QG-640 User Manual

Color Display Processor Card
for the Q-Bus

265-MU-00
Revision 12
March 4, 1991

matrox
electronic systems

Features

The QG-640 display processor card features:

- Q-bus compatibility
- 640×480 resolution
- 256 colors from a palette of 262,144
- On-board 32/16-bit display processor
- VLSI drawing processor
- 20,000 vectors/second
- 5,000 characters/second
- 1,000,000 pixels/second raster operations
- On-board high-level instruction set
- Low cost

Trademark Acknowledgements

Matrox is a registered trademark of Matrox Electronic Systems Ltd.

QG-640 User Manual

Color Display Processor Card
for the Q-Bus

265-MU-00
Revision 12
March 4, 1991

This Manual Is Valid For The Following Products		
Name	Hardware I.D.	Firmware I.D.
QG-640	Rev. 4	

MATROX Electronic Systems Limited
1055 St. Regis Boulevard
Dorval, Quebec
CANADA H9P 2T4
Telephone: (514)685-2630 Telex: 05-822798
FAX: (514)685-2853

Features

The QG-640 display processor card features:

- Q-bus compatibility
- 640×480 resolution
- 256 colors from a palette of 262,144
- On-board 32/16-bit display processor
- VLSI drawing processor
- 20,000 vectors/second
- 5,000 characters/second
- 1,000,000 pixels/second raster operations
- On-board high-level instruction set
- Low cost

Trademark Acknowledgements

Matrox is a registered trademark of Matrox Electronic Systems Ltd.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

SECTION 1.1 Introduction

1.1 Introduction

The QG–640 is a high-level, 640×480×8 resolution graphics card for the Q-bus. This manual provides all of the information required to install and operate the QG–640.

This rest of this document is divided into the following areas of information:

Chapter	Information
Chapter 2	The technical specifications of the QG–640.
Chapter 3	A functional description of the capabilities of the QG–640.
Chapter 4	Explains how to program the QG–640.
Chapter 5	Command descriptions for the software of the QG–640.
Appendix A	A brief installation and verification procedure.
Appendix B	The as-shipped strap location drawings.
Appendix C	Circuit board layout diagrams.
Appendix D	Lookup table data.
Appendix E	A command list sample.
Appendix F	Default parameter values of the QG–640.
Appendix G	Sample programs to read the status register of the QG–640.
Appendix H	Maintenance and warranty information.
Appendix I	A summary of the software commands for the QG–640.

We believe this manual contains all the information needed to get your QG–640 operational; however, if you do have problems feel free to call or write our Customer Support department at the telephone number or address shown on the cover of this manua . They will be happy to answer any questions you may have.

Chapter 2

Specifications

2.1 Technical Specifications

- Ordering Information :
  - QG–640 (640×480×8 Q-bus Graphics Controller).
  - PG-OCABLE-4 (4-foot video cable).
- Bus :
  - Q-bus plug-in.
- Resolution :
  - 640×480 pixels×8 bits at 60Hz non-interlaced.
  - 256 colors from a palette of 262,144.
- Performance :
  - 35,000 vectors/second.
  - 5,000 characters/second.
  - 1,200,000 pixels/second raster operations.
- Video Timing :
  - Refresh Rate : 50 Hz or 60 Hz, interlaced or non-interlaced
  - Video Frequency : 25 MHz.
  - Horizontal Scan Frequency : 30.63 kHz 640×480 screen resolution.
  - Vertical Frame Rate : 60.07 Hz.
- Memory Map :
  - Strappable to any 16-word block in the 8K I/O page (only odd bytes used).
  - The interrupt level is strappable to any of the four Q-bus levels.
  - The interrupt vector can be strapped anywhere from 0 to 777 octal.
- Interrupts :
  - Interrupt level can be strapped to any one of four Q-bus levels.

- Connectors :
  - Two independent 10-pin AMP output connectors:
    1. +5V via 1K pull-up resistor
    2. No connection.
    3. Red video.
    4. GND (Red).
    5. Green video.
    6. GND.
    7. Blue video.
    8. GND (Blue).
    9. Composite sync.
    10. GND (Sync).
  - Q-bus connector.
- Self Test :
  - ROM-based self test to on-board LEDs at power-on and available on command.
- Power Requirements :
  - +5 VDC 4 A (typical).
- Dimensions :
  - Standard dual height Q-bus card.
  - 160 mm width.
  - 233.68 mm height.
- Environment :
  - 0° C to 55° C operating temperature.
  - 0% to 95% humidity - noncondensing.
  - 7,000 feet maximum altitude.
- Storage :
  - -40° C to 60° C.
  - 5% to 100% humidity - noncondensing.

Chapter 3

Functional Description

3.1 Functional Description

The Matrox QG–640 is a low-cost, high-performance, single-board color graphics processor. It is an excellent display controller for such applications as instrumentation, simulators, process control setups, and medical systems.

The QG–640 has 640×480 display resolution, eight bits per pixel, a 256 of 262,144-color lookup table, and a 60 Hz non-interlaced video refresh rate. The QG–640 has two pipelined processors executing 128 KBytes of ROM based firmware. The high level command set by freeing the Host CPU from managing the display bit map provides a simple interface to any Q-bus graphic application. All drawing functions are handled by the on-board pipelined CPU and Hitachi HD63484 ACRTC drawing processor, resulting in very high system drawing speeds.

3.2 Hardware

The QG–640 uses a microprocessor with a 32-bit internal architecture and a 16-bit bus. This processor acts as the command processor and provides the intelligence to process high level commands into instructions for the graphics processor. The on-board CPU also has the processing power to provide virtual coordinate addressing and matrix transforms. This allows you to choose the coordinate space to be in two dimensions (2D) or three dimensions (3D), with the QG–640 performing the necessary 3D to 2D transforms. The command processor uses a 512-byte FIFO queue for commands from the system CPU. The interface for the Q-bus is discussed in greater detail in Section 4.3. There are 128 KBytes of ROM that provide software to parse commands and to generate instructions for the graphics processor. In addition, there are 128 KBytes of internal RAM for command lists, user fonts, and internal variables. The graphics processor draws primitive graphics forms directly into the video display buffer.

The video display buffer provides output data which is passed through two independent lookup tables (LUT), one for each of two output channels. You can load either LUT with any 256 colors from a palette of 262,144, permitting changes to any color on the display without altering the video display buffer.

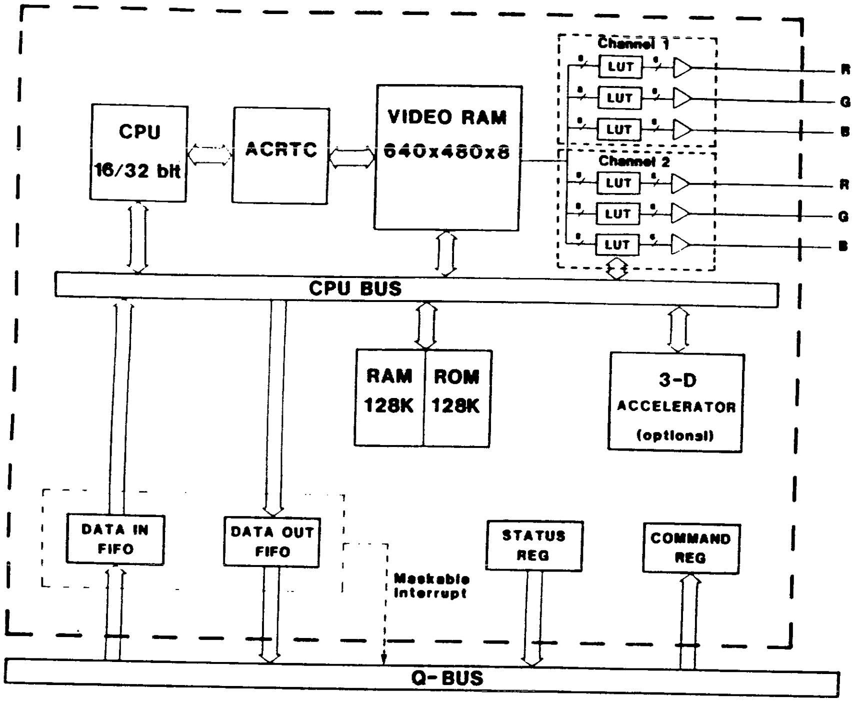


Figure 3.1: QG–640 Block Diagram

3.3 Coordinate Space and Transforms

The QG–640 has firmware in ROM to enable it to draw in either the 2D or 3D virtual work spaces, or directly to the 2D screen. In both work spaces, the axes have 32-bit values. You can define both the window and the viewport. The window is the section of the virtual work space that you wish to be mapped to the viewport. The viewport is the physical area of the screen that can be modified. While you can always modify the entire virtual work space, only the viewport pixels that correspond to points in the window are affected by graphics commands. The results of drawing commands on areas inside the virtual work space, but outside of the window, will not appear on the screen or be saved — images that pass through the window will be clipped as they are mapped to the viewport.

When drawing in 2D, you are provided with a set of 2D graphics commands. These commands draw the graphics primitives: points, lines, arcs, circles, rectangles, ellipses, and polygons. You can set masks so that dashed lines and patterns are produced in filled figures. The virtual points are mapped to the real display coordinates (pixel locations) by the QG–640 (see Figure 3.2). You also have the option of drawing in the frame buffer using direct screen operations. For a more detailed discussion of 2D drawing, see Chapter 4.

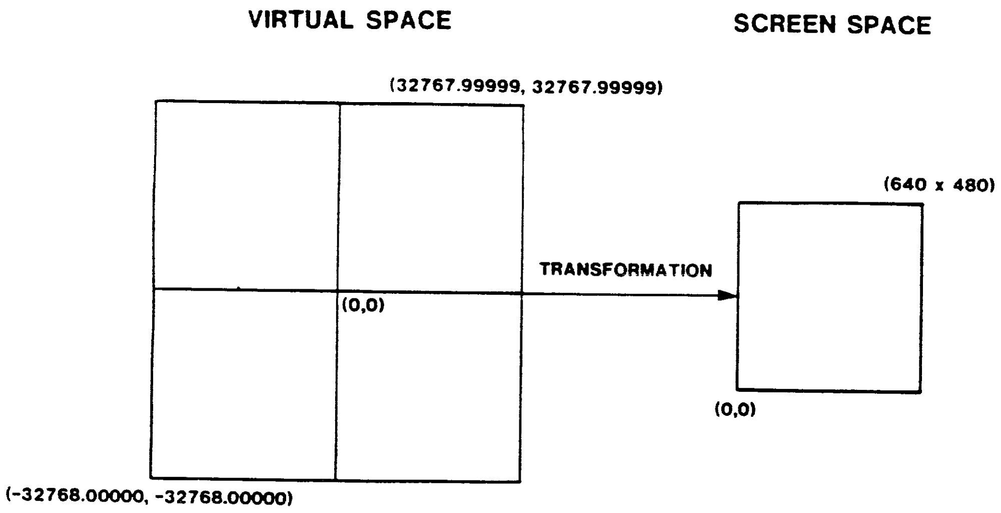


Figure 3.2: 2D Virtual Space to Pixel Mapping

In 3D, you have access to the virtual coordinate system as well as full control over viewing angles and distances. The QG–640 uses a modelling matrix to rotate, scale, and translate the virtual coordinates of the 3D object. A viewer reference point matrix is used to translate a point to the center of the currently defined viewport. This viewing matrix affects the angle of rotation by moving the eye about the object, leaving the object stationary (see Chapter 4). You can also set the angle and distance in the 3D to 2D transform.

3.4 Graphics Attributes and Primitives

The QG–640 presents you with a drawing model consisting of a pen and ink. The pen has two positions, the 2D and 3D current points. The ink has 256 colors, those stored in the output lookup table. Drawing operations use the current color. The current points can be moved to any location in their respective coordinate spaces with a single command and the current color can be selected from any of the LUT colors, also with a single command. Primitives are drawn from the appropriate current point in the current color — some relocate the current point, others do not (see Table 3.1). When drawing an image in the display buffer, the color indices used depend on several graphics attributes. These attributes are described in Section 4.5.

The graphics commands provide the ability to draw simple geometric figures with single commands. These figures can be drawn with patterned lines, and filled in the case of closed figures. The Area Pattern and Line Pattern masks determine how the figure is drawn. The QG–640 also has the ability to mask off any of the eight display buffer bit planes from read and write operations. This allows you to load different images into the buffer and to perform image overlays.

2D Command	3D Command	Effect	Current Point Moved?
ARC		Draws arc	No
CIRCLE		Draws circle	No
DRAW	DRAW3	Draws line	Yes
DRAWR	DRAWR3	Draws line	Yes
ELIPSE		Draws ellipse	No
MOVE	MOVE3	Moves current point	Yes
MOVER	MOVER3	Moves current point	Yes
POINT	POINT3	Colors current point	No
POLY	POLY3	Draws polygon	No
POLYR	POLYR3	Draws polygon	No
RECT		Draws rectangle	No
RECTR		Draws rectangle	No
SECTOR		Draws pie slice	No

Table 3.1: Drawing Command Summary

The 2D command set provides instructions to draw arcs, circles, ellipses, lines, points, polygons, and rectangles. In 3D, you can draw lines, points, and polygons.

3.5 Text

Text is specified in 2D space. There are two predefined fonts and two user-defined fonts. The first predefined font is drawn as thin stroke, vector based characters; the second as fat, smooth characters that are constructed with lines whose thickness is proportional to the character size. You can set the size, angle of rotation, and aspect ratio of the characters. The size and justification about the current point can also be set.

3.6 Direct Screen Operations

One of the major features of the QG–640 is the ability to perform block moves of pixel data. You can copy a block from one part of the display buffer to another. Using a single command, you define the block to be transferred, its destination, and the major and minor directions in which it is to be read or written. Using different transfer directions when reading and writing blocks, you have the ability to perform inversions and rotations. The inversion of a block of pixels is illustrated in Figure 3.3.

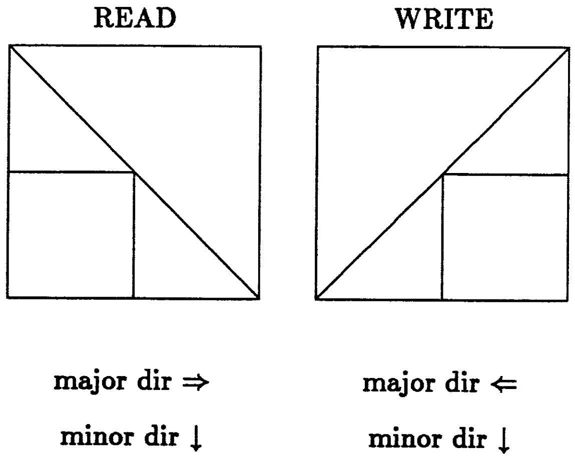


Figure 3.3: Raster Transfer of Pixels

Images can also be transferred to and from the Q-bus. Pixel values can be sent through the system, allowing quick reading and writing of images. This makes the QG–640 a useful tool for displaying images.

There are 14 direct screen operations supported by the QG–640. These commands are specified directly in screen coordinates and allow you to plot pixels directly onto the display, bypassing the modelling mechanism. This results in faster drawing speeds.

Chapter 4

Programming the QG-640

4.1 Introduction

This chapter explains how to program the QG-640. Related commands are assembled into groups and explanations are given as to how the commands in a group are used together to perform various tasks. Although the formats of many commands are given, this chapter is not intended as a command reference — Chapter 5, which contains the command descriptions arranged in alphabetical order, is better suited for that purpose. Rather, this chapter is intended as an overview of the QG-640 geared toward the programmer.

A programmer operates the QG-640 by sending it commands. The format of the commands depends on which of two command modes you are using: ASCII or Hex.

In ASCII mode, the commands are issued as ASCII strings that form keywords, ASCII decimal value parameters, and ASCII character parameters. The string 'CLEARS␣23', for example, instructs the QG-640 to clear the screen to color index 23. Command names in this mode have a short form which can be used for brevity. For example, 'CLEARS␣23' can also be sent as 'CLS␣23'. ASCII mode provides ease of operation since the keywords are mnemonic in nature and the parameters are decimal values. Commands in this mode do, however, take more space than commands in Hex mode.

Hex mode allows commands to be sent and stored in a more compressed format. Binary opcodes are used instead of keywords and binary values instead of ASCII values for parameters. For example, the Hex mode equivalent to 'CLEARS␣23' is 'OF 17'. Hex mode commands lack the mnemonic character of ASCII mode commands, but they can be stored in less space and sent to the QG-640 in less time than ASCII mode commands. See Section 4.2 for a more detailed explanation of the two command modes.

In this chapter, commands are described and examples are given in ASCII mode format only. Chapter 5 provides descriptions of each command in both formats.

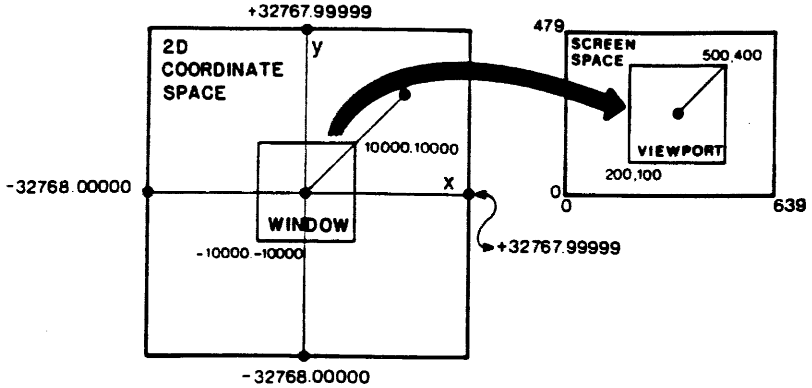


Figure 4.1: The 2D Drawing Environment

To draw in 2D, a window and a viewport are defined to map all or part of the 2D virtual coordinate space to the screen. Graphics attributes such as color, line style, and drawing mode are selected and graphics primitives, text commands, and fill commands are used to draw the image. The following string example defines the window and viewport shown in Figure 4.1 and draws a line in them. The operations specified by this code will become clear as you read this chapter. The ␣ characters represent any one of several delimiters.

The valid delimiters are listed in Section 4.2, which explains the conventions used to describe commands in this manual.

```
CLEARS␣0
WINDOW␣-10000␣10000␣-10000␣10000␣
VWPORT␣200␣500␣100␣400␣
MOVE␣0␣0␣
DRAW␣20000␣20000␣
```

The remainder of this chapter is organized as follows:

- Section 4.3 is an overview of the QG-640 bus interface.
- Section 4.4 discusses coordinate spaces, windows, and viewports.
- Section 4.5 explains graphics attributes.
- Section 4.6 discusses graphics primitives.
- Section 4.7 covers the text commands.
- Section 4.8 explains fills.

3D drawing is a little more involved than 2D drawing. You draw in a 3D coordinate space that is mapped to the same window and viewport used by the 2D coordinate space. A number of transforms can be specified to determine how the drawing is mapped to the viewport. These transforms define the following aspects of the image:

- The scaling, rotation, and translation (position) of the image in the 3D coordinate space.
- The position and direction of view of the viewer with respect to the 3D coordinate space.
- The hither and yon (front and back) clipping planes.
- The distance of the viewer from the viewing plane and the angle of view.

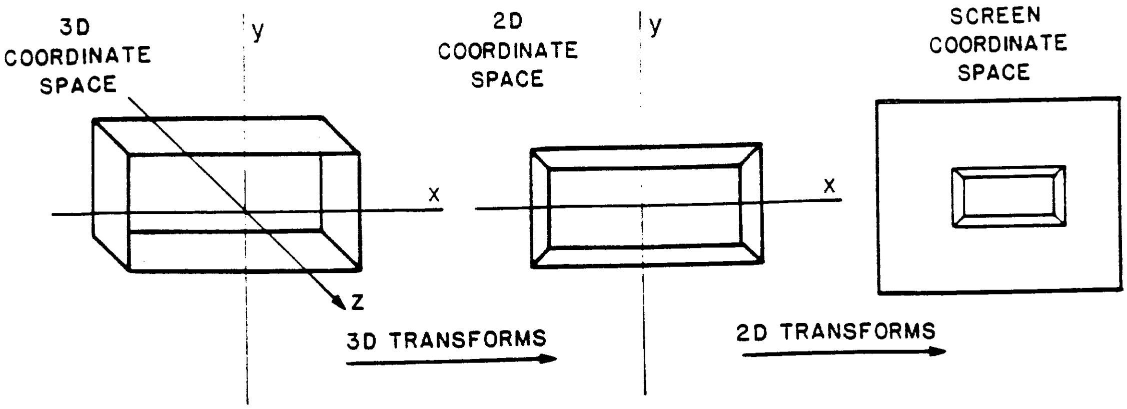


Figure 4.2: The 3D Drawing Environment

The 3D transforms and coordinate space are described in Section 4.4.

Drawings can be stored in the QG-640 as command lists. A command list can be run (drawn) as required. For example, if a diagram is in a command list and is to be drawn on another part of the screen, you can set a new transform, clear the screen, and run the command list. Command lists are explained in Section 4.9.

Certain operations can be performed directly on the screen, bypassing the coordinate spaces and transforms. Raster operations copy from one part of the screen to another, and between the screen and system memory. These operations are described in Section 4.10.

4.1.1 512×512 Mode

The QG-640 can be strapped to an alternate display mode: 512× 512 pixels. In this manual, rather than express all screen coordinates as two possible ranges (640×480 and 512×512), only the 640×480 range will be used. When in 512×512 mode, the command parameters relating to the screen size will have limits of [0...511, 0...511] instead of [0...639, 0...479]. Some of the default flag settings are also affected (refer to the Appendices). The following commands are affected:

IMAGER IMAGEW PDRAW RASTOP
RASTRD RASTWR SARC SCIRC
SDRAW SDRAWR SELIPS SMOVE
SMOVER SPOLY SPOLYR SRECT
SRECTR SSECT VWPORT XMOVE

4.2 Command Format

4.2.1 Documentation Conventions

Throughout this chapter and Chapter 5, the following conventions are used to describe the QG–640 commands:

- Parameter names are printed in lowercase block characters.
- Hexadecimal values are printed in typewriter style characters.
- Command keywords are printed in uppercase roman characters.
- The ␣ character is used to indicate the position of a delimiter.

4.2.2 ASCII Command Format

When the QG–640 is in ASCII Command Mode (the default mode), commands are sent to the QG–640 as either uppercase or lowercase ASCII character strings. A command string consists of a keyword identifying the command, parameters (where required), and delimiter characters.

The keywords for most commands have a long form and a short form. For example, the long form of the draw command is DRAW and the short form is D. Parameters are either ASCII decimal numbers or text strings enclosed by quotes. Delimiters can be:

- A space character
- The tab character
- A comma
- A semicolon
- A carriage return
- A line feed
- A hyphen acts as a delimiter when it identifies negative values.
- A plus sign acts as a delimiter when it identifies positive values.

To draw a line from the current pen position to the xy coordinate {100,200}, the following ASCII string can be used:

DRAW␣100␣200␣

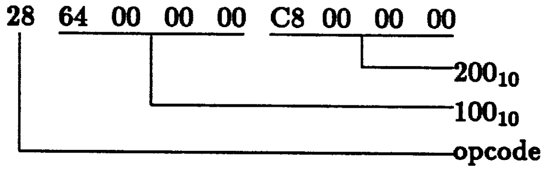
where ␣ is any of the delimiters described previously.

The ASCII Command Mode, with its mnemonic commands, is particularly suited for use in a user-interactive mode. The CA␣ command is used to set the QG–640 to ASCII Command Mode.

4.2.3 Hex Mode

When the QG–640 is in Hex Mode, commands are sent as binary byte values. A command consists of a single byte opcode followed by binary parameter values. In this manual, these values are given as hexadecimal numbers.

For example, to draw a line from the current pen position to the xy coordinate {100,200}, the following command can be used:



```
28 64 00 00 00 C8 00 00 00
```

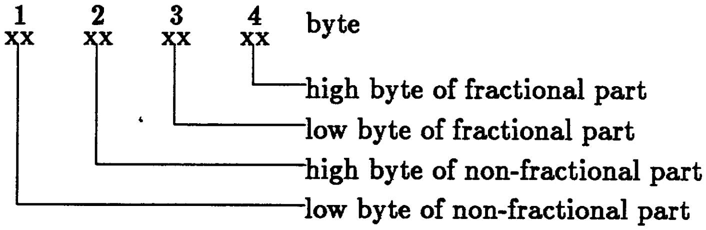
4.2.4 Parameter Types

The QG–640 uses three numerical types: Chars, Ints, and Reals. The way these are sent is dependent on the current command mode.

In ASCII Mode, the Char parameter type is an ASCII character code. In Hex Mode, it is a single byte value in the range 0 to 255.

An Int in ASCII Mode is an ASCII decimal value from -32767 to +32767 inclusive. An unsigned Int is an ASCII positive decimal value from 0 to 65535. A hyphen immediately preceding an ASCII Int indicates a negative value. In Hex Mode, an Int is a two-byte binary value with the low byte first. Hex Mode negative Ints use two's complement form.

A Real has two parts: a fractional part and a non-fractional part. In ASCII Command Mode, a Real is an ASCII decimal real number from -32768.00000 to +32767.99999 (the decimal is optional when the fractional part is 0). In Hex Command Mode, a real number is represented by four bytes using the following format:


1
2
3
4
byte
xx
xx
xx
xx
high byte of fractional part
low byte of fractional part
high byte of non-fractional part
low byte of non-fractional part

where the value of the bytes is determined by multiplying the decimal real number by 65536 and converting the result to hexadecimal form. For example, 3.142 becomes:

$$3.142_{10} \times 65536_{10} = 205914_{10} = 0003245A_{16}$$

The non-fractional part is equal to 0003 and the fractional part 245A. The real is sent as 03 00 5A 24.

This method is also valid for calculating negative real numbers – the positive hex number is calculated and then 2s complemented. In this way, -3.142 is calculated to be FC FF A6 DB.

4.3 Communications

This section gives you an overview of the QG–640 bus interface. It also provides information for writing software drivers for the board.

The QG–640 accepts commands and graphics data from the Host CPU via the Q-bus. The QG–640 is also capable of returning output reports and frame buffer dumps via the Q-bus to the Host CPU.

Communications between the system and the QG–640 can be summarized into six categories:

Categories	Direction	Locations Used
Control	Host ⇒ QG–640	Command Register
Graphics Commands	Host ⇒ QG–640	Data In FIFO
Error Reports	QG–640 ⇒ Host	Data Out Register and Status Register
Data Output	QG–640 ⇒ Host	Data Out Register and Status Register
Board Status	QG–640 ⇒ Host	Status Register
Interrupts	QG–640 ⇒ Host	See Subsection 4.3.2

Refer to Figure 4.3 for a diagram depicting the QG–640's communications scheme.

4.3.1 Bus Interface

The QG–640 provides you with four programmable registers. They are:

- Status
- Data In FIFO
- Data Out
- Command

All registers can be accessed on a byte or word basis. (See Appendix A for details on the addresses of these ports.)

