HP 1000 A-Series Computer Systems

# HP 12065A
# Color Video Output Interface

## Reference Manual


This Manual Contains
Installation, Service, & Programming
Information

HEWLETT PACKARD

HEWLETT-PACKARD COMPANY
Roseville Networks Division
8000 Foothills Boulevard
Roseville, California 95678

MANUAL PART NO. 12065-90001
E0984
Printed in U. S. A.
September 1984

# PRINTING HISTORY

The Printing History below identifies the Edition of this Manual and any Updates that are included. Periodically, update packages are distributed which contain replacement pages to be merged into the manual, including an updated copy of this Printing History page. Also, the update may contain write-in instructions.

Each reprinting of this manual will incorporate all past updates; however, no new information will be added. Thus, the reprinted copy will be identical in content to prior printings of the same edition with the user-inserted update information. New editions of this manual will contain new information, as well as updates.

First Edition ..... September 1984

# REQUEST FOR DETAILED HARDWARE INFORMATION

**HP 1000 A-Series Computer Systems**

**HP 12065A COLOR VIDEO OUTPUT INTERFACE**
**Reference Manual**

**12065-90001 September 1984**

A more detailed description of the HP 12065A hardware is being considered for our OEM customers and others with hardware dependent applications. For a copy, return this form with your suggestions on information that should be included, and a brief description of your application. Attach additional sheets if necessary.

This form requires no postage stamp if mailed in the United States. For locations outside the U.S., your local HP representative will ensure that your comments are forwarded.

---

**FROM:** **Date** _____

Name _____

Company _____

Address _____

_____

_____

# HP Computer Museum
# www.hpmuseum.net

**For research and education purposes only.**

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

# BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 256 ROSEVILLE, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Publications Manager
Hewlett-Packard Company
Roseville Networks Division
8000 Foothills Boulevard
Roseville, CA 95678

FOLD FOLD

# SAFETY CONSIDERATIONS

**GENERAL** - This product and relation documentation must be reviewed for familiarization with safety markings and instructions before operation.

## SAFETY SYMBOLS

Instruction manual symbol: the product will be marked with this symbol when it is necessary for the user to refer to the instruction manual in order to protect the product against damage.

Indicates hazardous voltages.



Indicates earth (ground) terminal (sometimes used in manual to indicate circuit common connected to grounded chassis).

**WARNING**

**The WARNING sign denotes a hazard. It calls attention to a procedure, practice, or the like, which, if not correctly performed or adhered to, could result in injury. Do not proceed beyond a WARNING sign until the indicated *conditions* are fully understood and met.**

**CAUTION**

The CAUTION sign denotes a hazard. It calls attention to an operating procedure, practice, or the like, which, if not correctly performed or adhered to, could result in damage to or destruction of part or all of the product. Do not proceed beyond a CAUTION sign until the indicated conditions are fully understood and met.

**CAUTION**

## STATIC SENSITIVE DEVICES

When any two materials make contact, their surfaces are crushed on the atomic level and electrons pass back and forth between the objects. On separation, one surface comes away with excess electrons (negatively charged) while the other is electron deficient (positively charged). The level of charge that is developed depends upon the type of material. Insulators can easily build up static charges in excess of 20,000 volts. A person working at a bench or walking across a floor can build up a charge of many thousands of volts. The amount of static voltage developed depends on the rate of generation of the charge and the capacitance of the body holding the charge. If the discharge happens to go through a semiconductor device and the transient current pulse is not effectively diverted by protection circuitry, the resulting current flow through the device can raise the temperature of internal junctions to their melting points. MOS structures are also susceptible to dielectric damage due to high fields. *The resulting damage can range from complete destruction to latent degradation.* Small geometry semiconductor devices are especially susceptible to damage by static discharge.

The basic concept of static protection for electronic components is the prevention of static build-up where possible and the quick removal of already existing charges. The means by which these charges are removed depend on whether the charged object is a conductor or an insulator. If the charged object is a conductor such as a metal tray or a person's body, grounding it will dissipate the charge. However, if the item to be discharged is an insulator such as a plastic box/tray or a person's clothing, ionized air must be used.

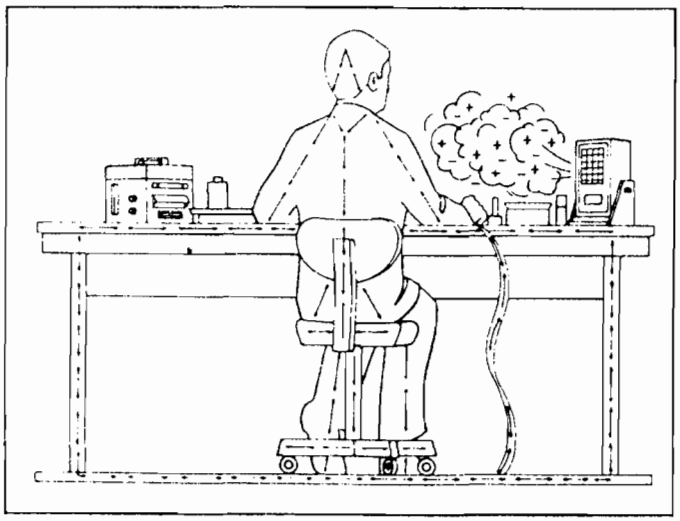
*Effective anti-static systems must offer start-to-finish protection for the products that are intended to be protected.* This means protection during initial production, in-plant transfer, packaging, shipment, unpacking and *ultimate use.* Methods and materials are in use today that provide this type of protection. The following procedures are recommended:

1. All semiconductor devices should be kept in "antistatic" plastic carriers. Made of transparent plastics coated with a special "antistatic" material which might wear off with excessive use, these inexpensive carriers are designed for short term service and should be discarded after a period of usage. *They should be checked periodically to see if they hold a static charge greater than 500 volts in which case they are rejected or recoated.* A 3M Model 703 static meter or equivalent can be used to measure static voltage, and if needed, carriers (and other non-conductive surfaces) can be recoated with "Staticide" (from Analytical Chemical Laboratory of Elk Grove Village, Ill.) to make them "antistatic."

2. Antistatic carriers holding finished devices are stored in transparent static shielding bags made by *3M Company.* Made of a special three-layer material (nickle/polyester/polyethylene) that is "antistatic" inside and highly conductive outside, they provide a Faraday cage-like shielding which protects devices inside. "Antistatic" carriers which contain semiconductor devices should be kept in these shielding bags during storage or in transit.

Individual devices should only be handled in a static safeguarded work station.

3. A typical static safeguarded work station is shown below including grounded conductive table top, wrist strap, and floor mat to discharge conductors as well as ionized air blowers to remove charge from nonconductors (clothes). Chairs should be metallic or made of conductive materials with a grounding strap or conductive rollers.



**SAFETY EARTH GROUND** - This is a safety class I product and is provided with a protective earthing terminal. An uninterruptible safety earth ground must be provided from the main power source to the product input wiring terminals, power cord, or supplied power cord set. Whenever it is likely that the protection has been impaired, the product must be made inoperative and be secured against any unintended operation.

**BEFORE APPLYING POWER** - Verify that the product is configured to match the available main power source per the input power configuration instructions provided in this manual.

If this product is to be energized via an auto-transformer (for voltage reduction) make sure the common terminal is connected to the earth terminal of the main power source.

## SERVICING

**WARNING**

**Any servicing, adjustment, maintenance, or repair of this product must be performed only by qualified personnel.**

**Adjustments described in this manual may be performed with power supplied to the product while protective covers are removed. Energy available at many points may, if contacted, result in personal injury.**

**Capacitors inside this product may still be charged even when disconnected from its power source.**

**To avoid a fire hazard, only fuses with the required current rating and of the specified type (normal blow, time delay, etc.) are to be used for replacement.**

**WARNING**

**EYE HAZARD**

**Eye protection must be worn when removing or inserting integrated circuits held in place with retaining clips.**

# PREFACE

**Purpose:** This manual contains step-by-step instructions for the installation of the HP 12065A Video Output Interface card into an HP 1000 A-Series computer. In addition, this manual equips the application programmer with the ability to properly configure the I/O driver, perform various card specific operations, and display graphic images on a video monitor.

This manual assumes a working knowledge of the HP 1000 A-Series computer and associated RTE-A operating system calls. Programming examples use FORTRAN. Knowledge of computer graphics and terminology would prove helpful.

The HP 12065A is supported with Graphics/1000-II. Device handlers associated with the interface have been written. Graphics/1000-II programmers will use this manual to build device-dependent requests and commands that are not directly accessible through Graphics/1000-II.

**Organization:** This manual is organized as follows:

Section 1: A general description of the HP 12065A is provided. Reference information and card specifications are included.

Section 2: Installation instructions to properly install the card into the computer and verify operation are provided.

Section 3: This section provides programming information. System calls needed to communicate with the card, along with card request and command formats recognized by card firmware, are described in detail.

Section 4: The Maintenance section discusses self-tests and other tests that can be performed to ensure proper operation of the video output card.

**Using this manual:** Read Section 1 for an overview of the HP 12065A. Section 2 must be read by persons installing the card. Programmers will find Section 3 necessary.

The appendices should be reviewed for supporting information. A complete listing of video monitor parameter configurations is provided in Appendix A. Appendix B describes the card's variance from the RS-343-A specification. Appendix C provides a simple programming example for sending card requests and graphics commands to the card, while Appendix E provides a quick reference summary of all card requests and commands. Appendix D illustrates card performance data.

**Comment:** Thank you for your selection of the video output interface and other technological products provided by Hewlett-Packard. As our valued customer, meeting your needs is among our highest goals. To this end, we encourage you to provide us with comments and/or suggestions using the reader comment sheet enclosed.

# CONTENTS

# CONTENTS (continued)

# CONTENTS (continued)

# GENERAL INFORMATION

SECTION 1

## DESCRIPTION

The HP 12065A is a medium performance interface designed for generating color displays on a video monitor. It is designed to operate in an HP 1000 A-Series computer. On-card ROM firmware and microprocessor control provide for local intelligence and a variety of display capabilities.

The card can be configured for 512 x 512 or 576 x 455 pixel resolution, corresponding to display aspect ratios of 1:1 or 4:3 respectively. Each display pixel is represented in the frame buffer by four bits, which point to sixteen colormap registers containing color intensity data. Sixteen colors, or eight colors with alphanumeric text overlay, can be displayed on a monitor at one time. Each color is represented in the registers by twelve bits: four bits each for the red, green, and blue intensity levels. This makes a total of 4096 colors, of which up to sixteen can be selected at one time. Independent blink control for each colormap register can be performed by dynamically changing register data.

The card has a standard character set in ROM, and can accept a downloaded character set into RAM. The character sets can be displayed graphically in various sizes, or as alphanumeric text overlay. Character color is programmed as discussed above. Scrolling, normally used with alphanumeric text output, is provided.

The card features several polygon fill styles to choose from, and provides user-definable rectangular fill patterns. In addition, a flash-fill capability is provided which takes advantage of the faster writing speeds to the frame buffer when the screen is blanked. Direct frame buffer reads and writes can be used to input and output entire screens of data instead of redrawing frequently used images.

Input and output to the two on-card RS-232-C ports are supported for 3-wire hardwired connections. Therefore, devices such as keyboards, touchscreens, and mouse units can be connected directly to the video output interface.

This manual provides the information necessary for installing the card in your system and for building the various requests and commands that implement the card's capabilities.

## PRODUCT STRUCTURE

The standard HP 12065A Video Output Interface includes:

| Part Number | Description |
|---|---|
| 12065-60001 | Video output card assembly |
| 12065-63001 | Video cable (3.0 meters, RG 179/U, BNC connectors) |
| 12065-90001 | Video Output Interface reference manual |
| 12065-90003 | Color Video Interface Device Handlers manual |

Option 001 to the HP 12065A adds a 3.0 meter cable (part number 12065-63002) for connection to the RS-232-C ports on the card. This cable terminates at one end in a single edge connector that attaches to the video interface card. The other end terminates in a female 25-pin RS-232-C connector for attachment to peripheral devices; the signals lines in this connector can be used to attach one or two peripheral devices to the card.

A diagnostic test hood (part number 12065-67001) is available for purchase separately from your HP Sales and Service Office.

## REFERENCE MANUALS

The following manuals are referenced in this manual:

| Manual Part Number | Description |
|---|---|
| 92841-90001 | Graphics/1000-II Device-independent Graphics Library Programmer's Reference Manual |
| 12065-90003 | HP 12065A Color Video Interface Device Handlers Manual |
| 92077-90007 | RTE-A Programmer's Reference Manual |
| 92077-90011 | RTE-A Driver Reference Manual |
| 92077-90019 | RTE-A Driver Designer's Manual |
| 92059-90001 | Macro/1000 Reference Manual |
| 92007-90001 | HP 1000 System Designer's Guide |

## SPECIFICATIONS

The following specifications apply to the Video Output Interface:

**Backplane:** HP 1000 A-Series

**Electrical Specifications:**

| DC Supply | Maximum Current |
|---|---|
| +12 volts | 0.062 amps |
| -12 volts | 0.018 amps |
| +5 volts | 3.760 amps |

| Maximum Power Used | |
|---|---|
| +12 volts | 0.7 watts |
| -12 volts | 0.2 watts |
| +5 volts | 19.8 watts |

**Environmental Specifications:**

Temperature:
Operating: 0 to 55 degrees Celsius
Non-operating: -40 to 75 degrees Celsius

Relative Humidity:
5% to 95%, non-condensing

Altitude:
Operating: to 4.6 km
Non-operating: to 15.3 km

**Radio Frequency Interference:**

Meets FCC class A
Meets VDE class A

**Physical Interfaces:**

Video output -- compatible with RS-343-A (refer to Appendix B for details of the variation of this interface from the RS-343-A specification)

Data communications -- dual RS-232-C channels

**Number of Monitors:**

5 maximum, daisy-chained

**Video Output Voltages:**

Data signals: maximum +0.714 volts ±0.2 volts
Sync signal: +0.286 volts ±0.08 volts

**Signal Format:**

Composite, that is, horizontal and vertical sync is superimposed on the green signal line

Non-interlaced

**Monitor Distances:**

RG 59/U: 75 meters maximum to the most distant monitor beyond the 12065-63001 cable provided

RG 11/U: 150 meters maximum to the most distant monitor beyond the 12065-63001 cable provided

**Monitor Resolution:**

576 x 455 pixels (4:3 aspect ratio)

512 x 512 pixels (1:1 aspect ratio)

**Memory Maps:**

4 planes, partitioned as:

3 planes producing 8 colors from a palette of 4096 colors, plus 1 overlay plane for alphanumeric characters

or

4 planes producing 16 colors from a palette of 4096 colors

**Polygon Area Fill:**

8 fill styles

**Write Modes:**

Flash fill: blank the screen; write vectors, pixels, and characters; display the screen

Update: single character or vector writes to the existing display

| | |
|---|---|
| **Hardware Vector Generator:** | Refer to Appendix D for vector performance graphs. |
| **Character Display:** | Variable size and orientation (90-degree increments) |
| | On-board ROM storage for standard character set |
| | On-board RAM storage for user-downloadable character sets |
| | On-board character field blinking |
| | Refer to Appendix D for characters-per-second performance graphs. |
| **Other Display Features:** | Scrolling |
| | On-board blink control of all memory planes |
| | Direct pixel memory reads and writes |
| **Data Communication Ports:** | two RS-232-C, three-wire connections |
| **Baud Rates:** | 300, 1200, 2400, 9600 (default) |

# INSTALLATION

SECTION 2

Take the following steps to install the Video Output Interface in your A-Series computer. These steps will be explained in detail in the remainder of this section.

1. Always follow proper anti-static procedures when handling the Video Output Interface card (or, for that matter, any other interface cards in your system). Anti-static procedures are described on pages iii and iv at the beginning of this manual.

2. Make sure that the power drawn by the video card is within the power budget of your computer. (The HP 1000 System Designer's Guide, part number 92007-90001, discusses power budgeting.)

3. Set the switches on the video card. Switch SW1 sets the select code of the card and the self test mode. Switch SW2 sets the display parameters for the card's video output signals.

4. Turn off the power to your computer and install the video card in the computer's backplane.

5. Connect the video output cable from the video card to your video monitor. Connect the RS-232-C cable to any peripheral devices you will use with the card. (Pin assignments for the RS-232-C cable are given later in this section.) Turn the computer's power back on.

6. Verify the proper operation of the video card by checking the results of self test. You may also want to adjust your video monitor at this time.

## POWER BUDGET

Power consumption specifications for the Video Output Interface card are listed in the specifications in Section 1 of this manual. Add up the power consumption specifications at each voltage for all cards that you will plug into the backplane of your computer, and make sure that the total power consumption does not exceed the amount that your computer can supply. The HP 1000 System Designer's Guide contains information that can help you in this task.

## SWITCH SETTINGS

There are two sets of switches on the video output card, SW1 and SW2. Their locations are shown in Figure 2-1.

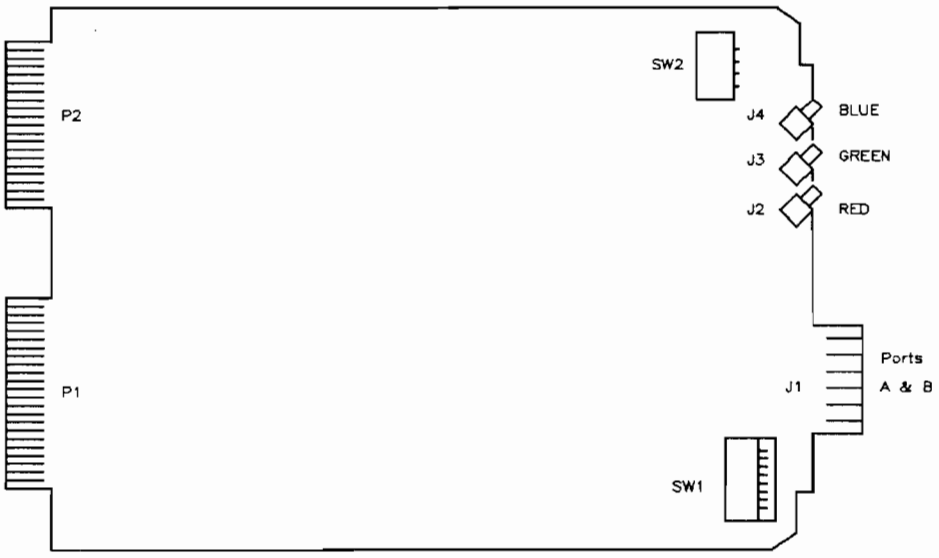



Figure 2-1: Location of Card Switches

SW1 contains eight positions, numbered 1 through 8; position 1 (SW1-1) is the most significant bit (bit 7). Positions SW1-3 through SW1-8 (bits 5 through 0) establish the video card's select code, which is used by the system to uniquely identify each card in an HP 1000 A-Series system. Set these switches to the select code that you have chosen for the video card.

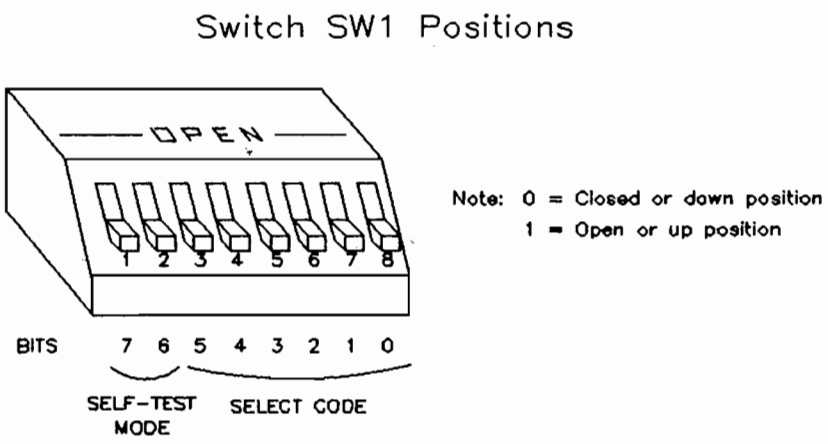



Figure 2-2: Switch SW1

Positions SW1-1 and SW1-2 (bits 7 and 6) select the video pattern that is displayed at the end of self test. These switches are read by the firmware when the card has finished testing its on-board electronics, and the appropriate test pattern is displayed. Self-test modes can be reconfigured in software (see the Write Card Configuration video card request). The mode definitions are provided below.

| Mode | Description |
|---|---|
| 0 0 | Displays the Graphics/1000-II 15 default colors in vertical bands and the default highlight mode. The first six colors are black, white, red, green, yellow, and blue. |
| 0 1 | Displays 15 vertical bands in increasing levels of gray. Since the gray scale represents a uniform ramp of intensity data, this mode is useful for testing the video signal without the monitor, such as with an oscilloscope. Alternatively, a monitor can be properly tested and adjusted for proper color tracking; there should be gradual steps in the gray scale without color hues in the vertical bands. |
| 1 0 | Displays a grid pattern for adjusting monitor convergence, and horizontal and vertical linearity. Convergence refers to the quality of the blue, green, and red electron beams converging to the same point. |
| 1 1 | Repeats self test continuously. This is useful for the diagnosis of intermittent hardware problems. Note that changing the switches during continuous self test will automatically reconfigure the self test mode to the new switch setting selected. |

The second switch, SW2, configures the monitor drive signal and the display format.

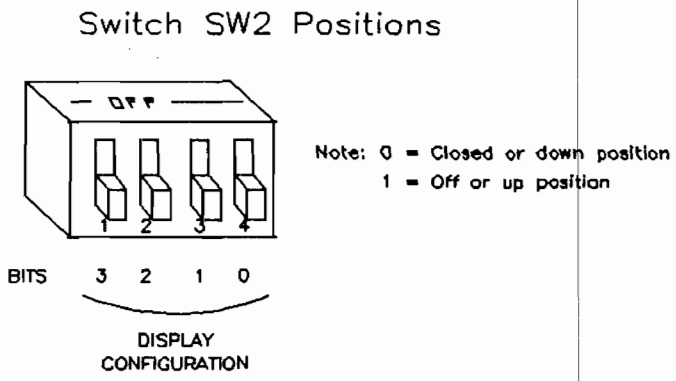



Figure 2-2: Switch SW2

There are sixteen possible configurations. They are shown with their respective parameter values in Appendix A. On power-up and self-test, the configuration indicated by the display switch positions is implemented. The configuration can be modified in software (see the Write Card Configuration video card request).

During power-up, various graphic display attributes (such as background color, color plane operation, text size, etc.) are assigned default values. See the DEFAULT_ALL_ATTRIBUTES and other commands for default values.

# CARD INSTALLATION

To install the card in the backplane:

1. Turn off the computer's power.

2. Using anti-static procedures (as given on pages iii and iv of this manual), plug the card into the backplane. Make sure that the component side of the card is facing up or to the right (depending on the orientation of the slots in your computer). Push the card into the backplane until it is firmly seated.

# CONNECTION TO PERIPHERALS

## Video Monitor

The video monitor cable (12065-63001) comprises three coaxial cables bound in a single jacket. The card end of the cable terminates in three brass connectors; the cables have red, black, and blue markings. Connect these to the red, green, and blue terminals on the card. (These are the brass terminals on the right half of the front edge of the card; they are, from left to right, the red, green, and blue signals.) Connect the braided metal shield cable to a ground point on the chassis of the computer.

The monitor end of the cable terminates in three silver-colored BNC connectors. Again, the cables have red, black, and blue markings. Attach the appropriate connectors to the red, green, and blue terminals of your monitor. If there is loose cable left over after you have connected the cable to the monitor, it should be neatly coiled up and tied; this will reduce unwanted radio frequency emissions.

If you are using only one monitor, the impedance for each input (each color) should be set to 75 ohms. If you are daisy-chaining several monitors together (up to the maximum of five monitors), all but the last should be set to high input impedances; the last monitor should be set to input impedances of 75 ohms.

Monitors that can be used with the video card must be able to accept composite video signals, that is, horizontal and vertical sync signals are superimposed on one of the color input signals. Monitors which require non-composite signals, or sync signals on a separate line, cannot be used.

Note that if you are using a monochrome monitor, you should connect the green signal to the monitor. The green signal carries the synchronizing information.

To power up the monitor, follow the monitor manufacturer's instructions.

## RS-232-C Peripherals

The RS-232-C cable (option #001 to the 12065A product) terminates in a 12-pin hooded edge connector on the card end of the cable. Plug the hooded connector onto the J1 edge of the card, with the cable exiting down or to the right (depending on the orientation of the cards in your computer). Connect the braided metal shield cable to a ground point on the chassis of the computer.

The RS-232-C cable has a female 25-pin RS-232-C connector for making three-wire connections to RS-232-C peripheral devices, such as keyboards, trackballs, and mice. (Note that these ports are not general purpose RS-232-C ports; they are for use with graphics input devices in conjunction with the capabilities of the video card, and as such can be used only with DGL calling sequences or with ID.50 driver-level calling sequences.) The connector contains the wiring for making two three-wire RS-232-C connections; this wiring is listed in the table below. If you want to connect a single three-wire RS-232-C peripheral that has a standard 25-pin male connector, just plug it into the female connector. If you want to connect two three-wire RS-232-C peripherals you can wire up your own male connector, or you can purchase a "Y" cable (part number 8120-3569, available through your HP Sales and Service Office) that breaks out the wiring into two standard three-wire female 25-pin RS-232-C connectors.

Pin assignments for the RS-232-C cable are as follows.

| Signal | Card Connector | Peripheral Connector |
|---|---|---|
| transmit A | pin E | pin 2 |
| receive A | pin 1 | pin 3 |
| ground | pin 2 | pin 7 |
| transmit B | pin C | pin 14 |
| receive B | pin 3 | pin 16 |
| shield | braided cable with grounding lug | pin 1 |



Note that "transmit" and "receive" are from the point of view of the RS-232-C device, not the point of view of the video card. For example, peripheral A sends signals to the card through pin 2 of its connector and pin E of the card connector.

When all cables have been connected, turn the computer's power back on.

## VERIFICATION

With the monitor powered up and properly connected, restoring power to the computer will start self-test operations. The red LED on the card should light or blink. After the LED goes out, the monitor should display the pattern as configured by SW1-1 and SW1-2.

You may want to verify that everything is working properly. You can test the operation of the card by again running the self-test. You can run this test from hardware (by cycling power on the computer or by pushing the reset button on the processor card) or from software (by issuing commands from a program). See the Maintenance section of this manual for complete details on running and interpreting the self-test.

At this time you may also want to adjust your monitor for optimum performance. Instructions for adjusting the HP 13279B monitor are given in the Maintenance section of this manual. These instructions should also be useful if you are using a different type of monitor.

# PROGRAMMING INFORMATION

SECTION 3

This section describes the driver configuration, input/output (I/O) requests, and card commands for operation of the HP 12065A. Recall that *EXEC 1* and *EXEC 2* calls are for Read and Write I/O requests, while *EXEC 3* calls are for I/O device control. For programming the HP 12065A, these calls are used extensively. RTE-A EXEC calls, and associated error codes, are fully described in the *RTE-A Programmer's Reference Manual.*

Although beyond the scope of this manual, RTE Assembly language calling sequences can be used to program the HP 12065A. General RTE Assembly language calls are described in the *Macro/1000 Reference Manual.*

The HP 12065A is compatible with the Graphics/1000-II product. For detailed programming information, refer to the *Graphics/1000-II Device-independent Graphics Library Programmer's Reference Manual* and the *HP 12065A Color Video Interface Device Handlers Manual.*

## PROGRAMMING OVERVIEW

The card's firmware routines interpret information received and control the card's activities. For proper firmware response, information sent to the card must be correctly formatted.

One component of this process is configuration of the I/O driver. The driver must be properly configured to interact with the firmware. This is accomplished by a simple I/O device control EXEC call.

Programming the card consists of two basic types of transactions:

**Video Card Requests** are requests of the card to perform a task or operation, such as initiating self-tests, configuring the card, or processing graphics commands.

**Graphics Commands** are used to output graphic images on a display monitor, such as lines and polygons in various colors. Graphics commands require the use of a specific video card request (see the Write Graphics Commands video card request) to be downloaded and processed by the card.

