Part Number 800-1377-02
Rev: 50 of 20 February 1986

# Sun-3 Color Graphics Board

# Hardware Reference Manual

Sun Microsystems, Inc.,
2550 Garcia Avenue,
Mountain View,
California 94043
(415) 960-1300

# Credits and Trademarks

Sun Microsystems, SunStation and Sun Workstation are registered trademarks of Sun Microsystems, Incorporated. Sun3 is a trademark of Sun Microsystems, Incorporated.

VMEbus is a trademark of Motorola Corporation.

Multibus is a trademark of Intel Corporation.

UNIX is a trademark of AT&T Bell Laboratories.

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instructions manual, may cause interference to radio communications. It has been tested and found to comply with the limits for a Class A computing device pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

# Contents

# Tables

# Figures

# 1

# Introduction

# Introduction

This manual describes the Sun-3 color board for people who need to interact significantly with this board. It provides a functional description of the board, and lists and describes all the addressable facilities on the board.

## 1.1. Using this Manual

The following sections provide information to help you use this manual.

### Audience

The main audience for this manual is people who want to write programs to interact with the color board, without using *pixrect*. Note that you can use *pixrect*, which provides full access to the graphic capabilities of the Sun workstation, without the information in this manual.

The audience might also include persons who want to understand the Sun-3 color board, either to repair it, or for any other reason.

### Manual Organization

This manual provides different types of information about the color board. Chapter 2 describes what the board does at a high level, and introduces some of the principles used. Chapter 3 lists the various addressable devices in the board, and describes how to access them and what they do. Chapter 4 provides in-depth descriptions of the board functions, and Chapter 5 provides examples of the code that sets-up and interfaces this board.

### Fonts in Text

In this manual, we use fonts to make things a little clearer. The most common fonts are Roman, `typewriter`, *italic*, and bold. We use them as follows:

Roman
: Roman font is the standard for normal text, just as it appears here.

Typewriter
: `Typewriter font` is mostly used for information in displays.

Italic
: *Italic font* is either used for notes, or it represents a variable for which you or the computer must substitute some real value. For example:

> This field contains a pointer to register *nn*

Bold
: **Bold font** indicates that something deserves more attention than the surrounding text.

## 1.2. Glossary

In order to avoid confusion, this manual uses standard definitions for some common words. These are:

Frame Buffer
: The memory that contains a copy of the image on the video monitor.

RasterOp
: A unit which manipulates data in the frame buffer. Also called ROPC (RasterOp Chip).

RasterOp Mode
: An addressing mode that runs data through the ROPCs.

Pixel Mode
: An addressing mode where each byte represents a 'PEL' (picture element) on the screen.

Word Mode
: An addressing mode where the entire frame buffer appears as 8 128K X 1 bit stacks of memory.

Double Buffering
: A feature that enables the color board to maintain two images of frame buffer memory, one for the foreground (displayed on the video monitor), and another for the background.

Asserted
: A signal is asserted when it is ON or HIGH.

Deasserted
: A signal is deasserted when it is OFF or LOW.

## 1.3. References

Refer to the following documents for additional information:

*Principles of Interactive Computer Graphics* (Newmann & Sproul),

*VLSI Technology, Inc. IC V16160 Data Sheet.*

*Pixrect Reference Manual* — Part Number: 800-1254

*Sun-3 Color Board Engineering Manual* — Part Number: 800-1366

*Sun Configuration Procedures* — Part Number: 813-2000

# 2

# Color Board Overview

# Color Board Overview

This chapter contains an overview of the architecture and function of the Sun-3 color board. It describes what the board does, at a high level.

This chapter is intended to familiarize you with the basic functions of the board, and the basic concepts it uses. Subsequent chapters go into greater detail.

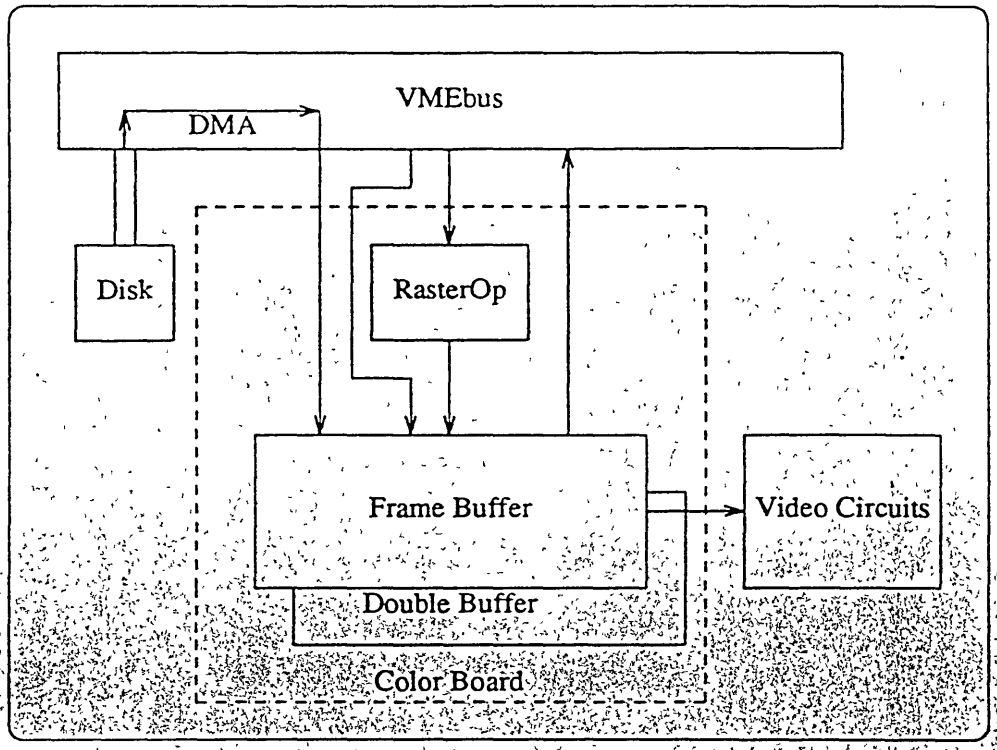
NOTE *Most programmers use Pixrects to create graphics on the Sun. Pixrects handle all interaction with the color board, while allowing you to take full advantage of the Sun graphics capability. Users of Pixrects have no need to read this manual.*

*For additional information on Pixrects, see "The Pixrect Reference Manual" (#800-1254).*

The color board is a VMEbus slave that provides a megabyte of RAM called a frame buffer, and circuits to manipulate the data in this RAM. The frame buffer contains the image on the video monitor; data goes from there directly to the video circuits.

The following figure shows the major blocks of the board, and its relationship to the rest of the system:

Figure 2-1 *The Color Board in the System*




## 2.1. Addressing

Jumpers on the color board select the board's base address, and whether it decodes 2 or 4 Mbytes of space starting at that base address. The board itself contains 1 Mbyte of frame buffer RAM and 1 Mbyte of control space, which contains registers, the color map, and an optional DMA space. With 2 Mbyte decoding, the frame buffer RAM occupies the first Mbyte, and the control space occupies the second. With 4 Mbyte decoding, the first 3 Mbytes provide three different methods of accessing the (same) 1 Mbyte frame buffer RAM, and the control space occupies the 4th Mbyte.

The default is to use 4 Mbyte decoding, with the base address at 4 Mbytes. This makes the board occupy from 4 to 8 Mbytes in VME address space.

## 2.2. RasterOps and Image Manipulation

RasterOp chips provide the main tool for manipulating the image in the frame buffer RAM. The RAM is organized as 8 planes of 128 Kbytes each, and each of these has a RasterOp chip assigned to it. The RasterOp chips each have a number of writable registers; when data is written to frame buffer RAM, it is manipulated according to the instructions in the registers of the relevant RasterOp chip.

The control space also contains a pseudo RasterOp unit, which acts like all 8 RasterOps in parallel. Changes directed to this unit change all the RasterOps. A register allows you to mask any of the 8 planes, protecting them selectively from all accesses.

## 2.3. Color Map

The color map is a 256 by 24 bit RAM circuit that uses ECL logic for fast access but is shadowed in a TTL circuit. The 24 bit locations each contain bits which describe a color; 8 bits of red, 8 bits of blue, and 8 bits of green. It uses a positive coloring scheme; increasing the value of the 8 bits assigned to each color increases the intensity of that color. Turning all 24 bits OFF creates black, and turning them all ON creates white.

The color map acts as a lookup table; 8-bit locations in the frame buffer (pixels) provide an index to a location in the color map. This allows each 8-bit pixel to display a 24-bit color. The color map can contain 256 out of 16M possible color combinations.

## 2.4. Optional Features

Some color boards contain an extra Mbyte of RAM for double buffering, and a DMA circuit. Double buffering creates 2 frame buffer RAMs; this provides the ability to display one on the screen while updating the other. A register allows you to designate one the background and the other the foreground, and to toggle this selection.