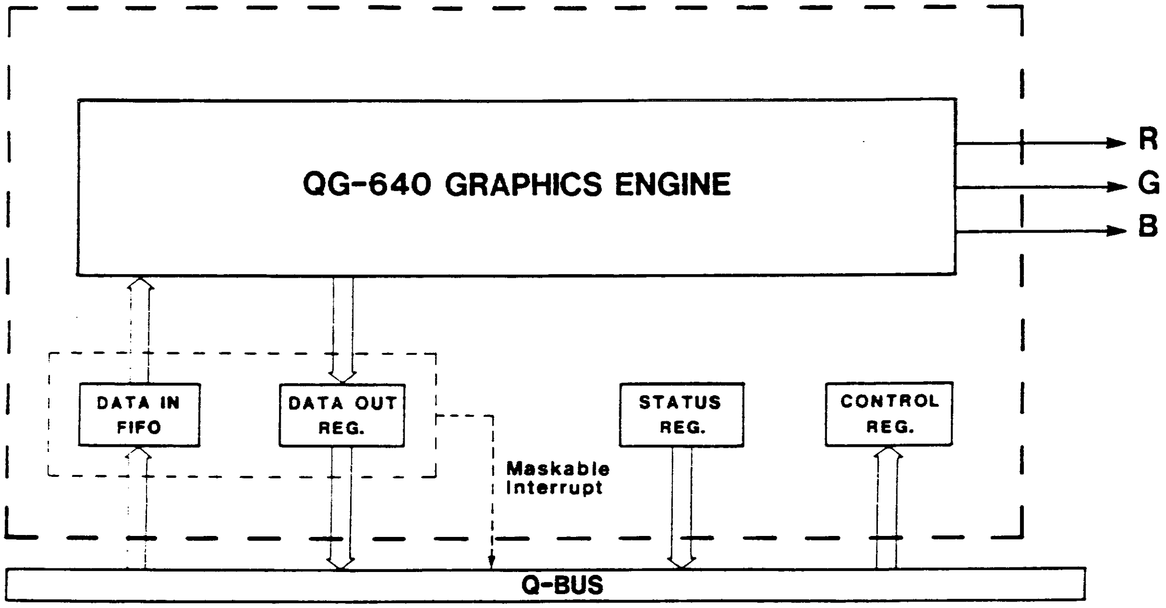
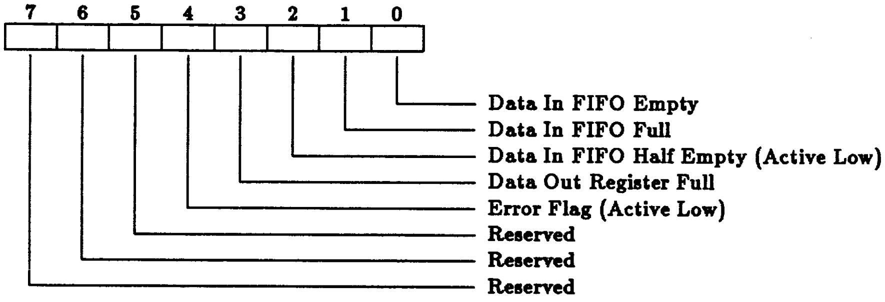


Figure 4.3: QG–640 Communications Scheme

4.3.1.1 The Status Register

The Status Register is an eight-bit, read-only register containing board error information, and information on the current state of the Data In FIFO.



Bit 0: Data In FIFO Empty. A 1 in this bit specifies the FIFO is empty. A 0 in this bit specifies otherwise.

Bit 1: Data In FIFO Full. A 1 in this bit denotes the FIFO is full (512 bytes). A 0 in this bit denotes otherwise.

Bit 2: Data In FIFO Half Empty. A 0 in this bit specifies the FIFO is half empty (256 bytes). A 1 in this bit specifies otherwise.

Bit 3: Data Out Register Full. A 1 in this bit specifies the Data Out Register is full. A 0 in this bit states otherwise.

Bit 4: Error Flag. The Error Flag describes the contents of the Data Out Register. A 0 in this bit denotes an error code. A 1 in this bit denotes data.

Bits 5 - 7: Reserved for future use.

A sample program to read the QG–640's Status Register is provided in the Appendices.

4.3.1.2 The Data In FIFO

The Data In FIFO is an eight-bit, write-only port to the QG–640's internal FIFO, and is used to pass commands and data to the QG–640. The on-board microprocessor reads the commands and parameters, and performs the necessary QG–640 graphics operations.

The input section of the bus interface consists of a 512-byte FIFO, and 3 bits of status information in the Status Register. The FIFO is implemented in hardware, and if its protocol is obeyed, is capable of operating as fast as the system bus. Obeying the FIFO protocol implies never writing to a full FIFO.

A CPU on the QG–640 is provided to empty the FIFO. This CPU can be busy when the Host CPU accesses the FIFO. The time taken by the QG–640 to read any number of bytes from the FIFO is determined by the commands contained in those bytes.

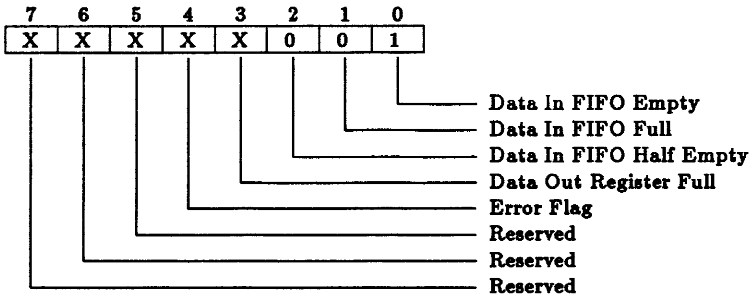
Bit 0 of the Status Register is the FIFO Empty Status bit, and is set to 1 whenever the FIFO is totally empty. This bit is reset to 0 when the Host CPU writes a byte to the FIFO, and set to 1 when the QG–640 reads the last byte from the FIFO.

Bit 1 of the Status Register is the FIFO Full Status bit, and is set to 1 whenever the FIFO is totally full (512 bytes). This bit is set after the Host CPU writes the filling byte to the FIFO, and reset to 0 when the QG–640 reads a byte from the FIFO, leaving room for at least one byte. Note that when the FIFO is full and additional data is written to the QG–640, the FIFO remains unchanged and the new data is discarded.

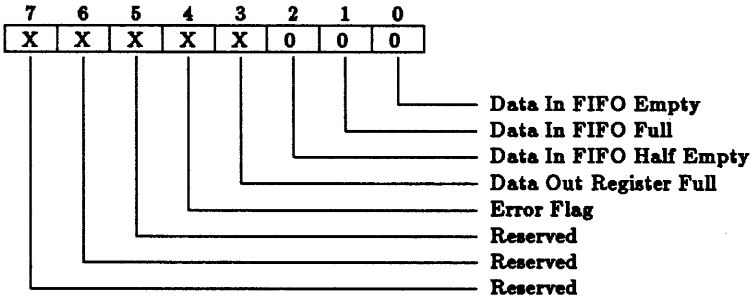
Bit 2 of the Status Register is the FIFO Half Empty Status bit, and is reset to 0 whenever there are **at least** 256 empty spaces in the FIFO. This bit is set to 1 when there are more than 256 bytes of the FIFO currently holding data not yet read by the QG–640. Thus, when the FIFO is empty (bit 0 of the Status Register set to 1), the FIFO is also half empty (bit 2 of the Status Register reset to 0) since there are **at least** 256 free bytes in the FIFO (in fact, there are 512 free bytes at this point).

The following pages provide an example of how to send commands to the QG–640.

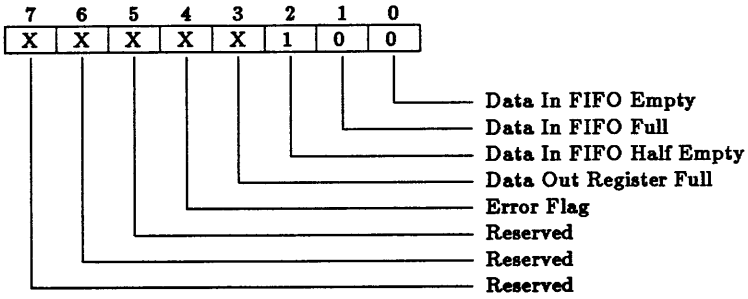
1. Suppose the QG–640 has just been reset and the FIFO is now empty:



2. The Host now sends 254 bytes of commands to the FIFO. The FIFO is neither empty nor full, since only 254 bytes have been sent; however, the FIFO is half empty, since there is room for at least 256 more bytes of data.

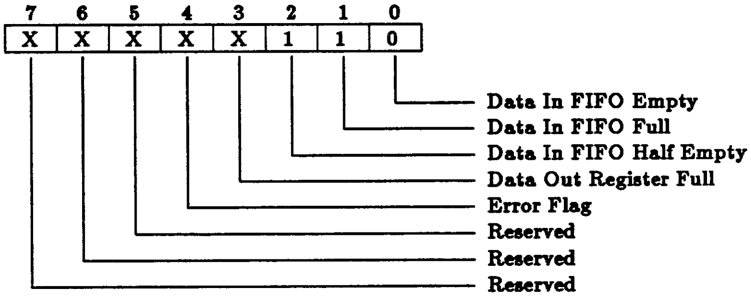


3. The Host sends another three bytes of data to the FIFO. The FIFO is still neither empty nor full, but there is now room for only 255 more bytes, since 257 have been sent; the FIFO is no longer at least half empty.



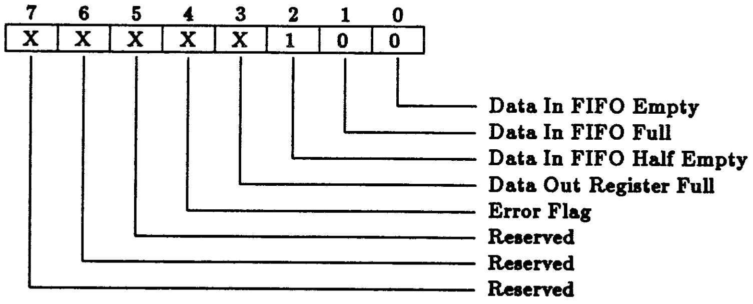
7	6	5	4	3	2	1	0
X	X	X	X	X	1	0	0

4. At this point, the Host by reading the Status Register knows there are less than 256 free bytes in the FIFO, since the FIFO is more than half (but not yet completely) full. Here is where the danger of overrunning the FIFO becomes critical, especially in a context where the transferring process may not know the previous number of bytes transferred, or where more than one process is accessing the QG–640. The only safe way to continue is to check the FIFO Full bit (bit 1 of the Status Register) before each byte is sent. If the Host does this for another 255 bytes, it will stop when the FIFO is full.

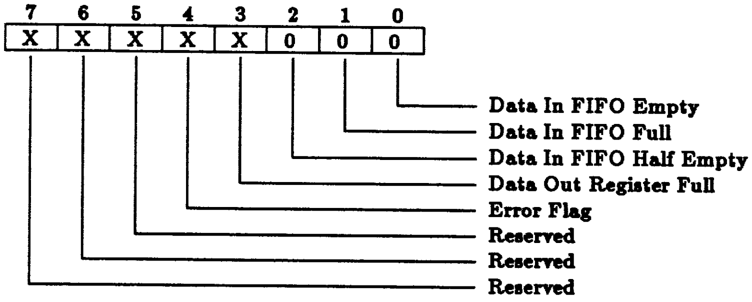


7	6	5	4	3	2	1	0
X	X	X	X	X	1	1	0

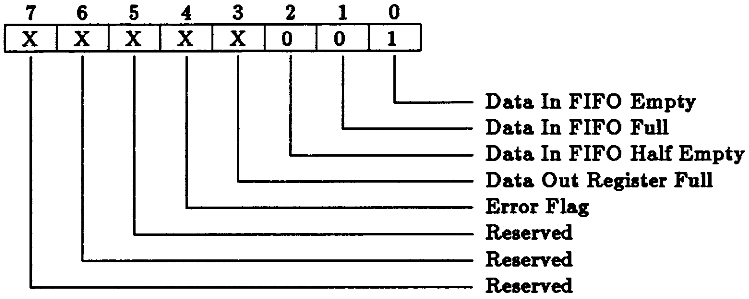
5. Now the Host must wait for the QG 640 to catch up. The Host continues to poll the Status Register. After the QG–640 reads one byte from the FIFO, the FIFO is no longer full and the status becomes:



6. As the QG–640 continues to read, it reads enough data to leave at least 256 free bytes in the FIFO, and it becomes half empty:



7. Finally, the QG–640 continues its reading of the FIFO until the FIFO is completely empty:



Note that the previous example has been simplified to separate the roles of the Host and the QG–640. In reality, the QG–640 could remove bytes from the FIFO as fast as the Host could send them, or the QG–640 could run a previously stored command list from its on-board RAM and not read the FIFO until the command list is processed.

To avoid the above conditions from occurring, an alternate procedure is suggested to send data to the QG–640. This procedure sends data by polling the FIFO Half Empty flag bit of the Status Register. For example, to transfer a buffer of "buffer length" bytes to the QG–640, do the following:

1. Initialize the Host Buffer Pointer (HBP) to "*start of user buffer*", and initialize Bytes Remaining (BR) to "*buffer length – 1*".
2. Check the Status Register. If the FIFO is not half empty (bit 2 set to 1), repeat Step 2; else, proceed to Step 3.
3. If $BR - 256 \geq 0$, transfer 256 bytes (1/2 FIFO) to the QG–640; else, transfer BR bytes.
4. Calculate the following:
   - $HBP = HBP + 256$
   - $BR = BR - 256$
5. If $BR > 0$, repeat Step 2; else, proceed to Step 1.

The above procedure eliminates polling the FIFO Full flag of the Status Register before each byte is transferred; however, you must wait for the FIFO Half Empty flag to be set.

4.3.1.3 The Data Out Register

The Data Out Register is an eight-bit, read-only register used to receive data from the QG–640. Image data, flag read data, command list read data, and error messages can be read from this register. The Host differentiates between error information and other data by checking bit 4 of the Status Register.

The Data Out Register is only one byte deep. When it is full, the Data Out Register Full bit of the Status Register is set to 1. If the Data Out Register holds requested output data, the Data Out Register Full bit and the Error Flag bit (bits 3 and 4 of the Status Register) are set to 1.

If the Data Out Register contains error information, the Data Out Register Full bit is set to 1 and the Error Flag bit is set to 0.

If the Data Out Register is empty, the Error Flag bit value is undefined (bit 4 of the Status Register may be 0 or 1). Thus, the Data Out Register Full bit must be checked before the Error Flag bit, and qualifying output becomes the second step of the reading process, to be performed only after determining that a byte of output data is ready in the Data Out Register.

Error messages are the result of parameter and range checking on the QG–640, and "run-time" errors occurring during the processing of a stored command list.

By default, the QG–640 runs with error message reporting disabled so that only solicited output data, such as image data and command list data, appear in the Data Out Register. Error messages are one byte long.

Output data reports (Error Flag bit set to 1) are the result of a command that reads back the status (for example, FLAGRD, LUTXRD), or a command that reads back screen pixel data (for example, RASTRD, IMAGER).

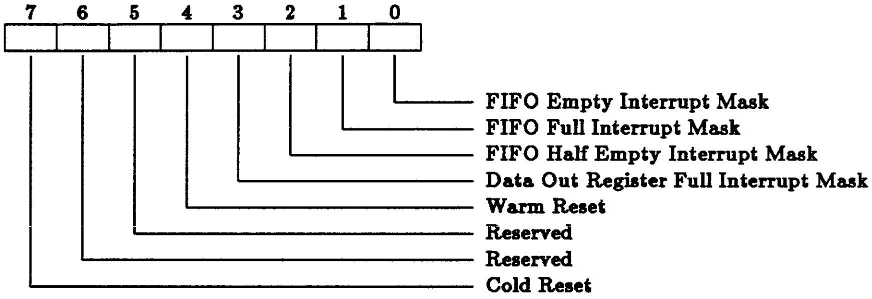
In Hex communications mode, the total number of bytes returned by any QG–640 command is given in the command description, with the exception of the IMAGER (Image Read) command which returns pixel data in a run-length encoded format.

In ASCII communications mode, the total number of bytes returned by a command is unknown since a 1-byte hex value may be returned as 1, 2, or 3 ASCII bytes (that is, 0 to 255). Note that when more than one value is returned, each value is separated by a comma (,) and the last value is terminated with a carriage return (CR, 0D hex). For example, hex values FF 00 FF will be returned as ASCII values: 255, 0, 255 (CR).

Note that on cold reset, the Data Out Register contains one byte of data for system initialization. Consequently, it is essential to read the Data Out Register following a cold reset to clear the Data Out Register Full bit (reset bit 3 of the Status Register to 0).

4.3.1.4 The Command Register

The Command Register directs the QG–640's interrupt strategy, and can reset (warm or cold) the board. An interrupt is enabled when the corresponding mask bit is set to 1. The Command Register is shown below:


7
6
5
4
3
2
1
0
FIFO Empty Interrupt Mask
FIFO Full Interrupt Mask
FIFO Half Empty Interrupt Mask
Data Out Register Full Interrupt Mask
Warm Reset
Reserved
Reserved
Cold Reset

Bit 0: FIFO Empty Interrupt Mask. Write a 1 to this bit to enable the FIFO Empty Interrupt. Write a 0 to this bit to disable the FIFO Empty Interrupt.

Bit 1: FIFO Full Interrupt Mask. Write a 1 to this bit to enable the FIFO Full Interrupt and a 0 to disable the interrupt.

Bit 2: FIFO Half Empty Interrupt Mask. Write a 1 to this bit to enable the FIFO Half Empty Interrupt. Write a 0 to this bit to disable the FIFO Half Empty Interrupt.

Bit 3: Data Out Register Full Interrupt Mask. Write a 1 to this bit to enable the Data Out Register Full Interrupt and a 0 to disable it.

Bit 4: Warm Reset. Write a 1 to this bit to terminate the execution or definition of a command list without affecting any of the board's parameters or board's memory. Write a 0 to this bit to disable the warm reset operation. (Wait a few microseconds before reverting the bit to 0.)

Note: The warm reset function empties the QG-640's input FIFO. The host must wait for the FIFO to empty before it sends any new commands to the QG-640's input FIFO. The host polls bit 0 of the Status Register to learn if the FIFO is empty.

Bits 5 - 6: Reserved for future use.

Bit 7: Cold Reset.

1. Write hexadecimal value 80 (0x80) to the command register.
2. Wait a minimum of 100 microseconds before attempting to access the board again.
3. Write a zero to the command register.
4. Wait a minimum of 150 milliseconds before attempting to access the board again.
5. Poll the status register, waiting until the data out register full bit is high (set). This indicates that the start-up value is available at the data out port register.
6. Read the data out port register and verify that the start-up value is correct. It must be 0x20.
7. Read the status register and verify that the data out port bit is clear. If it is not clear, return to Step 5.
8. The board is now ready to accept commands.

In addition, after a power up, you must follow Steps 3 through 7 before sending any data to the QG-640. If this procedure is not followed, the board may not perform correctly. This procedure resets the hardware and initializes the QG–640 with the default parameter values listed in Appendix F.

4.3.2 Interrupts

The QG–640 is capable of generating a maskable interrupt on the Host system bus, under any or all of the following conditions:

- FIFO Empty
- FIFO Full
- FIFO Half Empty
- Data Out Register Full

Each interrupt is individually enabled by setting its corresponding mask bit in the Command Register. The QG–640's interrupt levels, vectors, and positional dependence are strappable as indicated in Appendix A.

The Host writes a 1 to enable the given interrupt, and a 0 to disable it. After a cold reset or power up, all mask bits are reset to 0.

Interrupts are generated according to the state of the corresponding bits in the Status Register. This implies that an interrupt will continue to be generated provided the condition remains unchanged. For example, if the FIFO Empty Interrupt is enabled, the interrupt is generated continuously until at least one byte is written to the FIFO. Furthermore, the bits in the Status Register are not latched. In the case of the FIFO Full Interrupt, this means that the condition causing the interrupt (FIFO Full) may no longer be the current setting of the Status Register bits because the QG–640's on-board CPU may have cleared some bytes from the FIFO during the Host CPU interrupt sequence.

To complete the interrupt service cycle, the Host must issue a read to the Command Register. This restores the QG–640 interrupt generator.

A typical interrupt service routine for the QG–640 would follow the sequence below:

1. Read the Status Register to determine the source of the interrupt.
2. Service the request.
3. Set the new interrupt mask.
4. Issue a read to the Command Register to restore the interrupt generator.

The Host must not attempt to set the interrupt mask with an instruction that performs a read-modify-write cycle on the Command Register, or the interrupt can recur immediately. As soon as the Command Register is read, the interrupt generator on the QG–640 is restored, and can generate an interrupt before the Host's setting of the mask bits has taken effect.

4.3.3 Transfer Under Interrupt Control

The polling method allows you to communicate with single-user/single-task systems, but is not adequate for multitasking systems. For this reason, you should use the following interrupt control sequence:

1. Initialize the Host Buffer Pointer (HBP) to the "*start of the user buffer*", and Bytes Remaining (BR) to "*buffer length – 1*".
2. Set the FIFO Half Empty Interrupt mask in the Command Register (bit 2 set to 1).
3. Read the Command Register to restore the interrupt generator of the QG–640.

   Note: The transfer is performed by the interrupt service routine described below.
4. Wait for the I/O to be marked as complete by the interrupt service routine when the transfer is finished.

The QG–640 interrupt service routine for the FIFO Half Empty condition comprises the following steps:

1. Validate the interrupt request; the FIFO Half Empty bit (bit 2) of the Status Register is reset to 0. If the FIFO is not half empty, this is a spurious interrupt (assuming the QG–640 has had all of its other interrupts disabled).
2. If $BR - 256 \geq 0$, transfer 256 bytes (1/2 FIFO) to the QG–640; else, transfer BR bytes.
3. Perform the following operations:
   - $HBP = HBP + 256$
   - $BR = BR - 256$
4. If $BR > 0$, enable the FIFO Half Empty interrupt (bit 2 of the Command Register set to 1); else, disable the interrupt mask bits (bits 0 through 3 of the Command Register reset to 0), and mark the I/O as complete.
5. Read the Command Register to restore the interrupt generator of the QG–640.

4.4 Transforms

The QG–640 displays images on a video screen using a physical coordinate space of 640× 480 pixels or 512× 512 pixels (as set by the straps described in Appendix A). This is the maximum resolution of the displayed image. The user, however, draws the images in one of two virtual coordinate spaces which have a much higher resolution. Transforms are used to map images in the virtual coordinate space to real screen coordinate space in such a way that the maximum resolution is always maintained. For example, a user could use the QG–640 to draw a very detailed picture of a tree. When the whole tree was displayed, the screen resolution would only allow larger details such as branches, the trunk, and the form of the tree to be seen. However, if the picture in the virtual coordinate space was detailed enough, you could redraw the picture, zooming in on one leaf.

The two virtual coordinate spaces are: the 2D coordinate space with x and y axes and the 3D coordinate space with x, y, and z axes. The coordinates on each axis run from -32768.00000 to +32767.99999. Figure 4.4 shows the two virtual coordinate spaces, and illustrates the relationship between one another and the screen space.

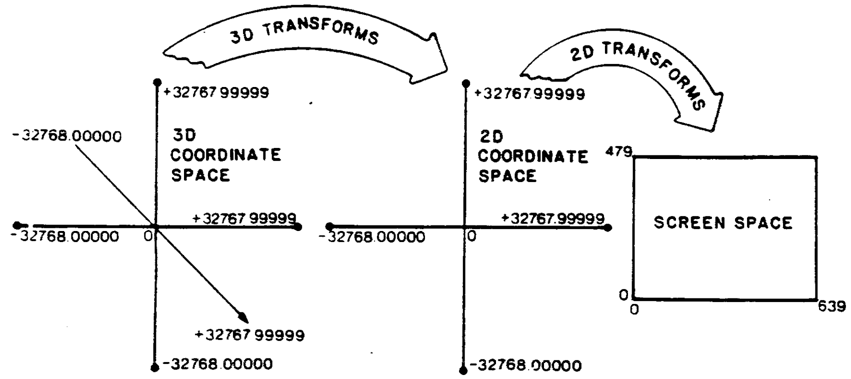


Figure 4.4: Coordinate Spaces

4.4.1 Two Dimensional Transforms

The 2D work space uses Cartesian coordinates with the origin in the center and coordinates going from -32768.00000 to +32767.99999 on each axis. The WINDOW and VWPORT commands are used to map a rectangular section of this coordinate space to the display. The WINDOW command has the following format:

$$\text{WINDOW}_\sqcup x_{1\sqcup} x_{2\sqcup} y_{1\sqcup} y_2$$

where the parameters x_1 and y_1 form one coordinate pair, and x_2 and y_2 form another. These coordinate pairs specify the two opposing corners of a rectangular section of the work space. This section is referred to as a window. Any image drawn in the window will be mapped to the current viewport – a rectangular section of the screen space. If you do not specify a window, the default 640× 480 window centered on the coordinate space origin will be used.

The VWPORT command defines the viewport, and has the following format:

VWPORT␣x_1␣x_2␣y_1␣y_2

where coordinate pairs $\{x_1. y_1\}$ and $\{x_2. y_2\}$ specify the opposing corners of a rectangular section. In this case, however, the coordinates must be given in screen coordinates rather than work space coordinates. As indicated in Figure 4.4, the screen coordinate space has its origin in the lower left corner, has 640 (0-639) points on the x axis, and 480 (0-479) points on the y axis. If a viewport is not specified, the viewport will include the entire screen. Note that the viewport's maximum x and y coordinates depend on the strapped screen size.

The command string in Section 4.1 that defines the window and viewport in Figure 4.1 illustrates how you can define different windows and viewports.

4.4.2 Three Dimensional Transforms[1]

You draw 3D pictures in the 3D work space. Their position, size, and viewpoint are determined by a number of transforms. Modelling transforms determine the scale (size), rotation, and position (translation) of the picture within the coordinate space. Viewing transforms determine the viewer's position and direction of view with respect to the coordinate space. The clipping function's hither and yon clipping planes slice off the front and the back of the picture when used. 3D to 2D transforms project the 3D image onto the 2D coordinate space. Once the image is in the 2D coordinate space, it is mapped to the screen by the window to viewport transforms that were described in the previous section.

The 3D transforms allow you to manipulate the graphic object and the viewer's perspective. For example, when using a routine to draw a house, if you want to get two houses in different parts of the 3D coordinate space, you can set up the translation transform for one position and then run the routine to draw the first house. You would then set up the translation transform for another position and run the same command list to draw the second house.

Figure 4.5 shows how a house is displayed when the default parameters for the 3D transforms are used. Refer to the Appendices for the command list that draws this house. In the following pages, several examples are used to illustrate how different transform settings affect the house.

[1] For more detailed discussions on graphic transforms refer to *Principles of Interactive Computer Graphics* by Newman and Sproull (McGraw-Hill, 1979) or *Fundamentals of Interactive Computer Graphics* by Foley and van Dam (Addison-Wesley, 1982).

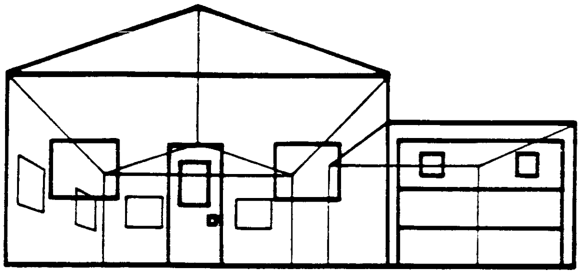


Figure 4.5: Default House

4.4.2.1 Modelling Transforms

The modelling transforms are the first transforms to affect objects being drawn. There are three different modelling transforms:

- the translation transform, which moves objects in the coordinate space by offsetting their coordinates as they are drawn.
- the rotation transform, which rotates the object around each of the three axes.
- the scaling transform, which determines the size of the object.

Each x, y, and z coordinate set in a graphic object's description is multiplied by the modelling matrix (M). This modelling matrix can be loaded directly by using the MDMATX command, or can be modified by any of the five modelling commands: MDROTX, MDROTY, MDROTZ, MDTRAN, and MDSCAL. When a modelling command is received, the modelling matrix is multiplied by a temporary matrix set up by the command. The temporary matrices used by the modelling commands are:

- the three rotation matrices (R_x, R_y, R_z)
- the translation matrix (T)
- the scaling matrix (S).