*Thus, to draw and display an image on a monitor, it is necessary to send the video card request for processing graphics commands along with the graphics commands that define the image.*

Video card requests and graphics commands are sent to the card in the data buffer of I/O write EXEC calls. There can be only one video card request per EXEC call, while there can be multiple graphics commands per EXEC call.

This section will discuss the specific details of each video card request and graphics command. However, to illustrate the concepts discussed above, refer to Appendix C for a very simple programming example.

Appendix D provides card output performance data. Appendix E provides a quick reference guide for all video card requests and graphics commands.

# DRIVER-FIRMWARE INTERACTION

Drivers are operating system modules used for implementing I/O requests made by user programs and formatted by the system. Driver operation is primarily directed from a system generated table, called the driver's DVT (Device Table). There is a DVT for each device in the system. In the form of 16-bit words, the driver's DVT maintains all information pertaining to the operation of a specific request. Drivers retrieve and post information in the DVT. *DVT1, DVT15, DVP1* and *DVP2* are examples of specific driver DVT words. The HP 12065A firmware was designed to operate with the general purpose parallel interface driver ID. 50 (REV. 2401 or later). For detailed information on drivers and tables, refer to the *RTE-A Driver Reference Manual* and the *RTE-A Driver Designer's Manual.*

## Driver Configuration

The driver must be configured to operate with the video output card firmware. The driver configuration is set by the values of two DVT parameter area words associated with the driver: DVP1 and DVP2. Both DVP1 and DVP2 can be set during system generation, or can be set in a user program by using an *ID. 50 Control Request* as follows (refer to the *RTE-A Driver Reference Manual* for a discussion on ID. 50 Control Requests):

**CALL EXEC (3,4000B + LU,100000B,36B)**

| | |
|---|---|
| *3* | EXEC code for I/O Device Control |
| *4000B + LU* | Specifies:<br>1) the next two parameters in the command string are driver configuration parameters and will be placed in DVP1 and DVP2 respectively, and<br>2) the logical unit number of the selected device |
| *100000B* | The parameter stored in DVP1 |
| *36B* | The value stored in DVP2:<br><br>bit 0 = 0: Selects 16 bit parallel data transfers<br><br>bit 1 & bit 2 = 1: Clears DMA AUTO bit on input and output I/O requests<br><br>bit 3 = 1: DVP1 is OR'd with the Request Code placed in DVT15 by each I/O request, forming the card control word of the I/O request (i.e., the card control word indicates whether the request is a read or write)<br><br>bit 4 = 1: Card control register (R31) is not set with the DVP1 value when the driver request completes, and card interrupts are reset only after asynchronous interrupts are serviced by the driver<br><br>bits 5 through 15: Undefined and should be set to 0 |

# VIDEO CARD REQUESTS

Video card requests are coded information through which specific tasks or operations are performed, such as initiating self-tests, performing graphics commands, or reading card status. Video card requests are written to the card as the first word in the data buffer of an I/O EXEC write request. There can be only one video card request per EXEC call.

For simplicity, we have divided video card requests into two general categories: Write requests and Read requests.

**VIDEO CARD REQUESTS**

| WRITE REQUESTS | READ REQUESTS |
|---|---|
| Self-Test<br>Write Card Configuration<br>Write RS-232 Port<br>Flush RS-232 FIFO Buffer<br>Enable Card Interrupts<br>Write Soft Character Set<br>Write Raw GDC Chip Commands<br>Write Frame Buffer<br>Write Graphics Commands | Read Card Status<br>Read RS-232 Port<br>Read Raw GDC Chip Data<br>Read Frame Buffer<br>Read Graphics Status Buffer |

## Video Card Write Requests

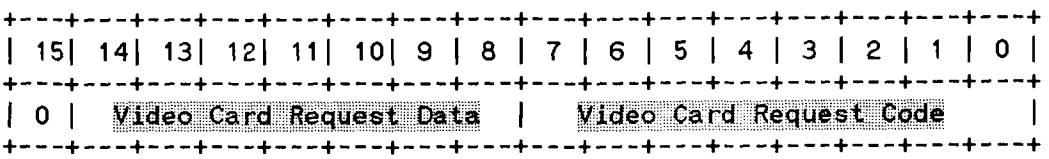
The following I/O EXEC call is used to send the video card request to the card.

**EXEC (2,LU,BUFR,BUFLN)**

| | |
|---|---|
| *2* | EXEC code for I/O Write request |
| *LU* | Device selection by the logical unit number |
| *BUFR* | The buffer array containing the data to be written. The first word in the buffer must be the Video Card Request |
| *BUFLN* | Buffer length (+ words) |

The card firmware reads the first word of the I/O request's data buffer (BUFR) for the video card request.

A video card request is divided into the following fields:

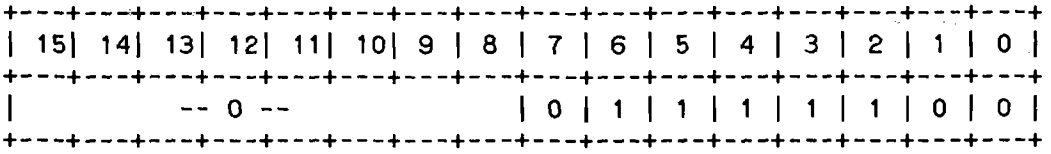



Where:

Video Card Request Data - Uniquely defined for each Video Card Request Code
Video Card Request Code - Defines type of operation to be performed by the card

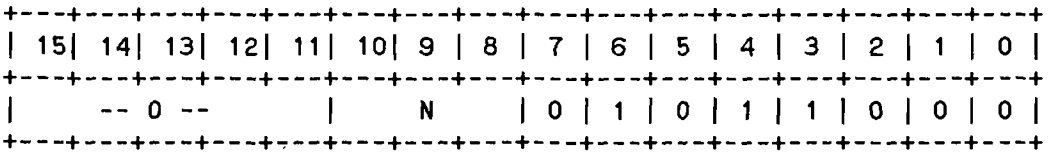
The bit patterns associated with each video card write request are described in the following paragraphs.

**SELF-TEST.** To initiate a video card self-test, the following code is placed in the data buffer. Since only a one-word buffer is required , BUFLN = 1.




For more information on self-tests, see the paragraph Self-Tests and Card Status Requests.

**WRITE CARD CONFIGURATION.** You can configure display parameters, port parameters, or data transfer parameters using the Write Card Configuration request. In this case, a buffer with buffer length of two is needed. The first word contains a video card request code *and* video card request data:




Where:

N - A number which identifies one of the configuration words described below

As specified by the value of N, the firmware configures the card using the data contained in the second word of the data buffer. Data associated with each value of N is described on the following pages.

