

DIVO (Digital Video Option) XIO Board Owner's Guide

Document Number 007-3524-002

CONTRIBUTORS

Written by Carolyn Curtis

Illustrated by Dany Galgani, Chris Pirazzi, Dan Young, Cheri Brown, and
Carolyn Curtis

Production by Kirsten Pekarek

Engineering contributions by Ashok Yerneni, Scott Pritchett, Vince Uttley, Chris
Pirazzi, Ed Miskiewicz, Andrew Khau, Will McCann, and Paul Spencer

St Peter's Basilica image courtesy of ENEL SpA and InfoByte SpA. Disk Thrower
image courtesy of Xavier Berenguer, Animatica.

© 1997, Silicon Graphics, Inc.— All Rights Reserved

The contents of this document may not be copied or duplicated in any form, in whole
or in part, without the prior written permission of Silicon Graphics, Inc.

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure of the technical data contained in this document by
the Government is subject to restrictions as set forth in subdivision (c) (1) (ii) of the
Rights in Technical Data and Computer Software clause at DFARS 52.227-7013
and/or in similar or successor clauses in the FAR, or in the DOD or NASA FAR
Supplement. Unpublished rights reserved under the Copyright Laws of the United
States. Contractor/manufacturer is Silicon Graphics, Inc., 2011 N. Shoreline Blvd.,
Mountain View, CA 94043-1389.

FCC Warning

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

Attention

This product requires the use of external shielded cables in order to maintain
compliance pursuant to Part 15 of the FCC Rules.

DIVO (Digital Video Option) XIO Board Owner's Guide
Document Number 007-3524-002

European Union Statement

This device complies with the European Directives listed on the “Declaration of Conformity” which is included with each product. The CE mark insignia displayed on the device is an indication of conformity to the aforementioned European requirements.



International Special Committee on Radio Interference (CISPR)

This equipment has been tested to and is in compliance with the Class A limits per CISPR publication 22, Limits and Methods of Measurement of Radio Interference Characteristics of Information Technology Equipment; Germany’s BZT Class A limits for Information Technology Equipment; and Japan’s VCCI Class 1 limits.

Canadian Department of Communications Statement

This digital apparatus does not exceed the Class A limits for radio noise emissions from digital apparatus as set out in the Radio Interference Regulations of the Canadian Department of Communications.

Attention

Le présent appareil numérique n’émet pas de perturbations radioélectriques dépassant les normes applicables aux appareils numériques de Classe A prescrites dans le Règlement sur les interferences radioélectriques établi par le Ministère des Communications du Canada.

VCCI Class 1 Statement for Japan

この装置は、情報処理装置等電波障害自主規制協議会（VCCI）の基準に基づくクラス A 情報技術装置です。この装置を家庭環境で使用すると電波妨害を引き起こすことがあります。この場合には使用者が適切な対策を講ずるよう要求されることがあります。

DIVO (Digital Video Option) XIO Board Owner’s Guide
Document Number 007-3524-002

Chinese Class A Warning

警告使用者：

這是甲類的資訊產品，在居住的環境中使用時，可能會造成射頻干擾，在這種情況下，使用者會被要求採取某些適當的對策。

Silicon Graphics, the Silicon Graphics logo, OpenGL, Geometry Engine, and IRIS are registered trademarks and IRIX, XIO, Onyx, Onyx2, Origin, Origin200, Origin2000, Graphics Library, REACT, XFS, and Sirius Video are trademarks of Silicon Graphics, Inc. QuickTime is a registered trademark of Apple Computer, Inc. Abekas is a registered trademark of Carlton International Corporation, Carlton Communications PLC. Gennum is a registered trademark of Gennum, Inc. Videomedia is a registered trademark and V-LAN is a trademark of VideoMedia, Inc.

Contents

List of Figures ix

List of Tables xiii

About This Guide xv

Audience xv

Structure of This Guide xvi

Other Documents xvii

Conventions xvii

- 1. DIVO Features and Capabilities** 1
 - DIVO Features 1
 - DIVO Panel 3
 - Digital Video Ports 6
 - Color-Space Converters 7
 - Interpolation and Decimation Filters 7
 - DIVO Audio 7

- 2. Programming DIVO 9**
 - VL Basics for DIVO 10
 - VL Concepts 11
 - VL Syntax Elements 11
 - VL Object Classes 12
 - VL Nodes for DIVO 13
 - VL Data Transfer Functions 15
 - Compression Through the VL 16
 - DIVO Controls 16
 - Setting Field Dominance 21
 - VL Support for the General-Purpose Interface (GPI) 24
 - Using VL_GPI_OUT_MODE 24
 - Using VL_GPI_STATE 25
 - Using VL_TRANSFER_TRIGGER 26
 - VL Support for the Vertical Inline Time Clock (VITC) 27
 - DIVO Events 27
 - Setting Inline Controls 29
 - Reporting 29
- A. DIVO I/O Panel Connector Specifications 31**
 - DIVO Connectors 31
 - GPI Interface 33
 - GPI Pinouts 33
 - GPI Transmitter 36
 - GPI Receiver, Switch Closure Mode, and Current Sense Mode 38
- B. Setting Up DIVO for Your Video Hardware 41**
 - Setting Up Digital Source Video 41
 - Setting Up the Output (Drain) 43
 - Setting Up Sync 44
 - Setting Up Internal Sync 44
 - Setting Up External Sync 45
 - Saving Settings 46

C.	Pixel Packings and Color Spaces	47
	DIVO Pixel Packings	47
	Packings and Color Spaces	48
	Packing Diagram Conventions	48
	Packings and Library Tokens	50
	Packing Naming Conventions	51
	8-Bit Pixel Packings	53
	16-Bit Pixel Packings	55
	20-Bit Pixel Packings	57
	24-Bit Pixel Packings	58
	32-Bit Pixel Packings	60
	36-Bit Pixel Packing	69
	48-Bit Pixel Packings	70
	64-Bit Pixel Packings	72
	Sampling Patterns	76
	4:4:4 and 4:4:4:4 Sampling	76
	4:2:2 and 4:2:2:4 Sampling	77
	Color Spaces	78
	Determining the Color Space	79
D.	Color-Space Conversions	81
	DIVO Color Spaces	82
	RGB	82
	YUV	83
	CCIR	83
	RP-175 Compressed RGB	84
	Mathematical Operations Performed During Conversions	84
	Implications of Color Space Conversions	85
	Precision of Color Conversions Done by DIVO	85
	Range Issues For Color Conversions Done by Any Means	85
	Example Color Conversions	89
	Example 1: 100% Color Bars	89
	Example 2: Luminance Ramp	93
	Example 3: Simultaneous Chroma/Luma Ramp	97

- E. Programming Methods for Real-Time Digital Media Recording and Playback 101**
 - Direct I/O 102
 - Scatter/Gather I/O 103
 - Multiprocessing 107
 - Asynchronous I/O 108
 - File Formats 108

- F. Diagnostics 111**
 - divo_confidence Functionality 111
 - Running divo_confidence 112
 - divo_confidence Output 114

 - Index 119**

List of Figures

Figure 1-1	DIVO Board Architecture	3
Figure 1-2	DIVO I/O Panel	4
Figure 1-3	DIVO Video Top-Level System Diagram (Onyx2 System)	6
Figure 2-1	Simple VL Path	11
Figure 2-2	Fields and Frames for NTSC and PAL	21
Figure 2-3	Output GPI (OFF Assumed to Be Low)	24
Figure A-1	GPI Connectors	33
Figure A-2	GPI Pinouts	33
Figure A-3	GPI Jumper Locations (Factory Setting)	35
Figure A-4	Example GPI Interface	36
Figure A-5	GPI Transmitter Electrical Specifications	37
Figure A-6	Jumpering for GPI Switch Closure (Factory Setting)	39
Figure A-7	Jumpering for GPI Current Sense Mode	39
Figure B-1	DIVO Ports	41
Figure B-2	Selecting Digital Input Video Format in vcp	42
Figure B-3	Selecting Video Drain Format	43
Figure B-4	Setting Standalone or Genlock Sync	44
Figure B-5	GEN IN Port on the DIVO I/O Panel	45

Figure C-1	VL_PACKING_444_8	49
Figure C-2	VL_PACKING_4_8	53
Figure C-3	VL_PACKING_R444_332	54
Figure C-4	VL_PACKING_444_332	54
Figure C-5	VL_PACKING_242_8	55
Figure C-6	VL_PACKING_R242_8	55
Figure C-7	VL_PACKING_X4444_5551	56
Figure C-8	VL_PACKING_444_5_6_5	56
Figure C-9	VL_PACKING_242_10	57
Figure C-10	VL_PACKING_R242_10	57
Figure C-11	VL_PACKING_444_8	58
Figure C-12	VL_PACKING_R444_8	59
Figure C-13	VL_PACKING_4444_6	60
Figure C-14	VL_PACKING_4444_8	61
Figure C-15	VL_PACKING_R4444_8	62
Figure C-16	VL_PACKING_R0444_8	63
Figure C-17	VL_PACKING_0444_8	64
Figure C-18	VL_PACKING_4444_10_10_10_2	65
Figure C-19	VL_PACKING_2424_10_10_10_2Z	66
Figure C-20	VL_PACKING_R2424_10_10_10_2Z	66
Figure C-21	VL_PACKING_242_10_in_16_L	67
Figure C-22	VL_PACKING_242_10_in_16_R	67
Figure C-23	VL_PACKING_R242_10_in_16_L	68
Figure C-24	VL_PACKING_R242_10_in_16_R	68

Figure C-25	VL_PACKING_444_12	69
Figure C-26	VL_PACKING_4444_12	70
Figure C-27	VL_PACKING_444_10_in_16_L	71
Figure C-28	VL_PACKING_4444_10_in_16_L	72
Figure C-29	VL_PACKING_4444_10_in_16_R	73
Figure C-30	VL_PACKING_4444_12_in_16_L	73
Figure C-31	VL_PACKING_4444_12_in_16_R	74
Figure C-32	VL_PACKING_4444_13_in_16_L	74
Figure C-33	VL_PACKING_4444_13_in_16_R	75
Figure C-34	4:4:4 Sampling	76
Figure C-35	4:2:2 Sampling	77
Figure D-1	RGB Cube in CCIR Space	87
Figure D-2	Color Cube With Luminance/Chrominance Ramp Vector	88
Figure D-3	100% Color Bars: Cr/R	90
Figure D-4	100% Color Bars: Y/G	91
Figure D-5	100% Color Bars: Cb/B	92
Figure D-6	Luminance Ramp: Cr/R	94
Figure D-7	Luminance Ramp: Y/G	95
Figure D-8	Luminance Ramp: Cb/B	96
Figure D-9	Chroma/Luma Ramp: Cr/R	98
Figure D-10	Chroma/Luma Ramp: Y/G	99
Figure D-11	Chroma/Luma Ramp: Cb/B	100
Figure F-1	/usr/diags/DIVO Contents	112

List of Tables

Table 1-1	Interface for Video Equipment	4
Table 1-2	DIVO Panel LEDs	5
Table 2-1	DIVO Node Controls	17
Table 2-2	Controls for DIVO	18
Table 2-3	DIVO Events	28
Table A-1	Return Loss for DIVO Video and Genlock Channels	31
Table A-2	Characteristics for DIVO Digital Video Out Channels	31
Table A-3	Usage for LINK A and LINK B in 4:2:2:4 Mode	32
Table A-4	Usage for LINK A and LINK B in 4:4:4:4 Mode	32
Table A-5	GPI Pinouts	34
Table A-6	GPI Transmitter Electrical Specifications	37
Table A-7	GPI Receiver Input Optoisolator	38
Table C-1	DIVO Packings	51
Table C-2	VL_COLORSPACE Options	79
Table D-1	Clamping Ranges for RGB Component Conversions	82
Table F-1	divotest Output Identifiers	114

About This Guide

The Digital Video Option (DIVO) board is a video option that provides Onyx 2, Origin200, and Origin2000 workstations and servers with broadcast-quality video. The option also provides 16 channels of audio.

Note: This option requires IRIX 6.4 or later.

Features of the DIVO option are controlled with the Video Library (VL) and the Audio Library (AL). VL device-independent calls and controls are explained in the *Digital Media Programming Guide* (007-1799-060 or later; online only). That manual also gives information on using the AL.

Audience

This guide was written for the sophisticated video user in a professional or research environment. You should be familiar with video standards, the operation of the Onyx2, Origin200, or Origin2000 workstation or server, and the VL information in the *Digital Media Programming Guide*.

Many current Silicon Graphics owner's guides, programming guides, and user's guides are available through the World Wide Web: <http://techpubs.sgi.com/library>.

Structure of This Guide

This guide includes the following chapters and appendices:

- Chapter 1, “DIVO Features and Capabilities,” outlines the main components of the DIVO option.
- Chapter 2, “Programming DIVO,” describes using the VL to accomplish common specific tasks.
- Appendix A, “DIVO I/O Panel Connector Specifications,” summarizes technical specifications for the DIVO board.
- Appendix B, “Setting Up DIVO for Your Video Hardware,” describes connecting video equipment to DIVO board connectors and using the control panel *vcp* to configure the DIVO board for the equipment.
- Appendix C, “Pixel Packings and Color Spaces,” sets forth all packing formats used by the DIVO hardware.
- Appendix D, “Color-Space Conversions,” explains DIVO color spaces, the mathematical operations performed during conversions, and the implications of color space conversions.
- Appendix E, “Programming Methods for Real-Time Digital Media Recording and Playback,” explains programming concepts, such as real-time disk I/O, and gives examples.
- Appendix F, “Diagnostics,” explains use of the diagnostic script.

An index completes this guide.

Other Documents

The *Digital Media Programming Guide* is available with the IRIX digital media development environment software (*dmedia_dev*). This guide is also online in the following locations:

- IRIS InSight Library: From the Toolchest, choose Help > Online Books > SGI EndUser or SGI Admin, and select the applicable owner's or hardware guide. Once you are in the library, choose Catalogs > Hardware Catalog and look under Owner's Guides for the applicable owner's guide.
- Technical Publications Library: If you have access to the Internet, enter the following URL in your Web browser location window:
<http://techpubs.sgi.com/library>

Conventions

In command syntax descriptions and examples, square brackets ([]) surrounding an argument indicate an optional argument. Variable parameters are in *italics*. Replace these variables with the appropriate string or value.

In text descriptions, IRIX filenames are in *italics*.

Helvetica Bold font is used for labels on hardware, such as the names of ports on the I/O panel.

Messages and prompts that appear on-screen are shown in typewriter font. Entries that are to be typed exactly as shown are in **boldface typewriter font**.

In each chapter or appendix in which it occurs, the *Digital Media Programming Guide* is referred to by its full title at the first occurrence and thereafter as the DMPG.

DIVO Features and Capabilities

DIVO is a video option for Onyx2 graphics, Origin200, and Origin2000 deskside and server workstations. Supporting the SMPTE 259 10-bit digital video standard, it fully integrates video into Silicon Graphics workstation and server environments.

The option utilizes one XIO slot and provides independent dual-link 10-bit serial digital component input and output ports. Depending on the system type, multiple DIVO boards can be installed on a system for video server applications.

This chapter discusses

- “DIVO Features” on page 1
- “DIVO Panel” on page 3
- “Digital Video Ports” on page 6
- “Color-Space Converters” on page 7
- “Interpolation and Decimation Filters” on page 7
- “DIVO Audio” on page 7

DIVO Features

DIVO features include

- dual-link 10-bit serial digital video (SMPTE 259) streaming to and from system memory
- AES3-1992 (AES/EBU) embedded audio and ancillary data (SMPTE 272M) with up to 24-bit precision and sample rates of 32 KHz, 44.1 KHz, and 48 KHz; up to 16 channels output and 16 channels input
- lossless built-in compression and decompression using adaptive entropy coding to approximately 2:1 compression
- transparent color-space conversion between YUV and RGB

- flexible data-packing capability to facilitate easy integration to OpenGL component packing methods
- unadjusted system time/media stream count (UST/MSC) hardware-supported audio/video synchronization mechanisms
- low latency in video transfers to/from system memory (typically less than one frame)
- support for fields and frames
- active video or data mode capture and playback
- vertical internal time clock (VITC) extraction and insertion
- hardware error detection and handling (EDH) and link autophasing
- support for SDDI/CSDI interfaces
- for each video pipe, two channels of input and output trigger signal pairs based on the general-purpose interface (GPI)
- output genlocking

The DIVO board supports video and audio data transfers to and from system memory only. You can view the video in real time on the Onyx2 workstation using the OpenGL interface to copy video images to graphics. DIVO supports a wide variety of packing formats to facilitate easy integration of video in graphics.

To capture graphics to video, you can use OpenGL to read pixels into memory. However, the coordinate system differs between video and Open GL; under Open GL, the origin is at the lower left corner and in Video, origin is in the upper left corner. To adjust for this difference, the image must be inverted in memory before it is sent to the DIVO board for output.

Besides sending an image out to the DIVO board, you can use the GVO graphics option to get zero latency transcoding to CCIR 601-2 digital video.

For controlling videotape recorders, you can use a direct RS-422 connection to the deck with third-party software, or an RS-422 V-LAN controller option and V-LAN software from Silicon Graphics with the on-board GPI triggering mechanism.

DIVO is fully integrated into the Silicon Graphics Digital Media Library interfaces. The Video Library (VL) API has been enhanced to support some of its advanced features.

DIVO Panel

Figure 1-1 shows the top-level diagram of DIVO board. Each DIVO board has two independent pipes, each with its own dedicated R4650 processor and SDRAM.

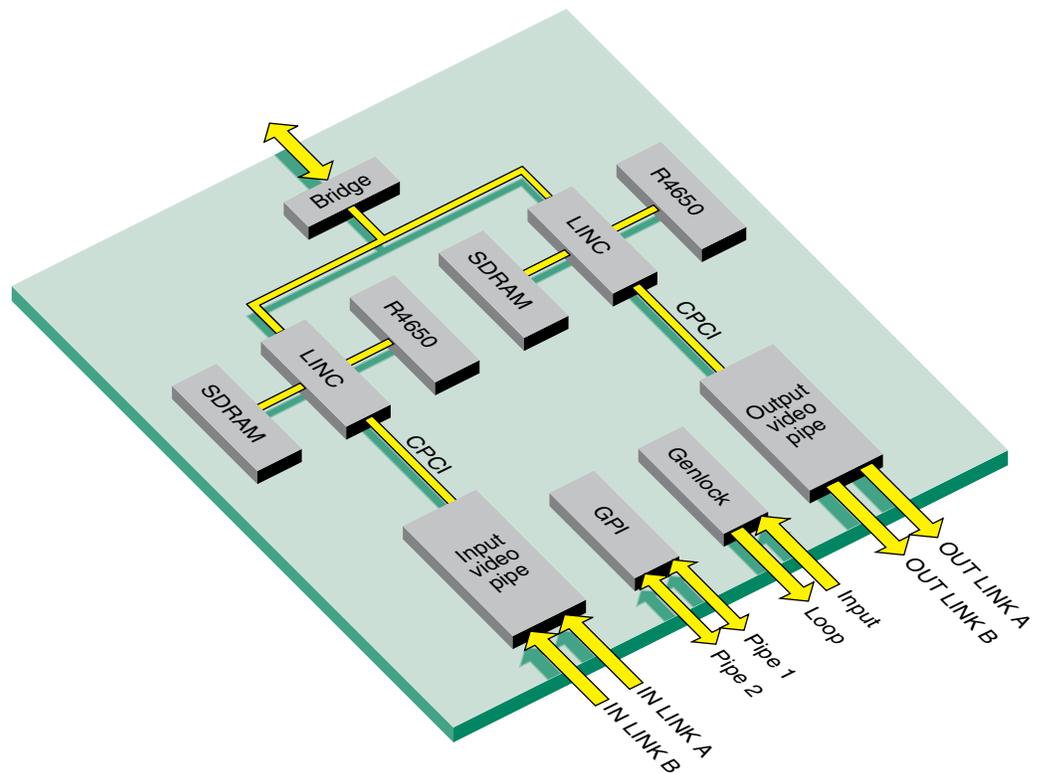


Figure 1-1 DIVO Board Architecture

Figure 1-2 shows features of the DIVO I/O panel. Although the board is installed vertically in the chassis, Figure 1-2 shows the panel sideways to aid in reading the connector and LED labels.

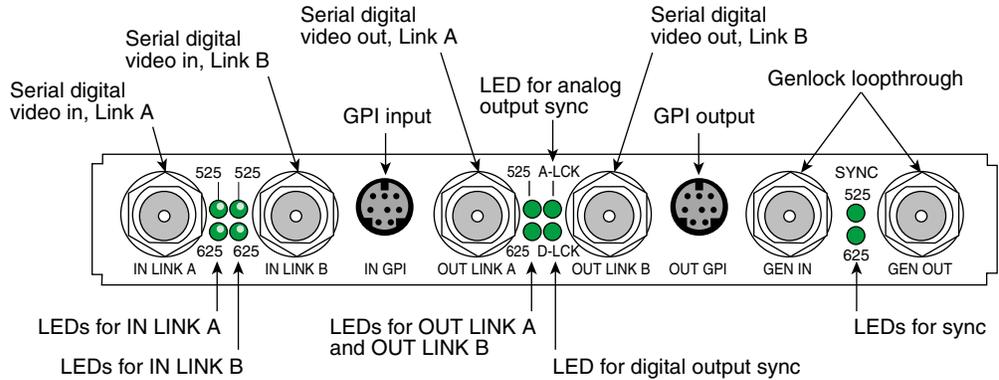


Figure 1-2 DIVO I/O Panel

Table 1-1 summarizes DIVO board external connectors that interface with video equipment.

Table 1-1 Interface for Video Equipment

Connector	Format	Use
IN LINK A, IN LINK B	10-bit CCIR 601-2 75-ohm BNCs Terminated, unbalanced	Serial digital video input from digital tape deck or other recording device. Conforms to SMPTE 259M for component video, SMPTE 272M for embedded audio, and SMPTE 266M for DVITC. Both inputs autophased.
OUT LINK A, OUT LINK B	10-bit CCIR 601-2 75-ohm BNCs	Serial digital video output to digital tape deck or other recording device. Conforms to SMPTE 259M for component video, SMPTE 272M for embedded audio, and SMPTE 266M for DVITC. Note: The transfer mode (packing format) selected determines LINK A and LINK B usage, as explained in Table A-3 and Table A-4 in Appendix A, "DIVO I/O Panel Connector Specifications."
GEN IN	75-ohm BNC Loopthrough, unbalanced, unterminated	External analog sync source (precision time base or other source of house sync) or analog loopthrough.

Table 1-1 (continued) Interface for Video Equipment

Connector	Format	Use
GEN OUT	75-ohm BNC Loophthrough, unbalanced, unterminated	External reference loop out; passive loophthrough for genlock input with buffered signal to workstation. Note: If you attach a cable to one GEN connector, you must attach either another cable to other equipment accepting analog sync or a 75-ohm BNC terminator to the other GEN connector.
GPI IN, GPI OUT	8-pin mini-DIN	General Purpose Interface for each video port; frame-accurate event triggering to or from source or destination (tape deck or digital recorder). Configurable for switch closure (factory setting) or current sense operation.

See Appendix A, "DIVO I/O Panel Connector Specifications," for technical details of the connectors, including GPI pinouts.

Table 1-2 summarizes the function of the LEDs on the panel.

Table 1-2 DIVO Panel LEDs

LED	Purpose
Leftmost LEDs between IN LINK A and IN LINK B (525 and 625)	Top LED lights when valid 525-line serial digital signal detected on IN LINK A . Bottom LED lights when valid 625-line serial digital signal detected on IN LINK A .
Rightmost LEDs between IN LINK A and IN LINK B (525 and 625)	Valid 525-line (top LED) or 625-line (bottom LED) serial digital signal detected on IN LINK B .
Leftmost LEDs between OUT LINK A and OUT LINK B (525 and 625)	Valid 525-line (top LED) or 625-line (bottom LED) serial digital signal detected on OUTLINK A and OUT LINK B ; these outputs are locked together, regardless of whether OUT LINK B is used.
A-LCK	Output is locked to an analog source: standalone, genlock (to another sync source), or free run.
D-LCK	Output is locked to a digital source (IN LINK A or IN LINK B).
SYNC 525 and 625	Valid 525-line sync source (top LED) or 625-line sync source (bottom LED) detected.

Figure 1-3 diagrams how the DIVO board interacts with other workstation components.

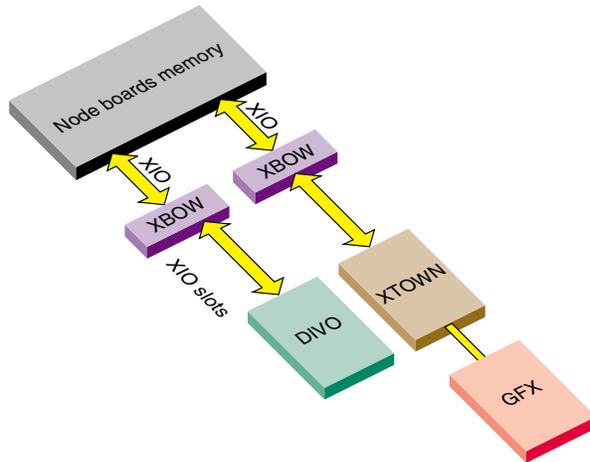


Figure 1-3 DIVO Video Top-Level System Diagram (Onyx2 System)

Digital Video Ports

The DIVO board has two independent 10-bit serial digital video ports for equipment that complies with the CCIR 601-2 standard. The ports can be configured for 4:4:4:4 or 4:2:2:4 in dual-link mode, or 4:2:2 in single-link mode where alpha is ignored.