Since matrix multiplication is not commutative, the order in which modelling commands are sent will affect the form of the modelling matrix.

The default modelling matrix is the identity matrix. It can be reset to identity at any time by issuing the MDIDEN command. The modelling matrix can be read by issuing a MATXRD command with a parameter of 1.

Rotation and scaling transforms require an origin. This origin is the center of rotation, expansion, and contraction for graphic objects. The modelling origin is set using the MDORG command with the following format:

```
MDORG␣ox␣oy␣oz
```

The parameters form an x,y,z coordinate set that specifies the modelling origin with respect to the graphic object's original coordinates. For example, the default house is centered on {0, 50, 0}. The following command will set modelling origin to this point:

```
MDORG 0,50,0
```

The MDROTX, MDROTY, and MDROTZ commands are used to rotate graphic objects. They have the following formats:

```
MDROTX␣deg
MDROTY␣deg
MDROTZ␣deg
```

where deg is the angle of rotation to be performed. The sine and cosine of these angles are calculated and entered into the rotation matrices as shown below:

$$R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & sin\theta & 0 \\ 0 & -sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_y = \begin{pmatrix} cos\theta & 0 & -sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_z = \begin{pmatrix} cos\theta & sin\theta & 0 & 0 \\ -sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The right-hand rule for rotation is used. This rule defines the positive x, y, and z directions to follow the first finger, second finger, and thumb of a right hand when they are held at right angles to each other (see Figure 4.6). These axes rotate around the modelling origin, as illustrated in Figure 4.6.

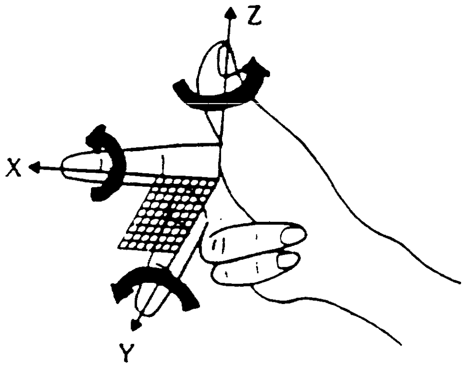


Figure 4.6: Rotation Direction

The default modelling matrix is an identity matrix which will not affect the graphic object. The MDIDEN command will reset the modelling matrix to identity. In the examples of modelling transforms, this command is used to reset the matrix so that the effects of one transform can be isolated from the others.

The following commands reset the modelling transforms, set the modelling origin and the rotation transforms, and then run command list number 100. When command list 100 is the house routine, the result will be as shown in Figure 4.7.

```
MDIDEN␣
MDORG␣0␣50␣0␣
MDROTX␣45␣
MDROTY␣45␣
MDROTZ␣45␣
CLRUN␣100␣
```

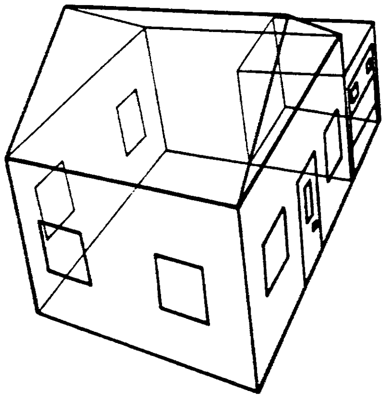


Figure 4.7: Rotation Example

The MDSCAL command is used to scale graphic objects. It has the following format:

```
MDSCAL␣sx␣sy␣sz
```

where sx, sy, and sz are entries in the scaling matrix with the following form:

$$S = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

This matrix is used to multiply the size of the graphic object along each axis by the corresponding parameter. For example, if sx is 2 the graphic object is doubled along its x axis. If sy is .5, the graphic object's size along the y axis is halved.

The MDTRAN command is used to move a graphic object from its drawing coordinates to a different position. The command format is as follows:

```
MDTRAN␣tx␣ty␣tz
```

where the parameters are values to be added to the x,y,z coordinates. These values are entered into the translation matrix in the following manner:

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{pmatrix}$$

The following command string makes two half-size copies of the house in different positions as illustrated in Figure 4.8:

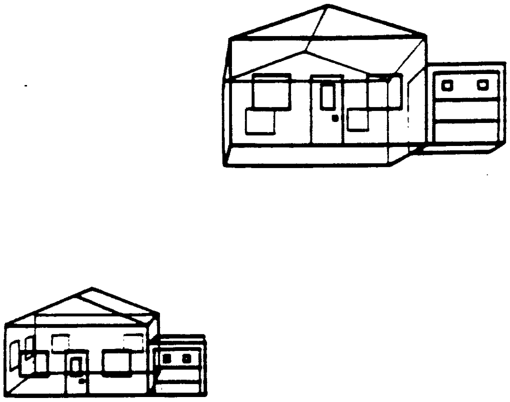


Figure 4.8: Translation Example

```
CLEARS␣0␣
MDIDEN␣
MDSCAL␣.5␣.5␣.5␣
MDTRAN␣50␣40␣50␣
CLRUN␣100␣
MDTRAN-150-150-150␣
CLRUN␣100␣
```

4.4.2.2 Viewing Transforms

The QG–640 uses a viewing transformation to position the center of the viewport with respect to the viewer. The viewing transformation establishes a viewing reference point, which is mapped into the center of the viewport. The viewer is positioned somewhere on the surface of a sphere that has its center at the viewing reference point, as illustrated in Figure 4.9. The radius of the sphere and the amount of the coordinate space that is mapped to the viewport are determined by the 3D to 2D transformation, which is described further on. Our examples up to this point have used the default viewing reference point and viewer position – the viewer reference point is in the center of the coordinate space and the viewer is looking down the positive z axis.

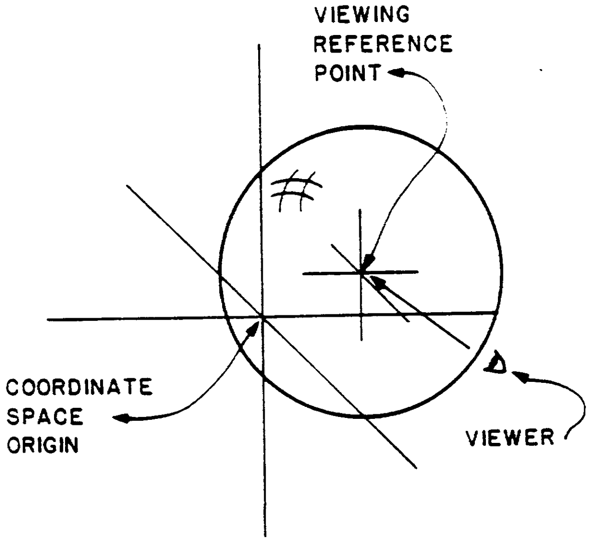


Figure 4.9: Viewing Reference Point

As is the case with the modelling transform, the viewing transform uses a master matrix (the viewing matrix). You can load the viewing matrix directly with the VWMATX command, or can alter various aspects of it with the viewing commands (VWRPT, VWROTX, VWROTY, VWROTZ). The viewing commands function like the modelling commands in that they set up temporary matrices that are used to multiply the viewing matrix. And like the modelling commands, the order in which they are used has an effect on the final view. You can read the current viewing matrix at any time by issuing the MATXRD command with a parameter of 2.

The VWIDEN command is similar to the MDIDEN command, and it is used in the examples to reset the viewing matrices to isolate the effects of the matrix that is being used.

The VWRPT command specifies the viewing reference point. It has the following format:

```
VWRPT␣x␣y␣z
```

where x, y, and z specify the point that is to be in the center of the field of view (the center of the viewport).

The VWROTX, VWROTY, and VWROTZ commands determine the position of the viewer on the viewing sphere. The command formats are as follows:

```
VWROTX␣deg
VWROTY␣deg
VWROTZ␣deg
```

where deg is the angle that the viewer is to be moved around the corresponding axis using the directions indicated in Figure 4.6. Note that the axes used by these commands are parallel to the coordinate system axes, but that their origin is at the viewing reference point. The QG–640 takes the sine and cosine of the angle and enters them into the viewing rotation matrices in the following format:

$$VWR_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$VWR_y = \begin{pmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$VWR_z = \begin{pmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The following list of commands clears the display, resets modelling and viewing transforms, sets the viewing reference point to 0,0,0 (the default value), and moves the viewer's position 90° around the Y axis so that the X axis is viewed from above. Command list number 100 is then run in order to draw a house. Figure 4.10 shows the result.

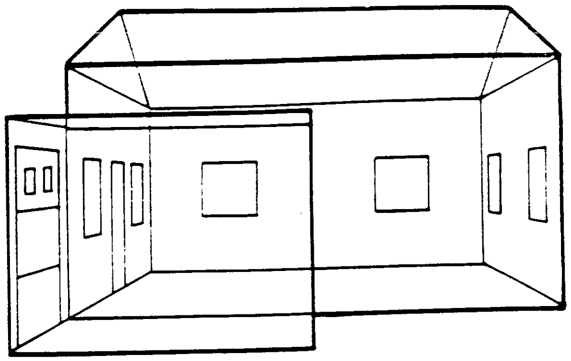


Figure 4.10: Viewing Transform Example

```
CLEARS␣0␣
MDIDEN␣
VWIDEN␣
VWRPT␣0␣50␣0␣
VWROTY␣90␣
CLRUN␣100␣
```

4.4.2.3 Hither and Yon Clipping

The WINDOW command, which was previously discussed, clips the sides of the drawing. There are commands to clip in front of a given plane and behind a given plane as well. These operations are referred to as hither and yon clipping respectively. To use hither or yon clipping, the clipping planes must be specified and the clipping enable flags must be set. The clipping planes are set with the following commands:

```
DISTH␣dist
DISTY␣dist
```

where dist in the DISTH command is the distance from the viewing reference point to the hither (foreground) clipping plane, and dist in the DISTY command is the distance from the viewing reference point to the yon (background) clipping plane. Negative values are closer to the viewer than positive values.

The commands that enable or disable clipping have the following format:

```
CLIPH␣flag
CLIPY␣flag
```

where flag is 1 (enables clipping) or 0 (disables clipping). CLIPH controls hither clipping and CLIPY controls yon clipping.

The following commands clear the screen, set the clipping planes and flags, and then run command list 100. The result is a house with the front and back clipped off (shown in Figure 4.11).

```
CLEARS␣0␣
VWRPT␣0␣0␣0␣
VWIDEN␣
DISTH-90␣
DISTY␣90␣
CLIPH␣1␣
CLIPY␣1␣
CLRUN␣100␣
```

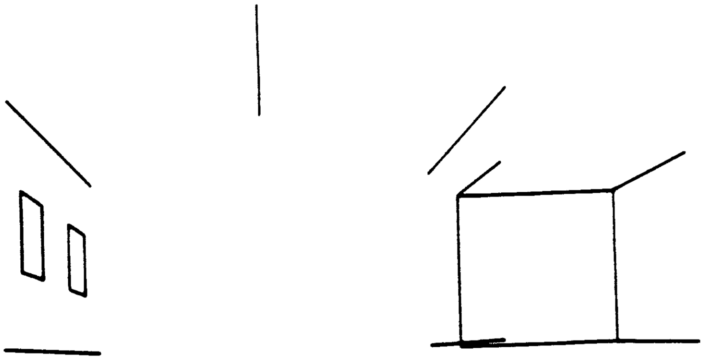


Figure 4.11: Clipping Example

Clipping should be disabled when it is not required; the extra calculations required when clipping is enabled decrease performance.

4.4.2.4 Three Dimensional to Two Dimensional Projection

In addition to the VWROT commands and the hither and yon clipping parameters, there are three other factors that affect the appearance of a 3D object on the screen:

- the distance of the viewer from the object
- the projection angle
- the current window position

The QG–640 projects the area around the viewing reference point onto the 2D coordinate space. The size of this area depends on two parameters: the viewing angle and the viewing distance as illustrated in Figure 4.12. The viewing angle specifies the number of degrees on the horizontal axis and the vertical axis of the viewer's field of view (default is 60°), centered on the viewing reference point, and the viewing distance is the distance that you are from the

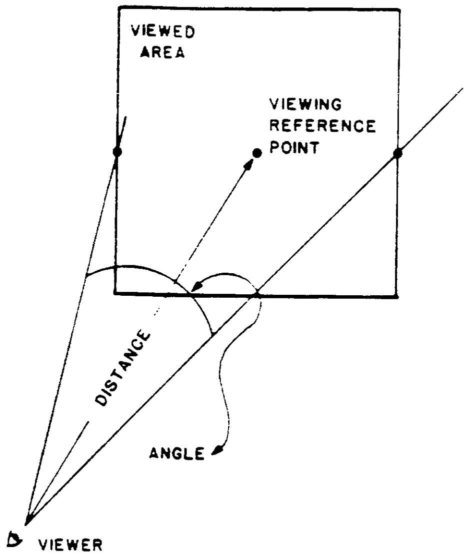


Figure 4.12: Viewing Angle and Viewing Distance

viewing reference point (default is 500). Using a camera as an analogue, the viewing angle would be determined by the type of lens (wide angle, narrow angle, etc.) and the viewing distance would be determined by the distance of the camera from the subject. If the viewing angle is larger, more of the 3D coordinate space is projected into the window. Likewise, if the viewer moves farther away from the viewing reference point, more of the 3D coordinate space is projected into the window.

The DISTAN command is used to specify the viewing distance. Its format is as follows:

DISTAN␣dist

where dist is the distance (specified in 3D coordinate point units) of the viewer from the viewing reference point.

The PROJCT command is used to set the viewing angle and the type of perspective that is to be used for the projection. Its format is as follows:

PROJCT␣angle

where angle is the number of degrees (horizontal and vertical) in a field of view with the viewing reference point at its center. An angle of 0°is a special case. It specifies an orthographic parallel (non-oblique) projection. When this type of projection is used, the viewing distance has no effect on the size of the picture.

The QG–640 uses the following formulas to convert 3D coordinates to 2D coordinates:

$$x_{2D} = \frac{1}{\text{dist} - z_{vw}} \times x_{vw} \times \frac{windowdiagonal}{2 \times tan\frac{angle}{2}}$$

$$y_{2D} = \frac{1}{\text{dist} - z_{vw}} \times y_{vw} \times \frac{windowdiagonal}{2 \times tan\frac{angle}{2}}$$

The QG–640 does not automatically map the view into the current window; however, the transfomations used do guarantee that the viewing reference point is mapped to the origin of the 2D virtual space. So if your window includes the 0,0 coordinate, you will see your viewing reference point on the screen, and you can adjust the window position as required to see any part of the object that is not in the window.

Window size, however, does not affect any of the projections except the 2D and 3D orthographic cases. That is to say, the window size is ineffective in displays with PROJCT angles greater than 0°.

This is because the 2D virtual coordinates from the equations above are next passed through another transform to bring them to screen coordinates. This final transform has the following form:

$$X_{scrn} = (X_{2d} - X - windowleft) \times \frac{(viewportsize)}{windowsize} + X_{viewportleftedge}$$

Substituting for $X - 2d$ and separating out the constant terms leaves:

$$X_{scrn} = \frac{1}{dist - Z_{vw}} \times \frac{windowdiagonal}{2 \times \tan(\frac{angle}{2})} \times \frac{viewportsize}{windowsize} + K$$

If the current window is close to being square, the ***windowdiagonal*** is close enough to the ***windowsize*** in both the x coordinate and y coordinate transforms so they will cancel out for all practical purposes.

Also note that since ***dist*** is in the denominator, larger distances give smaller screen images. Similarly, since the tangent of half the projection angle is in the denominator, when the angle is bigger, the screen image is smaller (especially for large angles).

The following command string uses the 3D to 2D transform to zoom in on the house as shown in Figure 4.13. The 3D to 2D transform converts the 3D coordinates to 2D coordinates, then the window to viewport mapping converts the 2D coordinates to screen coordinates.

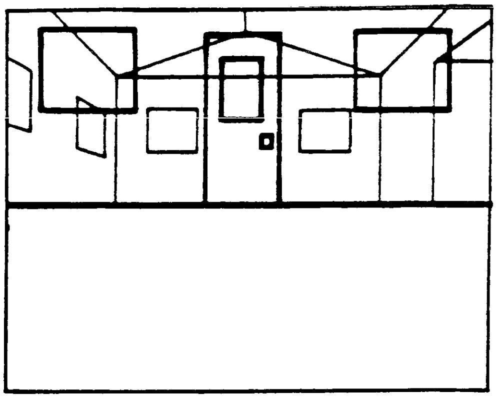


Figure 4.13: 3D To 2D Projection Example

```
CLEARS␣0␣
MDIDEN␣
VWIDEN␣
CLIPH␣0␣
CLIPY␣0␣
DISTAN␣300␣
CLRUN␣100␣
```

4.5 Graphic Attributes

After all of the transforms described in the preceding section have been performed, the resulting image is drawn by loading 8-bit color indices into pixel locations in the display buffer. The display buffer is a 640× 480 array of pixel locations that is mapped onto the display screen through a color lookup table. This lookup table determines the color that corresponds to each index. Figure 4.14 illustrates the relation of the display buffer to the screen.

When drawing an image in the display buffer, the color indices used depend on several graphics attributes. These attributes are:

- the current index
- the current line style
- the current drawing mode
- the current mask

4.5.1 Drawing Mode

The current drawing mode affects all the other modes. There are five drawing modes:

- Replace
- Complement
- OR
- AND
- XOR

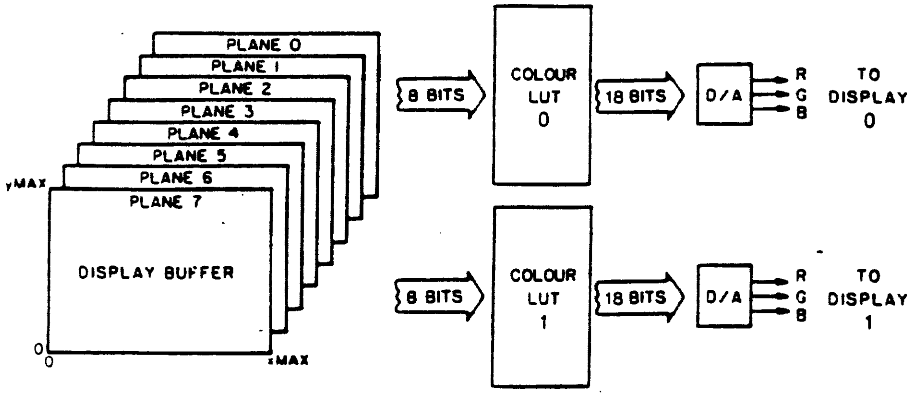


Figure 4.14: The Output Stage

You select the mode with the following command:

LINFUN␣mode

where mode is a Char from 0 through 4.

When Replace Drawing Mode is active, lines and fills are drawn by replacing the contents of pixel locations with the current index.

When Complement Drawing Mode is active, the QG–640 draws lines and fills by complementing the current contents of pixel locations. For example, the default contents of the display buffers is index 0 in all pixel locations; in Complement Drawing Mode, the QG–640 would draw a line on this background by changing the index of every pixel in the line to 255, since 255 (FF) is the complement of 0 (00). The advantage of this mode is that it allows individual graphic objects to be erased easily without affecting underlying graphic objects or the background. For example, to erase a line that was just drawn, we would merely redraw it, and it would be complemented back to its previous form. The disadvantage of Complement Drawing Mode is that the color displayed is affected by the underlying color.

The XOR Drawing Mode is a more general form of the Complement Drawing Mode and can be used for similar applications. It, however, allows more flexibility, since it XORs the current index with the current values of underlying pixels to obtain the new pixel values as a line is drawn. Drawing the same line twice in this mode results in no line, since the second line reverses the first.

The OR Drawing Mode ORs the current index with the current values in underlying pixels, and the AND Drawing Mode ANDs the current index with the current values in underlying pixels. The uses for these two drawing modes are less common; however, the experienced programmer should be able to put them to use in certain applications.

4.5.2 Color

You select the current index by issuing the COLOR command, which has the following format:

COLOR␣index

where index is a value from 0 to 255. A color index is not a color in itself; it is the address of a location in the lookup table. As the display buffer is scanned, the value in each pixel location is sent to the lookup table. The lookup table provides three values to the digital-to-analog converter. These values are used to generate the three analog signals to drive the red, green, and blue guns of the color display. Each lookup table location has 18 bits that are mapped into the digital-to-analog converter (D/A) inputs as indicated in Figure 4.15.

Referring to Figure 4.15, you will see that there are 64 intensity values for each of the three primary colors. The color that appears on the screen depends on the combination of these

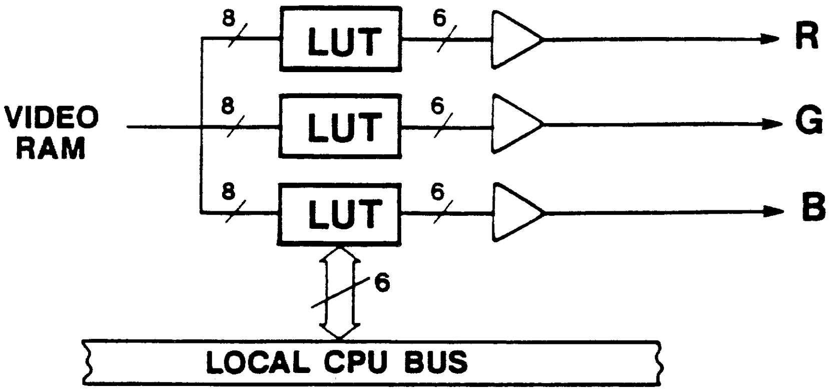


Figure 4.15: Lookup Table Bit Map

values. For example, a lookup table value of FF FF 00 generates bright yellow, 00 FF FF generates bright cyan, and 00 00 00 generates black.

The LUTX, LUT, and LUTINT commands allow you to load various color values into the lookup table. The LUTX and LUT commands write values into single lookup table locations, and the LUTINT command initializes the whole lookup table to one of several sets of predetermined values. The format of the LUTX command follows:

```
LUTX␣index␣r␣g␣b␣
```

where index is the index of a lookup table location, and r, g, and b are values from 0 to 255 specifying the intensity of the red, green, and blue elements respectively for that location. Due to the hardware configuration of the QG–640, only the high six bits of r, g, and b are used. The LUT command is similar to the LUTX command except that only the four low bits are loaded into the four high bits of the lookup table entry. The LUT is provided in order to maintain software compatibility with other Matrox products. The following LUTX command string sets lookup table location 4 to bright yellow:

```
LUTX␣4␣255␣255␣0
```

The following LUT command string will put bright yellow into the lookup table location 4:

```
LUT␣4␣15␣15␣0
```

The LUTINT command has the following format:

```
LUTINT␣set
```

where set is a number specifying one of several sets of values to be loaded into the lookup table. Table 4.1 lists these sets. (Refer to the appendices for lookup table contents.)

Set	Description
0	Color-cone
1	2 surface
2	rrgggbbb
3	rrrggbbb
4	rrrgggbb
5	6-level rgb
253	Alternate saved LUT
254	Saved LUT 1
255	Saved LUT 2

Table 4.1: List Of Lookup Table Value Sets

Set 0 has values that generate colors in the standard color cone used by graphic artists. The relationship between the color index and the color that is generated by it is arbitrary. The values of the predefined lookup table can be found in the appendices.

Sets 2 to 5 are arranged in such a way that there is a relationship between the format of the color index and the color that it generates. When Set 2, 3, or 4 is in the lookup table, the color index is divided into three binary numbers: a red number, a green number, and a blue number. The number of bits in each binary number depends on the lookup table set as shown below:

		76543210	bit
Set 2 index	=	rrgggbbb	
Set 3 index	=	rrrggbbb	
Set 4 index	=	rrrgggbb	

The value of these numbers determines the intensity of the red, green, and blue components of the color. The two-bit intensity values are related to the three-bit intensity values as shown in Table 4.2.

For example, if Set 2 is in the lookup table, index 63 (00111111) selects bright cyan.

When Set 5 is in the lookup table, the relationship of the index to the color selected is as follows:

$$index = (r \times 36) + (g \times 6) + b$$

where r, g, and b are intensity values from 0 through 5 for the color components of the selected color.

Set 1 has a special set of color values designed to provide two superimposed display surfaces. When Set 1 is in the lookup table, the index is divided into two subindices: ones in the low four bits select the underlying color, and ones in the high four bits select the overlying color. Zeroes in all four high bits makes the foreground surface transparent, allowing the underlying

Value		
2-Bit	3-Bit	Intensity
0	0	0
-	1	3
1	2	5
-	3	7
-	4	9
2	-	10
-	5	11
-	6	13
3	7	15

Table 4.2: 2-Bit/3-Bit Correspondence

surface to show through. In the subsection on 'Masking Bit Planes' which follows we explain how to use the MASK command to write to one surface or the other.

Sets 253, 254, and 255 load the lookup table with sets of lookup table values that you have previously saved using the LUTSAV and LUTSTO commands. The LUTSAV command, which has no parameters, saves the current contents of the lookup table to a special on-board memory buffer reserved for Set 255. The LUTSTO is similar to the LUTSAV command except that it allows two sets of lookup table contents to be stored. It has a parameter which specifies that the current lookup table be saved to Set 255 or to a second buffer reserved for Set 254. Subsequent LUTSAV and LUTSTO commands overwrite any lookup table sets that may have already been saved in the lookup table buffers.

You can read the contents of a lookup table location by issuing the LUTRD command or the LUTXRD command. These commands have the following formats:

LUTRD␣index and LUTXRD␣index

where index is a value from 0 to 255 specifying the lookup table location to be read. The QG–640 will copy the contents of the specified lookup table into the Data Out Register.

There are two output lookup tables on the QG–640, one for each output channel. The VDISP command enables you to select one of these lookup tables for the purpose of modifying it. PMASK is applied to the selected lookup table. The VDISP command has the following format:

VDISP flag

where flag is a Char equal to 0 or 1. The currently selected output lookup table can be determined using a FLAGRD 36 command. The output on J2 is affected when Char = 0, while the output on J1 is affected when Char = 1.

The QG–640 has two masks on the video data output to the lookup tables, one for each output. These masks are used to disable bit planes from the frame buffer to the lookup table. Each mask is set using the PMASK command which has the following format:

PMASK **mask**

where **mask** is the new value from 0 to 255 given to PMASK. Zeroes will prevent access to their corresponding bit planes and ones will permit access. The current values of PMASK can be read using a FLAGRD 37 command. A plane that is masked will always send a zero to the lookup table. Masked planes are those with a 0 in the corresponding PMASK bit.

Under certain conditions, primitives may generate both a background and a foreground. When a patterned line is drawn, for example, the pattern is made up of a foreground and a background, a character cell has a foreground and a background, and any command that produces filled areas creates a foreground and a background if the fill is in the form of a pattern. In such a case, using the COLMOD command specifies the color mode that determines whether the background is transparent or is the color last specified by the background color index. The background color is specified by the BCOLOR command.

The COLMOD command has the following format:

COLMOD␣**mode**

where **mode** is a Char equal to 0 or 1. When parameter **mode** is 0, the QG–640 is set to replace color mode and the background is set to the color specified by the BCOLOR command. When **mode** is 1, the QG–640 is set to foreground color mode and the background is drawn transparent.

The BCOLOR command has the following format:

BCOLOR␣**index**

where **index** is a Char from 0 to 255 specifying the background color index. For example, the following command sets the background index to 24 when COLMOD is set to 0:

BCOLOR␣24

4.5.3 Line Texture And Blinking Pixels

Lines can have texture as well as color. The texture is determined by the current line pattern, which you set with the LINPAT command. LINPAT has the following format:

LINPAT␣**pattern**

where **pattern** is a word with the line bit pattern. For example, the decimal value 61680 is equivalent to the binary value 1111000011110000. Issuing the following command:

```
LINPAT␣61680␣
```

causes lines to be drawn with four pixels in the current index alternating with four transparent pixels that allow the underlying index to show through (1 = current index, 0 = transparent).