The DMA circuit allows you to take an image stored on disk and write it directly to the frame buffer.

# 3

# Software Model

# Software Model

This chapter provides a software model of the color board. It lists the addressable devices on the board and describes what they do.

## 3.1. Address Space Assignment

Jumpers on the color board select 2 or 4 Mbyte address decoding, and either 24-bit or 32-bit VMEbus addressing. They also select the board's base address. The defaults are 4 Mbyte address decoding and 24-bit VMEbus addressing, with the base at 4 Mbytes. The jumpers are listed and described in the "Sun Configuration Procedures" (#813-2000).

The following table shows the address offsets with both 2 Mbyte and 4 Mbyte address decoding. The 2 Mbyte address decoding reclaims space by causing the color board to ignore the word-mode and pixel-mode spaces, as shown in the table:

Table 3-1 *Frame Buffer Addressing*

| *Offset from Base* | *Accessed Entity* | *Description* |
|---|---|---|
| 0x000000 — 0x0FFFFF<br>Ignored in 2 Mbyte decoding | Word-Mode Memory | Frame buffer appears as a stack of eight memory planes. Each memory plane is equivalent to 128 Kbytes of system memory. A 32-bit word access addresses 32 adjacent bits within a bit-plane. Eight, 16 and 32-bit accesses. |
| 0x100000 — 0x1FFFFF<br>Ignored in 2 Mbyte decoding | Pixel-Mode Memory | Frame buffer appears as 1 million 8-bit-deep pixels. Memory planes masked by Per-Plane Mask Reg are write and read protected. Eight, 16 and 32-bit accesses. Multiple byte accesses access multiple pixels. |
| 0x200000 — 0x2FFFFF, or | RasterOp Memory | |

Table 3-1 *Frame Buffer Addressing— Continued*

| *Offset from Base* | *Accessed Entity* | *Description* |
|---|---|---|
| 0x000000 — 0x0FFFFF | | The eight RasterOp addressing modes are specified by the ropmode field in the status register. They are described later in this chapter. |
| 0x300000 — 0x31FFFF, or 0x100000 — 0x01FFFF | Control Registers and Color Maps | This address range contains the color board control devices. |
| 0x320000 — 0x3FFFFF, or 0x120000 — 0x1FFFFF | DMA Window Space | This space acts similarly to pixel mode memory, during DMA accesses. |

The following tables show the addresses in the control space. The first shows 2 Mbyte address decoding, and the second shows 4 Mbyte decoding:

Table 3-2 *Control Space Addressing — 2 Mbyte Decoding*

| *Offset from Base* | *Accessed Entity* |
|---|---|
| 0x100000 — 0x10001E | RasterOp unit — bit-plane 0. |
| 0x101000 — 0x10101E | RasterOp unit — bit-plane 1. |
| 0x102000 — 0x10201E | RasterOp unit — bit-plane 2. |
| 0x103000 — 0x10301E | RasterOp unit — bit-plane 3. |
| 0x104000 — 0x10401E | RasterOp unit — bit-plane 4. |
| 0x105000 — 0x10501E | RasterOp unit — bit-plane 5. |
| 0x106000 — 0x10601E | RasterOp unit — bit-plane 6. |
| 0x107000 — 0x10701E | RasterOp unit — bit-plane 7. |
| 0x108000 — 0x10801E | On write, Pseudo RasterOp unit — will write to all ROPC enabled by Per-Plane Mask register. |
| 0x109000 | Status Register (16-bit) |
| 0x10A001 | Per-Plane Mask Register (8-bit) |
| 0x10B000 | Double Buffering Register (8-bit) |
| 0x10C000 | DMA Window Origin (A19 to A04) (16-bit) |
| 0x10D000 | DMA Window Width Register (8-bit) |
| 0x10D001 | Frame Count Register (8-bit) |
| 0x10F001 | VME Interrupt Vector Register (8-bit) |
| 0x110000 — 0x1101FE | Red Shadow Color Map — Entries 0 to 255. |
| 0x110200 — 0x1103FE | Green Shadow Color Map — Entries 0 to 255. |
| 0x110400 — 0x1105FE | Blue Shadow Color Map — Entries 0 to 255. |

Table 3-3 *Control Space Addressing — 4 Mbyte Decoding*

| *Offset from Base* | | *Accessed Entity* |
|---|---|---|
| 0x300000 – | 0x30001E | RasterOp unit — bit-plane 0. |
| 0x301000 – | 0x30101E | RasterOp unit — bit-plane 1. |
| 0x302000 – | 0x30201E | RasterOp unit — bit-plane 2. |
| 0x303000 – | 0x30301E | RasterOp unit — bit-plane 3. |
| 0x304000 – | 0x30401E | RasterOp unit — bit-plane 4. |
| 0x305000 – | 0x30501E | RasterOp unit — bit-plane 5. |
| 0x306000 – | 0x30601E | RasterOp unit — bit-plane 6. |
| 0x307000 – | 0x30701E | RasterOp unit — bit-plane 7. |
| 0x308000 – | 0x30801E | On write, Pseudo RasterOp unit — will write to all ROPC enabled by Per-Plane Mask register. |
| | 0x309000 | Status Register (16-bit) |
| | 0x30A001 | Per-Plane Mask Register (8-bit) |
| | 0x30B000 | Double Buffering Register (8-bit) |
| | 0x30C000 | DMA Window Origin (A19 to A04) (16-bit) |
| | 0x30D000 | DMA Window Width Register (8-bit) |
| | 0x30D001 | Frame Count Register (8-bit) |
| | 0x30F001 | VME Interrupt Vector Register (8-bit) |
| 0x310000 – | 0x3101FE | Red Shadow Color Map — Entries 0 to 255. |
| 0x310200 – | 0x3103FE | Green Shadow Color Map — Entries 0 to 255. |
| 0x310400 – | 0x3105FE | Blue Shadow Color Map — Entries 0 to 255. |

Each RasterOp unit has fourteen 16-bit registers. The following table shows their address offsets from the base address of the RasterOp chip:

Table 3-4 *RasterOp Unit Register Addressing*

| *ROPC Base Offset* | *Data Bits* | *Accessed Entity* |
|---|---|---|
| 0x00 | D15..D0 | Destination Register |
| 0x02 | D15..D0 | Source Register 2 — Least-significant word of SRC. |
| 0x04 | D15..D0 | Source Register 1 — Most-significant word of SRC. |
| 0x06 | D15..D0 | Pattern Register. |
| 0x08 | D15..D0 | Mask1 Register — Enabled when "Opcount" equal to zero. |
| 0x0A | D15..D0 | Mask2 Register — Enabled when "Opcount" equals "Width." |
| 0x0C | D15..D9 | Write as zeros, Read as Don't Care |
| | D8 | Sourceload Bit — If zero, SRC2 loaded from system data bus, SRC1 loaded from SRC2. If asserted, SRC1 loaded from system data bus, SRC2 loaded from SRC1. |
| | D7..D4 | |
| | D3..D0 | SRC Shift Amount — SRC data bits output to function unit are bits "Shift Amount" through "Shift Amount + 15." |
| 0x0E | D15..D8 | Write as zeros, Read as Don't Care |
| | D7..D0 | Function Register. |
| 0x10 | D15..D0 | Width Register — Specifies width of Raster in words. |
| 0x12 | D15..D0 | Opcounter — Loaded from Width Register when Opcounter equal to zero. Decremented every frame buffer write. Controls enabling of mask registers. |
| 0x14 | D15..D0 | Decoder output latch — Read-Only. For diagnostic purposes. Gives output of RasterOp unit based on current register values. |
| 0x16 | D15..D0 | Manual load destination — Loads destination and strobes LD.DST pin. For diagnostic purposes. |
| 0x18 | D15..D0 | Manual load source — Loads source register and strobes LD.SRC pin. For diagnostic purposes. |
| 0x1A | D15..D8 | Write as zeros, Read as Don't Care |
| | D7..D0 | Flag register for applications software |

## 3.2. Control Registers

This section describes the each of the control registers. Their addresses were provided in an earlier table.