Each port consists of two unidirectional interconnections, Link A and Link B:

- In 4:4:4:4 mode, Link A carries Y plus Cr and Cb from even-numbered sample points; Link B carries alpha plus Cr and Cb from odd-numbered sample points.
- In 4:2:2:4 mode, Link A carries Y plus Cr and Cb; Link B carries alpha only.

The video format selected determines Link A and Link B usage. For more information, see the following standards, which contain provisions for video signals:

- CCIR 601-2: Encoding Parameters of Digital Television for Studios (4:2:2 component video signals, single link)
- ANSI/SMPTE 125M-1992: Television—Component Video Signal 4:2:2—Bit-Parallel Digital Interface

- SMPTE Recommended Practice (RP) 175-1993: Digital Interface for 4:4:4:4 Component Video Signals (Dual Link)
- SMPTE 259M-1993: Television—10-Bit 4:2:2 Component and $4f_{sc}$ NTSC composite Digital Signals—Serial Digital Interface
- SMPTE RP 157-1990: Key Signals
- SMPTE 272M: Television - Formatting AES/EBU Audio and Auxiliary Data into Digital Video Ancillary Data Space

Color-Space Converters

Four color spaces are native to DIVO: full-range RGBA, compressed range RGB (RP-175), CCIR601, and full-range YUV. The video interface supports only RP-175 and CCIR 601-2. The memory interface supports all four color spaces. See “Color Spaces” in Appendix C for an explanation of full- and compressed-range color spaces.

DIVO uses the Genum GF9105 component digital transcoder. The GF9105 uses 13-bit multiplier coefficients and provides up to 13-bit output resolution, allowing for transparent color-space conversion between YUV and RGB.

Interpolation and Decimation Filters

The GF9105 transcoder also provides interpolation and decimation filtering between the 4:2:2:4 and 4:4:4:4 sampling rates. Both interpolation and decimation operations are fully compliant with the CCIR 601-2 standard.

DIVO Audio

DIVO provides one 16-channel wide input device and one 16-channel wide output device, which are compliant with the SMPTE 272M standard. Applications can open either device as 2, 4, 8, or 16 channels. Unused output channels are set to zero; unused input channels are discarded. DIVO audio supports sample rates of 32 KHz, 44.1 KHz, and 48 KHz. It supports 20- and 24-bit word sizes. Access to the audio is through the Audio Library (AL) interfaces specified in the DMPG.

Programming DIVO

The DIVO board supports the Video Library (VL) and the Audio Library (AL) programming APIs. The APIs are described in the *Digital Media Programming Guide* (007-1799-060 or later; hereafter referred to as the DMPG).

This chapter explains

- “VL Basics for DIVO” on page 10
- “Compression Through the VL” on page 16
- “DIVO Controls” on page 16
- “Setting Field Dominance” on page 21
- “VL Support for the General-Purpose Interface (GPI)” on page 24
- “VL Support for the Vertical Inline Time Clock (VITC)” on page 27
- “DIVO Events” on page 27
- “Setting Inline Controls” on page 29
- “Reporting” on page 29

VL Basics for DIVO

To build programs that run under VL, you must

- install the *dmedia_dev* and *dmedia_eoe* options
- link with *libvl*
- include *dmedia/vl.h* and *dmedia/vl_DIVO.h* for device-dependent functionality

The client library for VL is */usr/lib32/libvl.so*. The header files for the VL are in */usr/include/dmedia*; the main file is *vl.h*. This file contains the main definition of the VL API and controls that are common across all hardware. Several useful digital media programming examples are in */usr/share/src/dmedia/tools*.

Note: When building a VL-based program, you must add *-lvl* to the linking command.

For more information on the Video Library and the API usage, see the latest version of the DMPG.

This section explains

- “VL Concepts” on page 11
- “VL Syntax Elements” on page 11
- “VL Object Classes” on page 12
- “VL Nodes for DIVO” on page 13
- “VL Data Transfer Functions” on page 15

VL Concepts

The Video Library defines a basic set of primitives and mechanisms to specify interconnections and controls to achieve the desired setup. The two central concepts for VL are

- *path*: an abstraction for a way of moving data around
- *node*: an endpoint of the path

The basic nodes are a *source* (such as a VTR) and a *drain* (such as memory). Figure 2-1 diagrams the simplest VL path, with one of each of these two nodes.

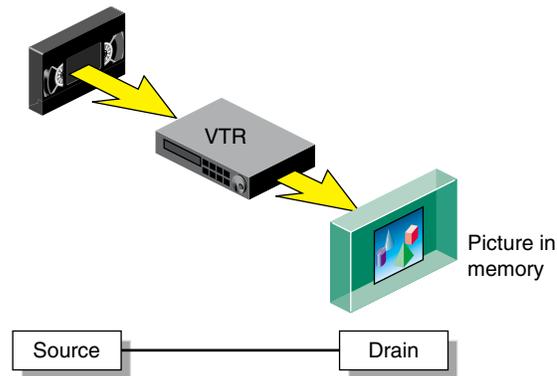


Figure 2-1 Simple VL Path

Another type of node besides source and drain is the *device* node (a global control), such as trigger, GPI, sync, or default source. DIVO nodes are further discussed in “VL Nodes for DIVO” on page 13.

VL Syntax Elements

VL syntax elements are as follows:

- VL types and constants begin with uppercase VL; for example, VLServer
- VL functions begin with lowercase vl; for example, `vlOpenVideo()`

VL Object Classes

The VL recognizes these classes of objects:

- *devices*, each including sets of nodes
- *nodes*, which are sources, drains, and internal nodes (as discussed in the preceding section)
- *paths*, connecting sources and drains (as discussed in the preceding section)
- *buffers*, for sending and receiving field/frame data to and from host memory

DIVO requires the use of DMbuffers (digital media buffers) and not the original ring buffer mechanisms (VL buffers) used with earlier Silicon Graphics video options. The new buffering scheme is much more flexible and versatile than the older VL buffer-based scheme. See Chapter 5 of the DMPG.

DMbuffers, an abstraction of main memory, allow efficient and API-independent interchange of data between the different digital media libraries. For example, video fields can be captured into DMbuffers via VL and then displayed in graphics using OpenGL. They can also be passed between two processes without the data having to be copied explicitly. Refer to Chapter 5, “Digital Media Buffers,” in the DMPG for details.

- *events*, for monitoring video I/O status
- *controls*, or parameters that modify how data flows through nodes; for example:
 - video device parameters, such as sync source
 - video data parameters such as packing, size, and color space

VL controls fall into two categories:

- *device-global* or *device-independent* (prefix VL_), which can be used by several Silicon Graphics video products

For details of the device-independent controls, refer to the DMPG.

- *device-dependent* (prefix VL_DIVO_), specific to a particular video device, in this case, DIVO

Both types of VL controls are explained in this chapter with respect to their usage with DIVO.

VL Nodes for DIVO

Use **vlGetNode()** to specify nodes. This call returns the node's handle, which is used when setting controls or setting up paths. Its function prototype is:

```
VLNode vlGetNode(VLServer svr, int type, int kind, int number)
```

In this prototype, variables are as follows:

<i>svr</i>	Names the server (as returned by vlOpenVideo()).
<i>type</i>	Specifies the type of node: <ul style="list-style-type: none"> VL_SRC: source, such as a digital tapedeck connected to an input port <p>Note: The DIVO option has only one input.</p> <ul style="list-style-type: none"> VL_DRN: drain, such as system memory VL_DEVICE: global control, such as a default source; Table 2-1 summarizes the values for this type <p>Note: If you are using VL_DEVICE, the <i>VLNode</i> should be set to 0.</p>
<i>kind</i>	Specifies the kind of node: <ul style="list-style-type: none"> VL_VIDEO: connection to a video device equipment; for example, a video tapedeck or camera VL_MEM: workstation memory
<i>number</i>	Number of the node in cases of two or more identical nodes, such as two video source nodes. The default value for all <i>kinds</i> is 0.

VL_ANY can also be used as a value for *number* to reference the first available node of the specified *type* and *kind*.

In general, a path on DIVO has a memory node and a video node. The following fragment creates a digital video input source node and a memory drain node, and creates the path:

```
VLServer svr;
VLPath path;
VLNode src;
VLNode drn;

/*Set up video source node */
VLControlValue timing,format, val;
src = vlGetNode(svr, VL_SRC, VL_VIDEO, VL_ANY);
/*Set up memory drain node */
drn = vlGetNode(svr, VL_DRN, VL_MEM, VL_ANY);
/* Create source-to-drain path */
if((path = vlCreatePath(svr, VL_ANY, src, drn)) < 0){
    fprintf(stderr,"%s\n",vlStrError(vlGetErrno()));
    exit(1);
}

/* Set up path with shared src and drn node */
vlSetupPaths(svr, (VLPathList)&path, 1, VL_SHARE, VL_SHARE);
```

After nodes are specified, use **vlSetControl()** to specify parameters:

- video nodes: video timing (drain only) and format (for example digital component)
- memory nodes: timing, packing, and color space

Controls for each node are defined in “DIVO Controls” later in this chapter, and are summarized in Table 2-3.

VL Data Transfer Functions

This section summarizes VL data transfer categories, and gives the basic steps of creating an application. For DIVO, VL data transfers always involve memory (video to memory, memory to video) and require setting up a DMbuffer pool.

In the VL programming model, the process of creating a VL application consists of these steps:

1. Open a connection to the video daemon (**vlOpenVideo()**).
2. Specify nodes on the data path (**vlGetNode()**).
3. Create the path (**vlCreatePath()**).
4. Optional step: add more nodes to a path (**vlAddNode()**).
5. Set up the hardware for the path (**vlSetupPaths()**).
6. Specify path-related events to be captured (**vlSelectEvents()**, **vlAddCallback()**).
7. Set input and output parameters (controls) for the nodes on the path (**vlSetControl()**); video format and timing must be specified.
8. Create a dmBuffer pool to hold data for memory transfers (**vlDMGetParams()**, **dmBufferSetPoolDefaults()**, **dmBufferCreatePool()**, **vlGetTransferSize()**).
9. Register the buffer (**vlDMPoolRegister()**, **vlDMPoolDeregister()**).
10. Start the data transfer (**vlBeginTransfer()**).
11. Get the data (**vlDMBufferGetValid()**, **vlDMBufferPutValid()**, **dmBufferAllocate()**, **dmBufferAllocateSize()**, **dmBufferGetPoolState()**, **dmBufferGetPoolFD()**, **dmBufferSetPoolSelectSize()**, **dmBufferMapData()**, **dmBufferFree()**) to manipulate frame data.
12. Handle data stream events (**vlSelectEvents()**, **vlNextEvent()**, **vlPending()**).
13. Clean up (**vlEndTransfer()**, **vlDMPoolDeregister()**, **vlDestroyPath()**, **vlCloseVideo()**).

Note: Error handling (**vlPerror()**) is accomplished throughout.

Compression Through the VL

Compression is handled via enhancements to the VL API; the DIVO board does not support the Compression Library API.

Compression in the VL is supported by adding compression-related controls on memory nodes. The control `VL_COMPRESSION` specifies the compression. Based on the compression type, additional controls specify compression-related parameters. Table 2-2 later in this chapter summarizes these controls.

DIVO can use Rice compression, a lossless entropy coding mechanism that provides an average compression of 2:1. In some cases, compression can add to the size of the data being transferred, but is limited to a maximum of 1% increase. Tests at Silicon Graphics show that this compression mechanism can reduce data in ratios of 2:1 to 6:1.

Please refer to the source examples (*divo_vidtomem.c* for encoding and *divo_memptovid.c* for decoding) in the directory `/usr/share/src/dmedia/video/DIVO`. For encoding, specify the controls `VL_COMPRESSION`, `VL_RICE_COMP_PRECISION`, and `VL_RICE_COMP_SAMPLING`. For decoding, the control information is already embedded in the stream; you need specify only the data type, by using the `VL_COMPRESSION` control. For best memory utilization with Rice compression, use variably sized buffers.

DIVO Controls

To determine the available devices (that is, video options in the workstation, such as the DIVO board) and the nodes available on them, run *vlinfo*. To determine possible controls for each device, enter

```
vlinfo -1
```

Note: VL controls specified as true with `vlSetControl()` are executed immediately. However, they are not guaranteed to happen at a specific time. For better precision on the execution of these controls, see “Setting Inline Controls” on page 29.

To set controls for DIVO nodes, use **vlSetControl()**. The following example sets video format and timing on a node:

```

timing.intVal = VL_TIMING_525_CCIR601;
format.intVal = VL_FORMAT_RGB;

if (vlSetControl(svr, path, drn, VL_TIMING, &timing) <0)
{
    vlPerror("VlSetControl:TIMING");
    exit(1);
}
if (vlSetControl(svr, path, drn, VL_FORMAT, &format) <0)
{
    vlPerror("VlSetControl:FORMAT");
    exit(1);
}

```

For details on **vlSetControl()** and **vlGetControl()**, see the latest version of the DMPG.

Tables in this section summarize

- device-global controls for DIVO
- controls for DIVO nodes
- control values and uses

Table 2-1 summarizes supported node controls for DIVO.

Table 2-1 DIVO Node Controls

Control	Video Source	Memory Source	Video Drain	Memory Drain
VL_ASPECT		X (read only)		X (read only)
VL_CAP_TYPE		X		X
VL_COLORSPACE		X		X
VL_COMPRESSION		X		X
VL_DIVO_LOOPBACK	X			
VL_DIVO_RASTER_MODE		X		X
VL_FIELD_DOMINANCE	X		X	
VL_FORMAT	X		X	

Table 2-1 (continued) DIVO Node Controls

Control	Video Source	Memory Source	Video Drain	Memory Drain
VL_GPI_OUT_MODE	X		X	
VL_GPI_STATE	X		X	
VL_OFFSET	X (read only)	X	X (read only)	X
VL_PACKING		X		X
VL_RICE_COMP_PRECISION		X		X
VL_RICE_COMP_SAMPLING		X		X
VL_SIZE	X (read only)	X	X (read only)	X
VL_SYNC			X	
VL_SYNC_SOURCE			X	
VL_TIMING	X (read only)	X	X	X
VL_TRANSFER_TRIGGER	X		X	
VL_ZOOM	X (read only)	X (read only)	X (read only)	X (read only)

Table 2-2 summarizes the values and uses of controls for DIVO.

Table 2-2 Controls for DIVO

Control	Values or Range	Use
VL_ASPECT	Aspect (read-only).	Reads aspect ratio.
VL_CAP_TYPE	Memory nodes: VL_CAPTURE_FIELDS VL_CAPTURE_INTERLEAVED VL_CAPTURE_NONINTERLEAVED	Selects type of frame(s) or field(s) to capture.
VL_COLORSPACE	VL_COLORSPACE_RGB (full-range RGB) VL_COLORSPACE_CCIR601 (compressed-range YUV) VL_COLORSPACE_RP175 (compressed-range RGB) VL_COLORSPACE_YUV (full-range YUV)	Specifies color space of video data in memory.

Table 2-2 (continued) Controls for DIVO

Control	Values or Range	Use
VL_COMPRESSION	VL_COMPRESSION_NONE VL_COMPRESSION_RICE VL_COMPRESSION_JPEG VL_COMPRESSION_MPEG2 VL_COMPRESSION_DVCPRO	Specifies compression option for video. See <i>vl.h</i> for information on compression-specific controls. For example, to access Rice entropy coding, use VL_COMPRESSION_RICE as the compression control for the memory node.
VL_DIVO_LOOPBACK	VL_DIVO_LOOPBACK_ON VL_DIVO_LOOPBACK_OFF	Specifies if the video source on input should be from the output pipe.
VL_DIVO_RASTER_MODE	VL_DIVO_RASTER_VIDEO VL_DIVO_RASTER_DATA	Grabs video specified by the VL_SIZE and VL_OFFSET. Grabs or puts the full raster; used for SDDI or CSDI interfaces.
VL_FIELD_DOMINANCE	VL_F1_IS_DOMINANT VL_F2_IS_DOMINANT Note: Frames that are output are deinterlaced differently depending on the choice of output field dominance. Deinterlacing is specified in the application.	Identifies frame boundaries in a field sequence; see “Setting Field Dominance.”
VL_FORMAT	VL_FORMAT_DIGITAL_COMPONENT_SERIAL VL_FORMAT_DIGITAL_COMPONENT_DUAL_SERIAL	Sets video format in or out: Serial 4:2:2:4 Serial 4:4:4:4
VL_GPI_OUT_MODE	Conditions: VL_GPI_OUT_XFER_START VL_GPI_OUT_XFER_STOP	Specifies when the GPI_OUT line is asserted, to control downstream devices in a studio environment. For more information, see “VL Support for the General-Purpose Interface (GPI).” Asserts GPI_OUT at BeginTransfer. Asserts GPI_OUT at EndTransfer.
VL_GPI_STATE	State: VL_GPI_CLEAR VL_GPI_OFF VL_GPI_ON VL_GPI_PULSE (transition for one field time)	Sets/Gets the state of output_gpi lines. For more information, see “VL Support for the General-Purpose Interface (GPI)” on page 24.

Table 2-2 (continued) Controls for DIVO

Control	Values or Range	Use
VL_OFFSET	Any position within the video raster.	Sets the position within the video raster to stuff bits.
VL_PACKING	Supported packings; see Appendix C, “Pixel Packings and Color Spaces,” for information.	Sets packing format for memory source or drain node.
VL_RICE_COMP_PRECISION	VL_RICE_COMPRESSION_8 VL_RICE_COMPRESSION_10 VL_RICE_COMPRESSION_12 VL_RICE_COMPRESSION_13	Specifies the component size.
VL_RICE_COMP_SAMPLING	VL_RICE_COMPRESSION_422 VL_RICE_COMPRESSION_4224 VL_RICE_COMPRESSION_444 VL_RICE_COMPRESSION_4444	Specifies the sampling resolution.
VL_SIZE	Any size of the raster.	Size plus offset or origin should not exceed the raster dimensions.
VL_SYNC	VL_SYNC_INTERNAL VL_SYNC_GENLOCK	Sets sync mode for source or drain; on source, this is set to VL_SYNC_GENLOCK.
VL_SYNC_SOURCE	VL_DIVO_SYNC_HOUSE VL_DIVO_SYNC_DIGITAL_INPUT_LINK_A VL_DIVO_SYNC_DIGITAL_INPUT_LINK_B VL_DIVO_SYNC_DBOARD	Selects the genlock source: VL_DIVO_SYNC_DBOARD is used when the compressed stream provides a sync source.
VL_TIMING	VL_TIMING_525_CCIR601 VL_TIMING_625_CCIR601	Sets or gets video timing: 13.50 MHz, 720 x 486 13.50 MHz, 720 x 576
VL_TRANSFER_TRIGGER	VL_TRIGGER_NONE VL_TRIGGER_GPI VL_TRIGGER_VITC VL_TRIGGER_MSC	Specifies the conditions under which transfers begin on a path (video nodes only); see “Using VL_TRANSFER_TRIGGER” in this chapter.
VL_ZOOM	Zoom factor (read-only)	Reads zoom factor of video stream.

Setting Field Dominance

Field dominance identifies the frame boundaries in a field sequence; that is, it specifies which pair of fields in a field sequence constitute a frame. The control VL_FIELD_DOMINANCE allows you to specify whether an edit occurs on the nominal video field boundary (field 1) or on the intervening field boundary (field 2).

- F1 dominant: The edit occurs on the nominal video field boundary.
- F2 dominant: The edit occurs on the intervening field boundary.

Figure 2-2 shows fields as defined for NTSC and PAL.

Note: In digital formats and other input modes, half lines become full lines.

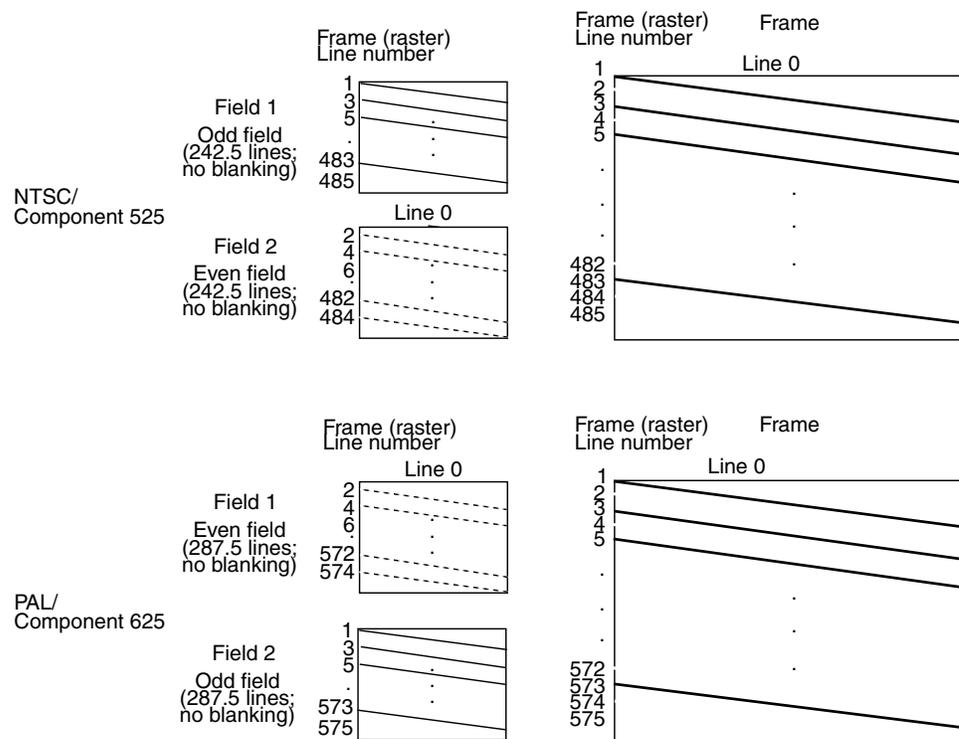


Figure 2-2 Fields and Frames for NTSC and PAL

Users typically want to edit on Field 1 boundaries, where Field 1 is defined as the first field in the video standard's two-field output sequence. 525 standards send the second (whole) raster line out to begin the first field, and the first (half) raster line out to begin the second field; 625 standards send the first (half) raster line out to begin the first field, and the second (whole) raster line to begin the second field.

Some users may want to edit on F2 boundaries, which fall on the field in between the video standard's frame boundary. To do so, use this control, then program your deck to select F2 edits.

NTSC users might need to vary their field dominance choice, depending on the origin of the input material they are to edit.

Note: To output a set of frames, they must be deinterlaced into fields differently, depending on the choice of output field dominance. For example, when F1 dominance is selected, the field with the topmost line must be the first field to be transferred; when F2 dominance is selected, the field with the topmost line must be the second field to be transferred. Deinterlacing must be specified in the application; the following code fragment contains an example of how to consult the field dominance control to determine deinterleave order.

```
/*
 * Set the memory node's timing based upon the video drain's timing,
 * which has been set up by the daemon from the defaults file, or by
 * the user via vcp.
 *
 * When we get around to reading image files, we'll check the file
 * size against the size reported by the VL for this node: if the file
 * size does not match the format's, we'll punt.
 */

int is_525, F1_is_first;
VLControlValue drn timing, dominance;
if (vlGetControl(svr, MEMtoVIDPath, drn, VL_TIMING, &drainTiming) < 0)
{
    vlPerror("GetControl(VL_TIMING) on video drain failed");
    exit(1);
}
if (vlSetControl(svr, MEMtoVIDPath, src, VL_TIMING, &drainTiming) < 0)
{
    vlPerror("SetControl(VL_TIMING) on memory source failed");
    exit(1);
}
```

```
/*
 * Read the video drains's field dominance control setting and timing,
 * then set a variable to indicate which field has the first line, so
 * readimage() will know how to deinterleave frames to fields.
 */
if (vlGetControl(svr, MEMtoVIDPath, drn,
    VL_FIELD_DOMINANCE, &dominance) < 0) {
    vlPerror("GetControl(VL_FIELD_DOMINANCE) on video drain failed");
    exit(1);
}

is_525 = (drainTiming.intVal == VL_TIMING_525_CCIR601));

switch (dominance.intVal) {
    case VL_F1_IS_DOMINANT:
        if (is_525) {
            F1_is_first = 0;
        } else {
            F1_is_first = 1;
        }
        break;
    case VL_F2_IS_DOMINANT:
        if (is_525) {
            F1_is_first = 1;
        } else {
            F1_is_first = 0;
        }
        break;
}
```

VL Support for the General-Purpose Interface (GPI)

The DIVO board has two GPI connectors, each associated with one of the serial digital video ports. Each port has two transmit and two receive channels, lines 0 and 1 in and out.

The VL API supports the GPI as a device-independent interface. It supports GPI triggers in three `vlSetControl()` interfaces. The union `VLControlValue` has been extended to support the controls for

- output triggering, discussed in “Using `VL_GPI_OUT_MODE`” on page 24
- explicitly setting and querying the GPI lines, discussed in “Using `VL_GPI_STATE`” on page 25
- setting up triggering to begin transfers on a path, discussed in “Using `VL_TRANSFER_TRIGGER`” on page 26

Note: See the section “GPI Interface” in Appendix A for extensive hardware information on the GPI interface.

Using `VL_GPI_OUT_MODE`

Use `VL_GPI_OUT_MODE` to program the `gpi_out` line. Two conditions are supported for asserting the GPI line: `transfer_start` and `transfer_stop`. You can have multiple trigger conditions outstanding.

Figure 2-3 diagrams the effect of `VL_GPI_OUT_MODE` values.

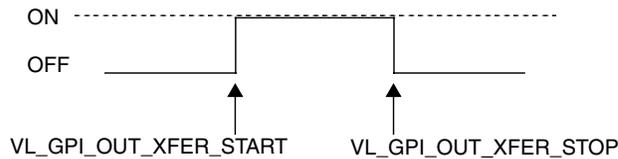


Figure 2-3 Output GPI (OFF Assumed to Be Low)

Note: In this section, OFF is assumed to be low. `VL_CLEAR_GPI` changes the state of the GPI signal to low.