Color indices can also be given a blink attribute to make them blink with the BLINKX command. It has the following format:

```
BLINKX␣index␣red␣green␣blue␣ontime␣offtime
```

where **index** specifies the lookup table index to blink. The parameters **red**, **green**, and **blue** are values from 0 through 255 that compose the color that the index is to blink to. The time that the affected pixels will be the blink color is specified by **ontime** in $\frac{1}{60}$ seconds. The time that the pixels are their normal color is set by **offtime** in $\frac{1}{60}$ seconds. Only the high six bits of each color entry are used. A similar command, BLINK, is provided for software compatibility with other Matrox products.

4.5.4 Masking Bit Planes

If you refer to Figure 4.14 again, you will note that the display buffer is composed of eight bit planes – one for each of the eight bits in the color index. The MASK command can mask off specified bit planes so that they cannot be overwritten when the QG–640 draws in the display buffer. The MASK command has the following format:

```
MASK␣planemask
```

where **planemask** is an eight-bit value (0-255). Zeroes will prevent access to their corresponding bit planes and ones will permit access. For example, the value 240 (11110000) masks access to the four least significant bit planes.

The mask allows the display buffer to be divided into different display surfaces. This is particularly useful when used in conjunction with the Set 1 lookup table values. For example, to superimpose the layers of artwork for a multilayer printed circuit board, you could draw one layer with the four lower bit planes masked off, and then mask off the high four bits and draw the second layer. The image already on the lower bit planes would not be affected.

4.6 Primitives

The QG–640 maintains two points, analogous to the position of a pen on a piece of paper, called the current points. These points are moved through the 2D and 3D coordinates spaces, respectively, to draw an image the same manner that a pen is moved over paper to draw an image. The commands that move the current points are called graphic primitives, and are explained in this section.

There are two main categories of graphics primitives: 2D and 3D. The command names in the two groups are similar. The 3D keywords are distinguished from their 2D counterparts by a 3 as the last character. Note, however, that not all the 2D primitives have 3D counterparts. In this section, all of the 2D primitives and then the 3D primitives are described.

4.6.1 Two Dimensional Primitives

When drawing on a piece of paper, the pen is not always on the paper. It must be raised and moved occasionally to start new lines. The same is true for drawing with the QG–640. The MOVE and MOVER commands are provided to move the pen in the 2D coordinate space without drawing. The format of the MOVE command is as follows:

MOVE␣x␣y

where x and y are Reals specifying a coordinate pair in 2D. This command moves the current point to the indicated point without drawing.

The format of the MOVER command is as follows:

MOVER␣Δx␣Δy

where Δx and Δy are Reals specifying the distance that the current point is to be moved from its current position. Note that the 'R' termination on this and other command names identify the command as using relative coordinates.

The POINT command draws a dot at the current point in the current index or complemented index, depending on the current drawing mode. The POINT command has no argument.

To draw a straight line (also called a vector), use either a DRAW or a DRAWR command. These commands have the same parameters as the MOVE and MOVER commands. Their effect is the same except that the DRAW and DRAWR commands draw lines from the old current point to the new current point.

The following example will clear the screen, move the current point to the center of the coordinate space, and then draw a point. The current point is moved again (using relative coordinates this time) and two lines are drawn – one using relative coordinates and the other using absolute coordinates. The result is illustrated in Figure 4.16.

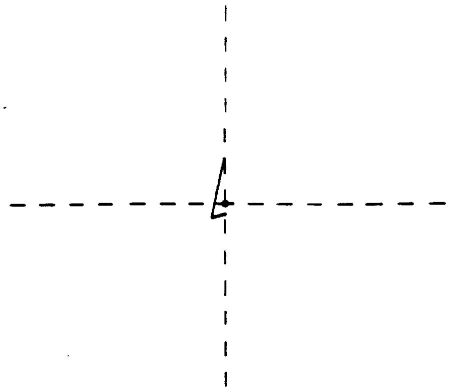


Figure 4.16: Example: Moves, Lines, And Points

```
COLOR␣24␣
CLEARS␣0␣
MOVE␣0␣0␣
POINT␣
MOVER␣0-10␣
DRAWR-20-5␣
DRAW␣0␣60␣
```

There are several graphic primitives that use a sequence of straight lines to draw polygons. These include the RECT, RECTR, POLY, and POLYR commands. RECT and RECTR draw rectangles. RECT uses absolute coordinates, and RECTR uses relative coordinates. The format for the RECT command is as follows:

```
RECT␣x␣y
```

with the rectangle being drawn with its diagonal from the current point to x and y, which are Reals. The current point does not move.

The format of the RECTR command is as follows:

```
RECTR␣Δx␣Δy
```

with the rectangle being drawn with its diagonal from the current point to Δx and Δy, which are Reals. The current point does not move.

The POLY and POLYR commands draw general polygons. The format of the POLY command is as follows:

```
POLY␣npts␣x1␣y1␣x2␣y2 ... xn␣yn
```

where npts is a value from 0 to 255 giving the number of vertices in the polygon. The remainder of the argument is a series of coordinate pairs specifying the positions of the vertices in the order that they are to be drawn. This command leaves the current point unchanged.

The POLYR command is similar. However, instead of absolute coordinates, it uses coordinates relative to the current point.

The following command string draws a rectangle using absolute coordinates, a rectangle using relative coordinates, a hexagon using relative coordinates, and then a hexagon using absolute coordinates. The result is shown, combined with the result of the previous example, in Figure 4.17.

```
MOVE␣20-50␣
RECT-20-60␣
MOVE␣60␣180␣
RECTR-120␣40␣
MOVE␣50␣180␣
POLYR␣6␣0␣0␣60-160-30-280-70-280-160-160-100␣0␣
POLY␣6␣30-55␣20-65-20-65-30-55-20-45␣20-45␣
```

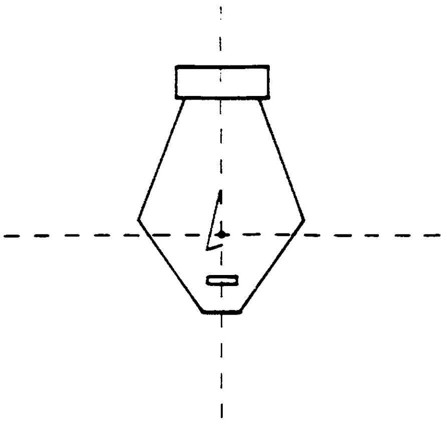


Figure 4.17: Example: Polygons

The QG–640 has three commands that can draw curved lines:

- CIRCLE, which draws a circle
- ARC, which draws an arc of a circle
- ELIPSE, which draws an ellipse

The format of the CIRCLE command is as follows:

```
CIRCLE␣radius
```

where radius is a Real specifying the radius of the circle to be drawn. The circle's center is on the current point.

The format of the ARC command is as follows:

```
ARC␣radius␣deg0␣deg1
```

where radius is a Real specifying the radius of the arc, deg0 is an Int giving the starting angle, and deg1 is an Int giving the ending angle. The starting angle and ending angle are measured in degrees counterclockwise from the positive x axis.

The ELIPSE command has the following format:

```
ELIPSE␣xradius␣yradius
```

where xradius is the distance from the center of the ellipse to its circumference on the x axis and yradius is the distance from the center to the circumference on the y axis. The center of the ellipse is on the current point.

There is a primitive that combines curved and straight lines; the SECTOR command. This command draws sections of circles shaped like pieces of pie. Its parameters are the same as those used by the ARC command. The SECTOR command, however, draws lines from the ends of the arc to the center of the arc.

The following command string draws two circles, two ellipses, two arcs, and two circle segments. Figure 4.18 shows these elements combined with the results of the two preceding examples.

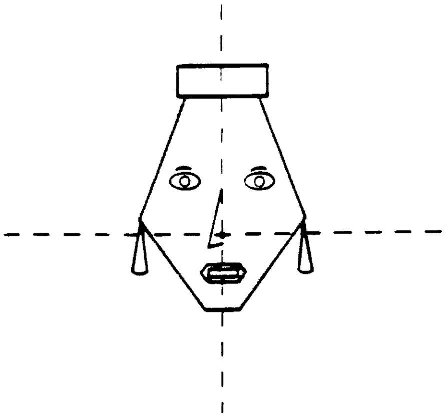


Figure 4.18: Example: Circles, Ellipses, Arcs, and Sectors

```
MOVE␣50␣70␣
CIRCLE␣10␣
ELIPSE␣30␣20␣
ARC␣30␣45␣135␣
MOVE-50␣70␣
CIRCLE␣10␣
ELIPSE␣30␣20␣
ARC␣30␣45␣135␣
MOVE␣110␣10␣
SECTOR␣60␣265␣275␣
MOVE-110␣10␣
SECTOR␣60␣265␣275␣
```

4.6.2 Three Dimensional Primitives

The QG–640 has the following 3D primitives:

```
MOVE3
MOVER3
POINT3
DRAW3
DRAWR3
POLY3
POLYR3
```

These commands function in the same way as their 2D counterparts. They, however, require an extra coordinate parameter: a coordinate for the z direction.

The following command string uses all the 3D primitives to draw the house shown in Figure 4.19. The three dots on the end of the roof are there only to illustrate the use of the POINT command.

```
CLEARS␣0␣
MOVE3-100-30␣50␣
POLYR3␣4␣0␣0␣0␣200␣0␣0␣200␣60␣0␣0␣60␣0␣
DRAWR3␣0␣0-100␣
POLYR3␣4␣0␣0␣0␣200␣0␣0␣200␣60␣0␣0␣60␣0␣
MOVE3␣-100␣30␣50␣
DRAWR3␣0␣0-100␣
MOVE3␣100-30␣50␣
DRAWR3␣0␣0-100␣
MOVE3␣100␣30␣50␣
DRAWR3␣0␣0-100␣
POLY3␣4␣100␣30-50␣100␣60␣0-100␣60␣0-100␣30-50␣
MOVE3-100␣30␣50␣
DRAW3-100␣60␣0␣
MOVE3␣100␣30␣50␣
DRAW3␣100␣60␣0␣
MOVE3␣100␣40-20␣
POINT3␣
MOVER3␣0␣0␣20␣
POINT3␣
MOVER3␣0␣0␣20␣
POINT3␣
```

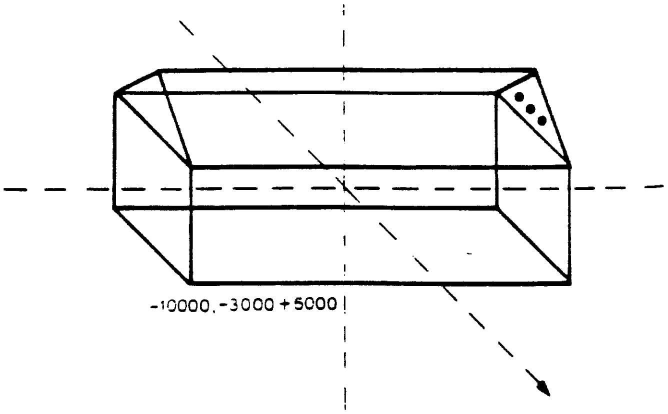


Figure 4.19: 3D Example

4.7 Fills

There are two methods to fill areas of the screen with solid colors and patterns: primitive fills and area fills. The primitive fill (PRMFIL) command fills closed primitives (polygons, ellipses, sectors, etc.) as they are drawn. This command has the following format:

```
PRMFIL␣flag
```

where flag is 0, or 1, and sets the current primitive fill flag. If the flag parameter is 0, closed primitives are drawn unfilled. If flag is 1, closed primitives are drawn filled. The primitive fill function works with both 2D and 3D filled primitives.

The following commands draw a box, using the PRMFIL command, to fill one side as illustrated in Figure 4.20:

```
CLEARS␣0␣
MOVE3-100-50␣50␣
POLYR3␣4␣0␣0␣0␣200␣0␣0␣200␣100␣0␣0␣100␣0␣
DRAWR3␣0␣0-100␣
PRMFIL␣1␣
POLYR3␣4␣0␣0␣0␣200␣0␣0␣200␣100␣0␣0␣100␣0␣
MOVE3␣-100␣50-50␣
DRAWR3␣0␣0␣100␣
MOVE3␣100␣50-50␣
DRAWR3␣0␣0␣100␣
MOVE3␣100-50-50␣
DRAWR3␣0␣0␣100␣
```

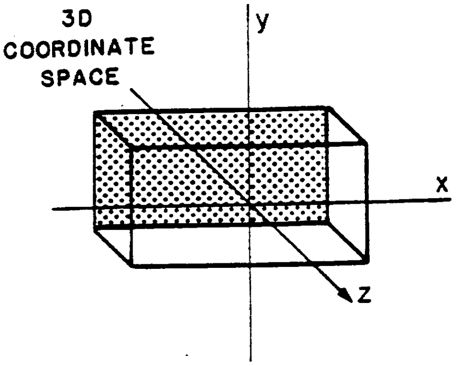


Figure 4.20: Primitive Fill Example

The primitive fill function is powerful and easy to use; however, it can only fill closed primitives. To fill other areas, one of two more general area fill commands can be used: AREA and AREABC. These commands, which function only in the 2D work space, fill outward from the current point until they reach a specified boundary. Their boundary is defined differently in each command. The AREA command has no parameters and fills with the current index outward from the current point until it encounters indices other than the current index or the index of the current point (see Figure 4.21`

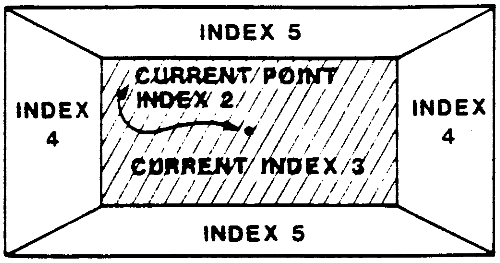


Figure 4.21: AREA Fill

The AREABC command specifies the boundary of the area to be filled. It has the following format:

AREABC␣bindex

where bindex is the index of the boundary to fill to.

The AREA and AREABC commands determine whether or not to continue filling by reading pixel indices and comparing them to the seed index, and either the index at the current point or the boundary index. The indices read are affected by both the mask set by the MASK command and the fill mask. The fill mask is active only during area fill operations. It is set by the FILMSK command, which has the following format:

FILMSK␣mask

where mask is an eight bit value (0-255) that is logically ANDed with pixel indices read during an area fill. The AND operation takes place before the indices are compared with the boundary index and either the current index (AREABC), or the current index and the index at the current point (AREA).

Both masks give flexibility in boundary specification. When the AREA command is used, these masks allow certain boundary colors to be ignored by masking them to look like either the current index or the index at the current point. When the AREABC command is used, the masks allow more than one index in the boundary to be used by making some indices look like the specified boundary index.

A fill does not have to be done with a solid color. The AREAPT command is provided so that you can specify a pattern composed of the filling index (COLOR) and the underlying index (BCOLOR). The command has the following format:

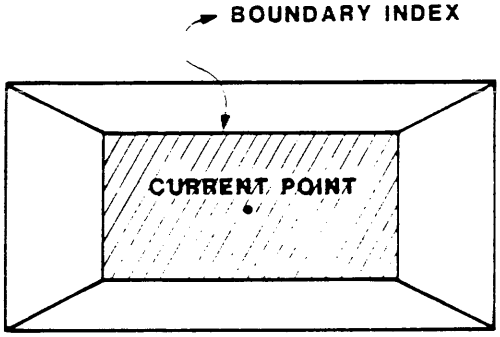


Figure 4.22: AREABC Fill

```
AREAPT␣pattern
```

where pattern is a 16-word array that forms a 16× 16 pixel bit mapped pattern. Zeroes in the bit map allow the underlying index (BCOLOR) to show through. The following command string defines the pattern shown in Figure 4.23.

```
AREAPT ␣1     ␣2     ␣4      ␣8
       ␣16    ␣32    ␣64     ␣128
       ␣256   ␣512   ␣1024   ␣2048
       ␣4096  ␣8192  ␣16384  ␣32768␣
```

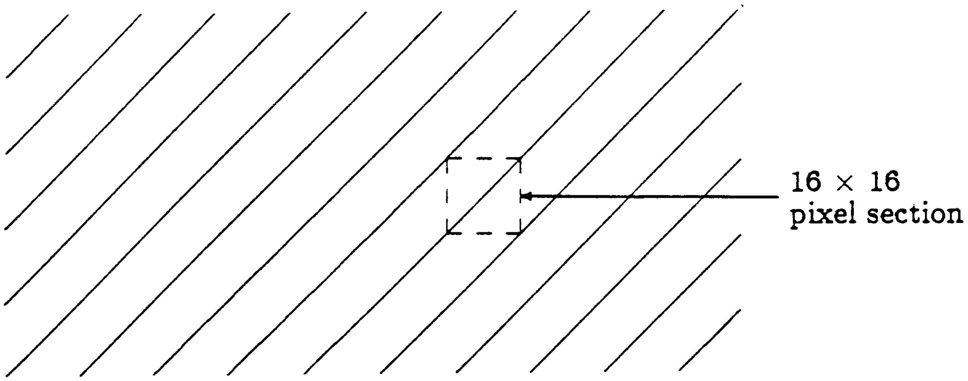


Figure 4.23: AREA Pattern Example

The AREA and AREABC commands can be used to fill 3D drawings after they have been projected onto the 2D coordinate space. The CONVRT command will map the 3D current point to the 2D current point. Thus the 2D current point can be positioned in the area that you wish to be filled.

The following command string draws a tetrahedron, illustrated in Figure 4.24, and fills one side.

```
CLEARS␣0␣
COLOR␣24␣
MOVE3␣0␣100␣0␣
DRAW3␣100-60␣0␣
DRAW3-100-60␣0␣
DRAW3␣0␣100␣0␣
DRAW3␣0␣0␣170␣
DRAW3-100-60␣0␣
MOVE3␣0␣0␣170␣
DRAW3␣100-60␣0␣
MOVE3-10␣0␣0␣
CONVRT␣
COLOR␣70␣
AREABC␣24␣
```

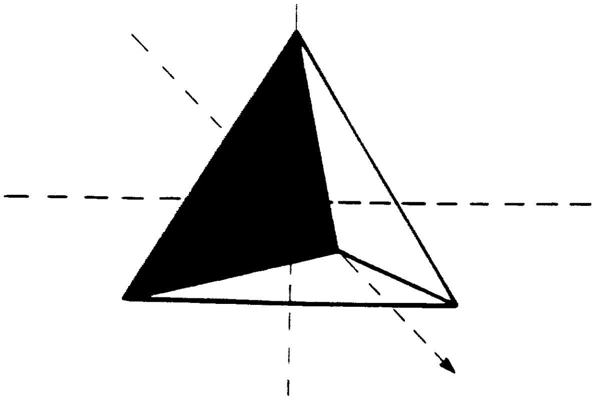


Figure 4.24: AREABC Fill Example

4.8 Text

The following commands are provided to draw text:

TEXT	Draws text using standard font.
TEXTP	Draws text using user font.
TEXTC	Draws text using standard fonts (Hex mode only).
TEXTPC	Draws text using user fonts (Hex mode only).
TSTYLE	Selects fat and raster text or thin and vector text.
TDEFIN	Defines raster text characters for user font.
GTDEF	Defines vector text characters for user font.
TJUST	Sets text justification relative to current point.
TSIZE	Sets text size.
TASPCT	Sets text aspect ratio.
TANGLE	Sets drawing angle.
TCHROT	Sets character rotation.
RDEFIN	Defines raster text characters for user fonts 1 to 15.
RFONT	Selects active user raster font.

There are two character fonts available: the standard font and the user font. Each of these fonts has two different styles of text. The standard font has thin text or fat text, and the user font has raster text or vector text.

The TEXT and TEXTP commands, followed by a text string, are used to write on the screen. The TEXT command has the following format:

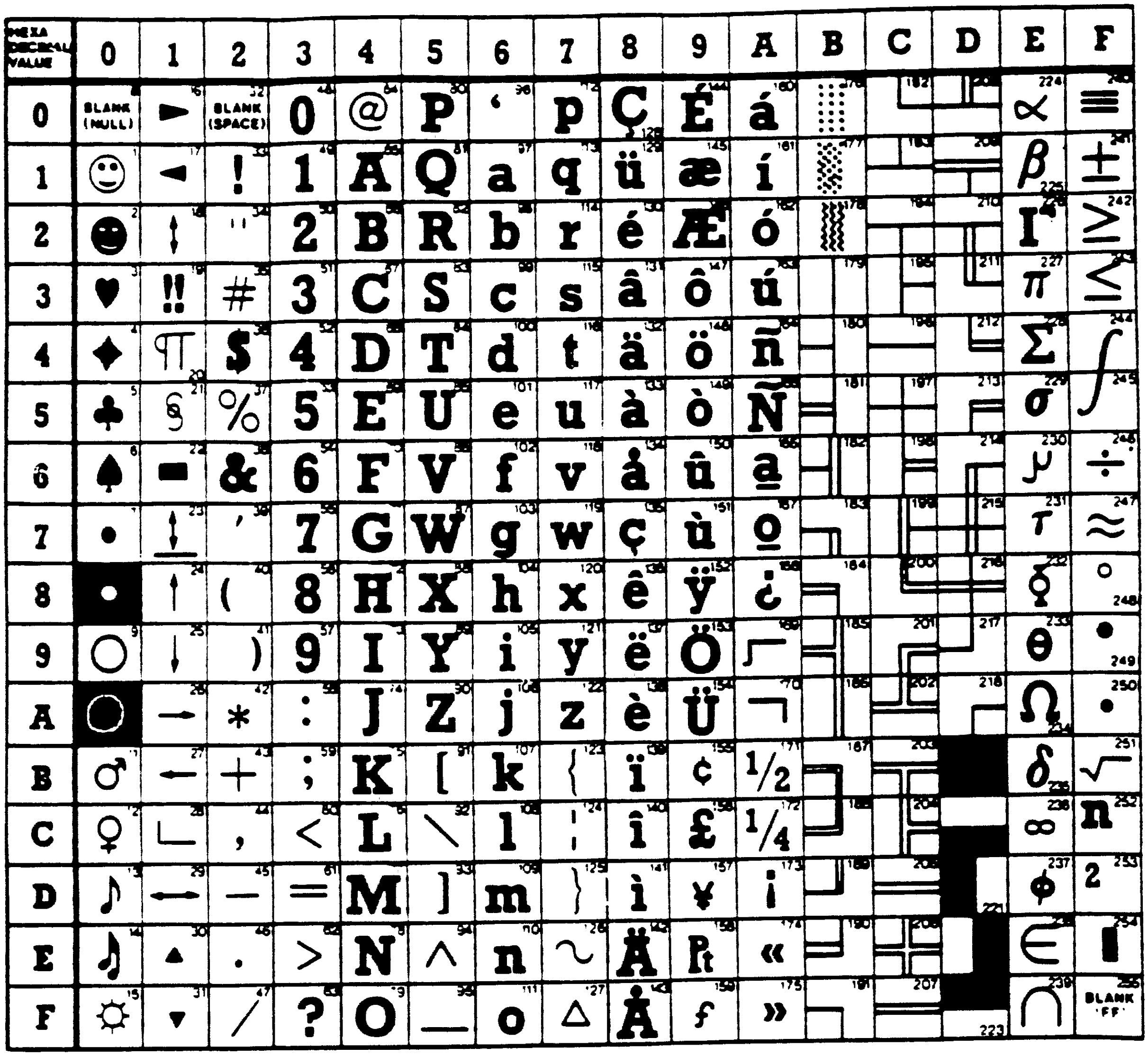
TEXT␣string

where string is a string of characters enclosed by either single or double quotes. The QG–640 draws the string with the standard character font (Figure 4.25) justified about the 2D current point as specified by the most recent TJUST command. The TEXTP command is similar, but it uses the user font defined by the TDEFIN and GTDEF commands.

The TEXTC and TEXTPC commands, followed by the string count and the text string, are used to display the specified text string. These commands are used in hex format only. The TEXTC command has the following format:

TEXTC␣count␣char␣char␣ ... char

where count specifies the number of characters in the string, and each char is a string character.



HEXA DECIMAL VALUE	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	BLANK (NULL)	►	BLANK (SPACE)	0	@	P	`	p	Ç	É	á	░	└	╨	α	≡
1	☺	◄	!	1	A	Q	a	q	ü	æ	í	▒	┴	╤	β	±
2	☻	↕	"	2	B	R	b	r	é	Æ	ó	▓	┬	╥	Γ	≥
3	♥	‼	#	3	C	S	c	s	â	ô	ú	│	├	╙	π	≤
4	♦	¶	$	4	D	T	d	t	ä	ö	ñ	┤	─	╘	Σ	⌠
5	♣	§	%	5	E	U	e	u	à	ò	Ñ	╡	┼	╒	σ	⌡
6	♠	▬	&	6	F	V	f	v	å	û	ª	╢	╞	╓	µ	÷
7	•	↨	'	7	G	W	g	w	ç	ù	º	╖	╟	╫	τ	≈
8	◘	↑	(	8	H	X	h	x	ê	ÿ	¿	╕	╚	╪	Φ	°
9	○	↓	)	9	I	Y	i	y	ë	Ö	⌐	╣	╔	┘	θ	•
A	◙	→	*	:	J	Z	j	z	è	Ü	¬	║	╩	┌	Ω	•
B	♂	←	+	;	K	[	k	{	ï	¢	½	╗	╦	█	δ	√
C	♀	∟	,	<	L	\	l	¦	î	£	¼	╝	╠	▄	∞	ⁿ
D	♪	↔	-	=	M	]	m	}	ì	¥	¡	╜	═	▌	φ	2
E	♫	▲	.	>	N	^	n	~	Ä	₧	«	╛	╬	▐	∈	■
F	☼	▼	/	?	O	_	o	△	Å	ƒ	»	┐	╧	▀	∩	BLANK 'FF'

Figure 4.25: The Standard Font

The TEXTPC command is similar to the TEXTC command except that it draws a programmable text string at the current point.

The TJUST command sets the position of the text to the left of, to the right of, or centered on the current point. It also sets the position of the text above, below, or centered on the current point (see Figure 4.26). The command format is as follows:

```
TJUST␣horiz␣vert
```

where **horiz** and **vert** specify the position of text as follows:

horiz

1 Start of text line is at the current point.
2 Center (horizontally) of text line is at the current point.
3 End of text line is at the current point.

vert

1 Bottom of text is at the current point.
2 Center (vertically) of text is at the current point.
3 Top of text is at the current point.

```
*MATROX   MAT*ROX   MATROX*
*MATROX   MAT*ROX   MATROX*   * current point
*MATROX   MAT*ROX   MATROX*
```

Figure 4.26: Justification Options

With the standard font, either fat or thin text can be selected with the TSTYLE command. Slim text characters are always one pixel wide irrespective of their size. The lines that make up fat characters, on the other hand, become wider as the characters become larger. Fat style characters are the same as slim characters when the default text size is used; the scaling effect becomes noticeable only as text sizes become larger.

If you use the user font, you can use either vector text or raster text, provided that you have created the characters that you want to use. Use the GTDEF command to create vector text characters and use the RDEFIN or TDEFIN commands to create raster text characters. Note that whereas fat and thin characters with the same code coexist, raster text characters and vector text characters with the same code do not. That is to say that you cannot create both a vector text character and a raster text character for the same font position. If you attempt to display a character that you have not defined, the HLGE will use the corresponding standard font thin character.

Subsection 4.8.2 explains how to define characters for the user font.