### Status Register

The status register contains 16 bits, and is cleared when a bus reset is issued. It contains the following fields:

Table 3-5 *Bit Assignments for Status Register*

| *Bit* | *Name* | *Function* |
|---|---|---|
| D0 | video_enab | When asserted this bit enables the video DACs. |
| D1 | update_ecmap | When both this bit and the bit by the same name in the status register are deasserted the host software can read or write the TTL shadow color map. When either of these bits are asserted, either the TTL or ECL color map will be transferred to the other during vertical retrace. When cleared, this bit takes effect immediately: when set, this bit is sampled by the leading edge of vertical retrace. |
| D2 | inten | When asserted, an interrupt is generated at the start of vertical retrace. Clearing the bit stops a pending interrupt. |
| D3 | ropmode0 | LSB of 3-bit field specifying current RasterOp-mode. |
| D4 | ropmode1 | Bit in 3-bit field specifying current RasterOp-mode. |
| D5 | ropmode2 | MSB of 3-bit field specifying current RasterOp-mode. |
| D6 | inpend | Read-only. Asserted if board is interrupting. |
| D7 | retrace | Read-only. Vertical retrace in progress. |
| D8 | Resolution | Read-only. — "0" => 1152x900 display; "1" => 1024x1024 display. |
| D9 | Flag09 | Read-only. User-system definable jumper — must be 0 to run Pixrect software. |
| D10 | Flag10 | Read-only. User-system definable jumper — must be 0 to run Pixrect software. |
| D11 | Flag11 | Read-only. User-system definable jumper. |
| D12 | Flag12 | Read-only. User-system definable jumper. |

Table 3-5 *Bit Assignments for Status Register— Continued*

| *Bit* | *Name* | *Function* |
|---|---|---|
| D13 | Flag13 | Read-only. User-system definable jumper. |
| D14 | Flag14 | Read-only. User-system definable jumper. |
| D15 | Flag15 | Read-only. User-system definable jumper. |

**Per-Plane Mask Register**

The ppmask register is an 8-bit register used for restricting frame buffer access to selected bit-planes of the frame buffer memory. It also restricts access to the registers of the RasterOp chip for that bit-plane. Bit zero of ppmask corresponds to plane zero (the low order bit of a pixel) and so on. A "0" bit in the ppmask register prevents modification of the corresponding bit-plane during writes, and causes a zero to be returned from that bit-plane during reads.

The ppmask register applies to both pixel-mode and parallel word-mode RasterOp memory access. In parallel word-mode, data is written from the RasterOp chips to all bit-planes in parallel, and the per-plane mask register selectively enables/disables bit-planes from being written. These modes are described later in this chapter.

**Double Buffering Register**

The double buffering register is a 16-bit register that is cleared when a bus reset is issued. It contains the following fields:

Table 3-6 *Bit Assignments for Double Buffering Register*

| *Bit* | *Name* | *Function* |
|---|---|---|
| D0-D7 | Reserved | These bits are currently not used. |
| D8 | update_ecmap | When both this bit and the bit by the same name in the status register are deasserted, the host software can read or write the TTL shadow color map. When either of these bits are asserted, either the TTL or ECL color map is transferred to the other during vertical retrace, depending on the state of the read_ecmap bit. When cleared, this bit takes effect immediately; when set, this bit is sampled by the leading edge of vertical retrace. |
| D9 | wait | This bit can be used to determine when a color map update or switch of foreground/background has taken effect. When *wait* is low, writing a "1" sets the bit. The bit then remains high until a complete vertical retrace period has elapsed, after which it clears itself. |

Table 3-6 *Bit Assignments for Double Buffering Register— Continued*

| *Bit* | *Name* | *Function* |
|---|---|---|
| D10 | fast_read | When asserted, a read access to the frame buffer memory returns an immediate DTACK but also returns invalid data. Using this mode can speed operations like scrolling by 50%. |
| D11 | read_ecmap | This bit specifies the direction of data transfer between the ECL and TTL color maps when "update_ecmap" is active. When cleared, the ECL map is loaded from the TTL map; when set, the TTL map is loaded from the ECL map. |
| D12 | nowrite_a | When cleared, writes to double-buffered memory set A are enabled. When set, writes to double-buffered memory set A are inhibited. |
| D13 | nowrite_b | When cleared, writes to double-buffered memory set B are enabled. When set, writes to double-buffered memory set B are inhibited. |
| D14 | read_b | When cleared, frame buffer read and "hidden-read" cycles access memory set A. When set, frame buffer read and "hidden-read" cycles access memory set B. Memory sets are described in the chapter *Color Board Functions.* |
| D15 | display_b | When this bit is low, the contents of memory set A are output to the CRT. When this bit is set, the contents of memory set B are output to the CRT. This bit is synchronized with the start of vertical retrace; changes do not take place until the start of vertical retrace. This prevents "tearing", where you see one picture on top of the display and another picture on the bottom. |

**DMA Window Origin Register**

This register specifies the beginning of a DMA window. It represents the upper left hand corner of the window, and it must be on a pixel boundary (mod 16). The use of the DMA window is described in the chapter *Color Board Functions.*

**DMA Window Width Register**

This register holds a number representing the width of the DMA window divided by 16. It represents the width of a line in the DMA window; when the number of pixels divided by 16, minus 1 ((pixels /16) -1) is equal to the value in this window, it starts a new line.

**Frame Count Register**

This read-only register is incremented by the hardware each vertical retrace.

**VME Interrupt Vector Register**

This register contains the 8-bit interrupt vector which the color board provides to the VMEbus when it produces a vectored interrupt. Note that the color board can also use a polled interrupt scheme implemented with the *inpend* bit in the status register.

## 3.3. Addressing Modes

In the 4MB space, the first megabyte is word-mode memory space, the second megabyte is pixel-mode memory space, the third megabyte is RasterOp memory space and the fourth megabyte contains the control registers, the color maps and the optional DMA window space. However, when used in the UNIX environment the word and pixel-mode memory spaces are never used, and this addressing space can be reclaimed. Shrinking the addressing space to 2MByte is essential for a three board, 24-bit color configuration that must fit into the 24-bit addressing sub-space of the VME.

The two non-RasterOp addressing modes are:

Word-Mode Memory
: The frame buffer appears as a stack of eight memory planes. Each memory plane is equivalent to 128 Kbytes of system memory. An 8/16/32-bit word access addresses 8/16/32 adjacent bits within a bit-plane. This mode doesn't use the frame buffer RasterOp chips.

Pixel-Mode Memory
: The frame buffer appears as 1 million 8-bit-deep pixels. Memory planes masked by the per-plane mask register are read and write protected (masked bits return zeros on read). 8/16/32-bit accesses transfer 1/2/4 pixels simultaneously. Pixel-mode doesn't use the frame buffer RasterOp chips.

The remaining eight addressing modes, which are selected by the *ropmode* bits in the status register, use the RasterOp chips. There is one RasterOp chip (ROPC) for each of the eight bit planes of the frame buffer. As described in the chapter *Color Board Functions*, each RasterOp chip has source, destination and pattern registers upon which to perform a three operand RasterOp. The source and destination registers may be automatically loaded on read or write accesses to the frame buffer memory. The source register may be loaded from the data bus on a read or write access, the destination register may be loaded from the frame buffer memory on a read or a write access, and the pattern register is loaded by writing directly to it. Note that reads and writes to memory and RasterOp chips are only performed on planes not masked by the per-plane mask register.

The RasterOp chips are 16-bit devices and not all 32-bit VME bus cycles appear software compatible with 16-bit VME bus cycles. All 32-bit bus cycles are broken down into separate 16-bit cycles on the board. Therefore, in RasterOp modes 0,1,4,5 and 6 (see the following table), a 32-bit read followed by a 32-bit write is not the same as a 16-bit read followed by a 16-bit write followed by a 16-bit read followed by a 16-bit write. Note that 32-bit data transfers on MC68010 CPUs are broken into two 16-bit bus cycles and that this incompatibility between Sun-2 and Sun-3 color boards may be masked by a Sun-2 CPU.

The RasterOp Mode table uses the following shorthand notation for ropmode: The first character specifies either parallel (P) or single (S) transfer. The second character specifies either load ROPC destination registers on a read (R) or write (W) access. The third character specifies either load ROPC source registers on a read (R) or write (W) transfer. The last three characters specify word- (WRD) or pixel- (PIX) mode transfer. The eight RasterOp addressing modes are described in the following table:

Table 3-7 *RasterOp Addressing Modes*

| RasterOp Addressing Modes | | |
|---|---|---|
| *Addressing Mode* | *Notation* | *Description* |
| 1 — Parallel Word Mode with RasterOp | PRWWRD | One or two 16-bit words are written in parallel to each plane not masked by the per-plane mask register. Accesses address 1 to 8 bit positions within 16/32 adjacent pixels. This mode is useful for painting 1-bit/pixel data into the frame buffer.<br><br>An 8/16-bit frame buffer read access loads the *destination* register on each RasterOp chip (ROPC), and an 8/16-bit frame buffer write loads the bus data into all ROPC *source* registers. They also write the ROPC output to frame buffer memory.<br><br>Thirty-two bit bus transfers are equivalent to separate 16-bit bus transfers with an incrementing address. |