The following pseudocode segment illustrates a setup for gpi_out line 1 to toggle at the beginning and end of transfer:

```

VLControlValue val;

                                /* make sure the GPI line is high */
val.gpi_state.gpi      = VL_GPI_OUT;
val.gpi_state.instance = <which GPI line>;          /* integer */
val.gpi_state.state    = VL_GPI_OFF;
vlSetControl(svr,path,VL_GPI_STATE,&val);

                                /* transfer start */
val.gpi_out.condition  = VL_GPI_OUT_XFER_START;
val.gpi_out.instance   = <which GPI output line >;
val.gpi_out.state      = VL_GPI_ON;

vlSetControl(svr,path,VL_GPI_OUT_MODE,&val);

                                /* transfer stop */
val.gpi_out.condition  = VL_GPI_OUT_XFER_STOP;
val.gpi_out.instance   = <which GPI output line >;
val.gpi_out.state      = VL_GPI_OFF;

vlSetControl(svr,path,VL_GPI_OUT_MODE,&val);

```

To clear all outstanding trigger controls on a particular line, use the gpi_state control with the clear flag.

Using VL_GPI_STATE

Use VL_GPI_STATE to query the state of the input GPI lines and to set or get the state of output GPI lines. The states are ON, OFF, PULSE (transition for one field time), and CLEAR.

The following code fragment clears all output triggers on the specified line:

```

val.gpi_state.gpi      = VL_GPI_OUT;
val.gpi_state.instance = <which GPI line>;
val.gpi_state.state    = VL_GPI_CLEAR;

vlSetControl(svr,path,VL_GPI_STATE,&val);

```

To get the GPI state on an input line, use

```
val.gpi_state.gpi = VL_GPI_IN;
val.gpi_state.instance = <which GPI line>;
vlGetControl (svr, path, VL_GPI_STATE, &val);
```

Using VL_TRANSFER_TRIGGER

The VL_TRANSFER_TRIGGER control specifies the conditions under which transfers begin on a path based on external GPI triggers. This control is valid only on video nodes.

The following code illustrates an input GPI-based trigger transfer setup:

```
VLTransferDescriptor desc;
int num_images = 30; /* 1 second of video */
int trigger = TRUE; /* enable trigger transfers */

if (trigger) {
    val.xfer_trigger.triggerType = VL_TRIGGER_GPI;
    val.xfer_trigger.value.instance = 1; /* GPI line 1 */
    desc.trigger = VLDeviceEvent;
}
else { /* reset for subsequent transfers */
    val.xfer_trigger.triggerType = VL_TRIGGER_NONE;
    desc.trigger = VLTriggerImmediate;
}

vlSetControl(svr, path, src, VL_TRANSFER_TRIGGER, &val);

desc.mode = VL_TRANSFER_MODE_DISCRETE;
desc.count = num_images;
desc.delay = 0;

vlBeginTransfer(svr, path, 1, &desc);
```

Note: The trigger type is a persistent device control setting and remains valid until reset. Thus the VL_TRANSFER_TRIGGER control must be set to VL_TRIGGER_NONE if subsequent transfers are of a nontrigger type.

VL Support for the Vertical Inline Time Clock (VITC)

DIVO always extracts VITC information from the fields as they are captured and passes this information to the application as buffer information associated with the dmBuffer. To extract this information, call **vlDMBufferGetVideoInfo()**. In the structure returned, a valid bit (`validinfo`) indicates if the VITC is valid. Refer to the programming example `divo_vitc.c` in the directory `/usr/share/src/dmedia/video/DIVO`.

To insert VITC, fill the `DMBufferVideoInfo*` structure and call the function **vlDMBufferSetVideoInfo()**.

DIVO Events

The VL provides several ways of handling data stream events, such as completion or failure of data transfer, vertical retrace event, loss of the path to another client, lack of detectable sync, or dropped fields or frames. The method you use depends on the kind of application you are writing:

- For a strictly VL application, use
 - **vlSelectEvents()** to choose the events to which you want the application to respond
 - **vlCallback()** to specify the function called when the event occurs
 - your own event loop or a main loop (**vlMainLoop()**) to dispatch the events
- For an application that also accesses another program or device driver, or if you are adding video capability to an existing X or OpenGL application, set up an event loop in the main part of the application and use the IRIX file descriptor (FD) of the event(s) you want to add.

For more information on these functions, see Chapter 4 in the *Digital Media Programming Guide*.

Table 2-3 summarizes events for DIVO. For DIVO, this table supersedes the table of events in Chapter 14, “VL Event Handling,” in the DMPG; DIVO supports only the events listed in Table 2-3.

Table 2-3 DIVO Events

Event	Use
VLSyncLost	Sync is not detected
VLStreamStarted	Stream started delivery
VLStreamStopped	Stream stopped delivery
VLSequenceLost	A field/frame was dropped
VLControlChanged	A control on the path has changed
VLTransferComplete	A field/frame transfer has completed
VLTransferFailed	A transfer has failed and DMA is aborted
VLEvenVerticalRetrace	Vertical retrace event for an even field
VLOddVerticalRetrace	Vertical retrace event for an odd field
VLDeviceEvent	A dev0 ice-specific event
	A transfer error was discovered; field may be invalid

Setting Inline Controls

Because of the asynchronous nature of the implementation, controls executed with `vlSetControl()` have no guarantees as to when they are executed once transfers are in progress. However, in certain situations, it is useful to be able to specify when controls on a path are executed; for example, to play out video clips with different packings or color-space formats without having to stop transfers while a memory-to-video operation is in progress.

Inline controls specify control changes to happen between buffers. For example, if you want to play out two video clips that have different packing formats in memory, the application would set up the path, queue the buffers from the first clip, set up inline controls to match the next clip, and queue the buffers from the second clip. The syntax for usage is

```
int vlSetControlInLine(VLServer svr, VLPath path, VLNode node,
    VLNode refnode, VLControlType control, VLControlValue *val)
```

In this syntax, *refnode* is the reference node, identifying a unique connection in a path with more than two nodes.

Inline controls are generally applied on a memory node being used as the source node. Control changes are queued to the hardware along with the buffers and are executed in order. To change packing control inline, for example, use

```
VLNode mem;
mem = vlGetNode(svr, VL_SRC, VL_MEM, VL_ANY);
val.intval = VL_PACKING_444_12;
vlSetControlInLine(svr, path, mem, mem, VL_PACKING, &val);
```

Reporting

The `DMediaInfo` structure has been enhanced to report the Unadjusted System Time (UST) and VITC information.

DIVO makes use of the error events noted in Chapter 4 of the DMPEG, as well as `VLTransferErrorEvent`, which reports nonfatal video transfer errors, including error detection and handling (EDH) errors. The `VLTransferComplete` and `VLSequenceLost` events also report the Media Stream Count (MSC) of the field or frame transferred or failed.

DIVO I/O Panel Connector Specifications

This appendix summarizes hardware specifications for the DIVO option:

- “DIVO Connectors” on page 31
- “GPI Interface” on page 33

DIVO Connectors

Table A-1 summarizes return loss for the **IN LINK A**, **IN LINK B**, and **GEN IN** connectors.

Table A-1 Return Loss for DIVO Video and Genlock Channels

Channel	Value
IN LINK A, IN LINK B	> 15 dB @ 270 MHz
GEN IN	> 40 dB @ 6 MH

Table A-2 summarizes output characteristics for the **OUT LINK A** and **OUT LINK B** connectors.

Table A-2 Characteristics for DIVO Digital Video Out Channels

Characteristic	Value
Amplitude	800 mV +/-10%
Rise/fall time	.75 nsec to 1.5 nsec
Overshoot	<10% p-p
Alignment jitter	<740 ps p-p

Table A-3 explains the use of **LINK A** and **LINK B** connectors for 4:2:2:4 mode. If **LINK B** is not used in 4:2:2:4 format, the resulting format is 4:2:2. The **LINK A** connector carries 10-bit wide UVY information; the **LINK B** connector carries 10-bit alpha. Usage is similar for 10-bit RGBA.

Table A-3 Usage for LINK A and LINK B in 4:2:2:4 Mode

Sample	LINK A	LINK B
0	Cb ₀	x
1	Y ₀	A ₀
2	Cr ₀	x
3	Y ₁	A ₁

Table A-4 explains the use of **LINK A** and **LINK B** connectors for 4:4:4:4 mode. The **LINK A** connector carries a 4:2:2 sampled portion of 10-bit wide UVY; the **LINK B** connector carries the remaining 10-bit UV samples and 10-bit alpha. Usage is similar for 10-bit RGBA.

Table A-4 Usage for LINK A and LINK B in 4:4:4:4 Mode

Sample	LINK A	LINK B
0	Cb ₀	Cb ₁
1	Y ₀	A ₀
2	Cr ₀	Cr ₁
3	Y ₁	A ₁

The **GEN OUT** and **GEN IN** connectors make up a passive genlock loopthrough connection. If you attach a cable to one **GEN** connector, you must attach to the other **GEN** connector either a 75-ohm BNC terminator or a cable to other equipment accepting analog sync. If another cable is connected, it must ultimately be terminated.

GPI Interface

For each video pipe, the General Purpose Interface (GPI) provides two channels of input and output trigger signal pairs. This section explains

- “GPI Pinouts” on page 33
- “GPI Transmitter” on page 36
- “GPI Receiver, Switch Closure Mode, and Current Sense Mode” on page 38

GPI Pinouts

The DIVO board has two GPI connectors, each associated with one of the serial digital video ports (two transmit and two receive channels each). Figure A-1 points out the GPI connectors on the DIVO panel.

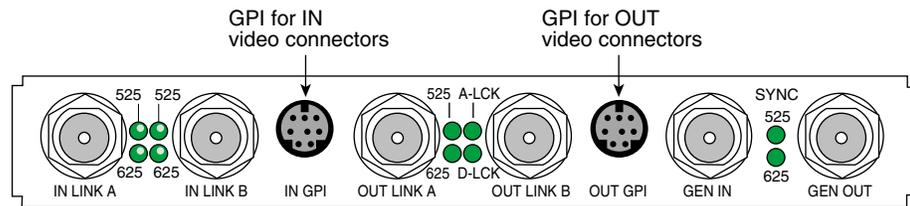


Figure A-1 GPI Connectors

Figure A-2 shows pinouts for the GPI; the information is applicable for both the **IN GPI** and **OUT GPI** connectors. In this figure, CCT denotes contact closure transmit, and CCR denotes contact closure receive.

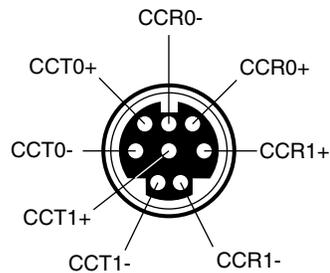


Figure A-2 GPI Pinouts

Each +/- signal pair of the same name applies to one channel of either a receive or transmit optical device. Table A-5 gives the meaning of the pins in Figure A-2.

Table A-5 GPI Pinouts

Pin	Symbol	Name	Channel
8	CCT0+	Contact Closure Transmit +	0
4	CCT0-	Contact Closure Transmit -	0
5	CCT1+	Contact Closure Transmit +	1
2	CCT1-	Contact Closure Transmit -	1
6	CCR0+	Contact Closure Receive +	0
7	CCR0-	Contact Closure Receive -	0
3	CCR1+	Contact Closure Receive +	1
1	CCR1-	Contact Closure Receive -	1

Each GPI header (row of four pins) configures one of four receiver channels: two channels for GPI in and two channels for GPI out. Figure A-3 shows the location of the jumper pins on the board.

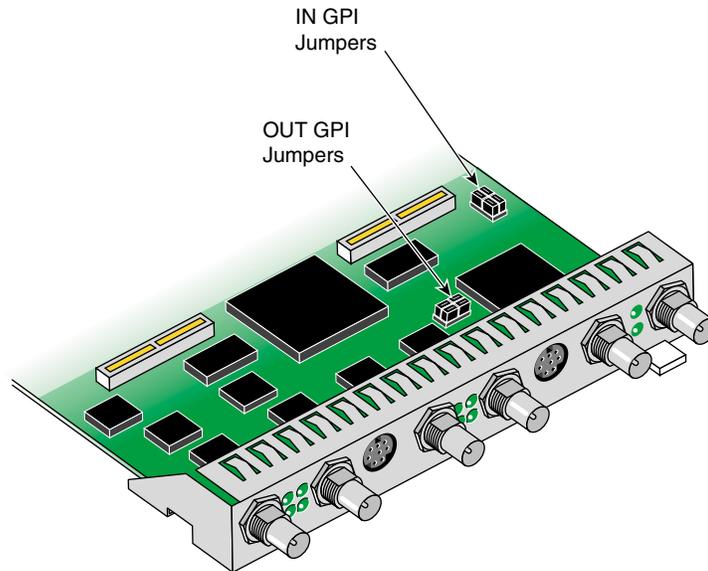


Figure A-3 GPI Jumper Locations (Factory Setting)

Note that the jumpers for the **OUT GPI** connector are near the **OUT GPI** mini-DIN connector, while the jumpers for the **IN GPI** connector are relatively far away from the **IN GPI** connector.

For the factory setting of switch closure mode, two jumpers are factory-installed, shorting pins 1-2 and pins 3-4. These jumpers need not be moved unless you wish to use current sense mode. You can choose to mix the modes for the various channels. This reconfiguration is typically performed by a Silicon Graphics System Service Engineer when the DIVO board is installed in the chassis. Switch closure and current sense modes are explained in “GPI Receiver, Switch Closure Mode, and Current Sense Mode” on page 38.

Figure A-4 shows GPI headers and jumpering. The printed circuit board (PCB) reference designators are included to aid identification of the header associated with each GPI receiver channel.

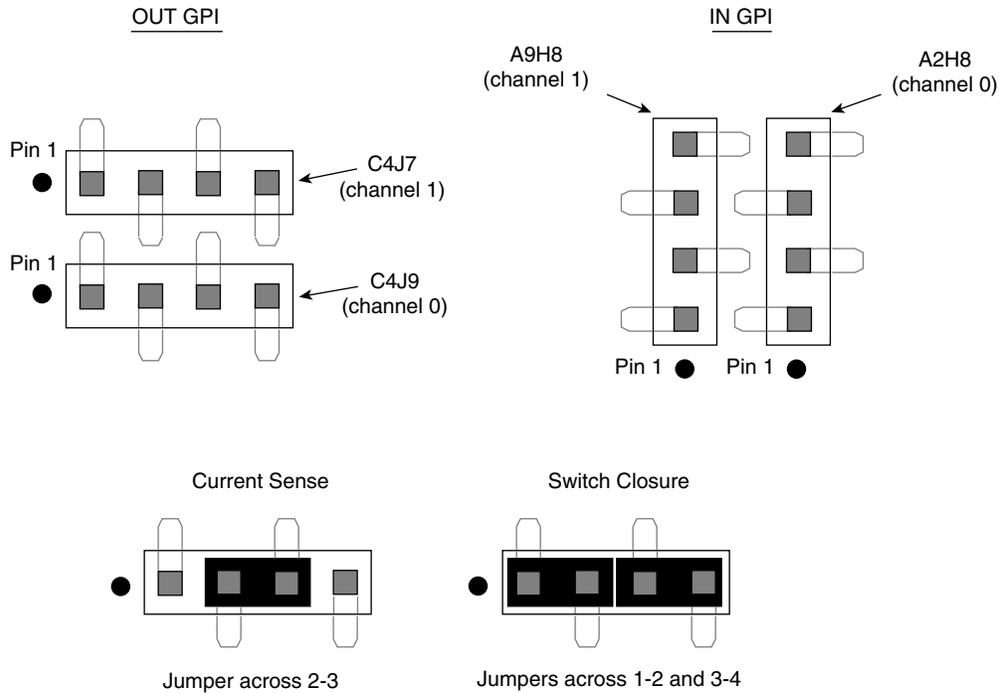


Figure A-4 Example GPI Interface

Note: For information on VL controls for configuring the GPI ports, see “VL Support for the General-Purpose Interface (GPI)” in Chapter 2.

GPI Transmitter

GPI contact closure transmit (CCT) outputs use an optically coupled solid-state relay (SSR) to provide a means of electrical isolation for destination equipment. The GPI transmitter is triggered by a computer command which forward-biases the internal LED, which in turn drives the output MOSFET, closing the contacts of the SSR.

When the GPI trigger is off, a high resistance exists between the CCT+/- terminals. When the GPI is on (triggered by the computer), a low resistance exists between the terminals.

Figure A-5 and Table A-6 show electrical specifications for the GPI transmitter.

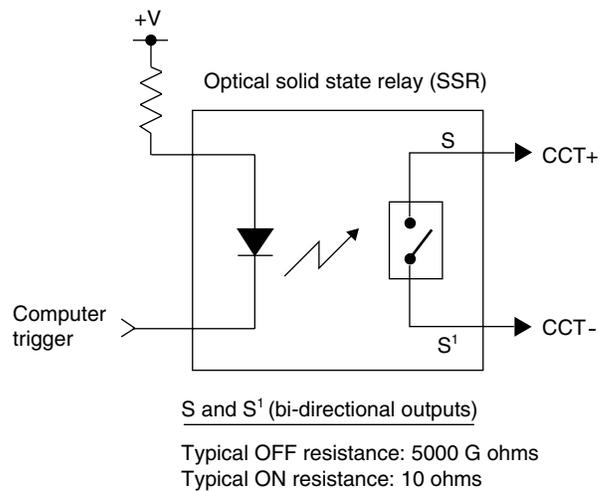


Figure A-5 GPI Transmitter Electrical Specifications

Table A-6 GPI Transmitter Electrical Specifications

Parameter	Value
On resistance	10 ohms typical, 15 ohms maximum
Off resistance	5000 G ohms
Current limit	360 mA typical, 460 mA maximum
Output capacitance	60 pF
Continuous DC load current	180 mA
Output power dissipation	600 mW
Isolation voltage	3750 V rms

The GPI transmitter can be interfaced to the destination equipment by tying the CCT- terminal to GND and using the CCT+ terminal as a current sink. The input device can consist of a logic device with active pullup, an optoisolator LED with series-limiting resistor, or relay primary with series-limiting resistor.

The GPI transmitter’s logic sense can be swapped (inverted) by tying the CCT+ terminal to the logic power supply (VCC) of the destination equipment and using the CCT- terminal to drive the input of the receiving device.

GPI Receiver, Switch Closure Mode, and Current Sense Mode

GPI Contact Closure Receive (CCR) inputs use an optical isolator device to provide a means of electrical isolation from source equipment. The device consists of a bidirectional input LED optically coupled to a bipolar transistor. A voltage pulse applied across the CCR+/- pins causes the LED to become forward-biased and to produce a GPI trigger to the computer.

Table A-7 summarizes electrical specifications for the GPI receiver optoisolator.

Table A-7 GPI Receiver Input Optoisolator

Parameter	Value
Forward voltage (V_F)	1.55 V, 1.2 V typical ($I_F = 10$ mA)
Continuous forward current (I_F)	30 mA
Peak forward current	1000 mA (10 μ s duration, 1% DC)
Reverse current (I_R)	0.1 μ A, 100 μ A maximum ($V_R = 6$ V)
Isolation surge voltage (V_{10})	2500 VAC _{RMS} (t = 1 min)

Figure A-6 shows switch closure jumpering, which creates a digital pulse.

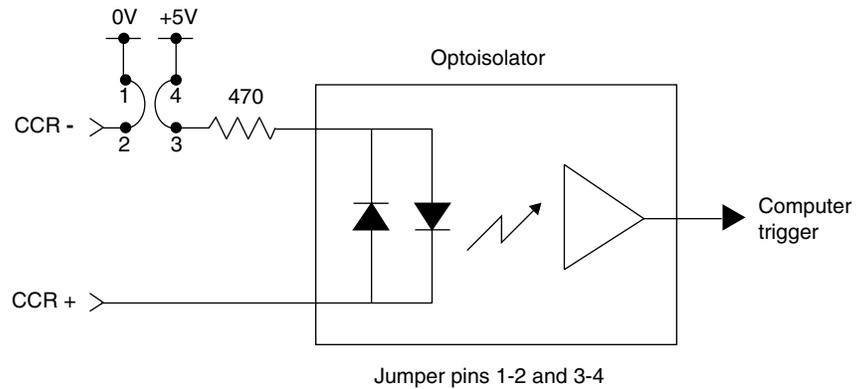


Figure A-6 Jumpering for GPI Switch Closure (Factory Setting)

In switch closure mode, the +5 V power supply and ground of the DIVO board are not electrically isolated from the chassis of the source equipment.

Figure A-7 shows current sense jumpering.

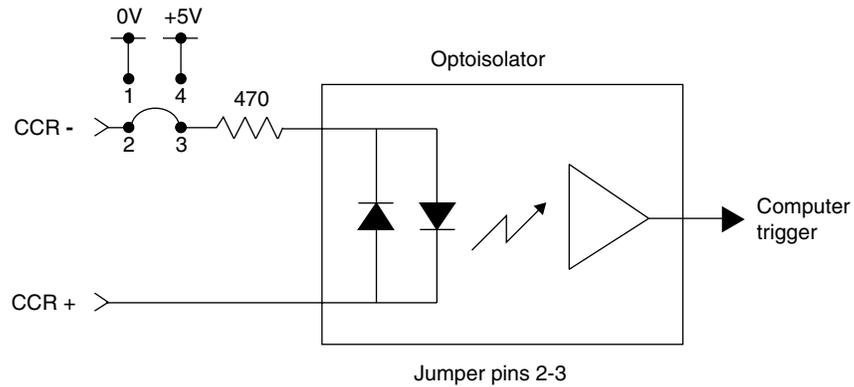


Figure A-7 Jumpering for GPI Current Sense Mode

In current sense mode, the DIVO board is electrically isolated from the chassis of the source equipment.

For switch closure mode, the GPI receiver can be interfaced to the source equipment by tying the CCR+ and CCR- terminals across the output terminals of an optoisolator, solid-state relay, or any device that acts like a single-pole contact switch. A GPI trigger is generated as long as the source switch is closed.

Note: Polarity of the CCR+/- signals must be observed for the source equipment in switch closure mode.

For current sense mode, the CCR+ and CCR- signals can be interfaced by tying the CCR+ terminal to the output of a TTL or CMOS logic device, and by tying the CCR- terminal to GND of the source equipment. Whenever the logic device is sourcing current (driving a logic high), a GPI trigger is generated.

In current sense mode, the logic sense can be swapped (inverted) by moving the CCR- signal from GND to the logic power supply (typically VCC) of the source equipment. The CCR+ signal remains connected to the output of the logic device; however, in this configuration an open collector type device can be used. Whenever the logic device is sinking current (driving a logic low), a GPI trigger is generated.

Setting Up DIVO for Your Video Hardware

This appendix illustrates how to attach video equipment to connectors on the DIVO I/O panel and how to use the video control panel *vcp* to set the DIVO board to match your installation.

This appendix explains

- “Setting Up Digital Source Video” on page 41
- “Setting Up the Output (Drain)” on page 43
- “Setting Up Sync” on page 44
- “Saving Settings” on page 46

Figure B-1 shows connectors on the DIVO I/O panel.

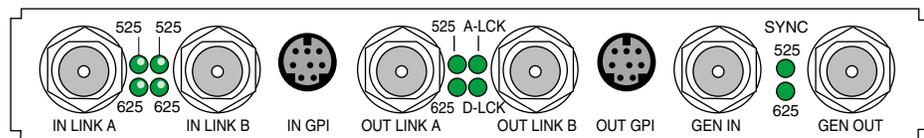


Figure B-1 DIVO Ports

Setting Up Digital Source Video

DIVO has two 10-bit digital video input ports (**IN LINK A** and **IN LINK B**) for equipment that complies with the CCIR 601 standard. The ports can be configured for 4:4:4:4 or 4:2:2:4 dual-link mode; for 4:2:2 single-link mode, ignore the alpha:

- In 4:4:4:4 mode, Link A carries Y plus the U and V from even-numbered sample points; Link B carries alpha plus the U and V from odd-numbered sample points.
- In 4:2:2:4 mode, Link A carries Y plus the U and V from all sample points; Link B carries alpha only.

To set up DIVO for a digital video source, follow these steps:

1. Connect a video device to IN LINK A and, if you are using a device for alpha key data, also **IN LINK B**. If you use only one input, it must be **IN LINK A**.
2. Call up the panel:
`/usr/sbin/vcp`
3. In the Inputs(s): DIVO Digital Video Source section of the control panel *vcp* for the channel(s) you are using, select the format that matches your equipment, as shown in Figure B-2.

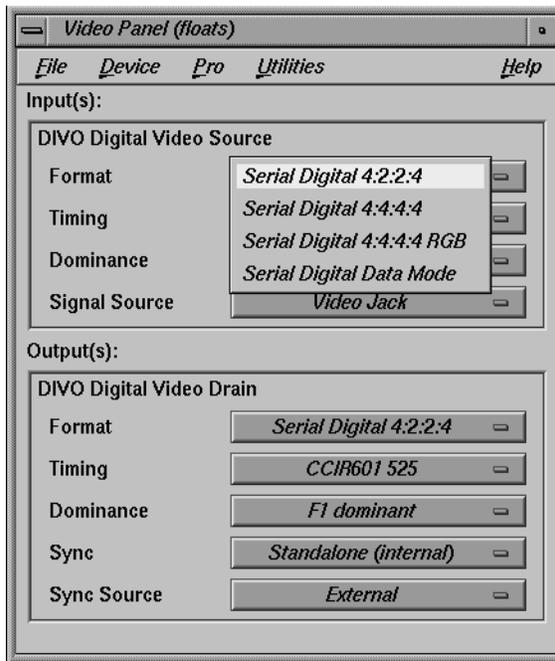


Figure B-2 Selecting Digital Input Video Format in *vcp*

4. In the Digital Video Source portion of the panel for the channel(s) you are using, select the timing that matches your equipment: CCIR 525 or CCIR 625.

