7	6	5	4	3	2	1	0
MMIO-ONLY	R	R	R	R	R	R	R

Bits 6-0 Reserved

Bit 7 MMIO-ONLY - Memory-mapped I/O register access only
0 = If MMIO is enabled, both programmed I/O and memory-mapped I/O register accesses are allowed
1 = If MMIO is enabled, only memory-mapped I/O register accesses are allowed

External Bus Request Control Register (SRA)

Read/Write
Power-On Default: 00H

Address: 3C5H, Index 0AH

7	6	5	4	3	2	1	0
2 MCLK	R	P VALUE 5	4	3	2	1	0

Bits 5-0 P VALUE
The integer equivalent of the binary value in this field is the number of MCLK units less one a secondary memory controller is allowed to retain control of the memory bus before the Vision864 drops its bus grant. See Section 3.2 for a detailed explanation.

Bit 6 Reserved

Bit 7 2MCLK - 2 MCLK CPU writes to memory (linear addressing)
0 = 3 MCLK memory writes
1 = 2 MCLK memory writes

www.DataSheet4U.com

www.DataSheet4U.com

2.3 NEW Vision868 CR REGISTERS

The CR registers described in this section have been added for the Vision868 and are not found in the Vision864.

Device ID High Register (CR2D)

Read Only
Power-On Default: 88H

Address: 3?5H, Index 2DH

This register contains the same value as the upper byte of the PCI Vendor ID (Index 00H) register.

7	6	5	4	3	2	1	0
CHIP ID HIGH							

Bits 7–0 CHIP ID HIGH

Device ID Low Register (CR2E)

Read Only
Power-On Default: 90H

Address: 3?5H, Index 2EH

7	6	5	4	3	2	1	0
CHIP ID LOW							

Bits 7–0 CHIP ID LOW

Revision Register (CR2F)

Read Only
Power-On Default: 00H

Address: 3?5H, Index 2FH

7	6	5	4	3	2	1	0
REVISION LEVEL							

Bits 7–0 REVISION LEVEL

www.DataSheet4U.com

www.DataSheet4U.com

2.4 NEW Vision868 ENHANCED COMMANDS REGISTERS

The Enhanced Commands registers described in this section have been added for the Vision868 and are not found in the Vision864.

Current Y-Position 2 Register (CUR_Y2)

Read/Write Address: 82EAH
Power-On Default: Undefined

For polygons or trapezoids, this is the starting pixel vertical position for the second of two edges to be drawn. Reading this register produces the current vertical drawing coordinate for the second edge being drawn.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	R	R	R	CURRENT Y-POSITION 2											

Bits 11–0 CURRENT Y-POSITION 2

Bits 15–12 Reserved

Current X-Position 2 Register (CUR_X2)

Read/Write Address: 86EAH
Power-On Default: Undefined

For polygons or trapezoids, this is the starting pixel horizontal position for the second of two edges to be drawn. Reading this register produces the current horizontal drawing coordinate for the second edge being drawn.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	R	R	R	CURRENT X-POSITION 2											

Bits 11–0 CURRENT X-POSITION 2

Bits 15–12 Reserved

www.DataSheet4U.com

www.DataSheet4U.com

Y-Coordinate 2/Axial Step Constant 2 Register (Y2_AXSTP2)

Read/Write Address: 8AEAH
Power-On Default: Undefined

For polygons and 4-point trapezoids, this is the ending vertical position for the second of two edges to be drawn. For Bresenham parameter trapezoids, this is the axial step constant for the second of two edges to be drawn.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	R	R	R	Y-COORDINATE 2											

Bits 11–0 Y-COORDINATE 2

Bits 15–12 Reserved

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	R	LINE PARAMETER AXIAL STEP CONSTANT 2													

Axial Step Constant = 2 * (min(|dx|,|dy|)) In other words, when drawing a line from point A to point B, determine the change in the X coordinate from A to B and the change in the Y coordinate from A to B. Take the smaller of the two changes and multiply its absolute value by 2.

Bits 13–0 LINE PARAMETER AXIAL STEP CONSTANT 2

Bits 15–14 Reserved

X-Coordinate 2 Register (X2)

Read/Write Address: 8EEAH
Power-On Default: Undefined

For polygons and 4-point trapezoids, this is the ending horizontal position for the second of two edges to be drawn. For Bresenham parameter trapezoids, this is the diagonal step constant for the second of two edges to be drawn.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	R	R	R	X-COORDINATE 2											

Bits 11–0 X-COORDINATE 2

Bits 15–12 Reserved

www.DataSheet4U.com

www.DataSheet4U.com

Line Error Term 2 Register (ERR_TERM2)

Read/Write Address: 92EAH
Power-On Default: Undefined

For Bresenham parameter trapezoids, this is the error term for the second of two edges to be drawn.

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

Error Term = 2 * min(|dx|,|dy|) - max(|dx|,|dy| - 1 if the starting X < the ending X
Error Term = 2 * min(|dx|,|dy|) - max(|dx|,|dy| if the starting X ≥ the ending X

See the Destination Y-Position/Axial Step Constant (8AE8H) register for an explanation of the terms used in these equations.

Bits 13–0 ERROR TERM 2

Bits 15–14 Reserved

Major Axis Pixel Count 2 Register (MAJ_AXIS_PCNT2)

Read/Write Address: 96EAH
Power-On Default: Undefined

This register specifies the length (in pixels) of the major (longest) axis for solid and textured lines and the width for rectangles, image transfers, BitBLTs and PatBLTs. For Bresenham parameter trapezoids, this is the major axis length for the first of two edges to be drawn.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	R	R	R	RECTANGLE WIDTH 2/LINE PARAMETER MAX 2											

Bits 11–0 RECTANGLE WIDTH 2/LINE PARAMETER MAX 2
The value is the number of pixels along the major axis - 1.

Bits 15–12 Reserved

www.DataSheet4U.com

www.DataSheet4U.com

Drawing Command 2 Register (CMD2)

Write Only Address: 9AEAH
Power-On Default: Undefined

For Bresenham parameter trapezoid fills, this register defines the drawing direction for the second edge to be drawn. The drawing direction for the first edge is specified in bits 7-5 of 9AE8H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								DRWG-DIR 2.							
R	R	R	R	R	R	R	R	2	1	0	R	R	R	R	R

Bits 4-0 Reserved

Bits 7–5 DRWG-DIR 2 - Select Drawing Direction 2
In the following table, the line is drawn from left to right or a +X and from right to left for a -X, down for a +Y and up for a -Y. X or Y maj specifies the longest axis. Axial drawing is specified in bit 3 of 9AE8H.

7–5	x-y (Axial - bit 3 = 0)
000	–Y,X maj,–X
001	–Y,X maj,+X
010	–Y,Y maj,–X
011	–Y,Y maj,+X
100	+Y,X maj,–X
101	+Y,X maj,+X
110	+Y,Y maj,–X
111	+Y,Y maj,+X

Bits 15-8 Reserved

www.DataSheet4U.com

www.DataSheet4U.com

ROP Mix Selection Register (ROPMIX)

Read/Write Address: D2E8H
Power-On Default: Undefined

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	R	R	R	R	R	R	MP	ROP MIX SELECTION							

Bits 7–0 ROP MIX SELECTION

See Appendix A for a listing of the 256 mix selections available.

Bit 8 MP - Mono pattern select
0 = ROPBLT pattern is color
1 = ROPBLT pattern is monochrome

If a monochrome pattern is selected, the background and foreground colors are specified in E6E8H and EEE8H respectively.

Bits 15–9 Reserved

ROPBLT Pattern Background Color Register (PAT_BG_COLOR)

Read/Write Address: E6E8H
Power-On Default: Undefined

This register defines the background color for the ROPBLT pattern when bit 8 of D2E8H is set to 1 to specify a monochrome pattern.

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

Bits 31–0 BACKGROUND COLOR
If bit 9 of BEE8_EH is set to 1 or if MMIO is enabled, this becomes a 32-bit register. If bit 9 of BEE8_EH is cleared to 0, this is two 16-bit registers. In 32 bpp mode with 16-bit registers, the upper and lower doublewords are read or written sequentially, depending on the state of the RSF flag (bit 4 of BEE8H, Index EH). If RSF = 0, the lower 16 bits are accessed. If RSF = 1, the upper 16 bits are accessed. The RSF flag toggles automatically when a doubleword is read or written.

www.DataSheet4U.com

www.DataSheet4U.com

Pattern Y (PAT_Y)

Read/Write Address: EAE8H
Power-On Default: Undefined

This register defines the vertical coordinate top edge of an 8x8 pixel pattern programmed into off-screen memory. This is used with the Bresenham parameter trapezoidal fill pattern, trapezoidal 4-point trapezoid fill pattern, polygon fill pattern and ROPBLT command types.

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

Bits 11–0 PATTERN Y

Bits 15–12 Reserved

Pattern X (PAT_X)

Read/Write Address: EAEAH
Power-On Default: Undefined

This register defines the horizontal coordinate of the left side of an 8x8 pixel pattern programmed into off-screen memory. This is used with the Bresenham parameter trapezoidal fill pattern, trapezoidal 4-point trapezoid fill pattern, polygon fill pattern and ROPBLT command types.

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

Bits 11–0 PATTERN X

Bits 15–12 Reserved

www.DataSheet4U.com

www.DataSheet4U.com

ROPBLT Pattern Foreground Color Register (PAT_FG_COLOR)

Read/Write Address: EEE8H
Power-On Default: Undefined

This register defines the foreground color for the ROPBLT pattern when bit 8 of D2E8H is set to 1 to specify a monochrome pattern.

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

Bits 31–0 FOREGROUND COLOR
If bit 9 of BEE8_EH is set to 1 or if MMIO is enabled, this becomes a 32-bit register. If bit 9 of BEE8_EH is cleared to 0, this is two 16-bit registers. In 32 bpp mode with 16-bit registers, the upper and lower doublewords are read or written sequentially, depending on the state of the RSF flag (bit 4 of BEE8H, Index EH). If RSF = 0, the lower 16 bits are accessed. If RSF = 1, the upper 16 bits are accessed. The RSF flag toggles automatically when a doubleword is read or written.

www.DataSheet4U.com

www.DataSheet4U.com

2.5 NEW Vision868 PCI CONFIGURATION SPACE REGISTERS

The PCI Configuration Space registers described in this section have been added for the Vision868 and are not found in the Vision864.

Class Code

Read Only Address: 08H
Power-On Default: 30000H

The class code is contained in bits 31-8. These 3 bytes are hardwired to 30000H to specify that the Vision868 is a VGA-compatible display controller. Bits 7-0 contain the Revision ID, and are present in the Vision864.

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

Interrupt Line

Read/Write Address: 3CH
Power-On Default: 00H

This register contains interrupt line routing information written by the POST program during power-on initialization.

7	6	5	4	3	2	1	0
INTERRUPT LINE							

Bits 7–0 INTERRUPT LINE

Interrupt Pin

Read Only Address: 3DH
Power-On Default: 01H

This register is hardwired to a value of 1 to specify that INTA is the interrupt pin used.

7	6	5	4	3	2	1	0
INTERRUPT PIN							

www.DataSheet4U.com

Bits 7–0 INTERRUPT PIN

www.DataSheet4U.com

2.6 NEW Vision868 VIDEO ENGINE REGISTERS

The Video Engine registers described in this section have been added for the Vision868. They are not found in the Vision864.

Video Engine NOP Register

Read/Write Address: 8080H
Power-On Default: 0000 0000H

Software must make a dummy write to this register when it is finished sending Video Engine commands. Graphics Engine commands can then follow. This NOP prevents simultaneous Video Engine and Graphics Engine operation.

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

Bits 31–0 NOP - No operation

Video Engine Control Register

Read/Write Address: 8088H
Power-On Default: 0000 0000H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	R	R	R	DDA ACCUMULATOR INITIAL VALUE											

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FILT	CSC	DITH	OTT	R	KM	DM INDEX			INPUT DATA FORMAT			CS	OUTPUT DATA FORMAT		

Bits 11–0 DDA ACCUMULATOR INITIAL VALUE
Digital Differential Analyzer accumulator initial value = -MAX(W_s, Wd), where W_s is the width in pixels of the source and W_d is the width in pixels of the destination.

Bits 15–12 Reserved

www.DataSheet4U.com

www.DataSheet4U.com

Bits 18–16 OUTPUT DATA FORMAT
000 = 8 bits/pixel, RGB, 3.3.2 -can be translated with output table
001 = Reserved
010 = Reserved
011 = 32 bits/pixel, RGB, x.8.8.8
100 = 16 bits/pixel, YCbCr, 4:2:2
101 = 16 bits/pixel, raw
110 = 15 bits/pixel RGB, 5.5.5
111 = 16 bits/pixel RGB, 5.6.5

Bit 19 CS - Color Space
0 = YCbCr, 16-240 range
1 = YUV

Bits 22–20 INPUT DATA FORMAT
000 = 8 bits/pixel, RGB, 3.3.2 -can be translated with output table
001 = Reserved
010 = Reserved
011 = 32 bits/pixel, RGB, x.8.8.8
100 = 16 bits/pixel, YCbCr, 4:2:2
101 = 16 bits/pixel, raw (scaling only)
110 = 15 bits/pixel RGB, 5.5.5
111 = 16 bits/pixel RGB, 5.6.5

Bits 25–23 DM INDEX - Dithering Matrix Index
This is auto-incremented at the end of each line. It should be initialized to 0 at the beginning of each new frame or block of lines.

Bit 26 KM - Key Mask Enable
0 = Disable - bypass function
1 = Enable

Bit 27 Reserved

Bit 28 OTT - Output Table Translation Enable
0 = Disable - bypass function
1 = Enable

Bit 29 DITH - Dithering
0 = Disable - bypass function
1 = Enable

Bit 30 CSC - Color Space Converter
0 = Disable - bypass function
1 = Enable

Bit 31 FILT - Filter
0 = Disable - bypass function
1 = Enable

www.DataSheet4U.com

www.DataSheet4U.com

Video Engine Stretch/Filter Constants Register

Read/Write Address: 808CH
Power-On Default: 0000 0000H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FILTER		R	R	R	K2										

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SS	HD	SEN	R	R	K1										

Bits 10–0 K2
If $W_s \geq W_d$, then K2 = W_d-W_s; If $W_s < W_d$, then K2 = W_s-W_d

W_s is the width in pixels of the source and W_d is the width in pixels of the destination.

Bits 13–11 Reserved

Bits 15–14 FILTER - Filter characteristics
00 = Bi-linear (default)
01 = Linear, 0-2-4-2-0
10 = Linear, 1-2-2-2-1
11 = Reserved

Bits 26–16 K1
If $W_s \geq W_d$, then K1 = W_d; If $W_s W_d$, then K1 = W_s

W_s is the width in pixels of the source and W_d is the width in pixels of the destination.

Bits 28-27 Reserved

Bit 29 SEN - Sense
0 = Normal. Logic 1 in key mask enables video, logic 0 in key mask enables graphics
1 = Reverse. Logic 0 in key mask enables video, logic 1 in key mask enables graphics

Bit 30 HD - Host data
0 = Data from screen
1 = Data from host

Bit 31 SS - Shrink/Stretch, Scale Down/Up
0 = Stretch/Scale Up
1 = Shrink/Scale Down

www.DataSheet4U.com

www.DataSheet4U.com

Video Engine Source/Destination Step Register

Read/Write Address: 8090H
Power-On Default: 0000 0000H

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	R	R	SOURCE STEP												

Bits 12–0 DESTINATION STEP

This is the pitch that is added to the write address following a memory write cycle. For a horizontal stretch, this register should be programmed with an 8 for 2-MByte memory configurations or larger (8 bytes/write) or a 4 for 1-MByte memory configurations (4 bytes/write). For a vertical stretch, the step is the number of bytes/pixel times the number of pixels/line (bytes/line.

Bits 15–13 Reserved

Bits 28–16 SOURCE STEP

This is the pitch that is added to the read address following a memory read cycle. For a horizontal stretch, this register should be programmed with an 8 for 2-MByte memory configurations or larger (8 bytes/write) or a 4 for 1-MByte memory configurations (4 bytes/write). For a vertical stretch, the step is the number of bytes/pixel times the number of pixels/line (bytes/line).

Bits 31–29 Reserved

www.DataSheet4U.com

www.DataSheet4U.com

Video Engine Crop Register

Read/Write Address: 8094H
Power-On Default: 0000 0000H

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	R	R	R	START											

Bits 11–0 LENGTH

Value = Actual number of pixels to process

Bits 15–13 Reserved

Bits 27–16 START

Value = Number of pixels to not be written after starting

Bits 15–13 Reserved

Video Engine Source Base Address Register

Read/Write Address: 8098H
Power-On Default: 0000 0000H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SOURCE BASE ADDRESS [15:0]															

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	R	R	R	R	R	R	R	SOURCE BASE ADDRESS [23:16]							

Bits 23–0 SOURCE BASE ADDRESS
Source base address (in bytes) in display memory for the image to be processed by the Video Engine. The source address is in display memory only for the second pass of a 2-pass operation. Otherwise, the source is system memory with the data provided by the Host. In this case, only bits 1-0 of this register are relevant. The decimal value of bits 1-0 is the number of bytes to discard from the start of each pixel line stream from the Host. This allows adjustment for non-doubleword-aligned pixel data in system memory.

www.DataSheet4U.com

www.DataSheet4U.com

Bits 31–24 Reserved

Video Engine Destination Base Address Register

Read/Write Address: 809CH
Power-On Default: 0000 0000H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DESTINATION BASE ADDRESS [15:0]															

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
VEB								DESTINATION BASE ADDRESS [23:16]							

Bits 23–0 DESTINATION BASE ADDRESS [23:0]
Destination base address (in bytes) in display memory for the image to be processed by the Video Engine.

Bits 30–24 Reserved

Bit 31 VEB - Video Engine busy (Read Only)
0 = Video Engine not busy
1 = Video Engine busy

2.7 Vision864 REGISTERS REMOVED FROM THE Vision868

All backward compatibility registers found in the Vision864 are removed from the Vision868. These include all registers described in Sections 12 and 13 of the *Vision864 Graphics Accelerator Data Book*.

www.DataSheet4U.com

www.DataSheet4U.com

Section 3: Functional Changes

This section describes a number of enhancements provided by the Vision868 as compared with the Vision864. The new features related to Enhanced mode programming support are described in Section 4. The new Video Engine capability for the Vision868 is described in Section 5.

3.1 CHIP IDENTIFICATION

Three new chip identification registers are provided. CR2D is hardwired with 88H. This is the upper word of the device I.D.. CR2E is hardwired with 90H. This is the low byte of the device I.D. CR2F contains the revision level. CR30, the old I.D. register, contains E1H.

3.2 SHARED FRAME BUFFER MEMORY BANDWIDTH ALLOCATION

The Vision868 provides the same bus request/bus grant shared frame buffer capability as the Vision864. However, several related new capabilities are added to enhance usability.

Since the Vision868 is responsible for screen and DRAM refreshing, it must ensure that the secondary controller does not retain control of the memory bus for too long. It uses the P parameter to provide this control. The 6-bit P parameter field (bits 5-0 of SRA) is programmed in units of MCLKs. The value programmed specifies one less than the number of MCLKs that the secondary controller can retain control of the memory bus before the Vision868 removes the bus grant. For example, a programmed value of 5 specifies that the Vision868 will remove the bus grant after 6 MCLKs. At maximum, the bus can be granted for 64 MCLKs.

How long the secondary controller can retain control without causing problems is a function of many variables, including the M and N parameter settings, the pixel data bandwidth of the RAMDAC, the needs of the CPU and Graphics Engine to access display memory and the timing of the bus request (during screen active or inactive). In general, the secondary controller should retain control for a period less than the time that would normally be allocated to the M parameter, and the time allocated to the P parameter should be subtracted from the time allocated to the M parameter. For pure live video display without graphics updating, the M parameter can be set to 0 (allowing 1 memory cycle) and the rest of the non-FIFO fill bandwidth given to the P parameter. See Section 7.4, Display Memory Access Control, of the Vision864 data book for more information on the M and N parameters.

For the Vision868, the M parameter field is 6 bits instead of 5 bits for the Vision864. This doubles the amount of memory bandwidth that can be allocated to Graphics Engine/CPU accesses.

3.3 PCI ENHANCEMENTS

The Vision868 contains several enhancements for PCI bus operation.

3.3.1 Interrupt Support

PCI configuration space registers Index 3CH (Interrupt Line) and Index 3DH (Interrupt Pin) are added to support handling of interrupts.

www.DataSheet4U.com

www.DataSheet4U.com

When in PCI mode, the $\overline{\text{INTA}}$ pin floats. Interrupt generation drives it active low. When the interrupt is removed, the $\overline{\text{INTA}}$ pin again floats.

When in VL-Bus mode, the SINTR pin is driven high. Interrupt generation drives it low and then releases it to be driven high again (edge triggered).

3.3.2 RAMDAC Snooping

For PCI bus configurations, setting bit 5 of the PCI Command register (Index 04H) to 1 causes the Vision868 to snoop for RAMDAC writes. This means that the Vision868 will write the data to the local RAMDAC but will not assert $\overline{\text{DEVSEL}}$. The ISA controller can then also generate a write cycle to a secondary RAMDAC. The Vision868 always provides the data for RAMDAC reads.

If snooping is enabled, the 3C8 and 3C9 DAC registers must be written separately with byte writes (no word writes).

If bit 5 of the PCI Command register is cleared to 0, the Vision868 claims all RAMDAC read and write cycles.

3.3.3 Disconnect Related to FIFO Status

If bits 3 and 6 of CR66 are set to 1, pixel data writes made with the command FIFO full or memory reads made with the Command FIFO not empty will generate a PCI disconnect.

3.3.4 Plug and Play Support

When the Vision868 powers up in PCI mode, it defaults to linear addressing and memory-mapped I/O enabled at a relocatable base address of 7000 0000H. This allows the PCI system to reconfigure as required for plug and play.

3.4 PACKED 24-BITS/PIXEL SUPPORT

In 32 bits/pixel modes, the Graphics Engine can process two pixels simultaneously (64 bits). The Vision868 also provides accelerated packed 24 bits/pixel operation. This means that 2 2/3 pixels can be processed simultaneously, resulting in an approximately 25% performance improvement. Packed 24 bits/pixel operation is selected by setting bits 5-4 of CR50 to 10b.