Table 3-7 *RasterOp Addressing Modes— Continued*

| RasterOp Addressing Modes | | |
|---|---|---|
| *Addressing Mode* | *Notation* | *Description* |
| 2 — Single Pixel Mode with RasterOp | SRWPIX | This mode accesses one, two or four 8-bit pixels in the frame buffer. This mode is useful for painting 8-bit/pixel data into the frame buffer.<br><br>On 8/16-bit writes, the data is transferred to the ROPC *source* registers and on to the frame buffer memory. A byte write loads all ROPC *source* registers with the value of that bit in the byte which corresponds to the ROPC's bit-plane. That is, bit zero of the data byte goes to all bits of the source register for bit-plane zero and so on. A 16-bit write loads the odd and even bits in the ROPC source registers such that ROPC plane 0 gets an alternating pattern consisting or bits 1 and 0, ROPC plane 1 gets an alternating pattern consisting of bits 3 and 3, ROPC plane 2 gets an alternating pattern consisting of bits 5 and 4, and so on, up to ROPC plane 7, which gets an alternating pattern consisting of bits F and E. The output from the ROPCs is masked to allow only the addressed pixels to be written to the unmasked frame buffer memory planes.<br><br>An 8/16-bit frame buffer read access loads the ROPC *destination* registers. A byte read returns an 8-bit pixel and a 16-bit read returns two adjacent pixels.<br><br>Thirty-two bit bus transfers are equivalent to separate 16-bit bus transfers with an incrementing address. |

Table 3-7 *RasterOp Addressing Modes— Continued*

| RasterOp Addressing Modes | | |
|---|---|---|
| *Addressing Mode* | *Notation* | *Description* |
| 3 — Parallel Word Mode with RasterOp and Hidden Read | PWWWRD | This addressing mode is similar to PRWWRD. However read accesses have no effect on the RasterOp chips, and write accesses generate a hidden read cycle to memory before the write cycle.<br><br>In a write access, a read cycle loads the ROPC *destination* registers with the data in the frame buffer memory at the address to be written, and the write cycle loads the ROPC *source* registers from the data bus as before. The ROPC output is then written to all bit-planes enabled by the per-plane mask register.<br><br>Read accesses in this mode merely return the data accessed; they do not load the ROPC source or the destination registers. All 8/16-bit frame buffer write accesses in this mode take two memory cycles to complete. 32-bit read accesses also take two 16-bit memory cycles to complete, and 32-bit write accesses take four 16-bit memory cycles to complete.<br><br>Thirty-two bit bus transfers are equivalent to separate 16-bit bus transfers, each with an incrementing address. |

Table 3-7 *RasterOp Addressing Modes— Continued*

| RasterOp Addressing Modes | | |
|---|---|---|
| *Addressing Mode* | *Notation* | *Description* |
| 4 — Single Pixel Mode with RasterOp and Hidden Read | SWWPIX | This addressing mode is similar to SRWPIX. However, read accesses have no effect on the RasterOp chips, and write accesses generate a hidden read cycle to memory before the write cycle.<br><br>In a 8/16-bit write access, the read cycle loads the ROPC *destination* registers with the data in the frame buffer memory at the address to be written, and the write cycle loads the ROPC *source* registers from the data bus as before. The ROPC output is then written to one or two 8-bit pixels in the frame buffer. 32-bit accesses are equivalent to two consecutive 16-bit accesses.<br><br>Read accesses in this mode merely return the accessed data; they do not load the ROPC source or the destination registers. All 8/16-bit frame buffer write accesses in this mode take two memory cycles to complete. 32-bit read accesses also take two 16-bit memory cycles to complete, and 32-bit write accesses take four 16-bit memory cycles to complete.<br><br>Short and integer bus transfers in this mode are always equivalent regardless of the mix of Sun-2 and Sun-3 color and CPU boards. |

Table 3-7 *RasterOp Addressing Modes— Continued*

| RasterOp Addressing Modes | | |
|---|---|---|
| *Addressing Mode* | *Notation* | *Description* |
| 5 — Parallel Word Mode with RasterOp | PRRWRD | This addressing mode is useful for moving a region from place to place in the frame buffer. A single 16-bit move instruction (*fba++ = *fbb++) copies 16 8-bit pixels. Since this mode does not load the destination on a write, it is not useful for ROP functions which need the destination (for example, PIX_SRC ^ PIX_DST).<br><br>A 8/16-bit frame buffer read loads all 8 ROPC *source* registers (128 bits) in parallel from frame buffer memory. A 16-bit frame buffer write writes the ROPC output to all memory planes enabled by the per-plane mask register without loading either the source or destination registers.<br><br>A 32-bit bus read cycle performs two 16-bit read cycles, which causes the second 16-bit read to overwrite the data from the first read. With a Sun-3 CPU and Sun-3 color board, only 16-bit cycles (short data) should be used with this RasterOp mode. With the Sun-3 CPUs this causes little loss in performance. For performance reasons, 32-bit data (int) should still be used when using a Sun-2 CPU. |

Table 3-7 *RasterOp Addressing Modes— Continued*

| RasterOp Addressing Modes | | |
|---|---|---|
| *Addressing Mode* | *Notation* | *Description* |
| 6 — Parallel Pixel Mode with RasterOp | PRWPIX | On write cycles, this pixel mode is a combination of pixel-mode SRWPIX and word-mode PRWWRD. Read cycles are identical to pixel-mode SRWPIX. On writes the ROPC source registers are loaded as with mode SRWPIX. For review, on 8-bit write cycles, data bit zero (D0) goes to all source register bits for bit-plane 0, and data bit one (D1) goes to all source register bits for bit-plane 1, and so on for all eight bit planes. On 16-bit memory write cycles, datum <D8,D0,D8,D0,D8,D0,D8,D0,D8,D0,D8,D0,D8,D0,D8,D0> is transferred to the source register for bit-plane 0, datum <D9,D1,D9,D1,D9,D1,D9,D1,D9,D1,D9,D1,D9,D1,D9,D1> is transferred to the source register for bit-plane 1, and so on. This mode differs from SRWPIX in that the pixels updated on write cycles are equivalent to the pixels that could be written with a modified address in mode PRWWRD; this mode writes to 8 or 16 adjacent pixels aligned, respectively, on modulo 8 and 16 pixel boundaries.<br><br>Because this mode provides a means for broadcasting 8 or 16 adjacent pixels with the same value, 32-bit VME cycles make no sense. A 32-bit cycle in this mode runs two back-to-back 16-bit cycles to the same address. |
| 7 — Parallel Word Mode with RasterOp and Hidden Read | PWRWRD | This addressing mode is similar to PRRWRD except that on write cycles, the frame buffer executes a read cycle to load the ROPC *destination* registers before the write cycle is executed. As with mode PRRWRD, read accesses load the ROPC *source* registers on all bit-planes, and return the accessed datum. This mode is used for copying within the frame buffer when the ROP function requires the destination (for example, PIX_SRC \| PIX_DST).<br><br>A 32-bit bus read cycle performs two 16-bit read cycles, and causes the second 16-bit read to overwrite the data from the first read. With a Sun-3 CPU and Sun-3 color board, use only 16-bit cycles (short data) with this RasterOp mode. With the Sun-3 CPUs this causes little loss in performance. For performance reasons, 32-bit data (int) should still be used with a Sun-2 CPU. |

Table 3-7 *RasterOp Addressing Modes— Continued*

| RasterOp Addressing Modes | | |
|---|---|---|
| *Addressing Mode* | *Notation* | *Description* |
| 8 — Parallel Pixel Mode with RasterOp and Hidden Read | PWWPIX | This addressing mode is similar to PRWPIX. However, read accesses have no effect on the RasterOp chips, and write accesses generate a hidden read cycle to memory before the write cycle. In a write access, the read cycle loads the ROPC *destination* registers with the data in the frame buffer memory at the address written to and the write cycle loads the ROPC *source* registers from the data bus as before. The ROPC output is then broadcast to 8 or 16 adjacent pixels. Write accesses take two memory cycles to complete using this addressing mode.<br><br>Because this mode provides a means for broadcasting 8 or 16 adjacent pixels with the same value, 32-bit VME cycles make no sense. A 32-bit cycle in this mode runs two back-to-back 16-bit cycles to the same address. |

The following table shows the relationship between the RasterOp mode bits in the status register, the type of access, and whether the LD_DST and LD_SRC registers get loaded:

| ROPmode bits in Status | Mode Description | Reads | | Writes | |
|---|---|---|---|---|---|
| | | Dst Reg loaded? | Src Reg loaded? | Dst Reg loaded? | Src Reg loaded? |
| 000 | All 8 planes | Yes | No | No | Yes |
| 001 | 1 pixel | Yes | No | No | Yes |
| 010 | All 8 planes | No | No | Yes | Yes |
| 011 | 1 pixel | No | No | Yes | Yes |
| 100 | All 8 planes | Yes | Yes | No | No |
| 101 | 16 pixels | Yes | No | No | Yes |
| 110 | All 8 planes | No | Yes | Yes | No |
| 111 | 16 pixels | No | No | Yes | Yes |

All addressing modes specified as "word-mode" can be further delineated by the address range corresponding to each bit-plane. The breakdown of address offsets to bit-planes follows:

Table 3-8 *Frame Buffer Addressing in Word Mode*

| *Offset from Word-Mode Base* | *Accessed Entity* |
|---|---|
| 0x000000 – 0x01FFFE | Word-mode bit-plane 0. |
| 0x020000 – 0x03FFFE | Word-mode bit-plane 1. |
| 0x040000 – 0x05FFFE | Word-mode bit-plane 2. |
| 0x060000 – 0x07FFFE | Word-mode bit-plane 3. |
| 0x080000 – 0x09FFFE | Word-mode bit-plane 4. |
| 0x0A0000 – 0x0BFFFE | Word-mode bit-plane 5. |
| 0x0C0000 – 0x0DFFFE | Word-mode bit-plane 6. |
| 0x0E0000 – 0x0FFFFF | Word-mode bit-plane 7. |

### Pixel-Mode Addressing

In pixel-mode, each pixel occupies a byte value. Pixel number 0 is in the upper left corner of the screen, and successive pixels are displayed horizontally towards the right. At each "MOD 1152" pixel address, a new vertical line is started. Mapped as memory, the pixel data written to the device enters the frame buffer without modification. Mapped through the RasterOp unit, the data value written to the device is modified as described in the previous section. Regardless of RasterOp support, an arbitrary selection of bit-planes can be masked from reads or writes by the Per-Plane Mask Register. Pixel-mode accesses can be 8, 16 or 32-bits. The 16- and 32-bit accesses assume that two or four pixels are packed into each datum.