Setting Up the Output (Drain)

To set up the digital video output, follow these steps:

1. Connect the video equipment to **OUT LINK A** and, if you are using a device for alpha key data, also **OUT LINK B**. If you use only one output, it must be **OUT LINK A**.
2. If necessary, call up the panel (`/usr/sbin/vcp`).
3. In the Output(s): DIVO Video Drain section of the control panel, select the format that matches your equipment, as shown in Figure B-3.

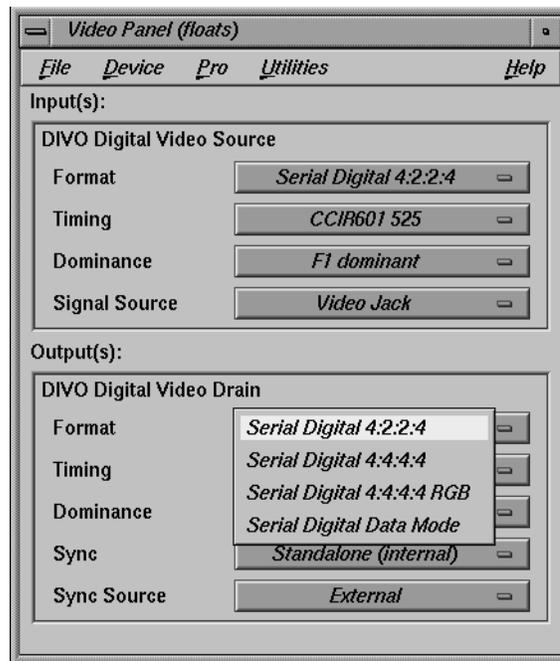


Figure B-3 Selecting Video Drain Format

4. Select the timing that matches your equipment: CCIR 525 or CCIR 625.
5. To set field dominance, at the "Input Timing" menu item select "F1 dominant" for the edit to occur on the nominal video field boundary, or "F2 dominant" for the edit to occur on the intervening field boundary. See "Setting Field Dominance" in Chapter 2 for more information on field dominance.

Setting Up Sync

This section explains

- “Setting Up Internal Sync” on page 44
- “Setting Up External Sync” on page 45

Setting Up Internal Sync

In the Output(s): DIVO Digital Video Drain section of the control panel, select the Sync format that matches your equipment:

- standalone (not synced to another device): select Standalone (internal)
- output sync to an external source connected to the genlock in: select Genlock

These two choices toggle, as shown in Figure B-4.

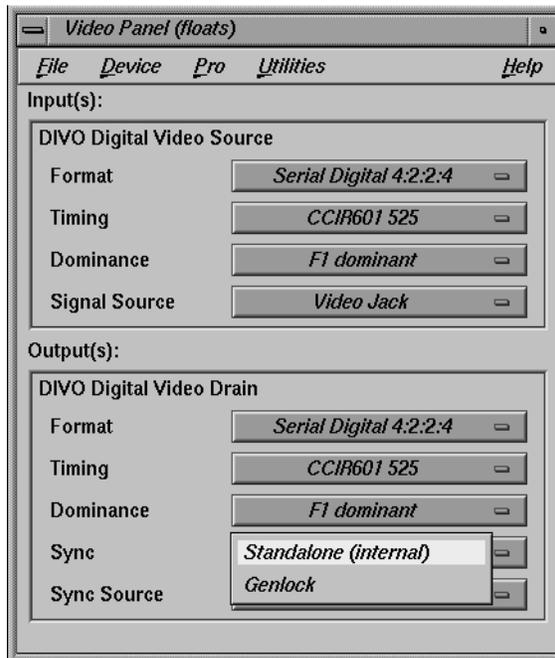


Figure B-4 Setting Standalone or Genlock Sync

Setting Up External Sync

To set up DIVO for an external sync source, follow these steps:

1. Connect the sync source equipment to one of the following connectors:
 - the **GEN IN** BNC on the I/O panel, as diagrammed in Figure B-5.

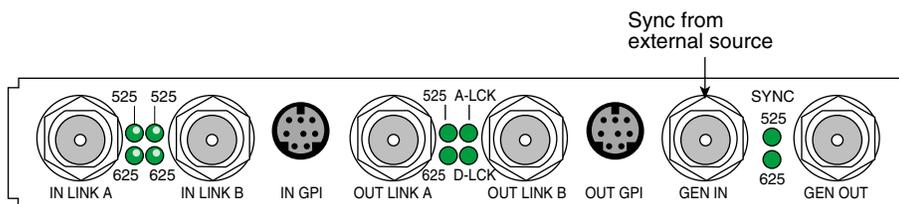


Figure B-5 GEN IN Port on the DIVO I/O Panel

- **IN LINK A** or **IN LINK B** (if the device is not already cabled to this connector)
2. If you are using the same signal for other equipment, attach a BNC cable to the **GEN OUT** BNC to loop the signal through the board. Make sure the final element in the chain is terminated.

If DIVO is the last element in the sync chain, make sure a terminator is attached to the **GEN OUT** BNC.

3. If necessary, call up the panel (`/usr/sbin/vcp`).
4. Select the appropriate setting in Output(s): Sync Source:
 - select External for a device connected to **GEN IN**
 - select Digital Input Link A or B if you are syncing to the device attached to **IN LINK A** or **IN LINK B**, respectively

Saving Settings

Once you have set values in *vcp* to match your installation, save them; they are written to */usr/etc/video/videod.defaults*. Select "Restore Settings" on the video control panel File menu to load the values in this file to *vcp*.

The last settings saved are automatically loaded every time the system is reinitialized. If the panel is running, current settings are in effect.

Note: You do not need to open the panel to put its settings into effect.

You can also use File menu choices to restore the factory defaults and close the panel.

Pixel Packings and Color Spaces

This appendix explains

- “DIVO Pixel Packings” on page 47
- “Sampling Patterns” on page 76
- “Color Spaces” on page 78

DIVO Pixel Packings

This section presents each packing used by the DIVO hardware, giving a diagram and its tokens in the pertinent libraries. It explains

- “Packings and Color Spaces” on page 48
- “Packing Diagram Conventions” on page 48
- “Packings and Library Tokens” on page 50
- “Packing Naming Conventions” on page 51
- “8-Bit Pixel Packings” on page 53
- “16-Bit Pixel Packings” on page 55
- “20-Bit Pixel Packings” on page 57
- “24-Bit Pixel Packings” on page 58
- “32-Bit Pixel Packings” on page 60
- “36-Bit Pixel Packing” on page 69
- “48-Bit Pixel Packings” on page 70
- “64-Bit Pixel Packings” on page 72

Packings and Color Spaces

A packing

- determines which of the four components are sampled, either RGBA or VYUA (more correctly, CrYCbA)
- determines the sampling pattern (for example, 4:4:4:4 or 4:2:2:4), which specifies where and how often each component of the image is sampled
- allocates a certain number of bits to represent the component samples, and positions those samples along with possible padding in memory; each sample is an unsigned number, unless specified otherwise in the description of the packing

A color space

- determines the color in each component by specifying the color set (see Table C-2)
- specifies a canonical minimum and maximum value for each component, either full range or compressed range (headroom range); see “Color Spaces” on page 78 for an explanation

In most Silicon Graphics libraries, a single token encodes both color space and packing. For example, `VL_PACKING_RGBA_8` is a 32-bit packing in the RGBA color space. For the VL of DIVO and other advanced products, the two parameters are specified separately with different controls: `VL_PACKING` and `VL_COLORSPACE`. The color space must be defined with the `VL_COLORSPACE` control.

Packing Diagram Conventions

In all illustrations, as you move from left to right:

- each byte goes from the most significant bit to the least significant bit
- the bytes increase in memory address by 1
- component samples go from most significant bit to least significant bit

Each illustration shows the smallest repeating spatial pattern of component samples that is a multiple of 8 bits wide. No additional padding or alignment is to be inferred. For example, a 24-bit-per-pixel diagram, such as that for VL_PACKING_444_8 (Figure C-1), indicates 3-byte quantities packed together in memory; the values are not padded out to 32-bit boundaries.

Pixel 1		
Byte 1	Byte 2	Byte 3
r r r r r r r r	g g g g g g g g	b b b b b b b b
v v v v v v v v	y y y y y y y y	u u u u u u u u

Figure C-1 VL_PACKING_444_8

The packing defines a bit layout, but for convenience, as shown in Figure C-1, the component slots are filled with the RGBA or VYUA color set as appropriate. (See “Color Spaces” later in this appendix for more information.) For chroma components, Cr and Cb are more accurate terms than V and U, because the analog NTSC video specification ANSI/SMPTE 170M uses V and U with a slightly different meaning. However, this chapter uses the letters V and U in the illustrations of packings for typographical convenience.

Packings that use 4:2:2 sampling also show the location of each component sample: left and right for 4:2:2. The diagrams assume row-major, left-to-right ordering of pixels in memory.

An x (“don’t care”) in a bit means the following:

- Readers may get any garbage in this bit.
- Writers can leave this bit as garbage.

A 0 means the following:

- Readers may assume this bit is zero.
- Writers can leave this bit as garbage.

An *s* indicates a padding bit that is a sign extension bit. For the DIVO option, this convention applies only to the more significant bits in 12-bit and 13-bit packings with rightward orientation; that is, `VL_PACKING_4444_12_in_16_R` and `VL_PACKING_4444_13_in_16_R`.

A *p* indicates a padding bit in the least significant bits of a left-justified 10-, 12-, or 13-bit word, such as `VL_PACKING_R242_10_in_16_L` or `VL_PACKING_4444_13_in_16_L`:

- Readers can assume that the bits are replicated from the component found in the same word: With bits numbered starting with 0 for the least significant, there are *n* contiguous *p* bits to the right of the component. The *p* bits contain a copy of bits [9,9-*n*+1] of the component.
- Writers can leave the *p* bits as garbage.

The DIVO device can natively transfer data of all the packings shown in this appendix in real time.

Packings and Library Tokens

Following each packing diagram are comments and library tokens for that packing, listing, where applicable, the color set (RGBA or VYUA) and the library (VL, OpenGL, and DM) for each library token.

- DM refers to the tokens in `/usr/lib/dmedia/dm_image.h`, which are used by several libraries (`libdmedia` (`dmParams`, `dmIC`, `dmColor`), `libmoviefile`, `libmovieplay`, and others). See “Color Spaces” in this appendix for more information.
- For most packings, two indications are given for VL:
 - VL, new style, includes the packing control value and a color-space control value; for example, `VL_PACKING_4_8 + VL_COLORSPACE_{CCIR,YUV}`. For DIVO, you set packing and color space separately for memory nodes. In contrast to Sirius Video, `VL_COLORSPACE` replaces `VL_FORMAT` on DIVO memory nodes. The new definitions provide a more flexible way to specify memory layout of pixels and their color spaces.
 - VL, old style (for example, `VL_PACKING_Y_8_P`) is included for reference; these tokens are still recognized in case you are using programs for earlier Silicon Graphics video options that include these. It is not recommended for new development.

Packing Naming Conventions

In packing tokens, the following applies:

- `_L` and `_R` appended to the end of tokens with padding (0 bits) indicate that the 0 bits are at the left end or the right end of the pattern, respectively; for example, `VL_PACKING_4444_10_in_16_L` and `VL_PACKING_4444_10_in_16_R`.
- `X` before the numerical part of the token at the end of a token indicates a component order other than the standard (RGBA or ABGR, VYUA or AUYYV); for example, `VL_PACKING_X4444_5551`, which uses ARGB order.
- `R` before the numerical part of the token indicates reverse order of the components; for example, `VL_PACKING_242_8` and `VL_PACKING_R242_8` have the same pattern of component bits, but their order is reversed in `VL_PACKING_R242_8`.
- `Z` at the end of the token name means that the packing is padded to the word boundary; for example, the packing in `VL_PACKING_2424_10_10_10_2Z` is 30 bits per pixel, but it is padded to 32 bits per pixel.

Table C-1 lists the DIVO packings in the order of the number of bits in the pattern of component samples—the order in which they are described in the rest of this section.

Table C-1 DIVO Packings

Packing	Bits	Color Space
<code>VL_PACKING_4_8</code>	8	VYUA monochrome/luma only
<code>VL_PACKING_R444_332</code>	8	RGBA
<code>VL_PACKING_444_332</code>	8	RGBA
<code>VL_PACKING_242_8</code>	16	VYUA
<code>VL_PACKING_R242_8</code>	16	VYUA
<code>VL_PACKING_X4444_5551</code>	16	RGBA
<code>VL_PACKING_444_5_6_5</code>	16	RGBA
<code>VL_PACKING_242_10</code>	20	VYUA
<code>VL_PACKING_R242_10</code>	20	VYUA
<code>VL_PACKING_444_8</code>	24	RGBA/VYUA

Table C-1 (continued) DIVO Packings

Packing	Bits	Color Space
VL_PACKING_R444_8	24	RGBA/VYUA
VL_PACKING_4444_6	24	RGBA/VYUA
VL_PACKING_4444_8	32	RGBA/VYUA
VL_PACKING_R4444_8	32	RGBA/VYUA
VL_PACKING_R0444_8	32	RGBA/VYUA
VL_PACKING_0444_8	32	RGBA/VYUA
VL_PACKING_4444_10_10_10_2	32	RGBA/VYUA
VL_PACKING_2424_10_10_10_2Z	32	VYUA
VL_PACKING_R2424_10_10_10_2Z	32	VYUA
VL_PACKING_242_10_in_16_L	32	VYUA
VL_PACKING_242_10_in_16_R	32	VYUA
VL_PACKING_R242_10_in_16_L	32	VYUA
VL_PACKING_R242_10_in_16_R	32	VYUA
VL_PACKING_444_12	36	RGBA/VYUA (signed)
VL_PACKING_4444_12	48	RGBA/VYUA (signed)
VL_PACKING_444_10_in_16_L	48	RGBA/VYUA
VL_PACKING_4444_10_in_16_L	64	RGBA/VYUA
VL_PACKING_4444_10_in_16_R	64	RGBA/VYUA
VL_PACKING_4444_12_in_16_L	64	RGBA (signed)
VL_PACKING_4444_12_in_16_R	64	RGBA (signed)
VL_PACKING_4444_13_in_16_L	64	RGBA (signed)
VL_PACKING_4444_13_in_16_R	64	RGBA (signed)

The packings are explained in these categories:

- “8-Bit Pixel Packings” on page 53
- “16-Bit Pixel Packings” on page 55
- “20-Bit Pixel Packings” on page 57
- “24-Bit Pixel Packings” on page 58
- “32-Bit Pixel Packings” on page 60
- “36-Bit Pixel Packing” on page 69
- “48-Bit Pixel Packings” on page 70
- “64-Bit Pixel Packings” on page 72

8-Bit Pixel Packings

Figure C-2 shows the VL_PACKING_4_8, an 8-bit packing useful for VYUA monochrome/luma only.

Pixel 1
Byte 1
y y y y y y y y

Figure C-2 VL_PACKING_4_8

This packing is

- VL_PACKING_4_8 + VL_COLORSPACE_{CCIR,YUV} in the VL, new style
- VL_PACKING_Y_8_P in the VL, old style
- GL_LUMINANCE GL_UNSIGNED_BYTE in OpenGL
- DM_IMAGE_PACKING_LUMINANCE in DM

Figure C-3 shows VL_PACKING_R444_332, an 8-bit packing in the RGBA color space.

Pixel 1
Byte 1
b b g g g r r r

Figure C-3 VL_PACKING_R444_332

This packing is

- VL_PACKING_R444_332 + VL_COLORSPACE_{RGB,RP175} in the VL, new style
- VL_PACKING_RGB_332_P in the VL, old style
- DM_IMAGE_PACKING_BGR233 in DM

Figure C-4 shows VL_PACKING_444_332, an 8-bit RGBA packing.

Pixel 1
Byte 1
r r r g g g b b

Figure C-4 VL_PACKING_444_332

This packing is

- VL_PACKING_444_332 + VL_COLORSPACE_{RGB,RP175} in the VL, new style
- GL_RGB GL_UNSIGNED_BYTE_3_3_2_EXT in OpenGL
- DM_IMAGE_PACKING_RGB332 in DM

16-Bit Pixel Packings

Figure C-5 shows VL_PACKING_242_8, a 16-bit VYUA packing.

Pixels 1-2			
Byte 1	Byte 2	Byte 3	Byte 4
v v v v v v v v	y y y y y y y y	u u u u u u u u	y y y y y y y y
left			right

Figure C-5 VL_PACKING_242_8

This rarely used packing is VL_PACKING_242_8 + VL_COLORSPACE_{CCIR,YUV} in the VL. It samples chroma and luma in a 4:2:2 pattern. See “Sampling Patterns,” later in this appendix.

Figure C-6 shows VL_PACKING_R242_8, a 16-bit 4:2:2 VYUA packing. The most commonly used 4:2:2 packing, it is used by other Silicon Graphics video hardware as well as DIVO hardware.

Pixels 1-2			
Byte 1	Byte 2	Byte 3	Byte 4
u u u u u u u u	y y y y y y y y	v v v v v v v v	y y y y y y y y
left			right

Figure C-6 VL_PACKING_R242_8

This packing is

- VL_PACKING_R242_8 + VL_COLORSPACE_{CCIR,YUV} in the VL, new style
- VL_PACKING_YVYU_422_8 in the VL, old style
- GL_YCRCB_422_SGIX GL_UNSIGNED_BYTE in OpenGL
- DM_IMAGE_PACKING_CbYCrY in DM

Figure C-7 shows VL_PACKING_X4444_5551, a 16-bit RGBA packing that corresponds to the QuickTime file 16-bit uncompressed format with alpha.

Pixel 1	
Byte 1	Byte 2
a r r r r r g g	g g g b b b b b

Figure C-7 VL_PACKING_X4444_5551

This packing is

- VL_PACKING_X4444_5551 + VL_COLORSPACE_{RGB,RP175} in the VL, new style
- VL_PACKING_ARGB_1555 in the VL, old style
- DM_IMAGE_PACKING_XRGB1555 in DM (even though the upper bit is really alpha)

Figure C-8 shows VL_PACKING_444_5_6_5, a 16-bit RGBA packing.

Pixel 1	
Byte 1	Byte 2
r r r r r g g g	g g g b b b b b

Figure C-8 VL_PACKING_444_5_6_5

This packing is VL_PACKING_444_5_6_5 + VL_COLORSPACE_{RGB,RP175} in the VL, new style.

20-Bit Pixel Packings

Figure C-9 shows VL_PACKING_242_10, a 20-bit VYUA packing.

Pixels 1-2				
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
v v v v v v v v	v v y y y y y y	y y y y u u u u	u u u u u u y y	y y y y y y y y
left			right	

Figure C-9 VL_PACKING_242_10

This packing is VL_PACKING_242_10 + VL_COLORSPACE {CCIR,YUV}.

Figure C-10 shows VL_PACKING_R242_10, a 20-bit VYUA packing.

Pixels 1-2				
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
u u u u u u u u	u u y y y y y y	y y y y v v v v	v v v v v v y y	y y y y y y y y
left			right	

Figure C-10 VL_PACKING_R242_10

This packing is VL_PACKING_R242_10 + VL_COLORSPACE {CCIR,YUV}.

24-Bit Pixel Packings

Figure C-11 shows VL_PACKING_444_8, a 24-bit RGBA/VYUA packing.

Pixel 1		
Byte 1	Byte 2	Byte 3
r r r r r r r r	g g g g g g g g	b b b b b b b b
v v v v v v v v	y y y y y y y y	u u u u u u u u

Figure C-11 VL_PACKING_444_8

This packing is

- RGBA:
 - GL_RGB GL_UNSIGNED_BYTE in OpenGL
 - VL_PACKING_444_8 + VL_COLORSPACE_{RGB,RP175} in the VL, new style
 - VL_PACKING_BGR_8_P in the VL, old style
 - GL_RGB GL_UNSIGNED_BYTE in OpenGL
 - DM_IMAGE_PACKING_RGB in DM
- VYUA:
 - VL_PACKING_444_8 + VL_COLORSPACE_{RGB,RP175} in the VL, new style
 - VL_PACKING_UYV_8_P in the VL, old style

Figure C-12 shows VL_PACKING_R444_8, a 24-bit RGBA/VYUA packing.

Pixel 1		
Byte 1	Byte 2	Byte 3
b b b b b b b b	g g g g g g g g	r r r r r r r r
u u u u u u u u	y y y y y y y y	v v v v v v v v

Figure C-12 VL_PACKING_R444_8

This packing is

- RGBA:
 - VL_PACKING_R444_8 + VL_COLORSPACE_{RGB,RP175} in the VL, new style
 - VL_PACKING_RGB_8_P in the VL, old style
 - DM_IMAGE_PACKING_BGR in DM
- VYUA:
 - VL_PACKING_R444_8 + VL_COLORSPACE_{CCIR,YUV} in the VL, new style
 - DM_IMAGE_PACKING_CbYCr in DM

Figure C-13 shows VL_PACKING_4444_6, a 24-bit DIVO-only packing with 6 bits per pixel.

Pixel 1		
Byte 1	Byte 2	Byte 3
r r r r r r g g	g g g g b b b b	b b a a a a a a
v v v v v y y y	y y y u u u u u	u u a a a a a a

Figure C-13 VL_PACKING_4444_6

This packing is

- RGBA: VL_PACKING_4444_6 + VL_COLORSPACE_{RGB,RP175} in the VL, new style
- VYUA: VL_PACKING_4444_6 + VL_COLORSPACE_{CCIR,YUV} in the VL, new style

32-Bit Pixel Packings

This section explains

- “OpenGL-Like 32-Bit Pixel Packing” on page 61
- “IRIS GL-Like 32-Bit Pixel Packings” on page 62
- “32-Bit Pixel Packing for QuickTime” on page 64
- “4:4:4:4 10_10_10_2 32-Bit Pixel Packing” on page 65
- “4:2:2:4 10_10_10_2 32-Bit Pixel Packings” on page 66
- “4:2:2 10_in_16 32-Bit Pixel Packings” on page 67

OpenGL-Like 32-Bit Pixel Packing

Figure C-14 shows VL_PACKING_4444_8, an OpenGL-like 32-bit packing. This packing, supported by many Silicon Graphics video products, is the most commonly used OpenGL packing.

Pixel 1			
Byte 1	Byte 2	Byte 3	Byte 4
r r r r r r r r	g g g g g g g g	b b b b b b b b	a a a a a a a a
v v v v v v v v	y y y y y y y y	u u u u u u u u	a a a a a a a a

Figure C-14 VL_PACKING_4444_8

This packing is

- RGBA:
 - VL_PACKING_4444_8 + VL_COLORSPACE_{RGB,RP175} in the VL, new style
 - VL_PACKING_ABGR_8 in the VL, old style
 - GL_RGBA GL_UNSIGNED_BYTE in OpenGL (the default)
 - DM_IMAGE_PACKING_RGBA in DM
- VYUA:
 - VL_PACKING_4444_8 + VL_COLORSPACE_{CCIR,YUV} in the VL, new style
 - VL_PACKING_AUYV_4444_8 or VL_PACKING_AUYV_8 in the VL, old style

IRIS GL-Like 32-Bit Pixel Packings

Figure C-15 shows VL_PACKING_R4444_8, an IRIS GL-like 32-bit packing. This packing, supported by many Silicon Graphics video products, is the default IRIS GL packing.

Pixel 1			
Byte 1	Byte 2	Byte 3	Byte 4
a a a a a a a a	b b b b b b b b	g g g g g g g g	r r r r r r r r
a a a a a a a a	u u u u u u u u	y y y y y y y y	v v v v v v v v

Figure C-15 VL_PACKING_R4444_8

This packing is

- **RGBA:**
 - VL_PACKING_R4444_8 + VL_COLORSPACE_{RGB,RP175} in the VL, new style
 - VL_PACKING_RGBA_8 in the VL, old style
 - GL_ABGR_EXT GL_UNSIGNED_BYTE in OpenGL
 - DM_IMAGE_PACKING_ABGR in DM
- **VYUA:**
 - VL_PACKING_R4444_8 + VL_COLORSPACE_{CCIR,YUV} in the VL, new style
 - VL_PACKING_YUVA_4444_8 in the VL, old style

Figure C-16 shows VL_PACKING_R0444_8, an IRIS GL-like 32-bit packing. This packing is supported by many Silicon Graphics video products.

Pixel 1			
Byte 1	Byte 2	Byte 3	Byte 4
x x x x x x x x	b b b b b b b b	g g g g g g g g	r r r r r r r r
x x x x x x x x	u u u u u u u u	y y y y y y y y	v v v v v v v v

Figure C-16 VL_PACKING_R0444_8

- **RGBA:**
 - VL_PACKING_R0444_8 + VL_COLORSPACE_{RGB,RP175} in the VL, new style
 - VL_PACKING_RGB_8 in the VL, old style
 - DM_IMAGE_PACKING_XBGR

Use DM_IMAGE_PACKING_ABGR instead of this packing unless you specifically want to inform a piece of software (such as dmColor) not to spend processing time on the alpha channel.
- **VYUA:**
 - VL_PACKING_R0444_8 + VL_COLORSPACE_{CCIR,YUV} in the VL, new style
 - VL_PACKING_YUV_444_8 in the VL, old style

32-Bit Pixel Packing for QuickTime

Figure C-17 shows VL_PACKING_0444_8, a 32-bit packing used for QuickTime files (uncompressed format without alpha).