Most of the hardware acceleration features available for 32 bits/pixel operation are also available for packed 24 bits/pixel operation. Those functions for which hardware acceleration is not provided for packed 24 bits/pixel operation are:

- Textured line draw
- BitBLT across the plane
- Color comparison
- Video Engine operation (Vision868)

Image transfers (CPU as data source) can be performed in packed 24 bits/pixel mode.

When using packed 24 bits/pixel operation, all drawing coordinates (e.g., current X,Y or destination X,Y) must be positive numbers.

3.5 RAMDAC ACCESS CYCLES

The RAMDAC strobes $\overline{\text{DACRD}}$ and $\overline{\text{DACWR}}$ are based on MCLK for the Vision864. They are based on SCLK for the Vision868. The PCI bus RAMDAC access functional timing for the Vision868 is shown in Figures 3-1 and 3-2. VL-Bus cycles have the same timing characteristics.

The active pulse of $\overline{\text{DACRD}}$ or $\overline{\text{DACWR}}$ is 2 SCLKs if bit 1 of CR50 is cleared to 0 and 4 SCLKs if this bit is set to 1. The latter setting is required for 50 MHz VL-Bus operation to ensure a pulse width greater than 50 ns. Selecting a 4 SCLK active pulse width extends the total cycle by 10 SCLKs.

3.6 50 MHz VL-BUS SUPPORT

Setting bit 3 of CR40 to 1 adds a wait state to VL-Bus cycles by delaying SRDY by one clock. This is required for 50 MHz VL-Bus operation. In addition, bit 1 of CR50 must be set to 1 to extend the RAMDAC read and write strobe pulses.

www.DataSheet4U.com

www.DataSheet4U.com

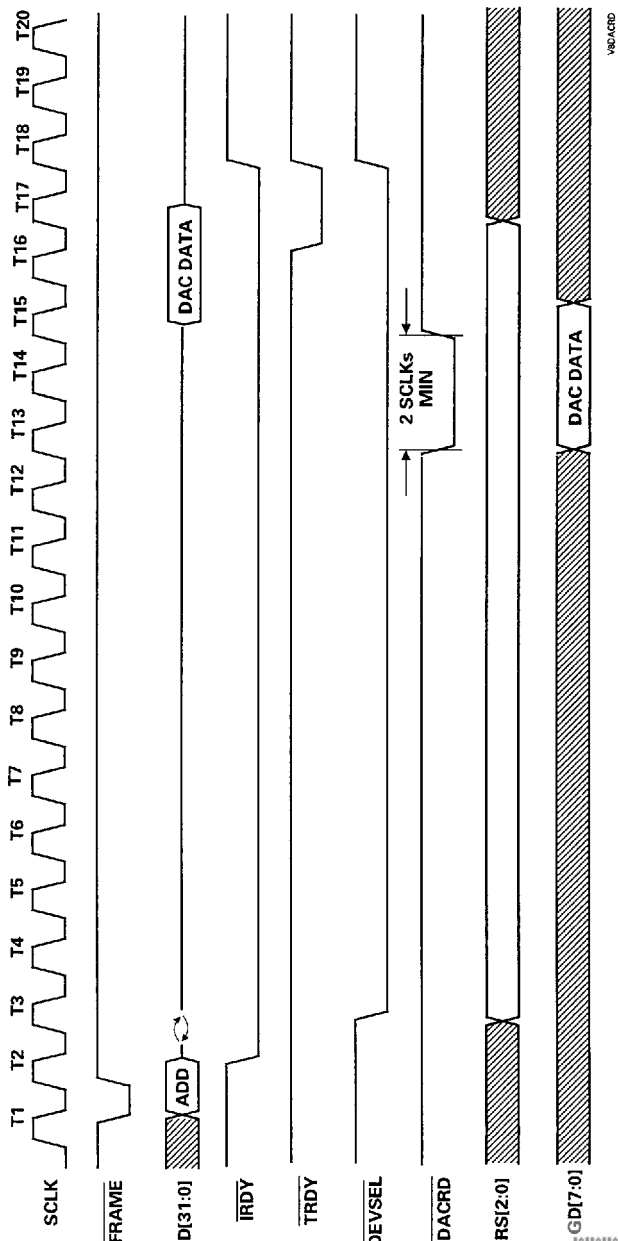


Figure 3-1. RAMDAC PCI Read Timing

www.DataSheet4U.com

www.DataSheet4U.com

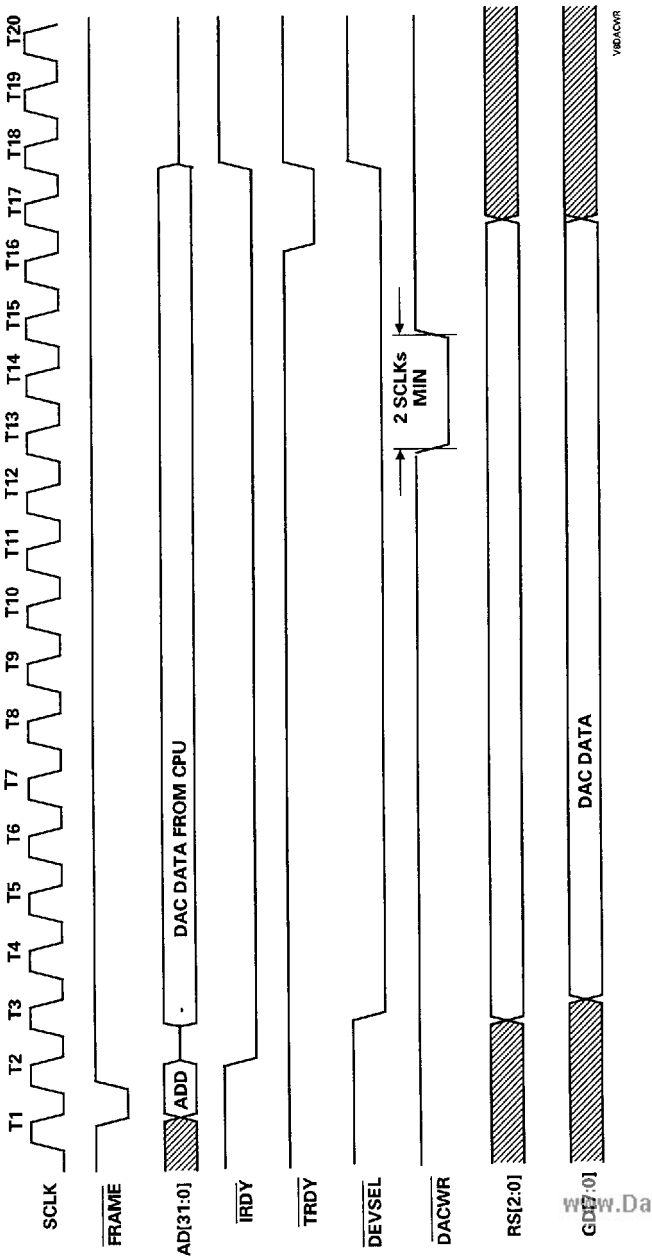


Figure 3-2. RAMDAC PCI Write Timing

www.DataSheet4U.com

www.DataSheet4U.com

3.7 WRITE PER BIT SUPPORT

The Vision868 does not support write per bit and bit 0 of CR53 is now reserved.

3.8 READ/MODIFY/WRITE TIMING

The read/modify/write cycles for fast page and EDO memories have been modified from the Vision864. These are shown in Figures 3-3 and 3-4.

For fast page mode operation, read data is latched by the rising edge of $\overline{CAS}$. For 2 cycle EDO operation, the read data is latched by the rising edge of MCLK following a $\overline{CAS}$ rising edge.

Note that both Figure 3-3 and 3-4 are based on a $\overline{RAS}$ precharge of 2.5 MCLKs.

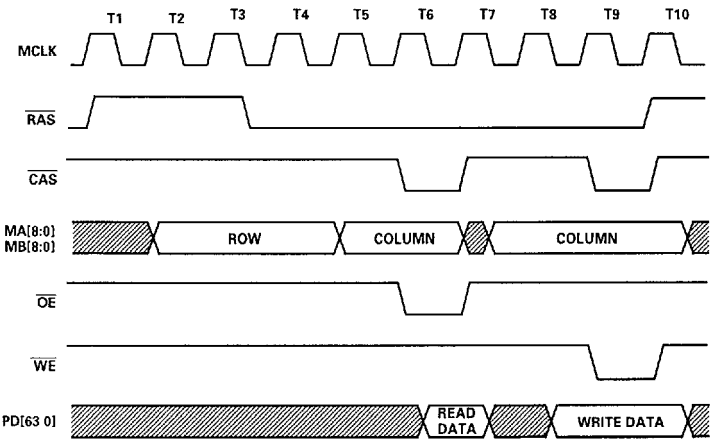
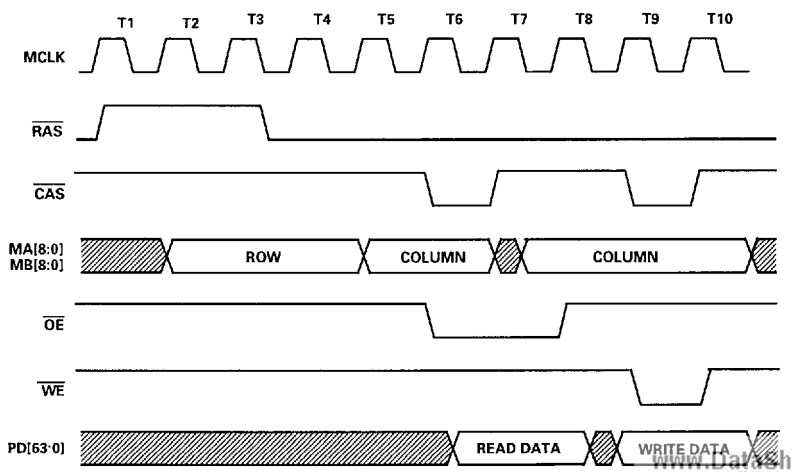


Figure 3-3. Fast Page Mode Read/Modify Write Timing



www.DataSheet4U.com

Figure 3-4. EDO Mode Read/Modify Write Timing

www.DataSheet4U.com

3.9 1 CYCLE EDO DRAM SUPPORT

Bits 3-2 of CR36 are cleared to 00b to indicate that 1 cycle EDO DRAM operation is being used. Note that $\overline{CAS}$/$\overline{OE}$/$\overline{WE}$ stretch capability provided by bits 1-0 of CR68 cannot be used with 1 cycle EDO operation. This mode of operation uses standard EDO DRAMs with more aggressive timing.

The functional timing for 1 cycle EDO reads is provided by Figure 3-5. The DRAM drives valid read data after the first $\overline{CAS}$ falling edge. The chip latches the data on the second falling $\overline{CAS}$ edge. This means that a dummy cycle is required at the end to latch the last read. At the end of the cycle, the memory controller remains in page mode and checks for another memory request. If one is pending, it performs the 1.5 MCLK $\overline{RAS}$ precharge as shown in Figure 3-5. If no request is pending, the $\overline{RAS}$ precharge may be delayed from that shown in the figure and will be 2.5 MCLKs when it occurs.

The functional timing for 1 cycle EDO writes is provided by Figure 3-6. Write data is latched by the DRAM on the falling edge of $\overline{CAS}$. No dummy cycle is required. At the end of the cycle, the

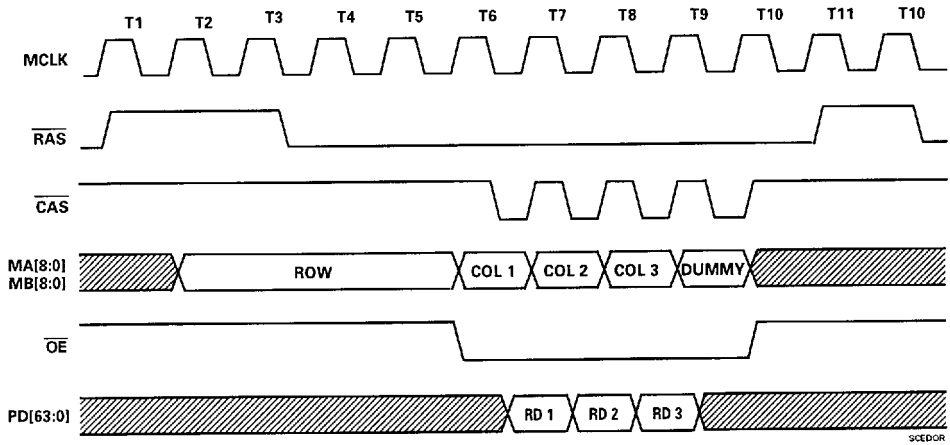
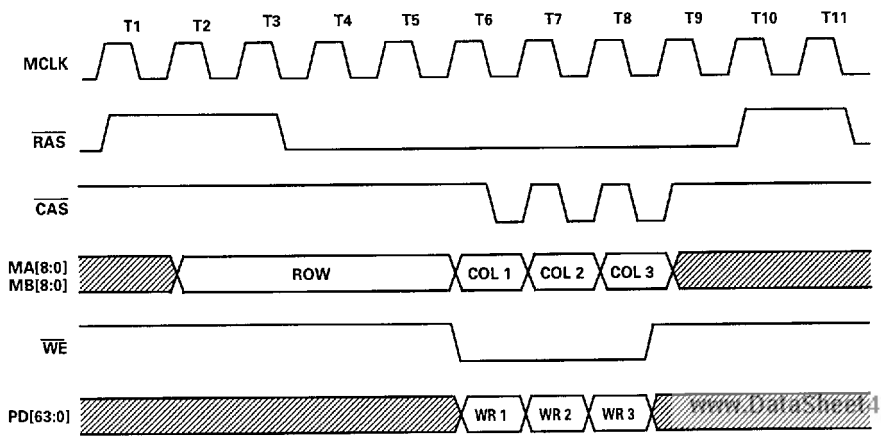


Figure 3-5. 1 Cycle EDO Read Timing



www.DataSheet4U.com

Figure 3-6. 1 Cycle EDO Write Timing

www.DataSheet4U.com

memory controller remains in page mode and checks for another memory request. If one is pending, it performs the 1.5 MCLK $\overline{RAS}$ precharge as shown in Figure 3-6. If no request is pending, the $\overline{RAS}$ precharge may be delayed from that shown in the figure and will be 2.5 MCLKs when it occurs.

Figure 3-7 shows a read/modify/write cycle with 1 cycle EDO operation. A dummy cycle is added between the read and write.

All line draws, image transfers, linear addressing and pixel formatter operation are not supported with 1 cycle EDO. 2 cycle EDO operation will automatically be used with these functions.

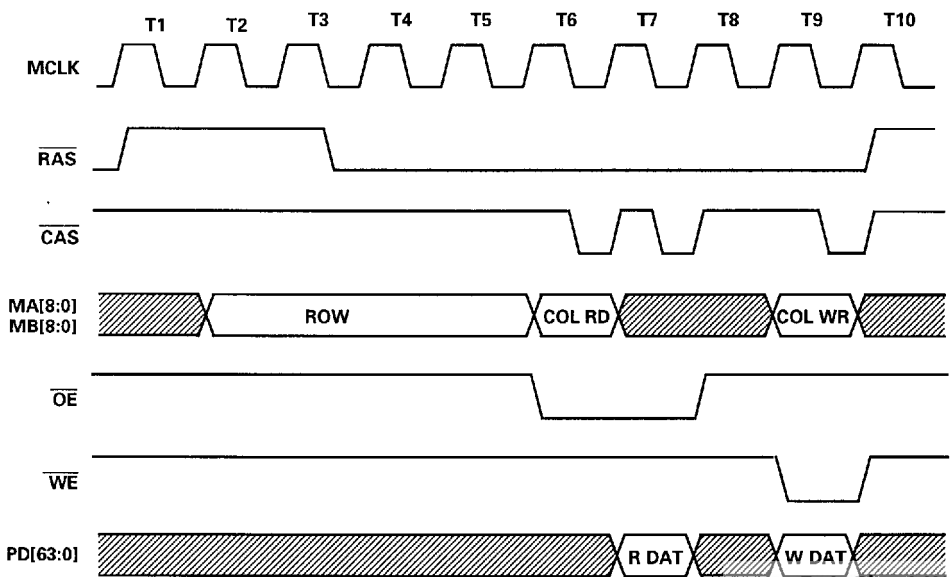
3.10 BURST MODE DRAM SUPPORT

Bits 3-2 of CR36 are set to 01b to indicate that burst mode DRAM operation is being used. Note that $\overline{CAS}$/$\overline{OE}$/$\overline{WE}$ stretch capability provided by bits 1-0 of CR68 cannot be used with burst mode operation.

With burst mode DRAM, only the initial column address is required. An internal counter in the DRAM then generates subsequent sequential column addresses within the same page. Valid read data is driven by the DRAM after the falling edge of each $\overline{CAS}$ and is latched by the next falling edge of $\overline{CAS}$. Write data is driven to the DRAM on each $\overline{CAS}$ rising edge.

When bursting is not feasible, such as across a page, for non-sequential addressing and for read/modify/writes, the memory controller automatically use standard EDO operation instead.

All line draws, image transfers, linear addressing and pixel formatter operation are not supported with burst mode DRAM. Standard EDO operation will automatically be used with these functions.



www.DataSheet4U.com

Figure 3-7. 1 Cycle EDO Read/Modify/Write Timing

www.DataSheet4U.com

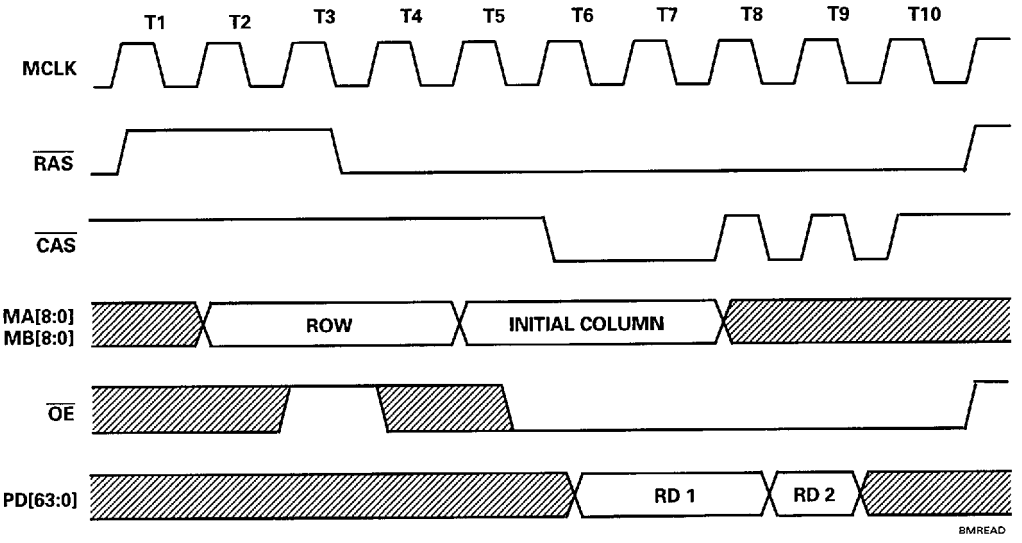


Figure 3-8. Burst Mode Read Timing

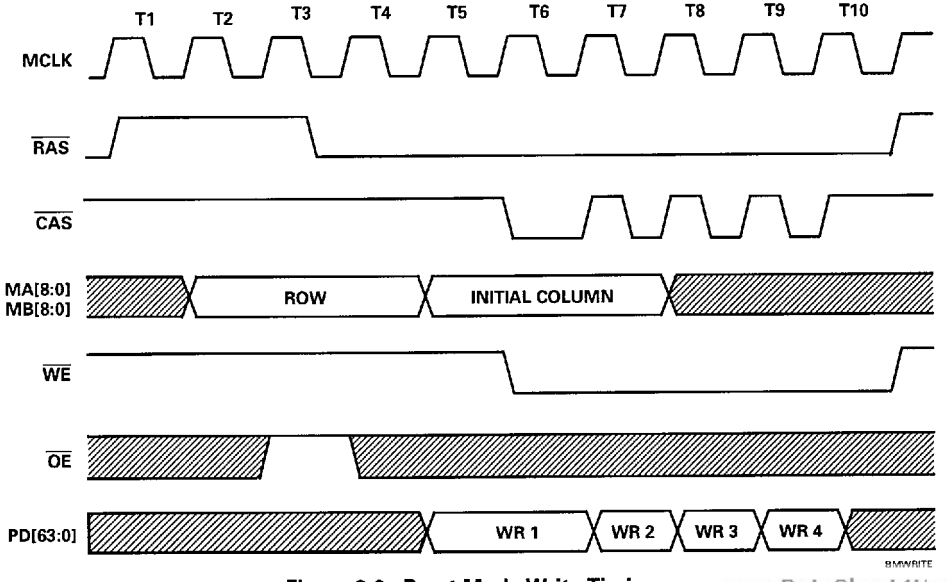


Figure 3-9. Burst Mode Write Timing

www.DataSheet4U.com

www.DataSheet4U.com

Section 4: Electrical Data

4.1 MAXIMUM RATINGS

Table 4-1. Absolute Maximum Ratings

Ambient temperature	0° C to 70° C
Storage temperature	-40° C to 125° C
DC Supply Voltage	-0 5V to 7 0V
I/O Pin Voltage with respect to V_{SS}	-0.5V to V_{DD}+0.5V

4.2 DC SPECIFICATIONS

Table 4-2. DC Specifications (VDD = 5V ± 5%, Operating Temperature 0° C to 70° C)

Symbol	Parameter	Min	Max	Unit
V_{IL}	Input Low Voltage		0 8	V
V_{IH}	Input High Voltage	2.0 (Note 1)		V
V_{OL}	Output Low Voltage		V_{SS} + 0.4	V
V_{OH}	Output High Voltage	2.4		V
I_{OL1}	Output Low Current	4 (Note 2)		mA
I_{OH1}	Output High Current	-2		mA
I_{OL2}	Output Low Current	8 (Note 3)		mA
I_{OH2}	Output High Current	-4		mA
I_{OL3}	Output Low Current	16 (Note 4)		mA
I_{OH3}	Output High Current	-8		mA
I_{OL4}	Output Low Current	24 (Note 5)		mA
I_{OH4}	Output High Current	-10		mA
I_{OL5}	Output Low Current (TTL)	9 (Note 6)		mA
I_{OH5}	Output High Current (TTL)	-9		mA
I_{OZ}	Output Tri-state Current		1	µA
C_{IN}	Input Capacitance		5	pF
C_{OUT}	Output Capacitance		5	pF
I_{CC}	Power Supply Current		425	mA

www.DataSheet4U.com

www.DataSheet4U.com

Notes for Table 4-2

1. 2.4 V Max for SCLK
2. I_{OL1}, I_{OH1} for pins $\overline{\text{BLANK}}$, $\overline{\text{CAS}}$[7:0], $\overline{\text{DACRD}}$, $\overline{\text{DACWR}}$, GD[7:0], HSYNC, $\overline{\text{INTA}}$ (SINTR) MA[8:0], MB[8:0], PA[15:0], PD[63:0], RS[2:0], $\overline{\text{RMSTRD}}$, STWR, VSYNC, $\overline{\text{BGNT}}$
3. I_{OL2}, I_{OH2} for pin VCLK
4. I_{OL3}, I_{OH3} for pins $\overline{\text{OE}}$[1:0], $\overline{\text{RAS}}$[1:0]
5. I_{OL4}, I_{OH4} for pins PAR ($\overline{\text{ABEN}}$), $\overline{\text{STOP}}$ ($\overline{\text{DBEN}}$), $\overline{\text{DEVSEL}}$ ($\overline{\text{LOCA}}$), $\overline{\text{TRDY}}$ ($\overline{\text{SRDY}}$), $\overline{\text{WE}}$[1:0]
6. I_{OL5}, I_{OH5} for pins AD[31:0], DBDIR

Note

Pin names for VL-Bus configurations are shown in parentheses.