4.8.1 Character Attributes

Text has the following attributes:

Attribute	Command
Color	COLOR
Angle	TANGLE
Rotation	TCHROT
Size	TSIZE
Aspect ratio	TASPCT

	STANDARD FONT		USER FONT	
	THIN TEXT	FAT TEXT	VECTOR TEXT	RASTER TEXT
TANGLE	variable	variable	variable	0°
TCHROT	variable	0°	variable	0°
TSIZE	variable	variable	variable	as defined
TASPCT	variable	1.5	variable	as defined

Table 4.3: Character Attribute Use Restrictions

All text is drawn in the current color, set by the COLOR command. However, not all other attributes are variable with all the other text types. Table 4.3 indicates what the restrictions are. The TSIZE and TASPCT commands set the size and aspect ratio of the text characters. The format of the TSIZE command is as follows:

TSIZE␣size

where size specifies the number of horizontal coordinate space points between the start of one character and the next. The height of the characters is determined by the aspect ratio command, which has the following format:

TASPCT␣ratio

where ratio is character cell height divided by character cell width. Setting the width to 10 and aspect ratio to 1 produces character cells 10× 10 points in size. The aspect ratio of the characters on the screen also depends on the window to viewport map and the size of the characters.

The TANGLE and TCHROT commands change the angle of the text in various ways. The TANGLE command sets the angle of the text string, and the TCHROT command rotates the characters about their centers. Thus, both types of slanted text shown in Figure 4.27 can be easily produced, as well as variations in between. For both commands, the command argument is an angle from the x axis in counterclockwise direction.

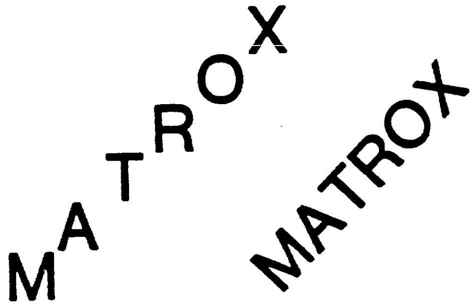


Figure 4.27: Slanted Text

The following command string draws large (50 pixels wide) thin angled characters rotated and centered on the current point. The standard character set and an aspect ratio of 1 are used. The result is illustrated in Figure 4.28.

```
CLEARS␣0␣
TJUST␣2␣2␣
TSIZE␣50␣
TSTYLE␣1␣
TANGLE␣45␣
TCHROT␣45␣
TASPCT␣1␣
MOVE␣0␣0␣
TEXT␣'QG–640'␣
```

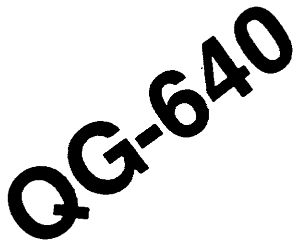


Figure 4.28: Text Example

4.8.2 Defining Characters for the User Font

At reset, the user font is empty, but characters can be defined in it by the RDEFIN command, TDEFIN command, or the GTDEF command.

Characters created with the GTDEF command, and the characters in the standard thin font, are formed from vector command lists similar to the command lists used to save graphics commands. These vector commands provide the basis for rotation, scaling, and translation. The format for the GTDEF command is as follows:

```
GTDEF␣character␣n␣width␣array
```

where character is a number from 0 to 255 identifying the character, n is the number of values in array, width is the width of the character in character vectors, and array is an array of vector parameters. Each entry in array gives a direction, a distance, and a draw/move flag. In ASCII Command Mode, character, n, and width are Ints and each vector parameter in array is composed of three Ints. In Hex Command Mode, character, n, and width are byte values and each vector parameter in array is composed of a single value. The format of the vector parameters and their direction code are shown in Figures 4.29, 4.30, and 4.31.

For example, the following code defines an 'A' in ASCII Mode:

```
GTDEF␣65␣7␣8␣
1␣7␣2␣
1␣2␣1␣
1␣3␣0␣
1␣2␣7␣
1␣7␣6␣
0␣4␣2␣
1␣7␣4␣
```

and in Hex mode:

```
89 42 07 08 74 51 58 57 7E 22 7C
```

The QG-640 allows you to define up to 16 raster fonts in memory, labeled from 0 to 15. The raster characters are bit maps and cannot be transformed, so you must define them as you want to see them.

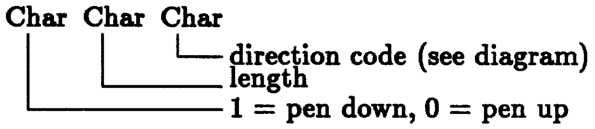


Figure 4.29: ASCII Command Mode Vector Parameter Format

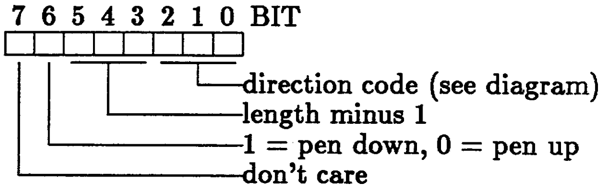


Figure 4.30: Hex Command Mode Vector Parameter Format

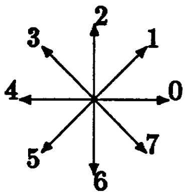


Figure 4.31: Vector Parameter Direction Codes

User Raster Font 0

User raster font 0 characters are defined using the TDEFIN command. For this font, each character must be defined separately. The maximum cell size of these characters is 255×255 pixels. This font is the PGC compatible user definable raster font.

The TDEFIN characters are bit maps and cannot be transformed, so you must define them as you want to see them. The command format is as follows:

```
TDEFIN␣n␣x␣y␣array
```

where n is a number from 0 to 255 identifying a character position in the font, x is the character cell width in screen coordinates, y is the character cell height in screen coordinates, and array is an array of bytes that forms the bit map of the character being defined.

User Raster Fonts 1 to 15

User raster fonts 1 to 15 are special fonts optimized for drawing speed. Each font must be defined "a complete font at a time", using the RDEFIN command. All characters in a given font of this type must have the same cell dimension; the maximum size is 16×16 pixels.

User Raster Font Selection

Only one of the 16 user raster fonts can be active at any one time. The raster font used to draw characters (0 to 15), with the TEXTP and TEXTPC commands, is selected using the RFONT command. This command also specifies the aspect ratio of the characters drawn, with a choice of any combination of single/double height and single/double width.

The following command string creates the character shown in Figure 4.32 and assigns it to character "A" (code 65).

```
TDEFIN␣65␣8␣4␣
1␣1␣1␣1␣0␣0␣0␣0␣
1␣0␣0␣1␣0␣0␣0␣0␣
1␣0␣0␣1␣0␣0␣0␣0␣
1␣1␣1␣1␣1␣1␣1␣0␣
```

Figure 4.32: TDEFIN Example

4.9 Command Lists

A command list is a list of QG–640 commands stored in memory. You define a command list using a CLBEG command followed by the command list terminated with a CLEND command. The format of the CLBEG command is as follows:

```
CLBEG␣clist
```

where clist is a number from 0 to 255 identifying the command list. The CLEND command has no argument.

Once defined, a command list can be run by issuing either a CLRUN command or a CLOOP command. The CLRUN runs a command list once; the CLOOP command runs a command list a specified number of times. The format of the CLRUN command is as follows:

```
CLRUN␣clist
```

where clist is a number from 0 to 255 identifying the command list that is to be run.

The format of the CLOOP command is as follows:

```
CLOOP␣clist␣count
```

where clist is a number from 0 to 255 identifying the command list to be run, and count is a number from 0 to 65535 specifying the number of times the command list is to be repeated.

The following commands define a command list and run it twice. Because the last two commands in the command list change the modelling transform, the loop gives two different views of the same object (shown in Figure 4.33). Nothing will be drawn on the screen until a CLRUN is issued.

```
CLEARS␣0␣
CLBEG␣1␣
MOVE3-100␣50␣0␣
POLYR3␣4␣0␣0␣0␣200␣0␣0␣200␣50␣0␣0␣50␣0␣
DRAWR3␣0␣0␣100␣
POLYR3␣4␣0␣0␣0␣200␣0␣0␣200␣50␣0␣0␣50␣0␣
MOVE3-100␣100␣0
DRAWR3␣0␣0␣100␣
MOVE3␣100␣100␣0␣
DRAWR3␣0␣0␣100␣
MOVE3␣100␣50␣0
DRAWR3␣0␣0␣100␣
MDROTY␣45␣
MDTRAN␣0-125␣0␣
CLEND␣
MDIDEN␣
CLOOP␣1␣2␣
```

Once a command list has been defined, it can be user-read and modified. The CLRD command allows you to read the specified command list. The CLMOD command allows you to modify a command list. The CLRD function has the following format:

```
CLRD␣clist
```

where clist is the name of the command list to be read. The command list is sent, in hexadecimal, to the Data Out Register. The data consists of one word giving the number of bytes in the list, followed by the command list. The CLRD command is used to locate the offset of the bytes to be replaced. To remove a byte without replacing it, use a NOOP command to fill its place.

The CLMOD command is used to edit command lists. It replaces a section of a command list with an array of bytes provided in the command argument. The command has the following format:

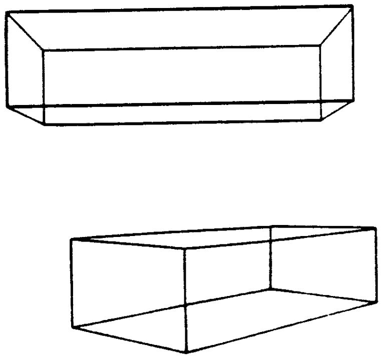


Figure 4.33: Command List Example

```
CLMOD␣clist␣offset␣nbytes␣bytes␣
```

where clist is the command list to be modified, offset is the offset in bytes from the start of the command list to the start of the section to be replaced, nbytes is the number of bytes to be replaced, and bytes is an array of replacement bytes. The number of bytes in the replacement array (bytes) must be exactly the same as the number of bytes in nbytes.

When using the CLMOD command, keep in mind that real coordinates (32 bits) are not stored in memory in the same order as they are received from the Host. When you specify a real number, it is in the form of:

[low integer][high integer][low fraction][high fraction]

The previous form is received by the Host and stored in memory in the following form:

[low fraction][high fraction][low integer][high integer]

When a coordinate is stored in a command list, the firmware exchanges the bytes so that the fractional part is stored first. When a CLRD command is processed, a reverse exchange is made so that coordinates appear just as they were sent.

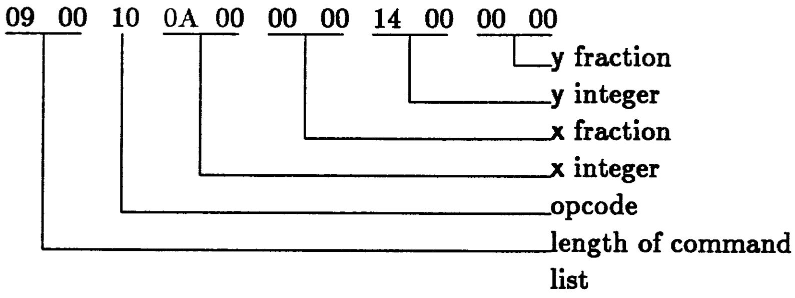
Using the CLMOD command on a section of a real coordinate, stored in a command list, performs no exchange. Therefore:

- Modifying the first byte of a coordinate modifies its [low fraction].
- Modifying the second byte of a coordinate modifies its [high fraction].
- Modifying the third byte of a coordinate modifies its [low integer].
- Modifying the fourth byte of a coordinate modifies its [high integer].

For example:

```
CLBEG␣1␣
MOVE␣10␣20␣
CLEND␣
CLRD␣1␣
```

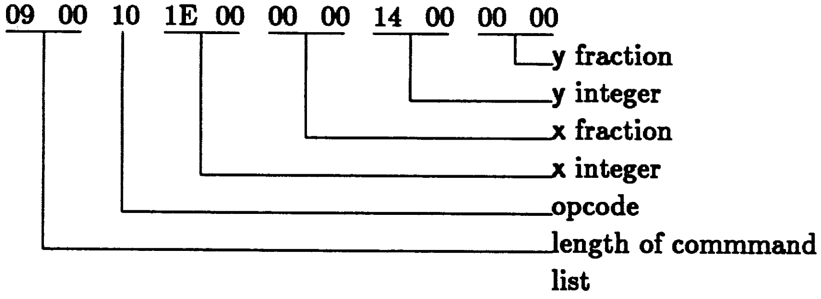
The Data Out Register contains:



CLMOD␣1␣3␣1␣30␣
CLRD␣1␣

The previous CLMOD command modified the third byte in clist, which is the low byte of the integer part of x.

The Data Out Register contains:



The modified byte **seems** to be the second byte in the command list, but in fact, it is the third byte because the CLRD command exchanges real coordinates.

4.10 Direct Screen Operations

4.10.1 Drawing

The QG–640 has a number of commands which allow you to bypass the normal coordinate space/transform sequence and work directly in the display buffer. The 'S' series commands are graphics primitives that draw directly on the screen. They are the same as the 2D primitives except that they use screen coordinates instead of 2D coordinates. They are faster than the 2D primitives. Pictures created with the 'S' series commands are clipped to the current viewport, and the end points of lines are not drawn. For the 'S' series primitives to function properly, the window and viewport must have exactly the same coordinates. This means that the window and the viewport must be set to equal the x and y capacity of the Matrox board. Any direct screen operations must be performed only on points visible in the currently displayed window/viewport. The screen coordinates upon which you want to operate must be:

$$0 \leq x \leq 639$$
$$0 \leq y \leq 479$$

For example, if the viewport is full screen, the window must have corners at 0,0 and 639,479. The "S" commands are listed below:

SARC
SCIRC
SDRAW
SDRAWR
SELIPS
SMOVE
SMOVER
SPOLY
SPOLYR
SRECT
SRECTR
SSECT

The PDRAW command provides the ability to perform a series of moves and line draws in direct screen mode with a single command. The command format is as follows:

$$\text{PDRAW}\sqcup x_1 \sqcup y_1 \sqcup x_2 \sqcup y_2 \; \ldots x_n \sqcup y_n \sqcup$$

where x and y are Int screen coordinates. The most significant bit of the y coordinate is used to specify either a move or a draw, and the most significant bit of the x coordinate is used to specify either to continue or to stop.

4.10.2 Pixel Moves

The IMAGER and IMAGEW commands transfer lines or parts of lines of pixel values between the system memory and the display buffer. The RASTRD and RASTWR commands move rectangular sections of the display buffer to and from the system memory. The RASTOP command moves rectangular sections of the display memory from one section of the display memory to another performing an optional logical function.

The IMAGER command has the following format:

```
IMAGER␣line␣x1␣x2
```

where line is a y coordinate indicating a horizontal line of pixels in the screen coordinate space, x1 is an x coordinate indicating the starting point on the line, and x2 is an x coordinate indicating the end point (included) on the line. The specified pixel values are copied to the Data Out Register. The data format depends on whether the QG–640 is in ASCII Mode or Hex Mode.

In ASCII Mode, a line is returned in the following format:

```
IW,line,x1,x2,x,x,x ... (CR)
```

where 'IW' is a header identifying the string as the result of a IMAGER command, line is the line number, x1 and x2 specify a line segment, the x's represent ASCII decimal color indices separated by commas, and the carriage return (CR) ends the string.

In Hex Mode, the data is run-length encoded. Two or more contiguous pixels having the same index are encoded in two bytes: the first with the number of pixels less one, and the second with the index. When two or more contiguous pixels have different indices, the number of pixels less one is sent in a byte with the most significant bit set, then the values of the indices for each pixel are sent in a series of separate bytes. Since the most significant bit in the initial byte is used to differentiate the two types of code, only seven bits remain to give the number of pixels in the series, limiting the number to 128. For example, a series of pixels with the values 1 1 1 1 2 3 4 5 5 5 5 6 7 would be encoded as 03 01 82 02 03 04 03 05 81 06 07.

The IMAGEW command has the following format:

```
IMAGEW␣line␣x1␣x2␣data␣
```

where line, x1, and x2 specify a set of pixels in the same way as the IMAGER command, and data is the pixel data that is to be written into the specified pixels. The data format is the same as that used for the IMAGER command.

Although the RASTRD and RASTWR commands also transfer data directly between the display memory and the system memory, they differ from IMAGER and IMAGEW in that they simulate a raster scan of a specified rectangular area, incorporating certain logical functions. Run-length encoding is not used. The format of the RASTWR command is as follows:

```
RASTWR␣oper␣dir␣x0␣y0␣x1␣y1
```

where oper specifies a logical operation (see Table 4.4), dir specifies a subset of the major and minor scan directions (see Table 4.5), x0,y0 specify, in screen coordinates, one corner of the rectangle to be scanned, and x1,y1 specify the diagonally opposed corner.

Raster Functions	
Function Code	Operation
0	Copy
1	OR
2	AND
3	XOR

Table 4.4: Logic Operations

Scanning Direction		
Direction	Major Direction	Minor Direction
0	⇒	↑
1	⇒	↓
2	⇐	↑
3	⇐	↓

Table 4.5: Scan Directions

The QG–640 scans by reading a line of pixels along the major scan direction, moving one scan line in the minor direction and then repeating the process. As each pixel is passed in the scan, the specified logical operation is performed on the data from the FIFO and the data in the pixel location. The result is then written into the location. Figure 4.34 shows a typical scan.

The RASTRD command is the same as the RASTWR command except that it copies data from the scanned area to the output port with no logical operations performed on the data. In both commands, each index is passed in a single byte and until the transfer is completed, no other commands are interpreted by the QG–640. The number of bytes transferred is $(x_1 - x_0 + 1) \times (y_1 - y_0 + 1)$.

The following command string XORs data from the FIFO with data in the specified rectangle and writes the results into the rectangle. Figure 4.34 shows the scan directions.

```
RASTWR␣3␣1␣100␣100␣400␣300␣
```

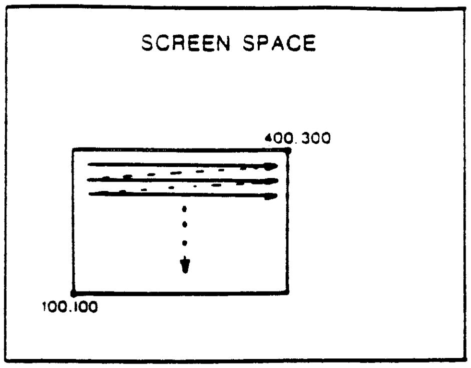


Figure 4.34: Raster Scan

The QG–640 has a third raster command, RASTOP, which uses the same general format to copy rectangular areas from one part of the screen to another. It has the following format:

RASTOP␣oper␣srcdir␣destdir␣x_0␣x_1␣y_0␣y_1␣x'_0␣y'_0

where oper specifies a logical operation, srcdir is the scan direction in the source rectangle, destdir is the scan direction in the destination rectangle, x_0, x_1, y_0, and y_1 specify the source rectangle, and x'_0, y'_0 specify the lower left corner of the destination rectangle.

The following command string copies the contents of the left rectangle in Figure 4.35 to the right rectangle. Note that by using different source and destination scan directions, a mirror image was drawn.

RASTOP␣0␣1␣3␣100␣100␣300␣300␣400␣100␣

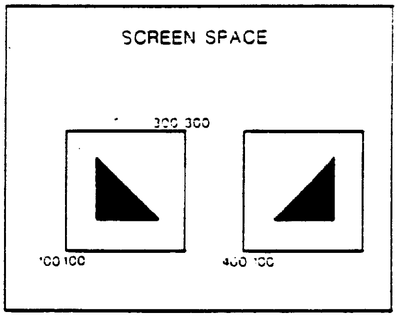


Figure 4.35: RASTOP Example

4.11 Read Back Commands

The QG–640 supports a number of read back commands that allow you to determine the exact values of the QG–640's parameters. The read back commands are:

- Command List Read (CLRD)
- Flag Read (FLAGRD)
- Image Read (IMAGER)
- LUT Read (LUTXRD and LUTRD)
- Matrix Read (MATXRD)

These commands are detailed in the command summary chapter.

When a read back command is executed, the QG–640 puts the requested information in the Data Out Register. When in ASCII mode, the data is returned as ASCII decimal numbers terminated by a carriage return. Some commands return multiple values; the individual command descriptions give the data formats in both ASCII and Hex communication modes.

Note that if a read back is requested and the Data Out Register is full, the QG–640 will halt and wait for you to read the register.

4.12 Error Handling

If you have set the Error Enable Flag in the communications area, the QG–640 will return error codes in the current communication mode. The QG–640 will return the error code as an ASCII character when in ASCII mode and as a hex value in a single byte when in Hex mode. The return messages and codes are summarized in Table 4.6.

Code	Meaning
0	Parameter out of range.
1	Wrong data type – need integer.
2	Ran out of memory.
3	Arithmetic overflow.
4	Wrong data type – need digit.
5	Opcode not recognized.
6	Recursion in command list.
7	Command lists nested more than 16 deep.
8	String or command list too long.
9	Area fill error.
A	Missing parameter.

Table 4.6: Summary of Error Codes and Messages

4.13 Graphics Input Support

Many applications require the use of a graphics input device such as a mouse, joystick, or trackball. This input device is used by the software to move a cursor, to frame areas of text, to draw lines, or to implement some other function. In a computer-aided design program, a mouse might be used to move a cursor and specify points to be interconnected.

The QG–640 provides the following three commands to help the programmer implement graphics input functions:

XHAIR
XMOVE
RBAND

XHAIR displays a graphic cursor, XMOVE moves the cursor, and RBAND enables either the rubber band vector or the rubber band rectangle. **All three commands draw directly to the screen and do not affect the screen space (frame buffer) in any way that would affect the user.**

XHAIR has the following format:

```
XHAIR␣flag␣xsize␣ysize
```

where **flag** determines the type of cursor displayed, which is located at the current cursor position. The size of the graphic cursor is set by **xsize** and **ysize**. Parameter **flag** is a Char having the following meaning:

Flag	Action
0	Disable graphic cursor
1	Enable cross hair cursor, clipped on screen
2	Not supported
3	Enable cross hair cursor, clipped on viewport
4	Not supported
5	Enable box outline cursor, clipped on screen
6	Enable box outline cursor, clipped on viewport
7	Enable filled box cursor, clipped on screen
8	Enable filled box cursor, clipped on viewport

The **xsize** and **ysize** parameters are given in screen coordinates and determine the horizontal and vertical dimensions of the graphic cursor respectively. The QG–640 draws the graphic cursor in complement drawing mode, so that its color is affected only by what is already on the screen and not by the current index.

The XMOVE command has the following format:

```
XMOVE␣x␣y
```

where x and y are the screen coordinates of a new graphic cursor position. XMOVE changes the graphic cursor coordinates regardless of whether or not the graphic cursor is currently enabled.

The RBAND command has the following format:

```
RBAND␣flag
```

where flag is a Char from 0 to 2. Parameter flag determines the type of rubber band effects according to the following values:

0 - Disables the rubber band vector and rectangle.

1 - Enables the rubber band vector. The current graphic cursor position, at the time when the rubber band vector is enabled, becomes the anchor point. A line is drawn between the anchor point and the graphic cursor position. Each time a new graphic cursor position is set, the line from the anchor point to the old graphic cursor position is erased and a new line is drawn between the anchor point and the newly defined graphic cursor position. When the rubber band vector is disabled, the most recent rubber band vector is erased and the graphic cursor is left at the graphic cursor position.

2 - Enables the rubber band rectangle. The rubber band rectangle is the same as the rubber band vector except that instead of a line being drawn between the anchor point and the graphic cursor position, a rectangle is drawn with one corner at the anchor point and the diagonally opposite corner on the current graphic cursor position.

Note: since the rectangle is drawn in complement mode, the part of the rectangle that overlaps the graphic cursor will be lost when the graphic cursor display is enabled. For this reason, it is best to disable graphic cursor display when using the rubber band rectangle.

The following sequence of commands illustrates the use of the graphics input commands. The first two commands enable a cross hair graphic cursor display and move the cross hair to screen coordinates {100,200}. The next two commands enable the rubber band vector, establish the anchor point, and move the cross hair to {500,400}. The rubber band function draws a line from the anchor point to the cross hair position. The last command moves the cross hair to {500,50}, and the rubber band function erases the first line and draws a line to the new cross hair position. Figure 4.36 shows the process.

```
XHAIR␣1␣100␣100␣
XMOVE␣200␣250␣
RBAND␣1␣
XMOVE␣500␣400␣
XMOVE␣500␣50␣
```

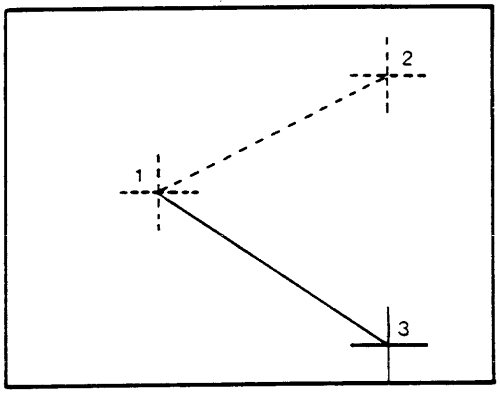


Figure 4.36: Graphics Input Example

Chapter 5

Command Descriptions

5.1 Command Descriptions

The following pages contain descriptions of the commands used by the QG–640. These commands are arranged in alphabetical order by command name and use the conventions set out in Chapter 4 to distinguish hexadecimal numbers, command names, and parameters from regular text. The parameter types use the definitions that are also laid out in Chapter 4.

ARC Draw an Arc

Command

- Long Form: ARC radius angle1 angle2
- Short Form: AR radius angle1 angle2
- Hex Form: 3C radius angle1 angle2

Parameter Type

- radius = Real
- angle1 = Int
- angle2 = Int

Description

The ARC command draws a circular arc using the currently selected color. The center is at the 2D current point. The start and finish angles are specified in the command. The angle can be any Int value (angles greater than 360° and less than -360° are handled as modulo 360). Negative radii will result in 180° being added to both angles. This command does not affect the 2D current point.

Example

CODE:

```
ASCII: AR 100.00 0 180
HEX: 3C 64 00 00 00 00 00 B4 00
```

RESULT: An arc with radius 100 from 0° to 180° (a semi-circle) is drawn about the 2D current point.

Error

Overflow

See Also

circle, color, linfun, linpat

Area Fill AREA

Command

- Long Form: AREA
- Short Form: A
- Hex Form: C0

Parameter Type None

Description

The AREA command sets all the pixels in a closed area to the current color. The closed area starts from the 2D current point and continues outward in all directions until a boundary with a color different from that of the starting pixel's original color is reached. The data tested is ANDed with the fill mask (FILMSK) and the bit plane mask *(MASK)* before comparing colors. The start pixel's original color should not be the current color.

Example CODE:

ASCII: A
HEX: C0

RESULT: The bounded area that contains the 2D current point is changed to the current color.

Error None

See Also AREAPT, FILMSK, MASK

AREABC Area Fill to Boundary Color

Command

- Long Form: AREABC index
- Short Form: AB index
- Hex Form: C1 index

Parameter Type

- index = Char

Description

The AREABC command fills a closed area bounded by color index with the current color. The closed area starts from the 2D current point and continues outward in all directions until reaching a boundary of pixels of color index. All pixel data read is ANDed with the fill mask (FILMSK) and the bit plane mask (MASK) before testing for the boundary.

Example

CODE:

```
ASCII: AB 100
HEX: C1 64
```

RESULT: The color of the area containing the 2D current point and bounded by color index is changed to the current color.