Pixel 1			
Byte 1	Byte 2	Byte 3	Byte 4
x x x x x x x x	r r r r r r r r	g g g g g g g g	b b b b b b b b
x x x x x x x x	v v v v v v v v	y y y y y y y y	u u u u u u u u

Figure C-17 VL_PACKING_0444_8

This packing is

- **RGBA:**
 - VL_PACKING_0444_8 + VL_COLORSPACE_{RGB,RP175} in the VL, new style
 - DM_IMAGE_PACKING_XRGB in DM
- **VYUA:** VL_PACKING_0444_8 + VL_COLORSPACE_{CCIR,YUV}

4:4:4:4 10_10_10_2 32-Bit Pixel Packing

Figure C-18 shows VL_PACKING_4444_10_10_10_2, the 32-bit 4:4:4:4 10_10_10_2 packing.

Pixel 1			
Byte 1	Byte 2	Byte 3	Byte 4
r r r r r r r r	r r g g g g g g	g g g g b b b b	b b b b b b a a
v v v v v v v v	v v y y y y y y	y y y y u u u u	u u u u u u a a

Figure C-18 VL_PACKING_4444_10_10_10_2

This packing is

- **RGBA:**
 - VL_PACKING_4444_10_10_10_2 + VL_COLORSPACE_{RGB,RP175} in the VL, new style
 - VL_PACKING_A_2_BGR_10 in the VL, old style
 - GL_RGBA GL_UNSIGNED_INT_10_10_10_2_EXT in OpenGL
- **VYUA:**
 - VL_PACKING_4444_10_10_10_2 + VL_COLORSPACE_{CCIR,YUV} in the VL, new style
 - VL_PACKING_A_2_UYV_10 in the VL, old style

4:2:2:4 10_10_10_2 32-Bit Pixel Packings

Figure C-19 shows VL_PACKING_2424_10_10_10_2Z, the 4:2:2:4 10_10_10_2 32-bit VYUA packing. Only DIVO uses this packing.

Pixels 1-2							
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
v v v v v v v v	v v y y y y y y	y y y y a a a a	a a a a a a 0 0	u u u u u u u u	u u y y y y y y	y y y y a a a a	a a a a a a 0 0
left				0 0	left	right	0 0

Figure C-19 VL_PACKING_2424_10_10_10_2Z

This packing is

- 4:2:2:4 sampling; see “Sampling Patterns” on page 76 in this appendix
- VL_PACKING_2424_10_10_10_2Z + VL_COLORSPACE_{CCIR,YUV} in the VL, new style

Figure C-20 shows VL_PACKING_R2424_10_10_10_2Z, an alternate 4:2:2:4 10_10_10_2 32-bit packing.

Pixels 1-2							
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
u u u u u u u u	u u y y y y y y	y y y y a a a a	a a a a a a 0 0	v v v v v v v v	v v y y y y y y	y y y y a a a a	a a a a a a 0 0
left				0 0	left	right	0 0

Figure C-20 VL_PACKING_R2424_10_10_10_2Z

This packing is

- 4:2:2:4 sampling; see “Sampling Patterns” on page 76 in this appendix
- VL_PACKING_R2424_10_10_10_2Z + VL_COLORSPACE_{CCIR,YUV} in the VL, new style
- VL_PACKING_AYU_AYV_10 in the VL, old style

4:2:2 10_in_16 32-Bit Pixel Packings

The diagrams of packings that use 4:2:2 sampling show the location (left and right) of each component sample. Only DIVO uses this packing.

Figure C-21 shows VL_PACKING_242_10_in_16_L, a DIVO-only 4:2:2 10_in_16 32-bit VYUA packing. For an explanation of the p bit, see “Packing Diagram Conventions” on page 48.

Pixels 1-2							
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
v v v v v v v v	v v p p p p p p	y y y y y y y y	y y p p p p p p	u u u u u u u u	u u p p p p p p	y y y y y y y y	y y p p p p p p
left		left		left		right	

Figure C-21 VL_PACKING_242_10_in_16_L

This packing is

- 4:2:2 sampling; see “Sampling Patterns” on page 76 in this appendix
- VL_PACKING_242_10_in_16_L + VL_COLORSPACE_{CCIR,YUV} in the VL, new style

Figure C-22 shows VL_PACKING_242_10_in_16_R, a DIVO-only 4:2:2 10_in_16 32-bit VYUA packing.

Pixels 1-2							
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
0 0 0 0 0 0	v v v v v v v v	0 0 0 0 0 0	y y y y y y y y	0 0 0 0 0 0	u u u u u u u u	0 0 0 0 0 0	y y y y y y y y
0 0 0 0 0 0 left		0 0 0 0 0 0 left		0 0 0 0 0 0 left		0 0 0 0 0 0 right	

Figure C-22 VL_PACKING_242_10_in_16_R

This packing is

- 4:2:2 sampling; see “Sampling Patterns” on page 76 in this appendix
- VL_PACKING_242_10_in_16_R + VL_COLORSPACE_{CCIR,YUV} in the VL, new style

Figure C-23 shows VL_PACKING_R242_10_in_16_L, a 4:2:2 10_in_16 32-bit VYUA packing. This packing is supported by several Silicon Graphics video products. For an explanation of the p bit, see “Packing Diagram Conventions” on page 48.

Pixels 1-2							
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
u u u u u u u u	u u p p p p p p	y y y y y y y y	y y p p p p p p	v v v v v v v v	v v p p p p p p	y y y y y y y y	y y p p p p p p
left	p p p p p p	left	p p p p p p	left	p p p p p p	right	p p p p p p

Figure C-23 VL_PACKING_R242_10_in_16_L

This packing is

- 4:2:2 sampling; see “Sampling Patterns” on page 76 in this appendix
- VL_PACKING_R242_10_in_16_L + VL_COLORSPACE_{CCIR,YUV} in the VL, new style
- VL_PACKING_YVYU_422_10 in the VL, old style

Figure C-24 shows VL_PACKING_R242_10_in_16_R, a DIVO-only 4:2:2 10_in_16 32-bit VYUA packing.

Pixels 1-2							
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
0 0 0 0 0 0 u u	u u u u u u u u	0 0 0 0 0 0 y y	y y y y y y y y	0 0 0 0 0 0 v v	v v v v v v v v	0 0 0 0 0 0 y y	y y y y y y y y
0 0 0 0 0 0	left	0 0 0 0 0 0	left	0 0 0 0 0 0	left	0 0 0 0 0 0	right

Figure C-24 VL_PACKING_R242_10_in_16_R

This packing is

- VYUA VL_PACKING_R242_10_in_16_R + VL_COLORSPACE_{CCIR,YUV}
- 4:2:2 sampling; see “Sampling Patterns” on page 76 in this appendix

36-Bit Pixel Packing

Figure C-25 shows VL_PACKING_444_12, the 36-bit packing, which has 12 bits per component. Only DIVO uses this packing.

Pixel 1					Pixel 2			
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
r r r r r r r r	r r r r g g g g	g g g g g g g g	b b b b b b b b	b b b b r r r r	r r r r r r r r	g g g g g g g g	g g g g b b b b	b b b b b b b b
v v v v v v v v	v v v v y y y y	y y y y y y y y	u u u u u u u u	u u u u v v v v	v v v v v v v v	y y y y y y y y	y y y y u u u u	u u u u u u u u

Figure C-25 VL_PACKING_444_12

This packing is

- RGBA: VL_PACKING_444_12 + VL_COLORSPACE_{RGB,RP175} in the VL, new style
- VYUA: VL_PACKING_444_12 + VL_COLORSPACE_{CCIR,YUV} in the VL, new style

The components in this packing are signed:

- For RGB and RP-175 color spaces, the components are often negative.
- For the YUV color space, they are negative when the incoming video signal has values outside of the nominal range.
- For the CCIR601 color space, they are always positive.

48-Bit Pixel Packings

Figure C-26 shows VL_PACKING_4444_12, a 48-bit packing, with 12 bits per component. Only DIVO uses this packing.

Pixel 1					
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
r r r r r r r r	r r r r g g g g	g g g g g g g g	b b b b b b b b	b b b b a a a a	a a a a a a a a
v v v v v v v v	v v v v y y y y	y y y y y y y y	u u u u u u u u	u u u u a a a a	a a a a a a a a

Figure C-26 VL_PACKING_4444_12

This packing is

- RGBA: VL_PACKING_4444_12 + VL_COLORSPACE_{RGB,RP175} in the VL, new style
- VYUA: VL_PACKING_4444_12 + VL_COLORSPACE_{CCIR,YUV} in the VL, new style

The components in this packing are signed:

- For RGB and RP-175 color spaces, the components are often negative.
- For the YUV color space, they are negative when the incoming video signal has values outside of the nominal range.
- For the CCIR601 color space, they are always positive.

Figure C-27 shows VL_PACKING_444_10_in_16_L, a 48-bit packing, with 10 bits per component and no alpha. For an explanation of the p bit, see “Packing Diagram Conventions” on page 48.

Pixel 1					
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
r r r r r r r r	r r p p p p p p	g g g g g g g g	g g p p p p p p	b b b b b b b b	b b p p p p p p
v v v v v v v v	v v p p p p p p	y y y y y y y y	y y p p p p p p	u u u u u u u u	u u p p p p p p

Figure C-27 VL_PACKING_444_10_in_16_L

This packing is

- RGBA: VL_PACKING_444_10_in_16_L + VL_COLORSPACE_{RGB,RP175} in the VL, new style
- VYUA: VL_PACKING_444_10_in_16_L, + VL_COLORSPACE_{CCIR,YUV} in the VL, new style

64-Bit Pixel Packings

Figure C-28 shows VL_PACKING_4444_10_in_16_L. For an explanation of the p bit, see “Packing Diagram Conventions” on page 48.

Pixel 1							
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
r r r r r r r r	r r p p p p p p	g g g g g g g g	g g p p p p p p	b b b b b b b b	b b p p p p p p	a a a a a a a a	a a p p p p p p
v v v v v v v v	v v p p p p p p	y y y y y y y y	y y p p p p p p	u u u u u u u u	u u p p p p p p	a a a a a a a a	a a p p p p p p

Figure C-28 VL_PACKING_4444_10_in_16_L

This packing is

- **RGBA:**
 - VL_PACKING_4444_10_in_16_L + VL_COLORSPACE_{RGB,RP175} in the VL, new style
 - VL_PACKING_ABGR_10 in the VL, old style
- **VYUA:**
 - VL_PACKING_4444_10_in_16_L + VL_COLORSPACE_{CCIR,YUV} in the VL, new style
 - VL_PACKING_AUYV_4444_10 in the VL, old style

Figure C-29 shows VL_PACKING_4444_10_in_16_R.

Pixel 1							
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
0 0 0 0 0 0 r r r r r r r r r r	0 0 0 0 0 0 g g g g g g g g g g	0 0 0 0 0 0 b b b b b b b b b b	0 0 0 0 0 0 a a a a a a a a a a				
0 0 0 0 0 0 v v v v v v v v v v	0 0 0 0 0 0 y y y y y y y y y y	0 0 0 0 0 0 u u u u u u u u u u	0 0 0 0 0 0 a a a a a a a a a a				

Figure C-29 VL_PACKING_4444_10_in_16_R

This packing is

- RGBA: VL_PACKING_4444_10_in_16_R + VL_COLORSPACE_{RGB,RP175}
- VYUA: VL_PACKING_4444_10_in_16_R + VL_COLORSPACE_{CCIR,YUV}

Figure C-30 shows VL_PACKING_4444_12_in_16_L, a 64-bit RGBA packing. For an explanation of the p bit, see “Packing Diagram Conventions” on page 48.

Pixel 1							
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
r r r r r r r r r r	r r r r p p p p	g g g g g g g g g g	g g g g p p p p	b b b b b b b b b b	b b b b p p p p	a a a a a a a a a a	a a a a p p p p

Figure C-30 VL_PACKING_4444_12_in_16_L

This packing is VL_PACKING_4444_12_in_16_L + VL_COLORSPACE_{RGB,RP175} in the VL, new style.

The components in this packing are signed:

- For RGB and RP-175 color spaces, they are often negative.
- For the YUV color space, they are negative when the incoming video signal has values outside of the nominal range.
- For the CCIR601 color space, they are always positive.

Figure C-31 shows VL_PACKING_4444_12_in_16_R, a 64-bit RGBA packing for use with extended RGB components.

Pixel 1							
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
s s s s r r r r	r r r r r r r r	s s s s g g g g	g g g g g g g g	s s s s b b b b	b b b b b b b b	s s s s a a a a	a a a a a a a a

Figure C-31 VL_PACKING_4444_12_in_16_R

This packing is VL_PACKING_4444_12_in_16_R + VL_COLORSPACE_{RGB,RP175} in the VL, new style.

Note: The components in this packing are signed, with positive and negative values varying by color space, as explained for VL_PACKING_4444_12_in_16_L. The s in the more significant bits in Figure C-31 indicates a sign extension padding bit.

Figure C-32 shows VL_PACKING_4444_13_in_16_L, a 64-bit RGBA packing for use with extended RGB components. For an explanation of the p bit, see “Packing Diagram Conventions” on page 48.

Pixel 1							
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
r r r r r r r r	r r r r p p p	g g g g g g g g	g g g g p p p	b b b b b b b b	b b b b p p p	a a a a a a a a	a a a a p p p

Figure C-32 VL_PACKING_4444_13_in_16_L

This packing is VL_PACKING_4444_13_in_16_L + VL_COLORSPACE_{RGB,RP175} in the VL, new style.

The components in this packing are signed, with positive and negative values varying by color space, as explained for VL_PACKING_4444_12_in_16_L.

Figure C-33 shows VL_PACKING_4444_13_in_16_R, a 64-bit packing for use with extended RGB components.

Pixel 1							
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
s s s r r r r r	r r r r r r r r	s s s g g g g g	g g g g g g g g	s s s b b b b b	b b b b b b b b	s s s a a a a a	a a a a a a a a

Figure C-33 VL_PACKING_4444_13_in_16_R

This packing is VL_PACKING_4444_13_in_16_R + VL_COLORSPACE_{RGB,RP175} in the VL, new style.

Note: The components in this packing are signed, with positive and negative values varying by color space, as explained for VL_PACKING_4444_12_in_16_L. The *s* in the more significant bits in Figure C-33 indicates a sign extension padding bit.

Sampling Patterns

Sampling patterns are

- “4:4:4 and 4:4:4:4 Sampling” on page 76
- “4:2:2 and 4:2:2:4 Sampling” on page 77

4:4:4 and 4:4:4:4 Sampling

Some of the diagrams in the “DIVO Pixel Packings” section indicate 4:4:4 or 4:4:4:4 sampling. This video industry terminology means that each of the three or four components is sampled at every pixel. Figure C-34 diagrams this sampling pattern.

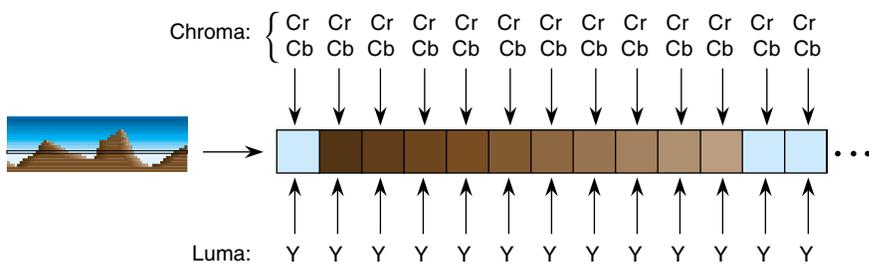


Figure C-34 4:4:4 Sampling

4:2:2 and 4:2:2:4 Sampling

The packings shown in diagrams that indicate 4:2:2 sampling make sense only in the VYUA color spaces. For every two pixels, there are two luma samples (two Y's) but only one chroma sample (one sample of Cr and Cb, which together determine the chroma), as shown in Figure C-35.

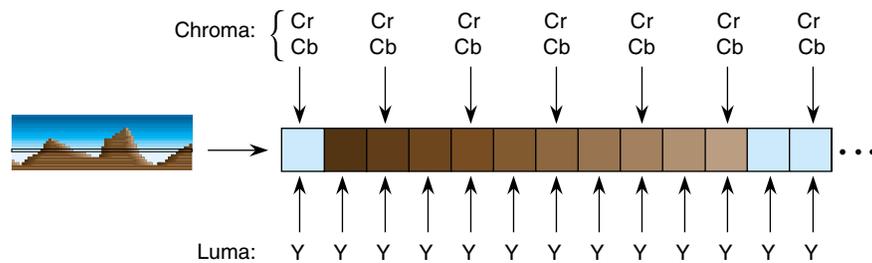


Figure C-35 4:2:2 Sampling

The chroma samples belong at the same instant in space as the left Y sample (the chrominance samples and the left Y are co-sited). The diagrams for 4:2:2 packings in the “DIVO Pixel Packings” section of this appendix show the location of each Y, Cr, or Cb component as left or right. The first pixel of each line is a left pixel.

Converting 4:4:4 video to 4:2:2 video is like converting 44.1 kHz audio into 22.05 kHz audio: just dropping every other Cr,Cb sample yields extremely poor results. Video devices that need to convert between 4:4:4 and 4:2:2 use carefully designed filters. The characteristics of the required filter are specified in ITU-R BT.601-4 (Rec. 601).

4:2:2 sampled packings that also include alpha are called 4:2:2:4. This method has one alpha value per pixel, like the Y value.

Color Spaces

Each component of an image has

- a color that it represents
- a canonical minimum value
- a canonical maximum value

Normally, a component stays within the minimum and maximum values. For example, for a luma signal such as Y, you can think of these limits as the black level and the peak white level, respectively. For an unsigned component with n bits, there are two possibilities for [minimum value, maximum value]:

- full range: $[0, (2^n)-1]$, which provides the maximum resolution for each component
- compressed (headroom) range, which provides numerical headroom, which is often useful when processing video images:
 - Cr and Cb: $[(2^n)/16, 15*(2^n)/16]$
 - Y, A, R, G, and B: $[(2^n)/16, 235*(2^n)/256]$

Compressed range is defined for 8 and 10 bits in ITU-R BT.601-4 (Rec. 601). For example:

- for 8-bit components:
 - Cr and Cb: [16, 240]
 - Y, A, R, G, B: [16, 235]
- for 10-bit components:
 - Cr and Cb: [64, 960]
 - Y, A, R, G, B: [64, 940]

Two sets of colors are commonly used together, RGB (RGBA) and YCrCb/YUV (VYUA). YCrCb (YUV), the most common representation of color from the video world, represents each color by a luma component called Y and two components of chroma, called Cr (or V), and Cb (or U). The luma component is loosely related to brightness or luminance, and the chroma components make up a quantity loosely related to hue. These components are defined rigorously in ITU-R BT.601-4 (also known as Rec. 601 and CCIR 601).

The alpha channel is not a real color. For that channel, the canonical minimum value means completely transparent, and the canonical maximum value means completely opaque.

For more information about color spaces, see *A Technical Introduction to Digital Video*, by Charles A. Poynton (New York: Wiley, 1996).

Determining the Color Space

For OpenGL, IRIS GL, and DM:

- the library constant indicates whether the data is RGBA or VYUA
- RGBA data is full-range by default
- VYUA data in DM can be full-range or compressed-range; you must determine this from context

Using the traditional VL_PACKING tokens from IRIX 6.2, the VL_PACKING constant indicates whether the data is RGBA or VYUA (as in VL_PACKING_UYV_8_P). The VL that comes with the DIVO option (for IRIX 6.4) makes all of the parameters (packing, set of colors, range of components) explicit:

- Use VL_PACKING to specify only the memory layout. The new memory-only VL_PACKING tokens are disjoint from the old, and the old tokens are still honored, so this change is backward-compatible.
- Use VL_COLORSPACE to specify the color space parameters, as shown in Table C-2.

Table C-2 VL_COLORSPACE Options

Color Set	Full-Range Components	Compressed-Range Components
RGBA	VL_COLORSPACE_RGB	VL_COLORSPACE_RP175
VYUA	VL_COLORSPACE_YUV	VL_COLORSPACE_CCIR

The option VL_COLORSPACE_NONE is useful when you want to treat CCIR 601 digital video as a raw 10-bit data stream (as in SDDI).

DIVO performs color-space conversion if the color space implied by VL_FORMAT on the video node disagrees with that implied by VL_COLORSPACE on the memory node.

Color-Space Conversions

The DIVO board supports four native color spaces—RGB, YUV, CCIR, and RP-175 compressed RGB. The choice of color space is determined by the format control for video sources and drains and by the color-space controls for memory sources and drains. If the color space selected for memory sources and drains matches that used by the current video format, no color-space conversions are performed. When DIVO performs color-space conversions, extreme care is taken to assure the correctness and precision of the result.

Understanding the capabilities of DIVO to perform color space conversions and the results of these conversions allows developers and end users to maximize the quality of their output. This appendix explains

- “DIVO Color Spaces” on page 82
- “Mathematical Operations Performed During Conversions” on page 84
- “Implications of Color Space Conversions” on page 85

The appendix concludes with examples.

DIVO Color Spaces

The DIVO option uses a minimum of ten bits of precision for each color component at all steps of its internal pipeline. Representations for the four native internal color representations are explained separately in this section.

RGB

RGB is the color space used by the graphics subsystem. RGB has the most accurate representation of visible colors, because all possible combinations are valid. This color space does not support superblack or other nonvisible color values. Each component is represented by a 10-bit value between 0 and 1023. Black has the value [0,0,0], and white is [1023,1023,1023]. Table D-1 summarizes the clamping range for each resulting RGB component for various conversions.

Table D-1 Clamping Ranges for RGB Component Conversions

When converting to...	Each resulting RGB component is clamped to the range
10-bit RGB	[0..1023]
8-bit RGB	[0..255]
12-bit signed RGB	[-2048..2047]
13-bit signed RGB	[-4096..4095]

DIVO uses this color space only at the memory interface.

Note: You should not normally use 4:2:2 coding with RGB data.

YUV

The YUV color space is obtained from RGB by the matrix transformation in Equation 1.

$$\text{Equation 1} \quad \begin{bmatrix} 0.500 & -0.419 & -0.081 \\ 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \end{bmatrix} \times \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 512 \\ 0 \\ 512 \end{bmatrix} = \begin{bmatrix} V \\ Y \\ U \end{bmatrix}$$

The V, Y, and U values range from [0..1023]. Black has the VYU value [512,0,512]. White has the value [512,1023,512].

DIVO uses this color space only at the memory interface. With proper filtering, 4:2:2 coding can be used.

CCIR

The CCIR color space is obtained from RGB by the matrix transformation in Equation 2.

$$\text{Equation 2} \quad \begin{bmatrix} 0.500 & -0.419 & -0.081 \\ 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \end{bmatrix} \times \begin{bmatrix} R \\ G \\ B \end{bmatrix} \times \begin{bmatrix} \frac{896}{1023} \\ \frac{876}{1023} \\ \frac{896}{1023} \end{bmatrix} + \begin{bmatrix} 512 \\ 64 \\ 512 \end{bmatrix} = \begin{bmatrix} Cr \\ Y \\ Cb \end{bmatrix}$$

The Cr, Y, and Cb 10-bit values are clamped to the range [4..1019]. Black has the CrYCb value [512,64,512]. White has the value [512,940,512]. For 8 bits, the values are clamped to the range [1..254]; black has the CrYCb value [128,16,128], and white has the value [128,235,128].

This color space is used by the component digital formats. The memory interface can use this color space. With proper filtering, 4:2:2 coding can be used.

RP-175 Compressed RGB

The RP-175 color space is obtained from RGB by the matrix transformation in Equation 3.

$$\text{Equation 3} \quad \frac{876}{1023} \times \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 64 \\ 64 \\ 64 \end{bmatrix} = \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix}$$

When converting to 10-bit R'G'B', the R', G', and B' values are clamped to the range [0..1023]. Black has the R'G'B' value [64,64,64]. White has the value [940,940,940]. Other clamping ranges are the same as the standard RGB case.

This color space is used by the component digital RGB format. The memory interface can use this color space. You should not use 4:2:2 coding with this color space.

Mathematical Operations Performed During Conversions

DIVO can process and store each color space explained in the previous section. For best precision, the input color space should be maintained through the processing path. For example, an application that implements DDR functionality could choose to store data in the native representation of the input signal: Data from a D1 deck should be stored as a 8-bit 4:2:2 in the CCIR color space. Data from a dual-link telecine could be stored as 4:4:4 10-bit RP-175 RGB. If the application works in this way, no conversions are performed and the data is passed directly through the system. In particular, CCIR601 data coming from a D1 deck is bit-accurate in this case.

However, it might not be desirable for the application to work this way. If that is the case, the application can use all of the conversion, decimation and interpolation capabilities of the DIVO option to perform real-time color space and 4:2:2 \leftrightarrow 4:4:4 conversions.

Conversions are performed only when absolutely required. Each incoming or outgoing stream can be converted from its current color space to any other color space.

Implications of Color Space Conversions

The two major concerns when performing conversions from one color space to another are *precision* and *range*.

Precision of Color Conversions Done by DIVO

The DIVO board stores colors with a minimum of 10 bits of precision at all steps in its pipeline. When performing color space conversions, the data is converted to 13-bit signed values before being passed to the matrix multipliers. The matrix multipliers have 13-bit coefficients and 26-bit accumulators. The most significant 14 bits of the matrix-multiplication result are passed on to additional hardware, which applies any needed offsets and then clamps to the proper range.

Silicon Graphics, Inc., has verified both through simulation and hardware testing that the maximal error for two conversions (RGB to CCIR to RGB) is two units out of 1024. The matrix coefficients have been biased to round slightly high rather than slightly low to avoid the type of problems that can otherwise easily occur in the blue component.

Range Issues For Color Conversions Done by Any Means

Different color spaces allocate the available bits of precision in different ways. The RGB space is designed to maximize the accuracy of color representations. The YUV and CCIR color spaces are designed to strongly decouple chrominance and luminance information.

Since RGB represents visible colors, it is contained inside the YUV and CCIR spaces. The CCIR and RP-175 color spaces also have a slight amount of additional headroom that was intended to prevent aliasing artifacts when Finite Impulse Response filtering operations are performed on the digital data.