4.3 AC SPECIFICATIONS

Notes:

1. All AC timings are based on an 80 pF test load.
2. Functional timing diagrams are found in the appropriate functional descriptions section, i.e., System Bus Interfaces, Display Memory or Miscellaneous Functions in the Vision864 data book or Functional Changes in this data booklet.

www.DataSheet4U.com

www.DataSheet4U.com

4.3.1 Clock Timing

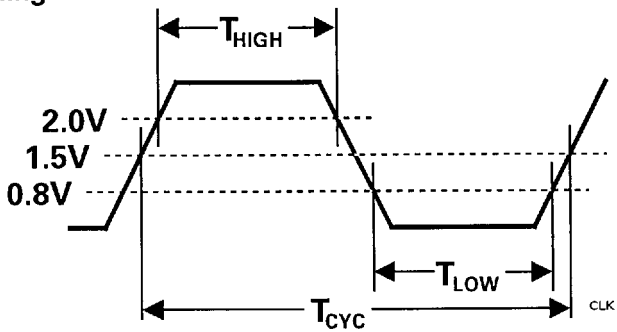


Figure 4-1. Clock Waveform Timing

Table 4-3. Clock Waveform Timing

Symbol	Parameter	Min	Max	Units	Notes
T_{CYC}	SCLK Cycle Time (VL-Bus)	20	125	ns	1
	SCLK Cycle Time (PCI)	30	125	ns	1
	MCLK Cycle Time	16.7	25	ns	3
	MCLK Cycle Time	20	25	ns	4
	DCLK Cycle Time (VGA Mode)	25	100	ns	1
	DCLK Cycle Time (Enhanced Mode)	10	100	ns	1
T_{HIGH}	SCLK High Time (VL-Bus)	8	80	ns	
	SCLK High Time (PCI)	12	80	ns	
	MCLK High Time	6	10	ns	3
	MCLK High Time	8	12	ns	4
	DCLK High Time (VGA Mode)	10	65	ns	
	DCLK High Time (Enhanced Mode)	4	65	ns	
T_{LOW}	SCLK Low Time (VL-Bus)	8	80	ns	
	SCLK Low Time (PCI)	12	80	ns	
	MCLK Low Time	6	10	ns	3
	MCLK Low Time	8	12	ns	4
	DCLK Low Time (VGA Mode)	10	65	ns	
	DCLK Low Time (Enhanced Mode)	4	65	ns	
	SCLK, MCLK, DCLK Slew Rate	1	4	V/ns	2

Notes:

1. $f_{DCLK} \geq 1/2\ f_{SCLK}$ to ensure valid writes to the clock chip.
2. Rise and fall times are specified in terms of the edge rate measured in V/ns. This slew rate must be met across the minimum peak-to peak portion of the clock waveform.
3. For fast page and standard EDO DRAM
4. For single cycle EDO and burst mode DRAM

www.DataSheet4U.com

www.DataSheet4U.com

4.3.2 Input/Output Timing

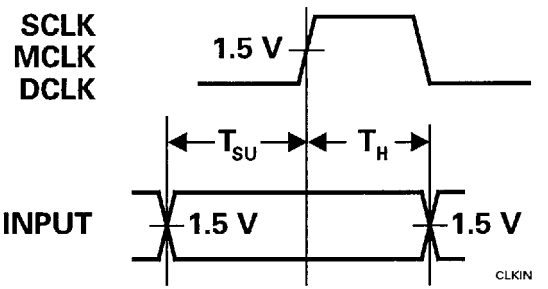


Figure 4-2. Input Timing

Table 4-4. SCLK-Referenced Input Timing

PCI Bus			
Symbol	**Parameter**	**Min**	**Units**
T_{SU}	AD[31:0], $\overline{C/BE[3:0]}$, $\overline{FRAME}$, $\overline{IRDY}$, $\overline{IDSEL}$ setup	7	ns
T_H	AD[31:0] hold	1	ns
T_H	$\overline{C/BE[3:0]}$, $\overline{FRAME}$, $\overline{IRDY}$, $\overline{IDSEL}$ hold	0	ns
VL-Bus			
Symbol	**Parameter**	**Min**	**Units**
T_{SU}	AD[31:2], $\overline{BE[3:0]}$, SM/$\overline{IO}$, SW/$\overline{R}$, $\overline{SADS}$ (address phase) setup	12	ns
T_H	AD[31:2], $\overline{BE[3:0]}$, SM/$\overline{IO}$, SW/$\overline{R}$, $\overline{SADS}$ (address phase) hold	0	ns
T_{SU}	AD[31:2], $\overline{BE[3:0]}$, D1, D0, $\overline{SADS}$ (data phase) setup	4	ns
T_H	AD[31:2], $\overline{BE[3:0]}$, D1, D0, $\overline{SADS}$ (data phase) hold	0	ns
T_{SU}	$\overline{RDYIN}$ setup	6	ns
T_H	$\overline{RDYIN}$ hold	1	ns
Miscellaneous			
Symbol	**Parameter**	**Min**	**Units**
T_{SU}	ROM Data GD[7:0] Setup (PCI)	5	ns
T_H	ROM Data GD[7:0] Hold (PCI)	7	ns
T_{SU}	General Input Port GD[7:0] setup	5	ns
T_H	General Input Port GD[7:0] hold	7	ns

www.DataSheet4U.com

www.DataSheet4U.com

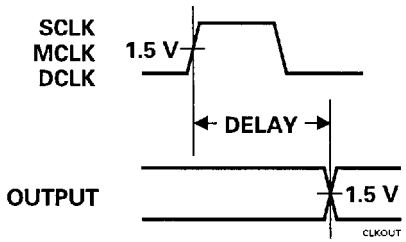


Figure 4-3. Output Timing

Table 4-5. SCLK-Referenced Output Timing

PCI Bus				
Parameter	**T_{MIN}**	**T_{MAX}**	**Units**	**Notes**
AD[31:0] valid delay	2	16	ns	1
$\overline{DEVSEL}$, PAR delay	2	11	ns	Medium $\overline{DEVSEL}$ timing used
$\overline{STOP}$ delay	2	11	ns	
$\overline{TRDY}$ delay	2	11	ns	
$\overline{INTA}$ delay	2	11	ns	
VL-Bus				
Parameter	**T_{MIN}**	**T_{MAX}**	**Units**	**Notes**
AD[31:2], D1, D0 valid delay	7	16	ns	
SINTR delay	5	30	ns	
$\overline{SRDY}$ delay	5	11	ns	
$\overline{LOCA}$ active delay	5	15	ns	
$\overline{LOCA}$ inactive delay	5	20	ns	
DBDIR delay	5	16	ns	
$\overline{ABEN}$, $\overline{DBEN}$ active delay (min non-overlap)	5	12	ns	
$\overline{ABEN}$, $\overline{DBEN}$ active delay (max non-overlap)	5	15	ns	
$\overline{ABEN}$, $\overline{DBEN}$ inactive delay	5	13	ns	
Miscellaneous				
Parameter	**T_{MIN}**	**T_{MAX}**	**Units**	**Notes**
RS[2:0] delay	5	25	ns	
$\overline{RMSTRD}$ delay	3	15	ns	
ROM Address valid delay (PCI)	5	30	ns	
AD[7:0] ROM Data valid delay (PCI)	5	30	ns	
$\overline{DACRD}$, $\overline{DACWR}$ delay	3	20	ns	

Note

1. Due to the timing for $\overline{TRDY}$ for read cycles, data is not sampled on the clock edge immediately following its becoming valid. This guarantees the PCI 2.0 specification time of 11 ns.

www.DataSheet4U.com

www.DataSheet4U.com

Table 4-6. MCLK-Referenced Input Timing

Symbol	Parameter	Min	Units
T_{SU}	PD[63:0] setup (EDO memory)	0	ns
T_H	PD[63:0] hold (EDO memory)	10	ns
T_{SU}	General Input Port GD[7:0] setup	5	ns
T_H	General Input Port GD[7:0] hold	7	ns

Table 4-7. MCLK-Referenced Output Timing

Parameter	T_{MIN}	T_{MAX}	Units
$\overline{RAS}$[1:0] active delay	5	15	ns
$\overline{RAS}$[1:0] inactive delay	5	14	ns
$\overline{CAS}$[7.0], $\overline{OE}$[1:0], $\overline{WE}$[1:0] active delay	7	15	ns
$\overline{CAS}$[7:0], $\overline{OE}$[1:0], $\overline{WE}$[1:0] inactive delay (min stretch) for fast page and standard EDO DRAM	6	8	ns
$\overline{CAS}$[7:0], $\overline{OE}$[1:0], $\overline{WE}$[1:0] inactive delay (max stretch) for fast page and standard EDO DRAM	6	17	ns
$\overline{CAS}$[7:0] active delay with respect to falling edge of MCLK for 1 cycle EDO and burst mode operation	6	15	ns
MA[8:0], MB[8:0] valid delay	5	24	ns
PD[63:0] valid delay	5	27	ns

Table 4-8. DCLK-Referenced Output Timing

Parameter	T_{MIN}	T_{MAX}	Units
$\overline{BLANK}$, HSYNC, VSYNC	3	17	ns
VCLK delay	7	14	ns
PA[15:0] valid delay	5	14	ns
STWR delay	3	20	ns
General Output Port Data GD[7:0] delay	5	25	ns

Table 4-9. Fast Page, 1 Cycle EDO, Burst Mode Memory Input Timing

Symbol	Parameter	Min	Units
T_{SU}	PD[63:0] setup to $\overline{CAS}$[7:0] inactive	0	ns
T_H	PD[63:0] hold from $\overline{CAS}$[7:0] inactive	5	ns

www.DataSheet4U.com

www.DataSheet4U.com

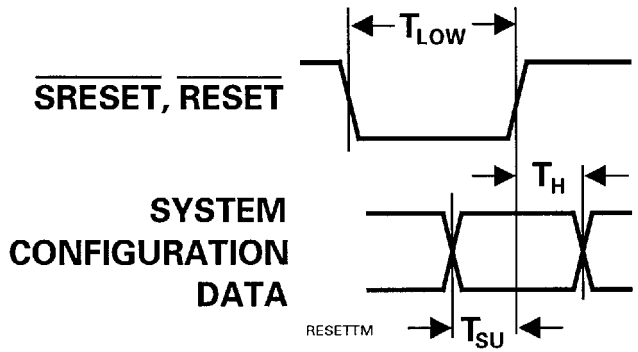


Figure 4-4. Reset Timing

Table 4-10. Reset Timing

Symbol	Parameter	Min	Units
T_{LOW}	$\overline{SRESET}$ (VL) or $\overline{RESET}$ (PCI) active pulse width	400	ns
T_{SU}	PD[23:0] setup to $\overline{SRESET}$ (VL) or $\overline{RESET}$ (PCI) inactive	20	ns
T_H	PD[23:0] hold from $\overline{SRESET}$ (VL) or $\overline{RESET}$ (PCI) inactive	10	ns

4.4 OUTPUT BUFFER MODEL

Figure 4-5 shows a first order output buffer model for the Vision868. The parameters are:

Parameter	Description
dV/dt	Mininum and maximum rate of change of the open circuit voltage source used in the buffer model
R_0	Minimum and maximum output impedance of the buffer model
C_0	Minimum and maximum capacitance used in the buffer model
L_P	Minimum and maximum package inductance
C_P	Minimum and maximum package capacitance

The parameter values appropriate for the Vision868 depend on the manufacturing process and can be obtained directly from S3.

www.DataSheet4U.com

www.DataSheet4U.com

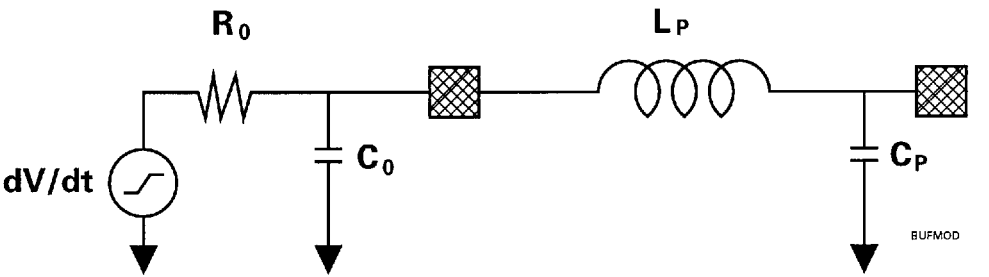


Figure 4-5. First Order Output Buffer Model

www.DataSheet4U.com

www.DataSheet4U.com

Section 5: Enhanced Mode Programming

Enhanced mode provides a level of performance far beyond what is possible with the VGA architecture. Hardware line drawing, BitBLT, rectangle fill and other drawing functions are implemented. Also implemented are data manipulation functions, such as data extension, data source selection, and read/write bitplane control. Hardware clipping is supported by 4 registers that define a rectangular clipping area. While in Enhanced mode, the display memory bit map can be updated in two ways. One is to have the CPU write directly to memory. (This is also possible in non-Enhanced modes via paging.) The other is to have the CPU issue commands to the Graphics Engine, which then controls pixel updating. This section explains these two methods and provides a comprehensive set of Enhanced mode programming examples.

5.1 MEMORY-MAPPED I/O

The Vision868 provides two memory-mapped I/O (MMIO) schemes. One method is identical to that provided by the Vision864 and provides compatibility with older software. This provides memory mapping of a limited number of Enhanced mode registers. The second method is new to the Vision868. It incorporates linear addressing and provides memory mapping of all registers. In addition, the Enhanced mode registers can be accessed by a packed method that improves performance by allowing two 16-bit registers to be accessed by a single 32-bit write cycle. More registers can be accessed via the packed method than by the standard method. Each of these MMIO methods is described below.

5.1.1 Backward-Compatible MMIO

Most of the Enhanced registers can be memory-mapped. This function is enabled by setting bits 4-3 of CR53 to 10b. When bit 3 of CR53 is cleared to 0, the old MMIO can also be enabled by setting bit 5 of 4AE8H to 1.

Image writes normally made via I/O addresses E2E8H and E2EAH (the Pixel Data Transfer registers) are made instead by accessing any memory location in the 32-KByte address space from A0000H to A7FFFH. This allows efficient use of the MOVSW and MOVSD assembly language commands. Accesses must be to even word or doubleword addresses, depending on the specification of the bus width via bits 10-9 of 98E8H. Software must not make E2E8H, E2EAH writes beyond the A7FFFH range.

Accesses to the Enhanced command registers are made to particular locations in the A8000H to AFFFFH address range as shown in Table 5-1. Both 16-bit reads and writes are supported. Only 32-bit writes (bit 9 of BEE8H_E set to 1) are supported.

www.DataSheet4U.com

www.DataSheet4U.com

Table 5-1. Enhanced Registers Memory Mapping

Register Mnemonic	I/O Address (Hex) Old MMIO = Axxxx	Register Mnemonic (Packed)	Packed MMIO Offset (Hex) (100 xxxx)
CUR_Y, CUR_X	82E8, 86E8	ALT_CURXY	8100, 8102
CUR_Y2, CUR_X2	82EA, 86EA	ALT_CURXY2	8104, 8106
DESTY_AXSTP, DESTX_DIASTP	8AE8, 8EE8	ALT_STEP	8108, 810A
Y2_AXSTP2, X2_DIASTP2	8AEA, 8EEA	ALT_STEP2	810C, 810E
ERR_TERM, ERR_TERM2	92E8, 92EA	ALT_ERR	8110, 8112
CMD, CMD2	9AE8, 9AEA	ALT_CMD	8118, 811A
SHORT_STROKE	9EE8		811C
BKGD_COLOR	A2E8		8120
FRGD_COLOR	A6E8		8124
WRT_MASK	AAE8		8128
RD_MASK	AEE8		812C
COLOR_CMP	B2E8		8130
BKGD_MIX, FRGD_MIX	B6E8, BAE8	ALT_MIX	8134, 8136
SCISSORS_T, SCISSORS_R	BEE8_1, BEE8_2		8138, 813A
SCISSORS_B, SCISSORS_R	BEE8_3, BEE8_4		813C, 813E
PIX_CNTL, MULT_MISC2	BEE8_A, BEE8_D		8140, 8142
MULT_MISC, READ_SEL	BEE8_E, BEE8_F		8144, 8146
MIN_AXIS_PCNT, MAJ_AXIS_PCNT	BEE8_0, 96E8	ALT_PCNT	8148, 814A
MAJ_AXIS_PCNT2	96EA		814C
ROPMIX	D2E8		8150
DESTX_DIASTP, MAJ_AXIS_PCNT	8EE8, 96E8	MAXX	8154, 8156
PIX_TRANS	E2E8, E2EA	PIX-TRANS	0000
PAT_BG_COLOR	E6E8		8164
PAT_Y, PAT_X	EAE8, EAEA	ALT_PAT	8168, 816A
PAT_FG_COLOR	EEE8		816C

5.1.2 New MMIO

The new MMIO method for the Vision868 provides a 64-MByte addressing window starting at the base address specified in CR59-5A. This space is divided into a 32-MByte space for little endian (Intel-style) addressing and a 32-MByte space for big endian (Power PC-style) addressing. All registers and data transfer locations are mapped into this area as shown in Table 5-2.

The new MMIO is enabled by setting bits 4-3 of CR53 to 11b. This is the default for a PCI bus configuration, allowing PCI software immediate access to all registers and the ability to relocate the address space. VL-Bus configurations power up with bits 4-3 of CR53 cleared to 00b, disabling both old and new MMIO operation.

www.DataSheet4U.com

If either the old or new MMIO is enabled, bit 7 of SR9 allows register access to be either programmed I/O (IN, OUT) or MMIO (MOV) or MMIO only.

www.DataSheet4U.com

Table 5-2. New MMIO Addresses

Description	Memory Map Offset
Low 32 MBytes: Little Endian	**A[25:0] Hex**
Linear Addressing (16M)	000 0000:0FF FFFF
Image Transfer Data (via E2E8, E2EA) (32K)	100 0000.100 7FFF
PCI Configuration Space	100 8000:100 803F
Packed Enhanced Registers	100 8100:100 816F
Current Y-Position Register	100 82E8
Current Y-Position 2 Register	100 82EA
VGA 3B? Registers	100 83B0:100 83BX
VGA 3C? Registers	100 83C0:100 83CX
VGA 3D? Registers	100 83D0:100 83DX
Setup Option Select Register (102H)	100 8502
Subsystem Status Enhanced Register (42E8H)	100 8504
Video Subsystem Enable Register (46E8H)	100 8508
Advanced Function Control Register (4AE8H)	100 850C
Enhanced Mode Registers	100 86E8:100 EEEA
Pixel Formatter Data Transfer	101 0000:101 3FFF
Pixel Formatter Mask Data	101 4000:101 7FFF
Pixel Formatter Registers	101 8080:101 809F
High 32 MBytes: Big Endian	**A[25:0] Hex**
Linear Addressing (16M)	200 0000.2FF FFFF
Image Transfer Data (via E2E8, E2EA) (32K)	300 0000:300 7FFF
PCI Configuration Space	300 8000:300 803F
Packed Enhanced Registers	300 8100:300 816F
Current Y-Position Register	300 82E8
Current Y-Position 2 Register	300 82EA
VGA 3B? Registers	300 83B0:300 83BX
VGA 3C? Registers	300 83C0:300 83CX
VGA 3D? Registers	300 83D0:300 83DX
Setup Option Select Register (102H)	300 8502
Subsystem Status Enhanced Register (42E8H)	300 8504
Video Subsystem Enable Register (46E8H)	300 8508
Advanced Function Control Register (4AE8H)	300 850C
Enhanced Mode Registers	300 86E8:300 EEEA
Pixel Formatter Data Transfer	301 0000:301 3FFF
Pixel Formatter Mask Data	301 4000:301 7FFF
Pixel Formatter Registers	301 8080:301 809F

The base address is taken from bits 31-26 of the linear address window position (bits 7-2 of CR59 or the high order 6 bits of the the PCI Base Address 0). This is concatenated with the address offset specified by the programmer per Table 5-2. The base address bits must not be all 0's, as this may cause operating system address spaces to be overwritten.

www.DataSheet4U.com

www.DataSheet4U.com

When the new MMIO is enabled, both 16-bit MMIO reads and writes are supported. Only 32-bit writes are supported.

5.1.2.1 Big/Little Endian Support

The Vision868 provides support for both big and little endian addressing. In addition to using the new MMIO addressing as shown in Table 5-2, the required byte swapping must be specified as shown in Table 5-3.

Table 5-3. Big Endian Byte Swap Select

Byte Swapping Affects:	Swapping Specified by:
Linear Addressing, Video Engine (read/write)	CR53, bits 2-1
Image Data (writes to Graphics Engine)	CR61, bits 6-5
All other registers/data areas (read/write)	CR54, bits 1-0

5.1.2.2 Packed MMIO Register Mapping

For improved performance, most of the Enhanced mode registers can also be accessed via a packed configuration. The 16-bit registers are paired so that two registers can be accessed via a single 32-bit write. Reads must be 16-bit. The address offsets from the base address for this packed configuration are given in Table 5-1. The packed register access function is enabled when the new MMIO is enabled (bit 3 of CR53 set to 1).