**N = 0: Set Display Parameters**

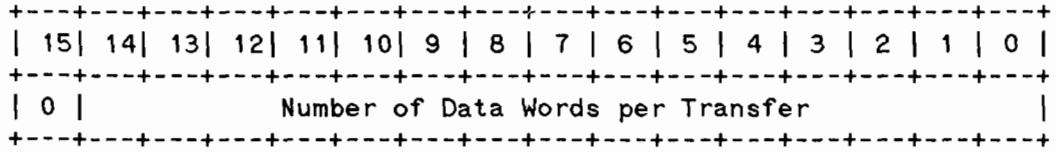
```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 15| 14| 13| 12| 11| 10| 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                 -- 0 --                       |    display    |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Where:

`display` - Signifies the configuration display code. Some of the parameters are shown below. See Appendix A for additional parameters.

| Display code | Resolution (h: horizontal) (v: vertical) | Horizontal Sweep Frequency | Vertical Scan Frequency |
|---|---|---|---|
| *0000* | 576h x 455v | 29.4kHz | 60.0 Hz |
| *0001* | 512h x 512v | 31.5 kHz | 57.5 Hz |
| *0010* | 576h x 455v | 28.1 kHz | 57.4 Hz |
| *0011* | 512h x 512v | 30.8 kHz | 56.1 Hz |
| *0100* | 576h x 455v | 27.0 kHz | 55.1 Hz |
| *0101* | 512h x 512v | 29.4 kHz | 53.6 Hz |
| *0110* | 576h x 455v | 25.9 kHz | 52.9 Hz |
| *0111* | 512h x 512v | 28.1 kHz | 51.4 Hz |
| *1000* | 576h x 455v | 25.0 kHz | 50.9 Hz |
| *1001* | 512h x 512v | 27.0 kHz | 49.3 Hz |
| *1010* | 576h x 455v | 24.0 kHz | 49.1 Hz |
| *1011* | 512h x 512v | 25.9 kHz | 47.3 Hz |
| *1100* | 576h x 455v | 23.2 kHz | 47.4 Hz |
| *1101* | 512h x 512v | 25.0 kHz | 45.5 Hz |
| *1110* | 576h x 455v | 22.4 kHz | 45.7 Hz |
| *1111* | 512h x 512v | 24.0 kHz | 43.9 Hz |

**N = 1: Frame Buffer Blocking Factor**

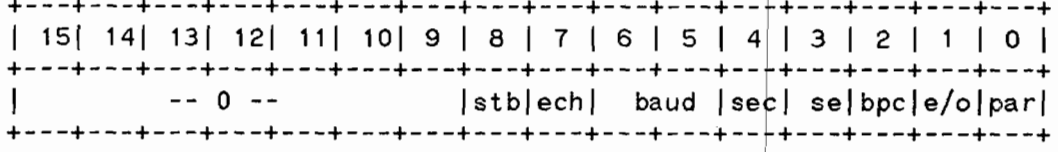



This word configures the number of data words per transfer when writing to, or reading from, the card frame buffer memory using the *Write Frame Buffer* and *Read Frame Buffer* video card requests. The number specified must match the number of actual data words to be transferred to and from the frame buffer during a single request. Stated in another way, the buffer length of the I/O EXEC call which writes to the frame buffer must be:

{Frame Buffer Blocking Factor + 1} words

since the video card request will be the first word of the buffer, and the actual data words will comprise the remainder of the buffer. The default and maximum frame buffer blocking factor is 16,384 words (one plane).

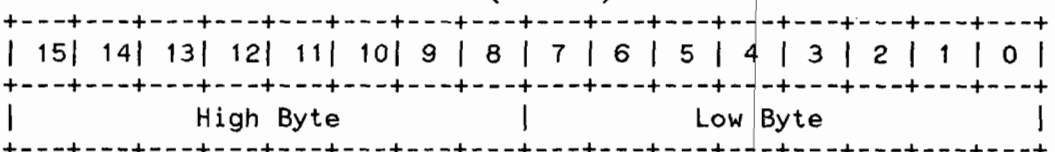
**N = 2: Port A Configuration**




Where:

| | | |
|---|---|---|
| *par* | parity | 0: no parity (default)<br>1: force parity |
| *e/o* | type of parity | 0: odd parity (default)<br>1: even parity |
| *bpc* | bits per character | 0: seven bits (default)<br>1: eight bits |
| *se* | video monitor screen echo enable | 0: disable screen echo (default)<br>1: enable screen echo |
| *sec* | video monitor screen echo cursor enable | 0: no echo (default)<br>1: enable screen echo cursor |
| *baud* | baud rate | 00: 300<br>01: 1200<br>10: 2400<br>11: 9600 (default) |
| *ech* | port echo enable | 0: no echo return to RS-232 port device (default)<br>1: echo return to RS-232 port device |
| *stb* | number of stop bits | 0: one stop bit (default)<br>1: two stop bits |

**N = 3: Definition of a Record (Port A)**

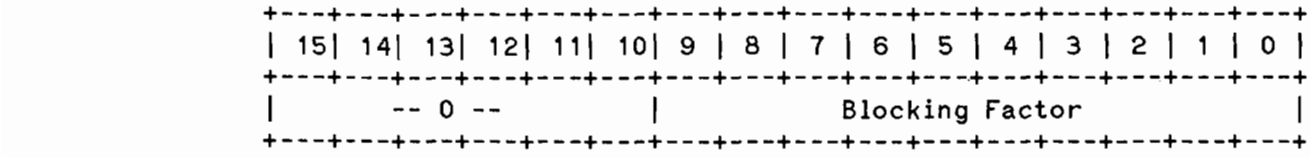



Where:

Low Byte - The number of bytes (maximum of 132) counted by the firmware to determine the record length. If the low byte is zero, record termination by count is disabled. The default condition is zero (disabled).

High Byte - The ASCII character representation, when matched by the firmware, terminates the record. If the high byte is set to zero, record termination by match is disabled. The default value is the *carriage return* character.

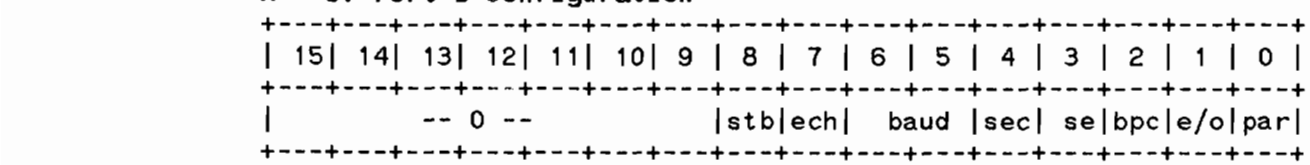
**N = 4: Port A Blocking Factor**




Where:

`Blocking Factor` - The maximum number of data words to be returned by a READ on the RS-232 port. The maximum value allowed is 132 words. The default value is 66 words.

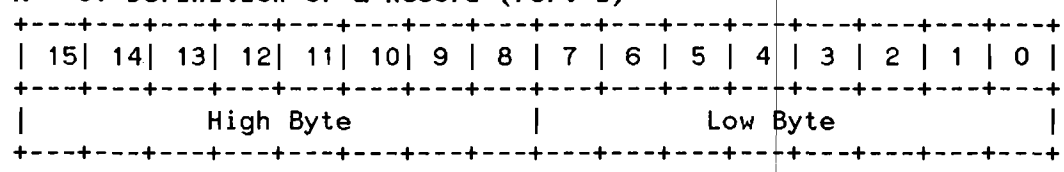
**N = 5: Port B Configuration**




Where:

| | | |
|---|---|---|
| *par* | parity | 0: no parity (default)<br>1: force parity |
| *e/o* | type of parity | 0: odd parity (default)<br>1: even parity |
| *bpc* | bits per character | 0: seven bits (default)<br>1: eight bits |
| *se* | video monitor screen echo enable | 0: disable screen echo (default)<br>1: enable screen echo |
| *sec* | video monitor screen echo cursor enable | 0: no echo (default)<br>1: enable screen echo cursor |
| *baud* | baud rate | 00: 300<br>01: 1200<br>10: 2400<br>11: 9600 (default) |
| *ech* | port echo enable | 0: no echo return to RS-232 port device (default)<br>1: echo return to RS-232 port device |
| *stb* | number of stop bits | 0: one stop bit (default)<br>1: two stop bits |

**N = 6: Definition of a Record (Port B)**

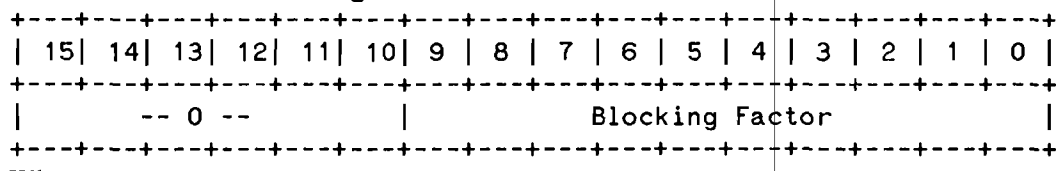



Where:

Low Byte - The number of bytes (maximum of 132) counted by the firmware to determine the record length. If the low byte is zero, record termination by count is disabled. The default condition is zero (disabled).

High Byte - The ASCII character representation, when matched by the firmware, terminates the record. If the high byte is set to zero, record termination by match is disabled. The default value is the *carriage return* character.

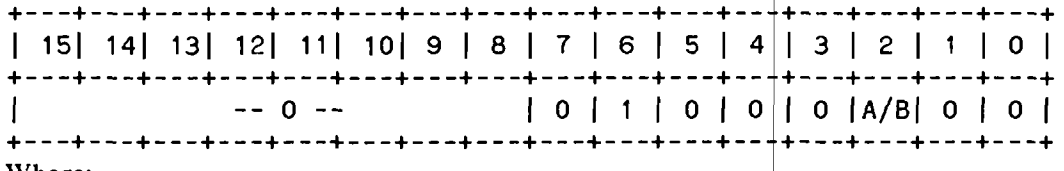
**N = 7: Port B Blocking Factor**




Where:

Blocking Factor - The maximum number of data words to be returned by a READ on the RS-232 port. The maximum value allowed is 132 words. The default value is 66 words.

**WRITE RS-232 PORT.** To write to one of the RS-232 ports, place the following code in the first word of the data buffer.




Where:

A/B - Port Selection
0: Write to port A
1: Write to port B

The remaining words of the data buffer are the RS-232 characters (e.g., ASCII characters). The characters are sent to the card's Dual Universal Asynchronous Receiver/Transmitter (DUART). Bit 2 in the video card request specifies whether to write to port A (primary channel) or port B (secondary channel). When outputting the data, the RS-232 configuration set by the *Write Card Configuration* request is in effect.

Since the RS-232 output uses a single linear buffer, any RS-232 output must be allowed to complete before another buffer is written to the same port; otherwise, some of the previous data may be lost. By reading the appropriate card status bit (refer to the *Read Card Status* video card request), the status of the RS-232 output can be determined. The capacity of the output buffer is 132 characters.

**FLUSH RS-232 FIFO BUFFER.** This video card request discards any accumulated data in the input buffer for the specified port.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -- 0 -- | | | | | | | | 0 | 1 | 0 | 0 | 1 | 0 | A/B | 0 |

Where:

A/B - Port Selection
0: Port A FIFO buffer flush
1: Port B FIFO buffer flush

**ENABLE CARD INTERRUPTS.** The card is enabled for system level interrupts by writing the following video card request code to the card (as the first word of the data buffer):

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

An interrupt can be generated from four possible events, two per RS-232 port: an input buffer overrun, and the receipt of a complete record. The status of these bits can be determined by reading words 0 and 1 returned by the *Read Card Status* video card request.

When servicing an interrupt, system interrupt handling routines normally deassert subsequent card interrupts until the routine has completed. This helps to assure that subsequent interrupts are serviced. A port interrupt handling routine is an example where such a need may exist.

The *Read Card Status* video card request deasserts interrupts. The Enable Card Interrupts request can then be used immediately before an EXEC (6,0,1) call in an interrupt handling program. (The EXEC 6 call terminates the interrupt handling program in a dormant state; it is available for servicing the next interrupt.)

**WRITE SOFT CHARACTER SET.** The card has an on-board character set stored in ROM, and can accept a downloaded (*soft*) character set into 816 words of allocated RAM. To download the soft character set, use the following video card request code as the first word of the I/O EXEC call data buffer:

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

The remaining words in the buffer are the data representing the soft characters.

The download RAM is divided into two sections. The first section is 288 words long. In this section, 3 words (6 bytes) are used to map a 5 x 7 bit character cell. A set bit indicates a "turned on" pixel. For example, the following represents the character **H:**

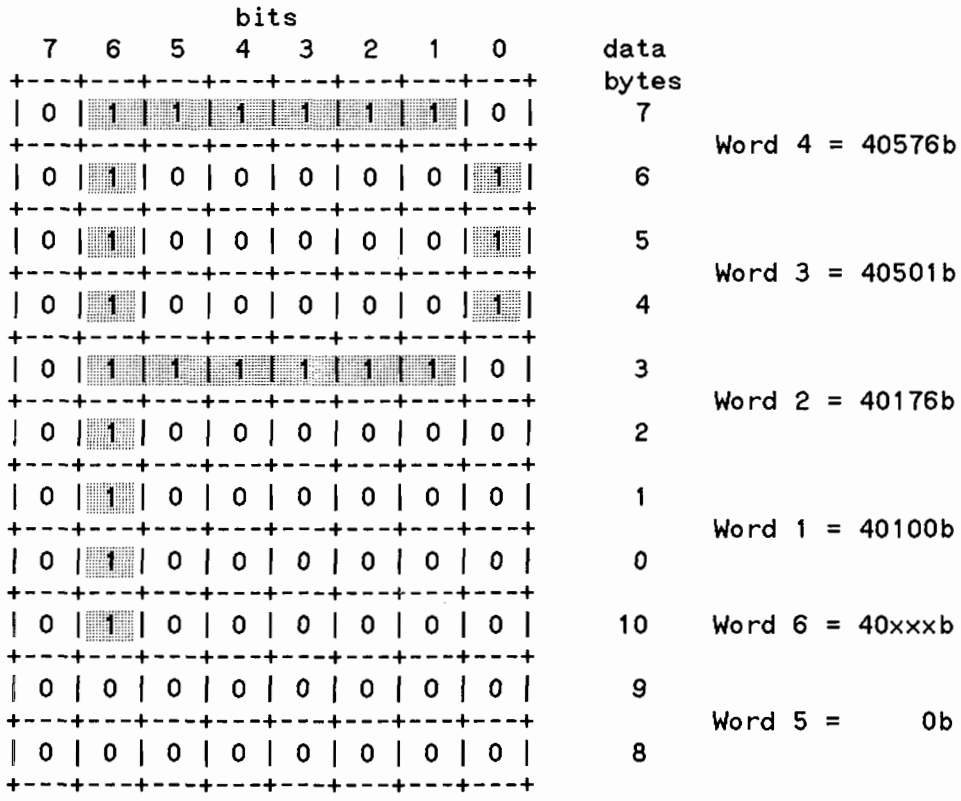
| data bytes 5 | 4 | 3 | 2 | 1 | 0 | bits |
|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 2 |
| 0 | 1 | 1 | 1 | 1 | 1 | 3 |
| 0 | 1 | 0 | 0 | 0 | 1 | 4 |
| 0 | 1 | 0 | 0 | 0 | 1 | 5 |
| 0 | 1 | 0 | 0 | 0 | 1 | 6 |
| 0 | 0 | 0 | 0 | 0 | 0 | 7 |
| Word 3 | Word 2 | Word 1 | | | | |
| 77400b | 4010b | 77410b | | | | |

Bit 7 of each byte is used for descenders, i.e., that part of the lowercase letter which descends below the main body of the letter. All bits of byte 5 should be zeroes to provide intercell spacing when outputting alphanumeric overlay characters.* Byte 5 is ignored when outputting graphics text characters.

The second section of the download RAM consists of the next 528 words. This section provides a higher resolution map, 7 x 11 bits. The following is one example of a **P:**

---

*If intercharacter spaces in the alphanumeric overlay text are not desired, then the map can be used as a 6 x 7 bit array.




Note that the above word values (in octal) assumes even numbered data bytes are the high bytes and the odd data bytes are the low bytes. However, since the low byte of word 6 starts the next soft character cell, its word values would not follow the same pattern.

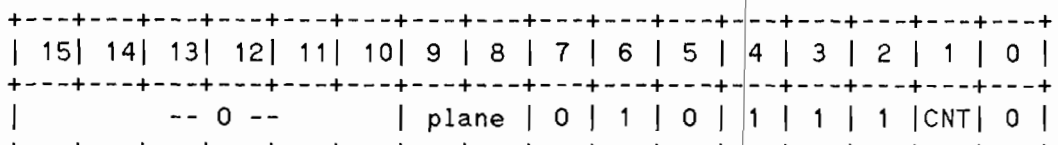
In the 7 x 11 map, bit 7 of each byte is ignored. Note the sequence of the bytes, and their relative positions when defining a character. The bottom pair of data bytes (bytes 8 and 9 in the above example) are used for descenders.

The two RAM sections are filled sequentially, byte by byte. Therefore, the data words must be packed. If you send 288 data words, then only the 5 x 7 map is loaded. If 816 data words are sent, then both maps are loaded. To fill the 7 x 11 map only, the use of 288 dummy words is required.

The alphanumeric overlay text routine uses only the 5 x 7 bit map. However, graphics text characters utilize both the 5 x 7 and the 7 x 11 bit maps. In normal operation, users will generate the same characters into both maps for greater flexibility with respect to graphics text output size (see the *TEXT_SIZE* attribute command).

The soft character set is configured or accessed by the *DESIGNATE_CHARACTER_SET*, *OVERLAY_TEXT* or *GRAPHICS_TEXT* commands. The downloaded characters, when accessed, are substituted for characters 32 through 127 (decimal) of the ASCII Character Set sequence. For example, the ASCII character T is the 84th character in the ASCII sequence, or the 52nd character beyond ASCII character number 32. With a shift to the soft character set, a T written to the card will be substituted by the 52nd character in the soft character set.

Writing to the frame buffer is accomplished using the following code in the first word of the data buffer:




Where:

plane -frame buffer plane designated to receive data
00: color plane 0
01: color plane 1
10: color plane 2
11: color plane 3

CNT -continuation bit
0: transfer is started at the beginning of the specified pla
1: transfer is started where the previous transfer ended

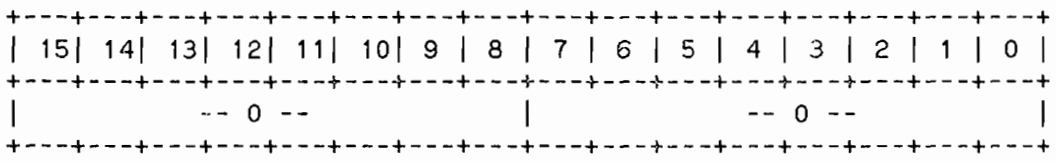
The subsequent data words are the frame buffer data. The number of 16-bit words transferred must be equal to the frame buffer blocking factor configuration previously discussed (see the paragraph *Write Card Configuration* video card request). Recall from that discussion, the default and maximum frame buffer blocking factor is 16,384 words (one plane). Thus, to write four planes of a display will require a *minimum* of four Write Frame Buffer video card requests.

When the Write Frame Buffer video card request is used, the display is *flash-filled*, that is, data is written to the frame buffer with the display screen blanked and subsequently unblanked. Flash-fill takes advantage of faster drawing times associated with blanked displays. For additional discussion on blanked screens, see the *BEGIN_FRAME* device control graphics command later in this section, and Appendix D.

After each Write Frame Buffer request completes, the screen is automatically unblanked. If it is desireable for the screen to be blanked during successive Write Frame Buffer requests (for example, when writing data to all four planes), then follow each request with a *BEGIN_FRAME* graphics command with the blanking enabled parameter set. The procedures for writing graphics commands is discussed in the Graphics Commands portion of this manual.

Although dependent on the particular application, the CNT bit is primarily used for performing multiple transfers from small buffers instead of a single transfer from a very large buffer. Note that setting the CNT bit will override the designated frame buffer plane. With the CNT bit set, the transfer is started where the previous transfer ended, which may or may not be the start of the designated plane.

**WRITE GRAPHICS COMMANDS.** For drawing an image on the screen display, graphics commands must be sent to the card in the data buffer of an I/O EXEC write. The following video card request code must be the first word of the data buffer.




The subsequent data words consist of the graphics commands. Each command has an implied or explicitly specified number of parameters. Parameter counts must be correct; otherwise, numerous errors may result. Graphics commands are discussed in detail in the Graphics Commands portion of this manual.

When drawing an image on a display screen, the normal procedure is to first issue certain graphics commands which configure attributes (such as linestyle and color), followed by other graphics commands which actually draw the image. Each command can be issued using individual data buffers; or the commands can be concatenated into a single data buffer. Since there is a fixed processing overhead associated with each buffer, a considerable performance advantage can be realized by sending several commands per buffer instead of a single command per buffer.

However, graphics commands are transferred only as fast as the video card can process each command; there could be a significant period of time when the transfer is in process. During this time, other read and write operations (for example, accessing the RS-232 ports) are not available.

## Video Card Read Requests

With the driver properly configured, the firmware expects the video card request to be located in the first word of the I/O EXEC call data buffer. Because of this, performing video card read requests becomes a two step programming process.

First, an I/O EXEC write (EXEC 2) is used to send the video card read request to the card in a one word buffer. Next, an I/O EXEC read (EXEC 1) initiates the read and establishes the buffer into which data is read.

**WRITE RAW GDC CHIP COMMANDS.** For the sophisticated user, direct commands to the Graphics Display Controller (GDC) chip used on the video output card are possible.* Use the following video card request code.

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

The subsequent data words consist of GDC chip commands. They must be packed. Multiple commands may be concatenated, as follows:

| | |
|---|---|
| data byte 0 | GDC Commands Word Count |
| data byte 1 | GDC Command 1 Parameter Count |
| data byte 2 | GDC Command 1 Opcode |
| data byte 3 | GDC Command 1 Parameter 1 |
| data byte 4 | GDC Command 1 Parameter 2 |
| . | . |
| data byte n | GDC Command 2 Parameter Count |
| data byte n+1 | GDC Command 2 Opcode |
| . | . |
| . | . |
| . | . |
| . | . |
| last data byte | Parameter Count = 0FFH (FF hex) |

The GDC Commands Word Count must equal the total number of GDC commands and parameter words (not bytes) in the data buffer. Each GDC command consists of a parameter count, followed by the proper opcode, and completed with the specified parameters. The parameter count, opcode, and subsequent parameters are all bytes, packed into the buffer. The data stream is terminated by a parameter of 0FFH.

The GDC chip has two internal registers, RA8 and RA9. If these registers are changed by any GDC commands issued, they must be set to 0FFH prior to returning to the standard card firmware. (Refer to the footnote below.)

---

*The NEC 7220 Graphics Display Controller chip is used. Although the format for passing data to the chip is provided, refer to the chip manufacturer's documentation for specific programming information.

An example of a data buffer that performs two GDC commands is provided below:

GDC Command Buffer Example

| | High Byte | Low Byte |
|---|---|---|
| word 0 | 0 | 84 |
| word 1 | 6 | 3 |
| word 2 | opcode | parameter 1 |
| word 3 | parameter 2 | parameter 3 |
| word 4 | 2 | opcode |
| word 5 | parameter 1 | parameter 2 |
| word 6 | 0FFH | 0 |

Word 0 is the video card request code. The high byte of word 1 indicates a total of 6 GDC command/parameter words. The low byte of word 1 indicates there are 3 parameters following the first opcode. Note that the low byte of word 6 is meaningless.

**WRITE FRAME BUFFER.** As an alternative to the time consuming process of rebuilding a screen display each time it is required, this request can be used to quickly output an entire screen display from A-series memory.

The video display screen is refreshed from the information contained in a 64K-word frame buffer on the card. By writing the display data to this buffer, the screen display is modified accordingly.

The frame buffer is divided into 4 groups, or planes. Each plane consists of a maximum 16,384 words (which represents one bit per pixel for a 512 x 512 pixel display configuration). The *Read Frame Buffer* video card request, discussed later, is used to read data from each plane, while the Write Frame Buffer request is used to write to each plane. When redisplaying a screen, it is important to write the data to the buffer plane from which it was originally read.

The I/O EXEC write is defined as before (see the paragraph *Video Card Write Requests*).

**EXEC (2,LU,BUFR,BUFLN)**

| | |
|---|---|
| *2* | EXEC code for I/O Write request |
| *LU* | Device selection by the logical unit number |
| *BUFR* | The buffer array containing the data to be written. The first word in the buffer must be the Video Request |
| *BUFLN* | Buffer length (+ words) |

BUFR will be a one word buffer containing the video card read request. BUFLN should be set to a buffer length of one word.

To complete the video card read request, the following I/O EXEC call is made:

**EXEC (1,LU,BUFR1,BUFLN1)**

| | |
|---|---|
| *1* | EXEC code for I/O Read request |
| *LU* | Device selection by the logical unit number |
| *BUFR1* | The buffer array to be filled with data from the card as specified by the video request |
| *BUFLN1* | Buffer length (+ words) |

As a general rule, when reading data from the card, it is very important to read the proper number of words associated with the video card read request; hence, BUFLN1 must be set correctly. Specifying too few words may result in a programming error interrupt (not generated by the card) and may schedule an interrupt handling routine if available. Specifying too many words may hang the request. (The *Read RS-232 Port* video card request is an exception and is discussed later.)

A two step process to perform reads presents a potential difficulty when interrupt handling routines are used. Interrupts may occur between the write and read EXEC calls; hence the video card read request may not properly complete. The situation becomes more complex if the interrupt handling routine also makes use of video card read requests. The use of the LURQ (Logical Unit Lock) system subroutine call to ensure read completion may be required. See the paragraph *Using Interrupts* presented later in this manual.

The various video card read requests are presented on the following pages.

**READ CARD STATUS.** To read card status, the following video card request code must be sent to the card in the buffer of the EXEC 2 call (write), followed by an EXEC 1 call (read) which specifies a buffer with length of 10 words:

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

Since a status read is often the first request after power-up, check the self-test LED on the card if the status read fails to complete. The card will not respond to backplane transactions if the self-test fails.*

The card and monitor configuration returned by a status read will match the present configuration established by either the hardware switch settings, or values reassigned in software (see the *Write Card Configuration* request).

A status read causes the card to deassert the IRQ- (Interrupt Request) line on any pending interrupt. If the program requesting the status read is not an interrupt handling program, and card interrupts are being used, the program should immediately enable card interrupts after the status read (see the *Enable Card Interrupts* request).

A Read Card Status request returns 10 words, all of which must be read. They are defined as follows:

**Word 0: Port A Dynamic Status**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ovr | | | | | aob | arp | | | | | | | | | |

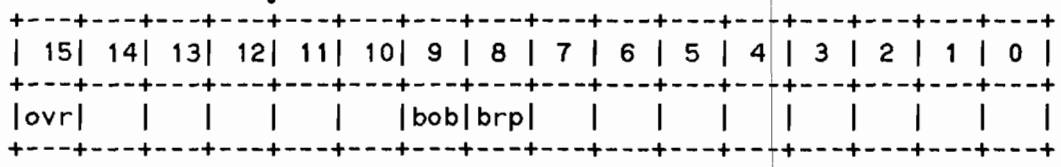
Where:

| | | |
|---|---|---|
| *ovr* | RS-232 port input buffer overrun status | 0: no overrun<br>1: overrun error (buffer overflow, characters lost) |
| *aob* | RS-232 port output buffer status | 0: output buffer empty, transfer complete<br>1: output in progress, transfer not yet complete |
| *arp* | RS-232 port status of record received (by count or match) | 0: complete record has not been received<br>1: complete record has been received |

If interrupts are enabled, bits 8 and 15 will generate interrupts when set.

*A system *control device* command can be made which will override the self-test failure. However, proper card operation cannot be guaranteed.

**Word 1: Port B Dynamic Status**

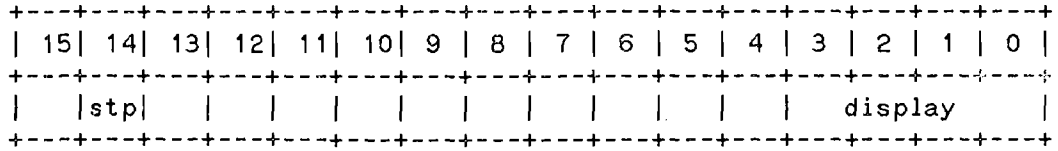



Where:

| | | |
|---|---|---|
| *ovr* | RS-232 port input buffer overrun status | 0: no overrun<br>1: overrun error (buffer overflow, characters lost) |
| *bob* | RS-232 port output buffer status | 0: output buffer empty, transfer complete<br>1: output in progress, transfer not yet complete |
| *brp* | RS-232 port status of record received (by count or match) | 0: complete record has not been received<br>1: complete record has been received |

If interrupts are enabled, bits 8 and 15 will generate interrupts when set.

**Word 2: Display Status**




Where:

`stp` -self-test pass flag
0: self-test failed
1: self-test passed

`display` -A code reflecting video monitor display parameter values which are set by card switch settings read at power-up, or software configuration using the *Write Card Configuration* request. Some display code parameters are shown below. A complete list is provided in Appendix A.

| Display code | Resolution | Horizontal Sweep Frequency | Vertical Scan Frequency |
|---|---|---|---|
| *0000* | 576h x 455v | 29.4kHz | 60.0 Hz |
| *0001* | 512h x 512v | 31.5 kHz | 57.5 Hz |
| *0010* | 576h x 455v | 28.1 kHz | 57.4 Hz |
| *0011* | 512h x 512v | 30.8 kHz | 56.1 Hz |
| *0100* | 576h x 455v | 27.0 kHz | 55.1 Hz |
| *0101* | 512h x 512v | 29.4 kHz | 53.6 Hz |
| *0110* | 576h x 455v | 25.9 kHz | 52.9 Hz |
| *0111* | 512h x 512v | 28.1 kHz | 51.4 Hz |
| *1000* | 576h x 455v | 25.0 kHz | 50.9 Hz |
| *1001* | 512h x 512v | 27.0 kHz | 49.3 Hz |
| *1010* | 576h x 455v | 24.0 kHz | 49.1 Hz |
| *1011* | 512h x 512v | 25.9 kHz | 47.3 Hz |
| *1100* | 576h x 455v | 23.2 kHz | 47.4 Hz |
| *1101* | 512h x 512v | 25.0 kHz | 45.5 Hz |
| *1110* | 576h x 455v | 22.4 kHz | 45.7 Hz |
| *1111* | 512h x 512v | 24.0 kHz | 43.9 Hz |