### Word-Mode Addressing

In word-mode, the frame buffer is configured as eight separate bit-planes. Each bit-plane occupies a separate 128 Kbyte block of the system address space, and each bit-plane is architecturally identical to the Sun-2 black and white frame buffer. In this addressing mode, writing to the device alters 8, 16 or 32 horizontally adjacent pixels. Word 0 is in the upper left corner of the display, and each scan line is 36 32-bit words wide (1152/32 = 36). The most significant bit of a word appears to the left of the least significant bit of a word. Data written in word-mode can be written directly to the frame buffer, or it can be optionally combined via the RasterOp unit with the data currently at that address. This set of addressing modes, in conjunction with the RasterOp chips, may be used for painting raster font text.

A word-mode memory access may only address one of the eight memory planes of the frame buffer. However, all RasterOp word-modes (PRWWRD, PWWWRD, PRRWRD, and PWRWRD) access all eight memory planes in parallel.

The color frame buffer memory is dual ported. One port connects to the synchronous system bus and the second is dedicated to video refresh, which has priority over system bus accesses. A new 16/32-bit datum can be read or written to the Sun-3 color frame buffer every 345/690 nsec.

# 4

# Color Board Functions

# Color Board Functions

This chapter describes the functions of the color board in detail. It describes RasterOp units, the color map, interrupt handling, double buffering, and DMA. The information should help programmers understand the functions enough to interact with them.

## 4.1. RasterOp Units

The RasterOp units are assigned one to a bit-plane, with the pseudo-RasterOP unit writing in parallel to all. The data paths connecting and coordinating these per-plane RasterOp units are somewhat complex. This section attempts to explain their interconnection; it assumes a general familiarity with the concepts and terminology of "RasterOp."

For more information on RasterOp units, see *Principles of Interactive Computer Graphics* (Newmann & Sproul), and *VLSI Technology, Inc. IC V16160 Data Sheet*.

RasterOp control registers can be accessed implicitly as well as explicitly. Explicit reads include, for example, reads to the RasterOp chips' registers. Implicit reads take place when the source and destination registers are loaded implicitly on certain read and write operations to RasterOp frame buffer memory.

Reads and writes to the pseudo RasterOp unit access all RasterOp units in parallel by writing to ropcontrol[CG3_ALLROP]. Only those ROPC units on bit-planes enabled by the *per-plane mask register* are actually modified. A read from the ALLROP unit reads the value of ROPC plane 0 registers.

Implicit writes to the RasterOp units only occur to the source and destination registers. All implicit writes occur to all RasterOp units in parallel with no performance penalty. The rules regarding implicit writes to the source and destination registers depend on the addressing mode used (addressing modes are described in the chapter *Software Model*):

> All explicit writes occur during RasterOp mode writes. During addressing modes PRWWRD, SRWPIX, or PRWPIX, frame buffer reads implicitly load the destination registers on all ROPC chips, frame buffer writes actually load the system bus data to the ROPC source registers, and the ROPC output is written to the frame buffer. The destination registers of the ROPC chips must be loaded any time the output of the ROPC depends on the existing frame buffer contents at the output address. This is true if the RasterOp function is a logical operation involving the destination (for example,

PIX_NOT( PIX_DST)). It is also true when masking the first and last word of a raster line.

With the addressing modes PWWWRD, SWWPIX, or PWWPIX, frame buffer reads have no effect on the ROPC chips, but frame buffer writes load the system bus data to the ROPC source registers. A hidden read cycle loads the destination registers in the RasterOp chips, and writes ROPC output to the frame buffer.

With addressing mode PWRWRD, a frame buffer read implicitly loads the source and a write implicitly loads the destination of the ROPC.

With addressing mode PRRWRD, a frame buffer read implicitly loads both the source and destination ROPC registers. A write loads neither, but the ROPC output is written to the frame buffer.

When using the RasterOp chips with *pixel-mode accesses*, ROPC source register bits are loaded based on the address and data on the system bus. The ROPC source register for frame buffer bit-plane *n* loads every other bit with the value of bit *n* in the data byte. If the address is even the even bits of the source register are loaded, otherwise the odd bits are loaded. On a word transfer (to an even address) the low order data byte loads the even bits and the high order byte loads the odd bits. Thus a word transfer to an even address causes the eight ROPC source registers to be loaded and the output of the ROPC are masked so that only two pixels, composed of two adjacent bits in the eight source registers, are written to the frame buffer. If the transfer is a byte, a single pixel is written to the frame buffer. In other words, the ROPC output in pixel-mode is written through a write enable so that only the addressed pixels get written.

The *right mask* and *left mask* registers in the RasterOp chips can be used to properly clip the ends of the raster lines of a region to be painted.

## 4.2. A Single RasterOp Unit

A RasterOp chip (ROPC) exists for each of the eight planes of frame buffer memory. The complete state of a ROPC may be read or written. Each ROPC unit consists of a destination register, 32-bit source register, pattern register, mask1 register, mask2 register, shiftcount register, function register, width register, opcount register, decoder output latch, and an opcontrol register. This section describes the function of a ROPC and the registers which control it.

The addresses of these registers are listed in the chapter *Software Model.*

NOTE *The rasterop chips are 16-bit devices and 32-bit frame buffer accesses are broken into separate 16-bit memory cycles to the RasterOp chips.*

### RasterOp Op Register

Each RasterOp has three operands; the source, destination, and pattern. The source is typically loaded into the ROPC source FIFO from the system data bus. The destination is typically loaded into the ROPC destination register from the frame buffer memory. The pattern is written directly to the ROPC pattern register. The ROPC generates an output to the frame buffer by performing a bitwise logical operation on these three operands. The particular logical operation is specified by the *op* register.

The RasterOps support 256 possible functions, mapping three boolean operands into a boolean result. The frame buffer's 8-bit function register specifies one of these by acting as a three-bits-in, one-bit-out lookup table for corresponding bits of the destination, source, and pattern. For example, suppose you want to set destination equal to (Dst OR Src), ignoring the value of the pattern. The function may be expressed in tabular form as follows:

Table 4-1 *RasterOp Register Functions*

| PAT | SRC | DST | DST' = SRC OR DST |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

The PAT, SRC, and DST columns in the table form an index running from zero (000) through seven (111). The eight bits of the result column uniquely specify the desired boolean function, and these are precisely the eight bits which are to be loaded into the frame buffer's function register. By convention, the least significant bit of the function appears at the top of the table, hence this function (Src OR Dst) is represented by the eight-bit value 11101110 (0xEE). This RasterOp function applies equally to each bit of a pixel; if bit *n* of the source is 1 and bit *n* of the destination is 0, then bit *n* of the output to the frame buffer will be 1. Examples of other function encodings are 0x0 (clear destination bits), 0xFF (set destination bits), and 0xCC (copy source to destination). To clear the entire screen, the constant function 0 is applied to the viewable rectangle. To flash (invert) a window, the function NOT Dst is performed on that window. The function SRC writes a character, the function NOT SRC writes the character inverted (black on white). Dst OR Src overstrikes (paints) the character, and Src OR Pat writes the character with a background pattern. Thus the Sun-3 Color Graphics board allows all 256 possible RasterOp functions, although in practice only a few are used.

## RasterOp Source Fifo and Shift Registers

A ROPC has a 32-bit source FIFO called *source1* (right hand 16-bits) and *source2* (left hand 16-bits). This FIFO is for aligning source data with arbitrary pixel locations in the frame buffer. Recall that word-mode writes to the frame buffer are written to fixed boundaries. The source operand used by the ROPC op function is extracted from the FIFO as any contiguous 16-bit field in the 32-bit (2 word) FIFO. The ROPC *shift* register determines which 16 bits to extract by giving an offset from the right end of the FIFO.