5.2 DIRECT BITMAP ACCESSING—LINEAR ADDRESSING

Linear addressing is useful when software requires direct access to display memory. The Vision868 provides two linear addressing schemes. One is the same as provided for the Vision864 and thus provides compatibility with older software. The second is integrated into the new memory-mapped I/O method and thus requires updated drivers for use.

5.2.1 Backward-Compatible Linear Addressing

Enhanced mode operation must be enabled before linear addressing is enabled. This means that bit 0 of 4AE8H is set to 1 to enable Enhanced mode functions and bit 3 of CR31 is set to 1 to specify Enhanced mode memory mapping.

The Vision868 provides linear addressing of up to 4 MBytes of display memory. Linear addressing of more than 64 KBytes requires that Intel-style CPUs be operated in 386 protected mode.

The Graphics Engine busy flag, bit 9 of 9AE8H, should be verified to be 0 (not busy) before linear addressing is enabled. This is done by setting bit 4 of CR58 or bit 4 of 4AE8H to 1. The size of the linear address window is set via bits 1-0 of CR58. The base address for the linear addressing window is set via CR59 and CR5A (or via the Base Address 0 (Index 10H) PCI configuration register for PCI systems).

www.DataSheet4U.com

The linear addressing window size can be set to 64 KBytes. The base address for the window is set by programming bits 31-16 of the window position in CR59-CR5A. This allows the CPU to be operated in real mode. If bit 0 of CR31 is set to 1, the memory page offset (64K bank) specified in bits 5-0 of CR6A

www.DataSheet4U.com

is added to the linear addressing window position base address, allowing access to up to 4 MBytes of display memory through a 64-KByte window.

5.2.2 New Linear Addressing

The first 16 MBytes of each 32M address space (big and little endian) are dedicated to linear addressing. A maximum of 4 MBytes of each address space (starting at the lowest address of the space) is usable with the Vision868. When the new MMIO is enabled (bits 4-3 of CR53 set to 11b), the base address is taken from bits 31-26 of the linear address window position (bits 7-2 of CR59 or the high order 6 bits of the the PCI Base Address 0). This is concatenated with the display memory address specified by the programmer.

In addition to enabling the new MMIO, the programmer must also enable linear addressing and specify the window size exactly as required for the old linear addressing. Note that since only bits 31-26 are used to specify the base address, A0000H cannot be specified and the 64K banking scheme possible with the old linear addressing cannot be used with the new linear addressing.

When big endian addressing is used, the required byte swapping for linear addressing is specified by bits 2-1 of CR53. This applies to both reads and writes.

5.3 BITMAP ACCESS THROUGH THE GRAPHICS ENGINE

When updating the display bitmap through the Graphics Engine, all CPU data moves through the Pixel Data Transfer registers (E2E8H and E2EAH). These can be memory mapped as explained in Memory Mapping of Enhanced Mode Registers later in this section.

The Graphics Engine manipulates the bits for each pixel to assign a color value, which is then translated by the RAMDAC into the color displayed on the CRT. Selected bits in a pixel can be masked off from being displayed by programming the DAC Mask register (3C6H). The Vision868 can manipulate 64 bits each clock cycle, from two 32-bit pixels to eight 8-bit pixels.

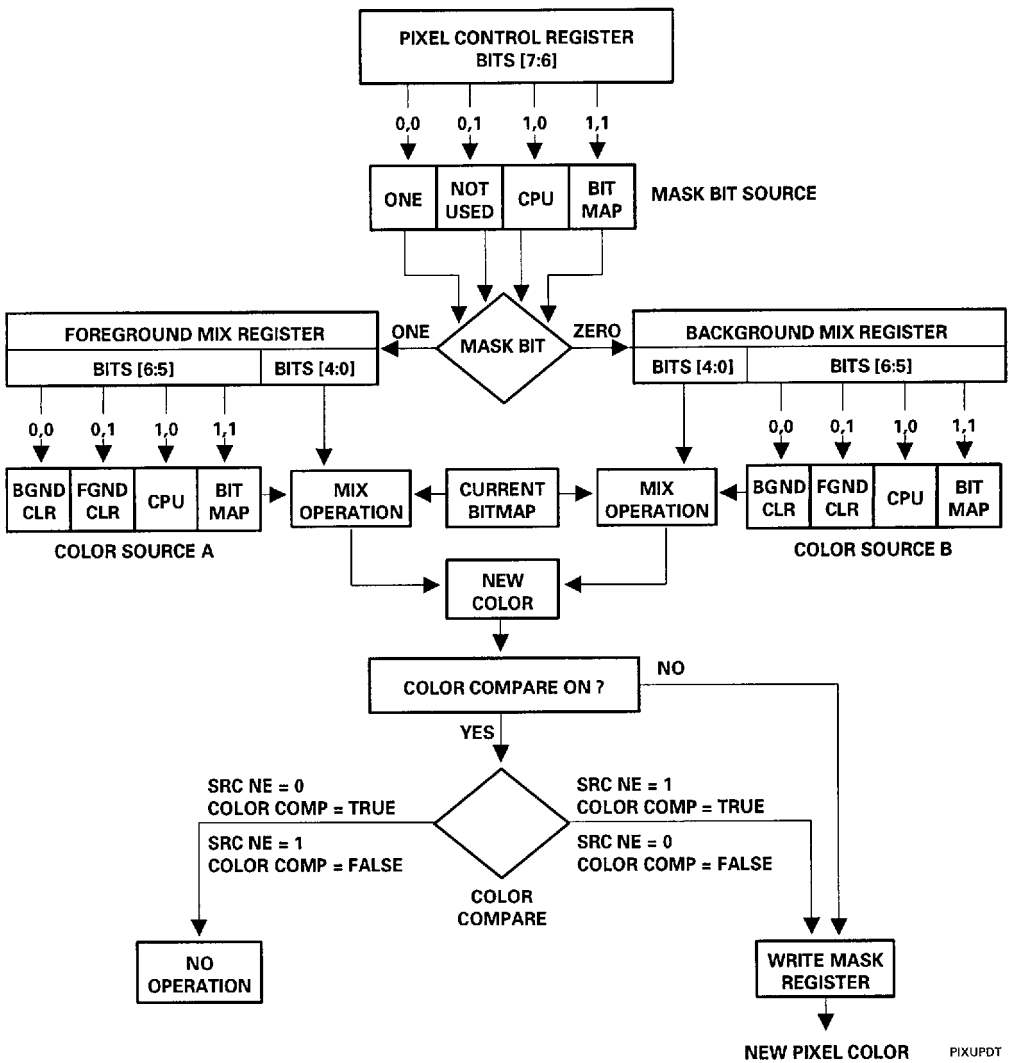
Figure 5-1 is a flowchart for the process of updating the color of each pixel in the display bitmap. Start at the block labeled 'New Color' in the middle of Figure 5-1. At this stage, a color has been determined that may or may not be used to update a pixel in the bitmap. How this color is determined will be covered later.

The first hurdle for the new color is the color compare process. If this is turned off (bit 8 of BEE8H, Index 0EH = 0), the new color is passed to the Write Mask register (AAE8H). If the plane to which the pixel update is directed has been masked off in this register, no update occurs. Otherwise, the new color value is written to the bitmap.

If color compare is enabled (bit 8 of BEE8H, Index 0EH = 1), the new color value (source) is compared to a color value programmed into the Color Compare (B2E8H) register. The sense of the color comparison is determined by the SRC NE (source not equal) bit (bit 7) of BEE8H, Index 0EH. If this bit is 0, the new pixel color value is passed to the write mask only when the source color does not match the color in the Color Compare register. If this bit is 1, the new pixel color value is passed to the write mask only when the source color matches the color in the Color Compare register. If the new pixel color value is not passed to the write mask, no update occurs. Notice that the source color is used for the comparison, as opposed to the destination (bitmap) color used by the standard VGA color compare operation.

www.DataSheet4U.com

www.DataSheet4U.com



www.DataSheet4U.com

Figure 5-1. Pixel Update Flowchart

www.DataSheet4U.com

There are two approaches to determining the new color. One approach is depicted in Figure 5-1 and is described next. The second approach is used with the ROPBLT command and is described at the end of this section.

For all commands except a ROPBLT, the new color is the result of a logical mix performed on a color source and the current color in the bitmap. For example, the color source could be XORed with the bitmap color. The new color can also be selected by operating on only the color source or the bitmap color, e.g., NOT color source. Both the color source and the logical mix operation are specified in either the Background Mix register (B6E8H) or the Foreground Mix register (BAE8H). Which of these two registers is used is determined by the settings of bits [7:6] of the Pixel Control register (BEE8H, Index 0AH).

To set up the pixel color updating scheme, the programmer specifies one of four color sources by writing bits 6-5 of the Background Mix and Foreground Mix registers. The color sources are:

- Background Color register (A2E8H)
- Foreground Color register (A6E8H)
- CPU (via the Pixel Data Transfer registers (E2E8H, E2EAH))
- Current display bitmap color

One of 16 logical operations is chosen by writing bits 3-0 of the Background Mix and Foreground Mix registers. Examples of logical operations are making the new pixel color index equal to the NOT of the current bitmap color or making the new color value equal to the XOR of the source and current bitmap color values.

When the logical operation and color source have been specified in the Background and Foreground Mix registers, bits 7-6 of the Pixel Control register are written to specify the source of the mask bit value. If the resulting mask bit is a 'ONE', the Foreground Mix register is used to determine the color source and mix. If the mask bit is a 'ZERO', the Background Mix register is used to determine the color source and mix. There are three sources for the mask bit value:

- Always ONE (Foreground Mix register used)
- CPU (via the Pixel Data Transfer registers (E2E8H, E2EAH))
- Bitmap (display)

Setting bits 7-6 to 00b sets the mask bit to 'ONE'. All drawing updates to the display bitmap use the Foreground Mix register settings. This setup is used to draw solid lines, through-the-plane image transfers to display memory and BitBLTs.

If bits 7-6 are set to 10b, the mask bit source is the CPU. After the draw operation command is issued to the Drawing Command register (9AE8), a mask bit corresponding to every pixel drawn on the display must be provided via the Pixel Data Transfer register(s). If the mask bit is 'ONE', the Foreground Mix register is used. If the mask bit is 'ZERO', the Background Mix register is used. Note that if the color source is the CPU, the mask bit source cannot also be the CPU, and vice versa. This setup is used to transfer monochrome images such as fonts and icons to the screen.

If bits 7-6 are set to 11b, the current display bit map is selected as the mask bit source. The Read Mask register (AAE8H) is set up to indicate the active planes. When all bits of the read-enabled planes for a pixel are a 1, the mask bit 'ONE' is generated. If any one of the read-enabled planes is a 0, then a mask bit 'ZERO' is generated. If the mask bit is 'ONE', the Foreground Mix register is used. If the mask bit is 'ZERO', the Background Mix register is used. Note that if the color source is the bitmap, the mask bit

www.DataSheet4U.com

www.DataSheet4U.com

source cannot also be the bitmap, and vice versa. This setting is used to BitBLT patterns and character images.

The ROPBLT command provides 256 raster operations that define the new color. Each operation involves one or more of a color source, an off-screen pattern and a destination color. These colors can be combined via various logical operations as specified in Appendix A. The source can be the CPU or current bitmap and can be color or monochrome. The pattern can also be either color or monochrome. The destination is always the current bitmap (screen) and is always color. Certain ROPBLT raster operations are equivalent to other commands. For example, if the source is the current bitmap and no pattern is used, this is the same operation as a BitBLT. ROPBLTs should not be used for image transfers (source is the CPU) when the raster operation does not include a source, e.g., Dn (NOT destination, or invert the destination bits). Instead, use the appropriate image transfer command.

5.4 PROGRAMMING

Three different programming schemes are available, I/O, standard MMIO and packed register MMIO. Examples of how each is used to assign vertical and horizontal coordinates to Current X and Y Position registers (82E8H and 86E8H) are:

I/O Format:

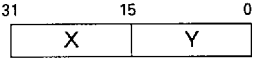
```
MOV  DX,CUR_X
MOV  AX,X
OUT  DX,AX
MOV  DX,CUR_Y
MOV  AX,Y
OUT  DX,AX
```

Standard MMIO Format:

```
Enable MMIO
Point ES to A000H (old MMIO) or base address (new MMIO)
Load x and y values into AX and BX
MOV  ES:[CUR_Y], BX
MOV  ES:[CUR_X], AX
```

Packed Register MMIO:

```
Enable MMIO
Point ES to A000H (old MMIO) or base address (new MMIO)
Load the x and y values into EAX (y value in the low word and x value in the high word), I.e.,
```




```
MOV  ES:[ALT_CURXY], EAX
```

The packed register MMIO scheme is the most efficient and is used where appropriate in the programming examples provided later in this section. All assume that the ES register points to A000H is the old MMIO is being used or the base address if the new MMIO is being used.

www.DataSheet4U.com

www.DataSheet4U.com

5.4.1 Notational Conventions

The REGMNEMONIC on the left hand side of the arrow is the register mnemonic of the I/O port being written into. Text following a ';' is a comment.

```
REGMNEMONIC ⇐ XXXXH                ; Load a hexadecimal value into the register.
REGMNEMONIC ⇐ XXXXD                ; Load a decimal value into the register.
REGMNEMONIC ⇐ XXXX                 ; Load a decimal value into the register
REGMNEMONIC ⇐ XXXXXXXXXXXXXXXXB    ; Load a binary value into the register.
```

Image transfers (CPU pixel data writes to the frame buffer) are notated as follows:

```
COUNT
PIX_TRANS ⇐ IMAGEDATA
```

The COUNT is the number of CPU writes. PIX_TRANS means either the E2E8H, E2EAH pixel transfer registers or the 32K memory space from A0000H to A7FFFH as explained in Section 5.1.1 above or to the alternate 32K area when using the new MMIO as shown in Table 5-2.

5.4.2 Initial Setup

All examples assume the desired mode is selected.

The Bitmap Access Through the Graphics Engine section earlier in this section explains in detail how the colors, mixes and the data extensions are set for each example. These registers need not be set repeatedly before a series of draw commands if they use the same colors, mixes and data extension.

All bitmap updates are affected by the settings in the clipping registers (BEE8H, Indices 1-4) and the choice of internal or external clipping (BEE8H, Index E, bit 5). These must be set up so they include the area being drawn into.

If color compare is to be used, it must be enabled by setting bit 8 of BEE8H, Index 0EH to 1. Bit 7 of this register determines whether a TRUE or FALSE comparison allows the pixel update to continue. The comparison color is programmed into the Color Compare register (B2E8).

All planes are enabled for writing unless explicitly set otherwise in an example. This is done via the Write Mask register (AAE8H).

www.DataSheet4U.com

www.DataSheet4U.com

5.4.3 Programming Examples

This section provides programming examples for the following Enhanced mode drawing operations:

- Solid Line
- Textured Line
- Rectangle Fill Solid
- Image Transfer—Through the Plane
- Image Transfer—Across the Plane
- BitBLT—Through the Plane
- BitBLT—Across the Plane
- PatBLT—Through the Plane
- PatBLT—Across the Plane
- Short Stroke Vectors
- Polyline
- Polygon Fill Solid
- Polygon Fill Pattern -
- 4-Point Method Trapezoid Fill Solid
- 4-Point Method Trapezoid Fill Pattern
- Bresenham Parameter Trapezoid Fill Solid
- Bresenham Parameter Trapezoid Fill Pattern
- ROPBLT
- Programmable Hardware Cursor

Some programming steps are repeated in multiple examples. They are explained in detail at their first occurrence. Therefore, readers are encouraged to work through the examples from first to last. The register mnemonics used in the examples are listed in Table 5-1. Other mnemonics used are:

Mnemonic	Description
NEW	Mix = 00111b in bits 4-0 of BAE8H or B6E8H. This overwrites the present bitmap color value with a new value.
XOR	Mix = 00101b in bits 4-0 of BAE8H or B6E8H. The current bitmap color is XORed with the new color.

www.DataSheet4U.com

www.DataSheet4U.com

5.4.3.1 Solid Line

This command draws a one pixel wide solid line from screen coordinates x1,y1 to x2,y2. Bresenham parameters are used to define the line. The Pixel Control register (BEE8H, Index AH) must be set to A000H to select the Foreground Mix register to specify the color source and mix type.

Setup:

Drawing a line using axial coordinates requires programming the axial step constant into the Destination Y-Position/Axial Step Constant (8AE8H) register (DESTY_AXSTP), the diagonal step constant into the Destination X-Position/Diagonal Step Constant (8EE8H) register (DESTX_DIASTP) and the error term into the Error Term (92E8H) register (ERR_TERM). Calculation of these Bresenham parameters is based on the MAX and MIN parameters as calculated below.

MAX = maximum(ABS(x2–x1), ABS(y2–y1))
MIN = minimum(ABS(x2–x1), ABS(y2–y1))

where maximum means choose the largest of the two terms in parentheses and minimum means choose the smallest. ABS means take the absolute value of the expression.

Bits 7-5 of the Drawing Command (9AE8H) register (CMD) specify the drawing direction. Setting bit 7 to 1 means that the Y drawing direction is positive (y1 < y2). Clearing bit 7 to 0 means the Y drawing direction is negative (y1 > y2). Setting bit 6 to 1 means that Y is the major (longer) axis (ABS(x2–x1) > ABS(y2–y1)). Clearing bit 6 to 0 means that X is the major axis. Setting bit 5 to 1 means that the X drawing direction is positive (x1 < x2). Clearing bit 5 to 0 means that the X drawing direction is negative (x1 > x2). These values replace the DDD sequence in the write to the CMD register shown in the pseudocode below.

The mix NEW represents a setting of 0111b in bits 3-0 of the Foreground Mix (BAE8H) register (FRGD_MIX). This overwrites the present bitmap color value with a new value.

The remainder of the setup is:

```
ES:[FRGD_MIX] ⇐ 0027H              ; color source is FRGD_COLOR, mix type is NEW
ES:FRGD_COLOR] ⇐ 00000002H         ; color index
ES:[PIXEL_CNTL] ⇐ A000H            ; FRGD_MIX provides color source and mix type
```

Drawing Operation:

```
                    31        15        0
ES:[ALT_CURXY] ⇐    | x1      | y1      |   ; set starting coordinate

ES:[MAJ_AXIS_PCNT] ⇐ MAX - 1       ; length in pixels of the major axis - 1
                    31        15        0
ES:[ALT_STEP] ⇐     | 2*(MIN–MAX) | 2*MIN |  ; diagonal and axial step constants

If the X drawing direction is positive then
    ES:[ERR_TERM] ⇐ 2 * MIN – MAX          ; error term
else if the X drawing direction is negative
    ES[ERR_TERM] ⇐ 2 * MIN – MAX – 1       ; error term
ES:[CMD] ⇐ 00100000DDD10001b       ; Draw line command (bits 15-13, 11), draw (as opposed to
                                   ; just move current position)(bit 4), bit 0 is always 1
```

www.DataSheet4U.com

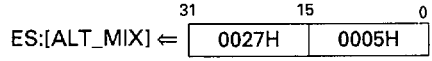
www.DataSheet4U.com

5.4.3.2 Textured Line

The line draw command can be used to draw a one pixel wide textured line from screen coordinates x1,y1 to x2,y2. The texture is created by (1) setting bits 7-6 of the Pixel Control register (BEE8H, Index AH) to A080H to specify the CPU as the source of the mask bit selecting the mix register, (2) specifying a background and foreground color, (3) setting bit 8 of the Command register (9AE8H) to 1 (wait for CPU data) and (4) setting bit 1 of the Command register to 1 (multi-pixel). When the pattern bit sent by the CPU is a 1, the Foreground Mix register specifies the the color source and mix. When the bit is a 0, the Background Mix register specifies the color source and mix. This example uses the mix NEW for the foreground mix, XOR for the background mix, foreground color index 2 and background color index 4. The 32-bit line texture/pattern (PATTERN) is 00110000111100110011000011110011b. This requires that bits 10-9 of the Command register be set to 10b to specify a 32-bit bus.

Setup:

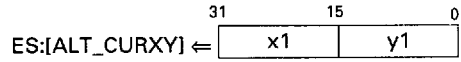
The XOR mix corresponds to a setting of 0101b in bits 3-0 of the Background Mix (B6E8H) register (BKGD_MIX). See the Solid Line example for an explanation of other parameters and registers used in this example.



; FRGD_COLOR is color source and NEW is mix,
; BKGD_COLOR is color source and XOR is mix

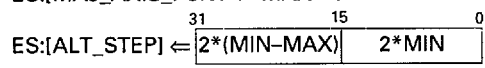
```
ES:[FRGD_COLOR] ⇐ 00000002H      ; color index
ES:[BKGD_COLOR] ⇐ 00000004H      ; color index
ES:[PIXEL_CNTL] ⇐ A080H          ; mask data selecting mix register is provided by the CPU
```

Drawing Operation:



; set starting coordinates

```
ES:[MAJ_AXIS_PCNT ⇐ MAX - 1      ; length in pixels of the major axis - 1
```



; diagonal and axial step constants

```
If the X drawing direction is positive then
    ES:[ERR_TERM] ⇐ 2 * MIN – MAX          ; error term
else if the X drawing direction is negative
    ES[ERR_TERM] ⇐ 2 * MIN – MAX – 1       ; error term
ES:[CMD] ⇐ 00100101DDD10011b     ; Draw line (bits 15-13, 11), 32-bit bus (bits 10-9), wait for data
                                 ; from the CPU (bit 8), draw (bit 4), multi-pixel (bit 1)

COUNT (of PATTERN dwords) = (MAX + 31)/32 (See Note)
PIX_TRANS ⇐ 00110000111100110011000011110011b    ; Output PATTERN to Pixel Data Transfer
                                                 ; registers COUNT times
```

Note

The COUNT of the number of writes required by the CPU is a function of the number of bits to be transferred and the width of the transfer (8, 16 or 32 bits as specified by bits 10-9 of the Drawing Command register (9AE8H)). The number of bits transferred per line must be an even multiple of the transfer width. If this is not the case, the last write per line must be padded with one or more dummy

www.DataSheet4U.com

www.DataSheet4U.com

bits to meet this requirement. For example, if the transfer width is 8 bits and nine bits are to be transferred for the line, two bytes must be written per line, with the upper 7 bits of the second byte padded. In general, the number of padding bits per line will vary from 0 to (n-1), where n is the transfer width in bits.