Any time a conversion operation is performed between CCIR and one of the other color spaces, the colors that are not representable in the destination color space must be somehow mapped into colors that are representable. The usual way to do this is to clamp each component to the available range in the destination color space. Other methods, such as projecting towards the center of the representable space, might produce results that appear to be better in some cases, but imply modification of the original signal and generally result in a loss of saturation.

During conversion from CCIR to YUV, the axes of the two spaces are parallel, so the result of this clamping operation is very predictable. Superblack and superwhite are clipped to black and white, respectively, and oversaturated colors might also be clipped.

During conversion from RGB to YUV or CCIR, clamping never occurs, because all RGB colors are representable in those color spaces.

During conversion from CCIR or YUV to RGB or RP-175 RGB, the results of clamping are much less intuitive, because these conversions involve rotation and scaling operations, with the result that the component axes in one color space don't align with those in the other.

DIVO also supports signed RGB representations with 12 or 13 significant bits. If one of these representations is used, the entire CCIR color space is representable and no clamping will occur. Application software must specifically select this mode and handle the (12/13)-bit data to gain this benefit.

Figure D-1 shows the RGB color cube inside the CCIR color space. The volume contained within the outer (CCIR) cube, but outside the inner (RGB) cube, represents "illegal" colors that cannot be displayed.

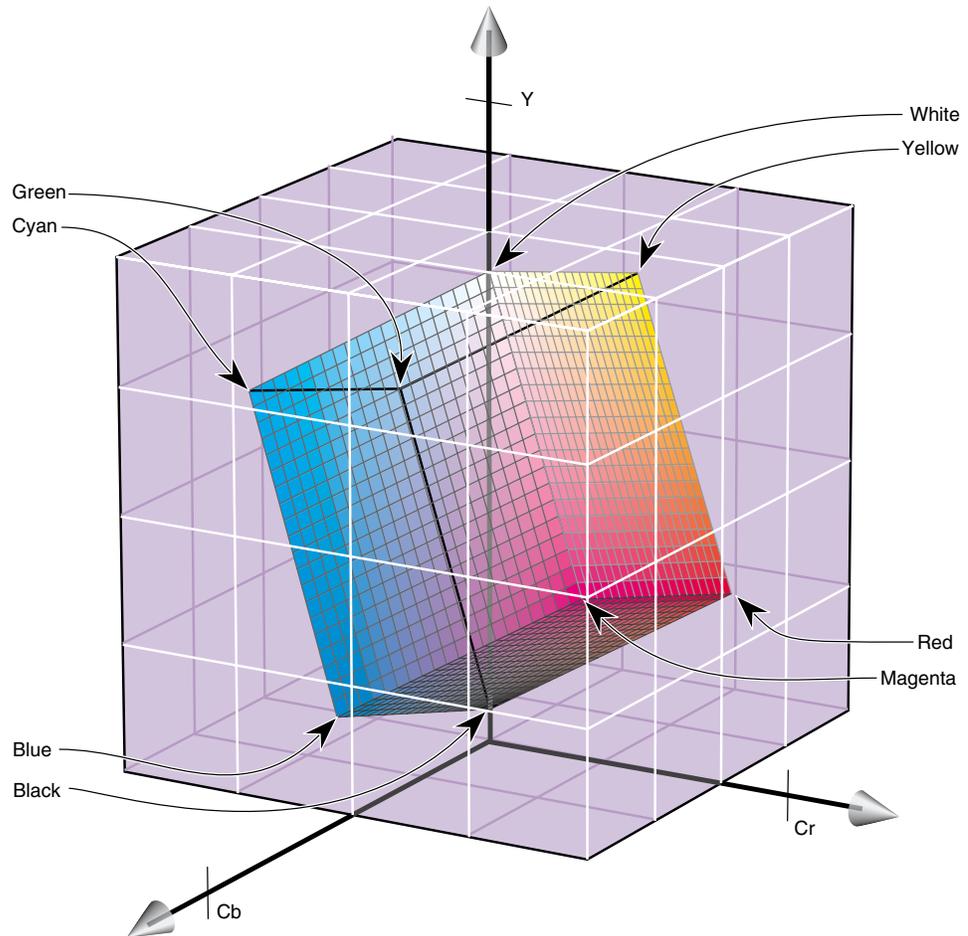


Figure D-1 RGB Cube in CCIR Space

As shown in the figure, the CCIR color space allocates almost three quarters of its available bit combinations to illegal colors. When any of these color values are converted to RGB, the result is clamped to the edge of the RGB cube. Since the inner cube contains the displayable colors, this clamping operation has no impact on them.

If CCIR is converted to RGB and back to CCIR using certain types of test signals, the output can appear to be vastly wrong. A common and extreme version of this is the signal that simultaneously ramps Cr, Y, and Cb from the minimum to maximum possible values.

In Figure D-2, the heavy diagonal line passing through the figure is the set of colors in the luma/chroma ramp test signal. As shown in the figure, a large portion of this pattern is outside the RGB cube. In fact, over two thirds of this pattern is outside the displayable range.

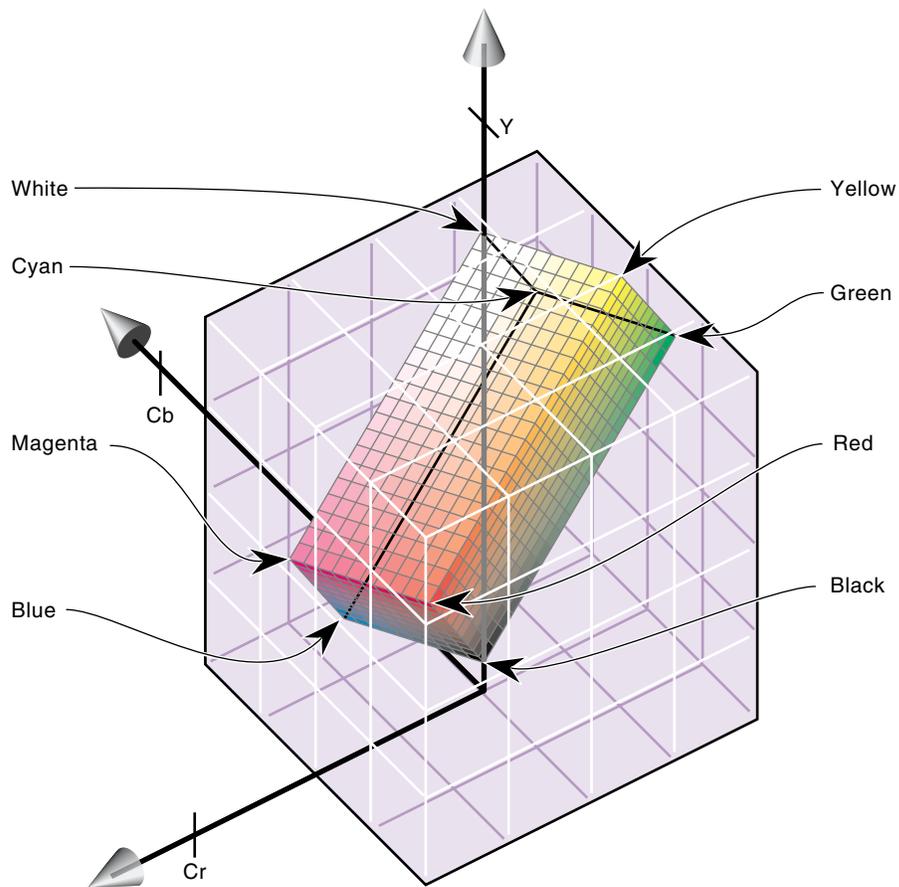


Figure D-2 Color Cube With Luminance/Chrominance Ramp Vector

Example Color Conversions

This section includes example graphs that display the results of converting from CCIR to eight or ten bit RGB and back. They show the same type of result you would see if you passed a digital signal through DIVO using the *soft_ee* program with RGB as the color space and an eight or ten-bit data-packing. If you use CCIR as the memory color space or use a data-packing with 12/13-bit signed representations, the output will look exactly like the input. If the memory color space matches the video color space, the output will be a bit-perfect copy of the input.

Example 1: 100% Color Bars

This example, like the other two in this section, consists of three graphs. Each graph displays the input CCIR pattern, intermediate RGB pattern, and output CCIR pattern for a given color component. Figure D-3 shows the red and Cr components, Figure D-4 the green and Y components, and Figure D-5 the blue and Cb components. In this example and the others, if the input and output CCIR values are identical, only two lines are shown.

In this example, conversion to RGB and back has no effect on the image. The 100% amplitude color bar signal lies within the visible range and therefore is perfectly represented in RGB.

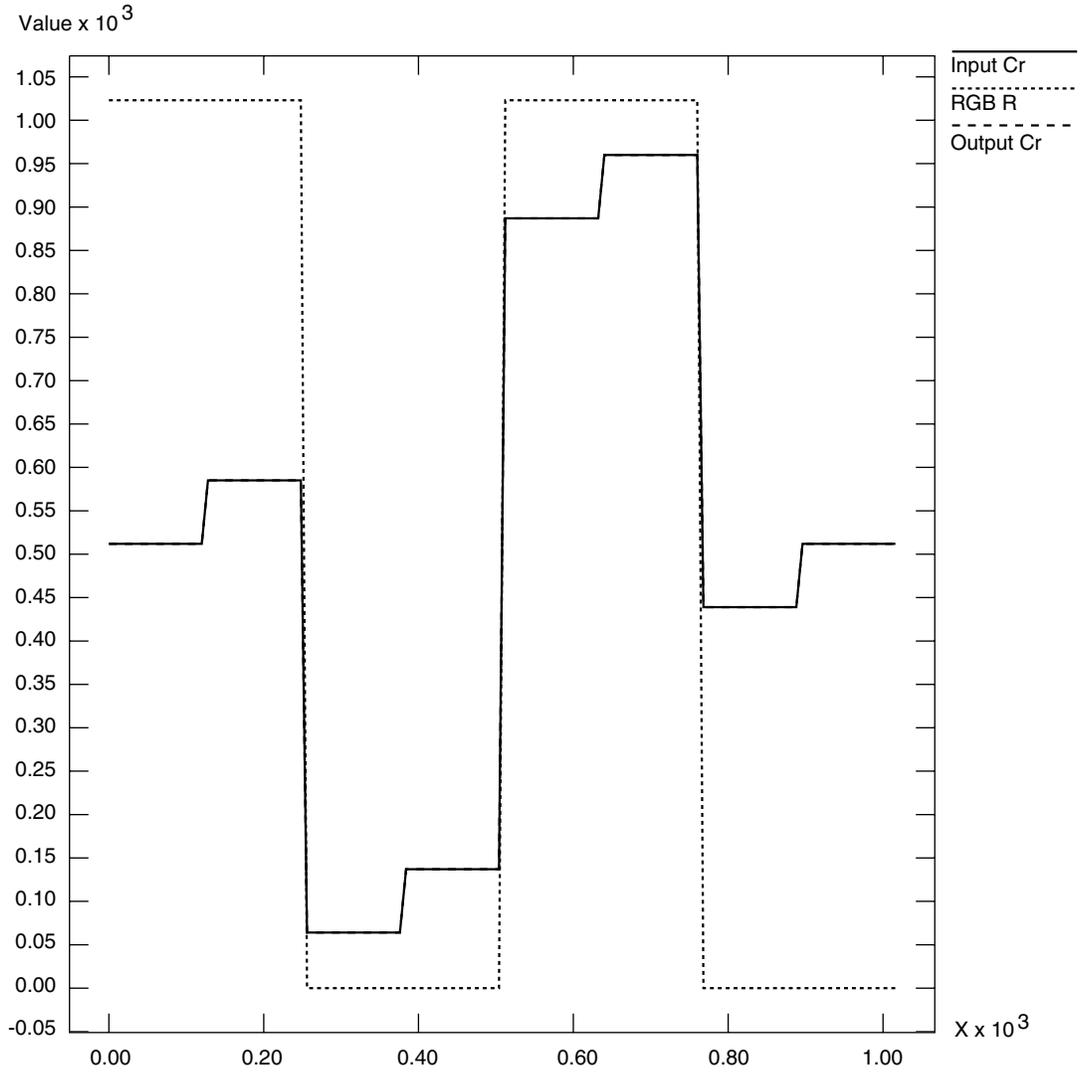


Figure D-3 100% Color Bars: Cr/R

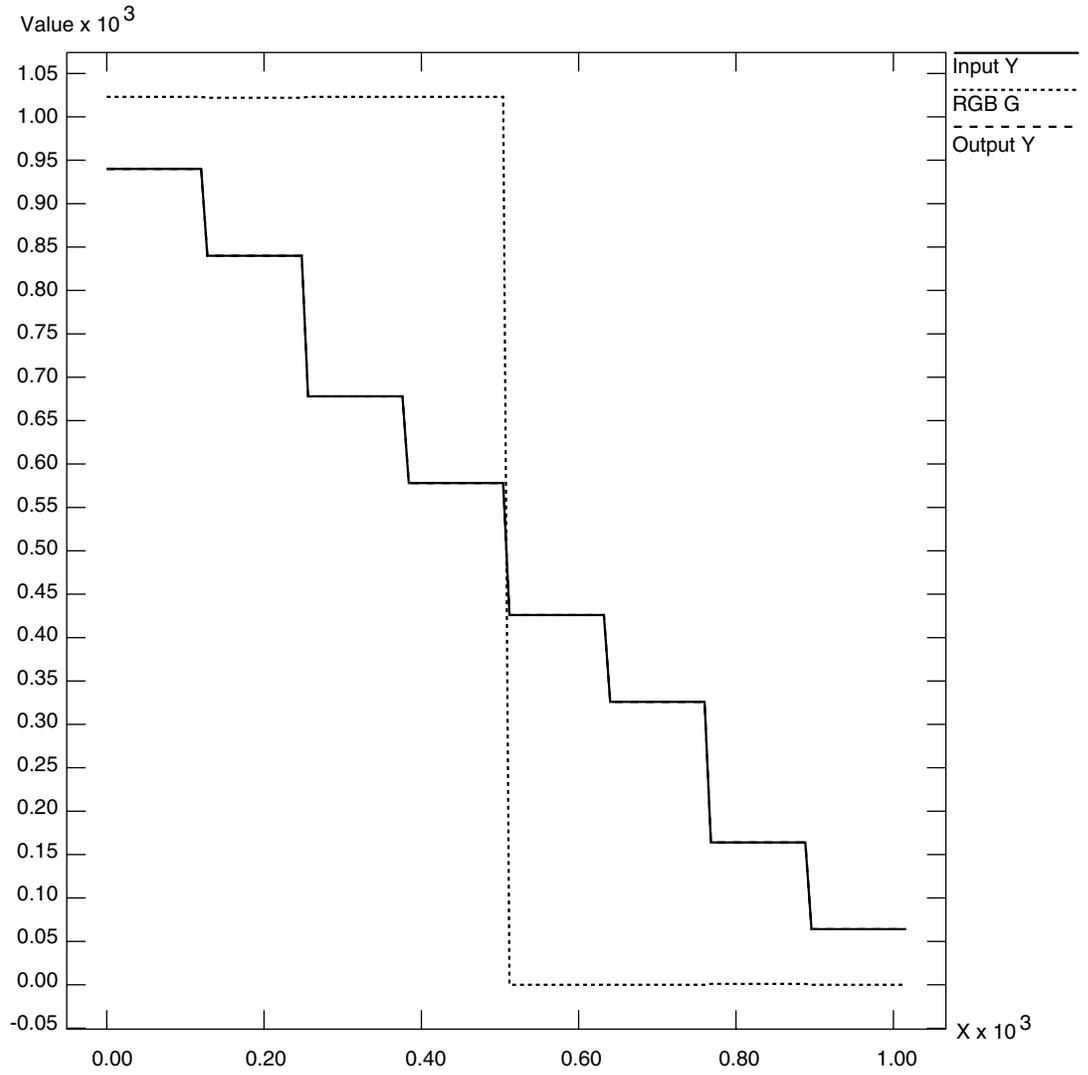


Figure D-4 100% Color Bars: Y/G

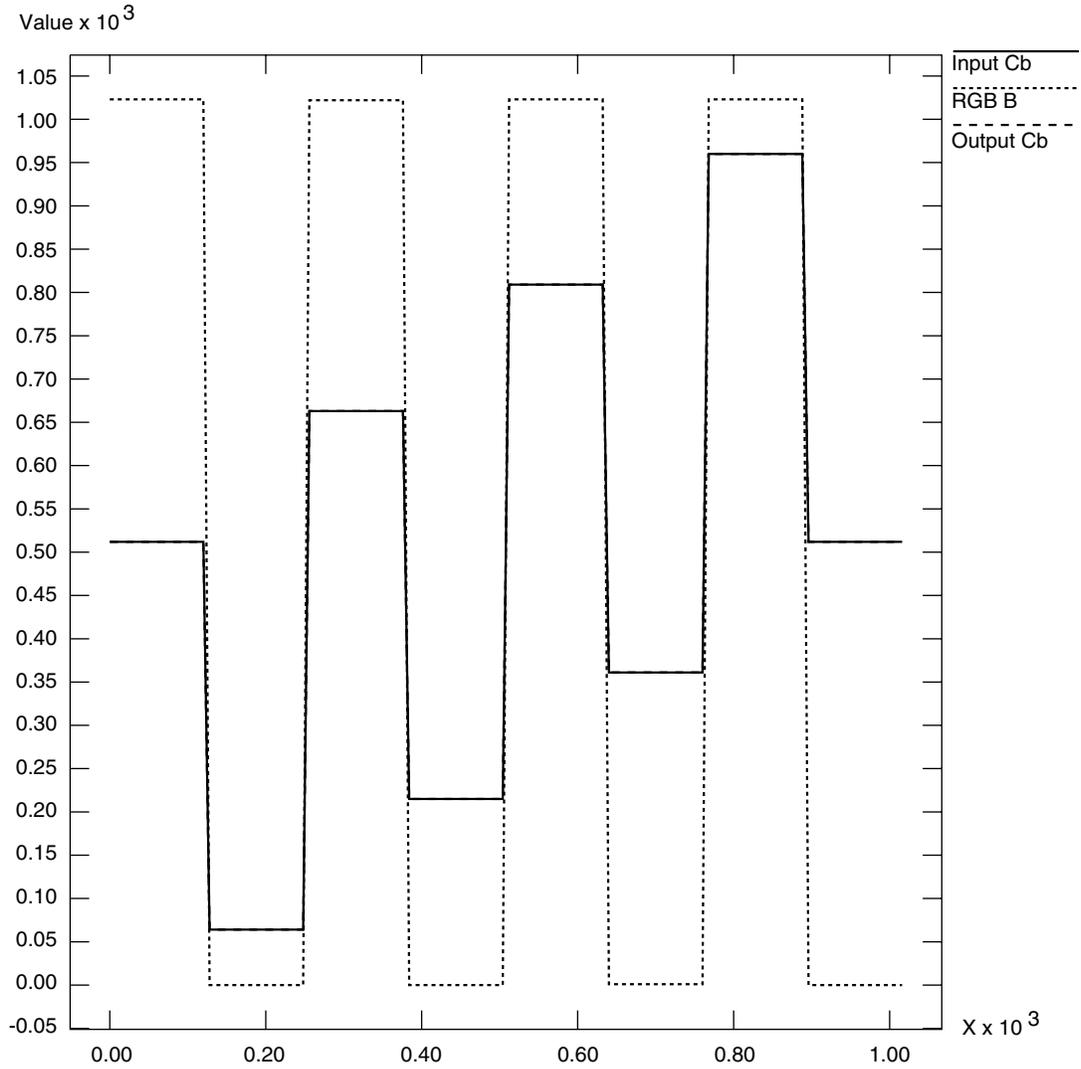


Figure D-5 100% Color Bars: Cb/B

Example 2: Luminance Ramp

In this example, the conversion to RGB and back affects only the superblack and superwhite regions. All luminance values that are blacker than black are clamped to black; all values whiter than white are clamped to white.

In the RGB color space, each component ramps from 0 to 1023 as the input luminance ramps from 64 (black) to 940 (white). This test pattern lies along the Y axis of the color cubes.