**Word 3: Frame Buffer Blocking Factor**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Number of Data Words per Transfer | | | | | | | | | | | | | | | |

This word identifies the number of data words per transfer when using the *Write Frame Buffer* or *Read Frame Buffer* requests. The number specified must match the number of actual data words (exclusive of the video card request word) to be transferred. The blocking factor value is configured by the *Write Card Configuration* request. The default value is 16,384 words (one plane).



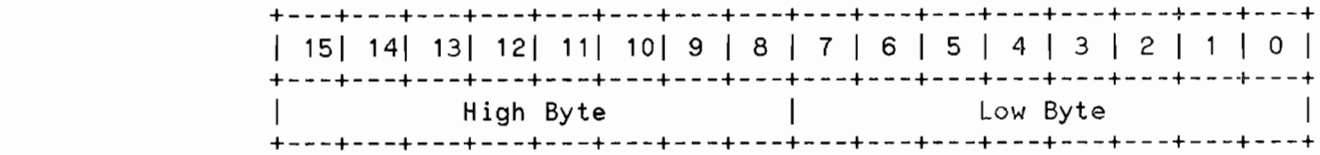
**Word 4: Port A Configuration**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | stb | ech | baud | | sec | se | bpc | e/o | par |

Where:

| | | |
|---|---|---|
| *par* | parity | 0: no parity (default)<br>1: force parity |
| *e/o* | type of parity | 0: odd parity (default)<br>1: even parity |
| *bpc* | bits per character | 0: seven bits (default)<br>1: eight bits |
| *se* | video monitor screen echo enable | 0: disable screen echo (default)<br>1: enable screen echo |
| *sec* | video monitor screen echo cursor enable | 0: no echo (default)<br>1: enable screen echo cursor |
| *baud* | baud rate | 00: 300<br>01: 1200<br>10: 2400<br>11: 9600 (default) |
| *ech* | port echo enable | 0: no echo return to RS-232 port device (default)<br>1: echo return to RS-232 port device |
| *stb* | number of stop bits | 0: one stop bit (default)<br>1: two stop bits |

**Word 5: Port A Record Definition**

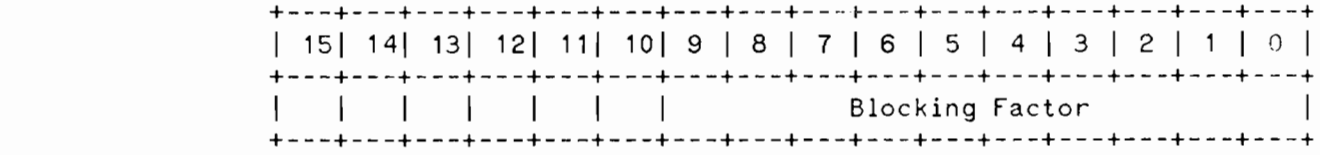



Where:

Low Byte - The number of bytes (maximum of 255) counted by the firmware to determine the record length. If the low byte is zero, record termination by count is disabled. The default condition is zero (disabled).

High Byte - The ASCII character representation, when matched by the firmware, terminates the record. If the high byte is zero, record termination by match is disabled. The default value is the *carriage return* character.

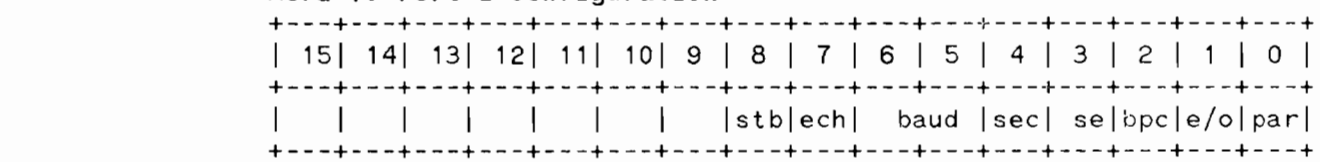
**Word 6: Port A Blocking Factor**




Where:

Blocking Factor - The maximum number of data words to be returned by a READ on the RS-232 port. The maximum value allowed is 132 words. The default value is 66 words.

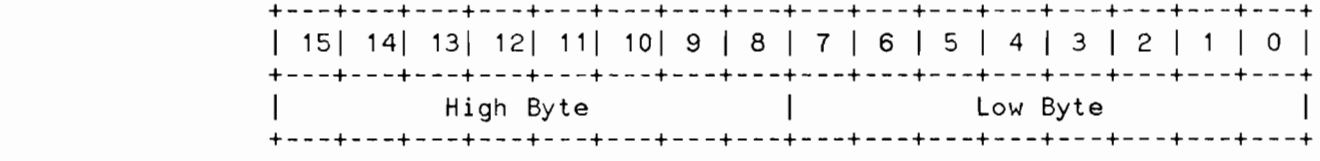
**Word 7: Port B Configuration**




Where:

The definitions of the bits are the same as in *Port A Configuration*, word 4, described above.

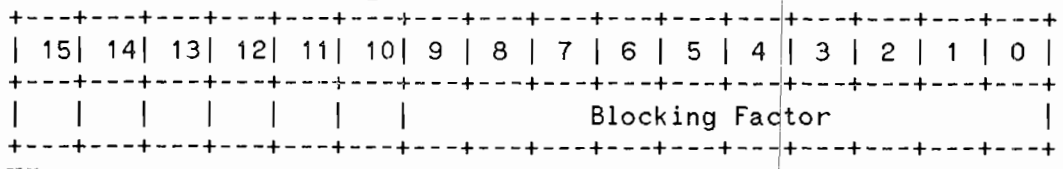
**Word 8: Port B Record Definition**




Where:

The definitions of High Byte and Low Byte are the same as in *Port A Record Definition*, word 5, described above.

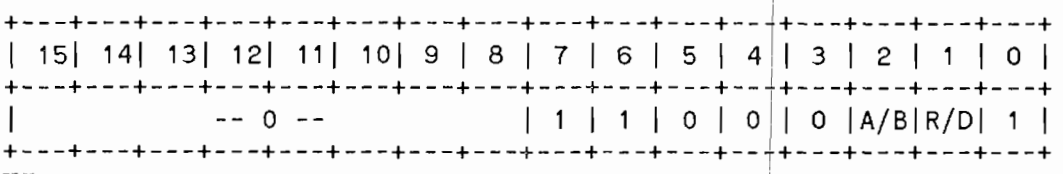
**Word 9: Port B Blocking Factor**




Where:

The `Blocking Factor` definition is the same as in *Port A Blocking Factor*, word 6, described above.

**READ RS-232 PORT.** To read from a port, the following video card request code is sent to the card in the buffer of an I/O EXEC write and followed by an I/O EXEC read. Things to consider when defining the EXEC read data buffer are discussed below.




Where:

A/B - Port Selection
0: read port A
1: read port B

R/D - Terminate Read by Record or Data
0: read to end of data or blocking factor
1: read to end of record or blocking factor

RS-232 character data is input to the port from an external device and buffered in a circular buffer of 132 bytes. The characters are read from the card in unpacked format; that is, each word of the buffer consists of the character data in the low byte, and zero in the high byte. When reading from the card, the buffer length (in words) specified in the EXEC request should be greater than or equal to the number of characters expected. The card will terminate the transfer when the condition specified by the R/D bit occurs.

If the R/D bit is zero, the read from the port will always terminate immediately with whatever data is in the incoming port buffer. Since the read terminates, inbound video card requests and graphics commands can be processed. If the R/D bit is one, the read will not terminate unless a record is present, or the number of characters present is equal to or greater than the port blocking factor.

Setting R/D to zero may be desireable if incoming data is short and comes in quickly, for example, inputs from locator devices such as RS-232 mouse devices or RS-232 bitpads. In these cases, the application program will need the ability to handle multiple data strings and save data fragments from the end of a read since data reconstruction may be required. If data is input continuously, and the input buffer is allowed to overflow, data may be lost unless dealt with by the application.

Setting the R/D bit to one is useful for record oriented inputs such as from a keyboard. The read can be configured to complete when the operator hits the carriage return key. Note that inbound requests and commands cannot be processed until after the operator responds to his prompt.

A record is defined by byte count or matching character (see paragraph *Write Card Configuration* video card request, configuration word N = 3 or 6). The RS-232 blocking factor is configured using the same request (see configuration word N = 4 or 7).

Depending on the application, it may be necessary to use a formatter for character data conversion, for example, ASCII numbers to real numbers. This task is left to the programmer.

**READ RAW GDC CHIP DATA.** For the sophisticated user, reading data from the Graphics Display Controller (GDC) chip may be desireable. Data can be read from the GDC by first writing the following video card request code in the buffer of an I/O EXEC write, followed by an I/O EXEC read with a data buffer of appropriate length:*

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Byte Count | | | | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

Where:

Byte Count - Number of bytes to read from the GDC chip, 127 bytes maximum per request, determined by the GDC command

Data words returned to the data buffer are unpacked bytes; that is, a GDC chip byte returned is in the low byte of a data word, while the high byte is meaningless. Therefore, the data buffer length of the I/O EXEC read must match the byte count of the video card request.

**READ FRAME BUFFER.** Screen display images can be stored in system memory by reading data from the frame buffer. To perform a frame buffer read, the following video card request code must be sent to the card using an I/O EXEC write and followed by an I/O EXEC read.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -- 0 -- | | | | | | plane | | 1 | 1 | 0 | 1 | 1 | 1 | CNT | 1 |

Where:

plane -frame buffer plane from which data is to be read
- 00: color plane 0
- 01: color plane 1
- 10: color plane 2
- 11: color plane 3 or overlay plane

CNT -continuation bit
- 0: transfer is started at the beginning of the specified plane
- 1: transfer is started where the previous transfer ended

*The NEC 7220 Graphics Display Controller chip is used. Refer to the chip manufacturer's documentation for available commands, characteristic of returned data, and other programming information.

The data words read are from a single frame buffer plane. The number of 16-bit words transferred must be equal to the frame buffer blocking factor configured by the *Write Card Configuration* video card request. The I/O EXEC read buffer length must be appropriately set. The default and maximum frame buffer blocking factor is 16,384 words (one plane). Therefore, to read four planes of a display will require a *minimum* of four Read Frame Buffer video card requests.

Although dependent on the particular application, the CNT bit is primarily used for performing multiple transfers to small buffers instead of a single transfer to a very large buffer. Note that setting the CNT bit will override the designated frame buffer plane. With the CNT bit set, the transfer is started where the previous transfer ended, which may or may not be the start of the designated plane.

To take advantage of faster reading times, frame buffer reads are performed with the screen blanked. After each Read Frame Buffer request completes, the screen is automatically unblanked. If desired, the screen may remain blanked for successive requests by issuing a *Begin_Frame* device control graphics command, after each request, with the blanking parameter enabled. *Begin_Frame* commands are discussed in the Graphics Commands portion of this manual.

To redraw the image, the Write Frame Buffer request is used.

**READ GRAPHICS STATUS BUFFER.** The *INQUIRE_DEVICE_ID* graphics command will provide data to a status buffer in user RAM. To read the status buffer, the following video card request code is sent to the card in the buffer of an I/O EXEC write, and followed by an I/O EXEC read with a buffer length of 8 words.:

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

See the command *INQUIRE_DEVICE_ID* for format of data returned. Note that status buffer data resulting from a graphics command will be overwritten by a subsequent command.

# GRAPHICS COMMANDS

Graphics commands are used to display images on a video monitor. They are sent to the card in the data buffer along with the Write Graphics Commands video card request.

**Graphical Configuration and Control.** The coordinate system used by the video card depends on the display configuration. Once the display is configured, the coordinate system is fixed. The resolution and aspect ratio are configured by the Write Card Configuration request. The coordinate system origin is at the upper left corner of the screen. X coordinates are the pixel count, starting from zero, from left to right across the screen. They are positive 16-bit integers. Y coordinates are pixel counts from the top to the bottom of the screen. They are also positive 16-bit integers. For example, a resolution of 576 x 455 and an aspect of 4:3 establish a range of X coordinates from 0 to 575 and a range of Y coordinates from 0 to 454.

Clipping, the process of removing those portions of graphic elements outside the viewport area, is not supported on the card. Maintaining graphic elements within the coordinate system boundaries is the user's responsibility.

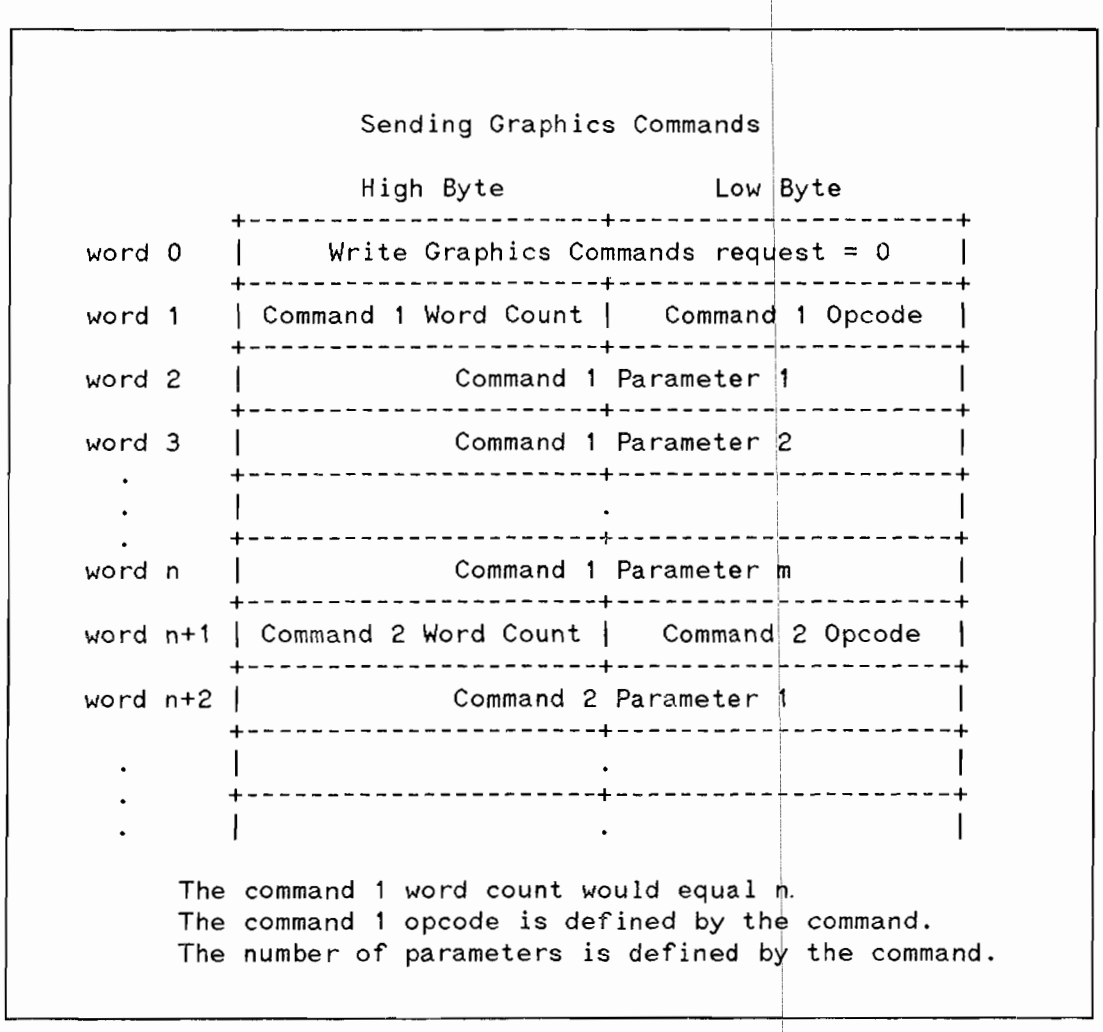
**Command Processing.** Commands are generally processed in packed binary mode, byte by byte. The high byte of a 16-bit word will be the first byte of the byte stream.

The high byte of the first command word is the length, in number of words, of the command. This word count is equal to one plus the number of parameter words in the command. The low byte is assigned the operation code (opcode) of the command to be performed. An opcode uniquely defines each command.

One or more parameter words normally complete each command. The parameter data type is inferred from the command. Some commands have variable length parameter lists; but because of the word count, there is no ambiguity.

Several commands may be concatenated within a single data buffer instead of using one data buffer per command. Due to processing overhead associated with each buffer, a performance advantage can be realized. However, note that during the transfer of a large buffer, other read and write operations are not available.

The general format of a data buffer with multiple graphics commands is shown below:




Refer to Appendix C for a simple illustration of writing graphics commands to the card. Appendix D provides card output performance data.

If a sequence of command parameters is missing or discarded, the result depends on the particular command involved.. In some cases, the missing parameters are defaulted. In other cases, the entire command will be ignored. Refer to the individual commands for details.

A system request to abort may cause a graphics command to terminate while being processed, and it is possible for a partial set of command parameters to be sent. However, a soft reset to the card is initiated which aborts the in-process command and sets the card back to its power-on defaults.

**Graphics Commands Overview.** Graphics commands can be categorized by function.

*Graphics Output Primitives* are output commands which produce basic graphic elements. They add to a drawn image. They are the only graphics commands that actually modify the frame buffer from which the screen refreshes its image.

*Output Attributes* set parameters which primarily control the appearance characteristics of an output primitive, for example, line color and style. Except for the *DEFINE_COLOR_TABLE* command, attribute commands affect only subsequent graphics output primitives; there will be no apparent change to an existing display image. If an output attribute command or parameter is ignored or not executed, the desired change in primitive appearance will not occur.

*Device Control commands* control certain screen characteristics.

*Inquiries* are used to determine card capabilities and present status. For the video output interface, there is a single inquiry command.

The available commands are listed on the next page, with details of each command provided on the following pages. Note the opcode associated with each command. It is the opcode that uniquely identifies each command to the firmware.

## Graphics Commands Summary

| Output Primitives | Opcode |
|---|---|
| POLYLINE | 25 |
| POLYGON | 26 |
| POLYMARKER | 27 |
| GRAPHICS_TEXT | 28 |
| OVERLAY_TEXT | 7 |
| CURSOR_UPDATE | 29 |
| CURSOR_OFF | 30 |
| RECTANGLE_FILL | 32 |
| CIRCLE_FILL | 33 |
| SCROLL | 35 |

| Output Attributes | Opcode |
|---|---|
| DEFAULT_ALL_ATTRIBUTES | 10 |
| DEFINE_COLOR_TABLE | 11 |
| COLOR_INDEX | 12 |
| LINESTYLE | 13 |
| LINEWIDTH | 14 |
| POLYGON_FILL_STYLE | 15 |
| (Reserved) | 16 |
| MARKER_SYMBOL | 17 |
| MARKER_LINE | 18 |
| DESIGNATE_CHARACTER_SET | 19 |
| TEXT_SIZE | 20 |
| TEXT_PATH | 21 |
| TEXT_SPACING | 22 |
| TEXT_ALIGNMENT | 23 |
| SET_RUBBER_BAND_START_POINT | 24 |
| RECTANGLE_FILL_PATTERN | 31 |
| DEFINE_SCROLLING_WINDOW | 34 |
| CHANGE_OVERLAY_MARGINS | 8 |
| CHANGE_OVERLAY_PLANE | 9 |

| Device Control | Opcode |
|---|---|
| BEGIN_FRAME | 1 |
| END_FRAME | 2 |
| CONTROL_BLINKING | 3 |
| CONTROL_INTERRUPTS | 4 |

| Inquiries | Opcode |
|---|---|
| INQUIRE_DEVICE_ID | 5 |
| (Reserved) | 6 |

**Graphics Output Primitives**

## POLYLINE, opcode 25

The POLYLINE primitive is used for drawing sequential vectors (or lines). The parameters of the command consist of coordinate pairs. An implicit move is done to the first coordinate pair as a starting point. A line is drawn to the second coordinate pair, and this pair becomes the new starting point. Additional coordinate pairs follow the pattern by first serving as line end points, then acting as line starting points.

The appearance of each line segment is determined by the line attribute settings of linestyle, linewidth, and color.

The POLYLINE command format is as follows:

POLYLINE Primitive

| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count = 2N + 3 | Opcode = 25 decimal |
| word 2 | Start Position coordinate X | |
| word 3 | Start Position coordinate Y | |
| word 4 | End Point X1 | |
| word 5 | End Point Y1 | |
| . | . | |
| . | . | |
| . | . | |
| word n | End Point XN | |
| word n+1 | End Point YN | |

N = Number of lines to be drawn.
If there are three coordinate pairs given, two lines will be drawn; X,Y to X1,Y1 and X1,Y1 to X2,Y2.

If coordinate parameters are missing, the command will be ignored. If coordinate parameters extend beyond screen boundaries, the command will be executed, but improper position and color will result.

## POLYGON, opcode 26

A polygon is a multisided, closed plane figure. Using the POLYGON primitive, a sequence of line segments are drawn to form the polygon boundaries. The polygon is drawn, closed, and filled using polygon attributes that are in effect during the draw.

The first parameters of the command are the X and Y coordinates from which the polygon starts. Subsequent coordinates identify additional vertices of the polygon. An implied connection is made between the starting coordinate pair and the last coordinate pair of the parameter list. The card will accept up to 127 vertices.

For possible polygon fill styles provided by the firmware, see the *POLYGON_FILL_STYLE* command. Polygon fills for self-intersecting polygons use a boundary-crossing algorithm to determine which side of a boundary to fill.

The POLYGON command format is as follows:

POLYGON Primitive

| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count = 2N + 3 | Opcode = 26 decimal |
| word 2 | Start Position coordinate X | |
| word 3 | Start Position coordinate Y | |
| word 4 | End Point X1 | |
| word 5 | End Point Y1 | |
| . | . | |
| . | . | |
| . | . | |
| word n | End Point XN | |
| word n+1 | End Point YN | |

N = Number of lines to be drawn, minus one. For example, for five boundary lines to be drawn, N = 4.

With off-screen coordinates, polygon boundary location errors and color errors are likely.

## POLYMARKER, opcode 27

With this command, a graphical symbol, or marker, can be used to highlight a specified point. The marker symbol selected is centered about each coordinate pair listed as command parameters.* To select a marker symbol and color, use the *MARKER_SYMBOL* and *COLOR_INDEX* attribute commands.

In addition, you can choose whether or not to draw a line between the coordinates using the *MARKER_LINE* attribute command. If lines are drawn, the existing polyline attributes in effect will govern line appearance. Lines are always drawn before markers to insure marker clarity.

POLYMARKER Primitive

| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count = 2N + 1 | Opcode = 27 decimal |
| word 2 | Coordinate X1 | |
| word 3 | Coordinate Y1 | |
| word 4 | Coordinate X2 | |
| word 5 | Coordinate Y2 | |
| . | . | |
| . | . | |
| . | . | |
| word n | Coordinate XN | |
| word n+1 | Coordinate YN | |

N = Number of positions to be marked.

Coordinates provided which are outside of screen boundaries will be executed; however, improper location and color are likely.

*Markers are always centered about a point. Note that this differs from character strings, which can be aligned relative to a point -- for example, left or right justified to the point.

## GRAPHICS_TEXT, opcode 28

Similar to alphanumeric overlay text, graphics text uses either the standard character set in ROM, or a soft character set downloaded into RAM. However, unlike overlay text, graphics text can be output in various sizes and in all four frame buffer planes simultaneously, similar to other graphic elements.

Command parameters consist of a coordinate pair and characters represented by ASCII code. The coordinate pair serves two purposes. First, it is a reference point around which text is aligned in various orientations. Second, it is the carriage return point, i.e., the resulting position when a carriage return character is sent.

When using the standard character set, the ASCII code sent will display the respective ASCII character. If a soft character set is configured, ASCII character codes 32 through 127 (decimal) will each display an assigned soft character. Soft characters are assigned to ASCII code via their relative position in RAM. See the Write Soft Character Set request for additional information.

Certain ASCII characters are used by the video card for special control functions (non-printing). They are shown with their associated decimal code below:

| Code | Character | Description |
|---|---|---|
| 08 | BS | Back Space (non-erasing) |
| 09 | HT | Horizontal Tab (1 space) |
| 10 | LF | Line Feed |
| 11 | VT | Inverse Linefeed (1 space Vertical Tab) |
| 13 | CR | Carriage Return |
| 14 | SO | Locking Shift Out |
| 15 | SI | Locking Shift In |

**SO** is used to shift to a soft character set. ASCII code (32 to 127 decimal) following the SO will display the assigned soft character. **SI** is used to shift back to the standard character set.

The characters are displayed according to the attributes presently configured. The *TEXT_PATH* attribute command allows character strings to be displayed parallel to either the vertical or horizontal axis. Also, individual characters can be rotated in 90 degree increments. The *TEXT_ALIGNMENT* attribute positions character strings relative to the coordinate pair. The *TEXT_SIZE* attribute configures character sizes which are integral multiples of the two character cell sizes, 5 x 7 and 7 x 11 bits. (For a discussion on character cells, see the Write Soft Character Set video card request.)

An implicitly controlled attribute is the start-of-text position resulting from a carriage return. For graphics text output, the alignment attribute that is in effect during the GRAPHICS_TEXT command positions the text relative to the carriage return point. The carriage return point can be changed during the GRAPHICS_TEXT command using the linefeed (LF) or inverse linefeed (VT) control characters.

The format of the command is as follows:

GRAPHICS_TEXT Primitive

| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count =(N/2)+ 3 | Opcode = 28 decimal |
| word 2 | Start Position coordinate X | |
| word 3 | Start Position coordinate Y | |
| word 4 | Character 1 | Character 2 |
| word 5 | Character 3 | Character 4 |
| . | . | |
| . | . | |
| . | . | |
| word n | Character N-1 | Character N |

N = Number of characters
(N must be even; character list
can end with a null character)

Text strings that run off the screen will result in position and color errors. An unrecognized control character may be ignored or cause spurious output.

## OVERLAY_TEXT, opcode 7

This primitive is used for outputting non-graphic alphanumeric text, which relates to text displayed on non-graphics terminals. It can be used where quick text output is desireable without disturbing the graphics image, for example, when displaying program status or error messages.

Non-graphic alphanumeric text uses only the 5 x 7 bit character cells for both ROM and RAM based character sets. Text is output in only one size.

Non-graphic text is normally drawn in only a single plane which masks or *overlays* any data in the remaining three planes. The overlay plane defaults to plane 3 (also referred to as the fourth plane, highest plane, or MSB - most significant bit - plane); the overlay plane can be reassigned using the *CHANGE_OVERLAY_PLANE* command.

The overlay text color is user selectable. The procedure is to assign the same color intensity value to the eight colormap registers indexed by a set bit in the overlay plane (see the *DEFINE_COLOR_TABLE* and *COLOR_INDEX* commands). Up to eight other colors are assigned to the remaining colormap registers which are indexed by the remaining three planes. For example, in the default case, white is assigned to colormap registers which are indexed by frame buffer plane values of 8 through 15 (corresponding to plane 3 bit set). Other colors are assigned to registers 0 through 7 (plane 3 bit is clear). Thus, in the default case, overlay text is displayed in white.

In unusual circumstances, non-graphics text which fully use 16 colors can be output. However, in normal operation, only character outlines will be distinguishable since the set bits of a character cell will match the surrounding color, while the cleared bits of the character cell will contrast with the set bits (see the Write Soft Character Set video card request for a discussion of character cells).

The video card maintains a *current position* for overlay text output. If text reaches the right side of the window, it continues one line down at the left side of the window. However, if text reaches the right side of the window at the window bottom, it continues on the left side of the window on the *same* line. The *CHANGE_OVERLAY_MARGINS* command allows setting screen boundaries (windows) for overlay text

Note that the scrolling capability can be used with overlay text (see the *DEFINE_SCROLLING_WINDOW* attribute and *SCROLL* primitive commands).

The format of the command is shown below.

OVERLAY_TEXT Primitive

| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count =(N/2)+ 2 | Opcode = 7 decimal |
| word 2 | Cursor Flag | Null/Optional Char |
| word 3 | Character 1 | Character 2 |
| word 4 | Character 3 | Character 4 |