With a transfer width of 8 bits, the number of byte writes required per line can be determined from the formula n = (MAX+7)/8, with n being truncated to an integer if the result contains a fraction. Thus a MAX = 11 transfer requires (11+7)/8 = 2 1/4 = 2 bytes. The formulas for all transfer widths are given below.

8-bit transfers: COUNT = (MAX+7)/8 bytes
16-bit transfers: COUNT = (MAX+15)/16 words
32-bit transfers: COUNT = (MAX+31)/32 dwords

www.DataSheet4U.com

www.DataSheet4U.com

5.4.3.3 Rectangle Fill Solid

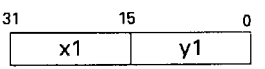
This command draws a solid rectangle with its top left corner at x1,y1, height = HEIGHT and width = WIDTH. The Pixel Control register (BEE8H, Index AH) must be set to A000H to select the Foreground Mix register to specify the color source and mix type. This example uses the mix NEW and color index 2. The drawing direction (bits 7-5 in the write to the CMD register below) is set to X positive, X major and Y positive (101b).

Setup:

```
ES:[FRGD_MIX] ⇐ 0027H          ; color source is FRGD_COLOR, NEW mix type
ES:[FRGD_COLOR] ⇐ 00000002H    ; color index
ES:[PIXEL_CNTL] ⇐ A000H        ; FRGD_MIX specifies the color source and mix type
```

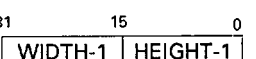
Drawing Operation:

ES:[ALT_CURXY] ⇐



; set starting coordinates

ES:[ALT_PCNT] ⇐



; rectangle width

```
ES:[CMD] ⇐ 0100000010110001b    ; Draw rectangle (bits 15-13, 11), draw (bit 4)
```

Note

The rectangle can be defined by specifying any one of the four corners and setting bits 7-5 accordingly. Always select X as the major axis (bit 6 =0). No matter how the rectangle is defined, it always fills from left to right and top to bottom.

Corner	X direction (bit 5)	Y direction (bit 7)
top left	positive (1)	positive (1)
top right	negative (0)	positive (1)
bottom left	positive (1)	negative (0)
bottom right	negative (0)	negative (0)

www.DataSheet4U.com

www.DataSheet4U.com

5.4.3.4 Image Transfer—Through the Plane

This command transfers a rectangular image from the CPU to the display memory through the plane. "through the plane" means the complete color index is transferred for each pixel, e.g., in 8 bits/pixel mode, one byte is required to transfer one pixel to memory. The image is stored as an array of pixels arranged in row major fashion (consecutively increasing memory addresses). The Pixel Control register must be set to A000H to select the Foreground Mix register to specify the color source and mix type. The color source must be specified as the CPU. Bit 12 of the Command register must be set to 1 (swap ON) for Intel-type architectures. Bit 8 of the Command register must be set to 1 (wait for CPU data) and bits 6 and 5 must also be set to 1 to specify X as the major axis and a left-to-right drawing direction. This example uses a mix type of NEW and x1,y1 is the top left corner of the rectangle on the screen. The height and width of the rectangle (in pixels) are HEIGHT and WIDTH. Doublword CPU writes are supported by setting bits 10-9 of the Command register to 10b.

Setup:

```
ES:[FRGD_MIX] ⇐ 0047H          ; color source is the CPU, mix type is NEW
ES:[PIXEL_CNTL] ⇐ A000H        ; FRGD_MIX is the source for color source and mix type
```

Drawing Operation:

```
                 31          15          0
ES:[ALT_CURXY] ⇐ |    x1     |    y1     |   ; set destination starting coordinates

                 31          15          0
ES:[ALT_PCNT] ⇐  | WIDTH-1   | HEIGHT-1  |   ; rectangle width
```

```
Wait for Graphics Engine not busy   ; loop till bit 9 of 9AE8H register is 0
ES:[CMD] ⇐ 01010101D0110001b        ; Draw rectangle (bits 15-13, 11), swap ON (bit 12),
                                    ; 32-bit transfers (bits 10-9), wait for CPU data (bit 8),
                                    ; always X Major (bit 6) & X Positive (bit 5), draw (bit 4)
```

COUNT (of image pixel data to transfer) = (See Note)
PIX_TRANS ⇐ IMAGEDATA; Output image data to the Pixel Data Transfer registers for COUNT dwords.

Note

The COUNT of the number of writes required by the CPU is a function of the number of pixels to be transferred, the width of the transfer (8, 16 or 32 bits as specified by bits 10-9 of the Drawing Command register (9AE8H)) and the color depth (bits/pixel). The number of pixels transferred per line must be an even multiple of the transfer width. If this is not the case, the last write per line must be padded with one or more dummy pixels to meet this requirement. For example, at 4 bits/pixel, each byte holds two pixels. If the transfer width is one byte and three pixels are to be transferred per line, two bytes must be written per line, with the upper nibble of the second byte a dummy pixel. If the transfer width is 16 bits, from one to three dummy pixels may be required to make the number of pixels per line an even multiple of 16. The number of word writes required per line can be determined from the formula n = (W+3)/4, with n being truncated to an integer if the result contains a fraction. Thus a six pixel transfer requires (6+3)/4 = 2.25 = 2 words. This is then multiplied by the height of the the image (in pixels) to determine the COUNT of words to be transferred. Similar procedures apply to every other combination of the variables affecting the COUNT. The formulas for all cases are given below, where W is the width of the image and H is the height of the image, both in pixels.

www.DataSheet4U.com

www.DataSheet4U.com

COUNT for 4 bits/pixel modes

8-bit transfers: COUNT = (W+1)/2 * H bytes
16-bit transfers: COUNT = (W+3)/4 * H words
32-bit transfers: COUNT = (W+7)/8 * H dwords

COUNT for 8 bits/pixel modes

8-bit transfers: COUNT = W * H bytes
16-bit transfers: COUNT = (W+1)/2 * H words
32-bit transfers: COUNT = (W+3)/4 * H dwords

COUNT for 16 bits/pixel modes

8-bit transfers: Do not use this combination
16-bit transfers: COUNT = W * H words
32-bit transfers: COUNT = (W+1)/2 * H dwords

COUNT for 32 bits/pixel modes

8-bit transfers: COUNT = Do not use this combination
16-bit transfers: COUNT = 2W * H words
32-bit transfers: COUNT = W * H dwords

Note that in 32 bits/pixel modes, the upper byte is a dummy byte providing padding for a 24-bit pixel.

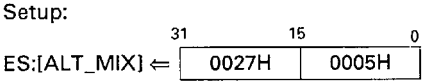
www.DataSheet4U.com

www.DataSheet4U.com

5.4.3.5 Image Transfer—Across the Plane

The image transfer command can also be used to transfer a rectangular image from the CPU to the display memory across the plane. "across the plane" means that each bit sent by the CPU is stored in display memory as a single pixel. These pixels are arranged in row major fashion (consecutively increasing memory addresses). An "across the plane" transfer is created by (1) setting bits 7-6 of the Pixel Control register (BEE8H, Index AH) to A080H to specify the CPU as the source of the mask bit selecting the mix register, (2) specifying a background and foreground color, (3) setting bit 8 of the Command register (9AE8H) to 1 (wait for CPU data) and (4) setting bit 1 of the Command register to 1 (multi-pixel). When the pattern bit sent by the CPU is a 1, the Foreground Mix register specifies the the color source and mix. When the bit is a 0, the Background Mix register specifies the color source and mix. This example uses a mix type of NEW, and x1,y1 is the top left corner of the rectangle on the screen. The height and width of the rectangle (in pixels) are HEIGHT and WIDTH. The monochrome image is translated so that pixels corresponding to a 1 in the bit image are given color index 4 and pixels corresponding to a 0 in the bit image are given color index 0. This example uses word transfers from the CPU as specified by setting bits 10-9 of the Command register to 01b for a 16-bit bus width.

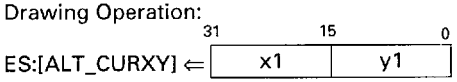
Setup:



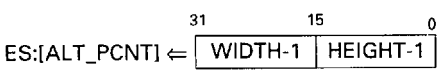
; FRGD_COLOR color source and mix is NEW
; BKGD_COLOR is color source and mix is XOR

```
ES:[FRGD_COLOR] ⇐ 00000004H     ; foreground color index 4
ES:[BKGD_COLOR] ⇐ 00000000H     ; background color index 0
ES:[PIXEL_CNTL] ⇐ A080H         ; selection of mix register is based on data from the CPU
```

Drawing Operation:



; set destination starting coordinates



; rectangle width

```
Wait for Graphics Engine not busy   ; loop till bit 9 of 9AE8H register is 0
CMD ⇐ 01010011D0110011b             ; Draw rectangle (bits 15-13, 11), swap ON (bit 12),
                                    ; 16-bit transfers (bits 10-9), wait for CPU data (bit 8),
                                    ; always X Major (bit 6) & X Positive (bit 5), draw (bit 4),
                                    ; multi-pixel (bit 1)
```

COUNT (of image pixel data to transfer) = ((WIDTH +15)/16)*HEIGHT words
PIX_TRANS ⇐ IMAGEDATA; Output image data to Pixel Transfer register for COUNT words

Notes

The COUNT of the number of writes required by the CPU is a function of the number of pixels to be transferred and the width of the transfer (8, 16 or 32 bits as specified by bits 10-9 of 9AE8H). Except for the case where bits 10-9 of 9AE8H are 11b, the number of pixels transferred per line must be an even multiple of the transfer width. If this is not the case, the last write per line must be padded with one or more dummy pixels to meet this requirement. For example, if the transfer width is 8 bits and nine pixels are to be transferred per line, two bytes must be written per line, with the upper 7 bits of the second byte padded. In general, the number of padding bits per line will vary from 0 to (n-1), where n is the transfer width in bits.

www.DataSheet4U.com

www.DataSheet4U.com

With a transfer width of 8 bits, the number of byte writes required per line can be determined from the formula n = (W+7)/8, with n being truncated to an integer if the result contains a fraction. Thus a 13-bit pixel transfer requires (13+7)/8 = 2.5 = 2 bytes. This is then multiplied by the height of the image (in pixels) to determine the COUNT of bytes to be transferred. Similar procedures apply to every other combination of the variables affecting the COUNT. The formulas for all cases are given below, where W is the width of the image and H is the height of the image, both in pixels.

8-bit transfers: COUNT = (W+7)/8 * H bytes (9AE8H_10-9 = 00b)
16-bit transfers: COUNT = (W+15)/16 * H words (9AE8H_10-9 = 01b)
32-bit transfers: COUNT = (W+31)/32 * H dwords (9AE8H_10-9 = 10b)
New 32-bit transfers: COUNT = (((W+7)/8*H)+3)/4 dwords (9AE8H_10-9 = 11b)

The differences between the two 32-bit transfer options are:

1. For 9AE8H_10-9 set to 10b, every line of the transfer must start with a fresh doubleword. In other words, all unneeded bits in a doubleword transfer for a given line are discarded. After a rectangular image is transferred, the current drawing position is a the bottom left, meaning the next rectangle, if drawn, will be below the previous rectangle.
2. For 9AE8H_10-9 set to 11b, only bits from the end of the line width to the next byte boundary are discarded. Data for the next line begins with the next byte. After a rectangular image is transferred, the current drawing position is a the top right, meaning the next rectangle, if drawn, will be to the right of the previous rectangle.

To write to a single plane, set the foreground mix to 'logical one' (0002H), the background mix to 'logical zero' (0001H), and the Write Mask register (AAE8H) to select the desired (single) plane for updates.

www.DataSheet4U.com

www.DataSheet4U.com

5.4.3.6 BitBLT—Through the Plane

This command copies a source rectangular area in display memory to another location in display memory. The Pixel Control register must be set to A000H to select the Foreground Mix register to specify the color source and mix type. The color source must be specified as the bitmap (display memory). Bit 6 of the Command register must be set to 1 to specify X as the major axis. For this example, assume x1,y1 is the top left corner of the source rectangle in display memory and x2,y2 is the top left corner of the destination rectangle. The rectangles can be overlapping or disjoint. The height and width (in pixels) of the rectangle being copied are HEIGHT and WIDTH.

Setup:

First, the values of the Srcx, Srcy, Destx and Desty must be determined.

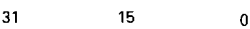
Case 1: Source and destination rectangles do not overlap

For X Positive, Y Positive: Srcx = x1, Srcy = y1, Destx = x2, Desty = y2

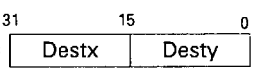
Case 2: Source and destination rectangles overlap

```
If x1 > x2
    then if X Positive, Srcx = x1, Destx = x2
else
    Srcx = x1 + WIDTH –1, Desty = x2 + WIDTH –1          ; X Negative

If y1 > y2
    then if Y Positive, Srcy = y1, Desty = x2
else
    Srcy = y1 + HEIGHT –1, Desty = y2 + HEIGHT –1        ; Y Negative

ES:[PIXEL_CNTL] ⇐ A000H          ; FRGD_MIX is the source of color source and mix type
ES:[FRGD_MIX] ⇐ 0067H            ; color source is display memory and mix type is NEW
```

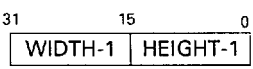
Draw Operation:

ES:[ALT_CURXY] ⇐ 

; set starting coordinates

ES:[ALT_STEP] ⇐ 

; set destination coordinates

ES:[ALT_PCNT] ⇐ 

; rectangle width and height

```
ES:[CMD] ⇐ 11000000D0D10001b     ; BitBLT (bits 15-13, 11), always X Major (bit 6) , draw (bit 4)
```

www.DataSheet4U.com

www.DataSheet4U.com

5.4.3.7 BitBLT—Across the Plane

This uses the same command as a BitBLT through the plane. However, instead of copying complete pixels (with color affected only by the mix), this 'across the plane" transfer uses only the bits in the color planes specified by setting the Read Mask register (AEE8H), e.g., bit 3 of every pixel, to determine the destination rectangle. With more than one plane enabled for read, if all the bits in the planes enabled for read are '1's then a '1' is read. If a bit in any one of the planes enabled for read is a '0', then '0' is read. An "across the plane" transfer is created by (1) setting bits 7-6 of the Pixel Control register (BEE8H, Index AH) to A0C0H to specify the bitmap as the source of the mask bit selecting the mix register, (2) programming the Read and Write Mask registers to specify the plane to read from and write to and (3) setting bit 1 of the Command register to 1 (multi-pixel). In this example, when the bit read is a 1, a 1 is copied as specified by the foreground mix. When the bit read is a 0, a 0 is copied as specified by the background mix. Assume x1,y1 is the top left corner of the source rectangle on the display, and x2,y2 is the top left corner of the destination rectangle. The image is read from plane 0 and written to plane 2. The rectangles could be overlapping or disjoint. The height and width (in pixels) of the rectangle are HEIGHT and WIDTH.

Setup:

First, the values of the Srcx, Srcy, Destx and Desty must be determined.

Case 1: Source and destination rectangles do not overlap

For X Positive, Y Positive: Srcx = x1, Srcy = y1, Destx = x2, Desty = y2

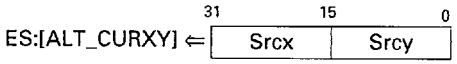
Case 2: Source and destination rectangles overlap

```
If x1 > x2
    then if X Positive, Srcx = x1, Destx = x2
else
    Srcx = x1 + WIDTH –1, Desty = x2 + WIDTH –1          ; X Negative

If y1 > y2
    then if Y Positive, Srcy = y1, Desty = y2
else
    Srcy = y1 + HEIGHT –1, Desty = y2 + HEIGHT –1        ; Y Negative

ES:[PIXEL_CNTL] ⇐ A0C0H          ; data from display memory selects mix register
                31          15          0
ES:[ALT_MIX] ⇐  | 0002H | 0001H |       ; result of foreground mix is always logical 1,
                                        ; result of background mix is always logical 0
ES:[RD_MASK] ⇐ 00000001H         ; read from plane 0
ES:[WRT_MASK] ⇐ 00000004H        ; plane 2 enabled for write
```

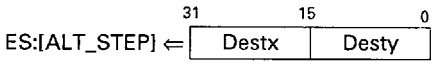
www.DataSheet4U.com

www.DataSheet4U.com

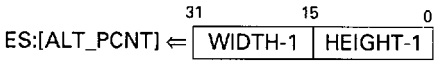
Draw Operation:



; set starting coordinates



; set destination coordinates



; rectangle width and height

```
ES:[CMD] ⇐ 11000000D0D10001b    ; BitBLT (bits 15-13, 11), always X Major (bit 6) , draw (bit 4),
                                ; multi-pixel (bit 1)
```

Note

It is possible to translate a monochrome image, e.g., text fonts, stored in a single plane in display memory into a 2-color image. This is accomplished by setting the mix registers differently and setting the desired background and foreground colors. If the source bit is a '1', then the corresponding pixel at the destination is colored with the foreground color index. The destination pixel is colored with the background color index if the corresponding source bit is a '0'. The setup for this is as follows:

```
ES:[WRT_MASK] ⇐ FFFFFFFFH       ; enable all planes for writing
ES:[FRGD_MIX] ⇐ 0027H           ; color source foreground, mix type NEW
ES:[BKGD_MIX] ⇐ 0007H           ; color source background, mix type NEW
ES:[FRGD_COLOR] ⇐ 00000004H     ; foreground color
ES:[BKGD_COLOR] ⇐ 00000001H     ; background color
```

www.DataSheet4U.com

www.DataSheet4U.com

5.4.3.8 PatBLT—Pattern Fill Through the Plane

An 8x8 pixel pattern is initially copied into an off-screen area of display memory using an image transfer operation or a direct write (linear addressing). This command then repeatedly tiles this source pattern into a destination rectangle of arbitrary size. The colors of the destination pixels are affected only by the mix selected. The destination rectangle must not overlap the source pattern. Each copy is aligned to an 8-pixel boundary (x coordinate = 0, 8, etc.), with pixels outside the destination rectangle boundary not being drawn. The Pixel Control register must be set to A000H to select the Foreground Mix register to specify the color source and mix type. The color source must be specified as the bitmap (display memory). Bit 6 of the Command register must be set to 1 to specify X as the major axis. In this example, assume x1,y1 is the top left corner of the pixel pattern and x2,y2 is the top left corner of the destination rectangle. The height and width (in pixels) of the rectangle are HEIGHT and WIDTH.

Setup:

```
ES:[PIXEL_CNTL] ⇐ A000H          ; FRGD_MIX is the source of color source and mix type
ES:[FRGD_MIX] ⇐ 0067H            ; color source is display memory, mix type is NEW
```

Draw Operation

```
                 31          15          0
ES:[ALT_CURXY] ⇐ |    x1     |    y1     |   ; set starting coordinates

                 31          15          0
ES:[ALT_STEP] ⇐  |    x2     |    y2     |   ; set destination coordinates

                 31          15          0
ES:[ALT_PCNT] ⇐  | WIDTH-1   | HEIGHT-1  |   ; rectangle width and height
```

```
ES:[CMD] ⇐ 11100000D0D10001b     ; PatBLT (bits 15-13,11), always X Major (bit 6) , draw (bit 4)
```

www.DataSheet4U.com

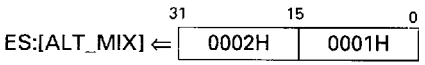
www.DataSheet4U.com

5.4.3.9 PatBLT—Pattern Fill Across the Plane

This uses the same command as a PatBLT through the plane. However, instead of copying complete pixels (with color affected only by the mix), this 'across the plane" transfer uses only the bits in the color planes specified by setting the Read Mask register (AEE8H), e.g., bit 3 of every pixel, to determine the destination rectangle. With more than one plane enabled for read, if all the bits in the planes enabled for read are '1's then a '1' is read. If a bit in any one of the planes enabled for read is a '0', then '0' is read. An "across the plane" transfer is created by (1) setting bits 7-6 of the Pixel Control register (BEE8H, Index AH) to A0C0H to specify the bitmap as the source of the mask bit selecting the mix register, (2) programming the Read and Write Mask registers to specify the plane to read from and write to and (3) setting bit 1 of the Command register to 1 (multi-pixel). In this example, when the bit read is a 1, a 1 is copied as specified by the foreground mix. When the bit read is a 0, a 0 is copied as specified by the background mix. In this example, assume x1,y1 is the top left corner of the pixel pattern and x2,y2 is the top left corner of the destination rectangle. The image is read from plane 0 and written to plane 2. The height and width of the destination rectangle are HEIGHT and WIDTH.

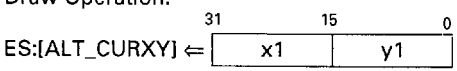
Setup:

```
ES:[PIXEL_CNTL] ⇐ A0C0H          ; data from display memory selects mix register
```

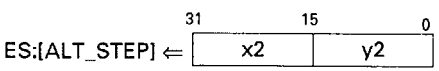



```
                                 ; result of foreground mix is always logical 1,
                                 ; result of background mix is always logical 0
ES:[RD_MASK] ⇐ 00000001H         ; read from plane 0
ES:[WRT_MASK] ⇐ 00000004H        ; plane 2 enabled for write
```

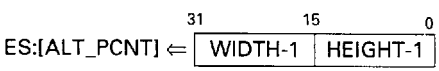
Draw Operation:



; set starting coordinates



; set destination coordinates



; rectangle width and height

```
ES:[CMD] ⇐ 11100000D0D10011b     ; PatBLT (bits 15-13, 11), always X Major (bit 6) , draw (bit 4),
                                 ; multi-pixel (bit 1)
```

Note

To expand the source mono pattern into a 2-color pattern, set the foreground mix to 27H, the background mix to 7H and the foreground and background colors as desired. Also set the write mask (AAE8H) to FFFFFFFFH. This needs to be set only once. It is altered only by another write.

www.DataSheet4U.com

www.DataSheet4U.com

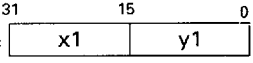
5.4.3.10 Short Stroke Vectors

This command rapidly draws short lines (up to 15 pixels in length). Such lines are constrained to one of the 8 directions at 45 degree increments starting at 0 degrees. The current point x1,y1 is set and a NOP command is issued to set all the desired drawing parameters without actually writing a pixel. For example, bit 2 (Last Pixel Off) would be set to 1 (OFF) for drawing connected lines until the last line is drawn. The short stroke vector parameters are then loaded in the Short Stroke Vector Transfer (9EE8H) register (SHORT_STROKE). Two vectors can be defined at a time, one in the low byte and one in the high byte. For the low byte, bits [7:5] define the direction, with bit 4 set to '1' for a draw operation or to '0' for a move current position operation. Bits 3-0 define the length of the short line. Let SSVD0, SSVD1, ...SSVDN–1 bytes be the short stroke vector data for N lines.