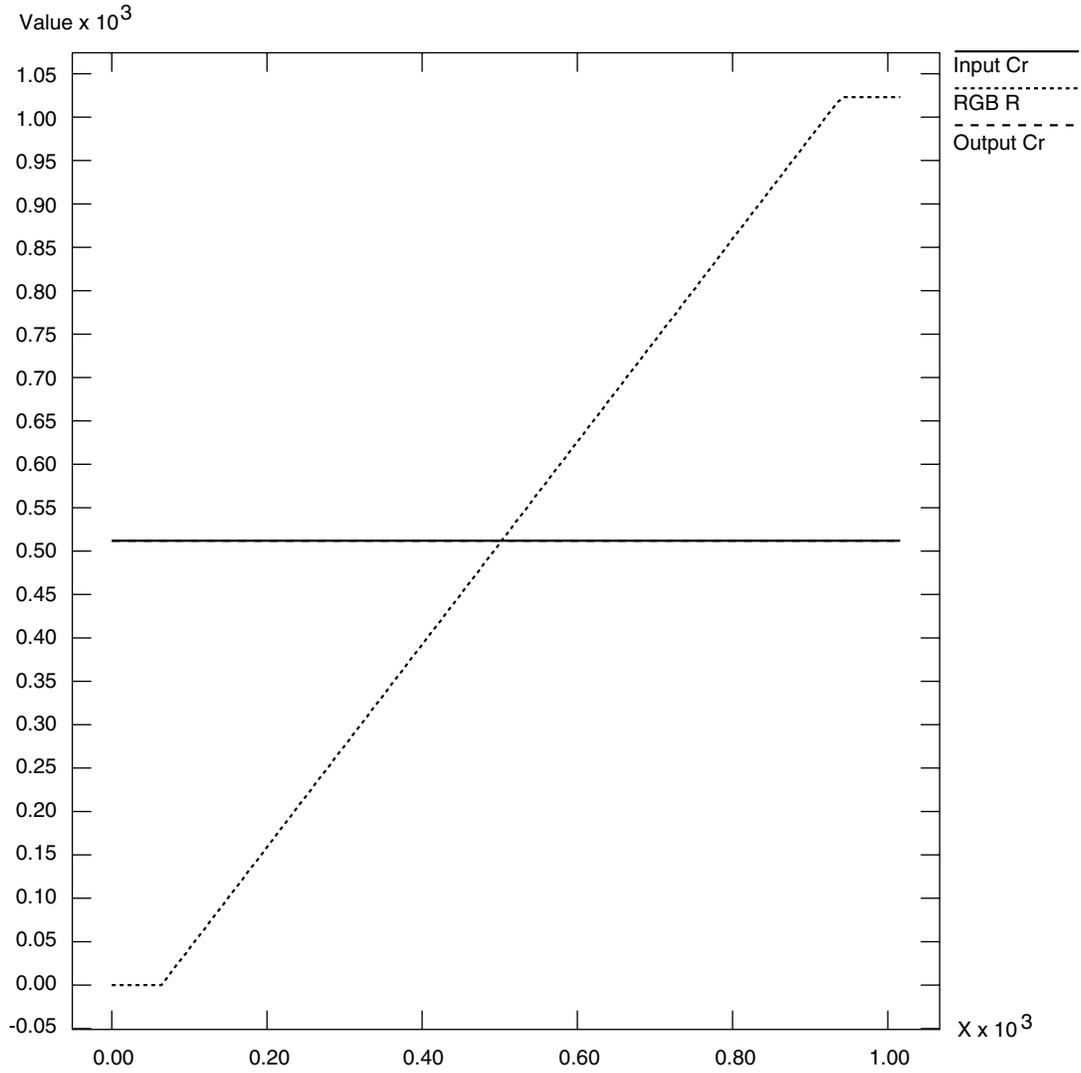


Figure D-6 Luminance Ramp: Cr/R

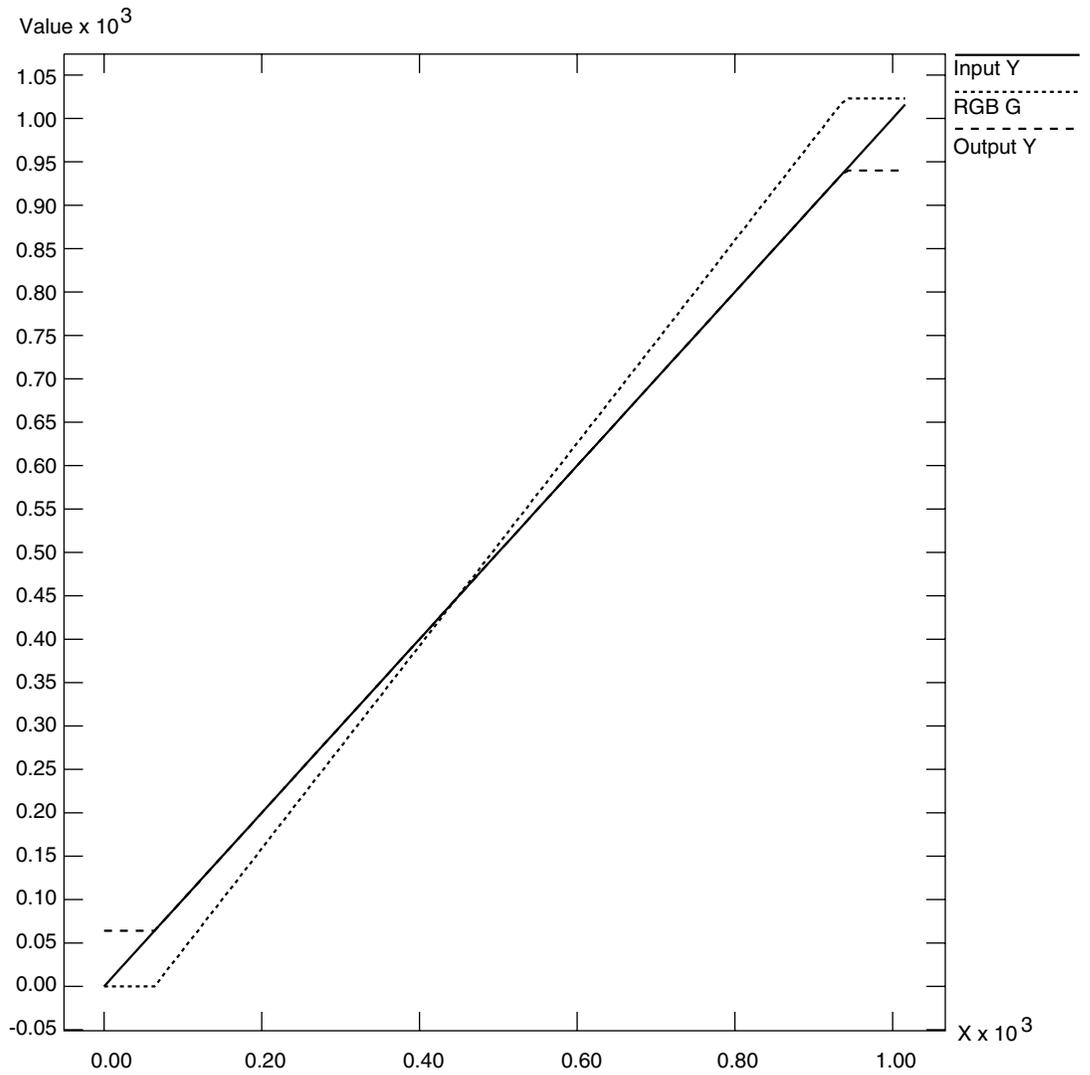


Figure D-7 Luminance Ramp: Y/G

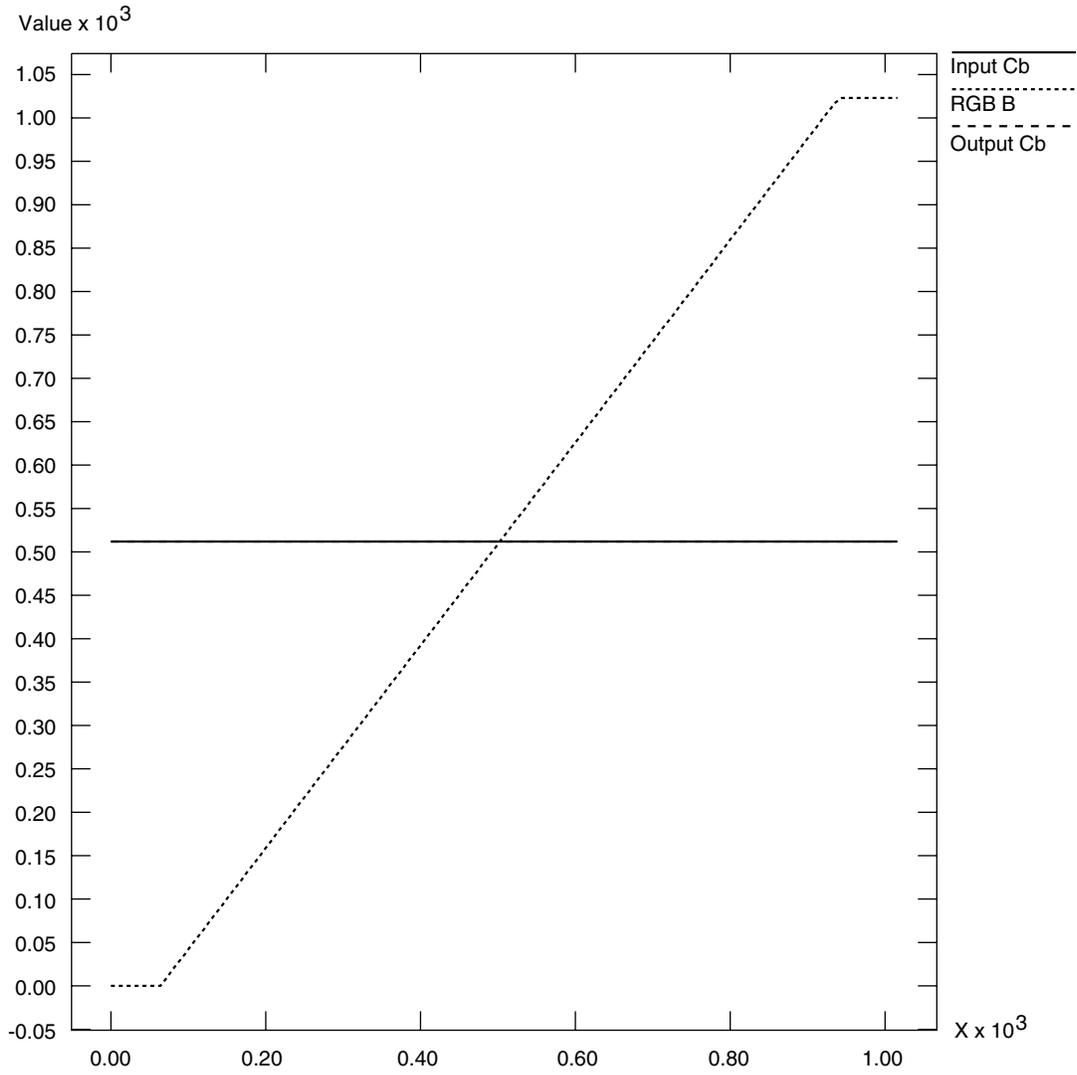


Figure D-8 Luminance Ramp: Cb/B

Example 3: Simultaneous Chroma/Luma Ramp

This example is the most extreme of the three, and shows how surprising the results of color conversions can be when arbitrary synthetic CCIR inputs are used.

Each CCIR input signal ramps from 0 to 1023 simultaneously. As mentioned in the first example, over two thirds of this pattern lies outside the legal range. The portion within the legal range is represented exactly, but the region outside is clamped to the RGB cube surface.

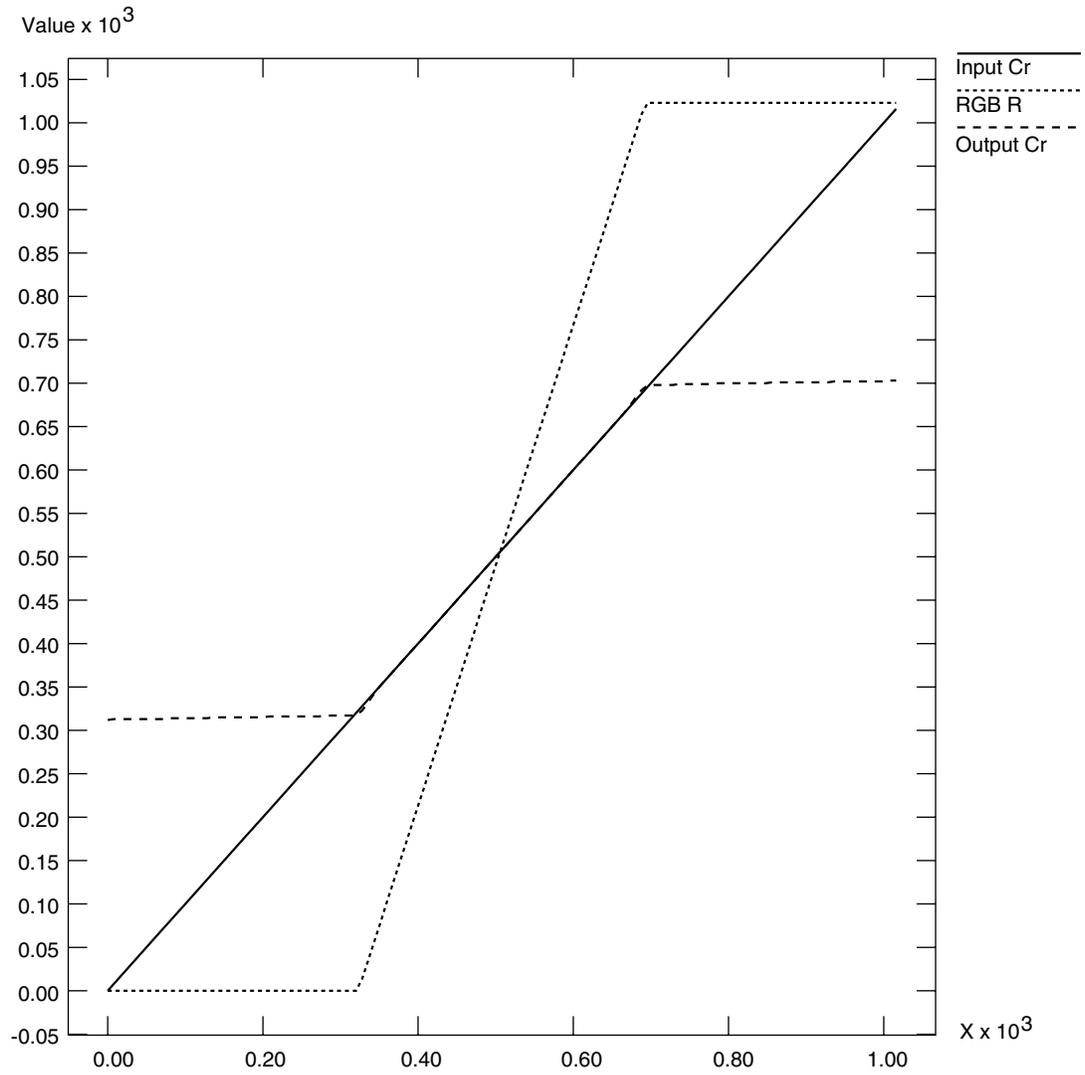


Figure D-9 Chroma/Luma Ramp: Cr/R

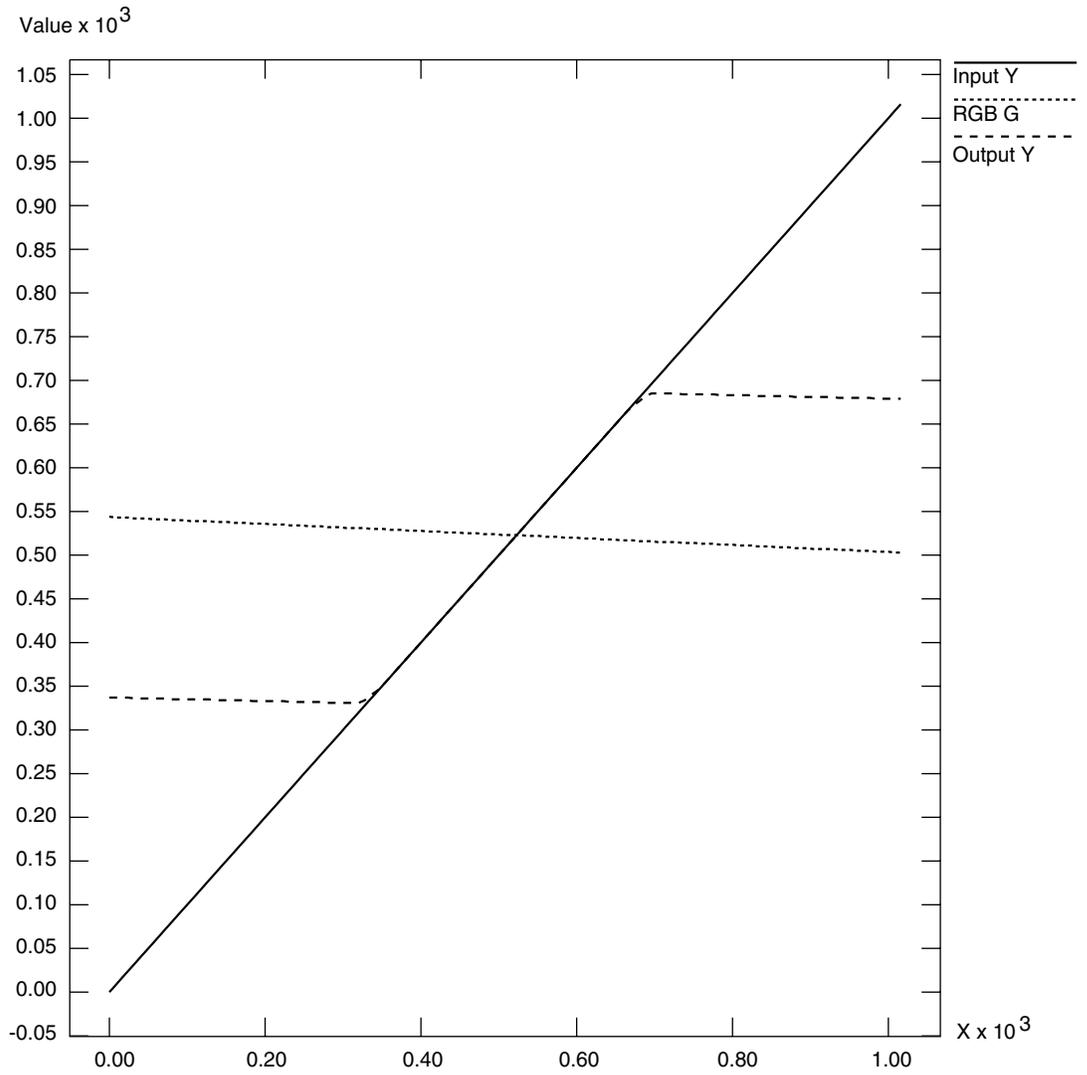


Figure D-10 Chroma/Luma Ramp: Y/G

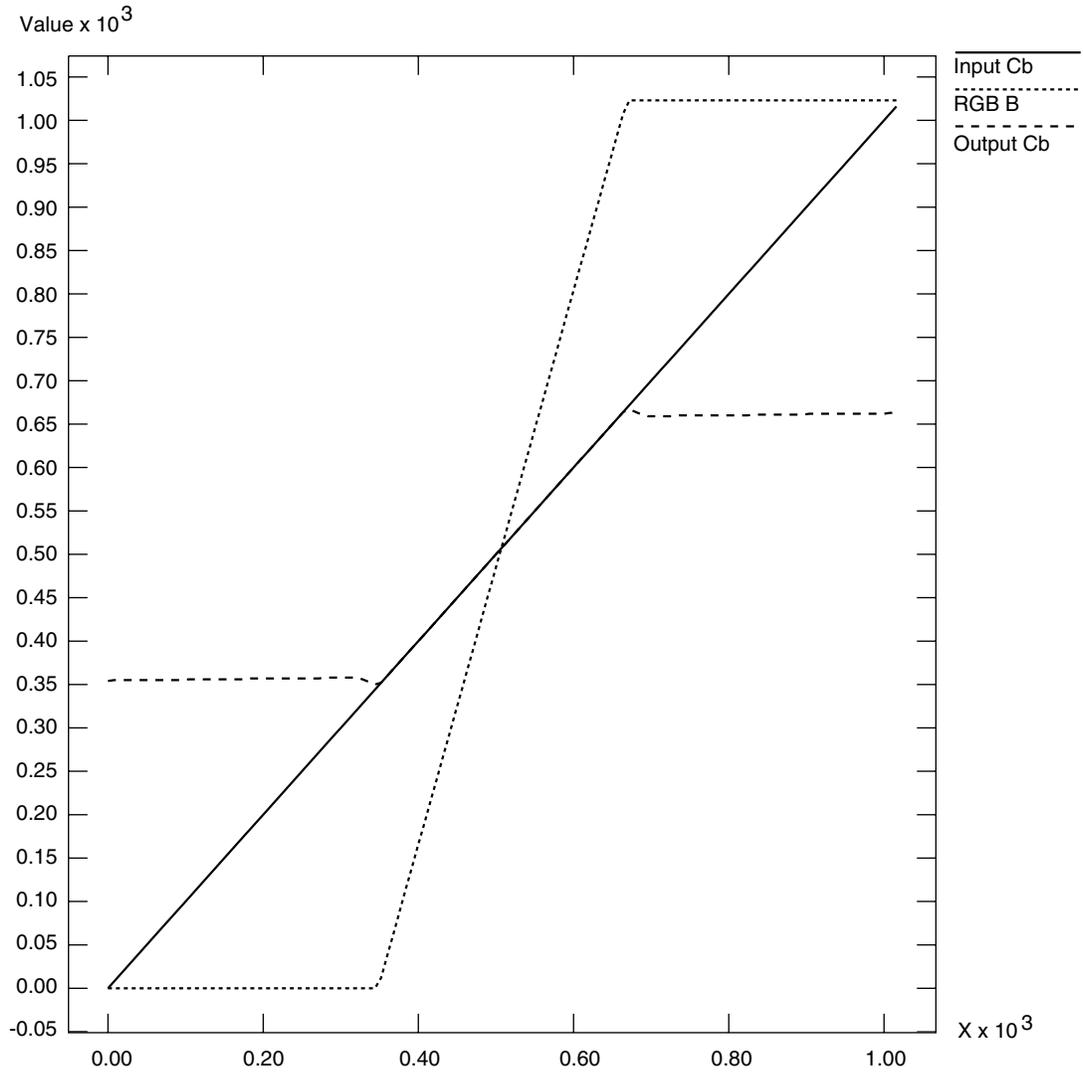


Figure D-11 Chroma/Luma Ramp: Cb/B

Programming Methods for Real-Time Digital Media Recording and Playback

This appendix explains the following real-time disk I/O concepts:

- “Direct I/O” on page 102
- “Scatter/Gather I/O” on page 103
- “Multiprocessing” on page 107
- “Asynchronous I/O” on page 108
- “File Formats” on page 108

The example source for the utilities discussed in this appendix can be found in `/usr/share/src/dmedia/tools`. The code examples are written to Digital Media buffers (DMbuffers), a real-time data transport facility. See the *Digital Media Programming Guide* (document number 007-1799-060 or later, hereafter referred to as the DMPG) for more details. The emphasis here is not on how data is acquired from or transported to the video device, but rather on how data is moved to disk in real time.

The DMPG covers basic digital media programming concepts; two simple programming examples in `/usr/share/src/dmedia`, `vidtomem.c` and `memtovid.c`, illustrate how video data is copied into and out of the DMbuffers for the simpler non-real-time case. Programming examples can also be found in the developer’s toolbox, <http://www.sgi.com/Technology/toolbox.html>. At an abstract level, high-bandwidth throughput is simple; the work is in the details, as explained in this appendix.

Direct I/O

The most efficient way to move data on and off a disk device is to use the XFS filesystem with direct I/O mode and large data transfer sizes. If large transfer sizes cannot be achieved, you can combine memory pages from noncontiguous locations using `writev(2)` or `readv(2)`. Finally, you can use asynchronous I/O to queue multiple I/O requests to the kernel without waiting for blocked calls to return. Other real-time software features and products, such as REACT, can be used to assure low-latency interrupts and high-priority scheduling, but are not absolutely necessary for digital media applications.

Normally, when a disk file is opened with no status flags specified, a call to `write(2)` for that file returns as soon as the data has been copied to a buffer managed by the device driver (see `open(2)`). The actual disk write may not take place until considerable time has passed. A common pool of disk buffers is used for all disk files.

Disk buffering is integrated with the virtual memory paging mechanism. A daemon executes periodically and initiates output of buffered blocks according to the age of the data and the needs of the system. You can force the writing of all pending output for a file by calling `fsync(2)` or by opening the file and specifying the `O_SYNC` flag. However, the process blocks until the data has been written to disk, and all output data must still be copied from the buffer in the user address space to a buffer in the kernel address space. See Chapter 8, "Optimizing Disk I/O for a Real-Time Program," in the *REACT Real Time Programmer's Guide* for details.

If you use the `O_DIRECT` flag, writes to the file take place directly from your program's buffer, and the data is not copied to a buffer in the kernel first. Because the filesystem cache is bypassed, your application must manage buffer alignment and block size specification. To use `O_DIRECT`, you must transfer data in quantities that are multiples of the filesystem block size. The following code shows how to query the filesystem block size and system DMA transfer size limit.

```
struct dioattr da;
struct stat fileStat;
char *ioFileName = "videodata";
int ioBlockSize, ioMaxXferSize;

ioFileFD = open(ioFileName,O_DIRECT | O_RDWR | O_CREAT | O_TRUNC,0644);
if (ioFileFD < 0)
    return(DM_FAILURE);
if (fcntl(ioFileFD, F_DIOINFO, &da) < 0)
    return(DM_FAILURE);
ioBlockSize = da.d_miniosz;
ioMaxXferSize = da.d_maxiosz;
```

The two important constraints of direct I/O with XFS are memory address alignment and buffer length. Direct I/O requires all memory addresses to be page-aligned. XFS requires buffers to be allocated as a multiple of the filesystem block size, *ioBlockSize*. DMbuffers are guaranteed to be page-aligned, but to ensure that the buffers are properly padded, you must set the buffer size, *bytesPerXfer*, to the size of the image data you will transfer rounded up to the nearest multiple of *ioBlockSize*.

```
VLServer vlServer;
VLPPath vlPath;
DMparams * paramsList;
int dmBufferPoolSize = 30; /* 1 second of video */
int vlBytesPerImage = vlGetTransferSize(vlServer, vlPath);
int ioBlocksPerImage = (vlBytesPerImage+ioBlockSize - 1) / ioBlockSize;
int bytesPerXfer = ioBlocksPerImage * ioBlockSize;

if (dmBufferSetPoolDefaults(paramsList, dmBufferPoolSize, bytesPerXfer,
    DM_TRUE, DM_TRUE) == DM_FAILURE) {
    fprintf(stderr, "error setting pool defaults\n");
    return(DM_FAILURE);
}
```

All Silicon Graphics systems have a configurable maximum DMA transfer size (see *system(1M)*). This value should be compared with the user's I/O request size.

```
if (bytesPerXfer > ioMaxXferSize) {
    fprintf("DMA request size is too small. Reconfigure with
        system()\n");
    return(DM_FAILURE);
}
```

Scatter/Gather I/O

As shown in DMPG Chapter 5, "Digital Media Buffers," and in the example programs *vidtomem.c* and *memtovid.c*, video data is generally transported to or from DMbuffers one image at a time using standard write and read functions that specify the number of bytes and a pointer to a buffer. However, large reads and writes can usually increase I/O performance. This technique reduces the number of transactions performed between the application, operating system, and I/O device, and can allow the device to optimize some of its activities. These advantages are particularly true with disk arrays.

Since the DMbuffer's memory pages are not guaranteed to be contiguous, standard reads or writes cannot be made across multiple buffers. The `readv(2)` and `writv(2)` interfaces allow an application to provide a list of I/O vectors, which are data structures consisting of an address and byte-count pair. Because the list of vectors is submitted to the operating system as a unit, it can be treated as a single large I/O request. Using `readv()` and `writv()` with direct I/O is particularly efficient.

The restrictions on buffer alignment and block size for `readv()/writv()` are similar to those of direct I/O. The address for each I/O vector must be page-aligned, but the length of each I/O vector must be a multiple of the system page size, rather than the filesystem block size, as is the case with direct I/O. Thus, the easy solution is to always use the larger of the two values, page size or filesystem block size. This requirement wastes some space, but is necessary to maintain functionality and performance. This calculation must be performed before `dmBufferSetPoolDefaults(3dm)` is called.

```
int ioAlignment, ioBlockSize;
ioAlignment = getpagesize();
if (ioAlignment > ioBlockSize)
    ioBlockSize = ioAlignment;
```

The maximum allowable number of I/O vectors can be queried with `sysconf(3C)`.

```
int ioVecCount=2; /* set default to two images */
long ioVecCountMax;
/* check for range */
ioVecCountMax = sysconf(_SC_IOV_MAX);
if (ioVecCount > ioVecCountMax) {
    ioVecCount = ioVecCountMax;
    fprintf(stderr, "cannot create more than %d I/O vectors\n",
            ioVecCountMax);
}
else if (ioVecCount <= 0)
    ioVecCount = 2;
```

The aggregate size of all the I/O vectors cannot exceed the maximum DMA transfer size, so you must check for this condition and adjust the number of I/O vectors if necessary:

```
int ioVecCount=2; /* set default to two images */
if (bytesPerXfer * ioVecCount > ioMaxXferSize)
    ioVecCount = ioMaxXferSize/bytesPerXfer;
```

When you work with video data using `readv()`/`writev()`, it is much easier to manage frames or an even number of fields with one I/O vector per field or frame. Most Silicon Graphics video devices can support either field or frame mode, which is selected with the `VL_CAPTURE_TYPE` device control (see Chapter 4, “Video I/O Concepts” of the DMPG). Hereafter, the term video image refers to a video data quantum: field or frame, depending on how the hardware is set up. The restriction of working on frame or even field boundaries is also relevant to the data file format, which is discussed at the end of this appendix.