Error

Boundary = current color

See Also

AREAPT, COLOR, FILMSK, MASK

Area Pattern AREAPT

Command

- Long Form: AREAPT **pattern**
- Short Form: AP **pattern**
- Hex Form: E7 **pattern**

Parameter Type

- **pattern** = 16 Unsigned Ints

Description

The AREAPT command sets the area pattern mask. The pattern mask defines a 16× 16 array which is repeated horizontally and vertically when drawing filled figures. Each value in **pattern** is 16 bits long and sets one row of the pattern mask. Since there are 16 bytes in **pattern**, the user is able to define the value of each pixel in the pattern mask. Pixels that are where the mask is set to 1 are changed to the current color; where the mask is set to 0, the pattern is transparent or set to the background color depending on the state of COLMOD. Setting all the bits in the mask (sending 16 bytes of 65535) causes areas to be filled solidly; this is the default after reset. The area pattern is used by the following commands when drawing a filled primitive:

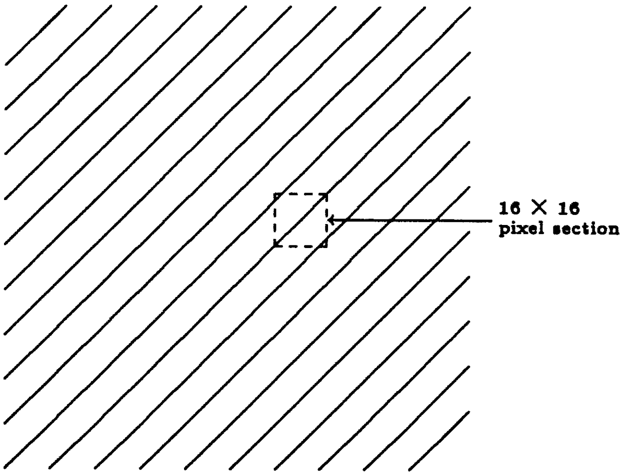
CIRCLE ELIPSE POLY POLYR
POLY3 POLYR3 RECT RECTR
SECTOR SCIRC SELIPS SPOLY
SPOLYR SRECT SRECTR SSECT

AREAPT Area Pattern

Example CODE:

```
ASCII:  AP  1      2      4      8
            16     32     64     128
            256    512    1024   2048
            4096   8192   16384  32768
HEX:   E7  00 01  00 02  00 04  00 08
           00 10  00 20  00 40  00 80
           01 00  02 00  04 00  08 00
           10 00  20 00  40 00  80 00
```

RESULT:



Error None

See Also AREA, AREABC, BCOLOR, COLMOD

Set Background Color **BCOLOR**

Command

- Long Form: BCOLOR index
- Short Form: BC index
- Hex Form: `CB` index

Parameter Type

- index = Char [0..255]

Description

This command sets the index of the background index to be used when COLMOD is set to 0.

Example

CODE:

ASCII : BCOLOR 24
HEX : `CB 18`

RESULT: The background index is changed to 24.

Error

None

See Also

COLMOD, AREAPT, LINPAT, TEXT

BLINK Blink – 8 Bit

Command

- Long Form: BLINK index red green blue ontime offtime
- Short Form: BL index red green blue ontime offtime
- Hex Form: C8 index red green blue ontime offtime

Parameter Type

- index = Char
- red, green, blue = Char [0..15]
- ontime, offtime = Char

Description

The BLINK command causes all the pixels having the color in the currently selected LUT entry to blink on and off. The currently selected LUT entry is specified by the index parameter. The periods, in $\frac{1}{60}$ seconds, are specified by ontime and offtime. During the on time, the pixel will have the original color; during the off time, the color will be the one defined by red, green, and blue. Up to four indices can be set to blink at any one time. Blink for a particular index is cancelled by issuing a second BLINK command specifying the index but with all the other parameters equal to zero.

Warning: Do not perform LUT changes on indices that are currently blinking.

Example

CODE:

```
ASCII: BL 15 255 0 0 30 30
HEX: C8 OF FF 00 00 1E 1E
```

RESULT: White (index 15) blinks to red once a second.

Error

Range is: Too many blinks specified, Color already blinking.

See Also

BLINK, LUT, LUTINT, LUTX, SBLINK

Blink – 8 Bit **BLINKX**

Command

- Long Form: BLINK **index red green blue ontime offtime**
- Short Form: BLX **index red green blue ontime offtime**
- Hex Form: E5 **index red green blue ontime offtime**

Parameter Type

- **index** = Char
- **red, green, blue** = Char
- **ontime, offtime** = Char

Description

The **BLINKX** command causes all the pixels having the color in the currently selected LUT entry to blink on and off. The currently selected LUT entry is specified by the **index** parameter. The periods, in $\frac{1}{60}$ seconds, are specified by **ontime** and **offtime**. During the on time, the pixel will have the original color; during the off time, the color will be the one defined by **red, green,** and **blue**. Up to four indices can be set to blink at any one time. Blink for a particular index is cancelled by issuing a second **BLINKX** command specifying the index but with all the other parameters equal to zero. All blinking color indices can be cancelled with the **SBLINK** command.

Warning: Do not perform LUT changes on indices that are currently blinking.

Example

CODE:

```
ASCII: BLX 15 255 0 0 30 30
HEX: E5 0F FF 00 00 1E 1E
```

RESULT: White (index 15) blinks to red once a second.

Error

Range is: Too many blinks specified, Color already blinking.

See Also

BLINK, LUT, LUTINT, LUTX, SBLINK

CA Communications ASCII

Command

- Long Form: CA␣
- Short Form: CA␣
- Hex Form: `43 41 20` or `D2`

Parameter Type

None

Description

The CA command sets the communication mode to ASCII. This command may be given when in either ASCII mode or Hex mode. Note that the Hex and ASCII forms of this command are identical.

Example

CODE:

ASCII: CA␣
HEX: `43 41 20 or D2`

RESULT: The communications mode is set to ASCII.

Error

None

See Also

CX

Circle CIRCLE

Command

- Long Form: CIRCLE radius
- Short Form: CI radius
- Hex Form: 38 radius

Parameter Type

- radius = Real

Description

The CI command draws a circle with radius radius centered on the 2D current point. The circle is filled if the PRMFIL flag is set. This command does not affect the 2D current point.

Example

CODE:

```
ASCII: CI 100
HEX: 38 64 00 00 00
```

RESULT: A circle with radius 100 is drawn from the 2D current point.

Error

Overflow

See Also

AREAPT, ARC, ELIPSE, LINFUN, LINPAT, PRMFIL, SECTOR

CLBEG Command List Begin

Command

- Long Form: CLBEG clist
- Short Form: CB clist
- Hex Form: 70 clist

Parameter Type

- clist = Char

Description

The CLBEG command begins the definition of the command list number clist. Commands are saved, without being executed, in the command list definition area. Defining a list using an already existing clist will overwrite the old command list. A command list may be up to 64 KBytes long.

Example

CODE:

```
ASCII: CB 2
HEX: 70 02
```

RESULT: Command list 2 is started.

Error Not enough memory, command list running.

See Also CLEND, CLOOP, CLDEL, CLMOD, CLRD, CLRUN

Command List Delete CLDEL

Command

- Long Form: CLDEL clist
- Short Form: CD clist
- Hex Form: 74 clist

Parameter Type

- clist = Char

Description

The CLDEL command deletes the definition of the command list specified by clist.

Example

CODE:

ASCII: CD 2

HEX: 74 02

RESULT: Command list 2 is deleted.

Error

Command list running.

See Also

CLBEG, CLEND

CLEARS Clear Screen

Command

- Long Form: CLEARS index
- Short Form: CLS index
- Hex Form: 0F index

Parameter Type

- index = Char

Description

The CLEARS command sets all the pixels in the display buffer to the color designated by index regardless of the value of MASK. The current color is not changed.

Note: This command affects not only the visible VRAM, but also the hidden space. If you want to clear only the visible buffer, use the FLOOD command.

Example

CODE:

ASCII: CLS 17
HEX: 0F 11

RESULT: Screen is filled with color 17.

Error

None

See Also

FLOOD

Command List End CLEND

Command

- Long Form: CLEND
- Short Form: CE
- Hex Form: 71

Parameter Type = None

Description

The CLEND command ends the command list currently being defined. After a CLEND, the controller resumes executing commands as they are received.

Example CODE:

ASCII: CE
HEX: 71

RESULT: The command list being defined is ended.

Error None

See Also CLBEG, CLDEL

CLIPH Clip Hither

Command

- Long Form: CLIPH flag
- Short Form: CH flag
- Hex Form: AA flag

Parameter Type

- flag = Char [0..1]

Description

The CLIPH command enables or disables hither plane clipping. Setting flag to 0 disables hither plane clipping; setting flag to 1 enables it.

Example

CODE:

CODE:
ASCII: CH 1
HEX: AA 01

RESULT: Hither clipping is enabled.

Error

None

See Also

DISTH

Clip Yon CLIPY

Command

- Long Form: CLIPY flag
- Short Form: CY flag
- Hex Form: AB flag

Parameter Type

- flag = Char [0..1]

Description

The CLIPY command enables or disables yon plane clipping. Setting flag to 0 disables yon plane clipping; setting flag to 1 enables it.

Example

CODE:

```
ASCII: CY 1
HEX: AB 01
```

RESULT: Yon clipping is enabled.

Error

None

See Also

DISTY

CLMOD Command List Modify

Command

- Long Form: CLMOD clist offset nbytes bytes
- Short Form: CM clist offset nbytes bytes
- Hex Form: 78 clist offset nbytes bytes

Parameter Type

- clist = Char
- offset = Unsigned Int
- nbytes = Unsigned Int
- bytes = nbyte's of Char

Description

The CLMOD command replaces nbytes bytes in command list clist, starting at byte number offset from the start of the command list, with the bytes contained in bytes. Note that bytes cannot be added or deleted, only replaced.

Example

CODE:

```
ASCII: CM 3 7 2 175 8
HEX: 78 03 07 00 02 00 AF 08
```

RESULT: The two bits in command list 3 with offsets 7 and 8 are replaced with CONVRT and POINT commands.

Error

None

See Also

CLMODX, CLRD, NOOP

Command List Loop **CLOOP**

Command

- Long Form: CLOOP clist count
- Short Form: CL clist count
- Hex Form: 73 clist count

Parameter Type

- clist = Char
- count = Unsigned Int

Description

The CLOOP command executes the command list clist, count times.

Example CODE:

```
ASCII: CL 4 300
HEX: 73 04 2C 01
```

RESULT: Command list 4 is executed 300 times.

Error Command list running, stack full.

See Also CLRUN

CLRD Command List Read

Command

- Long Form: CLRD clist
- Short Form: CRD clist
- Hex Form: 75 clist

Parameter Type

- clist = Char

Description

The CLR command ends the information stored in command list clist (hex form of the command) to the output channel. The first word in the data stream represents the number of bytes in the command list. It is followed by the bytes as they are stored.

Example

CODE:

```
ASCII: CRD 7
HEX: 75 07
```

RESULT: Command list 7 is listed to the Data Out register in hex.

Error

None

See Also

CLBEG, CLEND, CLDEL, CLRDX

Execute Command List CLRUN

Command

- Long Form: CLRUN clist
- Short Form: CR clist
- Hex Form: 72 clist

Parameter Type

- clist = Char

Description

The CLRUN command executes the commands in command list clist.

Example

CODE:

```
ASCII: CR 3
HEX: 72 03
```

RESULT: Command list 3 is executed.

Error

Command list running, stack full.

See Also

CLBEG, CLEND

COLMOD Color Mode

Command

- Long Form: COLMOD mode
- Short Form: CLM mode
- Hex Form: CA mode

Parameter Type

- mode = Char [0 or 1]

Description

Under certain conditions, primitives may generate both a background and a foreground. When we draw a patterned line, for example, the pattern is made up of a foreground and a background, a character cell has a foreground and a background, and any of the commands that produce filled areas produce a foreground and a background if the fill is in the form of a pattern. In such a case, the COLMOD command determines whether the background is transparent or is the color last specified by the BCOLOR command.

When mode is 0, this command sets the board to Replace Color Mode, with the result that backgrounds are given the background color set by the most recent BCOLOR command. When mode is 1, this command sets the board to Foreground Color Mode, with the result that backgrounds are drawn to be transparent.

Note that no background is drawn if the character type is graphic (vector text) and the cell rotation (TCHROT) is not a multiple of 90 . Default is Foreground Color Mode.

Example

CODE:

ASCII: COLMOD 0

HEX: CA 00

RESULT: The board enters Replace Color Mode.

Error

Range

See Also

BCOLOR, AREAPT, LINPAT, TEXT

Color COLOR

Command

- Long Form: COLOR index
- Short Form: C index
- Hex Form: 06 index

Parameter Type

- index = Char

Description

The COLOR command sets the current color to index.

Example CODE:

```
ASCII: C 12
HEX: 06 0C
```

RESULT: The current color is set to color 12.

Error Value out of range (ASCII only)

CONVRT Convert

Command

- Long Form: CONVRT
- Short Form: CV
- Hex Form: **AF**

Parameter Type None

Description

The CONVRT command maps the 3D current point to the 2D current point.

Example CODE:

ASCII: CV

HEX: **AF**

RESULT: The 3D current point is mapped to 2D and placed in the 2D current point.

Error Overflow

Communications Hexadecimal CX

Command

- Long Form: CX␣
- Short Form: CX␣
- Hex Form: 43 58 20 or D1

Parameter Type None

Description

The CX command sets the communication mode to hexadecimal. This command may be given when in either ASCII mode or Hex mode. Note that the Hex and ASCII forms of this command are identical.

Example CODE:

ASCII: CX␣
HEX: 43 58 20 OR D1

RESULT: The communication mode is set to hexadecimal.

Error None

See Also CA

DISTAN Distance

Command

- Long Form: DISTAN dist
- Short Form: DS dist
- Hex Form: B1 dist

Parameter Type

- dist = Real

Description

The DISTAN command sets the distance from the eye to the viewing reference point. This only affects 3D drawing. The default distance is 500.

Example

CODE:

ASCII: DS 1200

HEX: B1 B0 04 00 00

RESULT: Distance to viewing reference point is set to 1200.

Error

None

See Also

PROJCT

Distance Hither DISTH

Command

- Long Form: DISTH dist
- Short Form: DH dist
- Hex Form: A8 dist

Parameter Type

- dist = Real

Description

The DISTH command sets the distance from the viewing reference point to the hither clip plane. When hither clipping is enabled, no points closer to the viewer than the hither plane are displayed. The hither plane is parallel to the viewplane. Hither clipping affects only 3D drawing.

Example CODE:

```
ASCII: DH -12.00
HEX: A8 F4 FF 00 00
```

RESULT: The hither plane is defined to be 12.00 units in front of the viewplane.

Error None

See Also CLIPH

DISTY Distance Yon

Command

- Long Form: DISTY dist
- Short Form: DY dist
- Hex Form: A9 dist

Parameter Type

- dist = Real

Description

The DISTY command sets the distance from the viewing reference point to the yon clip plane. When yon clipping is enabled, no points farther from the viewer than the yon plane are displayed. The yon plane is parallel to the viewplane. Yon clipping affects only 3D drawing.

Example

CODE:

ASCII: DY 12.00

HEX: A9 0C 00 00 00

RESULT: The yon plane is defined to be 12.00 units behind the viewplane.

Error

None

See Also

CLIPY

Draw DRAW

Command

- Long Form: DRAW x y
- Short Form: D x y
- Hex Form: 28 x y

Parameter Type

- x = Real
- y = Real

Description

The DRAW command draws a line from the 2D current point to (x, y) and positions the 2D current point at (x, y). Both the first and the last pixels of the line are drawn.

Example CODE:

```
    ASCII: D 10.0 12.0
    HEX: 28 0A 00 00 00 0C 00 00 00
RESULT: A line is drawn from the 2D current point to (10.0, 12.0).
```

Error Arithmetic overflow

See Also DRAWR, LINFUN, LINPAT, MOVE, MOVER

DRAW3 — Draw in 3D

Command

- Long Form: DRAW3 x y z
- Short Form: D3 x y z
- Hex Form: 2A x y z

Parameter Type

- x = Real
- y = Real
- z = Real

Description

The DRAW3 command draws a line from the 3D current point to (x, y, z) and moves the current point to (x, y, z).

Example

CODE:

```
ASCII: D3 5.0 10.0 12.0
HEX: 2A 05 00 00 00 0A 00 00 00 0C 00 00 00
```

RESULT: A line is drawn from the 3D current point to (5.0, 10.0, 12.0).

Error

Arithmetic overflow

See Also

DRAWR3, LINFUN, LINPAT, MOVE3, MOVER3

Draw Relative DRAWR

Command

- Long Form: DRAWR Δx Δy
- Short Form: DR Δx Δy
- Hex Form: 29 Δx Δy

Parameter Type

- Δx = Real
- Δy = Real

Description

The DRAWR command draws a line from the 2D current point to {(Δx, Δy) + 2D current point}. The 2D current point is moved to the end of the line. Both the first and the last pixels of the line are drawn.

Example

CODE:

```
ASCII: DR 100.00 200.00
HEX: 29 64 00 00 00 C8 00 00 00
```

RESULT: A line is drawn from the 2D current point to {2D current point + (100.00, 200.00)}.

Error

Arithmetic overflow

See Also

DRAWR, LINFUN, LINPAT, MOVE, MOVER

DRAWR3 Draw Relative in 3D

Command

- Long Form: DRAWR3 Δx Δy Δz
- Short Form: DR3 Δx Δy Δz
- Hex Form: 2B Δx Δy Δz

Parameter Type

- Δx = Real
- Δy = Real
- Δz = Real

Description

The DRAWR3 command draws a line from the 3D current point to {(Δx, Δy, Δz) + 3D current point} and moves the current point to the end of the line.

Example

CODE:

```
ASCII: DR3 5.0 10.0 12.0
HEX: 2B 05 00 00 00 0A 00 00 00 0C 00 00 00
```

RESULT: A line is drawn from the 3D current point to {(5.0, 10.0, 12.0) + 3D current point}.

Error

Arithmetic overflow

See Also

DRAW3, LINFUN, LINPAT, MOVE3, MOVER3

Ellipse ELIPSE

Command

- Long Form: ELIPSE **xradius yradius**
- Short Form: EL **xradius yradius**
- Hex Form: 39 **xradius yradius**

Parameter Type

- **xradius** = Real
- **yradius** = Real

Description

The ELIPSE command draws a 2D ellipse centered on the 2D current point. Its x and y radii, which are parallel to the screen's x and y axes, are given by **xradius** and **yradius**. The ellipse will be filled if drawn while the PRMFIL flag is set. This command does not affect the 2D current point.

Example

CODE:

```
ASCII: EL 32.00 128.00
HEX: 39 20 00 00 00 80 00 00 00
```

RESULT: An ellipse is drawn with x radius 32 and y radius 128.

Error

Overflow

See Also

AREAPT, LINFUN, LINPAT, PRMFIL

ERROR Error Reporting

Command

- Long Form: ERROR flag
- Short Form: ER flag
- Hex Form: 60 flag

Parameter Type

- flag = Unsigned Int

Description

The ERROR command enables (flag = 1) or disables (flag = 0) error reporting. The current value of flag can be read using a FLAGRD 38 command.

Example CODE:

ASCII: ER 0
HEX: 60 00

RESULT: Error reporting is disabled.

Error Value out of range.

See Also FLAGRD

Fill Mask FILMSK

Command

- Long Form: FILMSK mask
- Short Form: FM mask
- Hex Form: EF mask

Parameter Type

- mask = Char

Description

The FILMSK command defines the area fill mask to be the value mask. When an area fill command tests for a boundary index, pixel data is ANDed with this mask as well as MASK. Default value is no mask.

Example

CODE:

```
ASCII: FM 126
HEX: EF 7E
```

RESULT: Area fill mask is set to 126.

Error

None

See Also

AREA, AREABC, MASK

FLAGRD Flag Read

Command

- Long Form: FLAGRD flag
- Short Form: FRD flag
- Hex Form: 51 flag

Parameter Type

- flag = Char [1..48]

Description

The FLAGRD command places the current value of the flag specified by flag into the Data Out register. Values are read back using the current communication mode, in the same format as the instructions that wrote them. The values of flag are specified in the table on the following page.

Example

CODE:

```
ASCII: FRD 25
HEX: 51 19
```

RESULT: The amount of free memory is placed in the read back buffer.

Error

No such flag

See Also

RESETF

Flag Read FLAGRD

Flag	Name	Type of Value
1	AREAPT	16 Ints
2	CLIPH	1 Char
3	CLIPY	1 Char
4	COLOR	1 Char
6	DISTAN	1 Real
7	DISTH	1 Real
8	DISTY	1 Real
9	FILMSK	1 Char
10	LINFUN	1 Char
11	LINPAT	1 Int
12	MASK	1 Char
13	MDORG	3 Reals
14	2D current point	2 Reals
15	3D current point	3 Reals
16	PRMFIL	1 Char
17	PROJCT	1 Char
18	TANGLE	1 Int
19	TJUST	2 Chars
20	TSIZE	1 Real
21	VWPORT	4 Ints
22	VWRPT	3 Reals
23	WINDOW	4 Reals
24	Transformed 3D current point	3 Reals
25	Free memory (less than 64K)	1 Int
26	Current point of XHAIR	2 Ints
27	2D position of XHAIR	2 Reals
28	Screen Current Point	2 Ints
29	True free memory	1 Real *
30	Reserved	
31	Reserved	
32	TSTYLE	1 Char
33	TASPCT	1 Real
34	TCHROT	1 Int
35	Reserved	1 Char
36	VDISP	2 Chars
37	PMASK	1 Int
38	ERROR	1 Char
39	DISPLAY	2 Ints and 2 Chars **
41	COLMOD	1 Char
42	BCOLOR	1 Char
43	Board Type, Revision #	2 Chars ***

* This value is treated as a double precision integer.

** The values are returned in the following order: Max Screen X, Max Screen Y, Refresh Rate in Hz, and Interlace Flag (0 = non-interlaced, 1 = interlaced).

***Board Type = 6 for the QG-640.

FLOOD Flood

Command

- Long Form: FLOOD **index**
- Short Form: F **index**
- Hex Form: 07 **index**

Parameter Type

- **index** = Char

Description

The FLOOD command sets all the pixels in the defined viewport to the color specified by **index**. The current color is not changed and the command is subject to MASK.

Example

CODE:

```
ASCII: F 12
HEX: 07 0C
```

RESULT: The rectangular area defined by the viewport is filled with color 12.

Error

None

See Also

CLEARS, MASK

Graphics Text Font Define GTDEF

Command

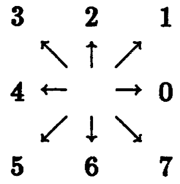
- Long Form: GTDEF ch n width array
- Short Form: GTD ch n width array
- Hex Form: 89 ch n width array

Parameter Type

- ch = Char
- n = Char
- width = Char [1..12]
- array = n values

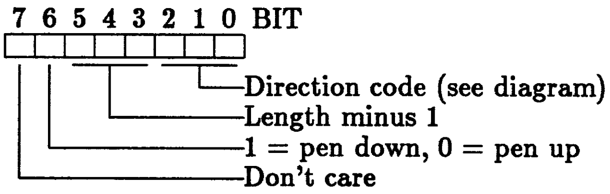
Description

The GTDEF command defines the character given by ch in the user font as a series of vector plots stored in the n values of array. The width of the character cell is given by width and the height is fixed at 12. The starting point for the definition is at (0,3) of the character cell. Each value in the array consists of three parts: the pen action, the length, and the direction. The pen action may be move (pen action = 0) or write (pen action = 1). The length may take a value from 1 to 8. The direction can be from 0 to 7 and is summarized in the diagram below:



Each of these values is specified by a separate number when in ASCII mode. In Hex mode, the values are packed into a single byte with the three low bits containing the direction, the next three bits containing the length less one, and the seventh bit containing the pen action. The format of the vector parameter is shown in the following diagram:

GTDEF Graphics Text Font Define



Notes :

- Any previous definition is lost. To reset a character to its default value, specify n as 00.
- Specifying characters using this command (rather than TDEFIN) will allow the characters to be enlarged and rotated.
- If you plan to define an entire font, it is faster to reset all previous characters starting by the last character (255, 254, 253, ..., 0) and then define the character font starting at 0, 1, 2, ..., 255.

Example CODE:

```
ASCII: GTD 65 7 8 1 7 2
                  1 2 1
                  1 3 0
                  1 2 7
                  1 7 6
                  0 4 2
                  1 7 4
HEX: 89 41 07 08 72 49 50 4F 76 1A 74
```

RESULT: The letter "A" is defined.

Error Not enough memory, Bad definition.

Image Read **IMAGER**

Command

- Long Form: IMAGER line x1 x2
- Short Form: IR line x1 x2
- Hex Form: D8 line x1 x2

Parameter Type

- line = Unsigned Int [0..479]
- x1 = Unsigned Int [0..639]
- x2 = Unsigned Int [0..639]

Description

The IMAGER command reads pixel values from the image currently being displayed and sends these values to the Read Back Buffer. Parameters line, x1 and x2 are measured in pixels from the lower left corner of the screen. When the communications mode is set to ASCII, the data is returned in the format of an IMAGEW command, with pixels separated by commas, and a prefix of IW line, x1, x2, x, x, ...(CR). Where IW is a header identifying the string as a result of an IMAGER command, line is the line number, x1 and x2 specify a line segment, the x's represent ASCII decimal color indices separated by commas, and the carriage return (CR) character ends the string.

Example

CODE:

```
ASCII: IR 50 0 256
HEX: D8 32 00 00 00 00 01
```

RESULT: The values of pixels 0 through 256 from line 50 are sent to the Data Out Register.

Error Value out of range.

See Also CA, CX, EXPAND, IMAGEW

IMAGEW Image Write

Command

- Long Form: IMAGEW line x1 x2 data
- Short Form: IW line x1 x2 data
- Hex Form: D9 line x1 x2 data

Parameter Type

- line = Unsigned Int [0..1023] or [0..479] depending on the state of the expand flag.
- x1 = Unsigned Int [0..2043] or [0..639] depending on the state of the expand flag.
- x2 = Unsigned Int [0..2043] or [0..639] depending on the state of the expand flag.
- data = ASCII: string of Chars
  Hex: run length encoded string

Description

The IMAGEW command writes pixel values to the image currently being displayed. Parameters line, x1, and x2 are measured in pixels from the lower left corner of the screen. When the communication mode is set to ASCII, the values of the pixels are expected to be ASCII numbers separated by blanks. When the communication mode is set to Hex, the input is expected to be in run-length encoded format.

In run length encoded form, the user sends either byte pairs or a count and a string of bytes. When the high bit of the first byte is not set, a byte pair is expected: the first byte represents the count less one, the second byte is the pixel value to be repeated count times. If the high bit is set, then the first byte is the length less one of the byte string which follows. In both cases, the count and the length only use the low seven bits for the value. See Section 4.10 for more information on run-length encoding.

Example

CODE:

```
ASCII: IW 50 0 10 0 0 0 0 0 0 0 0 0 0 0
HEX: D9 32 00 00 00 0A 00 00 00
```

RESULT: The values of pixels 0 through 10 of line 50 are set to 0.

Error

Value out of range.

See Also

CA, CX, IMAGER

Line Function LINFUN

Command

- Long Form: LINFUN function
- Short Form: LF function
- Hex Form: EB function

Parameter Type

- function = Char [0..4]

Description

The LINFUN command sets the drawing function to the function specified by the following table:

function	Mode Name
1	Complement
2	XOR
3	OR
4	AND

When Replace Mode is selected, drawing is done in the current color. Choosing Complement Mode will complement each pixel as it is drawn – the current color will be ignored. The remaining modes perform the specified logic operation between the pixel and the current color. Drawing is subject to MASK.

Example

CODE:

ASCII: LF 0

HEX: EB 00

RESULT: Drawing is performed in the current color.