Setup:

```
ES:[PIXEL_CNTL] ⇐ A000H          ; FRGD_MIX is the source of color source and mix type
ES:[FRGD_MIX] ⇐ 0027H            ; use the foreground color, mix type NEW
ES:[FRGD_COLOR]⇐ 00000004H       ; foreground color index 4
```

Draw Operation:



; set starting coordinates

```
ES:[CMD] ⇐ 00010010XXX11111b     ; NOP (bits 15-13, 11), byte swap (bit 12), 16-bit transfers
                                 ; (bits 10-9) , draw (bit 4), radial drawing direction (bit 3),
                                 ; last pixel off (bit 2), multi-pixel (bit 1)
```

While space available in the FIFO

```
ES:[SHORT_STROKE] ⇐ SSVD1 SHL 8 + SSVD0 ; SSVD1 shifted to high byte, SSVD0 in low byte
ES:[SHORT_STROKE] ⇐ SSVD3 SHL 8 + SSVD2 ; byte swap turned on to read vectors out in
                                        ; correct order
        .
        .
        .
ES:[SHORT_STROKE] ⇐ SSVDN-1 SHL 8 + SSVDN-2
```

www.DataSheet4U.com

www.DataSheet4U.com

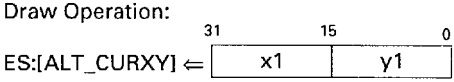
5.4.3.11 Polyline/2-Point Line

This command draws a line from point P1 (x1,y1) to point P2 (x2,y2). It can be used to draw an additional line from the end point of the last line drawn by specifying only the next end point. This can be repeated for as many polyline segments as desired. The Pixel Control register must be set to A000H to select the Foreground Mix register to specify the color source and mix type. The color source must be specified as the foreground color.

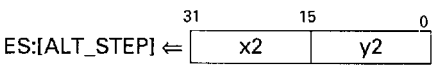
Setup:

```
ES:[FRGD_MIX] ⇐ 0027H              ; color source is FRGD_COLOR, NEW mix type
ES:[FRGD_COLOR]⇐ 00000002H         ; color index
ES:[PIXEL_CNTL] ⇐ A000H            ; FRGD_MIX specifies the color source and mix type
```

Draw Operation:



; set starting coordinates



; set destination coordinates

```
ES:[CMD]⇐ 0010100000010001b        ; draw 2-point line (bits 15-13, 11), draw (bit 4)
```

Repeat the last two instructions to draw additional polyline segments.

Note

This command is faster for drawing a 2-point line than the Solid Line drawing command. However, the Bresenham parameters are fixed in hardware and cannot be manipulated by the programmer as with the Solid Line command. A textured line cannot be drawn with this command.

www.DataSheet4U.com

www.DataSheet4U.com

5.4.3.12 Polygon Fill Solid

This command draws a polygon and fills it with a solid color. Any number of edges can be drawn, but the shape must be such that any horizontal line must intersect the polygon edges in no more than two places. To accomplish this, all edge segments must be drawn downward. The exception is that any edge can be horizontal. The Pixel Control register must be set to A000H to select the Foreground Mix register to specify the color source and mix type. This example uses the mix NEW and color index 2. This example shows how to draw the polygon shown in Figure 5-2.

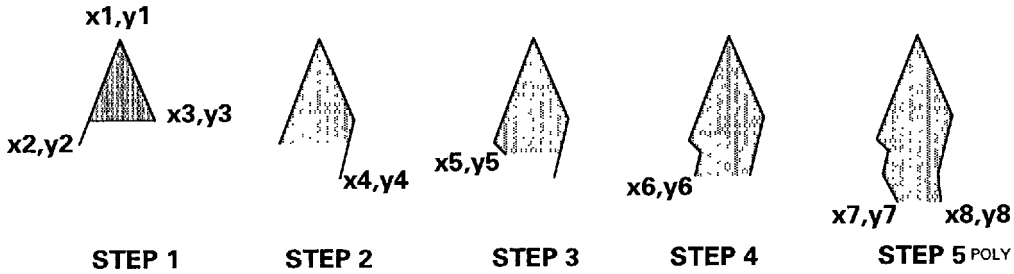


Figure 5-2. Polygon Example Drawing Steps

Setup:

```
ES:[FRGD_MIX] ⇐ 0027H          ; color source is FRGD_COLOR, NEW mix type
ES:[FRGD_COLOR] ⇐ 00000002H    ; color index
ES:[PIXEL_CNTL] ⇐ A000H        ; FRGD_MIX specifies the color source and mix type
```

Draw Operation:

```
ES:[ALT_CURXY] ⇐ | 31 ... 15 | 0 |
                 |     x1     |  y1  |   ; set starting coordinates for segment 1 of left side

ES:[ALT_STEP] ⇐ | 31 ... 15 | 0 |
                |     x2     |  y2  |    ; set destination coordinates for segment 1 of left side

ES:[ALT_CURXY2] ⇐ | 31 ... 15 | 0 |
                  |     x1     |  y1  |  ; set starting coordinates for segment 1 of right side

ES:[ALT_STEP2] ⇐ | 31 ... 15 | 0 |
                 |     x3     |  y3  |   ; set destination coordinates for segment 1 of right side

ES:[CMD]⇐ 0110000000010001b    ; draw polygon (bits 15-13, 11), draw (bit 4) - Step 1

ES:[ALT_STEP2] ⇐ | 31 ... 15 | 0 |
                 |     x4     |  y4  |   ; set destination coordinates for segment 2 of right side

ES:[CMD]⇐ 0110000000010001b    ; draw polygon (bits 15-13, 11) - Step 2

ES:[ALT_STEP] ⇐ | 31 ... 15 | 0 |
                |     x5     |  y5  |    ; set destination coordinates for segment 2 of left side
```

www.DataSheet4U.com

www.DataSheet4U.com

ES:[CMD]⇐ 0110000000010001b ; draw polygon (bits 15-13, 11) - Step 3

ES:[ALT_STEP] ⇐ | 31 x6 15 | y6 0 | ; set destination coordinates for segment 3 of left side

ES:[CMD]⇐ 0110000000010001b ; draw polygon (bits 15-13, 11) - Step 4

ES:[ALT_STEP] ⇐ | 31 x7 15 | y7 0 | ; set destination coordinates for segment 4 of left side

ES:[ALT_STEP2] ⇐ | 31 x8 15 | y8 0 | ; set destination coordinates for segment 3 of right side

ES:[CMD]⇐ 0110000000010001b ; draw polygon (bits 15-13, 11) - Step 4

The generation of this polygon is summarized in the following table.

Table 5-4. Polygon Fill Example Summary

	First Line				Second Line			
	ALT_CURXY		ALT_STEP		ALT_CURXY2		ALT_STEP2	
Step 1	x1	y1	x2	y2	x1	y1	x3	y3
CMD								
Step 2							x4	y4
CMD								
Step 3			x5	y5				
CMD								
Step 4			x6	y6				
CMD								
Step 5			x7	y7			x8	y8
CMD								

Notes

1. The current y for the first two line segments must be the same (y1 in the example). The current x for these two line segments can be the same (point) or different (horizontal top edge).
2. The fill proceeds downward until it reaches an end point for one of the edges. The next line segment is then drawn as an extension of this edge, with the fill again stopping at the first end point it reaches. For example, note how after segment 2 of the right side is drawn to x4,y4, the fill stops at x2,y2 of the first segment of the left side.
3. Segment 3 of the left side ends at the same vertical position as segment 2 of the right side (Step 4). When this occurs, both edges of the polygon must be extended by the next command. This is shown in Step 5.
4. For the step that closes the polygon (Step 5 in the example), the destination y positions must be be the same (y7 = y8 in the example) for the two line segments making the closure. As with the top edge, the x positions may be different (forming a horizontal bottom edge) or the same (a point).
5. When two lines join or cross at an angle other than 90 degrees, the common pixel will not be drawn.

www.DataSheet4U.com

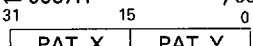
www.DataSheet4U.com

5.4.3.13 Polygon Fill Pattern

This command operates exactly as the polygon fill solid except that an 8x8 pixel pattern is tiled into the polygon instead of a solid color. The 8x8 pattern can either be color or mono and must first be programmed into off-screen memory using an image transfer or linear addressing. The colors of the destination pixels are affected only by the mix selected. The destination polygon must not overlap the source pattern. Each copy is aligned to an 8-pixel boundary (x coordinate = 0, 8, etc.), with pixels outside the destination polygon boundary not being drawn. For a color pattern. the Pixel Control register must be set to A000H to select the Foreground Mix register to specify the color source and mix type. The color source must be specified as the bitmap (display memory). For a mono pattern, the Pixel Control register must be set to A0C0H to select the bitmap as the source of the mask bit selecting the mix register. If the pattern bit is a 1, the Foreground Mix register is chosen, which must be programmed to select the foreground color. If the pattern bit is a 0, the Background Mix register is chosen, which must be programmed to select the background color.

Setup: (Color Pattern)

```
ES:[PIXEL_CNTL] ⇐ A000H          ; FRGD_MIX is the source of color source and mix type
ES:[FRGD_MIX] ⇐ 0067H            ; color source is display memory, mix type is NEW
```



; coordinates of upper left hand corner of 8x8 pattern

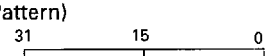
Draw Operation: (Color Pattern)

Same as for polygon fill solid except

```
ES:[CMD]⇐ 01101000000010001b     ; draw polygon with pattern fill (bits 15-13, 11)
```

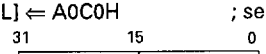
is substituted for each of the command lines.

Setup: (Mono Pattern)



; FRGD_COLOR color source and mix is NEW
; BKGD_COLOR is color source and mix is XOR

```
ES:[FRGD_COLOR] ⇐ 00000004H      ; foreground color index 4
ES:[BKGD_COLOR] ⇐ 00000000H      ; background color index 0
ES:[PIXEL_CNTL] ⇐ A0C0H          ; selection of mix register is based on data from the screen
```



; coordinates of upper left hand corner of 8x8 pattern

Draw Operation: (Mono Pattern)

Same as for polygon fill solid except

```
ES:[CMD]⇐ 01101000000010011b     ; draw polygon with pattern fill (bits 15-13, 11),
                                 ; multi-pixel (bit 1)
```

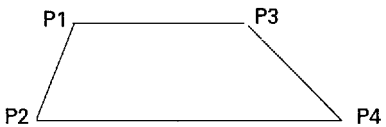
www.DataSheet4U.com

is substituted for each of the command lines.

www.DataSheet4U.com

5.4.3.14 4-Point Trapezoid Fill Solid

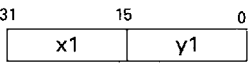
This command draws a solid trapezoid specified by edge 1 [points P1 (x1,y1) and P2 (x2,y2)] and edge 2 [points P3 (x3,y3) and P4 (x4,y4)]. P1 and P3 must be on the same horizontal line and P2 and P4 must be on a different lower horizontal line. The Pixel Control register must be set to A000H to select the Foreground Mix register to specify the color source and mix type. This example uses the mix NEW and color index 2.

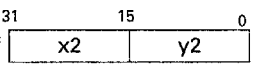


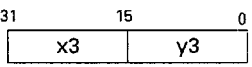
Setup:

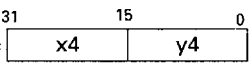
```
ES:[FRGD_MIX] ⇐ 0027H              ; color source is FRGD_COLOR, NEW mix type
ES:[FRGD_COLOR] ⇐ 00000002H        ; color index
ES:[PIXEL_CNTL] ⇐ A000H            ; FRGD_MIX specifies the color source and mix type
```

Draw Operation:

ES:[ALT_CURXY] ⇐  ; set starting coordinates for edge 1

ES:[ALT_STEP] ⇐  ; set destination coordinates for edge 1

ES:[ALT_CURXY2] ⇐  ; set starting coordinates for edge 2

ES:[ALT_STEP2] ⇐  ; set destination coordinates for edge 2


```
ES:[CMD]⇐ 1000000000010001b        ; draw 4-point trapezoid (bits 15-13, 11), draw (bit 4)
```

Note

The y coordinates for line 2 (y3 and y4) are not required. If they are programmed, they must be the same as y1 and y2 respectively.

www.DataSheet4U.com

www.DataSheet4U.com

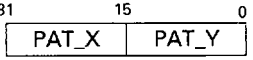
5.4.3.15 4-point Trapezoid Fill Pattern

This command operates exactly as the 4-point trapezoid fill solid except that an 8x8 pixel pattern is tiled into the trapezoid instead of a solid color. The 8x8 pattern can either be color or mono and must first be programmed into off-screen memory using an image transfer or linear addressing. The destination trapezoid must not overlap the source pattern. Each copy is aligned to an 8-pixel boundary (x coordinate = 0, 8, etc.), with pixels outside the destination trapezoid boundary not being drawn. For a color pattern. the Pixel Control register must be set to A000H to select the Foreground Mix register to specify the color source and mix type. The color source must be specified as the bitmap (display memory). For a mono pattern, the Pixel Control register must be set to A0C0H to select the bitmap as the source of the mask bit selecting the mix register. If the pattern bit is a 1, the Foreground Mix register is chosen, which must be programmed to select the foreground color. If the pattern bit is a 0, the Background Mix register is chosen, which must be programmed to select the background color.

Setup: (Color Pattern)

```
ES:[PIXEL_CNTL] ⇐ A000H          ; FRGD_MIX is the source of color source and mix type
ES:[FRGD_MIX] ⇐ 0067H            ; color source is display memory, mix type is NEW
```

ES:[ALT_PAT] ⇐



; coordinates of upper left hand corner of 8x8 pattern

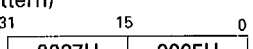
Draw Operation: (Color Pattern)

Same as for 4-point trapezoid fill solid except the command is as follows:

```
ES:[CMD]⇐ 0110100000010001b      ; draw trapezoid with pattern fill (bits 15-13, 11)
```

Setup: (Mono Pattern)

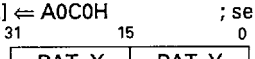
ES:[ALT_MIX] ⇐



; FRGD_COLOR color source and mix is NEW
; BKGD_COLOR is color source and mix is XOR

```
ES:[FRGD_COLOR] ⇐ 00000004H      ; foreground color index 4
ES:[BKGD_COLOR] ⇐ 00000000H      ; background color index 0
ES:[PIXEL_CNTL] ⇐ A0C0H          ; selection of mix register is based on data from the screen
```

ES:[ALT_PAT] ⇐



; coordinates of upper left hand corner of 8x8 pattern

Draw Operation: (Mono Pattern)

Same as for 4-point trapezoid fill solid except the command is as follows:

www.DataSheet4U.com

```
ES:[CMD]⇐ 1000100000010011b      ; draw trapezoid with pattern fill (bits 15-13, 11), multi-pixel (bit 1)
```

www.DataSheet4U.com

5.4.3.16 Bresenham Parameter Trapezoid Fill Solid

This command operates exactly as the 4-point trapezoid fill solid except the two edges are drawn with Bresenham parameters specified by the programmer. Calculation of these Bresenham parameters is based on the MAX and MIN parameters as calculated below.

MAX = maximum(ABS(x2–x1), ABS(y2–y1))
MIN = minimum(ABS(x2–x1), ABS(y2–y1))

where maximum means choose the largest of the two terms in parentheses and minimum means choose the smallest. ABS means take the absolute value of the expression. The fill must proceed downward (Y positive).

Setup:

```
ES:[FRGD_MIX] ⇐ 0027H          ; color source is FRGD_COLOR, NEW mix type
ES:[FRGD_COLOR] ⇐ 00000002H    ; color index
ES:[PIXEL_CNTL] ⇐ A000H        ; FRGD_MIX specifies the color source and mix type
```

Draw Operation

```
                                31          15          0
ES:[ALT_CURXY] ⇐                | x1        | y1        |   ; set starting coordinate for edge 1

ES:[MAJ_AXIS_PCNT] ⇐ MAX - 1    ; length in pixels of the major axis - 1 for edge 1

                                31          15          0
ES:[ALT_STEP] ⇐                 | 2*(MIN–MAX) | 2*MIN  |   ; diagonal and axial step constants for edge 1

If the X drawing direction is positive then
    ES:[ERR_TERM] ⇐ 2 * MIN – MAX          ; error term for edge 1
else if the X drawing direction is negative
    ES[ERR_TERM] ⇐ 2 * MIN – MAX – 1       ; error term for edge 1

                                31          15          0
ES:[ALT_CURXY2] ⇐               | x2        | y2        |   ; set starting coordinate for edge 2

ES:[MAJ_AXIS_PCNT2] ⇐ MAX - 1   ; length in pixels of the major axis - 1 for edge 2

                                31          15          0
ES:[ALT_STEP2] ⇐                | 2*(MIN–MAX) | 2*MIN  |   ; diagonal and axial step constants for edge 2

If the X drawing direction is positive then
    ES:[ERR_TERM] ⇐ 2 * MIN – MAX          ; error term for edge 2
else if the X drawing direction is negative
    ES[ERR_TERM] ⇐ 2 * MIN – MAX – 1       ; error term for edge 2

ES:[CMD2]⇐ 000000001DD00000b   ; edge 2 drawing direction (bits 7-5)
ES:[CMD]⇐ 101000001DD10001b    ; draw Bresenham parameter trapezoid (bits 15-13, 11)
                               ; edge 1 drawing direction (bits 7-5), draw (bit 4)
```

Note that the last two instructions can be packed, i.e, ES:[ALT_CMD], with CMD in the lower word (15-0) and CMD2 in the upper word (31-16). If they are not packed, CMD2 must precede CMD as shown.

www.DataSheet4U.com

www.DataSheet4U.com

5.4.3.17 Bresenham Parameter Trapezoid Fill Pattern

This command operates exactly as the Bresenham parameter trapezoid fill solid except that an 8x8 pixel pattern is tiled into the trapezoid instead of a solid color. The 8x8 pattern can either be color or mono and must first be programmed into off-screen memory using an image transfer or linear addressing. The destination trapezoid must not overlap the source pattern. Each copy is aligned to an 8-pixel boundary (x coordinate = 0, 8, etc.), with pixels outside the destination trapezoid boundary not being drawn. For a color pattern. the Pixel Control register must be set to A000H to select the Foreground Mix register to specify the color source and mix type. The color source must be specified as the bitmap (display memory). For a mono pattern, the Pixel Control register must be set to A0C0H to select the bitmap as the source of the mask bit selecting the mix register. If the pattern bit is a 1, the Foreground Mix register is chosen, which must be programmed to select the foreground color. If the pattern bit is a 0, the Background Mix register is chosen, which must be programmed to select the background color.

Setup: (Color Pattern)

```
ES:[PIXEL_CNTL] ⇐ A000H          ; FRGD_MIX is the source of color source and mix type
ES:[FRGD_MIX] ⇐ 0067H            ; color source is display memory, mix type is NEW
```

ES:[ALT_PAT] ⇐

31 — 15	0
PAT_X	PAT_Y

; coordinates of upper left hand corner of 8x8 pattern

Draw Operation: (Color Pattern)

Same as for Bresenham parameter trapezoid fill solid except the command is as follows:

```
ES:[CMD]⇐ 101010001DD10011b      ; draw Bresenham parameter trapezoid with pattern fill
                                 ; (bits 15-13, 11)
```

Setup: (Mono Pattern)

ES:[ALT_MIX] ⇐

31 — 15	0
0027H	0005H

; FRGD_COLOR color source and mix is NEW
; BKGD_COLOR is color source and mix is XOR

```
ES:[FRGD_COLOR] ⇐ 00000004H      ; foreground color index 4
ES:[BKGD_COLOR] ⇐ 00000000H      ; background color index 0
ES:[PIXEL_CNTL] ⇐ A0C0H          ; selection of mix register is based on data from the screen
```

ES:[ALT_PAT] ⇐

31 — 15	0
PAT_X	PAT_Y

; coordinates of upper left hand corner of 8x8 pattern

Draw Operation: (Mono Pattern)

Same as for Bresenham parameter trapezoid fill solid except the command is as follows:

www.DataSheet4U.com

```
ES:[CMD]⇐ 101010001DD10011b      ; draw Bresenham parameter trapezoid with pattern fill
                                 ; (bits 15-13, 11), multi-pixel (bit 1)
```

www.DataSheet4U.com

5.4.3.18 ROPBLTs

The ROPBLT function provides a full implementation of the 256 raster operations as defined by Microsoft for Windows. A listing and explanation of these is provided in Appendix A.

Each raster op has three operands: Source, Pattern and Destination. The Source pixel can be from the screen (current bitmap) or from the CPU (image transfer). When the source is the screen, the pixel depth is always the same for both the source and destination (4, 8, 16, 24/32 bits/pixel). When the source is the CPU, the pixel can be either color (same source and destination pixel depth) or mono (1 bit/pixel).

The Pattern is an 8x8 array of pixels located in off-screen memory. The pixels are either color or mono. If mono, pattern foreground and background registers define the pixel colors.

The Destination pixel is always the screen (current bitmap) and is always color (multi bits/pixel). This is the pixel that will be overwritten or left unchanged by the result of the operation.

Based on the above definitions, there are 6 valid ROPBLT cases:

Color Pattern

- Source = Screen, Color Pixels
- Source = CPU, Color Pixels
- Source = CPU, Mono Pixels

Mono Pattern

- Source = Screen, Color Pixels
- Source = CPU, Color Pixels
- Source = CPU, Mono Pixels

Programming examples for each of these cases are provided on the following pages.

www.DataSheet4U.com

www.DataSheet4U.com

Color Pattern Case 1 (Source = Screen, Color Pixels)

This command copies a source rectangular area in display memory to another location in display memory. The Pixel Control register must be set to A000H to select the Foreground Mix register to specify the color source. The color source must be specified as the bitmap (screen). The 8x8 pixel pattern must be copied to off-screen memory on an 8-pixel horizontal boundary and its location specified. Bit 6 of the Command register must be set to 1 to specify X as the major axis. For this example, assume x1,y1 is the top left corner of the source rectangle in display memory and x2,y2 is the top left corner of the destination rectangle. The rectangles can be overlapping or disjoint. The height and width (in pixels) of the rectangle being copied are HEIGHT and WIDTH.