The FIFO may be run in either direction. Bit 8 of the shift register determines the direction. Contiguous addressing may proceed either to the right or to the left

as needed in order to avoid overwriting the source rectangle when copying from location to location within the frame buffer. If the direction bit is 1 (right) a source load moves source1 to source2 and moves the system data bus value to source1. If the shift is non-zero then the FIFO may require priming (pre-load of the source1 register) at the start of a raster line. If the direction bit is 0, a load source operation moves source2 to source1 and moves the bus value to source2. In this case priming is done by pre-loading the source2 register. If the shift is zero then the source operand is taken from the end of the FIFO; if the direction bit is 1 then source operand = source2 else source operand = source1.

### RasterOp Destination and Pattern Registers

The destination operand is the 16-bit ROPC *dest* register which may be implicitly loaded on a frame buffer read or write depending on the ropmode field of the status register. The destination operand is aligned with 16-bit words in the frame buffer memory.

The pattern operand is the 16-bit ROPC *pattern* register which must be explicitly loaded by writing directly to it. It is always aligned on 16-bit frame buffer boundaries.

When a write to the frame buffer occurs the three operands are processed by the ROPC "decoder" according to the op function to produce a 16-bit ROPC output. This output is masked as required by the *opcount* and *width* registers and is stored in the *decoderout* register. If pixel-mode is being used the pixel is extracted from the 8 ROPC decoder outputs and is written to the frame buffer at the byte address specified on the system address bus. If word-mode is being used the 8 ROPC decoder outputs are written to the specified word address in the frame buffer.

### RasterOp Mask, Width, and Opcount Registers

The portion of a raster line to be written does not always begin and end on 16-bit boundaries. Therefore, the first and last portion of the raster line must be masked so that only the appropriate bits get written into the frame buffer. The ROPCs perform this masking via the *mask1*, *mask2*, *width*, and *opcount* registers.

NOTE *The first and last 16-bit word may be the same and both masks are then applied. The width and opcount registers control the application of the masks to the first and last 16-bit words of the raster line.*

To use the masks, software must determine the width (in 16-bit words) of the raster. The width register must be set to the number of 16-bit boundaries crossed by the raster. If the raster is entirely contained in 16-bits the width is zero. If the raster is only three bits wide but spans a 16-bit boundary, then the width of the raster is one.

The *opcount* is a variable register. At the start of a raster line it must be explicitly loaded with the same value as the width register. It is automatically decremented by the ROPC hardware every time the destination is implicitly or explicitly loaded.

NOTE *This is probably a design flaw in the ROPC. In some cases the opcount should auto-decrement when we are not loading the destination on frame buffer accesses.*

When the value of the opcount equals the value of the width register, (that is, the first 16-bit word of the raster), the mask1 register is used to mask bits in the destination from modification. A 1 bit in the mask1 register causes the corresponding bit in the decoder output to be given the value of the corresponding bit in the ROPC destination register. Therefore, the destination register must be implicitly loaded from the frame buffer memory on writes which need masking! A 0 bit in the mask1 register causes the corresponding bit in the decoder output to be given the value of the corresponding bit in the three operand RasterOp function output. When the opcount is zero and a destination load occurs, (that is, on the last 16-bit word of the raster line to be written), the mask2 register is used to mask destination data bits from modification. Also, on this last word the opcount is automatically loaded with the value of the width register and the cycle repeats. This mechanism improves graphics performance by allowing the software to set up the end masking and source shifting once for an entire rectangular RasterOp. An unwrapped inner loop can move all the words of the rectangle. There is no need to explicitly load any ROPC registers when moving from one scan line to the next in a raster operation. When the width register is set to zero, both mask1 and mask2 are simultaneously enabled.

### Other RasterOp Registers — Decoderout and x15 Registers

The *decoderout* register is primarily for diagnostic purposes. It holds the value of the output from the ROPC. This value is the result of applying the three operand RasterOp function and the end masking.

The *x15* register is an 8-bit read/write register for any user-defined purpose. It has no effect on the function of the RasterOp chip or Color board.

## 4.3. Color Map

To display the image in the Sun-3 color frame buffer memory, each 8-bit pixel is used as an index into a 256 element color lookup table. Each element of the table is 24 bits; 8 bits drive the red DAC, 8 drive the green, and 8 drive the blue. (For 24-bit color configurations, only the red DAC is used on each board). The color lookup tables consist of a high-speed ECL lookup table used during video display, and a TTL shadow color lookup table that can be accessed at any time by the host software. The TTL shadow mask eliminates the software problems associated with allowing updates of the color map only during vertical blanking.

The map acts like a 256 X 24 RAM bank: the values stored there represent colors. The low-order 8 bytes of each map entry represent red, the medium order byte represents green, and the low-order byte represents blue. The value of the color is proportional to the value in the byte: the higher the value, the higher the color density. No value (24 zeroes) creates black, and all colors (24 ones) creates white.

The ORed value of the *update_ecmap* control bit in the status register and *update_ecmap* control bit in the double buffering register defines the coupling between the host software and the color maps. When the ORed value is zero, the TTL color map can be read or written. When the ORed value is one, the TTL color map cannot be read or written and instead is used to load or unload the ECL color map during vertical retrace.

The *read_ecmap* control bit in the double buffering register controls the transfer direction between the maps. When *update_ecmap* and *read_ecmap* are set, the

ECL color map is transferred to the TTL color map. When *update_ecmap* is set and *read_ecmap* is cleared, the ECL color map is loaded from the TTL color map.

All entries to the color map are on 16-bit boundaries, and are stored consecutively by entry number (0 to 255) and by color (red, green, blue). Only data bits 0 through 7 are used. For example:

```
color map red, entry #0 is at address 0x110001 (data bits D7-D0)
color map red, entry #1 is at address 0x110003 (data bits D7-D0)
color map red, entry #2 is at address 0x110005 (data bits D7-D0)
color map red, entry #3 is at address 0x110007 (data bits D7-D0)

color map red, entry #255 is at addr  0x1101FF (data bits D7-D0)
color map grn, entry #0 is at address 0x110201 (data bits D7-D0)
color map grn, entry #1 is at address 0x110203 (data bits D7-D0)

color map grn, entry #255 is at addr  0x1103FF (data bits D7-D0)
color map blu, entry #0 is at address 0x110401 (data bits D7-D0)
color map blu, entry #1 is at address 0x110403 (data bits D7-D0)

color map blu, entry #255 is at addr  0x1105FF (data bits D7-D0)
```

The programmer should note that the status bits *update_ecmap* are cleared asynchronously but are set synchronously with the start of vertical retrace. Synchronizing the assertion of *update_ecmap* ensures that the entire color map is updated in a single vertical retrace period, and making deassertion of the update bits asynchronous allows programs to deassert the bit, immediately load the TTL map and then reassert *update_ecmap* without requiring any busy-wait polling loops. The status bit *read_ecmap* is always synchronized with the start of vertical retrace.

## 4.4. Interrupt Handling

Interrupts can be generated at the start of vertical retrace. Interrupts are useful for preventing tearing during mouse and pop-up menu activity. They are also very useful for double-buffering applications that wish to toggle foreground/background every *N* frames.

Enable interrupts by setting bit *inten* in the status register. At the start of a vertical retrace period, an interrupt is generated at VME interrupt level 4. During the VME interrupt acknowledge cycle, the board returns the VME interrupt vector stored in the interrupt vector register. The host software can use a unique vector for the color board or it can share that interrupt vector and determine the source of the interrupt by polling the status register bit *inpend*.

To ensure compatibility with Multibus device drivers, the VME interrupt remains pending until the *inten* bit is cleared.

## 4.5. Optional Double Buffering

Some color boards are manufactured with an extra megabyte of frame buffer RAM. This is used as an extra frame buffer, which shares its inputs and outputs with the original, and which is used for double-buffering images. While the contents of one frame buffer is being displayed on the video monitor, the other can be updated.

Bits in double-buffering register control the function of the double buffer. The buffers are designated *buffer a* and *buffer b*, or the *foreground RAM* and the *background RAM*. By changing the bits in the double buffering register, the programmer can toggle the definition of foreground and background, and select which buffer to write to or read from.

The relevant register bits are:

nowrite_a
: When this bit is cleared, writes to double-buffered memory set A are enabled. When this bit is set, writes to double-buffered memory set A are inhibited.

nowrite.b
: When this bit is cleared, writes to double-buffered memory set B are enabled. When this bit is set, writes to double-buffered memory set B are inhibited.

display.b
: This this bit is LOW, it causes the contents of *buffer b* to be displayed on the screen. Otherwise, the contents of *buffer a* are displayed.

read.b
: When this bit is cleared, frame buffer read and "hidden-read" cycles access memory set A. When this bit is set, frame buffer read and "hidden-read" cycles access memory set B.