Error

None

See Also

MASK

LINPAT Line Pattern

Command

- Long Form: LINPAT pattern
- Short Form: LP pattern
- Hex Form: EA pattern

Parameter Type

- pattern = Unsigned Int

Description

The LINPAT command sets the line drawing pattern mask to pattern. Each of the 16 bits in pattern represents a pixel (two pixels if EXPAND is 1) in subsequently drawn lines. Note that the first bit drawn is the high order bit. Pixels that are where the mask is set to 1 are changed to the current color; where the mask is set to 0, the pattern is set to the background color depending on the state of COLMOD. The pattern is repeated along the entire length of the line drawn when using one of the following commands (and PRMFIL is not set, in the case of closed figures):

ARC CIRCLE DRAW DRAWR
DRAW3 DRAWR3 ELIPSE POLY
POLYR POLY3 POLYR3 RECT
RECTR SECTOR

Example

CODE:

ASCII: LP 255

HEX: EA FF 00

RESULT: Dashed lines are drawn when the above commands are used.

Error

None

See Also

EXPAND, LINFUN, PRMFIL

Lookup Table **LUT**

Command

- Long Form: LUT **index r g b**
- Short Form: L **index r g b**
- Hex Form: **EE index r g b**

Parameter Type

- **index** = Char
- **r** = Char [0..15]
- **g** = Char [0..15]
- **b** = Char [0..15]

Description

The **LUT** command loads the three RGB intensity values into the LUT entry specified by **index**. The values sent by this command (residing in bits 0 through 3) are loaded into the four high bits (bits 4 through 7) of the lookup table. This process converts the **LUT** values to **LUTX** values by loading bit 0 to bit 4, bit 1 to bit 5, bit 2 to bit 6, and bit 3 to bit 7. **LUTX** is the preferred form of the command.

Example

CODE:

```
ASCII: L 15 2 4 8
HEX: EE 0F 02 04 08
```

RESULT: LUT entry 15 is set to **r** = 2, **g** = 4 and **b** = 8.

Error Out of range

See Also **LUTINT, LUTRD, LUTSAV, LUTSTO, LUTX, LUTXRD, VDISP**

LUTINT — Lookup Table Initialization

Command

- Long Form: LUTINT state
- Short Form: LI state
- Hex Form: EC state

Parameter Type

- state = Char

Description

The LUTINT command sets the LUT to the state specified by the following table:

State	Description
0	Color cone distribution
1	Foreground/background colors in the low 4 bits of a value code will be visible only if the high 4 bits are 0 (or invisible)
2	Value codes interpreted as R R G G G B B B
3	Value codes interpreted as R R R G G B B B
4	Value codes interpreted as R R R G G G B B
5	6-level RGB
253	Load LUT from LUT storage areas 0 and 1 alternately
254	Load LUT from LUT storage area 1
255	Load LUT from LUT storage area 0

Example

CODE:

ASCII: LI 255

HEX: EC FF

RESULT: LUT is loaded from LUT storage area 0.

Error

Value out of range.

See Also

LUT, LUTRD, LUTSAV, LUTSTO

Lookup Table Read **LUTRD**

Command

- Long Form: LUTRD index
- Short Form: LRD index
- Hex Form: 50 index

Parameter Type

- index = Char

Description

The LUTRD command reads the red, green, and blue values of the LUT entry specified by index and sends them to the output buffer. In ASCII mode, the three values are ASCII numbers separated by commas and terminated by a carriage return. In Hex mode, the LUT values are sent in three bytes, one byte for each color. This command reads back the high four bits (bits 4 through 7) of the LUT entry. An entry set with the LUTX command reads back the high four bits of the eight-bit value, and loads them into the low four bits (bits 0 through 3) of the lookup table. The order in which the values are loaded are: bit 7 to bit 3, bit 6 to bit 2, bit 5 to bit 1, and bit 4 to bit 0.

Example

CODE:

```
ASCII: LRD 25
HEX: 50 19
```

RESULT: Values of LUT entry 19 are returned.

Error

None

See Also

CA, CX, LUT, LUTINT, LUTSAV, LUTSTO, VDISP

LUTSAV Lookup Table Save

Command

- Long Form: LUTSAV
- Short Form: LS
- Hex Form: ED

Parameter Type None

Description

The LUTSAV command saves the current lookup table in storage area 0. These values may be reloaded into the LUT using a LUTINT 255 command. Each LUTSAV command overwrites any LUT data previously saved. Note that LUTSAV is identical to the LUTSTO 0 command.

Example CODE:

```
ASCII: LS
HEX: ED
```

RESULT: LUT data is stored in LUT storage area 0.

Error None

See Also LUT, LUTINT, LUTRD, LUTSTO

LUT Store LUTSTO

Command

- Long Form: LUTSTO flag
- Short Form: LST flag
- Hex Form: C9 flag

Parameter Type

- flag = Char [0..1]

Description

The LUTSTO command saves the current lookup table in one of two user areas. Note that LUTSAV and LUTSTO 0 are identical. Table 0 can be recalled by LUTINT 255 and Table 1 by LUTINT 254. Each LUTSTO command overwrites any LUT data previously saved in the specified user area.

Example

CODE:

```
ASCII: LST 1
HEX: C9 01
```

RESULT: The current LUT values are stored in Table 1.

Error

None

See Also

LUTINT, LUTSAV

LUTX — Lookup Table – 8 Bit

Command

- Long Form: LUTX r g b
- Short Form: LX r g b
- Hex Form: E6 index r g b

Parameter Type

- index = Char
- r = Char
- g = Char
- b = Char

Description

The LUTX command loads the three eight-bit RGB intensity values into the lookup table entry specified by **index.** The values sent to the digital-to-analog converter are dependent on the resolution of the converter.

Example

CODE:

```
ASCII: LX 15 2 4 8
HEX: E6 0F 02 04 08
```

RESULT: Lookup table entry 15 is set to r = 2, g = 4, b = 8.

Error

None

See Also

LUTINT, LUTRD, LUTSAV, LUTSTO, LUTXRD

Lookup Table Read – 8 Bit **LUTXRD**

Command

- Long Form: LUTXRD index
- Short Form: LXR index
- Hex Form: 53 index

Parameter Type

- index = Char

Description

The LUTXRD command reads the red, green, and blue values of the LUT entry specified by index and sends them to the output buffer. In ASCII mode, the three values are ASCII numbers separated by commas and terminated by a carriage return. In Hex mode, the LUT values are sent in three bytes, one byte for each color. Each LUT value is in the range 0 to 255.

Example

CODE:

```
ASCII: LXR 25
HEX: 53 19
```

RESULT: Values of LUT entry 19 are returned.

Error

None

See Also

CA, CX, LUTX, LUTINT, LUTSAV, LUTSTO

MASK Mask

Command

- Long Form: MASK **planemask**
- Short Form: MK **planemask**
- Hex Form: **E8** **planemask**

Parameter Type

- **planemask** = Char

Description

The **MASK** command sets the 8-bit read/write pixel data bit plane mask to the value contained in **planemask**. Each bit in **planemask** will enable the corresponding bit plane in the video buffer to be read or written. Zeroes written to all eight bits will prevent data from being written to any pixel data bit plane and will cause 0's to be returned when pixel data is read.

Example

CODE:

ASCII: MK 255

HEX: **E8 FF**

RESULT: All bit planes can be read or written.

Error

None

Matrix Read MATXRD

Command

- Long Form: MATXRD matrix
- Short Form: MRD matrix
- Hex Form: 52 matrix

Parameter Type

- matrix = Char [1..2]

Description

The MATXRD command copies the contents of the matrix specified by matrix to the output buffer. When matrix is 1, the contents of the 3D modelling transformation matrix are copied, when matrix is 2 the contents of the 3D viewing transformation matrix are copied. In ASCII mode, the matrix elements are written in four lines, each of which has four entries separated by commas and terminated by a carriage return. In Hex mode, each matrix element is written as four bytes with the following reading order.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Example

CODE:

ASCII: MRD 2

HEX: 52 02

RESULT: The contents of the viewing transformation matrix are copied to the output buffer.

Error

Value out of range.

See Also

CA, CX

MDIDEN Modelling Identity

Command

- Long Form: MDIDEN
- Short Form: MDI
- Hex Form: 90

Parameter Type None

Description

The MDIDEN command sets the modelling transformation matrix to the identity matrix.

Example CODE:

```
ASCII: MDI
HEX: 90
```

RESULT: The modelling transformation matrix is set to the identity matrix.

Error

None

See Also

DRAWR3, MDMATX, MOVE3, MOVER3, POINT3, POLY3, POLYR3

Modelling Matrix **MDMATX**

Command

- Long Form: MDMATX array
- Short Form: MDM array
- Hex Form: 97 array

Parameter Type

- array = 16 Reals

Description

The MDMATX command loads the modelling matrix directly from the data in array.

Example

CODE:

```
ASCII: MDM   36.25   12.00  128  2
                0    36.75  100  0
             72.5      0    2.5  0
            100.25     0     0   0
HEX:  97  24  00  00  40  0C  00  00  00
          80  00  00  00  02  00  00  00
          00  00  00  00  24  00  00  C0
          64  00  00  00  00  00  00  00
          52  00  00  80  00  00  00  00
          02  00  00  80  00  00  00  00
          64  00  00  40  00  00  00  00
          00  00  00  00  00  00  00  00
```

RESULT: The modelling matrix is set to the above data.

Error

Arithmetic overflow

See Also

MDORG, MDROTX, MDROTY, MDROTZ, MATXRD

MDORG Modelling Origin

Command

- Long Form: MDORG 0x 0y 0z
- Short Form: MDO 0x 0y 0z
- Hex Form: 91 0x 0y 0z

Parameter Type

- 0x = Real
- 0y = Real
- 0z = Real

Description

The MDORG command defines the origin section of the modelling transformation matrix used in modelling transformation scaling and rotating.

Example

CODE:

```
ASCII: MDO 0.0 12.5 1.0
HEX: 91 00 00 00 00 0C 00 00 80 01 00 00 00
```

RESULT: Origin is defined as x = 0, y = 12.5, and z = 1.

Error

None

See Also

MDROTX, MDROTY, MDROTZ, MATXRD

Modelling Rotate X Axis MDROTX

Command

- Long Form: MDROTX angle
- Short Form: MDX angle
- Hex Form: 93 angle

Parameter Type

- angle = Int

Description

The MDROTX command rotates the object about the x axis by angle.

Example

CODE:

ASCII: MDX 45
HEX: 93 2D 00

RESULT: The object is rotated by 45° about the x axis.

Error

Arithmetic overflow

See Also

MDMATX, MDORG, MDROTY, MDROTZ

MDROTY Modelling Rotate Y Axis

Command

- Long Form: MDROTY angle
- Short Form: MDY angle
- Hex Form: 94 angle

Parameter Type

- angle = Int

Description

The MDROTY command rotates the object about the y axis by angle.

Example

CODE:

ASCII: MDY 45

HEX: 94 2D 00

RESULT: The object is rotated by 45° about the y axis.

Error

Arithmetic overflow

See Also

MDMATX, MDORG, MDROTX, MDROTZ

Modelling Rotate Z Axis **MDROTZ**

Command

- Long Form: MDROTZ angle
- Short Form: MDZ angle
- Hex Form: 95 angle

Parameter Type

- angle = Int

Description

The MDROTZ command rotates the object about the z axis by angle.

Example

CODE:

ASCII: MDZ 45

HEX: 95 2D 00

RESULT: The object is rotated by 45° about the z axis.

Error

Arithmetic overflow

See Also

MDMATX, MDORG, MDROTX, MDROTY

MDSCAL Modelling Scale

Command

- Long Form: MDSCAL sx sy sz
- Short Form: MDS sx sy sz
- Hex Form: 92 sx sy sz

Parameter Type

- sx = Real
- sy = Real
- sz = Real

Description

The MDSCAL command changes the scaling component of the modelling matrix for 3D drawing.

Example

CODE:

```
ASCII: MDS 2 4 8
HEX: 92 02 00 00 00 04 00 00 00 08 00 00 00
```

RESULT: Scaling component is set to (2, 4, 8).

Error

Arithmetic overflow

See Also

MDMATX

Modelling Translation MDTRAN

Command

- Long Form: MDTRAN tx ty tz
- Short Form: MDT tx ty tz
- Hex Form: 96 tx ty tz

Parameter Type

- tx = Real
- ty = Real
- tz = Real

Description

The MDTRAN command moves the translation component of the modelling matrix for 3D drawing by (tx, ty, tz).

Example

CODE:

```
ASCII: MDT 2 4 8
HEX: 96 02 00 00 00 04 00 00 00 08 00 00 00
```

RESULT: Translation component is set to (2, 4, 8).

Error

Arithmetic overflow

See Also

MDMATX

MOVE Move

Command

- Long Form: MOVE x y
- Short Form: M x y
- Hex Form: 10 x y

Parameter Type

- x = Real
- y = Real

Description

The MOVE command moves the 2D current point to (x, y).

Example

CODE:

```
ASCII: M 10.0 12.0
HEX: 10 0A 00 00 00 0C 00 00 00
```

RESULT: The current point is moved to (10.0, 12.0).

Error

Arithmetic overflow

See Also

MOVER

Move in 3D MOVE3

Command

- Long Form: MOVE3 x y z
- Short Form: M3 x y z
- Hex Form: 12 x y z

Parameter Type

- x = Real
- y = Real
- z = Real

Description

The MOVE3 command moves the 3D current point to (x, y, z).

Example CODE:

ASCII: M3 5.0 10.0 12.0

HEX: 12 05 00 00 00 0A 00 00 00 0C 00 00 00

RESULT: The 3D current point is moved to (5.0, 10.0, 12.0).

Error Arithmetic overflow

See Also MOVER3

MOVER Move Relative

Command

- Long Form: MOVER Δx Δy
- Short Form: MR Δx Δy
- Hex Form: 11 Δx Δy

Parameter Type

- Δx = Real
- Δy = Real

Description

The MOVER command moves the 2D current point to {Δx, Δy) + 2D current point}.

Example

CODE:

```
ASCII: MR 10.0 12.0
HEX: 11 0A 00 00 00 0C 00 00 00
```

RESULT: The 2D current point is moved to
{(10.0, 12.0)+ 2D current point}.

Error

Arithmetic overflow

See Also

MOVE

Move Relative in 3D MOVER3

Command

- Long Form: MOVER3 Δx Δy Δz
- Short Form: MR3 Δx Δy Δz
- Hex Form: 13 Δx Δy Δz

Parameter Type

- Δx = Real
- Δy = Real
- Δz = Real

Description

The MOVER3 command moves the 3D current point by the displacement:

(Δx, Δy, Δz).

Example

CODE:

```
ASCII: MR3 5.0 10.0 12.0
HEX: 13 05 00 00 00 0A 00 00 00 0C 00 00 00
```

RESULT: The 3D current point is moved to {(5.0, 10.0, 12.0) + 3D current point}.

Error

Arithmetic overflow

See Also

MOVE3

NOOP No Operation

Command

- Long Form: NOOP
- Short Form: NOP
- Hex Form: 01

Parameter Type None

Description

The NOOP command does nothing. It can be used to hold a byte when editing command lists.

Example CODE:

ASCII: NOP
HEX: 01

RESULT: Nothing.

Error None

See Also CLMOD

Poly Draw PDRAW

Command

- Long Form: PDRAW $x_1, y_1, x_2, y_2, \cdots, x_n, y_n$
- Short Form: PD $x_1, y_1, x_2, y_2, \cdots, x_n, y_n$
- Hex Form: FF $x_1, y_1, x_2, y_2, \cdots, x_n, y_n$

Parameter Type

- x_i = Int
- y_i = Int

Description

The PDRAW command executes a stream of high speed screen moves and vector draws. This command operates in screen mode and consequently affects the 2D current point. The high bit of the x and y coordinates are used as flags. If the high bit of x_i is set to 1, then the command stream is terminated with the i^{th} coordinate pair. Otherwise the coordinate pair is accepted as a move or draw command. The high bit of the y coordinate is used to distinguish between a current point move (high bit set to 1) and a vector draw (high bit set to 0). The PDRAW command allows the highest drawing speeds to be attained.

Note: An easy way to calculate the value of a decimal number with the high bit set is: $n_{set} = n_o - 32768$. For example, to move to (125, 340), use the x = 125 and y = 340 – 32768 = –32428.

Example

CODE:

```
ASCII: PD 96 -32672 0 0 -1 0
HEX: FF 60 00 60 80 00 00 00 00 FF FF 00 00
```

RESULT: The current point will be moved to (96, 96) and a vector will be drawn to (0, 0).

Error

None

PMASK Pixel Mask

Command

- Long Form: PMASK bitmask
- Short Form: PM bitmask
- Hex Form: D6 bitmask

Parameter Type

- bitmask = Char

Description

This command sets the 8-bit output mask to the value contained in bitmask. Each 1 in bitmask will enable the corresponding bit plane in the frame buffer to be sent to the output lookup table. Zeroes written to all eight bits will cause data to be sent as zeroes to the output lookup table. The current value of PMASK can be read using a FLAGRD 37 command.

Example

CODE:

ASCII: PM 255
HEX: D6 FF

RESULT: All bits can be read or written.

Error

None

See Also

FLAGRD

Point POINT

Command

- Long Form: POINT
- Short Form: PT
- Hex Form: 08

Parameter Type None

Description

The POINT command sets the pixel located at the 2D current point to the current color. This command does not move the 2D current point.

Example CODE:

ASCII: PT

HEX: 08

RESULT: The pixel at the 2D current point is set to the current color.

Error None

See Also LINFUN, LINPAT

POINT3 Point in 3D

Command

- Long Form: POINT3
- Short Form: PT3
- Hex Form: 09

Parameter Type None

Description

The POINT3 command sets the pixel located at the 3D current point to the current color. This command does not move the 3D current point.

Example CODE:

ASCII: PT3

HEX: 09

RESULT: The pixel at the 3D current point is set to the current color.

Error None

See Also LINFUN, LINPAT

Polygon **POLY**

Command

- Long Form: POLY n $x_1\ y_1\ x_2\ y_2 \cdots x_n\ y_n$
- Short Form: P n $x_1\ y_1\ x_2\ y_2 \cdots x_n\ y_n$
- Hex Form: 30 n $x_1\ y_1\ x_2\ y_2 \cdots x_n\ y_n$

Parameter Type

- n = Char
- x_i = Real
- y_i = Real

Description

The POLY command draws a closed polygon in two dimensions. n is the number of vertices and x_i, y_i) the coordinates of the vertices. The polygon will be filled if the PRMFIL flag is set and subject to the LINPAT if PRMFIL is not set. The two dimensional current point will not be changed.

Example

CODE:

```
    ASCII: P 4 0 0 16 0 16 16 0 16
    HEX: 30  04 00 00 00 00 00 00 00 00
             10 00 00 00 00 00 000 10 00
             00 00 10 00 00 00 00 00 00
             00 10 00 00 00
RESULT: A 16× 16 square is drawn.
```

Error

Not enough memory, arithmetic overflow.

See Also

AREAPT, LINFUN, LINPAT, POLYR, PRMFIL

POLY3 Polygon in 3D

Command

- Long Form: POLY3 n $x_1\ y_1\ z_1 \cdots x_n\ y_n\ z_n$
- Short Form: P3 n $x_1\ y_1\ z_1 \cdots x_n\ y_n\ z_n$
- Hex Form: 32 n $x_1\ y_1\ z_1 \cdots x_n\ y_n\ z_n$

Parameter Type

- n = Char
- x_i = Real
- y_i = Real
- z_i = Real

Description

The POLY3 command draws a closed polygon where n is the number of vertices and (x_i, y_i, z_i) are the coordinates of the vertices. The polygon is filled if the PRMFIL flag is set and subject to the LINPAT if PRMFIL is not set. The 3D current point is not changed.

Example

CODE:

```
ASCII: P3 4 0 0 0 16 0 0 16 0 16 0 0 16
HEX: 32 04 00 00 00 00 00 00 00 00
        00 00 00 00 10 00 00 00 00
        00 00 00 00 00 00 00 10 00
        00 00 00 00 00 00 10 00 00
        00 00 00 00 00 00 00 00 00
        10 00 00 00
```

RESULT: A 16× 16 square is drawn along the xz plane.

Error

Not enough memory, arithmetic overflow.

See Also

AREAPT, LINFUN, LINPAT, POLYR3, PRMFIL

Polygon Relative **POLYR**

Command

- Long Form: POLYR n Δx_1 Δy_1 Δx_2 Δy_2 $\cdots$ Δx_n Δy_n
- Short Form: PR n Δx_1 Δy_1 Δx_2 Δy_2 $\cdots$ Δx_n Δy_n
- Hex Form: 31 n Δx_1 Δy_1 Δx_2 Δy_2 $\cdots$ Δx_n Δy_n

Parameter Type

- n = Char
- Δx_i = Real
- Δy_i = Real

Description

The POLYR command draws a closed polygon in 2D. Parameter n is the number of vertices and Δx_i, Δy_i) are the displacements from the current point of the vertices. The polygon will be filled if the PRMFIL flag is set and subject to the LINPAT if PRMFIL is not set. The 2D current point will not be changed.

Example

CODE:

```
ASCII: PR 4 0 0 16 0 16 16 0 16
HEX: 31 04 00 00 00 00 00 00 00 00
        10 00 00 00 00 00 00 00 10
        00 00 00 10 00 00 00 00 00
        00 00 10 00 00 00
```

RESULT: A 16× 16 square is drawn with the lower left corner on the current point.

Error

Not enough memory, arithmetic overflow.

See Also

AREAPT, LINFUN, LINPAT, POLY, PRMFIL

POLYR3 Polygon Relative in 3D

Command

- Long Form: POLYR3 n Δx_1 Δy_1 Δz_1 $\cdots$ Δx_n Δy_n Δz_n
- Short Form: PR3 n Δx_1 Δy_1 Δz_1 $\cdots$ Δx_n Δy_n Δz_n
- Hex Form: 33 n Δx_1 Δy_1 Δz_1 $\cdots$ Δx_n Δy_n Δz_n

Parameter Type

- n = Char
- Δx_i = Real
- Δy_i = Real
- Δz_i = Real

Description

The POLYR3 command draws a closed polygon where n is the number of vertices and (Δx_i, Δy_i, Δz_i) are the displacements from the current point of the vertices. The polygon is filled if the PRMFIL flag is set and subject to LINPAT if PRMFIL is not set. The 3D current point is not changed.

Example

CODE:

```
ASCII: PR3 4 0 0 0 16 0 0 16 0 16 0 0 16
HEX: 33 04 00 00 00 00 00 00 00 00
        00 00 00 00 10 00 00 00 00
        00 00 00 00 00 00 00 10 00
        00 00 00 00 00 00 10 00 00
        00 00 00 00 00 00 00 00 10
        00 00 00
```

RESULT: A 16× 16 square is drawn along the xz plane with the starting point being the current point.

Error

Not enough memory, arithmetic overflow.

See Also

AREAPT, LINFUN, LINPAT, POLY3, PRMFIL

Primitive Fill PRMFIL

Command

- Long Form: PRMFIL flag
- Short Form: PF flag
- Hex Form: E9 flag

Parameter Type

- flag = Char [0..1]

Description

The PRMFIL command sets the primitive fill flag to flag. When PRMFIL is set to 0, closed figures are drawn in outline only; when PRMFIL is set to 1, closed figures are filled with the current color in the current area pattern. PRMFIL affects the following commands:

CIRCLE ELIPSE POLY POLYR
POLY3 POLYR3 RECT RECTR
SECTOR SCIRC SELIPS SPOLY
SPOLYR SRECT SRECTR SSECT

Example

CODE:

ASCII: PF 0

HEX: E9 00

RESULT: Closed figures are drawn in outline only.

Error

None

See Also

AREAPT, BCOLOR, COLOR, COLMOD

PROJCT — Projection

Command

- Long Form: PROJCT angle
- Short Form: PRO angle
- Hex Form: B0 angle

Parameter Type

- angle = Int [0..179]

Description

The PROJCT command sets the viewing angle used in 3D to 2D transformations. When angle is 0°, an orthogonal projection is produced; otherwise, a perspective projection is produced. The default is 60°.

Example

CODE:

```
ASCII: PRO 0
HEX: B0 00 00
```

RESULT: Orthogonal projections are produced.

Error

Value out of range, arithmetic overflow.

See Also

DISTAN

Raster Operations **RASTOP**

Command

- Long Form: RASTOP **oper srcdir destdir** x_0 x_1 y_0 y_1 x'_0 y'_0
- Short Form: ROP **oper srcdir destdir** x_0 x_1 y_0 y_1 x'_0 y'_0
- Hex Form: DA **oper srcdir destdir** x_0 x_1 y_0 y_1 x'_0 y'_0

Parameter Type

- **oper** = Char [0..15]
- **srcdir** = Char [0..7]
- **destdir** = Char [0..7]
- x_0 = Unsigned Int [0..639] or [0..2047] depending on the state of the expand flag
- x_1 = Unsigned Int [0..639] or [0..2047] depending on the state of the expand flag
- y_0 = Unsigned Int [0..479] or [0..1023] depending on the state of the expand flag
- y_1 = Unsigned Int [0..479] or [0..1023] depending on the state of the expand flag
- x'_0 = Unsigned Int [0..639] or [0..2047] depending on the state of the expand flag
- y'_0 = Unsigned Int [0..479] or [0..1023] depending on the state of the expand flag

Description

The **RASTOP** command copies a rectangular area of the screen space, with the lower left corner (x_0, y_0) and upper right corner (x_1, y_1) (specified in pixels), to another area of the screen space starting at the lower left corner (x'_0, y'_0). The corners are included in the region. All bit planes are copied (subject to normal masking as specified by the **MASK** command). If the rectangles overlap, the user must select appropriate major and minor directions to ensure that the area is copied properly. The raster operation function is selected according to the following table and performed on a pixel by pixel basis on the source and the destination regions.

Raster Operation Functions		
Function	Operation	Mode Name
0	D = S	Copy
1	D = S | D	OR
2	D = S & D	AND
3	D = S ^ D	XOR

The direction of scanning of the source (input) region is specified by **srcdir**; the direction of scanning of the destination (output) is specified by **destdir**. Both are selected using the following table:

RASTOP Raster Operations

Scanning Direction		
direction	Major Direction	Minor Direction
0	⇒	↑
1	⇒	↓
2	⇐	↑
3	⇐	↓
4	⇑	→
5	⇓	→
6	⇑	←
7	⇓	←

If the source and the destination scanning directions are both equal to each other and are equal to 0, 1, 2, or 3, the raster operation will be able to use a special mode that greatly increases its speed.

Example CODE:

```
ASCII: ROP 0 0 0 320 639 240 479 0 0
HEX: DA 00 00 00 40 01 7F 02 F0 00 DF
01 00 00 00 00
```

RESULT: The upper right side of the screen is duplicated at the lower left.

Error Invalid operation, Invalid direction, Will not fit on screen.

Raster Read RASTRD

Command

- Long Form: RASTRD dir x_0 x_1 y_0 y_1
- Short Form: RRD dir x_0 x_1 y_0 y_1
- Hex Form: DB dir x_0 x_1 y_0 y_1

Parameter Type

dir = Char [0..7]
x_0 = Unsigned Int [0..639]
x_1 = Unsigned Int [0..639]
y_0 = Unsigned Int [0..479]
y_1 = Unsigned Int [0..479]

Description

The RASTRD command copies a rectangular area of the screen, with corners $\{x_0,y_0\}$ and $\{x_1,y_1\}$ to the output port.

The corners of the area, specified in pixels, are included in the region and all bit planes are copied (subject to normal masking as specified by the **mask** command). The coordinates specified cannot exceed the current screen size.