The following code fragment illustrates writing to disk. Upon the successful capture of a video image, the `VLTransferComplete` event is placed on the event queue. A pointer to a valid `DMbuffer` is returned by `vlDMBufferGetValid(3dm)`; then the actual video data is mapped into user space. Data is not written to disk until there are enough video images to complete an I/O vector.

case `VLTransferComplete`:

```

/* loop until we get a valid buffer */
while (((retval = vlDMBufferGetValid(vlServer, vlPath, vlDrnNode,
&dmbuffers[dmbuffer_index])) != VLSuccess) && (vlErrno == VLAGain))
    sginap(1);
if (retval == VLSuccess) {
    /* map data to I/O vectors */
    (videoData+iov_index)->iov_base =
        dmBufferMapData(dmbuffers[dmbuffer_index]);
    (videoData+iov_index)->iov_len = bytesPerXfer;

    /* increment the buffer index for the next image */
    dmbuffer_index = (dmbuffer_index+1) % dmBufferPoolSize;

    /* write data to disk when we have enough I/O vectors */
    if (!(++iov_index % ioVecCount)) {
        first_index = vlXferCount - iov_index + 1
        dataOffset = (off64_t) vlXferCount *
            (off64_t) bytesPerXfer;
        /* seek to the correct position in the file, must use
        * lseek64() as the 64-bit offset value is necessary for
        * XFS filesystems larger than 2 gigabytes
        */
        if (lseek64(ioFileFD, dataOffset, SEEK_SET) != dataOffset)
            return(DM_FAILURE);
    }
}

```

```

/* write the I/O vector to disk */
if (writev(ioFileFD, videoData, ioVecCount) < 0)
    return(DM_FAILURE);

/* the dmuffers are managed as a ring buffer,
 * dmuffer_free_index points to the next free buffer */
for (i=0, dmuffer_free_index = (dmuffer_index - 1);
     i < iov_index; i++, dmuffer_free_index--) {
    if (dmuffer_free_index < 0)
        dmuffer_free_index = dmuffer_max_index;

    dmBufferFree(dmBuffers[dmuffer_free_index]);
}

/* write the QuickTime movie offset data */
if (mvFormat == MV_FORMAT_QT) {
    last_index = first_index + iov_index;
    if (write_qt_offset_data() == DM_FAILURE)
        return(DM_FAILURE);
}

/* reset the I/O vector index */
iov_index = 0;
}
vlXferCount++;
}
else {
    fprintf(stderr, "cannot get a valid DM buffer: %s\n",
            vlStrError(vlErrno));
}
break;

```

The example for reading data from disk can be found in */usr/share/src/dmedia/tools*.

Multiprocessing

Some aspects of digital media programming lend themselves to a multiprocessing programming model. On a multiprocessor system, the various tasks of moving multiple streams of video and audio data on and off disk, serial I/O control of external video equipment and input devices, processing of video data, or the transport of video data in and out of the graphics framebuffer can be assigned to different processors. New processes must be created with all virtual space attributes (shared memory, mapped files, data space) shared. The following fragment illustrates how to create a process to perform video recording.

```
if ((video_recorder_pid = sproc(video_recorder, PR_SADDR|PR_SFDS)) < 0) {
    perror("video_recorder");
    exit(DM_FAILURE);
}
```

If you use multiprocessing, note the following caveats:

- When VL calls are made, VL objects such as VLServer, VLPath, VLNode, and so on, are passed through the kernel to the video driver. However, you cannot create any VL objects without first creating a VLServer, from which everything else is instanced.
- In a process share group, only one VL call whose arguments derive from a VLServer can execute at a time. This requirement applies even to VL calls that do not explicitly take a VLServer as an argument (for example, vlBufferAdvise(3dm)).
- You can use objects derived from a given VLServer in any number of threads as long as you use a locking scheme, such as usnewsema(3P) or pthread_mutex_init(3P), to make the use in each thread mutually exclusive of a use in any of the other threads.

The VL error state, returned by vlGetErrno(3dm), is global to a share group, not per VLServer. If a VL call using one VLServer in one thread executes simultaneously with a VL call using another VLServer in another thread, both calls try to set the error state returned by **vlGetErrno()**. This call should be global only to the thread, not to the entire process share group.

Asynchronous I/O

Asynchronous I/O allows an application to process multiple read or write requests simultaneously. On Silicon Graphics platforms, asynchronous I/O is available through the *aio* facility. This facility, based on *sproc(2)* ed processes, provides all of the benefits of multiprocessing for free. Because multiple I/O requests might be outstanding, when you use asynchronous I/O, the round-trip delay between making a request, having it serviced, and issuing another request is removed. Any process-scheduling delay between these steps is also eliminated.

Because asynchronous I/O operations complete out of sequence, the application must keep track of the order in which data appears in the DMbuffers. DMbuffers are contained in a DMbufferPool; the pool itself is unordered and buffers can be obtained and returned to the pool in any order. Ordering is achieved by a first-in-first-out queue and maintained only while the buffers reside in the queue. The application is free to impose any processing order once buffers are dequeued.

File Formats

Each time a DMbuffer is written to disk, an offset must be recorded for the QuickTime file.

```
MVid theMovie;
MVid mvImageTrack;
off64_t mvFieldGap= bytesPerXfer - vlBytesPerImage;
MVtimescale mvImageTimeScale=MV_IMAGE_TIME_SCALE_NTSC;
int mvFrameTime = 1001; /* for NTSC */
off64_t meta_data_offset;
int mv_frame_index;
MVframe mv_dummy_offset;
int i;

mvInsertTrackDataAtOffset (
    mvImageTrack,
    1,
    (MVtime) (i * mvFrameTime),
    (MVtime) mvFrameTime,
    mvImageTimeScale,
    (off64_t) meta_data_offset,
    vlBytesPerImage,
    MV_FRAMETYPE_KEY,
    0)
```

```

/* get the index for the libmovie data corresponding to this field.
 * this is necessary in order to set the gap and field sizes for the
 * fields in the frame.*/
mvGetTrackDataIndexAtTime(
    mvImageTrack,
    (MVtime) (i * mvFrameTime),
    mvImageTimeScale,
    &mv_frame_index,
    &mv_dummy_offset)

/* tell libmovie the field gap and sizes for each field in the frame */
mvSetTrackDataFieldInfo(
    mvImageTrack,
    mv_frame_index,
    vlBytesPerImage,          /* absolute size of field 1 */
    mvFieldGap,              /* gap between fields */
    vlBytesPerImage)        /* absolute size of field 2 */

```

When data recording completes, the following function must be called to properly close the QuickTime file.

```

write_qt_file_header(void)
{
    int flags;

    /* if direct I/O mode is enabled, disable it because the
     * movie library does not do direct I/O
     */
    if (ioFileFD) {
        fsync(ioFileFD);
        flags = fcntl(ioFileFD, F_GETFL);
        flags &= ~FDIRECT;
        if (fcntl(ioFileFD, F_SETFL, flags) < 0) {
            fprintf(stderr, "unable to reset direct I/O file status\n");
            return(DM_FAILURE);
        }
    }

    if (mvClose(theMovie) == DM_FAILURE) {
        fprintf(stderr, "unable to write movie file header %s\n",
            mvGetErrorStr(mvGetErrno()));
        return(DM_FAILURE);
    }
}

```

Diagnostics

The *divo_confidence* diagnostic test shell script verifies proper DIVO board operation. This script calls the diagnostic software for DIVO field service hardware diagnosis and board fault isolation. When the DIVO board and its software are installed, this shell script is run.

The script *divo_confidence* calls board-level tests and VL-based tests. It requires IRIX 6.4 or later.

The test suite is explained in these sections:

- “divo_confidence Functionality” on page 111
- “Running divo_confidence” on page 112
- “divo_confidence Output” on page 114

divo_confidence Functionality

A *divo_confidence* loop takes about five minutes per DIVO board. On the first DIVO board it finds, the script performs the following actions.

1. It runs a set of board-level tests on the first DIVO board it finds, printing the results at the end.
2. It runs VL-based tests on the same board, printing results at the end.
3. It repeats each set of tests.
4. It writes this output to a log file in */usr/tmp/DIVO/logs* that reflects the number of the DIVO board; for example:
 - */usr/tmp/DIVO/logs/divo_confidence.log0* contains test output for DIVO board 0 (the first board)
 - */usr/tmp/DIVO/logs/divo_confidence.log1* contains test output for DIVO board 1

5. It runs the same tests on all other DIVO boards it finds, in numerical order, with repeats as above, and writes them to a log file in `/usr/tmp/DIVO/logs` that reflects the number of the DIVO board.
6. It creates a summary of all test results for all boards and prints them to the screen as well as to a separate log file, `/usr/tmp/DIVO/logs/elog.<n>`.

Figure F-1 diagrams the contents of the `/usr/diags/DIVO` directory.

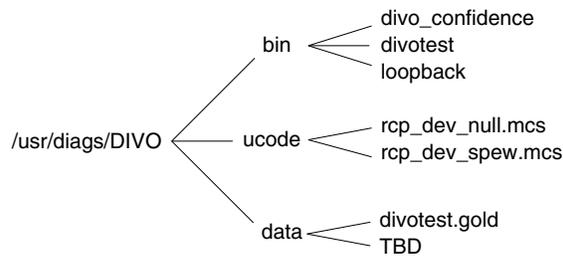


Figure F-1 /usr/diags/DIVO Contents

For detailed information on the board tests, see the `divotest(1M)` reference (man) page (`/usr/share/catman/a_man/cat7/divotest.z`), or use

```
/usr/diags/DIVO/bin/divotest -help
```

or

```
/usr/diags/DIVO/bin/loopback -help
```

Running `divo_confidence`

You must have root privilege to run `divo_confidence` from the local or remote console. One iteration of `divo_confidence` takes less than five minutes per DIVO board.

Follow these steps:

1. In `/usr/diags/DIVO/bin`, enter
`divo_confidence`
2. In the test summary, check the number of boards seen by the diagnostic test suite versus the number installed.

An example output for a system with one DIVO board follows.

```
=====
=====  DIVO TEST SUMMARY  =====
=====
DIVO board 0 serial #: CEK544
Test log for board 0: /usr/tmp/DIVO/logs/divo_confidence.log0
Started on:          Tue Apr 22 10:04:11 PDT 1997
Ended on:            Tue Apr 22 10:09:00 PDT 1997
Total loops run:    2
Overall test results: board 0 in slot io6 in module 1
passed!
```

```
=====
=====  DIVO TEST SUMMARY  =====
=====
```

The example output above notifies you that the DIVO board passed all tests. If DIVO board 0 had failed, output might be as follows.

```
=====
=====  DIVO TEST SUMMARY  =====
=====
DIVO board 0 serial #: CEK544
Logfile for board 0:  /usr/tmp/DIVO/logs/divo_confidence.log0
Started on:          Mon Apr 21 22:34:28 PDT 1997
Ended on:            Mon Apr 21 22:39:17 PDT 1997
Total loops run:    2
Overall test results: board 0 in slot io6 in module 1
failed!
-> Please replace DIVO board 0 in slot io6 in module 1.
-> View /usr/tmp/DIVO/logs/elog.0 for details of test failures.
```

```
=====
=====  DIVO TEST SUMMARY  =====
=====
```

To see ERROR and FAIL messages for each failed DIVO board in the system, look in the error log file for that board, */usr/tmp/DIVO/logs/elog.<n>*.

To see the complete *divo_confidence* output for an individual board, view its log file, */usr/tmp/DIVO/logs/divo_confidence.log<n>* (for example, */usr/tmp/DIVO/logs/divo_confidence.log0*). See Section , “*divo_confidence* Output,” for more information.

The *divotest* suite reports failures to the field-replaceable unit (FRU) level; for DIVO, this level is the board itself. If the board fails any test, contact your service provider to arrange replacement of the board.

divo_confidence Output

As mentioned previously, when *divo_confidence* runs the *divotest* suite, it prints results at the end of each test iteration. The output format is line-based and never more than 80 characters long. Table F-1 summarizes the four-character identifiers that head each line of output.

Table F-1 divotest Output Identifiers

Identifier	Meaning
TEST	Test start marker, generated at the beginning of a test; gives test's symbolic name and description. All lines up to next TEST line belong to this test.
RSLT	Test result, generated at the end of a test; gives test's symbolic name and the test result (PASS, FAIL, UNRESOLVED, UNTESTED).
DIAG	Diagnostic message: one or more of these lines precedes any FAIL or UNRESOLVED message. Message indicates components or wires that are possible causes of the failure.
INFO	Information useful to advanced user of the diagnostic test including test progress reports, exp/rcv pairs, and so on.
DEBUG	Gives information only when debugging output is turned on (not recommended for field use).
TIME	Time stamp, generated at important time boundaries such as the beginning and end of divotest.
META	Summarizes information for several tests or across multiple full test loops in table format, giving PASS and FAIL counts of each test and totals over all tests.
ABRT	Reports an exceptional error condition leading to the abortion of the diagnostic tests; probably caused by a malloc failure, unexpected system call failure, or assertion failure; rare.
FILES	<i>/usr/diags/DIVO/bin/divotest</i> contains executables for DIVO diagnostics. <i>/usr/diags/DIVO/ucode</i> is a directory of DIVO diagnostic microcode. <i>/usr/diags/DIVO/data</i> is a directory of data files for DIVO diagnostics. See Figure F-1.

The following is an example test output for *divo_confidence*—that is, both *divotest* and the VL-based tests—for a system with two DIVO boards correctly installed.

Note: For ease of understanding, iterations are indicated rather than reproduced in full.

```
===== DIVO_CONFIDENCE ===== DIVO_CONFIDENCE ===== DIVO_CONFIDENCE =====  
===== DIVO_CONFIDENCE ===== DIVO_CONFIDENCE ===== DIVO_CONFIDENCE =====
```

```
Starting divo_confidence script .....  
Found 2 DIVO(s) installed .....
```

```
divo_confidence script takes about 5 minutes per board to run .....  
Looping 2 time(s) .....
```

```
Test log for board 0 is /usr/tmp/DIVO/logs/divo_confidence.log0
```

```
Uname:                IRIX64 testsystem28-3 6.4 02121744 IP27  
Divotest is:          /usr/diags/DIVO/bin/divotest  
VL tests path:        /usr/dmedia/bin/DIVO
```

```
====> Tue Apr 22 10:04:12 PDT 1997  
====> Running divotest (ex_loop 1) on board 0 slot io6 module 1  
TIME      0.005   Tue Apr 22 10:04:12 1997  
CMDL      /usr/diags/DIVO/bin/divotest MODNUM=1 DEVNUM=0 REPEAT=10  
CMDL      -continue -fe0 -fel TRACE=/usr/tmp/DIVO/logs/divotest.0  
CMDL      -notime -noinfo -notrace  
TEST init0      Initialize INPIPE front-end hardware  
RSLT init0      PASS  
TEST init1      Initialize OUTPIPE front-end hardware  
RSLT init1      PASS  
TEST refresh0   Refresh INPIPE SDRAM  
RSLT refresh0   PASS  
TEST refresh1   Refresh OUTPIPE SDRAM  
RSLT refresh1   PASS  
TEST bridge     BRIDGE sanity test  
RSLT bridge     PASS  
TEST linc0sanity INPIPE LINC sanity test  
RSLT linc0sanity PASS  
TEST linc1sanity OUTPIPE LINC sanity test  
RSLT linc1sanity PASS  
TEST flash0sanity INPIPE FLASHROM sanity test  
RSLT flash0sanity PASS  
TEST flash1sanity OUTPIPE FLASHROM sanity test  
RSLT flash1sanity PASS  
TEST sdram0     INPIPE SDRAM stress test  
RSLT sdram0     PASS  
TEST sdram1     OUTPIPE SDRAM stress test  
RSLT sdram1     PASS  
TEST linc0mbox  INPIPE LINC mailbox test  
RSLT linc0mbox  PASS
```

```

TEST linc1mbox      OUTPIPE LINC mailbox test
RSLT linc1mbox      PASS
TEST linc0dma       INPIPE LINC DMA engine0/engine1 stress test
RSLT linc0dma       PASS
TEST linc1dma       OUTPIPE LINC DMA engine0/engine1 stress test
RSLT linc1dma       PASS

```

```

<<<you might see other tests as well as these>>>
<<<test series repeats>>>
<<<results of divotest appear>>>

```

META	ITERATION=10	PASSES	NON-PASSES
META	init0	10	0
META	init1	10	0
META	refresh0	10	0
META	refresh1	10	0
META	bridge	10	0
META	linc0sanity	10	0
META	linc1sanity	10	0
META	flash0sanity	10	0
META	flash1sanity	10	0
META	sdram0	10	0
META	sdram1	10	0
META	linc0mbox	10	0
META	linc1mbox	10	0
META	linc0dma	10	0
META	linc1dma	10	0
META	TOTAL	150	0

```

<<<VL-based tests run next>>>
<<<results of VL-based tests appear>>>

```

```

<<<both test series repeat>>>
<<<complete suite is run on other board installed>>>
<<<then results of all tests are summarized>>>

```

```
=====
===== DIVO TEST SUMMARY =====
```

```
DIVO board 0 serial #: CEK544
Test log for board 0: /usr/tmp/DIVO/logs/divo_confidence.log0
Started on:          Tue Apr 22 10:04:11 PDT 1997
Ended on:            Tue Apr 22 10:09:00 PDT 1997
Total loops run:    2
Overall test result: board 0 in slot io6 in module 1 passed!
```

```
DIVO board 1 serial #: CEK545
Test log for board 1: /usr/tmp/DIVO/logs/divo_confidence.log1
Started on:          Tue Apr 22 10:10:11 PDT 1997
Ended on:            Tue Apr 22 10:15:00 PDT 1997
Total loops run:    2
Overall test result: board 1 in slot io7 in module 1 passed!
```

```
===== DIVO TEST SUMMARY =====
=====
```

Index

Numbers

- 0 bit in packing, 49
- 0http
 - [//techpubs.sgi.com/library](http://techpubs.sgi.com/library), xv
 - [//www.sgi.com/Technology/toolbox.html](http://www.sgi.com/Technology/toolbox.html), 101
- 4:2:2
 - format, 6
 - sampling, 77
 - video, converting, 77
- 4:2:2:4
 - connector usage, 32
 - control for setting, 19
 - format, and Links A and B, 6, 41
 - sampling, 77
- 4:4:4
 - sampling, 76
 - video, converting, 77
- 4:4:4:4
 - connector usage, 32
 - control for setting, 19
 - format, and Links A and B, 6, 41
 - sampling, 76

A

- asynchronous I/O, 108
- audio, 7

B

- buffer, 12

C

- CCIR 601-2. *See* color space.
- CCIR color space, 83
- color space, 7, 78-79
 - compressed-range, 78-79
 - control, 17, 18
 - conversion, 81-100
 - math operations, 84
 - precision, 85
 - range, 85-88
 - converters, 7
 - full-range, 78-79
- compressed-range color space, 78-79
- compression, 16
 - control, 17, 18, 19, 20
 - Rice, 16
 - control, 18, 20
- control
 - determining for device, 16
 - device-dependent, 12
 - device-global, 12, 17-20
 - device-independent. *See* control, device-global.
 - DIVO-specific, 12
 - inline, 29
 - prefix, 12
 - setting, 16
 - values and uses, 18-20
- conventions, xvii
- current sense, 39-40

- D**
- decimation filter, 7
 - device, 12
 - controls, 17-20
 - determining, 16
 - diagnostics, 111-117
 - digital video drain, setting up, 43
 - digital video ports, 6-7
 - digital video source
 - setting up, 41-42
 - timing in panel, 42
 - direct I/O, 102-103
 - DIVO
 - audio, 7
 - board architecture, 3
 - connectors, 4
 - resistance, 4
 - controls for, 12, 16-20
 - digital video ports, 6-7
 - functional block diagram, 6
 - I/O panel, 41
 - LEDs, 4, 5
 - path, 14
 - ports, 6
 - setting up for hardware, 41-46
 - divo_confidence*, 111-117
 - DIVO I/O panel, 4
 - DMbuffer, 12
 - drain node. *See* node, drain.
 - dual-link mode, 6, 41-42
- E**
- events, 27-28
 - external sync source, 4, 5
- F**
- field dominance, 21-23, 43
 - control, 17, 19
 - file formats, 108-109
 - filter
 - decimation, 7
 - interpolation, 7
 - format control, 17, 19
 - full-range color space, 78-79
- G**
- general-purpose interface. *See* GPI.
 - GEN IN, 4, 5, 32
 - genlock, 32
 - genlock interface, 4, 5
 - GEN OUT, 4, 5, 32
 - GPI
 - control, 17, 19
 - hardware, 33-40
 - interface, 5
 - pinouts, 33-36
 - programming, 24-26
 - receiver, 38-40
 - interfacing, 40
 - transmitter, 36-38
 - interfacing, 38
 - graphics to video, 2
 - GVO graphics option, 2
- H**
- headroom-range color space. *See* compressed-range color space.
 - http, xv, 101

I

interpolation filter, 7

I/O

- asynchronous, 108
- direct, 102-103
- scatter/gather, 103-106

I/O panel, 4

K

kind, 13

L

LEDs, 4, 5

LINK A, 6, 41

- interface, 4
- transfer mode usage, 32

LINK B, 6, 41

- interface, 4
- transfer mode usage, 32

linking, 10

loopback control, 17, 19

loopthrough for genlock input, 4, 32

M

MSC (media stream count), 2, 29

multiprocessing, 107

N

node, 11, 12, 13-14

- drain, 11, 13
- source, 11, 13

O

offset control, 17, 20

OpenGL to read pixels into memory, 2

origin, different in OpenGL and video, 2

P

packing, 47-75

0 bit, 49

16-bit, 55-56

20-bit, 57

24-bit, 58-60

32-bit, 60-68

36-bit, 69

48-bit, 70-71

64-bit, 72-75

8-bit, 53-54

and sampling pattern, 48

control, 18, 20

native to DIVO, 50

x bit, 49

panel, 42-46

callup, 42

Digital Video Drain, 43

Digital Video Source, 42

external sync source, 45

restoring settings, 46

saving settings, 46

path, 11, 12, 14

R

raster mode control, 17, 19

return loss for IN connectors, 31

RGB, 78, 82

See also color space.

Rice compression, 16

control, 18, 20

RP-175 compressed RGB, 84

- S**
- sampling pattern, 76-77
 - and packing, 48
 - scatter/gather I/O, 103-106
 - size control, 18, 20
 - source node. *See* node, source.
 - specifications, 31-32
 - switch closure, 39-40
 - sync
 - connectors, 4, 32
 - control, 18, 20
 - setting up, 44-45
 - source control, 18, 20
- T**
- timing control, 18, 20
 - triggering, 26
 - control, 18, 20
 - type*, 13
- U**
- UST (unadjusted system time), 2
- V**
- vcp*, 41-46
 - callup, 42
 - See also* panel.
 - vertical inline time clock. *See* VITC.
 - Video Library. *See* VL.
 - VITC, 27
- VL**
- central concepts, 11
 - data transfer functions summarized, 15
 - header files, 10
 - object classes, 12
 - path, 11
 - requirements for running, 10
 - VL_ANY, 13
 - VL_ASPECT, 17, 18
 - VL_CAP_TYPE, 17, 18
 - VL_COLORSPACE, 17, 18
 - VL_COMPRESSION, 17, 19
 - VL_DIVO_LOOPBACK, 17, 19
 - VL_DIVO_RASTER_MODE, 17, 19
 - VL_FIELD_DOMINANCE, 17, 19, 21-23
 - VL_FORMAT, 17, 19
 - VL_GPI_OUT_MODE, 17, 19, 24-25
 - VL_GPI_STATE, 17, 19, 25
 - VL_MEM, 13
 - VL_OFFSET, 17, 20
 - VL_PACKING, 18, 20
 - VL_PACKING_0444_8, 64
 - VL_PACKING_242_10, 57
 - VL_PACKING_242_10_in_16_L, 67
 - VL_PACKING_242_10_in_16_R, 67
 - VL_PACKING_242_8, 55
 - VL_PACKING_2424_10_10_10_2Z, 66
 - VL_PACKING_4_8, 53
 - VL_PACKING_444_10_in_16_L, 71
 - VL_PACKING_444_12, 69
 - VL_PACKING_444_332, 54
 - VL_PACKING_444_5_6_5, 56
 - VL_PACKING_444_8, 49, 58
 - VL_PACKING_4444_10_10_10_2, 65

VL_PACKING_4444_10_in_16_L, 72
VL_PACKING_4444_10_in_16_R, 73
VL_PACKING_4444_12, 70
VL_PACKING_4444_12_in_16_L, 73
VL_PACKING_4444_12_in_16_R, 74
VL_PACKING_4444_13_in_16_L, 74
VL_PACKING_4444_13_in_16_R, 75
VL_PACKING_4444_6, 60
VL_PACKING_4444_8, 48, 61
VL_PACKING_R0444_8, 63
VL_PACKING_R242_10, 57
VL_PACKING_R242_10_in_16_L, 68
VL_PACKING_R242_10_in_16_R, 68
VL_PACKING_R242_8, 55
VL_PACKING_R2424_10_10_10_2Z, 66
VL_PACKING_R444_332, 54
VL_PACKING_R444_8, 59
VL_PACKING_R4444_8, 62
VL_PACKING_X4444_5551, 56
VL_RICE_COMP_PRECISION, 18, 20
VL_RICE_COMP_SAMPLING, 18, 20
VL_SIZE, 18, 20
VL_SYNC, 18, 20
VL_SYNC_SOURCE, 18, 20
VL_TIMING, 18, 20
VL_TRANSFER_TRIGGER, 18, 20, 26
VL_VIDEO, 13
VL_ZOOM, 18, 20
V-LAN, 2
vlGetNode(), 13
vlinfo, 16
vlOpenVideo(), 11, 13
vlSetControl(), 14, 16

X

x bit in packing, 49

Y

YUV, 78, 83

See also color space.

Z

zoom factor control, 18, 20

Tell Us About This Manual

As a user of Silicon Graphics products, you can help us to better understand your needs and to improve the quality of our documentation.

Any information that you provide will be useful. Here is a list of suggested topics:

- General impression of the document
- Omission of material that you expected to find
- Technical errors
- Relevance of the material to the job you had to do
- Quality of the printing and binding

Please send the title and part number of the document with your comments. The part number for this document is 007-3524-002.

Thank you!

Three Ways to Reach Us

- To send your comments by **electronic mail**, use either of these addresses:
 - On the Internet: techpubs@sgi.com
 - For UUCP mail (through any backbone site): *[your_site]!sgi!techpubs*
- To **fax** your comments (or annotated copies of manual pages), use this fax number: 650-932-0801
- To send your comments by **traditional mail**, use this address:

Technical Publications
Silicon Graphics, Inc.
2011 North Shoreline Boulevard, M/S 535
Mountain View, California 94043-1389