| . | . | |
| . | . | |
| . | . | |
| word n | Character N - 1 | Character N |

N = Number of characters (N must be even; character list can end with a null character)

The low byte of word 2 can be the null character or an optional character

When using the command, the high byte of the first parameter word is a cursor flag. A non-zero flag indicates that a block cursor will be drawn at the current position following the completion of the output string.

Character parameters are represented by ASCII code. When using the standard character set, the ASCII code used will display the respective ASCII character. If a soft character set is configured, ASCII codes 32 through 127 (decimal) will each result in the display of an assigned soft character. Soft characters are assigned to ASCII codes via their position in RAM. See the Write Soft Character Set request for additional information.

Certain ASCII characters may be imbedded in the parameter string for special control functions. They are shown with their associated decimal code below:

| Code | Character(s) | Description |
|---|---|---|
| 08 | BS | Back Space - move the current position left one position; if at the left of the window, then stay at the left of the window |
| 09 | HT | Horizontal Tab - move the current position right one position; if this is off the right of the window, go to the left side of the window one row down; if at the bottom right of the window, go to the left side of the window on the same line |
| 10 | (J) LF | Line Feed - move the current position down one line; if at the bottom, remain there |
| 11 | (K) VT | Vertical Tab - move the current position up one line; if at the top of the window, remain there |
| 13 | CR | Carriage Return - move the current position to the left margin of the window on the same line |
| 14 | SO | Shift Out - replace the character shown from the standard character set with the corresponding character from the soft set |
| 15 | SI | Shift In - use the standard character set for the character code indicated |
| 27 & 70 | ESC F | Escape F - move the cursor to the low home position, which is the lower left corner of the window |
| 27 & 74 | ESC J | Escape J - clear the current text plane |
| 27 & 104 | ESC h | Escape h - move the cursor to the home position, which is the upper left corner of the window |

Unrecognized control characters may be ignored or cause spurious output.

If a soft character set is configured, ASCII code in the range 33 through 126 decimal (corresponding to ASCII printing characters) which lack an assigned soft character in RAM will be ignored.

## CURSOR_UPDATE, opcode 29

The CURSOR_UPDATE primitive activates the graphic cursor display and places it at a specified location.

There are four types of graphic cursors available.

1. Digitizing cross cursor, which consists of two 40 pixel intersecting lines, one horizontal and the other vertical. The intersection is the specified point.
2. Full screen cross, consisting of a horizontal and a vertical line that extend across the length and height of the screen. The intersection is the specified point.
3. A rubber band box, which is the rectangle formed by a starting point as one corner, and the specified point as the diagonal corner.
4. A rubber band line, which is the line formed by a starting point as one end, and the specified point as the other.

The rubber band box and the rubber band line cursors require start points using the *SET_RUBBER_BAND_START_POINT* attribute command. If not provided, these start points will default to the upper left corner of the screen.

All four cursors can be visible on the screen at the same time. For example, a digitizing cursor with attached rubber band line can be created by combining the two cursor types. Two primitives are used; both specify the same position coordinates.

A cursor command with the same cursor type as a previous command will erase the previous cursor regardless of its location. The graphics cursor is normally created by drawing vectors that complement the appropriate bits in color plane 3 (the fourth plane) of the frame buffer memory. This is the default condition. Cursor color and plane control are governed by the *DEFINE_COLOR_TABLE* and *COLOR_INDEX* commands.

The command format is as follows:

CURSOR_UPDATE Primitive

| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count = 4 | Opcode = 29 decimal |
| word 2 | Cursor Type | |
| word 3 | Position coordinate X | |
| word 4 | Position coordinate Y | |

Cursor Type: 0 = full screen cross
1 = digitizing cross
2 = rubber band box
3 = rubber band line

The portion of a cursor drawn near the edge of the screen will be clipped to the screen boundaries.

A cursor command specifying an off-screen coordinate, or an unrecognized cursor type, will result in no cursor drawn.

## CURSOR_OFF, opcode 30

The CURSOR_OFF primitive turns off a specified cursor type, determined by a parameter value in the command.

The format of the command is as follows:

CURSOR_OFF Primitive

| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count = 2 | Opcode = 30 decimal |
| word 2 | Cursor Type | |

Cursor Type:
0 = full screen cross
1 = digitizing cross
2 = rubber band box
3 = rubber band line
16 = all cursors

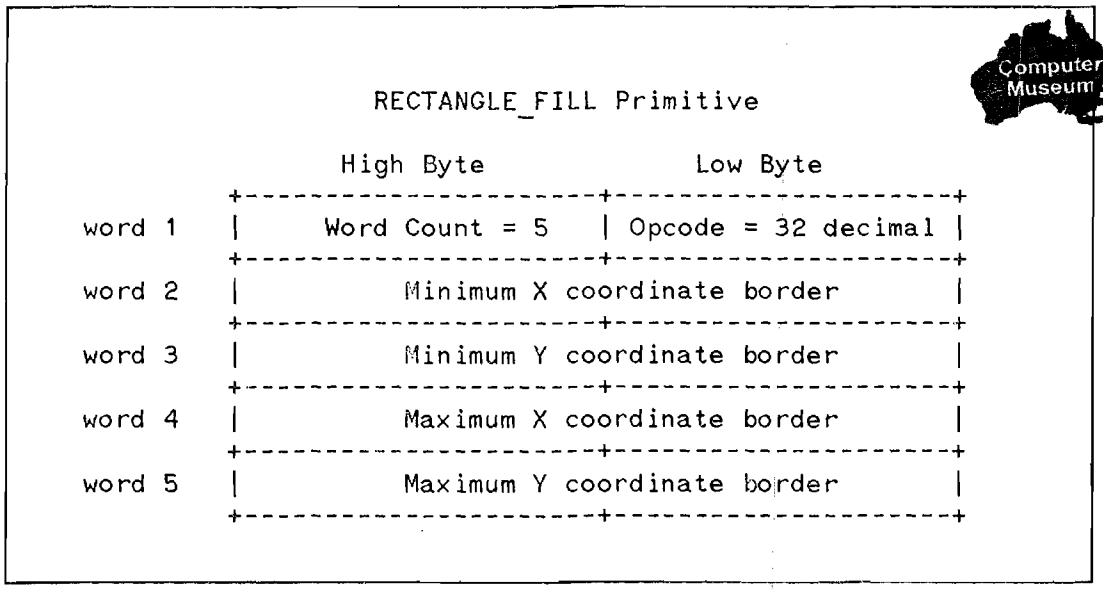
If a command is issued with an unrecognized cursor type, it will be ignored.

## RECTANGLE_FILL, opcode 32

The RECTANGLE_FILL primitive defines a rectangular boundary by specifying minimum and maximum X and Y coordinates in the parameter list. The rectangle is drawn, closed, and then filled with an 8 x 8 pixel pattern which is defined by the *RECTANGLE_FILL_PATTERN* attribute command.

Starting from the upper right corner of the rectangle, the selected 8 x 8 pixel pattern fills the rectangle row by row, traversing from right to left.

The format for the command is as follows:




With missing parameters, the command will be ignored. For coordinate errors, such as off-screen coordinates, the command will be executed with color and position errors likely.

## CIRCLE_FILL, opcode 33

The CIRCLE_FILL primitive draws and fills a circle with a solid color. Command parameters specify the X and Y coordinate center, and the circle radius.

Note that only a solid fill is allowed. The fill color is determined by the *DEFINE_COLOR_TABLE* settings and the *COLOR_INDEX* attribute command for polygons.

The command format is as follows:

CIRCLE_FILL Primitive

| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count = 4 | Opcode = 33 decimal |
| word 2 | Center coordinate X | |
| word 3 | Center coordinate Y | |
| word 4 | Circle Radius | |

Coordinate errors will likely result in location and color errors.

## SCROLL, opcode 35

The SCROLL primitive moves rows of pixels within a defined area of a display either up or down. It is normally used with alphanumeric overlay text, but will function with single plane graphic images as well.

The boundaries of the area to be scrolled, called the scrolling window, are specified by the *DEFINE_SCROLLING_WINDOW* attribute command.

For alphanumeric overlay text, setting the margins within the scrolling window boundaries is accomplished using the *CHANGE_OVERLAY_MARGINS* command.

Pixel rows moved off the scrolling window are removed from frame buffer memory, and therefore lost. They are replaced on the opposite side of the scrolling window with cleared pixel rows which can subsequently be filled with new information.

The command parameters specify the number of pixel rows to move and the direction, either up or down. The command format is as follows:

SCROLL Primitive

| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count = 3 | Opcode = 35 decimal |
| word 2 | Number of Pixel Rows | |
| word 3 | Direction | |

Direction: 1 = up
2 = down (default)

**Output Attribute Commands**

## DEFAULT_ALL_ATTRIBUTES, opcode 10

This command sets all attributes to their default values. As a method for setting attributes to a known state, this command is used during the normal power up sequence and after a system abort request.

The DEFAULT_ALL_ATTRIBUTES command does not affect the contents of the frame buffer, therefore the screen image shapes will not change. However, the appearance may change because the color table and index settings assume default values.

Default settings include the following:

Color Attributes: Background defaults to black (color 0), primitives default to white (color 1).

Color Plane Operation: Planes 0 through 2 are color planes, plane 3 is the overlay plane

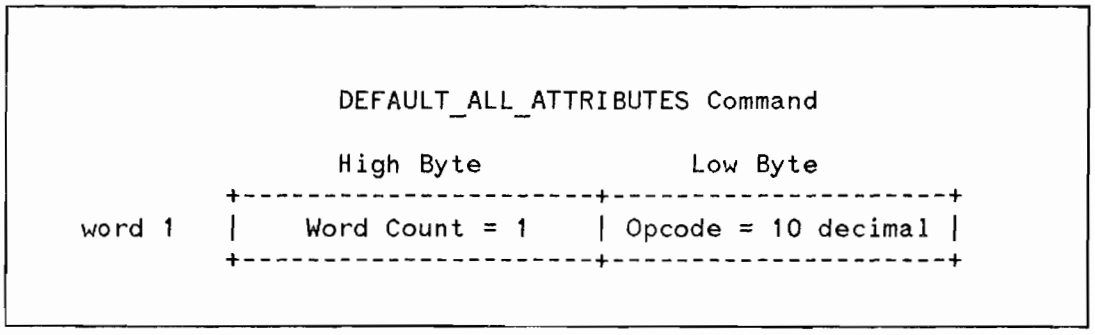
Line Style: Solid line

Line Width: 1 pixel

Text Size: 1 (5 x 7 pixel cell)

Overlay Text Current Position: Upper left corner of the screen

The command format is as follows:

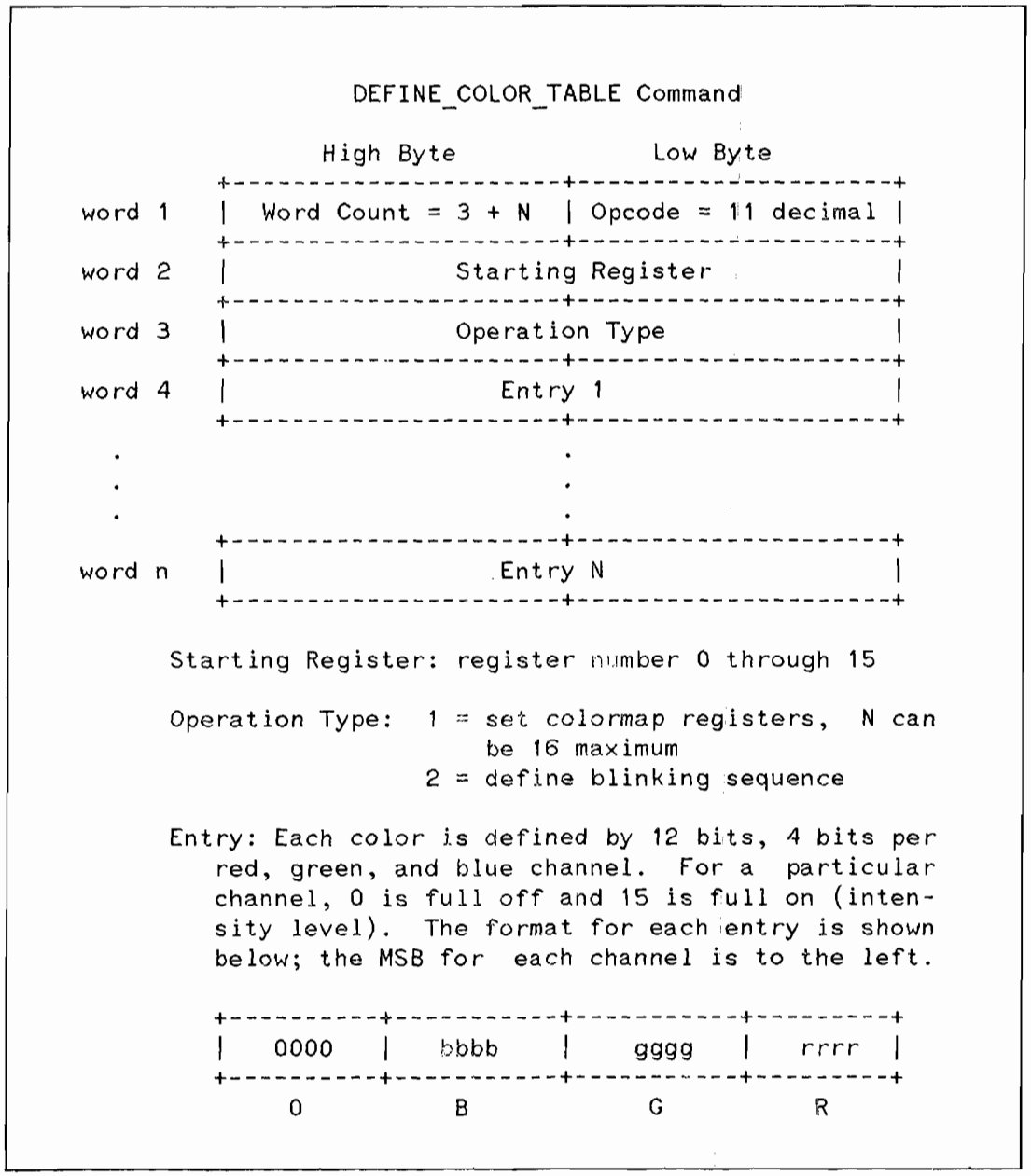



Note that there are no command parameter associated with this command.

## DEFINE_COLOR_TABLE, opcode 11

The DEFINE_COLOR_TABLE attribute command is used to load the colormap registers in RAM with colors. Each of 16 registers (0 through 15) can be loaded with one color from 4096 choices.

The command format is as follows:



The command can perform one of two functions, depending on the setting of the second parameter in the command string.

**Operation Type 1.** The set of colors that may be displayed on the screen is loaded. Since the command is retroactive, primitives using color attributes are immediately redisplayed with the new colors loaded. Each pixel is *indexed* to a particular register; by changing the register color, the pixel color will change. See the *COLOR_INDEX* command for setting and modifying the indices.

Indicated below are default colors assigned to the registers. Register 0 is the background color and defaults to black. All primitives default to white.

| Register | Color | (0, B, G, R) Values |
|---|---|---|
| 0 | Black | (0, 1, 1, 1) |
| 1 | White | (0,15,15,15) |
| 2 | Red | (0, 1, 1,15) |
| 3 | Green | (0, 1,15, 1) |
| 4 | Yellow | (0, 1,15,15) |
| 5 | Blue | (0,15, 1, 1) |
| 6 | Magenta | (0,15, 1,15) |
| 7 | Cyan | (0,15,15, 1) |
| 8-15 | (Overlay) | (0,15,15,15) |

The Starting Register is the register that will be filled with Entry 1. The registers increment by one up to register 15. Additional entries provided after register 15 is filled will be ignored.

If the Starting Register exceeds limits or is not recognized, the command is ignored.

Notice that (0,1,1,1) is used for black. This provides the pedestal needed by many monitors. In this case, (0,0,0,0) becomes *blacker than black* and is used during retrace.

**Operation Type 2.** The color entries are interpreted as a circular list in conjunction with the *CONTROL_BLINKING* device control command. After each expiration of the *CONTROL_BLINKING* time interval, the specified *CONTROL_BLINKING* color register is loaded with the next sequential color entry. At the end of the color entry list, entry 1 is loaded again and the cycle repeats.

For all 16 registers, a total of 120 entries can be defined.

## COLOR_INDEX, opcode 12

The color registers are filled with user selected colors using the *DEFINE_COLOR_TABLE* command. The COLOR_INDEX command links the basic graphic elements to the 16 color registers. This linkage determines the color of the output primitives.

A *color index* is a number, 0 to 15, representable by four bits. Each index acts as a pointer to a particular color register in RAM. For example, index 0 points to register 0, index 1 to register 1, and so on.

Each screen pixel is four planes deep in frame buffer memory. For each pixel, the value of the bits in the four planes define the color index value associated with the pixel. A pixel assumes a particular color because its color index value points to a color register.

For a given display, a pixel's color can be modified by either changing the color register data (*DEFINE_COLOR_TABLE* command), or by drawing over the pixel to change its color index number (pointing to a different color register). By using the *DEFINE_COLOR_TABLE* command, the appearance of the screen image will be changed without modifying the contents of the frame buffer.

The COLOR_INDEX command has three parameters. The `Attribute Identifier` parameter determines whether the command will apply to a particular primitive only or groups of primitives. For example, an attribute identifier value of 4 indicates that only graphics text colors are impacted; lines or marker symbols are not affected. A value of 3 indicates graphics text, lines and marker symbol colors are all affected.

Depending on the `Operation Type` described below, the `Color Entry` parameter provides the data with which to modify the value of a color index. In the simplest case, it is the value assigned to a color index.

The `Operation Type` parameter determines how the `Color Entry` parameter will modify color indices. There are four alternatives:

1. Complementary mode. The color entry is exclusively OR'd with the frame buffer (color index value). Using this mode, a figure drawn twice with the same color entry will be erased by the second drawing since the color index is complemented back to its original value. This method is used with cursor moves; an existing cursor is erased while a cursor at a different coordinate is drawn. For example, if a vector drawn with color index 2 intersects with a color index 4 vector, the intersection will be color index 6.

2. Non-Dominant mode. On average, this mode will be the fastest to use. The color entry is OR'd with the frame buffer. Thus, bits in the frame buffer are set, but never cleared. If a vector drawn with color index of 2 intersects with another color index 2 vector, the intersection will be color index 2. A color index 2 vector crossing a color index 4 vector will result in a color index 6 intersection. If color entry values are restricted to the range 0 through 7, the MSB plane 3 (the fourth plane) will not be modified. If color entry values are limited to even values, the lsb plane 0 (the first plane) will not be modified.

3. Dominant 3-plane mode. This is the default mode. Color entries are limited to values 0 through 7, and the msb plane 3 is used as an overlay plane. The frame buffer (color index) assumes the value of the color entry, except the plane 3 bit is ignored. If two vectors with color index values 0 through 7 intersect, the intersection will be the color of the last vector drawn. Since the fourth bit of the color entry - corresponding to color entry values of 8 and above - is ignored, an existing image previously drawn in the overlay plane will not be changed.

4. Dominant 4-plane mode. In this mode, the frame buffer assumes the value of the color entry. Since an overlay plane is not used, the full 16 colors can be used. The color entry becomes the color

index. If two vectors intersect, the color of the intersection will be the color of the last vector drawn.

The command format is described below.

COLOR_INDEX Command

| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count = 3 or 4 | Opcode = 12 decimal |
| word 2 | Attribute Identification | |
| word 3 | Color Entry | |
| word 4 | Operation Type | |

Attribute Identification:

0 = all (except background and cursors)
1 = lines only
2 = polygon fill
3 = lines, graphics text, markers
4 = graphics text only
5 = marker symbols only
6 = cursor type 0 (full screen cross)
7 = cursor type 1 (digitizing cross)
8 = cursor type 2 (rubber band box)
9 = cursor type 3 (rubber band line)

Color Entry:

0 to 15 if Operation Type is 1, 2, or 4
0 to 7 if Operation Type is 3 (default)

Operation Type (optional):

1 = Complementary. A set bit in the present color entry will cause the complement of the corresponding bits in the previous color index. (Exclusive OR)
2 = Non-Dominant. The bits in the present color entry are OR'd with the bits in the previous color index.
3 = Dominant 3-plane. The bits in the present color entry overrides the previous index, EXCEPT for the fourth bit, or MSB, which is ignored. (Default)
4 = Dominant 4-plane. The bits in the present color entry overrides the previous index.

## LINESTYLE, opcode 13

A linestyle is a recurring pattern of dots, blanks, short line segments and long line segments. This command specifies the pattern to be used when drawing a line.

Although there are no user defined linestyles on the card, six predefined linestyles are available. They are as follows:

| Linestyle | Description |
|---|---|
| 1 | Solid |
| 2 | 50% dash |
| 3 | 66% dash |
| 4 | Center dot between solid |
| 5 | Center dash between solid |
| 6 | Dotted |

All of the linestyles, except dotted, are *adaptive*. By adaptive, we mean that continuous vectors are drawn with an integral number of linestyle repeats. When a new line is drawn as a result of an implicit or explicit move or draw, the line pattern is reset to the beginning of the pattern. This ensures, for example, that a corner is closed with a solid portion of the linestyle.

An unrecognized linestyle (e.g., 0 or 7) will be ignored and will not change the existing linestyle. The default linestyle is 1.

Default lengths vary with the linestyle. For linestyles 1, 2, and 3, default length is 20 pixels. For linestyles 4 and 5, the default length is 40 pixels. Linestyle 6 consists of every other pixel set and has no default length. If length is specified as anything less than its default value, it will assume the default value.

The command format is described below:

LINESTYLE Command

| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count = 2 or 3 | Opcode = 13 decimal |
| word 2 | Linestyle | |
| word 3 | Length | |

## LINEWIDTH, opcode 14

This attribute command specifies the width of lines that are used by the *POLYLINE* primitive. The thinnest line that can be specified is one pixel in width. However, for lines drawn at angles - i.e., not horizontal or vertical - the *apparent* thickness may be slightly larger than the width specified.

The specified linewidth default is 1.

The command format is described below:

LINEWIDTH Command

| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count = 2 | Opcode = 14 decimal |
| word 2 | Linewidth | |

Linewidth: Specified in odd multiples of the thinnest line, as follows

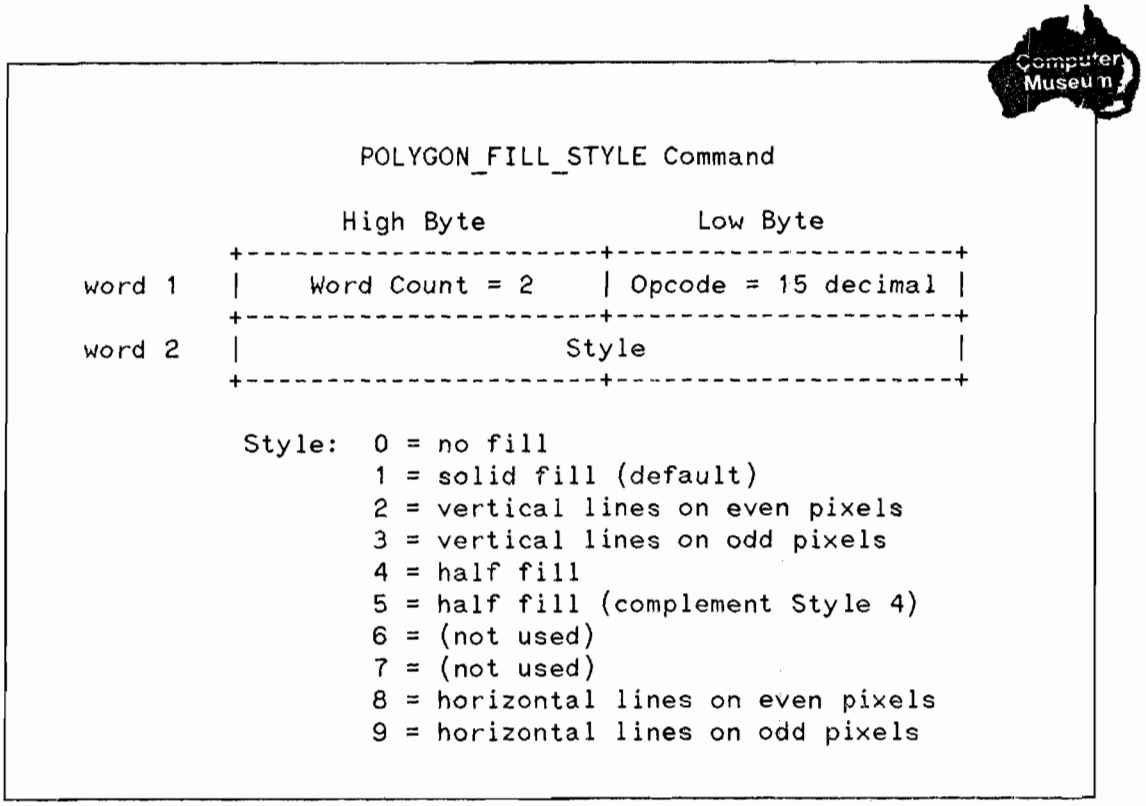
| Specified Width | Actual Width in Pixels |
|---|---|
| 0 | 1 |
| 1 | 1 (default) |
| 2 | 3 |
| 3 | 5 |
| 4 | 7 |
| . | . |
| n | (2n)-1 |
| . | . |
| 175 | 349 |

Note that thick lines will be drawn slightly past the coordinate pair specified. Thus, two thick lines meeting at a 90 degree angle to each other will be properly joined. When joining lines at acute angles, fine-tuning may be desireable.

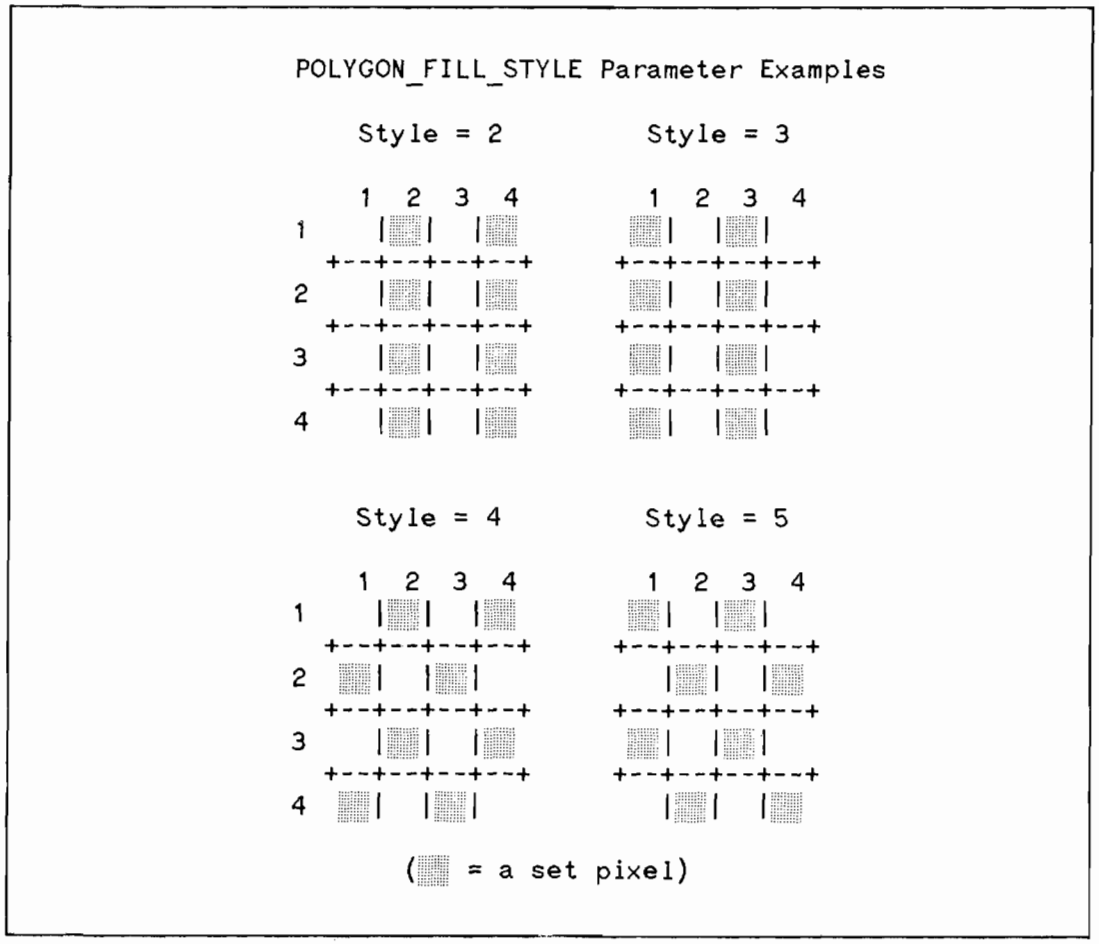
## POLYGON_FILL_STYLE, opcode 15

This attribute assigns the type of pattern to fill a polygon created by the *POLYGON* primitive. The default pattern is a solid fill.

The command format and fill patterns available are described below:




The designation between *even* and *odd* pixels within the same style is meant to show complementing patterns. This is shown in the examples below.




Unrecognized parameter values will result in unpredictable styles. These will be visible on the screen.

## MARKER_SYMBOL, opcode 17

This attribute specifies the marker symbol drawn by the *POLYMARKER* primitive. The default marker symbol is the dot.

The command format and available marker symbols are described below:

MARKER_SYMBOL Command

| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count = 2 | Opcode = 17 decimal |
| word 2 | Marker Number | |

Marker Number:

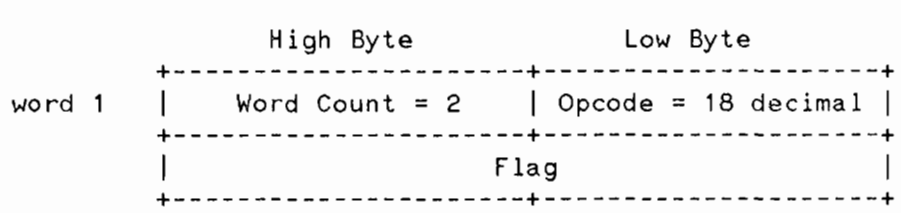
| | | |
|---|---|---|
| 1 = . (dot) | 7 = rectangle | 13 = 3 |
| 2 = + (plus) | 8 = diamond | 14 = 4 |
| 3 = * (asterisk) | 9 = rectangle | 15 = 5 |
| 4 = O | with cross | 16 = 6 |
| 5 = X | 10 = 0 | 17 = 7 |
| 6 = triangle | 11 = 1 | 18 = 8 |
| | 12 = 2 | 19 = 9 |

## MARKER_LINE, opcode 18

This attribute specifies whether or not lines are drawn between marker symbols when using the *POLYMARKER* primitive. If lines are specified, they are drawn *before* the marker symbols to prevent obscuring the markers. The lines are drawn using attributes presently in effect for the *POLYLINE* primitive.

The command format is described below.

MARKER_LINE Command




Flag: 0 = Lines not drawn
1 = Lines are drawn

In the default case, lines are not drawn.

## DESIGNATE_CHARACTER_SET, opcode 19

RAM based character sets can be downloaded to the card using the Write Soft Character Set video card request. The DESIGNATE_CHARACTER_SET command specifies whether the standard ROM based or downloaded RAM based character sets are used when executing the *GRAPHIC_TEXT* and *OVERLAY_TEXT* primitive commands.

The command format is described below. The default case is the standard ROM based character set.

DESIGNATE_CHARACTER_SET Command

| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count = 2 | Opcode = 19 decimal |
| word 2 | Character Set Number | |

Character Set Number:
0 = ROM set (default)
1 = RAM set for Overlay Text
2 = RAM set for Graphics Text
3 = RAM set for both Overlay and Graphics Text

Note that if RAM based character sets are specified but not installed, they will be represented by blanks.

## TEXT_SIZE, opcode 20

This command allows size selection of the character set when executing *GRAPHICS_TEXT* primitive commands.*

There are two basic character set sizes. One is a 5 x 7 bit cell, while the other is a 7 x 11 bit cell. Cell multiplication by integral values is used to obtain larger character cell sizes.

The command format and available character sizes are described below.

TEXT_SIZE Command

| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count = 2 | Opcode = 20 decimal |
| word 2 | Text Size | |

Text Size:

| | | |
|---|---|---|
| 1 | = 5 x 7 | character cell |
| 2 | = 7 x 11 | character cell |
| 3 | = 10 x 14 | character cell |
| 4 | = 14 x 22 | character cell |
| 5 | = 15 x 21 | character cell |
| 6 | = 21 x 33 | character cell |
| 7 | = 20 x 28 | character cell |
| 8 | = 28 x 44 | character cell |
| 9 | = 25 x 35 | character cell |
| 10 | = 35 x 55 | character cell |
| 11 | = 30 x 42 | character cell |
| 12 | = 42 x 66 | character cell |
| 13 | = 35 x 49 | character cell |
| 14 | = 49 x 77 | character cell |
| 15 | = 40 x 56 | character cell |
| 16 | = 56 x 88 | character cell |
| 17 | = 45 x 63 | character cell |
| 18 | = 63 x 99 | character cell |
| 19 | = 50 x 70 | character cell |
| 20 | = 70 x 110 | character cell |

The default text size is 1. The maximum text size is 20. A command with a text size greater than 20 will be interpreted as 20 and executed. Note that the character cell size does not necessarily increase with the `text size` parameter.

---

*This command does not apply to *OVERLAY_TEXT* output which is limited to one size only.

## TEXT_PATH, opcode 21

This command configures the orientation and path of character strings in any of four directions when drawn by a *GRAPHICS_TEXT* primitive command.

There are three text string parameters configured.

character path - describes the position of the next character. The character path of this text is *left to right.*

up-vector - describes the character orientation. This may not be intuitively obvious; the text you are reading has up-vector orientation *left to right.*

line path - describes the beginning of the next line of text (the direction of a line feed). The line path of this text is *top to bottom.*

The command format is described below. Note that character path defaults to 0, up-vector defaults to 0, and line path defaults to 3.

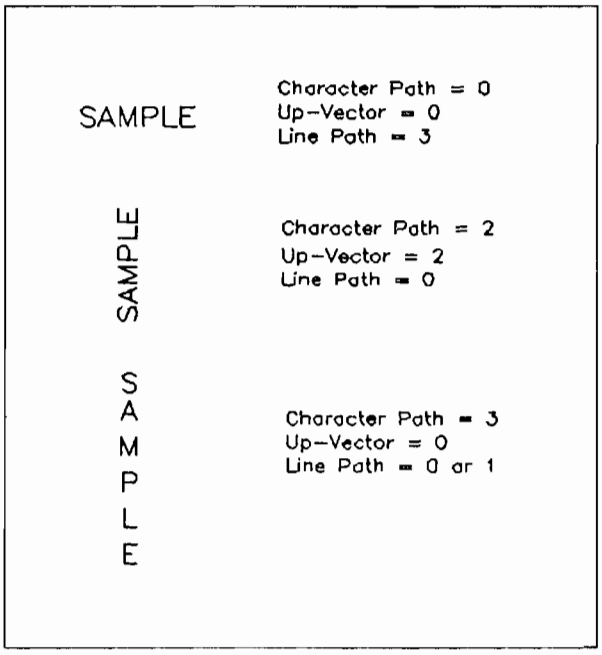
TEXT_PATH Command

| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count = 4 | Opcode = 21 decimal |
| word 2 | character path | |
| word 3 | up-vector | |
| word 4 | line path | |

Character Path, Up-Vector and Line Path are specified by two bits and have four possible values:

| Value | Meaning |
|---|---|
| 0 | left to right |
| 1 | right to left |
| 2 | bottom to top |
| 3 | top to bottom |

Some examples are provided below:




TEXT_PATH Parameter Examples