Setup:

First, the values of the Srcx, Srcy, Destx and Desty must be determined.

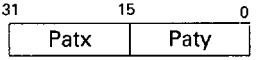
Case 1: Source and destination rectangles do not overlap

For X Positive, Y Positive: Srcx = x1, Srcy = y1, Destx = x2, Desty = y2

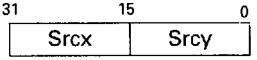
Case 2: Source and destination rectangles overlap

```
If x1 > x2
    then if X Positive, Srcx = x1, Destx = x2
else
    Srcx = x1 + WIDTH –1, Destx = x2 + WIDTH –1          ; X Negative

If y1 > y2
    then if Y Positive, Srcy = y1, Desty = y2
else
    Srcy = y1 + HEIGHT –1, Desty = y2 + HEIGHT –1        ; Y Negative

ES:[PIXEL_CNTL] ⇐ A000H          ; FRGD_MIX is the source of color source
ES:[FRGD_MIX] ⇐ 006XH            ; color source is screen, mix type is ignored
ES:[ROPMIX] ⇐ 00XXH              ; color pattern flag, XX = ROP code
```

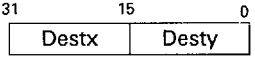
Draw Operation:

ES:[ALT_PAT] ⇐ 

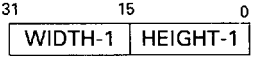
; set starting coordinates for pattern

ES:[ALT_CURXY] ⇐ 

; set starting coordinates

ES:[ALT_STEP] ⇐ 

; set destination coordinates

ES:[ALT_PCNT] ⇐ 

; rectangle width and height

ES:[CMD] ⇐ 11001000D0D10001b ; ROPBLT (bits 15-13, 11), always X Major (bit 6) , draw (bit 4)

www.DataSheet4U.com

www.DataSheet4U.com

Color Pattern Case 2 (Source = CPU, Color Pixels)

This command transfers a rectangular image from the CPU to the display memory through the plane. "through the plane" means the complete color index is transferred for each pixel, e.g., in 8 bits/pixel mode, one byte is required to transfer one pixel to memory. The image is stored as an array of pixels arranged in row major fashion (consecutively increasing memory addresses). The Pixel Control register must be set to A000H to select the Foreground Mix register to specify the color source. The color source must be specified as the CPU. ROP codes without a source operand must not be used. The 8x8 pixel pattern must be copied to off-screen memory on an 8-pixel horizontal boundary and its location specified. Bit 12 of the Command register must be set to 1 (swap ON) for Intel-type architectures. Bit 8 of the Command register must be set to 1 (wait for CPU data) and bits 6 and 5 must also be set to 1 to specify X as the major axis and a left-to-right drawing direction. For this example, x1,y1 is the top left corner of the rectangle on the screen. The height and width of the rectangle (in pixels) are HEIGHT and WIDTH. Doublword CPU writes are supported by setting bits 10-9 of the Command register to 10b.

Setup:

```
ES:[PIXEL_CNTL] ⇐ A000H          ; FRGD_MIX is the source for color source
ES:[FRGD_MIX] ⇐ 004XH            ; color source is the CPU, mix type is ignored
ES:[ROPMIX] ⇐ 00XXH              ; color pattern flag, XX = ROP code
```

Drawing Operation:

```
                 31          15          0
ES:[ALT_PAT] ⇐  | Patx      | Paty      |   ; set starting coordinates for pattern

                 31          15          0
ES:[ALT_STEP] ⇐ | x1         | y1        |   ; set destination starting coordinates

                 31          15          0
ES:[ALT_PCNT] ⇐ | WIDTH-1    | HEIGHT-1  |   ; rectangle width
```

```
Wait for Graphics Engine not busy   ; loop till bit 9 of 9AE8H register is 0
ES:[CMD] ⇐ 11001101D0110001b        ; ROPBLT (bits 15-13, 11), swap ON (bit 12),
                                    ; 32-bit transfers (bits 10-9), wait for CPU data (bit 8),
                                    ; always X Major (bit 6) & X Positive (bit 5), draw (bit 4)
```

COUNT (of image pixel data to transfer) = (See Note)
PIX_TRANS ⇐ IMAGEDATA; Output image data to the Pixel Data Transfer registers for COUNT dwords.

Note

The COUNT of the number of writes required by the CPU is a function of the number of pixels to be transferred, the width of the transfer (8, 16 or 32 bits as specified by bits 10-9 of the Drawing Command register (9AE8H)) and the color depth (bits/pixel). The number of pixels transferred per line must be an even multiple of the transfer width. If this is not the case, the last write per line must be padded with one or more dummy pixels to meet this requirement. For example, at 4 bits/pixel, each byte holds two pixels. If the transfer width is one byte and three pixels are to be transferred per line, two bytes must be written per line, with the upper nibble of the second byte a dummy pixel. If the transfer width is 16 bits, from one to three dummy pixels may be required to make the number of pixels per line an even multiple of 16. The number of word writes required per line can be determined from the formula n = (W+3)/4, with n being truncated to an integer if the result contains a fraction. Thus a six pixel transfer requires (6+3)/4 = 2.25 = 2 words. This is then multiplied by the height of the the image (in pixels) to determine the COUNT of words to be transferred. Similar procedures apply to every other combination

www.DataSheet4U.com

www.DataSheet4U.com

of the variables affecting the COUNT. The formulas for all cases are given below, where W is the width of the image and H is the height of the image, both in pixels.

COUNT for 4 bits/pixel modes

8-bit transfers: COUNT = (W+1)/2 * H bytes
16-bit transfers: COUNT = (W+3)/4 * H words
32-bit transfers: COUNT = (W+7)/8 * H dwords

COUNT for 8 bits/pixel modes

8-bit transfers: COUNT = W * H bytes
16-bit transfers: COUNT = (W+1)/2 * H words
32-bit transfers: COUNT = (W+3)/4 * H dwords

COUNT for 16 bits/pixel modes

8-bit transfers: Do not use this combination
16-bit transfers: COUNT = W * H words
32-bit transfers: COUNT = (W+1)/2 * H dwords

COUNT for 32 bits/pixel modes

8-bit transfers: COUNT = Do not use this combination
16-bit transfers: COUNT = 2W * H words
32-bit transfers: COUNT = W * H dwords

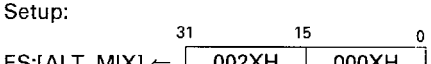
www.DataSheet4U.com

www.DataSheet4U.com

Color Pattern Case 3 (Source = CPU, Mono Pixels)

This command transfers a rectangular image from the CPU to the display memory across the plane. "across the plane" means that each bit sent by the CPU is stored in display memory as a single pixel. These pixels are arranged in row major fashion (consecutively increasing memory addresses). An "across the plane" transfer is created by (1) setting bits 7-6 of the Pixel Control register to A080H to specify the CPU as the source of the mask bit selecting the mix register, (2) specifying a background and foreground color, (3) setting bit 8 of the Command register (9AE8H) to 1 (wait for CPU data) and (4) setting bit 1 of the Command register to 1 (multi-pixel). ROP codes without a source operand must not be used. The 8x8 pixel pattern must be copied to off-screen memory on an 8-pixel horizontal boundary and its location specified. When the pattern bit sent by the CPU is a 1, the Foreground Mix register specifies the the color source and mix. When the bit is a 0, the Background Mix register specifies the color source and mix. For this example, x1,y1 is the top left corner of the rectangle on the screen. The height and width of the rectangle (in pixels) are HEIGHT and WIDTH. The monochrome image is translated so that pixels corresponding to a 1 in the bit image are given color index 4 and pixels corresponding to a 0 in the bit image are given color index 0. This example uses word transfers from the CPU as specified by setting bits 10-9 of the Command register to 01b for a 16-bit bus width.

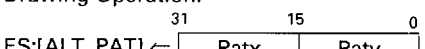
Setup:



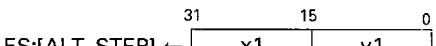
; FRGD_COLOR is color source
; BKGD_COLOR is color source

```
ES:[FRGD_COLOR] ⇐ 00000004H     ; foreground color index 4
ES:[BKGD_COLOR] ⇐ 00000000H     ; background color index 0
ES:[PIXEL_CNTL] ⇐ A080H         ; selection of mix register is based on data from the CPU
ES:[ROPMIX] ⇐ 00XXH              ; color pattern flag, XX = ROP code
```

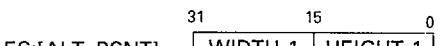
Drawing Operation:



; set starting coordinates for pattern



; set destination starting coordinates



; rectangle width

```
Wait for Graphics Engine not busy   ; loop till bit 9 of 9AE8H register is 0
CMD ⇐ 11001011D0110011b             ; ROPBLT (bits 15-13, 11), swap ON (bit 12),
                                    ; 16-bit transfers (bits 10-9), wait for CPU data (bit 8),
                                    ; always X Major (bit 6) & X Positive (bit 5), draw (bit 4),
                                    ; multi-pixel (bit 1)
```

COUNT (of image pixel data to transfer) = ((WIDTH +15)/16)*HEIGHT words
PIX_TRANS ⇐ IMAGEDATA; Output image data to Pixel Transfer register for COUNT words

Notes

The COUNT of the number of writes required by the CPU is a function of the number of pixels to be transferred and the width of the transfer (8, 16 or 32 bits as specified by bits 10-9 of 9AE8H). Except for the case where bits 10-9 of 9AE8H are 11b, the number of pixels transferred per line must be an even multiple of the transfer width. If this is not the case, the last write per line must be padded with one or more dummy pixels to meet this requirement. For example, if the transfer width is 8 bits and nine

www.DataSheet4U.com

www.DataSheet4U.com

pixels are to be transferred per line, two bytes must be written per line, with the upper 7 bits of the second byte padded. In general, the number of padding bits per line will vary from 0 to (n-1), where n is the transfer width in bits.

With a transfer width of 8 bits, the number of byte writes required per line can be determined from the formula n = (W+7)/8, with n being truncated to an integer if the result contains a fraction. Thus a 13-bit pixel transfer requires (13+7)/8 = 2.5 = 2 bytes. This is then multiplied by the height of the image (in pixels) to determine the COUNT of bytes to be transferred. Similar procedures apply to every other combination of the variables affecting the COUNT. The formulas for all cases are given below, where W is the width of the image and H is the height of the image, both in pixels.

8-bit transfers: COUNT = (W+7)/8 * H bytes (9AE8H_10-9 = 00b)
16-bit transfers: COUNT = (W+15)/16 * H words (9AE8H_10-9 = 01b)
32-bit transfers: COUNT = (W+31)/32 * H dwords (9AE8H_10-9 = 10b)
New 32-bit transfers: COUNT = (((W+7)/8*H)+3)/4 dwords (9AE8H_10-9 = 11b)

The differences between the two 32-bit transfer options are:

1. For 9AE8H_10-9 set to 10b, every line of the transfer must start with a fresh doubleword. In other words, all unneeded bits in a doubleword transfer for a given line are discarded. After a rectangular image is transferred, the current drawing position is a the bottom left, meaning the next rectangle, if drawn, will be below the previous rectangle.
2. For 9AE8H_10-9 set to 11b, only bits from the end of the line width to the next byte boundary are discarded. Data for the next line begins with the next byte. After a rectangular image is transferred, the current drawing position is a the top right, meaning the next rectangle, if drawn, will be to the right of the previous rectangle.

To write to a single plane, set the foreground mix to 'logical one' (0002H), the background mix to 'logical zero' (0001H), and the Write Mask register (AAE8H) to select the desired (single) plane for updates.

www.DataSheet4U.com

www.DataSheet4U.com

Mono Pattern Case 1 (Source = Screen, Color Pixels)

This command copies a source rectangular area in display memory to another location in display memory. It is identical to the Color Pattern Case 1 except that the Pattern Foreground and Background Colors registers are programmed, the Bitplane Read Mask register is programmed to select the same bitplane enabled by the Bitplane Write Mask register when the pattern is written to memory and the mono pattern flag is set in the ROPMIX register. The Pixel Control register must be set to A000H to select the Foreground Mix register to specify the color source. The color source must be specified as the bitmap (screen). The 8x8 mono pattern must be copied to off-screen memory on an 8-bit horizontal boundary and its location specified. Bit 6 of the Command register must be set to 1 to specify X as the major axis. For this example, assume x1,y1 is the top left corner of the source rectangle in display memory and x2,y2 is the top left corner of the destination rectangle. The rectangles can be overlapping or disjoint. The height and width (in pixels) of the rectangle being copied are HEIGHT and WIDTH.

Setup:

First, the values of the Srcx, Srcy, Destx and Desty must be determined.

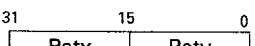
Case 1: Source and destination rectangles do not overlap

For X Positive, Y Positive: Srcx = x1, Srcy = y1, Destx = x2, Desty = y2

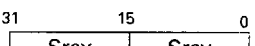
Case 2: Source and destination rectangles overlap

```
If x1 > x2
    then if X Positive, Srcx = x1, Destx = x2
else
    Srcx = x1 + WIDTH –1, Destx = x2 + WIDTH –1          ; X Negative

If y1 > y2
    then if Y Positive, Srcy = y1, Desty = y2
else
    Srcy = y1 + HEIGHT –1, Desty = y2 + HEIGHT –1        ; Y Negative

ES:[PIXEL_CNTL] ⇐ A000H          ; FRGD_MIX is the source of color source
ES:[FRGD_MIX] ⇐ 006XH            ; color source is screen, mix type is ignored
ES:[ROPMIX] ⇐ 01XXH              ; mono pattern flag, XX = ROP code
ES:[PAT_FG_COLOR] ⇐ 00000004H    ; foreground color index 4
ES:[PAT_BG_COLOR] ⇐ 00000000H    ; background color index 0
ES:[RD_MASK] ⇐ 00000001H         ; bitplane 0 enabled for reading
```

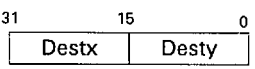
Draw Operation:

ES:[ALT_PAT] ⇐ 

; set starting coordinates for pattern

ES:[ALT_CURXY] ⇐ 

; set starting coordinates

ES:[ALT_STEP] ⇐ 

; set destination coordinates

www.DataSheet4U.com

www.DataSheet4U.com

ES:[ALT_PCNT] ⇐

31 — 16	15 — 0
WIDTH-1	HEIGHT-1

; rectangle width and height

ES:[CMD] ⇐ 11001000D0D10001b ; ROPBLT (bits 15-13, 11), always X Major (bit 6) , draw (bit 4)

www.DataSheet4U.com

www.DataSheet4U.com

Mono Pattern Case 2 (Source = CPU, Color Pixels)

This command transfers a rectangular image from the CPU to the display memory through the plane. It is identical to the Color Pattern Case 2 described earlier except that the Pattern Foreground and Background Colors registers are programmed, the Bitplane Read Mask register is programmed to select the same bitplane enabled by the Bitplane Write Mask register when the pattern is written to memory and the mono pattern flag is set in the ROPMIX register. "through the plane" means the complete color index is transferred for each pixel, e.g., in 8 bits/pixel mode, one byte is required to transfer one pixel to memory. The image is stored as an array of pixels arranged in row major fashion (consecutively increasing memory addresses). The Pixel Control register must be set to A000H to select the Foreground Mix register to specify the color source. The color source must be specified as the CPU. ROP codes without a source operand must not be used. The 8x8 mono pattern must be copied to off-screen memory on an 8-bit horizontal boundary and its location specified. Bit 12 of the Command register must be set to 1 (swap ON) for Intel-type architectures. Bit 8 of the Command register must be set to 1 (wait for CPU data) and bits 6 and 5 must also be set to 1 to specify X as the major axis and a left-to-right drawing direction. For this example, x1,y1 is the top left corner of the rectangle on the screen. The height and width of the rectangle (in pixels) are HEIGHT and WIDTH. Doublword CPU writes are supported by setting bits 10-9 of the Command register to 10b.

Setup:

```
ES:[PIXEL_CNTL] ⇐ A000H              ; FRGD_MIX is the source for color source
ES:[FRGD_MIX] ⇐ 004XH                ; color source is the CPU, mix type is ignored
ES:[ROPMIX] ⇐ 01XXH                  ; mono pattern flag, XX = ROP code
ES:[PAT_FG_COLOR] ⇐ 00000004H        ; foreground color index 4
ES:[PAT_BG_COLOR] ⇐ 00000000H        ; background color index 0
ES:[RD_MASK] ⇐ 00000001H             ; bitplane 0 enabled for reading
```

Drawing Operation:

```
                 31          15          0
ES:[ALT_PAT] ⇐   | Patx      | Paty      |     ; set starting coordinates for pattern

                 31          15          0
ES:[ALT_STEP] ⇐  | x1        | y1        |     ; set destination starting coordinates

                 31          15          0
ES:[ALT_PCNT] ⇐  | WIDTH-1   | HEIGHT-1  |     ; rectangle width
```

```
Wait for Graphics Engine not busy   ; loop till bit 9 of 9AE8H register is 0
ES:[CMD] ⇐ 11001101D0110001b        ; ROPBLT (bits 15-13, 11), swap ON (bit 12),
                                    ; 32-bit transfers (bits 10-9), wait for CPU data (bit 8),
                                    ; always X Major (bit 6) & X Positive (bit 5), draw (bit 4)
```

COUNT (of image pixel data to transfer) = (See Note)
PIX_TRANS ⇐ IMAGEDATA; Output image data to the Pixel Data Transfer registers for COUNT dwords.

Note

The COUNT of the number of writes required by the CPU is a function of the number of pixels to be transferred, the width of the transfer (8, 16 or 32 bits as specified by bits 10-9 of the Drawing Command register (9AE8H)) and the color depth (bits/pixel). The number of pixels transferred per line must be an even multiple of the transfer width. If this is not the case, the last write per line must be padded with one or more dummy pixels to meet this requirement. For example, at 4 bits/pixel, each byte holds two

www.DataSheet4U.com

www.DataSheet4U.com

pixels. If the transfer width is one byte and three pixels are to be transferred per line, two bytes must be written per line, with the upper nibble of the second byte a dummy pixel. If the transfer width is 16 bits, from one to three dummy pixels may be required to make the number of pixels per line an even multiple of 16. The number of word writes required per line can be determined from the formula n = (W+3)/4, with n being truncated to an integer if the result contains a fraction. Thus a six pixel transfer requires (6+3)/4 = 2.25 = 2 words. This is then multiplied by the height of the the image (in pixels) to determine the COUNT of words to be transferred. Similar procedures apply to every other combination of the variables affecting the COUNT. The formulas for all cases are given below, where W is the width of the image and H is the height of the image, both in pixels.

COUNT for 4 bits/pixel modes

8-bit transfers: COUNT = (W+1)/2 * H bytes
16-bit transfers: COUNT = (W+3)/4 * H words
32-bit transfers: COUNT = (W+7)/8 * H dwords

COUNT for 8 bits/pixel modes

8-bit transfers: COUNT = W * H bytes
16-bit transfers: COUNT = (W+1)/2 * H words
32-bit transfers: COUNT = (W+3)/4 * H dwords

COUNT for 16 bits/pixel modes

8-bit transfers: Do not use this combination
16-bit transfers: COUNT = W * H words
32-bit transfers: COUNT = (W+1)/2 * H dwords

COUNT for 32 bits/pixel modes

8-bit transfers: COUNT = Do not use this combination
16-bit transfers: COUNT = 2W * H words
32-bit transfers: COUNT = W * H dwords

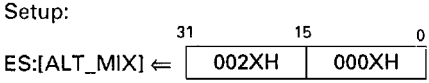
www.DataSheet4U.com

www.DataSheet4U.com

Mono Pattern Case 3 (Source = CPU, Mono Pixels)

This command transfers a rectangular image from the CPU to the display memory across the plane. It is identical to the Color Pattern Case 3 described earlier except that the Pattern Foreground and Background Colors registers are programmed, the Bitplane Read Mask register is programmed to select the same bitplane enabled by the Bitplane Write Mask register when the pattern is written to memory and the mono pattern flag is set in the ROPMIX register. "across the plane" means that each bit sent by the CPU is stored in display memory as a single pixel. These pixels are arranged in row major fashion (consecutively increasing memory addresses). An "across the plane" transfer is created by (1) setting bits 7-6 of the Pixel Control register to A080H to specify the CPU as the source of the mask bit selecting the mix register, (2) specifying a background and foreground color, (3) setting bit 8 of the Command register (9AE8H) to 1 (wait for CPU data) and (4) setting bit 1 of the Command register to 1 (multi-pixel). ROP codes without a source operand must not be used. The 8x8 mono pattern must be copied to off-screen memory on an 8-bit horizontal boundary and its location specified. When the pattern bit sent by the CPU is a 1, the Foreground Mix register specifies the the color source and mix. When the bit is a 0, the Background Mix register specifies the color source and mix. For this example, x1,y1 is the top left corner of the rectangle on the screen. The height and width of the rectangle (in pixels) are HEIGHT and WIDTH. The monochrome image is translated so that pixels corresponding to a 1 in the bit image are given color index 4 and pixels corresponding to a 0 in the bit image are given color index 0. This example uses word transfers from the CPU as specified by setting bits 10-9 of the Command register to 01b for a 16-bit bus width.