NOTE *The programming model of the color board assumes that programs not using double buffering write to both buffers. These programs can read from either set, and can display either set, without being affected by programs using double buffering. The Suntools default is to write both sets of RAM and to read from the foreground*

When writing to the background frame buffer only, a process should lock using a null rectangle so that the cursor and menu do not have to be lifted. (Recall that the kernel only lifts the cursor if it is inside the rectangle specified in the call to pw_lock.) This prevents the cursor from flickering when it is in a portion of the screen to which background writes are being made.

Transient objects such as cursors and menus should also only be written to the foreground buffer. In this manner, only the foreground buffer will need to be saved for replacement. This convention applies everywhere on the screen and serves to prevent damage to the background of double buffered windows.

### 6. Optional DMA Window and Frame Count

Double buffered boards come with a circuit to map read and write accesses in the offset range 0x32000-0x3FFFFE to an address defined by a pixel-mode address counter. The pixel-mode address counter is set to an initial value by writing to the *DMA base register*. This counter holds the starting pixel address of the window (upper-left corner) in 16-pixel increments. In other words, the DMA window base equals 16 times the value written to the register.

Read or write accesses into the offset range 0x320000-0x3FFFFF are remapped into the frame buffer: address bits A03 to A01 and LDS and UDS from the VME bus are appended to A19 to A04 from the DMA address counter. The data is then written directly to, or read from the frame buffer. The DMA window transfers are not mapped through the RasterOp chips; they function like the pixel-mode accesses described in the chapter *Software Model*.

After the number of DMA window data transfers equals the value of the write-only DMA width register multiplied by 16, the DMA base is incremented by CG3_WIDTH minus DMA_width_reg*16. This way, consecutive accesses into the DMA window space map properly into a raster on the display.

The vertical height of the DMA window is implicitly set by the number of bytes transferred into or out of the window space. It is the responsibility of the user software to stop the DMA transfer at the end of the DMA window, and reload the DMA base address counter.

## 4.7. 24-bit Color Configuration

Color boards can be configured for 24-bit color by installing three color boards in a system. This option, which requires additional hardware from Sun, allows the system to display all 16 million possible color combinations, instead of the 256 stored in a single color map.

The three boards in a 24-bit color configuration act independently, reside at different addresses, and must be configured in the UNIX *config* file. Only the red DACs and color map entries are used; the 8-bits from the first board generate the red, the 8 bits from the red map in the second board generate green, and the 8 bits from the red map in the third board generate blue. The first board also generates the video sync, vertical reset, a 92.94 MHz master clock, and a 5.81 MHz master clock to the other boards.

In all other aspects, these boards work as described in this manual.

NOTE *The 3-board 24-bit color option requires 8 Amp at -5 Volt even with depopulated DACs and hence may not be a legal configuration in some systems.*

### Installing 24-bit Color

The simplest addressing scheme is to place the boards in the 16MByte addressing space, with the first at **0x400000**, the second at 0x800000, and the third at 0xC00000. On a Sun-2 with limited addressing space, they can also be configured to occupy only 2MBytes of space each, but this will cause the diagnostic to generate spurious errors. On a Sun-3, they can be placed in the 4 Gbyte addressing space, but this requires modifying the existing UNIX driver. For more detail, see the "Engineering Manual" (#800-1366).

The "Sun Configuration Procedures" (#813-2000) show the jumpering required on each board in the single-board (standard) configuration, and in the 3-board,

24-bit color configuration. Note that the 24-bit color configuration requires adding a 5 nsec delay line to the first board, and a 3 nsec delay line to the second board. These are simple LC circuits provided in 3-pin SIPs; stuff pin 3 of the delay line into pin 6 of J100, stuff pin 2 of the delay line into a pad 0.1 inches below pad 8 on J100, and stuff pin #1 of the delay line into a pad 0.2 inches below pad 8 on J100.

# 5

# Programming the Color Board

# Programming the Color Board

This section provides detailed information about the functions and the registers of the Sun-3 Color Graphics board. These include the color map, the status register, and the per-plane RasterOp units.

## 5.1. Mapped I/O Hardware Interface

The files *cg3reg.h* and *memreg.h* contain the following description of the address window to the Sun-3 Color board. This structure defines the way in which the address bits to the Sun-3 color frame buffer are decoded. Once these structs have been memory mapped to a user's virtual space, they may be used to access the color board.

NOTE *The comments in italics are not part of the code.*

```
#define CG3_WIDTH       1152
#define CG3_LINEBYTES   (CG3_WIDTH/8)
#define CG3_HEIGHT      910
#define CG3_DEPTH       8

struct  cg3memfb {              /* non RasterOp frame buffer access */
Word mode accesses
        union bitplane {                        /* word-mode memory */
                long word[CG3_HEIGHT][CG3_WIDTH/(8*sizeof(long))];
                char pad[128*1024];
        } memplane[8];
        union byteplane {                       /* pixel-mode memory */
                u_char pixel[CG3_HEIGHT][CG3_WIDTH];
                char pad[1024*1024];
        } pixplane;
};
```

```
/*
 * Details of the status register.
 */
struct  cg3statusreg {
        unsigned flag15     : 1; /* User/system defineable jumper */
        unsigned flag14     : 1; /* User/system defineable jumper */
        unsigned flag13     : 1; /* User/system defineable jumper */
        unsigned flag12     : 1; /* User/system defineable jumper */
        unsigned flag11     : 1; /* User/system defineable jumper */
        unsigned flag10     : 1; /* User/system defineable jumper */
        unsigned flag09     : 1; /* User/system defineable jumper */
        unsigned resolution : 1; /* screen resolution */
                                 /* 0 = 900 x 1152 */
                                 /* 1 = 1024 x 1024 */
        unsigned retrace    : 1; /* read-only: monitor in retrace */
        unsigned inpend     : 1; /* read-only: interrupt pending */
        unsigned ropmode    : 3; /* RasterOp-mode */
        unsigned inten      : 1; /* enab interrupt at end of retrace */
        unsigned update_ecmap : 1; /* Enable transfer of TTL and ECL cmaps */
                                   /* during vertical retrace. Asyncronous */
                                   /* clear; assertion syncronized to start */
                                   /* of vertical retrace. Silently disables */
                                   /* writing to TTL cmap. ORed with bit by */
                                   /* same name in cg3doublebuf */
        unsigned video_enab : 1; /* enable video output from DACs */
};

/*
 * Details of double buffering register.
                Including bit assignments
 */
struct cg3doublebuf {
        unsigned display_b  : 1; /* Display memory set B or A. Syncronized */
                                 /* to start of vertical retrace */
        unsigned read_b     : 1; /* Read memory set B or A */
        unsigned nowrite_b  : 1; /* Do not update memory set B on writes */
        unsigned nowrite_a  : 1; /* Do not update memory set A on writes */
        unsigned read_ecmap : 1; /* ECL to TTL cmap transfer direction */
                                 /* Syncronized to start of Vretrace. */
        unsigned fast_read  : 1; /* Return invalid data but fast Dtack on rd */
        unsigned wait       : 1; /* Write a ''1'' to set. Bit will remain */
                                 /* high until a full vertical retrace period */
                                 /* has elapsed. The bit clears itself */
        unsigned update_ecmap : 1; /* ORed with cg3statusreg.update_ecmap */
                                   /* Bit duplicated to allow easier */
                                   /* syncronization of color map updates */
                                   /* with double buffering frame toggles */
        reserved7           : 1;
        reserved6           : 1;
        reserved5           : 1;
        reserved4           : 1;
        reserved3           : 1;
```

```
        reserved2           : 1;
        reserved1           : 1;
        reserved0           : 1;
};

/*
 * RasterOp hardware registers.
 */
struct memropc {
        u_short mrc_dest;               /* destination register     */
        u_short mrc_source2;            /* source2 register (left)  */
        u_short mrc_source1;            /* source1 register (right) */
        u_short mrc_pattern;            /* pattern register         */
        u_short mrc_mask1;              /* mask1 register           */
        u_short mrc_mask2;              /* mask2 register           */
        short   mrc_shift;              /* bit 0..3 shift count for source */
                                        /* bit 8    sourceload bit  */
        short   mrc_op;                 /* function                 */
        short   mrc_width;              /* word width               */
        short   mrc_opcount;            /* counts down the width    */
        short   mrc_decoderout;         /* decoder output           */
        short   mrc_x11;                /* manual load destination (diag)*/
        short   mrc_x12;                /* manual load source (diag) */
        short   mrc_x13;
        short   mrc_x14;
        short   mrc_x15;                /* flags register for applications */
};
/*
 * If only ROP-mode accesses are required, cg3fb is the only struct which must
 * be mapped in order to use all Color board features.
 */
                Corresponds to RasterOp memory and control registers
struct  cg3fb {
        union {                                 /* ROP-mode memory */
                union bitplane ropplane[8];     /* word-mode memory with ROP */
                union byteplane roppixel;       /* pixel-mode memory with ROP */
        } ropio;        RasterOp memory — 0x200000 to 0x2FFFFFF

        union {                         /*----- RasterOp chip registers */
                struct memropc ropregs;         /* normal register access */
                struct {
                        char pad[2048];         /* for pixmode src reg prime */
                        struct memropc ropregs; /* byte xfer loads alternate */
                } prime;                        /* src register bits */
                char pad[4096];
        } ropcontrol[9];        RasterOp control regs

        union {                         /*----- status register */
                struct cg3statusreg reg;
                short word;
                char pad[4096];
        } status;
```

```
        union {                         /*----- per-plane mask register */
                unsigned short reg;             /* 8 bit. 1 bit -> wr to plane*/
                char pad[4096];                 /* D07..D00 */
        } ppmask;

        union {                         /*----- double buffering register */
                struct cg3doublebuf reg;
                short word;
                char pad[4096];
        } doublebuf;

        union {                         /*----- dma window orign register */
                unsigned short reg;             /* Bits A19-A4 of DMA window */
                                                /* origin. Write-only. 16-bit */
                char pad[4096];
        } dmabase;

        union {
                unsigned short reg;     /*----- dma window width register */
                                                /* 8-bit. Write-only. */
                                                /* Reg*16 specifies width */
                                                /* of dma window. */
                char pad[4096];
        } width_framecnt;

        union {                         /*----- frame count register */
                unsigned char reg;              /* 8-bit. Even byte. */
                                                /* D15..D08. Read-only. */
                                                /* frame number modulo 256. */
                                                /* Increments on Vretrace. */
                char pad[4096];
        } framecnt;

        union {                         /*----- interrupt vector register */
                unsigned short reg;             /* line number 0..1024/4 */
                char pad[4096];
        } intrptvec;

        unsigned short redmap[256];             /* shadow color maps */
        unsigned short greenmap[256];           /* shadow color maps */
        unsigned short bluemap[256];            /* shadow color maps */
};
/*
 *      ROPMODE                 PARALLEL                LD_DST  LD_SRC   Description
 *                                                      ON      ON                  */
#define PRWWRD  0       /* parallel 8 plane,   read    write,    wrdmode */
#define SRWPIX  1       /* single     pixel,   read    write,    pixmode */
#define PWWWRD  2       /* parallel 8 plane,   write   write,    wrdmode */
#define SWWPIX  3       /* single     pixel,   write   write,    pixmode */
#define PRRWRD  4       /* parallel 8 plane,   read    read,     wrdmode */
#define PRWPIX  5       /* parallel16 pixel,   read    write,    pixmode */
#define PWRWRD  6       /* parallel 8 plane,   write   read,     wrdmode */
#define PWWPIX  7       /* parallel16 pixel,   write   write,    pixmode */
```

```
/*
 * ROP control unit numbers
 */
#define CG3_ROP0        0       /* RasterOp unit for bit-plane 0 */
#define CG3_ROP1        1       /* RasterOp unit for bit-plane 1 */
#define CG3_ROP2        2
#define CG3_ROP3        3
#define CG3_ROP4        4
#define CG3_ROP5        5
#define CG3_ROP6        6
#define CG3_ROP7        7
#define CG3_ALLROP      8       /* writes to all units enabled by PPMASK */
                                /* reads from plane zero ROPC registers  */

#define CG_SRC          0xCC
#define CG_DEST         0xAA
#define CG_MASK         0xf0
#define CG_NOTMASK      0x0f
#define CGOP_NEEDS_MASK(op)             ( (((op)>>4) ^ (op)) & CG_NOTMASK)

/*
 *----------Macros that return addresses into the frame buffer
 */
#define cg3_pixaddr(fb, x, y)                                           \
                (&(fb)->pixplane.pixel[(y)][(x)])
#define cg3_wordaddr(fb, plane, x, y)                                   \
                (&(fb)->memplane[(plane)].word[(y)][(x)>>4])
#define cg3_roppixaddr(fb, x, y)                                        \
                (&(fb)->ropio.roppixel.pixel[(y)][(x)])
#define cg3_ropwordaddr(fb, plane, x, y)                                \
                (&(fb)->ropio.ropplane[(plane)].word[(y)][(x)>>4])
#define cg3_prskew( x)                                                  \
                ((x) & 15)
```