An unrecognized parameter value is ignored with no change to that parameter's current setting. Parameter errors, such as character path not perpendicular to line path, will be apparent on the display screen.

# TEXT_SPACING, opcode 22

The TEXT_SPACING command allows selection of spacing between adjacent characters and between lines of text as attributes to the *GRAPHICS_TEXT* primitive.

The command parameters specify the number of spaces between character cells. The actual number of pixels used to represent a space is not proportional to the character size specified by the *TEXT_SIZE* command. For example, if text size is 3, and TEXT_SPACING specifies one space between characters, the actual space between cells will be one pixel. If text size is changed to 1, the character space will still be one pixel.

Default parameter values are one space between characters and four spaces between lines. Maximum values are 127 spaces.

The command format is described below.

TEXT_SPACING Command

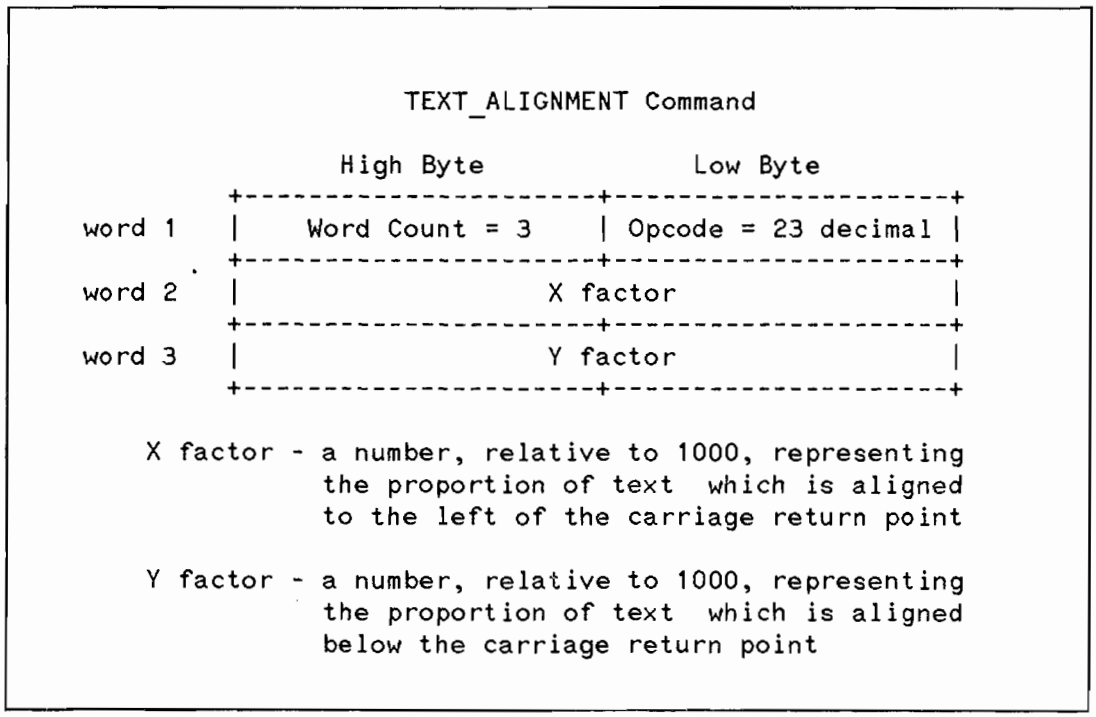
| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count = 3 | Opcode = 22 decimal |
| word 2 | 0 | Character Spacing |
| word 3 | 0 | Line Spacing |

Character Spacing - Number of spaces between characters
Line Spacing - Number of spaces between rows of text

## TEXT_ALIGNMENT, opcode 23

Recall that a carriage return point is established during each *GRAPHICS_TEXT* primitive, and can be changed within the primitive using linefeed (LF) or inverse linefeed (VT) control characters. With the TEXT_ALIGNMENT command, a string of graphics text can be positioned relative to the carriage return point. In other words, the string can be centered on, or right or left justified to the carriage return point.

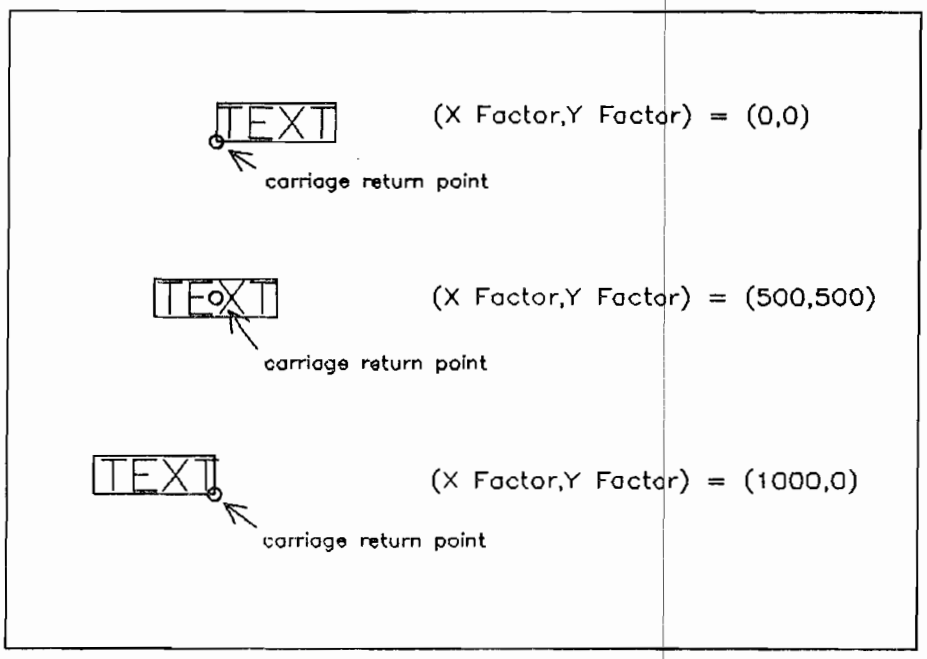
The command format is described below.




A line of text is a string of characters up to, but not including, carriage return and linefeed characters. You can envision a line of text as enclosed by a rectangle. The rectangle does not contain preceding or trailing spaces, but does include internal intercharacter spaces. Also, backspaces preceding a character are included.

An alignment with `X factor` and `Y factor` both set to zero, or (0,0), indicates the lower left corner of the rectangle is at the carriage return point. An alignment of (500,500) centers the string about the carriage return point. An alignment of (1000,0) puts the lower right corner of the rectangle at the carriage return point. These are illustrated in the examples below.

The default alignment factor is (0,0).




TEXT_ALIGNMENT Parameter Examples

# SET_RUBBER_BAND_START_POINT, opcode 24

This command establishes the fixed point for rubber band box or rubber band line cursors created by the *CURSOR_UPDATE* primitive. If the fixed point is not provided, the default point is the upper left corner of the display.

As with other attribute commands, this command will not draw the cursor, nor change the appearance of the cursor until the primitive is sent to the video card.

The command format is described below.

SET_RUBBER_BAND_START_POINT Command

| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count = 4 | Opcode = 24 decimal |
| word 2 | Cursor Type (2 or 3) | |
| word 3 | X Coordinate | |
| word 4 | Y Coordinate | |

Cursor Type: 2 = rubber band box
3 = rubber band line

# RECTANGLE_FILL_PATTERN, opcode 31

This command defines the area fill pattern used in rectangles drawn by the *RECTANGLE_FILL* primitive. Using this attribute, an 8 x 8 pixel cell is created; the user defines whether each pixel is set or cleared. The resulting cell pattern is repeatedly used to fill the rectangle.

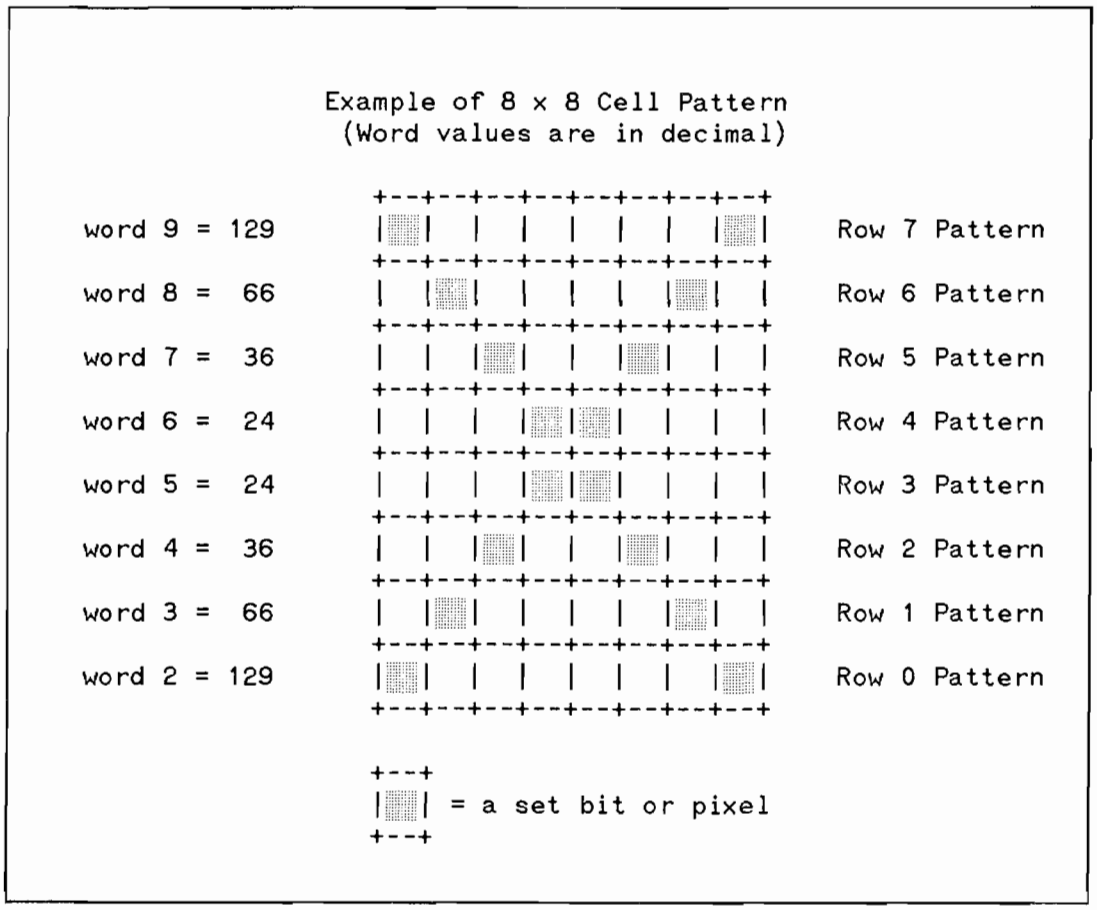
Starting at row 0, the bottom row of the cell, the programmer sets the desired bits. For example, 85 decimal sets every other bit starting with bit 0. Rows 1 through 7 are defined in a similar manner. Note that only the low byte of each parameter word is used to define the pattern; each row can contain a maximum value of 255 decimal which sets every bit.

The command format is described below.

RECTANGLE_FILL_PATTERN Command

| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count = 9 | Opcode = 31 decimal |
| word 2 | 0 (not used) | Row 0 Pattern |
| word 3 | 0 (not used) | Row 1 Pattern |
| word 4 | 0 (not used) | Row 2 Pattern |
| word 5 | 0 (not used) | Row 3 Pattern |
| . | . | |
| . | . | |
| . | . | |
| word 9 | 0 (not used) | Row 7 Pattern |

The following example shows a pattern that could be used to fill a rectangle.



## DEFINE_SCROLLING_WINDOW, opcode 34

This command defines a rectangular boundary by specifying minimum and maximum X and Y coordinate borders in the parameter list. Within this boundary, a single color plane of pixels can be scrolled using the *SCROLL* primitive. The plane designation parameter defines which of the four color planes will be scrolled. Plane 3 - the fourth and highest plane - is normally used for alphanumeric overlay text and is the default plane for scrolling.

Only integral multiples of 16 bits wide can be scrolled. Therefore, minimum and maximum X coordinate borders specified in the command are automatically expanded to accomodate scrolling. For example, if a minimum X of 5 and a maximum X of 18 were specified, the boundaries would fall between the first two groups of 16 bits (0 through 15, and 16 through 31). The resulting width of the scrolling window would be from 0 to 31. Y coordinate values are not restricted to 16 bit intervals.

Since scrolling is used primarily with alphanumeric overlay text, it will likely be necessary to set the overlay text margins within the scrolling window. See the *CHANGE_OVERLAY_MARGINS* command.

The command format is described below.

DEFINE_SCROLLING_WINDOW Command

| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count = 6 | Opcode = 34 decimal |
| word 2 | Minimum X coordinate border | |
| word 3 | Minimum Y coordinate border | |
| word 4 | Maximum X coordinate border | |
| word 5 | Maximum Y coordinate border | |
| word 6 | Plane | |

Plane: 0 = Color plane 0
1 = Color plane 1
2 = Color plane 2
3 = Color plane 3 (default)

Coordinates specified that are off the screen will result in color and position errors.

# CHANGE_OVERLAY_MARGINS, opcode 8

This command defines new borders for non-graphic alphanumeric overlay text. The default condition is the entire area of the display.

When used in conjunction with the *DEFINE_SCROLLING_WINDOW* command, overlay text can be confined to the scrolling window boundaries.

Command parameters consist of coordinates; the first coordinate pair specifies the left and bottom margins, while the second coordinate pair specifies the right and top margins.

The command format is described below.

CHANGE_OVERLAY_MARGINS Command

| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count = 5 | Opcode = 8 decimal |
| word 2 | Minimum X coordinate border | |
| word 3 | Minimum Y coordinate border | |
| word 4 | Maximum X coordinate border | |
| word 5 | Maximum Y coordinate border | |

## CHANGE_OVERLAY_PLANE, opcode 9

Alphanumeric overlay text is displayed in one plane which defaults to plane 3, the fourth or highest plane. This command allows overlay text to be written in either of the other three planes.

This command allows taking advantage of the faster speed when displaying overlay text compared to graphics text. In addition, the ability to change planes implies more than one color of overlay text can be on the screen at the same time.

The command format is described below.

CHANGE_OVERLAY_PLANE Command

| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count = 2 | Opcode = 9 decimal |
| word 2 | Plane Number | |

Plane Number: 0 = color plane 0
1 = color plane 1
2 = color plane 2
3 = color plane 3

The overlay plane is not write protected; vectors can be drawn using the overlay plane. It is the user's responsibility to ensure that overlay images are not distorted when it is undesireable to do so.

**Device Control Commands**

## BEGIN_FRAME, opcode 1

This command provides for blanking, and for clearing the display screen. If the screen is *blanked*, the screen appears black and the image is not visible, but frame buffer memory still holds the image. The image can be made visible by issuing the *END_FRAME* command.

*Clearing* the screen involves initializing frame buffer memory. By setting the bit corresponding to the appropriate plane, one or all of the frame buffer planes can be cleared.

If the command is used to clear frame buffer memory only, a new image drawn will be visible on the screen. If the command clears frame buffer memory and enables blanking, a new image drawn will write to the frame buffer but will not be visible until an *END_FRAME* command is executed. This is sometimes referred to as flash-filling a screen. The advantage to drawing with blanking enabled is that drawing time is significantly reduced; the savings depend on the image drawn.

The command format is described below.

BEGIN_FRAME Command

| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count = 3 | Opcode = 1 decimal |
| word 2 | Blanking | |
| word 3 | Initialize | |

Blanking: 0 = (parameter ignored)
1 = Blanking enabled

Initialize: 0 = 0000 = (parameter ignored)
1 = 0001 = Clear color plane 0
2 = 0010 = Clear color plane 1
3 = 0011 = Clear color plane 0 & 1
. . .
. . .
7 = 0111 = Clear color planes 0, 1, & 2
. . .
. . .
15 = 1111 = Clear all color planes 0, 1, 2, & 3

Ignored parameters imply that the parameter settings presently in effect will not change.

## END_FRAME, opcode 2

This command disables screen blanking as described in the *BEGIN_FRAME* command. Blanking is normally disabled.

The command format is described below.

END_FRAME Command

| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count = 1 | Opcode = 2 decimal |

## CONTROL_BLINKING, opcode 3

This command specifies the time interval and the colormap register that is loaded with the color sequence established by the *DEFINE_COLOR_TABLE* command. In addition, it is used to disable blinking for the specified colormap register.

In the default case, blinking is disabled. Before blinking can be enabled, a color sequence must be defined.

The blink interval parameter is defined in tenths of a second. If blink interval is zero, then blinking for the colormap register is disabled. If it is one, a new color is written into the register every tenth of a second. The blink interval is governed by the card's DUART, which generates internal card interrupts* each time its counter/timer reaches the time interval value. DUART interrupts must be enabled for blinking to occur. A blink interval of zero disables blinking but does not disable DUART interrupts.

DUART interrupts are enabled in the default case, and controlled by the *CONTROL_INTERRUPTS* command. The ability to control interrupts is important for synchronizing the blinking of registers, for example, when simulating motion or generating animation effects. Such synchronization is attained by first disabling interrupts, then sending the CONTROL_BLINKING command for each register to be blinked, and finally enabling interrupts. See the *CONTROL_INTERRUPTS* command.

The command format is described below.

CONTROL_BLINKING Command

| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count = 3 | Opcode = 3 decimal |
| word 2 | Register Number | |
| word 3 | Blink Interval | |

Register Number: 0 = Color register 0
1 = Color register 1
.
.
15 = Color register 15

Blink Interval: 0 to 32767 tenths of a second

*There are two types of interrupts associated with the video output card. One type is an I/O card level interrupt generated on the HP 1000 system backplane. Alternatively, the interrupt type mentioned here deals with internal interrupts to the resident microprocessor which controls internal card processes (see the *CONTROL_INTERRUPTS* command).

## CONTROL_INTERRUPTS, opcode 4

This command controls blinking of multiple colormap registers by enabling or disabling DUART interrupts. Interrupts are enabled in the default case.*

If a number of colormap registers have been configured to blink with the same time interval (using the *DEFINE_COLOR_TABLE* and *CONTROL_BLINKING* commands), the synchronization of blinking between the registers cannot be guaranteed. This may be undesirable for certain graphic requirements - for example, when simulating motion. Since blinking requires DUART interrupts enabled, control of interrupts can act as a master switch to simultaneously start all designated colormap registers.

A typical command sequence to synchronize blinking might be as follows:

1. Define a blink sequence for each register using the DEFINE_COLOR_TABLE command.
2. Disable UART interrupts using the CONTROL_INTERRUPTS command.
3. Enable blinking for each of the registers using the CONTROL_BLINKING command. Blinking will not yet occur because interrupts are disabled.
4. Enable DUART interrupts using the CONTROL_INTERRUPTS command.

The command format is described below.



CONTROL_INTERRUPTS Command

| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count = 2 | Opcode = 4 decimal |
| word 2 | Flag | |

Flag: 0 = Disable Interrupts
1 = Enable Interrupts

*There are two types of interrupts associated with the video output card. One type is an I/O card level interrupt generated on the HP 1000 system backplane. Alternatively, the interrupt type mentioned here deals with internal interrupts to the resident microprocessor which controls internal card processes.

**Inquiry Commands**

## INQUIRE_DEVICE_ID, opcode 5

With this command, the card sends standard device identification and configuration information to a status buffer in user RAM. The data is read using the Read Graphics Status Buffer video request.

Each time the command is issued, data is sent to the status buffer. Status buffer data is not concatenated; it is overwritten. To preclude the loss of data, the status buffer should be read each time the command is executed.

The command format is described below.

INQUIRE_DEVICE_ID Command
(System to Device)

| | High Byte | Low Byte |
|---|---|---|
| word 1 | Word Count = 1 | Opcode = 5 decimal |

Response to the command is provided by the card to the system in the following format.

INQUIRE_DEVICE_ID Response
(Device to System)

| | | |
|---|---|---|
| byte 0 | Word Count = 8 | |
| byte 1 | Response Opcode = 253 | |
| byte 2 | Command Opcode = 5 | |
| byte 3 | Firmware Rev. Code | |
| byte 4 | | |
| byte 5 | Part Number | |
| byte 6 | | |
| byte 7 | 12065A | |
| byte 8 | | |
| byte 9 | in ASCII | |
| byte 10 | Number of addressable | |
| byte 11 | X pixels | |
| byte 12 | Number of addressable | |
| byte 13 | Y pixels | |
| byte 14 | Display | |
| byte 15 | Status | |

Bytes 10 through 13 show the current screen dimensions configured

The Display Status code returned is described in Appendix A.

Display Status will reflect the display control switch settings (SW2) or the current software configuration implemented by the Write Card Configuration video card request.

# USING THE VIDEO OUTPUT CARD

With the driver properly configured, video card read and write requests are issued for various card level operations. One of these requests, the Write Graphics Commands request, is used to send graphics commands to the card. See Appendix C for a simple programming example.

Additional programming information and examples are provided on the following pages.

## Self-Tests and Card Status Requests

The card performs a self-test at power up or upon receipt of a Self-Test video card request from the host system. A light emitting diode (LED) on the card indicates self-test results. See Section 4.

Upon completion of self-test, instead of a visual verification of pass or fail, a card status request can be issued before any configuration or figure drawing is performed. (See the Read Card Status request.) Status request completion provides verification that the self-test completed.

The status read deasserts interrupts. Therefore, if interrupt routines are to be used, the Enable Card Interrupts request should be issued after the status read. See the paragraph RS-232 Port Considerations for an illustrative example.

## Using Interrupts

To read from the card, a two step programming process is required. First, an EXEC I/O Write (EXEC 2) is performed to send the video card read request to the card. Next, an EXEC I/O Read (EXEC 1) is issued to complete the read request. If interrupt routines are used, and an interrupt occurs between the EXEC I/O Write and Read calls, proper completion of the read request cannot be guaranteed unless appropriate precautions are taken.

Under these circumstances, proper read completion can be assured by providing exclusive access to an I/O device. For example, the LURQ (Logical Unit Lock) subroutine provides this service. See the paragraph RS-232 Port Considerations for an illustrative example.

In the special case of a Read Graphics Status Buffer request (subsequent to a graphics inquiry command), the availability and proper format of the information returned verifies that the read completed properly and was not interrupted.

If an interrupt handling routine is scheduled and running, a new interrupt may not be serviced since the routine is already scheduled. To help ensure each interrupt is individually serviced, it is necessary for the interrupt routine to deassert subsequent card interrupts until the routine has completed. When interrupts are reasserted, status bits (*overrun* and *record received* bits) are scanned to see if subsequent interrupts occurred. A set bit will cause the interrupt handling routine to be rescheduled. However, note that multiple interrupts occurring from the same source would attempt to set the same bit; the routine would be rescheduled only once and the bit is cleared.

The Read Card Status read request deasserts interrupts, while the Enable Card Interrupts write request enables interrupts.

For the interrupt handling routine to be scheduled, it must be restored (RP'd) under the system session, and enabled as the interrupt handling program.

## RS-232 Port Considerations

The video output card has two independent RS-232 ports to be used for interfacing with keypads, touchscreens, mice, or X-Y devices. Depending on the application, data input to a port can be read by the system for futher processing, or simply echoed to a video display device.

Each port is configured using the Write Card Configuration video card request. The ports are read from and written to using the Read RS-232 Port and Write RS-232 Port requests, respectively. Futhermore, the termination of a read is determined by a bit in the Read RS-232 Port request which may depend on the definition of a record or port blocking factor (established by the Write Card Configuration request).

The example on the following pages show the skeleton of a main program and an interrupt handling routine for keyboard inputs to an RS-232 port.

## Using RS-232 Ports

Main Program

| Fortran Code | Comments |
|---|---|
| FTN7X,L<br>PROGRAM MAIN<br>IMPLICIT INTEGER (a-z)<br>INTEGER rstat(0:9)<br>p = 44 | This is a main program. Assume the LU of the video card is 44 decimal. |
| call EXEC (3,4000B + p,100000B,36B) | Configure the driver if not configured at boot-up. |
| call EXEC (3,2000B + p,2HPO,2HRT,2HB ) | Program PORTB is enabled for scheduling in response to an asynchronous interrupt. (Although enabled, PORTB must be restored (RP'd) under the system session as an interrupt handling program before it can be scheduled.) |
| . | . |
| . | . |
| . | . |
| call LURQ (1B,p,1) | Lock the video card to this program. |
| call EXEC (2,p,331B,1)<br>call EXEC (1,p,rstat,10) | The EXEC 2 sends the Read Card Status request to the card, while the EXEC 1 triggers the read and provides the buffer for 10 words returned. Interrupts are deasserted. |
| call LURQ (0,p,1)<br>call EXEC (2,p,114B,1) | The video card is unlocked, and interrupts are enabled allowing the interrupt routine to be scheduled. |
| . | . |
| . | . |
| . | . |
| 9999 end | End of main program. |

Interrupt Handling Program

| Fortran Code | Comments |
|---|---|
| FTN7X,L<br>PROGRAM PORTB<br>IMPLICIT INTEGER (a-z)<br>INTEGER bufr(132)<br>INTEGER linef(4)<br>INTEGER rdef(2)<br>INTEGER conf(2)<br>INTEGER stat(0:9)<br>DATA rdef /3130B,6400B/<br>DATA conf /2530B,170B/<br>DATA len /4/<br>DATA linef /0,1407B,400B,12B/ | This program is a simple interrupt handling routine for inputs from a keyboard on the video card RS-232 port B. The routine is scheduled upon a carriage return character. This character returns the present position to the left margin. The interrupt routine reads characters input from port B, and causes a linefeed on the display. (Note: an overfilled buffer will also cause an interrupt. This is an error condition that can be handled by a separate routine.) |
| p = 44 | Assume the video card LU is 44 decimal. |
| call EXEC (2,p,rdef,2) | The Write Card Configuration request is sent to the card as rdef(1), specifying N = 6 (high byte of data word is a record termination character). rdef(2) puts the carriage return character into the high byte. |
| call EXEC (2,p,conf,2) | The Write Card Configuration request is sent to the card as conf(1), specifying N = 5 (port B configuration). conf(2) sets the baud rate to 9600. Video monitor screen echo and cursor echo are enabled. This implies that characters input on port B are immediately echoed to the monitor along with a cursor. |
| 10 continue | Return loop when the program is rescheduled after the EXEC (6,0,1) below. |
| call LURQ (1B,p,1) | The video card is locked to the interrupt routine until completion. |
| call EXEC (2,p,331B,1)<br>call EXEC (1,p,stat,10) | A Read Card Status request is made to clear the interrupt, and deassert subsequent card interrupts until this program completes. (If this was not an interrupt handling routine, the Enable Card Interrupt request should be issued next.) Ten words are returned to bufr. |
| if (IAND (stat(1),400B) .NE. 400B)<br>1 Goto 9000<br>if (IAND (stat(1),100000B) .EQ.<br>1 100000B) Goto 9910 | Complete record received is verified for Port B buffer. If not, the program transfers to the terminating sequence (statement 9000). If the interrupt was caused by Port B buffer overrun, an overrun error routine is employed at statement 9910. (continued) |

Interrupt Handling Program (continued)

| | | |
|---|---|---|
| | call EXEC (2,p,307B,1)<br>call EXEC (1,p,bufr,80) | The Read RS-232 Port request is used to read from port B buffer to the end of record. The end of record character (carriage return) was defined above. Note that 80 words are returned to bufr in unpacked format, i.e., characters in the low byte and 0 in the high byte. Thus 80 characters are returned. |
| | call EXEC (2,p,112B,1) | Since the data in the buffer has now been read, the Flush RS-232 request is sent clearing the input buffer of data. |
| | call EXEC (2,p,linef,len) | The Write Graphics Commands request (linef(1)) is used to send a linefeed character (linef(4)) and cursor flag (linef(3)) to the card, using the *OVERLAY_TEXT* primitive command (linef(2) indicates word count = 3 and opcode = 7 decimal). linef(2) flags a cursor at the current position. If linef(2) = 0, a cursor would not be visible on the display until echoed by a character input to port B (as previously configured). |
| 9000 | continue<br>call LURQ (0,p,1) | The video card is unlocked from this routine. |
| | call EXEC (2,p,114B,1)<br>call EXEC (6,0,1)<br>Goto 10 | The Enable Card Interrupts request is sent (114 octal). The program is then terminated in a dormant state. Upon a new interrupt, the program will begin execution at the next instruction after the EXEC 6 call. |
| 9910 | continue<br>.<br>.<br>Goto 10 | A separate routine can be used to handle interrupts due to a port buffer overrun. |
| 9999 | end | |

## Graphics/1000-II Considerations

The video output card is compatible with Graphics/1000-II. Device-independent Graphics Library (DGL) initialization calls (such as ZDINT) issued from Graphics/1000-II programs automatically configure the interface driver ID. 50.