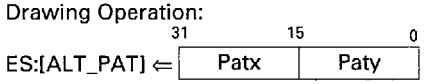
Setup:



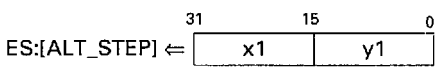
; FRGD_COLOR is color source
; BKGD_COLOR is color source

```
ES:[FRGD_COLOR] ⇐ 00000004H     ; foreground color index 4
ES:[BKGD_COLOR] ⇐ 00000000H     ; background color index 0
ES:[PIXEL_CNTL] ⇐ A080H         ; selection of mix register is based on data from the CPU
ES:[ROPMIX] ⇐ 01XXH             ; mono pattern flag, XX = ROP code
ES:[PAT_FG_COLOR] ⇐ 00000004H   ; foreground color index 4
ES:[PAT_BG_COLOR] ⇐ 00000000H   ; background color index 0
ES:[RD_MASK] ⇐ 00000001H        ; bitplane 0 enabled for reading
```

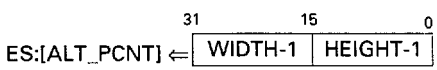
Drawing Operation:



; set starting coordinates for pattern



; set destination starting coordinates



; rectangle width

```
Wait for Graphics Engine not busy   ; loop till bit 9 of 9AE8H register is 0
CMD ⇐ 11001011D0110011b             ; ROPBLT (bits 15-13, 11), swap ON (bit 12),
                                    ; 16-bit transfers (bits 10-9), wait for CPU data (bit 8),
                                    ; always X Major (bit 6) & X Positive (bit 5), draw (bit 4),
                                    ; multi-pixel (bit 1)
```

COUNT (of image pixel data to transfer) = ((WIDTH +15)/16)*HEIGHT words

www.DataSheet4U.com

PIX_TRANS ⇐ IMAGEDATA; Output image data to Pixel Transfer register for COUNT words

www.DataSheet4U.com

Notes

The COUNT of the number of writes required by the CPU is a function of the number of pixels to be transferred and the width of the transfer (8, 16 or 32 bits as specified by bits 10-9 of 9AE8H). Except for the case where bits 10-9 of 9AE8H are 11b, the number of pixels transferred per line must be an even multiple of the transfer width. If this is not the case, the last write per line must be padded with one or more dummy pixels to meet this requirement. For example, if the transfer width is 8 bits and nine pixels are to be transferred per line, two bytes must be written per line, with the upper 7 bits of the second byte padded. In general, the number of padding bits per line will vary from 0 to (n-1), where n is the transfer width in bits.

With a transfer width of 8 bits, the number of byte writes required per line can be determined from the formula n = (W+7)/8, with n being truncated to an integer if the result contains a fraction. Thus a 13-bit pixel transfer requires (13+7)/8 = 2.5 = 2 bytes. This is then multiplied by the height of the image (in pixels) to determine the COUNT of bytes to be transferred. Similar procedures apply to every other combination of the variables affecting the COUNT. The formulas for all cases are given below, where W is the width of the image and H is the height of the image, both in pixels.

8-bit transfers: COUNT = (W+7)/8 * H bytes (9AE8H_10-9 = 00b)
16-bit transfers: COUNT = (W+15)/16 * H words (9AE8H_10-9 = 01b)
32-bit transfers: COUNT = (W+31)/32 * H dwords (9AE8H_10-9 = 10b)
New 32-bit transfers: COUNT = (((W+7)/8*H)+3)/4 dwords (9AE8H_10-9 = 11b)

The differences between the two 32-bit transfer options are:

1. For 9AE8H_10-9 set to 10b, every line of the transfer must start with a fresh doubleword. In other words, all unneeded bits in a doubleword transfer for a given line are discarded. After a rectangular image is transferred, the current drawing position is a the bottom left, meaning the next rectangle, if drawn, will be below the previous rectangle.
2. For 9AE8H_10-9 set to 11b, only bits from the end of the line width to the next byte boundary are discarded. Data for the next line begins with the next byte. After a rectangular image is transferred, the current drawing position is a the top right, meaning the next rectangle, if drawn, will be to the right of the previous rectangle.

To write to a single plane, set the foreground mix to 'logical one' (0002H), the background mix to 'logical zero' (0001H), and the Write Mask register (AAE8H) to select the desired (single) plane for updates.

www.DataSheet4U.com

www.DataSheet4U.com

5.4.3.19 Programmable Hardware Cursor

A programmable cursor is supported which is compatible with the Microsoft Windows (bit 4 of CR55 = 0) and X11 (bit 4 of CR55 = 1) cursor definitions. The cursor size is 64 pixels wide by 64 pixels high, with the cursor pattern stored in an off-screen area of display memory. Two monochrome images 64 bits wide by 64 bits high (512 bytes per image) define the cursor shape. The first bit image is an AND mask and the second bit image is an XOR mask. The following is the truth table for the cursor display logic.

AND Bit	XOR Bit	Displayed (Microsoft Windows)	Displayed (X11)
0	0	Cursor Background Color	Current Screen Pixel
0	1	Cursor Foreground Color	Current Screen Pixel
1	0	Current Screen Pixel	Cursor Background Color
1	1	NOT Current Screen Pixel	Cursor Foreground Color

The hardware cursor color is taken from the Hardware Graphics Cursor Foreground Stack (CR4A) and the Hardware Graphics Cursor Background Stack (CR4B) registers. Each of these is a stack of three 8-bit registers. The stack pointers are reset to 0 by reading the Hardware Graphics Cursor Mode register (CR45). The color value is then programmed by consecutive writes (low byte, second byte, third byte) to the appropriate (foreground or background) register.

Enabling/Disabling the Cursor

The hardware cursor is disabled when a VGA-compatible mode is in use. It can be enabled or disabled when in Enhanced mode (bit 0 of 4AE8H = 1), as follows.

```
CR39 ⇐ A0H          ; Unlock System Control registers
CR45_0 ⇐ 1          ; Enable hardware cursor
CR45_0 ⇐ 0          ; Disable hardware cursor
CR39 ⇐ 00H          ; Lock System Control registers
```

Positioning the Cursor

The cursor can be positioned at any point on the display, with the X,Y coordinates ranging from 0 to 2047. This enables the full cursor images to be displayed on the screen and partial cursor images to be displayed at the right edge and the bottom edge of the screen. The cursor offset OX,OY has to be set to 0,0 for a 1024x768 resolution. If X is > (1024 – 64) or Y is > (768 – 64), then a partial cursor is visible at the right edge or top edge of the screen respectively. Note that if Y ≥ 768 then the cursor is not visible; it is residing in the off-screen area.

A partial cursor image can be displayed at the left edge or the top edge of the screen. To enable partial cursor display at the top edge of the screen, Y is set to 0 and the Y offset register is set to OY (range from 0 to 63). This displays the bottom 64–OY rows of the cursor image at the currently set X position and the top edge of the screen. Similarly, a partial cursor can be displayed at the left edge of the screen by setting X to 0 and the X offset register to OX (range from 0 to 63). This displays the right 64–OX columns of the cursor image at the currently set X and the left edge of the screen. The following pseudocode illustrates cursor positioning.

```
CR39 ⇐ A0H                          ; Unlock System Control registers
CR46_10-8 ⇐ MS 3 bits of X cursor position
CR47_7-0 ⇐ LS 8 bits of X cursor position
```

www.DataSheet4U.com

www.DataSheet4U.com

```
CR49_7-0 ⇐ LS 8 bits of Y cursor position
CR4E_5-0 ⇐ Cursor Offset X position
CR4F_5-0 ⇐ Cursor Offset Y position
CR48_10-8 ⇐ MS 3 bits of Y cursor position
CR39 ⇐ 00H                              ; Lock System Control registers
```

The cursor position is updated by the hardware once each frame. Therefore, the programmer should ensure that the position is re-programmed no more than once for each vertical sync period.

Programming the Cursor Shape

The AND and the XOR cursor image bitmaps are 512 bytes each. These are stored in consecutive bytes of off-screen display memory, 512 AND bytes followed by 512 XOR bytes. The starting location must be on a 1024-byte boundary. This location is programmed into the Hardware Graphics Cursor Start Address registers (CR4C and CR4D) as follows:

```
CR39 ⇐ A0H                              ; Unlock System Control registers
CR4C_5-8 ⇐ MS 4 bits of the cursor storage start 1024-byte segment.
CR4D ⇐ LS 8 bits of the cursor storage start 1024-byte segment
CR39 ⇐ 0                                ; Lock System Control registers
```

The value programmed is the 1024-byte segment of display memory at which the beginning of the hardware cursor bit pattern is located. For example, for an 800x600x8 mode on a 1 MByte system, there are 1024 1K segments. Programming CR4C_11-8 with 3H and CR4D with FEH specifies the starting location as the 1022nd (0-based) 1K segment. The cursor pattern is programmed (using linear addressing) at FF800H offset from the base address of the frame buffer.

Note

If the cursor is not 64 bits by 64 bits, the given images should be padded to make the cursor image 64 bits by 64 bits. The padded area should be made transparent by padding the extra AND mask bits with '1's and the extra XOR bits by '0's.

5.5 RECOMMENDED READING

Graphics Programming for the 8514/A by Jake Richter and Bud Smith (M&T Publishing, Inc) provides extensive explanations and examples for programming most of the bits in the S3 Enhanced Registers.

Although not released at the time this data book was printed, the 3rd edition of *Programming Guide to the EGA and VGA Cards* by Richard F. Ferraro (Addison-Wesley Publishing Company, Inc) is scheduled to include a section on programming for S3 accelerator chips.

www.DataSheet4U.com

www.DataSheet4U.com

Section 6: Video Engine

This section describes the Video Engine provided by the Vision868. The Video Engine registers are described in Section 2.

6.1 VIDEO ENGINE OVERVIEW

The Video Engine obtains pixel data either from video memory or the CPU, operates on it and then writes to display memory. Particular pixels in memory can be masked off from being updated by the video data. The operations the Video Engine can perform are scaling (both horizontal and vertical), color space conversion (YUV to RGB) and dithering (reduce color depth from 24 bits/pixel to 16, 15 or 8 bits/pixel). The legal input output combinations are listed in Table 6-1. The input and output bit settings are for 8088H_22-20 and 8088H_18-16 respectively.

The Video Engine processes one pixel per MCLK regardless of the pixel depth. Input and output FIFOs allow burst/block transfers from and to memory to improve throughput.

New MMIO operation (bits 4-3 of CR53 set to 11b) is required for Video Engine access. If big endian addressing is required, bits 2-1 of CR53 control byte swapping.

The Video Engine cannot be used when the Vision868 is being operated in packed 24 bits/pixel mode (bits 5-4 of CR50 set to 10b). It does not take advantage of 1-cycle EDO or burst mode memory operation if one of these is specified (bits 3-2 of

Table 6-1 Video Engine Input/Output Combinations

		Output	000	011	100	101	110	111
	Width		8	32	16	16	16	16
Input		Format	RGB332	RGB888	YC/ YUV442	raw	RGB555	RGB565
000	8	RGB332	S	No	No	No	No	No
011	32	RGB888	S/D	S	No	No	S/D	S/D
100	16	YC/YUV 422	S/C/D	S/C	S	No	S/C/D	S/C/D
101	16	raw	No	No	No	S	No	No
110	16	RGB555	S/D	S	No	No	S	S
111	16	RGB565	S/D	S	No	No	S	S

S = Scaling
C = Color Space Conversion
D = Dithering

www.DataSheet4U.com

www.DataSheet4U.com

CR36 set to 10b or 11b. Instead, it will use standard fast page or EDO cycles.

The Video Engine and the Graphics Engine must not be in use at the same time. To ensure this, the Video Engine NOP register (8080H) must be written to after all Video Engine data/commands have been written. Similarly, a Graphics Engine NOP must be executed after a Graphics Engine access before the Video Engine can be written to again.

6.2 SCALING

Bit 31 of 808CH is cleared to 0 to specify a stretch and set to 1 to specify a shrink. Bit 30 of 808CH is cleared to 0 to specify display memory as the data source and set to 1 to specify the CPU as the data source. If display memory is the data source, the starting address for the data is specified in 8098H. The amount of stretch or shrink is specified by programming source and destination steps in 8090H and also programming the following DDA parameters.

The internal digital differential analyzer (DDA) is programmed by specifying the DDA accumulator initial value in bits 11-0 of 8088H and the K1 and K2 constants via bits 10-0 and 26-16 of 808CH.

The input data format is specified via bits 22-20 of 8088H. If 100b is specified, bit 19 of 8088H is cleared to 0 to specify YCbCr and set to 1 to specify YUV.

A single pass through the Video Engine produces a one-dimensional stretch or shrink (horizontal or vertical). This can be written directly to on-screen memory. A two-dimensional scaling requires two passes. The first pass writes the one-dimensionally scaled data to off-screen memory. The second pass BitBLTs the off-screen image to on-screen memory while performing the scaling in the other dimension. Both the amount of scaling and the dimension (horizontal or vertical) are controlled by the step constants programmed into 8090H.

Vertical scaling requires much more bandwidth than horizontal scaling. Therefore, vertical scaling should be done during the first pass and horizontal scaling during the second pass for stretching. The reverse is true for shrinking the image.

If filtering is required, it is enabled via bit 31 of 8088H. The type of filtering is specified via bits 15-14 of 808CH.

The output data format is specified via bits 18-16 of 8088H. If the output is 8 bits/pixel RGB, setting bit 28 of 8088H to 1 palettizes the data, i.e., each byte is a color look up table (LUT) address. This output protects the upper and lower 10 LUT slots, which are used for system colors. Therefore, it maps certain adjacent pairs of data values to the same LUT address so that all 256 values generate a non-protected color.

The output can be cropped by specifying a start value and length in 8094H. Alternately, the output can be masked against a mask pattern stored in main (system) memory. This is enabled by setting bit 26 of 8088H to 1. The mask contains one bit for each pixel in on-screen memory. The mask data must be sent by the CPU just prior to processing of the relevant screen pixels.

If bit 29 of 808CH is cleared to 0, a bit value of 1 in the mask allows the video to overwrite the graphics pixel and a value of 0 protects the graphics pixel from being overwritten. If this bit is set to 1, the effect is reversed.

6.3 COLOR SPACE CONVERSION

Either YUV or YCbCr 4-4-2 input can be converted to RGB format. YUV/YCbCr input is selected by setting bits 22-20 of 8088H to 100b. YUV or YCbCr is selected via bit 19 of 8088H (YUV = 1; YCbCr = 0). The output data format is selected via bits 18-16 of 8088H. This can be RGB-8 (=000b), RGB-15 (=110b), RGB-16 (=111b) or RGB-32 (=011b).

YUV/YCbCr data can be scaled before it is converted. If two pass scaling is being performed, color space conversion should be done during the horizontal pass.

6.4 DITHERING

www.DataSheet4U.com

If the bits/pixel (color depth) of the data in display memory is less than the input data (or the gener-

www.DataSheet4U.com

ated output of a color space conversion), the data must have its color depth reduced to match that used in display memory. This is called dithering. 24 bits/pixel can be dithered to 16, 15 or 8 bits/pixel. 16 or 15 bits/pixel can be dithered to 8 bits/pixel.

When the final output is 8 bits/pixel, the pixel value is an index into a 256 position color look up table (LUT). Windows uses the top and bottom 10 colors for its system colors. To avoid overwriting these colors, the Video Engine maps its 256 color values into the 236 LUT positions not used by Windows. The lost color values are spread evenly over the range to minimize the effect.

Dithering is enabled by setting bit 29 of 8088H to 1. The dithering matrix index must be programmed via bits 25-23 of 8088H.

Dithering is done after scaling and color space conversion. As with color space conversion, it should be done in the horizontal pass of a two pass scaling operation.

6.5 STATUS

Bit 31 of 809CH is read-only. When set to 1, it indicates the Video Engine is in use.

www.DataSheet4U.com

www.DataSheet4U.com

Appendix A: Listing of Raster Operations

The Vision868 supports all 256 triadic raster operations (ROPs) for BitBLTs as defined by Microsoft for Windows. The coding for these is found on the following pages.

The HEX value in the first column is the ROP code. This value must be programmed into bits 7-0 of D2E8H at the time that a ROPBLT command is executed.

The effect of the ROP is shown in reverse Polish notation in the second column. This is interpreted as follows:

S = Source bitmap

P = Pattern

D = Destination bitmap

The source bitmap can be either the CPU or the current screen, as selected via bits 6-5 of either A2E8H or A6E8H. A CPU source can be either monochrome or color. A screen source is always color.

The pattern, if present, is found at the off-screen memory location specified by EAE8H and EAEAH. The pattern may be either monochrome or color, as specified by bit 8 of D2E8H. If the pattern is monochrome, its background color is specified by E6E8H and its foreground color by EEE8H.

The destination bitmap is always the screen. It is always color (as opposed to monochrome).

The boolean operators used as as follows:

o = bitwise OR

x = bitwise EXCLUSIVE OR

a = bitwise AND

n = bitwise NOT (inverse)

For example, ROP 16H is PSDPSanaxx. The pattern is first ANDed with the source [PSD(PaS)naxx]. The result is inverted and then ANDed with the destination [PS((Da(notPaS))xx. This result is EXCLUSIVE ORed with the source. Finally, the result of this is EXCLUSIVE ORed with the pattern.

Programming using ROPBLTs is explained in Enhanced Mode Programming section.

www.DataSheet4U.com

www.DataSheet4U.com

HEX	In Reverse Polish
00	0
01	DPSoon
02	DPSona
03	PSon
04	SDPona
05	DPon
06	PDSxnon
07	PDSaon
08	SDPnaa
09	PDSxon
0A	DPna
0B	PSDnaon
0C	SPna
0D	PDSnaon
0E	PDSonon
0F	Pn
10	PDSona
11	DSon
12	SDPxnon
13	SDPaon
14	DPSxnon
15	DPSaon
16	PSDPSanaxx
17	SSPxDSxaxn
18	SPxPDxa
19	SDPSanaxn
1A	PDSPaox
1B	SDPSxaxn
1C	PSDPaox
1D	DSPDxaxn
1E	PDSox
1F	PDSoan
20	DPSnaa
21	SDPxon
22	DSna
23	SPDnaon
24	SPxDSxa
25	PDSPanaxn
26	SDPSaox
27	SDPSxnox
28	DPSxa
29	PSDPSaoxxn
2A	DPSana
2B	SSPxPDxaxn
2C	SPDSoax
2D	PSDnox
2E	PSDPxox
2F	PSDnoan
30	PSna
31	SDPnaon
32	SDPSoox
33	Sn
34	SPDSaox
35	SPDSxnox
36	SDPox
37	SDPoan
38	PSDPoax
39	SPDnox
3A	SPDSxox
3B	SPDnoan
3C	PSx
3D	SPDSonox
3E	SPDSnaox
3F	PSan
40	PSDnaa
41	DPSxon
42	SDxPDxa
43	SPDSanaxn
44	SDna
45	DPSnaon
46	DSPDaox
47	PSDPxaxn
48	SDPxa
49	PDSPDaoxxn
4A	DPSDoax
4B	PDSnox
4C	SDPana
4D	SSPxDSxoxn
4E	PDSPxox
4F	PDSnoan
50	PDna
51	DSPnaon
52	DPSDaox
53	SPDSxaxn
54	DPSonon
55	Dn
56	DPSox
57	DPSoan

www.DataSheet4U.com

www.DataSheet4U.com

HEX	In Reverse Polish
58	PDSPoax
59	DPSnox
5A	DPx
5B	DPSDonox
5C	DPSDxox
5D	DPSnoan
5E	DPSDnaox
5F	DPan
60	PDSxa
61	DSPDSaoxxn
62	DSPDoax
63	SDPnox
64	SDPSoax
65	DSPnox
66	DSx
67	SDPSonox
68	DSPDSonoxxn
69	PDSxxn
6A	DPSax
6B	PSDPSoaxxn
6C	SDPax
6D	PDSPDoaxxn
6E	SDPSnoax
6F	PDSxnan
70	PDSana
71	SSDxPDxaxn
72	SDPSxox
73	SDPnoan
74	DSPDxox
75	DSPnoan
76	SDPSnaox
77	DSan
78	PDSax
79	DSPDSoaxxn
7A	DPSDnoax
7B	SDPxnan
7C	SPDSnoax
7D	DPSxnan
7E	SPxDSxo
7F	DPSaan
80	DPSaa
81	SPxDSxon
82	DPSxna
83	SPDSnoaxn

HEX	In Reverse Polish
84	SDPxna
85	PDSPnoaxn
86	DSPDSoaxx
87	PDSaxn
88	DSa
89	SDPSnaoxn
8A	DSPnoa
8B	DSPDxoxn
8C	SDPnoa
8D	SDPSxoxn
8E	SSDxPDxax
8F	PDSanan
90	PDSxna
91	SDPSnoaxn
92	DPSDPoaxx
93	SPDaxn
94	PSDPSoaxx
95	DPSaxn
96	DPSxx
97	PSDPSonoxx
98	SDPSonoxn
99	DSxn
9A	DPSnax
9B	SDPSoaxn
9C	SPDnax
9D	DSPDoaxn
9E	DSPDSaoxx
9F	PDSxan
A0	DPa
A1	PDSPnaoxn
A2	DPSnoa
A3	DPSDxoxn
A4	PDSPonoxn
A5	PDxn
A6	DSPnax
A7	PDSPoaxn
A8	DPSoa
A9	DPSoxn
AA	D
AB	DPSono
AC	SPDSxax
AD	DPSDaoxn
AE	DSPnao
AF	DPno

www.DataSheet4U.com

www.DataSheet4U.com

HEX	In Reverse Polish
B0	PDSnoa
B1	PDSPxoxn
B2	SSPxDSxox
B3	SDPanan
B4	PSDnax
B5	DPSDoaxn
B6	DPSDPaoxx
B7	SDPxan
B8	PSDPxax
B9	DSPDaoxn
BA	DPSnao
BB	DSno
BC	SPDSanax
BD	SDxPDxan
BE	DPSxo
BF	DPSano
C0	PSa
C1	SPDSnaoxn
C2	SPDSonoxn
C3	PSxn
C4	SPDnoa
C5	SPDSxoxn
C6	SDPnax
C7	PSDPoaxn
C8	SDPoa
C9	SPDoxn
CA	DPSDxax
CB	SPDSaoxn
CC	S
CD	SDPono
CE	SDPnao
CF	SPno
D0	PSDnoa
D1	PSDPxoxn
D2	PDSnax
D3	SPDSoaxn
D4	SSPxPDxax
D5	DPSanan
D6	PSDPSaoxx
D7	DPSxan
D8	PDSPxax
D9	SDPSaoxn
DA	DPSDanax
DB	SPxDSxan

HEX	In Reverse Polish
DC	SPDnao
DD	SDno
DE	SDPxo
DF	SDPano
E0	PDSoa
E1	PDSoxn
E2	DSPDxax
E3	PSDPaoxn
E4	SDPSxax
E5	PDSPaoxn
E6	SDPSanax
E7	SPxPDxan
E8	SSPxDSxax
E9	DSPDSanaxxn
EA	DPSao
EB	DPSxno
EC	SDPao
ED	SDPxno
EE	DSo
EF	SDPnoo
F0	P
F1	PDSono
F2	PDSnao
F3	PSno
F4	PSDnao
F5	PDno
F6	PDSxo
F7	PDSano
F8	PDSao
F9	PDSxno
FA	DPo
FB	DPSnoo
FC	PSo
FD	PSDnoo
FE	DPSoo
FF	1

www.DataSheet4U.com