## 5.2. Coordinate System

The frame buffer memory is configured as a memory image and is not directly x,y addressable. The coordinate system starts at the upper left, with y increasing downward and x increasing to the right. In pixel-mode, the Sun-3 color frame buffer appears as one million 8-bit pixels via cg3memfb.pixplane.pixel[y][x]. In word-mode, the Sun-3 color frame buffer appears as a stack of eight 128 Kbyte monochrome frame buffers (cg3memfb.memplane[n].word[y][x/32]). Writes to the frame buffer can be optionally mapped through the RasterOp units via cg3fb.ropio.roppixel.pixel[y][x] or cg3fb.ropio.ropplane[n].word[y][x].

### Pixel Mode

In pixel-mode, pixel number 0 is in the upper-left corner of the frame buffer and pixel number 1151 is in the upper-right corner of the display. Scan line y in the frame buffer ($0 \le y < CG3\_HEIGHT$) always begins at pixel number y*CG3_WIDTH and ends at pixel number (y+1)*CG3_WIDTH - 1.

ord Mode

In word-mode, each scan line in the frame buffer is 36 32-bit words wide (1152 pixels/line divided by 32 pixels/word gives 36 long words/line). The MSB of word number zero for each bit-plane is located in the upper left corner of the frame buffer, and the LSB of word 35 for each bit-plane is located in the upper right corner of the frame buffer. Each successive scan line in the frame buffer occupies another 36 32-bit words.

## 5.3. Programming Examples

The following code sets up the Color board registers in preparation for copying a pixel rectangle having 1 bit per pixel (packed 16 pixels per short word) from main memory to the color frame buffer:

```
/*
 *  Srcskew is pixel address mod 16, ie. offset into a source word
 *  for the upper left corner of the source rectangle.
 *  Ma_top is the address of the first word of the source rectangle.
 */
srcskew = mprs_skew(src);
ma_top = (short*)mprs_addr(src);

/*
 *  Set the per-plane mask register to allow access to all planes.
 *  Set all ROPC op registers to the desired RasterOp function.
 *  Zero all ROPC pattern registers.
 */
fb->ppmask.reg = 0xFF;
fb->ropcontrol[CG3_ALLROP].ropregs.mrc_op = CG_SRC | CG_DEST;
fb->ropcontrol[CG3_ALLROP].ropregs.mrc_pattern = 0;

/*
 *  Set the end mask registers in all ROPCs.  The left mask is based
 *  on the left destination position skew (the left end of the raster),
 *  and the right mask is based on the right destination position skew.
 */
fb->ropcontrol[CG3_ALLROP].ropregs.mrc_mask1 =
        mrc_lmasktable[dst.pos.x & 0xF];
fb->ropcontrol[CG3_ALLROP].ropregs.mrc_mask2 =
        mrc_rmasktable[(dst.pos.x+dst.size.x-1) & 0xF];

/*
 *  The source load bit is 1 so the FIFO is moving source1 to source2.
 *  Therefore, the src and dst addressing will increment left to right.
 *  Set the shift and the sourceload bit in all ROPC shift registers.
 */
dstskew = cg3_prskew( dst.pos.x);
fb->ropcontrol[CG3_ALLROP].ropregs.mrc_shift =
        (1 * 256) | ((dstskew-srcskew) & 0xF);
```

```
/*
 *  The source FIFO must be primed under certain conditions.
 *  Width is the number of word boundaries crossed by a raster line.
 */
prime = (srcskew >= dstskew);
width = (dst.size.x + dstskew -1) / 16;
fb->ropcontrol[CG3_ALLROP].ropregs.mrc_width = width;
fb->ropcontrol[CG3_ALLROP].ropregs.mrc_opcount = width;

/*
 *  Use RasterOp parallel word-mode writes.  Load the dest register
 *  and source FIFO on writes.
 *  Set a pointer to the first word in the frame buffer memory to be
 *  written.
 */
fb->status.reg.ropmode = PWWWRD;
dst_addr = (short*)(&fb->ropio.ropplane[0].
        word[dst.pos.y][dst.pos.x / 16]);

/*
 *  For all raster lines in the rectangle, move all words of the line
 *  using a fastloop macro.  Adding ma_vert to the memory address bumps
 *  the address vertically downward one raster line.
 */
while (dst.size.y--) {
        src_addr = ma_top;          /* src_addr is a word address */
        dx = dst_addr;              /* dx is a word address */
        if (prime)
                fb->ropcontrol[CG3_ALLROP].ropregs.mrc_source1 =
                        *src_addr++;
        rop_fastloop(width, *dx++ = *src_addr++ );
        *dx++ = *src_addr;
        dst_addr += CG3_LINEBYTES;
        (char *)ma_top += ma_vert;
}
```

# Index