DGL routines use a majority of card firmware capabilities; however, ther some card features that are not supported by DGL, such as scrolling and text direction. In a Graphics/1000-II program, these card features can still be accessed. The following methods might be used.

1. EXEC I/O (system) calls. EXEC calls can be made directly to the card using video requests and command formats described earlier in this manual. However, because such calls bypass the DGL I/O buffer, proper synchronization of commands cannot guaranteed.

2. ZOESC, opcode 9051 (DGL) calls. The ZOESC subroutine allows an application program to perform device dependent graphical output. ZOESC calls are placed in the DGL I/O buffer for execution in proper sequence. For detailed information on making ZOESC calls, refer to the *Graphics/1000-II DGL Programmer's Reference Manual.*

The use of ZOESC, opcode 9051, calls in a Graphics/1000-II program is shown in the example on the next page.

**Using ZOESC Calls**

| Fortran Code | Comments |
|---|---|
| FTN7X,L<br>PROGRAM SCROLL<br>IMPLICIT INTEGER (a-z)<br>INTEGER ilista(9),ilistb(9),ilistc(9)<br><br>DATA ilista /35,5,1,-1,0,0,0,0,0/<br>DATA ilistb /34,128,128,383,383,3,-1,0,0/<br>DATA ilistc / 8,128,128,383,383,-1,0,0,0/<br><br>p = 44 | Using the ZOESC call, this program will pass the DEFINE_SCROLLING_WINDOW command (opcode 34), the CHANGE_OVERLAY_MARGINS command (opcode 8), and the SCROLL primitive command (opcode 35) to the video card. The -1 values terminate each data list. Assume the LU of the card is 44. |
| call ZBEGN | Initialize the DGL system. |
| call ZDINT (p,0,ierr) | The video card is enabled as the graphic display device. |
| . . . | . . . |
| call ZOESC (9051,9,0,ilistb,0,ierr) | Define the scrolling window. Assuming a 512 x 512 pixel display, the scrolling window is a 256 x 256 pixel square centered within the display. |
| call ZOESC (9051,9,0,ilistc,0,ierr) | Make the alphanumeric overlay text margins to be the same as the scrolling window boundaries. |
| call ZOESC (9051,9,0,ilista,0,ierr) | Scroll the pixel rows in the scrolling window. The pixel rows are rolled up by 5 rows. |
| . . . | . . . |
| call ZDEND<br>call ZEND<br>9999 end | The video card is disabled, the DGL system is terminated, and the program ended. |

For the graphics commands passed to the card, note that word counts in the high byte of the first word are not specified. Although word counts must normally be provided when passing commands, Graphics/1000-II automatically provides the word count to the firmware.

# MAINTENANCE

SECTION 4

Assuming that a monitor is properly connected and powered up, turning on the power to the computer will start CPU and video card self-test operations. The red LED on the video card should light or blink during self-test, and turn off after self-test. For reasons explained below, this cycle could take up to 20 seconds. After the LED goes out, the monitor should display the pattern as configured by SW1-1 and SW1-2 (see Section 2 and the paragraphs below for a discussion on card switches).

There are no specific maintenance procedures for the video card. However, to help troubleshoot or ensure proper card operation, this section will cover the following:

1) the video card self-test (operation and interpretation)

2) additional tests of the video card's output

3) adjustment of the video monitor



## SELF-TEST

### Set-up

#### Switches

The mode of the self-test is determined by the settings of switches SW1-1 and SW1-2 (bits 7 and 6) on the Video Output Interface Card. Switch pack SW1 is located at the front of the card on the left side. See Figure 2-1. The mode definitions are:

| Mode | Description |
|---|---|
| 0 0 | Displays the Graphics/1000-II 15 default colors (including black) in vertical bands and the default highlight mode. |
| 0 1 | Displays 15 vertical bands in increasing levels of gray. Since the gray scale represents a uniform ramp of intensity data, this mode is useful for testing the video signal without the monitor, such as with an oscilloscope. Alternatively, this pattern is useful for testing and adjusting for proper color tracking. |
| 1 0 | Displays a grid pattern for adjusting monitor convergence, and horizontal and vertical linearity. Convergence refers to the quality of the blue, green, and red electron beams converging to the same point. |
| 1 1 | Repeats self-test continuously. This is useful for detecting intermittent hardware problems. Note that changing the switches during continuous self-test will automatically reconfigure the self-test mode to the new switch setting selected. |

### Test Hood

You can install a loopback test hood (part number 12065-67001) to get a full external loopback test of the RS-232-C ports during the video card self-test. If no test hood is installed, an internal loopback test is performed during self-test.

## Initiation

You can initiate self-test in three ways:

1. Turn your computer's power off and then back on again.

2. Push the reset switch on the computer's processor board.

3. From a program, write a video card request of 174 octal (initiate self-test) to the video card. (For details on how to make this request, refer to the Programming section of this manual.)

## Execution

Two different self-tests are used to test the operation of the video card:

1. The host CPU self-test tests the I/O Master section of all I/O cards in the computer's backplane, including that of the video card.

2. The video card self-test uses the on-board processor (MC 68008) to test:

   a. The ROMs: a CRC algorithm confirms the contents and accuracy of ROM contents.

   b. The processor RAM (the 8 K-bytes dedicated to the on-board processor): a procedure verifies that reads and writes can be performed on all RAM locations.

   c. Both RS-232-C ports at all baud rates: if no test hood is present, an internal loopback test is performed. If a test hood is present, an external loopback test is performed.

   d. The Graphics Display Controller (GDC) chip and the frame buffer RAM: a boundary check is performed on the GDC's vertical scan frequency to ensure it falls within an acceptable range. Using the GDC, patterns are written to and read from the frame buffer RAM.

   e. On-board interrupt circuitry: a DUART timer interrupt is generated which verifies the operation of on-board interrupts to the MC 68008.

   After the above tests have completed, the self-test firmware continues according to the mode setting of switches SW1-1 and SW1-2. For three of the modes (0 0, 0 1, and 1 0) the card outputs a test pattern. This pattern is maintained until some other command to the video card changes it. For the fourth mode (1 1) the card repeats self-test continuously.

   If the self-test fails, testing stops and the card goes into an idle state.

   Execution of the video card self-test takes less than 10 seconds.

If you initiate the self-test by cycling your computer's power off and on or by pressing the reset switch on the CPU card, the CPU self-test will execute once and the video card self-test will execute twice. This is due to the nature of the computer reset sequence. In this sequence, the computer hardware executes the CPU self-test and then causes each I/O card to execute its self-test before passing control to the operating system. When the operating system takes control, the first thing it does is to reset each card, which causes the card's self-test to execute again. Note that this double execution of the self-test takes approximately 20 seconds to complete.

If you initiate the self-test from software, only the video card self-test will execute, and it will execute only once.

## Interpretation

A light emitting diode (LED) on the video card indicates the status of the self-test. The LED sequence varies, depending on whether a test hood is installed on the RS-232-C connector. The sequences are:

If a test hood is not installed:

1) LED turns on: self-test starts.

2) LED turns off: self-test passes.

If the LED remains on continuously, the self-test failed. (Allow at least 20 seconds for normal self-test completion before you conclude that the test failed.)

If a test hood is installed:

1) LED turns on: self-test starts.

2) LED blinks: self-test has detected the test hood on the RS-232-C ports.

3) LED turns off: self-test passes.

If the self-test failed, the LED turns on and remains on continuously. Note that if a test hood was installed but was not detected (the LED didn't blink), this is also a failure.

If your video card does not pass self-test, contact your HP Customer Engineer.

When self-test fails the video card goes into an idle state and does not process further commands sent to it. There is, however, a way to get past this situation and continue operating; just issue the command

CN,<lu>,0

(where <lu> is the logical unit number of the video card in your system) from the system command prompt. This will cause a software card reset (not the type that starts self-test), and your card can continue to operate if the problem is not too severe. This may allow you to continue your application until you can get the card repaired. (Note, of course, that HP does not guarantee that your application will work if you are using a card that does not pass self-test.)

# ADDITIONAL TESTS

There are additional tests you can perform that go beyond the video card self-test. (Because there is no way to loop back the video output signal into the video card, the video card self-test does not test the video output stages of the card.) Most of these tests require an oscilloscope.

You don't need to get out your oscilloscope in order to determine whether you have problems that weren't caught by the video card self-test. Any such problems usually have readily visible symptoms; in the absence of those symptoms you can pretty safely assume that your card is working properly. Of course, if you want to be absolutely sure (or if you just like messing about with test equipment), follow the procedures listed below.

Note that sometimes a badly adjusted monitor can cause symptoms that are similar to those of a bad video card, so we provide an adjustment procedure for the standard HP monitor below (and you can generally use this procedure for most monitors). Note, however, that it's possible to find yourself in a chicken-and-egg situation: a badly adjusted monitor may prevent you from displaying the test patterns that you need to adjust the monitor correctly in order to see whether the video card is generating the proper test patterns. In such a situation a look with the oscilloscope can tell you whether the problem is in the card or the monitor.

## Symptoms

The typical symptoms you are likely to see are:

1) A completely black screen.

2) A "funny looking" test pattern generated at the end of the video card self-test. The correct test patterns are described below. If the pattern does not resemble these, then it is likely that either the card or or the monitor are at fault. Tests for the card are described below. Possible adjustments to the monitor are subsequently described.

   a) Mode 0 0: 16 vertical bands of color, with the rightmost band blinking from black to white. The colors are the default Graphics/1000-II colors; the first six colors should be black, white, red, green, yellow, and blue.

   b) Mode 0 1: 15 vertical bands of gray tones. Each band is a lighter shade than the band to its left, and the leftmost band is twice as wide as the other bands.

   c) Mode 1 0: a grid pattern composed of white lines enclosing black boxes. There should be 25 boxes in a 5 x 5 pattern. Each box should have the same aspect ratio as the screen (4:3 or 1:1).

## Tests

If you see the symptoms given above, you can make the following tests:

1) Cable test. Set the self-test switches for mode 0 0 and run the self-test; this will cause the colored bands to be displayed. Make sure that the cables are connected from the outputs on the video card to the corresponding inputs on the monitor. Mixed up cables will result in colors being out of order. An open circuit in one of the cables will cause that color to be missing. (For example, if red is missing,

you'll have plenty of blues, greens, and blacks, but no reds, yellows, or whites.) If the green cable has an open circuit, the picture will be black; the green signal carries the sync pulses, and without sync you will get no picture.

2) Oscilloscope test. You can use on oscilloscope to look at the signal that is being transmitted on each of the color channels. Set the self-test switches for mode 0 1 and run the self-test; this will cause the gray scale bands to be displayed. Set your oscilloscope to settings somewhere near the following:

Input impedance: 50 ohms
Main sweep: 0.01 milliseconds
Vertical scale: 0.5 volts per division
Triggering: negative edge

Connect the oscilloscope to each of the output signals in turn, using the standard 3-meter cable supplied with the card. The red and blue signals should look like figure 4-1; the green signal should look like figure 4-2. Note that the vertical scale may be different, and that the width of the blanking interval may vary due to the setting of switch SW2. In any event, the width of the first step should be twice that of all other steps; the vertical height of the color signal from blank to highest intensity should be 0.7 volts ±10% at 75 ohms (0.6 volts ±10% at 50 ohms); the height of the sync signal should be 0.28 volts ±10% at 75 ohms (0.25 volts ±10% at 50 ohms); and the sync signal should be approximately 3.02 microseconds wide for an aspect ratio of 4:3, or 2.27 microseconds wide for an aspect ratio of 1:1.

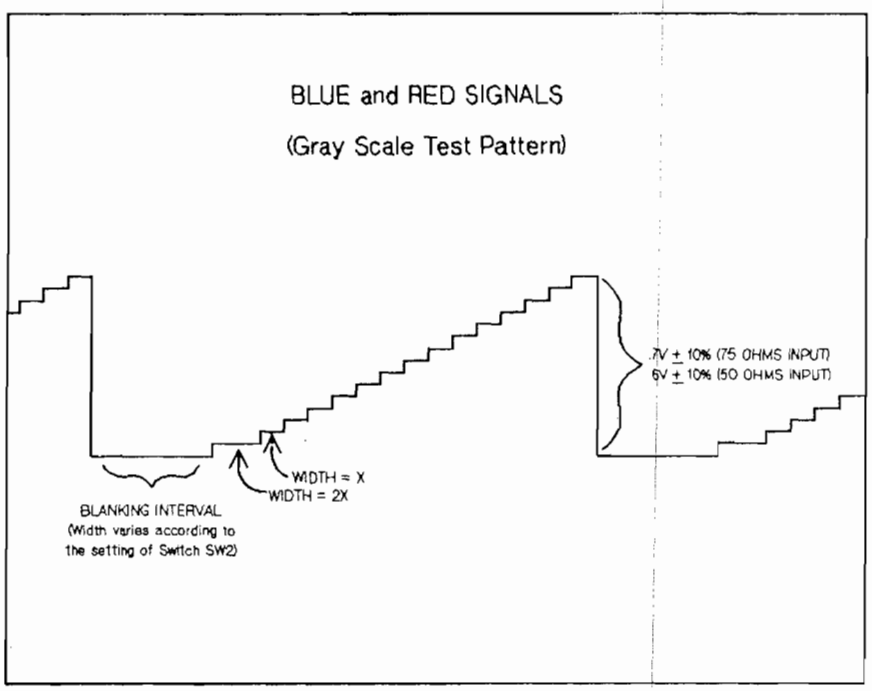



Figure 4-1. Red and blue test pattern signals.

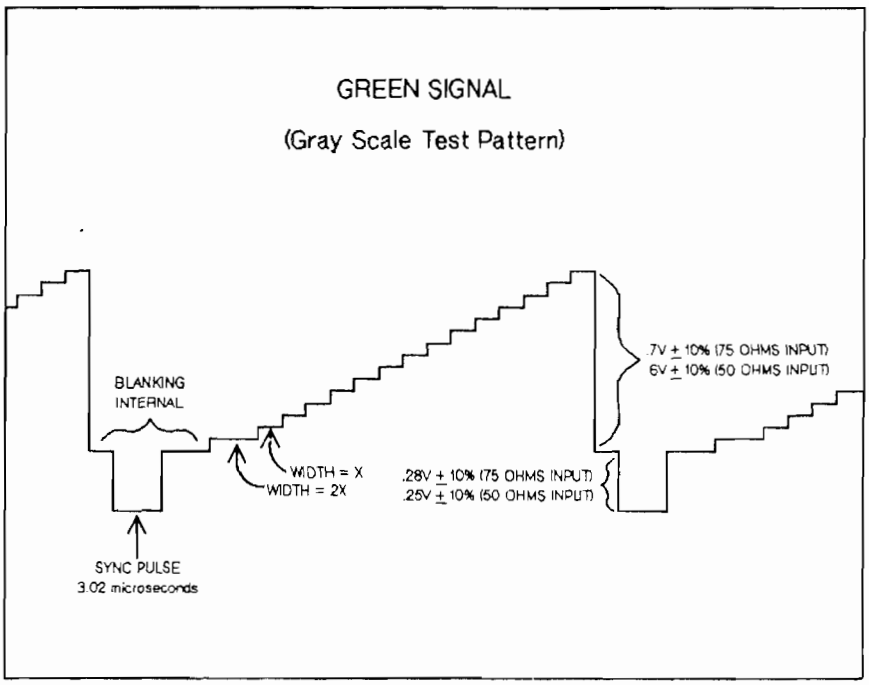



Figure 4-2. Green test pattern signal.

If you see signals like these on your oscilloscope, your video card and cable are OK. If you don't see signals like these, your video card or cable is bad; contact your HP Customer Engineer. If you see the right signals on your oscilloscope but the test patterns on your monitor don't look right, you may need to adjust your monitor. Read on.

## ADJUSTMENTS

Odd patterns appearing on the monitor may be monitor dependent. Because the video card uses composite signals, that is, horizontal and vertical sync are superimposed on the green signal line, the RS-343-A monitor you are using must support composite signals. Certain monitors may provide for selection of composite or non-composite signals. Conrac monitors, for example, have a jumper on an internal circuit board for such selection. Consult your monitor's manuals for configuring composite signal selection. The HP 13279B Color Monitor, discussed below, is preset for operation with composite signals.

The monitor you are using must support the horizontal sweep frequency configured on the video card. For example, the HP 13279B can be used with any frequency that the card can output. A course line jumper in the monitor (J11, described below) selects between a High, Medium, and Low horizontal scan frequency range. Further fine-tuning is accomplished via a potentiometer. Once the horizontal sweep frequency selection is made, a different potentiometer is adjusted for tuning the vertical scan frequency. Other adjustments are also described below.

The HP 13279B monitor is used with other Hewlett-Packard products, such as the HP 98627A and HP 97062A, color interfaces for the HP 9000 Series 200 and Series 500 computers, respectively. The HP 98627A is switch selectable for operating at 24.8 kHz or 15.75 kHz horizontal sweep frequencies. At the low frequency, the HP monitor's frequency select jumper (J11) would have to be placed in the "low" range. The HP 97062A can provide a 29.4 kHz horizontal sweep frequency, similar to the 12065A SW2 setting of 0000. Thus, J11 would be placed in the "medium" range on the monitor for both cards at that operating frequency. (Slight differences between the cards may require other minor adjustments.)

The adjustment procedure given here is for an HP 13279B monitor. If you are using some other monitor, you will probably be able to follow the same general procedure. You should consult the manufacturer's manuals for your monitor.

**WARNING**

**The HP 13279B monitor produces hazardous (potentially lethal) voltages. Adjustments to this monitor (or any other) should be attempted only by qualified service personnel.**

Take the following steps to adjust your monitor:

1) Set switch SW2 on the video card to match the characteristics of your monitor. (If you don't know the characteristics of your monitor, try 0000 or 0001 (depending on the aspect ratio of the screen) for starters and adjust from there. Appendix A has a complete list of the display parameters set by switch SW2.) The HP 13279B and similar monitors can be configured and used with most SW2 settings; however, for simplicity, the instructions below assume a setting of 0000 with the monitor coarse line frequency select jumper (J11) set for "Medium" as described below.

2) Set the switches on the front panel of your monitor. The SCREEN switches (R, G, and B) should all be out (on), and the CHANNEL switch should be out (channel A, color).

3) Set the self-test switches (SW1-1 and SW1-2) to mode 1 0 and run the self-test. This will send the grid test pattern to the monitor.

4) Turn off the power to the monitor. Unplug the power and signal cables from the back of the monitor.

5) Remove the screws from the bottom of the back panel of the monitor and from the sides of the monitor. Lift off the the cover by sliding it back and up. The top, back, and sides should come off as one unit.

6) Plug in the power and signal cables to the back of the monitor. **DO NOT TURN ON THE POWER YET!**

7) Check to make sure that the termination switches at the back of the monitor are set to the correct impedance. (If your monitor is the last or only one connected to the video card, the termination switches should be set to 75 ohms; if you have multiple monitors daisy-chained together the termination switches for all but the last one should be set to "high".)

8) The next several adjustments are made on the printed circuit board on one side of the monitor, the right side as you face the back of the monitor, or left side as you face the monitor screen. Figure 4-3 shows the location of each of the adjustments on this board. **The power should be off.**

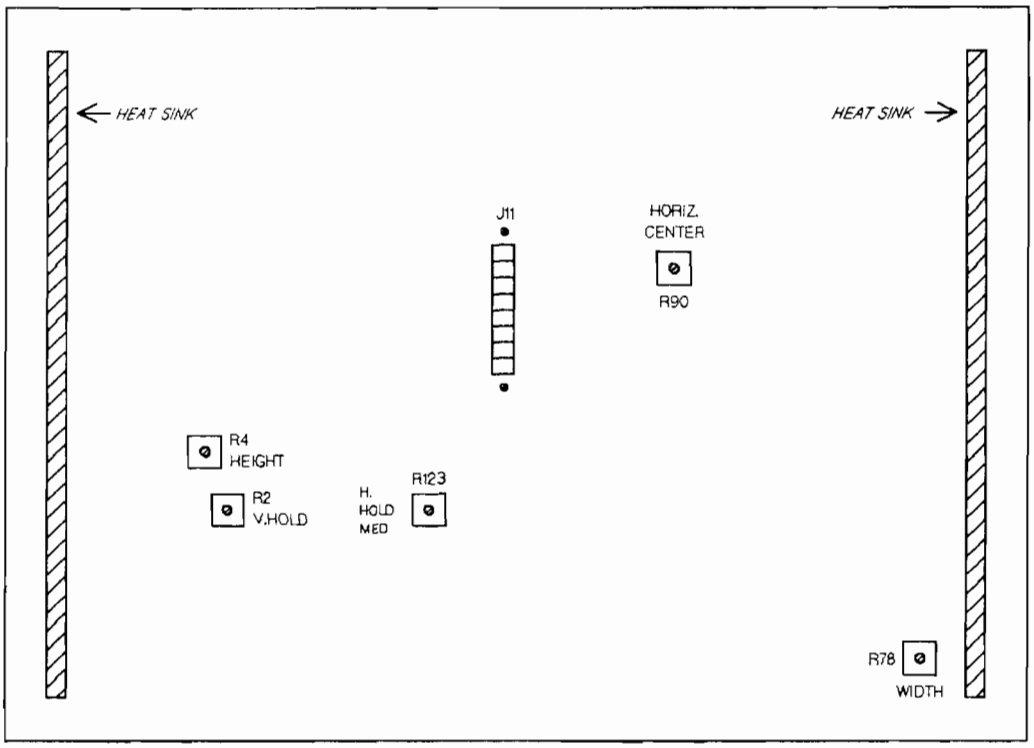



Figure 4-3. Location of Adjustments

a) Make sure that the coarse line adjustment jumper (jumper J11) is in the center position (one pin exposed on each side of the nylon connector). This jumper adjusts for the course horizontal sweep frequency range (Medium) set by SW2 (0000 = 29.4 kHz).

b) Turn the power to the monitor on. The grid test pattern will appear on the screen, although it is likely to have a "scrambled" appearance.

c) Adjust the H. HOLD MED potentiometer (R123) until you see steady vertical lines on the screen. (If this potentiometer is adjusted improperly, you will see dots all over the screen.) This adjustment is for fine-tuning of the horizontal sweep frequency.

d) Adjust the V. HOLD potentiometer (R2) until the grid pattern appears. This adjustment is for the vertical scan frequency.

e) The next three potentiometers must be adjusted together until the desired result is obtained. The potentiometers are: HORIZ. CENTER (R90), H. HOLD MED (R123), and WIDTH (R78). Adjust them until all six vertical lines of the grid pattern are visible and centered, and fill the screen.

f) Adjust the HEIGHT potentiometer (R4) to obtain the proper aspect ratio. Each grid should be in the ratio of 4:3 (horizontal:vertical) or 1:1, depending on the setting of switch SW2.

9) Turn off the power to the monitor. Unplug the power and signal cables.

10) Replace the cover; fasten it securely with the screws. (If the screws are not tightened down properly, the monitor will no longer meet RFI specifications.)

11) Connect the power and signal cables. Turn on the power to the monitor.

12) Change the self-test mode (SW1-7 and SW1-6) to 0 1 and run the self-test. This will display the gray scale test pattern on the monitor.

13) The next couple of adjustments involve the BRIGHT and CONT (brightness and contrast) potentiometers on the front panel of the monitor.

a) Push in the BRIGHT and CONT buttons to put those controls into preset mode. You can now adjust the PRESET screws for those controls.

b) Adjust the preset screws for BRIGHT and CONT until you see 15 vertical bands of gray. Each band should be progressively lighter than the band to its left. The leftmost band should be twice as wide as the other bands, and should be the same color as the black background of the screen.

14) Change the self-test mode to 0 0 and run the self-test. The test pattern of vertical color bands will apprear on the screen.

15) Check to see that the leftmost six bands are black, white, red, green, yellow, and blue. (If they are in the wrong order, change around the signal cables until they appear in the right order.) You can make minor adjustments to the color intensities using the LOW LIGHTS potentiometers (R, G, and B) on the front panel.

16) Change the self-test mode to 0 1 and run the self-test. The gray scale pattern will appear on the screen. Check to make sure that the bars are black, white, and gray, and that they are not tinted with color. If necessary, make adjustments with the LOW LIGHTS potentiometers until the bars are all black, white, or gray.

# DISPLAY PARAMETERS

APPENDIX
A

The monitor drive signal and display format are configured by the hardware switch SW2, or in software using the Write Card Configuration video request. There are sixteen possible configurations which are shown by the display codes below.

| Display code | Resolution (h: horizontal) (v: vertical) | Horizontal Sweep Frequency | Vertical Scan Frequency |
|---|---|---|---|
| *0000* | 576h x 455v | 29.4kHz | 60.0 Hz |
| *0001* | 512h x 512v | 31.5 kHz | 57.5 Hz |
| *0010* | 576h x 455v | 28.1 kHz | 57.4 Hz |
| *0011* | 512h x 512v | 30.8 kHz | 56.1 Hz |
| *0100* | 576h x 455v | 27.0 kHz | 55.1 Hz |
| *0101* | 512h x 512v | 29.4 kHz | 53.6 Hz |
| *0110* | 576h x 455v | 25.9 kHz | 52.9 Hz |
| *0111* | 512h x 512v | 28.1 kHz | 51.4 Hz |
| *1000* | 576h x 455v | 25.0 kHz | 50.9 Hz |
| *1001* | 512h x 512v | 27.0 kHz | 49.3 Hz |
| *1010* | 576h x 455v | 24.0 kHz | 49.1 Hz |
| *1011* | 512h x 512v | 25.9 kHz | 47.3 Hz |
| *1100* | 576h x 455v | 23.2 kHz | 47.4 Hz |
| *1101* | 512h x 512v | 25.0 kHz | 45.5 Hz |
| *1110* | 576h x 455v | 22.4 kHz | 45.7 Hz |
| *1111* | 512h x 512v | 24.0 kHz | 43.9 Hz |

**Display Parameters (Continued)**

| Display code | Horizontal Front Porch | Horizontal Synch Pulse | Horizontal Back Porch |
|---|---|---|---|
| *0000* | 0.51 microsec | 3.02 microsec | 3.27 microsec |
| *0001* | 0.51 microsec | 3.02 microsec | 4.02 microsec |
| *0010* | 0.51 microsec | 3.02 microsec | 4.78 microsec |
| *0011* | 0.51 microsec | 3.02 microsec | 4.78 microsec |
| *0100* | 1.27 microsec | 3.02 microsec | 5.54 microsec |
| *0101* | 1.27 microsec | 3.02 microsec | 5.54 microsec |
| *0110* | 2.02 microsec | 3.02 microsec | 6.29 microsec |
| *0111* | 2.02 microsec | 3.02 microsec | 6.29 microsec |
| *1000* | 2.78 microsec | 3.02 microsec | 7.05 microsec |
| *1001* | 2.78 microsec | 3.02 microsec | 7.05 microsec |
| *1010* | 3.54 microsec | 3.02 microsec | 7.80 microsec |
| *1011* | 3.54 microsec | 3.02 microsec | 7.80 microsec |
| *1100* | 4.29 microsec | 3.02 microsec | 8.56 microsec |
| *1101* | 4.29 microsec | 3.02 microsec | 8.56 microsec |
| *1110* | 5.05 microsec | 3.02 microsec | 9.31 microsec |
| *1111* | 5.05 microsec | 3.02 microsec | 9.31 microsec |

**Display Parameters (Continued)**

| Display code | Vertical Front Porch | Vertical Synch Pulse | Vertical Back Porch |
|---|---|---|---|
| *0000* | 238 microsec | 136 microsec | 816 microsec |
| *0001* | 222 microsec | 127 microsec | 794 microsec |
| *0010* | 249 microsec | 142 microsec | 853 microsec |
| *0011* | 228 microsec | 130 microsec | 813 microsec |
| *0100* | 259 microsec | 148 microsec | 889 microsec |
| *0101* | 238 microsec | 136 microsec | 850 microsec |
| *0110* | 270 microsec | 154 microsec | 925 microsec |
| *0111* | 249 microsec | 142 microsec | 888 microsec |
| *1000* | 280 microsec | 160 microsec | 961 microsec |
| *1001* | 259 microsec | 148 microsec | 926 microsec |
| *1010* | 291 microsec | 166 microsec | 998 microsec |
| *1011* | 270 microsec | 154 microsec | 964 microsec |
| *1100* | 302 microsec | 172 microsec | 1034 microsec |
| *1101* | 280 microsec | 160 microsec | 1002 microsec |
| *1110* | 312 microsec | 178 microsec | 1070 microsec |
| *1111* | 291 microsec | 166 microsec | 1040 microsec |

**Display Parameters (Continued)**

| Display code | Horizontal Retrace Interval | Vertical Retrace Interval | Aspect Ratio |
|---|---|---|---|
| *0000* | 6.80 microsec | 1191 microsec | 4:3 |
| *0001* | 7.55 microsec | 1143 microsec | 1:1 |
| *0010* | 8.31 microsec | 1243 microsec | 4:3 |
| *0011* | 8.31 microsec | 1170 microsec | 1:1 |
| *0100* | 9.83 microsec | 1296 microsec | 4:3 |
| *0101* | 9.83 microsec | 1224 microsec | 1:1 |
| *0110* | 11.34 microsec | 1349 microsec | 4:3 |
| *0111* | 11.34 microsec | 1279 microsec | 1:1 |
| *1000* | 12.8 microsec | 1402 microsec | 4:3 |
| *1001* | 12.8 microsec | 1333 microsec | 1:1 |
| *1010* | 14.4 microsec | 1455 microsec | 4:3 |
| *1011* | 14.4 microsec | 1388 microsec | 1:1 |
| *1100* | 15.9 microsec | 1508 microsec | 4:3 |
| *1101* | 15.9 microsec | 1442 microsec | 1:1 |
| *1110* | 17.4 microsec | 1561 microsec | 4:3 |
| *1111* | 17.4 microsec | 1497 microsec | 1:1 |

# VARIATIONS FROM THE RS-343-A SPECIFICATION

APPENDIX B

The HP 12065A Video Output Interface conforms closely to the EIA RS-343-A standard. It can be used in almost all applications that require RS-343-A output. The output from the video card departs from the standard in the following areas:

1) The horizontal sync pulse width is 3.02 microseconds. The standard is 2.75 microseconds ±0.25 microseconds.

2) The horizontal information supplied during the vertical sync is at the horizontal frequency. The standard calls for horizontal information during vertical sync to be supplied at twice the horizontal frequency.

3) The video card display signal amplitude is 0.714 volts ±0.2 volts. The standard is 0.714 volts ±0.1 volts.

4) The video card sync signal amplitude is 0.286 volts ±0.08 volts, with a sync-to-total signal ratio of 26.8% ±10%. The standard calls for a sync signal amplitude of 0.286 volts ± 0.05 volts, with a sync-to-total signal ratio of 26.8% ± 5%.

The above variations represent the specifications for which, in our opinion, strict compliance is unnecessary for proper operation of most monitors. The variations from the specification result from circuit complexity-versus-cost trade-off decisions made during card design. Because of the rigidity of the specification, most monitors provide the flexibility to operate not only with devices which meet the specification, but also with devices that operate out of range within reasonable limits.

# A SIMPLE PROGRAMMING EXAMPLE

APPENDIX C

This appendix provides a simple FORTRAN programming example which provides some insights into passing video card requests and graphics commands to the card. The program is listed on the next page and should be referred to in the following discussion.

The first two EXEC calls are interface driver control requests to properly configure the driver (ID. 50, Rev. 2401 or later) and to perform a card reset. Refer to the *RTE-A Driver Reference Manual* for additional information.

The final EXEC call is an I/O write request to the card. It contains a single video card request (the Write Graphics Command request) and three graphics commands.

The first word, bufr(1), of the I/O EXEC call data buffer identifies the request. In this case, it is the Write Graphics Commands request. Since there are additional words in the data buffer (as described below), the buffer length must be specified to include all the words in the buffer.

The Write Graphics Commands request is accompanied by three graphics commands, identified by bufr(2), bufr(3), and bufr(12), respectively. As required by command format, the *ishft* function places each command's word count into the high byte of the word; the low byte contains the command's identifying opcode.

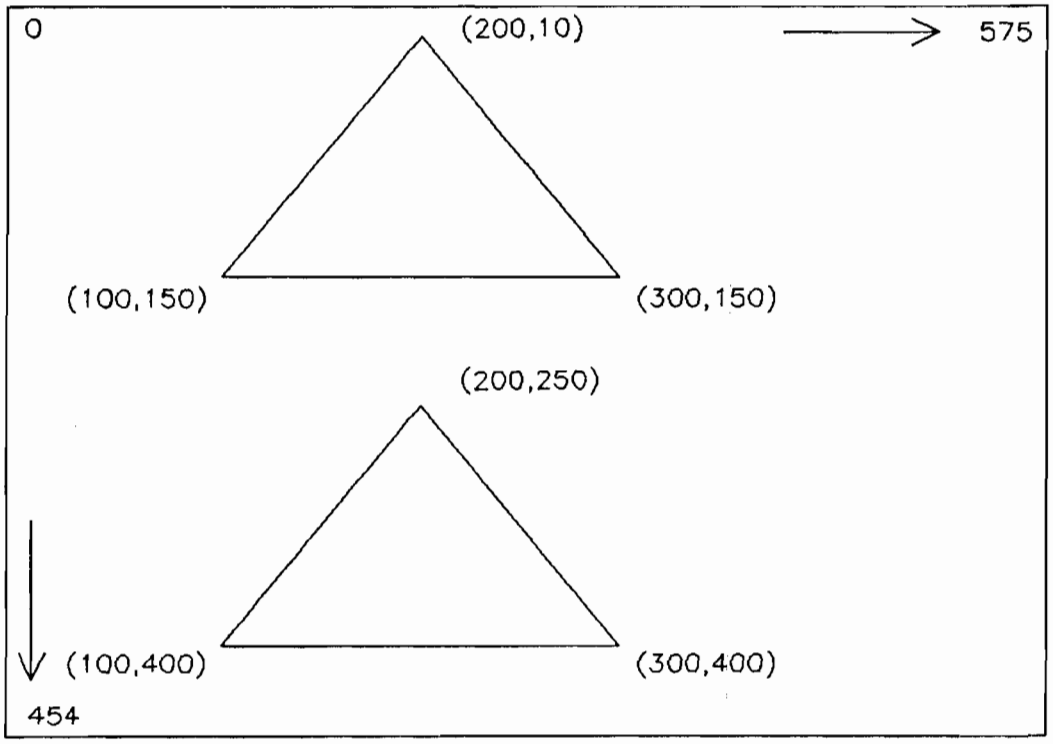
Note that the commands identified by bufr(3) and bufr(12) are different commands, but both are capable of drawing triangles. The words following each of these commands are command parameters.

**FORTRAN Program Example**

| Fortran Code | Comments |
|---|---|
| FTN7X,L<br>PROGRAM LINES<br>IMPLICIT INTEGER (a-z)<br>INTEGER bufr(20)<br>p = 44 | This program will draw 2 triangles using the *POLYLINE* and *POLYGON* graphics commands. Assume the LU of the video card is 44 decimal. |
| call EXEC (3,4000B + p,100000B,36B) | Driver control request to configure the driver (ID. 50). |
| call EXEC (3,0000B + p) | Driver control request to clear and reset the card. |
| bufr(1) = 0<br>bufr(2) = ishft(1,8) + 10 | bufr(1) defines the Write Graphics Commands video card request. It must be the first word of the I/O data buffer. bufr(2) is the *DEFAULT_ALL_ATTRIBUTES* graphics command. |
| bufr(3) = ishft(9,8) + 25<br>bufr(4) = 200<br>bufr(5) = 10<br>bufr(6) = 300<br>bufr(7) = 150<br>bufr(8) = 100<br>bufr(9) = 150<br>bufr(10) = 200<br>bufr(11) = 10 | bufr(3) specifies the *POLYLINE* graphics command for drawing lines and the number of words in this command. To draw the first triangle, bufr(4) and bufr(5) through bufr(10) and bufr(11) define the x,y coordinates of the points to which the *POLYLINE* command will draw lines. |
| bufr(12) = ishft(7,8) + 26<br>bufr(13) = 200<br>bufr(14) = 250<br>bufr(15) = 300<br>bufr(16) = 400<br>bufr(17) = 100<br>bufr(18) = 400 | bufr(12) specifies the *POLYGON* graphics command and the number of words in this command. bufr(13) through bufr(18) define the polygon (triangle) coordinates. |
| call EXEC (2,p,bufr,18) | The Write Graphics Commands request and 3 graphics commands are written to the card. |
| 9999 end | |

Note that the three graphics commands were concatenated using the same video card request and buffer. The commands could have been sent in separate buffers, each containing the Write Graphics Commands request as the first word.

When the program is run, the following image will be displayed on a monitor. The triangles will be white on black background. The numbers are shown for illustration only, and will not appear.



# PERFORMANCE

APPENDIX D

Performance data graphs are provided on the following pages.

**Blanked Screens.** Reading from and writing to the frame buffer occur only when the screen is blanked. During normal operation, this occurs during horizontal and vertical retrace intervals. If screens are blanked continuously, reading from and writing to the frame buffer can occur much more quickly (flash-filling).

Graph 1 shows vector performance with blanked screens. For connected vectors, note that there is little performance difference when using Graphics/1000-II compared to direct system I/O EXEC calls. For unconnected vectors, there appears to be significant performance differences. However, performance values for unconnected vectors reflect only the period that vectors are actually being drawn. The *moves* are not counted. If these invisible vectors were counted, vector performance would be much less.

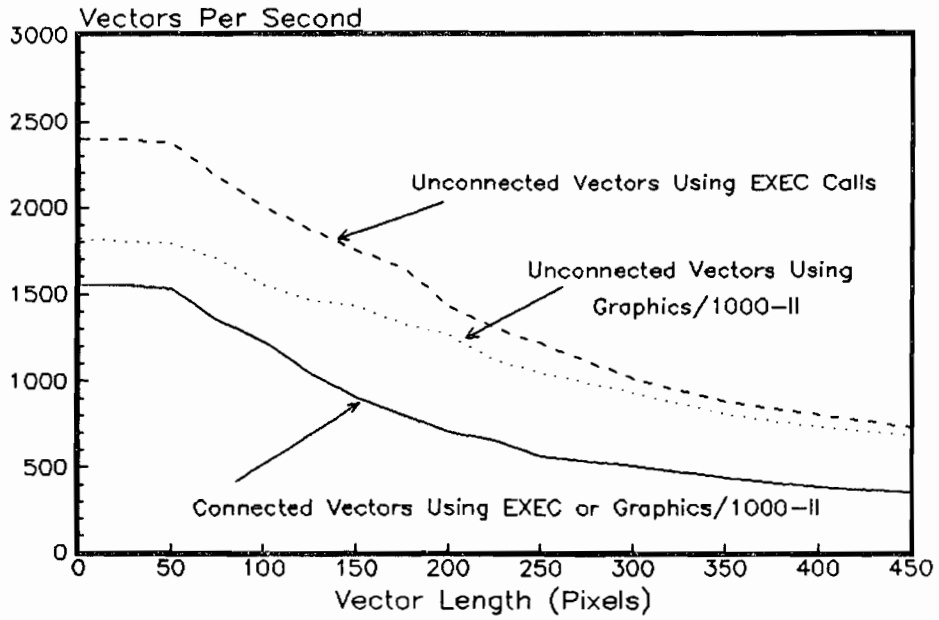
**Refresh Rate.** During normal operation (screen unblanked), the refresh rate plays a role in vector and character performance. The higher the frequency, the less time is available (during horizontal and vertical retrace) for frame buffer reads and writes. Graphs 2, 3, and 4 show that vector and character performance are higher at 50Hz than at 60Hz.

For characters, note that outputting alphanumeric overlay text is much faster than any of the graphics text capabilities. Alphanumeric overlay text is designed for quick text output, such as for status or error messages.

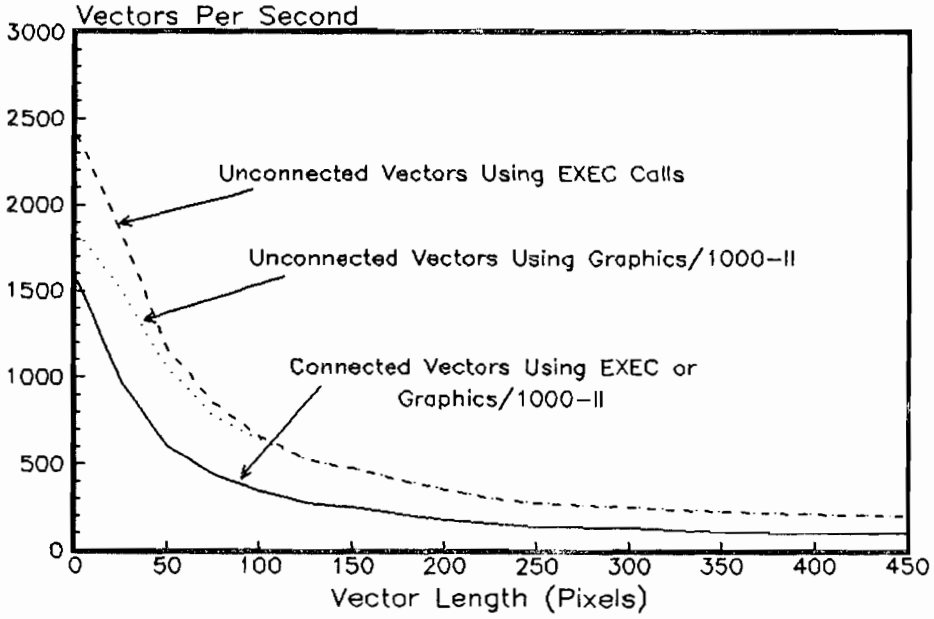
**System Buffering.** Graphics commands are transferred to the card only as fast as the card can process each command. System buffering allows a program to continue operation prior to fully completing I/O requests by buffering the requests in the system available memory (SAM). Hence, system buffering provides a performance advantage for computation-intensive programs, especially in low-end HP 1000 A-Series systems.

However, system buffering is not recommended for certain applications, such as cursor tracking with locator devices. In such cases, the speed of input combined with computation time and output delays will result in uncoordinated display images.

## Graph 1: Vector Performance
## Screen Blanked

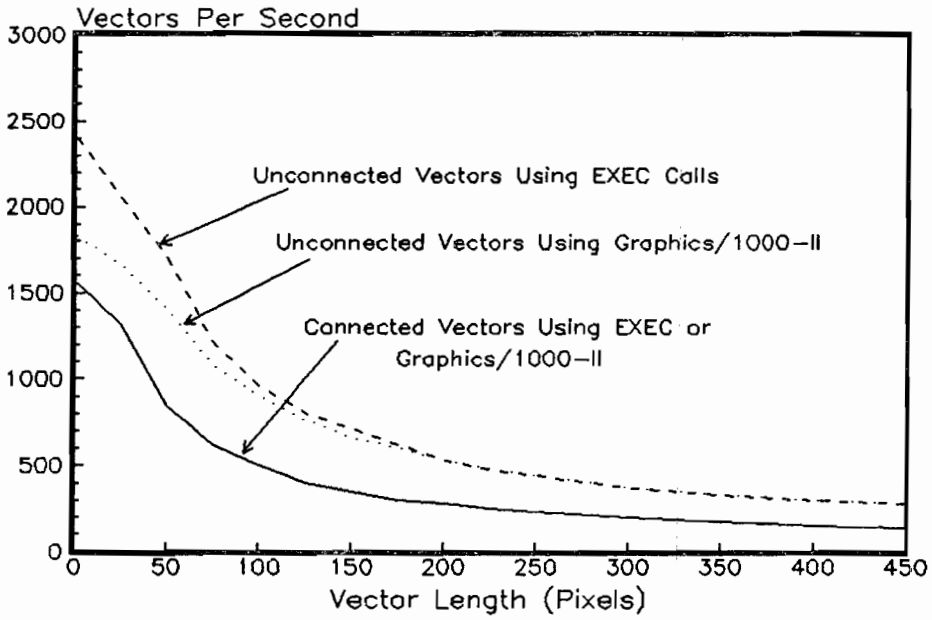



## Graph 2: Vector Performance
## 60 Hertz Refresh Rate



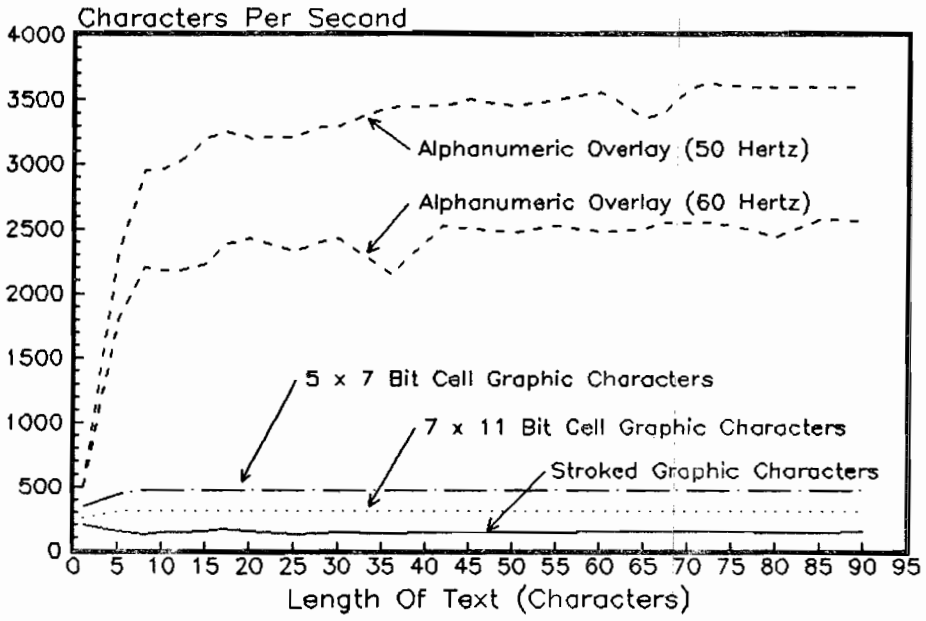
## Graph 3: Vector Performance

## 50 Hertz Refresh Rate




## Graph 4: Character Performance

## Alphanumeric Overlay and Graphic Characters



# REQUESTS AND COMMANDS QUICK REFERENCE

APPENDIX E

On the following pages are quick reference summaries of video card requests and graphics commands.

## Video Card Requests


### WRITE REQUESTS

SELF-TEST

| -- 0 -- | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |

WRITE CARD CONFIGURATION

| -- 0 -- | N | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |

N = 0: Set Display Parameters

| -- 0 -- | display code |

N = 1: Frame Buffer Blocking Factor

| 0 | Number of Data Words per Transfer |

N = 2: Port A Configuration

| -- 0 -- | stb | ech | baud | sec | se | bpc | e/o | par |

N = 3: Definition of a Record (Port A)

| High Byte | Low Byte |

N = 4: Port A Blocking Factor

| -- 0 -- | Blocking Factor |

N = 5: Port B Configuration

| -- 0 -- | stb | ech | baud | sec | se | bpc | e/o | par |

N = 6: Definition of a Record (Port B)

| High Byte | Low Byte |

N = 7: Port B Blocking Factor

| -- 0 -- | Blocking Factor |

WRITE RS-232 PORT

| -- 0 -- | 0 | 1 | 0 | 0 | 0 | A/B | 0 | 0 |

FLUSH RS-232 FIFO BUFFER

| -- 0 -- | 0 | 1 | 0 | 0 | 1 | 0 | A/B | 0 |

ENABLE CARD INTERRUPTS

| -- 0 -- | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |

WRITE SOFT CHARACTER SET

| -- 0 -- | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

WRITE RAW GDC CHIP COMMANDS

| -- 0 -- | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |

WRITE FRAME BUFFER

| -- 0 -- | plane | 0 | 1 | 0 | 1 | 1 | 1 | CNT | 0 |

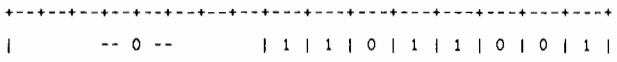
WRITE GRAPHICS COMMANDS

| -- 0 -- | -- 0 -- |

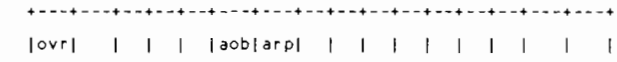
# Video Card Requests (Continued)

## READ REQUESTS

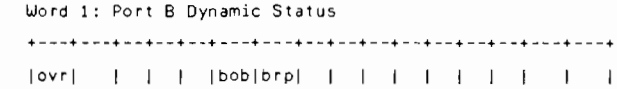
### READ CARD STATUS



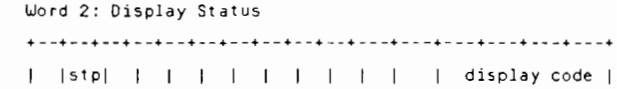

Word 0: Port A Dynamic Status



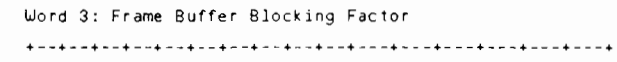

Word 1: Port B Dynamic Status



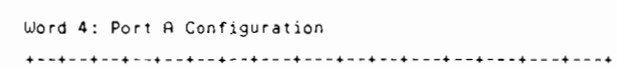

Word 2: Display Status



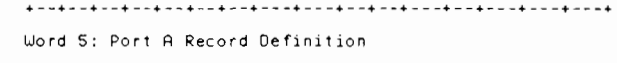

Word 3: Frame Buffer Blocking Factor



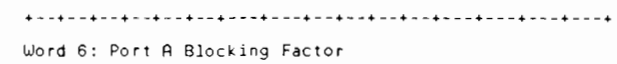

Word 4: Port A Configuration



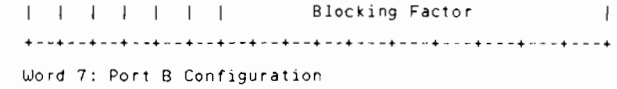

Word 5: Port A Record Definition



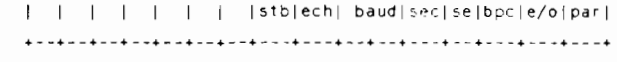

Word 6: Port A Blocking Factor



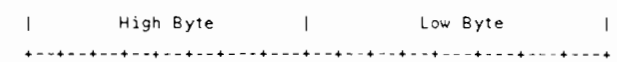

Word 7: Port B Configuration



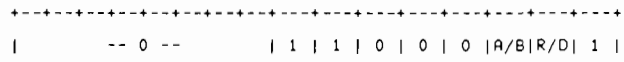

Word 8: Port B Record Definition



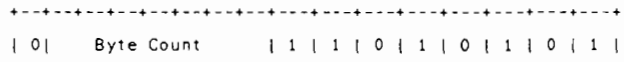

Word 9: Port B Blocking Factor



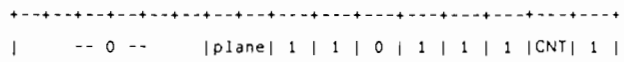

### READ RS-232 PORT



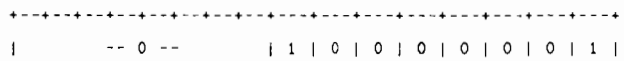

### READ RAW GDC CHIP DATA




### READ FRAME BUFFER




### READ GRAPHICS STATUS BUFFER



# Graphics Commands

## Output Primitives

### POLYLINE

| | | |
|---|---|---|
| word 1 | Word Count = 2N + 3 | Opcode = 25 decimal |
| word 2 | Start Position coordinate X | |
| word 3 | Start Position coordinate Y | |
| word 4 | End Point X1 | |
| word 5 | End Point Y1 | |
| . | . | |
| . | . | |
| . | . | |
| word *n* | End Point XN | |
| word n+1 | End Point YN | |

N = Number of lines to be drawn.
If there are three coordinate pairs given, two lines will be drawn; X,Y to X1,Y1 and X1,Y1 to X2,Y2.

### POLYGON

| | | |
|---|---|---|
| word 1 | Word Count = 2N + 3 | Opcode = 26 decimal |
| word 2 | Start Position coordinate X | |
| word 3 | Start Position coordinate Y | |
| word 4 | End Point X1 | |
| word 5 | End Point Y1 | |
| . | . | |
| . | . | |
| . | . | |
| word n | End Point XN | |
| word n+1 | End Point YN | |

N = Number of lines to be drawn, minus one.
For example, for five boundary lines to be drawn, N = 4.

### POLYMARKER

| | | |
|---|---|---|
| word 1 | Word Count = 2N + 1 | Opcode = 27 decimal |
| word 2 | Coordinate X1 | |
| word 3 | Coordinate Y1 | |
| word 4 | Coordinate X2 | |
| word 5 | Coordinate Y2 | |
| . | . | |
| . | . | |
| . | . | |
| word n | Coordinate XN | |
| word n+1 | Coordinate YN | |

N = Number of positions to be marked.

### GRAPHICS_TEXT

| | | | |
|---|---|---|---|
| word 1 | Word Count =(N/2)+ 3 | Opcode = 28 decimal | |
| word 2 | Start Position coordinate X | | |
| word 3 | Start Position coordinate Y | | |
| word 4 | Character 1 | Character 2 | |
| word 5 | Character 3 | Character 4 | |
| . | . | . | |
| . | . | . | |
| . | . | . | |
| word n | Character N-1 | Character N | |

N = Number of characters
(N must be even; character list can end with a null character)

## Primitives (Continued)

### OVERLAY_TEXT

| | | |
|---|---|---|
| word 1 | Word Count =(N/2)+ 2 | Opcode = 7 decimal |
| word 2 | Cursor Flag | Null/Optional Char |
| word 3 | Character 1 | Character 2 |
| word 4 | Character 3 | Character 4 |
| . | . | |
| . | . | |
| . | . | |
| word n | Character N - 1 | Character N |

N = Number of characters (N must be even; character list can end with a null character)

The low byte of word 2 can be the null character or an optional character

### CURSOR_UPDATE

| | | |
|---|---|---|
| word 1 | Word Count = 4 | Opcode = 29 decimal |
| word 2 | Cursor Type | |
| word 3 | Position coordinate X | |
| word 4 | Position coordinate Y | |

Cursor Type: 0 = full screen cross
1 = digitizing cross
2 = rubber band box
3 = rubber band line

### CURSOR_OFF

| | | |
|---|---|---|
| word 1 | Word Count = 2 | Opcode = 30 decimal |
| word 2 | Cursor Type | |

Cursor Type: 0 = full screen cross
1 = digitizing cross
2 = rubber band box
3 = rubber band line
16 = all cursors

### RECTANGLE_FILL

| | | |
|---|---|---|
| word 1 | Word Count = 5 | Opcode = 32 decimal |
| word 2 | Minimum X coordinate border | |
| word 3 | Minimum Y coordinate border | |
| word 4 | Maximum X coordinate border | |
| word 5 | Maximum Y coordinate border | |

### CIRCLE_FILL

| | | |
|---|---|---|
| word 1 | Word Count = 4 | Opcode = 33 decimal |
| word 2 | Center coordinate X | |
| word 3 | Center coordinate Y | |
| word 4 | Circle Radius | |

### SCROLL

| | | |
|---|---|---|
| word 1 | Word Count = 3 | Opcode = 35 decimal |
| word 2 | Number of Pixel Rows | |
| word 3 | Direction | |

Direction: 1 = up
2 = down (default)

## Output Attributes

### DEFAULT_ALL_ATTRIBUTES

| | | |
|---|---|---|
| word 1 | Word Count = 1 | Opcode = 10 decimal |

### DEFINE_COLOR_TABLE Command

| | | |
|---|---|---|
| word 1 | Word Count = 3 + N | Opcode = 11 decimal |
| word 2 | Starting Register | |
| word 3 | Operation Type | |
| word 4 | Entry 1 | |
| . | . | |
| . | . | |
| . | . | |
| word n | Entry N | |

Starting Register: register number 0 through 15

Operation Type: 1 = set colormap registers, N can be 16 maximum
2 = define blinking sequence

Entry: Each color is defined by 12 bits, 4 bits per red, green, and blue channel. For a particular channel, 0 is full off and 15 is full on (intensity level). The format for each entry is shown below; the MSB for each channel is to the left.

| 0000 | bbbb | gggg | rrrr |
|---|---|---|---|
| 0 | B | G | R |

### COLOR_INDEX

| | | |
|---|---|---|
| word 1 | Word Count = 3 or 4 | Opcode = 12 decimal |
| word 2 | Attribute Identification | |
| word 3 | Color Entry | |
| word 4 | Operation Type | |

Attribute Identification:

0 = all (except background and cursors)
1 = lines only
2 = polygon fill
3 = lines, graphics text, markers
4 = graphics text only
5 = marker symbols only
6 = cursor type 0 (full screen cross)
7 = cursor type 1 (digitizing cross)
8 = cursor type 2 (rubber band box)
9 = cursor type 3 (rubber band line)

Color Entry:

0 to 15 if Operation Type is 1, 2, or 4
0 to 7 if Operation Type is 3 (default)

Operation Type (optional):

1 = Complementary. A set bit in the present color entry will cause the complement of the corresponding bits in the previous color index. (Exclusive OR)
2 = Non-Dominant. The bits in the present color entry are OR'd with the bits in the previous color index.
3 = Dominant 3-plane. The bits in the present color entry overrides the previous index, EXCEPT for the fourth bit, or MSB, which is ignored. (Default)
4 = Dominant 4-plane. The bits in the present color entry overrides the previous index.

## Attributes (Continued)

### LINESTYLE

| | | |
|---|---|---|
| word 1 | Word Count = 2 or 3 | Opcode = 13 decimal |
| word 2 | Linestyle | |
| word 3 | Length | |

### LINEWIDTH

| | | |
|---|---|---|
| word 1 | Word Count = 2 | Opcode = 14 decimal |
| word 2 | Linewidth | |

Linewidth: Specified in odd multiples of the thinnest line, as follows

| Specified Width | Actual Width in Pixels |
|---|---|
| 0 | 1 |
| 1 | 1 (default) |
| 2 | 3 |
| 3 | 5 |
| 4 | 7 |
| . | . |
| n | (2n) - 1 |
| . | . |
| 175 | 349 |

### POLYGON_FILL_STYLE

| | | |
|---|---|---|
| word 1 | Word Count = 2 | Opcode = 15 decimal |
| word 2 | Style | |

Style:
0 = no fill
1 = solid fill
2 = vertical lines on even pixels
3 = vertical lines on odd pixels
4 = half fill
5 = half fill (complement Style 4)
6 = (not used)
7 = (not used)
8 = horizontal lines on even pixels
9 = horizontal lines on odd pixels

### MARKER_SYMBOL

| | | |
|---|---|---|
| word 1 | Word Count = 2 | Opcode = 17 decimal |
| word 2 | Marker Number | |

Marker Number:

| | | |
|---|---|---|
| 1 = . (dot) | 7 = rectangle | 13 = 3 |
| 2 = + (plus) | 8 = diamond | 14 = 4 |
| 3 = * (asterisk) | 9 = rectangle | 15 = 5 |
| 4 = 0 | with cross | 16 = 6 |
| 5 = X | 10 = 0 | 17 = 7 |
| 6 = triangle | 11 = 1 | 18 = 8 |
| | 12 = 2 | 19 = 9 |

### MARKER_LINE

| | | |
|---|---|---|
| word 1 | Word Count = 2 | Opcode = 18 decimal |
| | Flag | |

Flag: 0 = Lines not drawn
1 = Lines are drawn

### DESIGNATE_CHARACTER_SET

| | | |
|---|---|---|
| word 1 | Word Count = 2 | Opcode = 19 decimal |
| word 2 | Character Set Number | |

Character Set Number: 0 = ROM set (default)
1 = RAM set for Overlay Text
2 = RAM set for Graphics Text
3 = RAM set for both Overlay and Graphics Text

## Attributes (Continued)

### TEXT_SIZE

| | | |
|---|---|---|
| word 1 | Word Count = 2 | Opcode = 20 decimal |
| word 2 | Text Size | |

Text Size:
1 = 5 x 7 character cell
2 = 7 x 11 character cell
3 = 10 x 14 character cell
4 = 14 x 22 character cell
5 = 15 x 21 character cell
6 = 21 x 33 character cell
7 = 20 x 28 character cell
8 = 28 x 44 character cell
9 = 25 x 35 character cell
10 = 35 x 55 character cell
11 = 30 x 42 character cell
12 = 42 x 66 character cell
13 = 35 x 49 character cell
14 = 49 x 77 character cell
15 = 40 x 56 character cell
16 = 56 x 88 character cell
17 = 45 x 63 character cell
18 = 63 x 99 character cell
19 = 50 x 70 character cell
20 = 70 x 110 character cell

### TEXT_PATH

| | | |
|---|---|---|
| word 1 | Word Count = 4 | Opcode = 21 decimal |
| word 2 | character path | |
| word 3 | up-vector | |
| word 4 | line path | |

Character Path, Up-Vector and Line Path are specified by two bits and have four possible values:

| Value | Meaning |
|---|---|
| 0 | left to right |
| 1 | right to left |
| 2 | bottom to top |
| 3 | top to bottom |

### TEXT_SPACING

| | | |
|---|---|---|
| word 1 | Word Count = 3 | Opcode = 22 decimal |
| word 2 | -- 0 -- | Character Spacing |
| word 3 | -- 0 -- | Line Spacing |

Character Spacing - Number of spaces between characters
Line Spacing - Number of spaces between rows of text

### TEXT_ALIGNMENT

| | | |
|---|---|---|
| word 1 | Word Count = 3 | Opcode = 23 decimal |
| word 2 | X factor | |
| word 3 | Y factor | |

X factor - a number, relative to 1000, representing the proportion of text which is aligned to the left of the carriage return point

Y factor - a number, relative to 1000, representing the proportion of text which is aligned below the carriage return point

### SET_RUBBER_BAND_START_POINT

| | | |
|---|---|---|
| word 1 | Word Count = 4 | Opcode = 24 decimal |
| word 2 | Cursor Type (2 or 3) | |
| word 3 | X Coordinate | |
| word 4 | Y Coordinate | |

Cursor Type: 2 = rubber band box
3 = rubber band line

## Attributes (Continued)

### RECTANGLE_FILL_PATTERN

| | | |
|---|---|---|
| word 1 | Word Count = 9 | Opcode = 31 decimal |
| word 2 | 0 (not used) | Row 0 Pattern |
| word 3 | 0 (not used) | Row 1 Pattern |
| word 4 | 0 (not used) | Row 2 Pattern |
| word 5 | 0 (not used) | Row 3 Pattern |
| . | . | |
| . | . | |
| word 9 | 0 (not used) | Row 7 Pattern |

### DEFINE_SCROLLING_WINDOW

| | | |
|---|---|---|
| word 1 | Word Count = 6 | Opcode = 34 decimal |
| word 2 | Minimum X coordinate border | |
| word 3 | Minimum Y coordinate border | |
| word 4 | Maximum X coordinate border | |
| word 5 | Maximum Y coordinate border | |
| word 6 | Plane | |

Plane: 0 = Color plane 0
1 = Color plane 1
2 = Color plane 2
3 = Color plane 3 (default)

### CHANGE_OVERLAY_MARGINS

| | | |
|---|---|---|
| word 1 | Word Count = 5 | Opcode = 8 decimal |
| word 2 | Minimum X coordinate border | |
| word 3 | Minimum Y coordinate border | |
| word 4 | Maximum X coordinate border | |
| word 5 | Maximum Y coordinate border | |

X1 & Y1: Coordinate pair for lower left corner

X2 & Y2: Coordinate pair for upper right corner

### CHANGE_OVERLAY_PLANE

| | | |
|---|---|---|
| word 1 | Word Count = 2 | Opcode = 9 decimal |
| word 2 | Plane Number | |

Plane Number: 0 = color plane 0
1 = color plane 1
2 = color plane 2
3 = color plane 3

## Device Control and Inquiries

### BEGIN_FRAME

| | | |
|---|---|---|
| word 1 | Word Count = 3 | Opcode = 1 decimal |
| word 2 | Blanking | |
| word 3 | Initialize | |

Blanking: 0 = (parameter ignored)
1 = Blanking enabled

Initialize:

0 = 0000 = (parameter ignored)
1 = 0001 = Clear color plane 0
2 = 0010 = Clear color plane 1
3 = 0011 = Clear color plane 0 & 1
. . .
. . .
7 = 0111 = Clear color planes 0, 1, & 2
. . .
. . .
15 = 1111 = Clear all color planes 0, 1, 2, & 3

### END_FRAME

| | | |
|---|---|---|
| word 1 | Word Count = 1 | Opcode = 2 decimal |

### CONTROL_BLINKING

| | | |
|---|---|---|
| word 1 | Word Count = 3 | Opcode = 3 decimal |
| word 2 | Register Number | |
| word 3 | Blink Interval | |

Register Number: 0 = Color register 0
1 = Color register 1
. .
. .
15 = Color register 15

Blink Interval: 0 to 32767 tenths of a second

### CONTROL_INTERRUPTS

| | | |
|---|---|---|
| word 1 | Word Count = 2 | Opcode = 4 decimal |
| word 2 | Flag | |

Flag: 0 = Disable Interrupts
1 = Enable Interrupts

### INQUIRE_DEVICE_ID

(System to Device)

| | | |
|---|---|---|
| word 1 | Word Count = 1 | Opcode = 5 decimal |

INQUIRE_DEVICE_ID Response
(Device to System)

| | | |
|---|---|---|
| byte 1 | Word Count = 8 | |
| byte 2 | Response Opcode = 253 | |
| byte 3 | Command Opcode = 5 | |
| byte 4 | Firmware Rev. Code | |
| byte 5 | | |
| byte 6 | Part Number | |
| byte 7 | | |
| byte 8 | 12065A | |
| byte 9 | | |
| byte 10 | in ASCII | |
| byte 11 | Number of addressable | |
| byte 12 | X pixels | |
| byte 13 | Number of addressable | |
| byte 14 | Y pixels | |
| byte 15 | Display | |
| byte 16 | Status | |

Bytes 11 through 14 show the current screen dimensions configured

The Display Status code returned represents parameter values shown in Appendix A.

# INDEX

## A



## B



## C

## D

## E



## F



## G

## H

## I



## L



## M

## N



## O



## P

## R

## S





## T

## V



## W



## Z