This command will transfer $(x_1-x_0+1) \times (y_1-y_0+1)$ bytes. Until all data has been transferred, no commands will be interpreted by the QG–640. To abort an incomplete RASTRD, issue a cold reset by writing a 1 to the Cold Reset Flag.

RASTRD Raster Read

The direction of scanning the region is specified according to the following table:

Scanning Direction		
Direction	Major Direction	Minor Direction
0	⇒	↑
1	⇒	↓
2	⇐	↑
3	⇐	↓

Notes:

- On revision level 1 boards and up, the data read back with the RASTRD command is in hex bytes regardless of the communications mode used.
- During and following the processing of a RASTRD command, the value of the Error Flag bit in the Status Register becomes undefined. The Error Flag bit only assumes a significant value the next time the Data Out Register is written to. Therefore, it is also essential to read the value of the Data Out Register Full bit of the Status Register when interpreting the Error Flag bit.

Example CODE:

ASCII: RRD 0 0 511 0 511

HEX: DB 00 00 00 FF 01 00 00 FF 01

RESULT: Entire screen is read when 512 × 512 mode.

Error Value out of range.

See Also rastwr

Raster Write **RASTWR**

Command

- Long Form: RASTWR oper dir x_0 x_1 y_0 y_1
- Short Form: RWR oper dir x_0 x_1 y_0 y_1
- Hex Form: DC oper dir x_0 x_1 y_0 y_1

Parameter Type

oper = Char [0..3]
dir = Char [0..7]
x_0 = Unsigned Int [0..639]
x_1 = Unsigned Int [0..639]
y_0 = Unsigned Int [0..479]
y_1 = Unsigned Int [0..479]

Description

The RASTWR command copies a rectangular area of the screen, with corners $\{x_0,y_0\}$ and $\{x_1,y_1\}$ from the command FIFO.

The corners of the area, specified in pixels, are included in the region. All bit planes are copied (subject to normal masking as specified by the MASK command). The specified coordinates cannot exceed the current screen size.

The pixel combination operation performed (between old and new pixels) is specified by oper using the following table. Operation 0 will not use the old pixels, but will directly copy new pixel data into the screen memory.

Raster Write Function	
Oper.	**Operation**
0	replace
1	or ($\vee$)
2	and ($\wedge$)
3	xor ($\oplus$)

Note that existing pixels are read using MASK.

This command will transfer $(x_1 - x_0 + 1) \times (y_1 - y_0 + 1)$ bytes. Until this data has been transferred, no commands will be interpreted by the QG–640. To abort an incomplete RASTWR, issue a cold reset.

RASTWR Raster Write

The direction of scanning the region is specified according to the following table:

Scanning Direction		
dir	Major Direction	Minor Direction
0	⇒	↑
1*	⇒	↓
2*	⇐	↑
3*	⇐	↓

* Applicable only for oper = 0

Notes:

- On revision level 1 boards and up, the pixel data sent by the Host must be in hex only. Therefore, in ASCII communications mode (see example below), " RWR 0 0 0 639 0 479 " must be sent in ASCII, followed by 640×480 hex bytes.
- During and following the processing of a RASTWR command, the value of the Error Flag bit in the Status register becomes undefined. The Error Flag bit will only assume a significant value the next time the Data Out register is written to. Therefore, it is also essential to read the value of the Data Out Register Full bit of the Status register when interpreting the Error Flag bit.

Example CODE:

CODE:

ASCII: RWR 0 0 0 639 0 479

HEX: `DC 00 00 00 00 7F 02 00 00 DF 01`

RESULT: The entire screen is written from bus memory when in 640×480 mode.

Error Value out of range.

See Also RASTRD

Rubber Band Cross Hair RBAND

Command

- Long Form: RBAND flag
- Short Form: RB flag
- Hex Form: E1 flag

Parameter Type

- flag = Char [0..2]

Description

The RBAND command enables the rubber band vector (flag = 1), the rubber band rectangle (flag = 2), or disables both (flag = 0).

The cursor coordinates, at the time when either the rubber band vector or the rubber band rectangle is enabled, becomes the anchor point. When a new set of cursor coordinates is entered, a vector or a rectangle is drawn from the anchor to the new coordinates in complement mode. As the coordinates are changed the vector or rectangle is erased and redrawn from the anchor to the new cursor coordinates. When the rubber band is disabled, the vector or rectangle last drawn is erased and the cursor coordinate is left at the last coordinate pair entered.

When first enabled, the anchor and the cursor coordinate will be on the same point and the rubber band vector or rectangle will be drawn as a point.

Example

CODE:

ASCII: RB 2

HEX: E1 02

RESULT: The rubber band rectangle is enabled.

Error

Value out of range

See Also

XHAIR, XMOVE, XRECT

RDEFIN Raster Font Define

Command

- Long Form: RDEFIN font height width size start_char array
- Short Form: RDF font height width size start_char array
- Hex Form: 54 font height width size start_char array

Parameter Type

- font = Char [1..15]
- height = Char [0..16]
- width = Char [0..16]
- size = Char
- start_char = Char
- array = array of Char

Description

The user-definable raster fonts 1 to 15 are defined using the RDEFIN command. Each character in the font must have the same cell size, subject to the height and width parameters. The number of characters in the font, minus one, is specified by size and the ASCII code of the first character in the font is specified by start_char. In Hex mode, each row of a character cell is represented by a left justified packed string of bits, each bit representing one pixel.

Raster Font Define RDEFIN

Example CODE:

```
ASCII: RDEFIN 1 7 5 1 65 0 1 1 1 0
1 0 0 0 1
1 0 0 0 1
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
1 1 1 1 0
1 0 0 0 1
1 0 0 0 1
1 1 1 1 0
1 0 0 0 1
1 0 0 0 1
1 1 1 1 0
HEX: 54 01 07 05 01 41 70 88
88 F8 88 88 88 F0 88 88
F0 88 88 F0
```

RESULT: Font 1 is defined with two characters: A and B.

Error Parameter range

See Also RFONT, TEXTP, TEXTPC

RECT Rectangle

Command

- Long Form: RECT x y
- Short Form: R x y
- Hex Form: 34 x y

Parameter Type

- x = Real
- y = Real

Description

The RECT command draws a rectangle with one corner on the 2D current point and the diagonally opposite corner on (x, y). When the PRMFIL flag is set, the rectangle will be drawn filled; if PRMFIL is not set, drawing will be subject to LINPAT. The 2D current point remains unchanged.

Example

CODE:

```
ASCII: R 128 64
HEX: 34 80 00 00 00 40 00 00 00
```

RESULT: A rectangle is drawn with one corner on the 2D current point and the other on (128, 64).

Error

None

See Also

AREAPT, LINFUN, LINPAT, PRMFIL, RECTR

Rectangle Relative RECTR

Command

- Long Form: RECTR Δx Δy
- Short Form: RR Δx Δy
- Hex Form: 35 Δx Δy

Parameter Type

- Δx = Real
- Δy = Real

Description

The RECTR command draws a rectangle with one corner on the 2D current point and the diagonally opposite corner displaced from the 2D current point by (Δx, Δy). When the PRMFIL flag is set, the rectangle will be drawn filled; if PRMFIL is not set, drawing will be subject to LINPAT. The 2D current point remains unchanged.

Example

CODE:

```
ASCII: RR 128 64
HEX: 35 80 00 00 00 40 00 00 00
```

RESULT: A rectangle is drawn with one corner on the 2D current point and the diagonally opposed corner displaced by (128, 64).

Error

Arithmetic overflow

See Also

AREAPT, LINFUN, LINPAT, PRMFIL, RECT

RESETF Reset Flags

Command

- Long Form: RESETF␣
- Short Form: RF␣
- Hex Form: 04

Parameter Type None

Description

The RESETF command resets all flags and parameters to their default values, as specified in the following table. This is done automatically when the board is reset or the power turned on.

Example CODE:

ASCII: RF␣
HEX: 04

RESULT: All flags are reset.

Error None

See Also FLAGRD

Reset Flags RESETF

Flag	Nmae	Default Value	Description
1	AREAPT	65535 16 times	Solid area
2	CLIPH	0	Disabled
3	CLIPY	0	Disabled
4	COLOR	255	
6	DISTAN	500	
7	DISTH	-30000	
8	DISTY	30000	
9	FILMSK	255	All planes used
10	LINFUN	0	Set mode
11	LINPAT	65535	Solid lines
12	MASK	255	All planes on
13	MDORG	(0,0,0)	
14	2D current point	(0,0)	
15	3D current point	(0,0)	
16	PRMFIL	0	Off
17	PROJCT	60	
18	TANGLE	0	Horizontal
19	TJUST	1,1	Left, bottom
20	TSIZE	8	8× 12 cells
21	VWPORT	0,639*,0,479*	Entire screen
22	VWRPT	(0,0,0)	
23	WINDOW	-320,319,-240,239*	
24	Transformed 3D point	(0,0,0)	
25	none	none	Used in FLAGRD
26	XHAIR - current pt on screen	(320,240)*	
27	XHAIR - current pt in 2D	(0,0)	
28	Screen Current Pt	(320,240)*	
29	none	none	Used in FLAGRD
30	none	none	Used in FLAGRD
31	none	none	Used in FLAGRD
32	TSTYLE	0	'Fat' text
33	TASPCT	1.5	
34	TCHROT	0	
35	none	none	Used in FLAGRD
36	VDISP	0	
37	PMASK	255**	All LUT bits enabled
38	ERROR	none	Used in FLAGRD
39	Display Format	640*,480*,60**,0*	
41	COLMOD	1	Transparent
42	BCOLOR	0	Transparent

* These values are determined by straps on the QG–640 circuit board.

** These values are set only on reset and power up.

RFONT Select User Raster Font

Command

- Long Form: RFONT font h_aspect w_aspect
- Short Form: RFT font h_aspect w_aspect
- Hex Form: 55 font h_aspect w_aspect

Parameter Type]

- font = Char [0..15]
- h_aspect = Char [0..1]
- w_aspect = Char [0..1]

Description]

The RFONT command selects the font that will be used to draw user definable raster characters on the screen, using the TEXTP and TEXTPC commands. The font must have been previously defined using either the RDEFIN or TDEFIN commands.

The w_aspect and h_aspect parameters specify the aspect ratio of the characters. A value of 0 indicates single height/width and a value of 1 indicates double height/width.

Example CODE:

```
ASCII: RFONT 1 1 0
HEX: 55 01 01 00
```

RESULT: Font 1 will be selected when using the TEXTP and TEXTPC commands, in double height, and single width aspect ratio.

Error Parameter range

See Also RFONT, TEXTP, TEXTPC

Screen Arc SARC

Command

- Long Form: SARC **radius angle1 angle2**
- Short Form: SAR **radius angle1 angle2**
- Hex Form: **F4 radius angle1 angle2**

Parameter Type

- **radius** = Int
- **angle1** = Int
- **angle2** = Int

Description

The SARC command draws a circular arc using the currently selected color. The center is on the 2D current point. The radius, and start and finish angles are specified in the command. The angles can be any Int value (angles greater than 360° and less than -360° are handled as modulo 360). Negative radii will result in 180° being added to both angles. This command does not affect the 2D current point.

Note: For this command to function correctly, the viewport and the window must have exactly the same coordinates and must be equal to the maximum XY dimensions of the screen space; $x = 639$ and $y = 479$ if the viewport is full screen. The coordinates of the points you are operating on must be visible on the screen, and $0 \leq x \leq 639$; $0 \leq y \leq 479$. See Section 4.10.

Example

CODE:

```
ASCII: SAR 100 0 180
HEX: F4 64 00 00 00 B4 00
```

RESULT: An arc with radius 100 from 0° to 180° (a semi-circle) is drawn about the 2D current point.

Error Overflow

See Also SCIRC, COLOR, LINFUN, LINPAT

SBLINK — Stop Blink

Command

- Long Form: SBLINK␣
- Short Form: SBL␣
- Hex Form: **E4**

Parameter Type
None

Description

The SBLINK command sets all LUT entries currently assigned as blinking, by either the BLINK command or the BLINKX command, as static. If you only want to cancel blinking of one LUT entry, you can still use the BLINK and BLINKX commands. The SBLINK command is useful when you want to stop all blinking on the screen with one instruction.

All blinking colors are restored to their original color.

Example
CODE:

ASCII: SBL␣
HEX: **E4**

RESULT: All blinking pixels, if any, will stop blinking.

Error
None

See Also
BLINK, BLINKX

Screen Circle **SCIRC**

Command

- Long Form: SCIRC **radius**
- Short Form: SCI **radius**
- Hex Form: **F2 radius**

Parameter Type

- **radius** = Int

Description

The **SCIRC** command draws a circle with radius **radius** centered on the 2D current point. The circle is filled if the **PRMFIL** flag is set. This command does not affect the 2D current point.

Note: For this command to function correctly, the viewport and the window must have exactly the same coordinates and must be equal to the maximum XY dimensions of the screen space; x = 639 and y = 479 if the viewport is full screen. The coordinates of the points you are operating on must be visible on the screen, and $0 \leq x \leq 639$; $0 \leq y \leq 479$. See Section 4.10.

Example CODE:

```
ASCII: SCI 100
HEX: F2 64 00
```

RESULT: A circle with radius 100 is drawn from the 2D current point.

Error Overflow

See Also **SARC, SELIPS, LINFUN, LINPAT, PRMFIL, SSECT**

SDRAW Screen Draw

Command

- Long Form: SDRAW x y
- Short Form: SD x y
- Hex Form: FA x y

Parameter Type

- x = Int
- y = Int

Description

The SDRAW command draws a line from the 2D current point to (x, y) and positions the 2D current point to (x, y). This command does not draw the last pixel of a line.

Note: For this command to function correctly, the viewport and the window must have exactly the same coordinates and must be equal to the maximum XY dimensions of the screen space; x = 639 and y = 479 if the viewport is full screen. The coordinates of the points you are operating on must be visible on the screen, and $0 \leq x \leq 639$; $0 \leq y \leq 479$. See Section 4.10.

Example

CODE:

```
ASCII: SD 10 12
HEX: FA 0A 00 0C 00
```

RESULT: A line is drawn from the 2D current point to (10, 12).

Error Arithmetic overflow

See Also SDRAWR, LINFUN, LINPAT, SMOVE, SMOVER

Screen Draw Relative SDRAWR

Command

- Long Form: SDRAWR Δx Δy
- Short Form: SDR Δx Δy
- Hex Form: FB Δx Δy

Parameter Type

- Δx = Int
- Δy = Int

Description

The SDRAWR command draws a line from the 2D current point to {(Δx, Δy) + 2D current point}. The 2D current point is moved to the end of the line. This command does not draw the last pixel of a line.

Note: For this command to function correctly, the viewport and the window must have exactly the same coordinates and must be equal to the maximum XY dimensions of the screen space; $x = 639$ and $y = 479$ if the viewport is full screen. The coordinates of the points you are operating on must be visible on the screen, and $0 \leq x \leq 639$; $0 \leq y \leq 479$. See Section 4.10.

Example

CODE:

```
ASCII: SDR 100 200
HEX: FB 64 00 C8 00
```

RESULT: A line is drawn from the 2D current point to {2D current point + (100, 200)}.

Error

Arithmetic overflow

See Also

SDRAW, LINFUN, LINPAT, SMOVE, SMOVER

SECTOR Sector

Command

- Long Form: SECTOR **radius angle1 angle2**
- Short Form: S **radius angle1 angle2**
- Hex Form: 3D **radius angle1 angle2**

Parameter Type

- **radius** = Real
- **angle1** = Int
- **angle2** = Int

Description

The **SECTOR** command draws a pie shaped figure with the center on the current point, radius **radius**, and angles **angle1** and **angle2**. If **PRMFIL** is set, the sector will be filled, otherwise drawing will be subject to **LINPAT**. If **radius** is negative, 180° will be added to both angles. The angles are integers and are treated as modulo 360. This command does not affect the current point.

Example

CODE:

```
ASCII: S 50.25 45 135
HEX: 3D 32 00 00 40 2D 00 87 00
```

RESULT: A pie shaped sector is drawn with radius 50.25, starting at 45° and ending at 135°.

Error

Arithmetic overflow

See Also

AREAPT, LINFUN, LINPAT, PRMFIL

Screen Ellipse **SELIPS**

Command

- Long Form: SELIPS **xradius yradius**
- Short Form: SEL **xradius yradius**
- Hex Form: F3 **xradius yradius**

Parameter Type

- **xradius** = Int
- **yradius** = Int

Description

The SELIPS command draws a 2D ellipse centered on the 2D current point and whose x and y radii are given by **xradius** and **yradius**. The ellipse will be filled if drawn while the PRMFIL flag is set. This command does not affect the 2D current point.

Note: For this command to function correctly, the viewport and the window must have exactly the same coordinates and must be equal to the maximum XY dimensions of the screen space; x = 639 and y = 479 if the viewport is full screen. The coordinates of the points you are operating on must be visible on the screen, and $0 \leq x \leq 639$; $0 \leq y \leq 479$. See Section 4.10.

Example

CODE:

```
ASCII: SEL 32 128
HEX: F3 20 00 80 00
```

RESULT: An ellipse is drawn with x radius 32 and y radius 128.

Error

Overflow

See Also

AREAPT, LINFUN, LINPAT, PRMFIL

SMOVE — Screen Move

Command

- Long Form: SMOVE x y
- Short Form: SM x y
- Hex Form: F8 x y

Parameter Type

- x = Int
- y = Int

Description

The SMOVE command moves the 2D current point to (x, y).

Note: For this command to function correctly, the viewport and the window must have exactly the same coordinates and must be equal to the maximum XY dimensions of the screen space; $x = 639$ and $y = 479$ if the viewport is full screen. The coordinates of the points you are operating on must be visible on the screen, and $0 \leq x \leq 639$; $0 \leq y \leq 479$. See Section 4.10.

Example

CODE:

```
ASCII: SM 10 12
HEX: F8 0A 00 0C 00
```

RESULT: The 2D current point is moved to (10, 12).

Error

Arithmetic overflow

See Also

SMOVER

Screen Move Relative SMOVER

Command

- Long Form: SMOVER Δx Δy
- Short Form: SMR Δx Δy
- Hex Form: F9 Δx Δy

Parameter Type

- Δx = Int
- Δy = Int

Description

The SMOVER command moves the 2D current point to {(Δx, Δy) + 2D current point}.

Note: For this command to function correctly, the viewport and the window must have exactly the same coordinates and must be equal to the maximum XY dimensions of the screen space; $x = 639$ and $y = 479$ if the viewport is full screen. The coordinates of the points you are operating on must be visible on the screen, and $0 \leq x \leq 639$; $0 \leq y \leq 479$. See Section 4.10.

Example

CODE:

```
ASCII: SMR 10 12
HEX: F9 0A 00 0C 00
```

RESULT: The current point is moved to {(10, 12) + 2D current point}.

Error Arithmetic overflow

See Also SMOVE

SPOLY Screen Polygon

Command

- Long Form: SPOLY n x_1 y_1 x_2 y_2 . . . x_n y_n
- Short Form: SP n x_1 y_1 x_2 y_2 . . . x_n y_n
- Hex Form: FC n x_1 y_1 x_2 y_2 . . . x_n y_n

Parameter Type

- n = Char
- x_i = Int
- y_i = Int

Description

The SPOLY command draws a closed polygon directly on the screen. n is the number of vertices and (x_i, y_i) the coordinates of the vertices. The polygon will be filled if the PRMFIL flag is set and subject to the LINPAT if PRMFIL is not set. The 2D current point will not be changed.

Note: For this command to function correctly, the viewport and the window must have exactly the same coordinates and must be equal to the maximum XY dimensions of the screen space; x = 639 and y = 479 if the viewport is full screen. The coordinates of the points you are operating on must be visible on the screen, and $0 \leq x \leq 639$; $0 \leq y \leq 479$. See Section 4.10.

Example CODE:

```
ASCII: SP 4 0 0 16 0 16 16 0 16
HEX: FC 04 00 00 00 00 00 10 00 00 00 10 00 10 00 00 00 10 00
```

RESULT: A square 16× 16 is drawn.

Error Not enough memory, arithmetic overflow

See Also AREAPT, LINFUN, LINPAT, SPOLYR, PRMFIL

Polygon Relative **SPOLYR**

Command

- Long Form: SPOLYR n Δx_1 Δy_1 Δx_2 Δy_2 $\cdots$ Δx_n Δy_n
- Short Form: SPR n Δx_1 Δy_1 Δx_2 Δy_2 $\cdots$ Δx_n Δy_n
- Hex Form: FD n Δx_1 Δy_1 Δx_2 Δy_2 $\cdots$ Δx_n Δy_n

Parameter Type

- n = Char
- Δx_i = Int
- Δy_i = Int

Description

The SPOLYR command draws a closed polygon directly to the screen. Parameter n is the number of vertices and (Δx_i, Δy_i) the displacements of the vertices from the 2D current point. The polygon will be filled if the PRMFIL flag is set and subject to the LINPAT if PRMFIL is not set. The 2D current point will not be changed.

Note: For this command to function correctly, the viewport and the window must have exactly the same coordinates and must be equal to the maximum XY dimensions of the screen space; x = 639 and y = 479 if the viewport is full screen. The coordinates of the points you are operating on must be visible on the screen, and $0 \leq x \leq 639$; $0 \leq y \leq 479$. See Section 4.10.

Example

CODE:

```
ASCII: SPR 4 0 0 16 0 16 16 0 16
HEX: FD 04 00 00 00 00 10 00 00 00
        10 00 10 00 00 00 10 00
```

RESULT: A 16× 16 square is drawn, with the upper left corner on the 2D current point.

Error

Not enough memory, arithmetic overflow

See Also

AREAPT, LINFUN, LINPAT, SPOLY, PRMFIL

SRECT Screen Rectangle

Command

- Long Form: SRECT x y
- Short Form: SR x y
- Hex Form: F0 x y

Parameter Type

- x = Int
- y = Int

Description

The SRECT command draws a rectangle with one corner on the 2D current point and the diagonally opposite corner on (x, y). When the PRMFIL flag is set, the rectangle will be drawn filled. If PRMFIL is not set, drawing will be subject to LINPAT. The 2D current point remains unchanged.

Note: For this command to function correctly, the viewport and the window must have exactly the same coordinates and must be equal to the maximum XY dimensions of the screen space; $x = 639$ and $y = 479$ if the viewport is full screen. The coordinates of the points you are operating on must be visible on the screen, and $0 \leq x \leq 639$; $0 \leq y \leq 479$. See Section 4.10.

Example

CODE:

```
ASCII: SR 128 64
HEX: F0 80 00 40 00
```

RESULT: A rectangle is drawn with one corner on the 2D current point and the other on (128, 64).

Error

None

See Also

AREAPT, LINFUN, LINPAT, PRMFIL, SRECTR

Screen Rectangle Relative **SRECTR**

Command

- Long Form: SRECTR Δx Δy
- Short Form: SRR Δx Δy
- Hex Form: F1 Δx Δy

Parameter Type

- Δx = Int
- Δy = Int

Description

The SRECTR command draws a rectangle with one corner on the 2D current point and the diagonally opposite corner displaced from the 2D current point by (Δx, Δy). When the PRMFIL flag is set, the rectangle will be drawn filled. If PRMFIL is not set, then the drawing will be subject to LINPAT. The 2D current point remains unchanged.

Note: For this command to function correctly, the viewport and the window must have exactly the same coordinates and must be equal to the maximum XY dimensions of the screen space; x = 639 and y = 479 if the viewport is full screen. The coordinates of the points you are operating on must be visible on the screen, and $0 \leq x \leq 639$; $0 \leq y \leq 479$. See Section 4.10.

Example

CODE:

ASCII: SRR 128 64

HEX: F1 80 00 40 00

RESULT: A rectangle is drawn with one corner on the 2D current point and the other on (128, 64).

Error

Arithmetic overflow

See Also

AREAPT, LINFUN, LINPAT, PRMFIL, SRECTR

SSECT Screen Sector

Command

- Long Form: SSECT radius angle1 angle2
- Short Form: SS radius angle1 angle2
- Hex Form: F5 radius angle1 angle2

Parameter Type

- radius = Int
- angle1 = Int
- angle2 = Int

Description

The SSECT command draws a pie shaped figure with center on the 2D current point, radius radius, and angles angle1 and angle2. If PRMFIL is set, the sector will be filled; otherwise, drawing will be subject to LINPAT. If radius is negative, 180° will be added to both angles. The angles are integers and are treated as modulo 360. This command does not affect the 2D current point.

Note: For this command to function correctly, the viewport and the window must have exactly the same coordinates and must be equal to the maximum XY dimensions of the screen space; $x = 639$ and $y = 479$ if the viewport is full screen. The coordinates of the points you are operating on must be visible on the screen, and $0 \leq x \leq 639$; $0 \leq y \leq 479$. See Section 4.10.

Example

CODE:

```
ASCII: SS 50 45 135
HEX: F5 32 00 2D 00 87 00
```

RESULT: A pie shaped sector is drawn having radius 50, starting at 45° and going through to 135°.

Error

Arithmetic overflow

See Also

AREAPT, LINFUN, LINPAT, PRMFIL

Self Test STEST

Command

- Long Form: STEST flag
- Short Form: STEST flag
- Hex Form: 62 flag

Parameter Type

- flag = Char [0..255]

Description

The STEST command initiates a self test according to the values in the following table:

flag	TEST
0	Display Test Screen
1	ROM Test
2	RAM Test
3	VRAM Test
4	ACRTC Test
5	Echo Test*
6	Eat Input Test*

* Continuous tests that run until a warm reset is issued.

Results are reported in the output port using the following format:

x	x	x	S	t3	t2	t1	t0

The test number is returned using bits t0 to t3. The S bit is set if the test passes, and is zero if the test fails. In ASCII mode this means that a successfully completed test will return a value equal to or greater than 16.

STEST Self Test

Example CODE:

ASCII: STEST 1
HEX: 62 01

RESULT: The ROM is tested and the result is sent to the output port.

Error Value out of range

Text Angle TANGLE

Command

- Long Form: TANGLE angle
- Short Form: TA angle
- Hex Form: 82 angle

Parameter Type

- angle = Int

Description

The TANGLE command sets the rotation angle for text; specifically the angle of the baseline (the imaginary line that characters are drawn on). The angle is specified by angle. The default is the normal left to right drawing angle 0°. Depending on the TROT flag, the resulting text appears to the right of the current point (in the first quadrant, with the current point being the origin) or it rotates around the current point. TANGLE does not affect the rotation of the individual characters; character rotation is specified using TCHROT.

Example

CODE:

```
ASCII: TA 270
HEX: 82 0E 01
```

RESULT: Characters are drawn vertically top to bottom.

Error

None

See Also

TCHROT, TEXT, TEXTP, TROT

TASPCT Text Aspect Ratio

Command

- Long Form: TASPCT ratio
- Short Form: TASP ratio
- Hex Form: 8B ratio

Parameter Type

- ratio = Real

Description

The TASPCT command sets the text aspect ratio for style 1 characters (see TSTYLE). The aspect ratio is the ratio of character height to width, the default is 1.5 (when TSIZE = 8, this represents a character 12 pixels high by 8 pixels wide). Parameter ratio must be greater than zero.

Example

CODE:

```
ASCII: TASP 2
HEX: 8B 02 00 00 00
```

RESULT: Characters are drawn twice as high as they are wide.

Error

Value out of range.

See Also

TEXT, TEXTP, TSIZE, TSTYLE

Text Character Rotation TCHROT

Command

- Long Form: TCHROT angle
- Short Form: TCR angle
- Hex Form: 8A angle

Parameter Type

- angle = Int

Description

The TCHROT command sets the angle of rotation for characters. Only text style 1 is rotated; text style 0 is unaffected. The rotation is independent of the baseline rotation set by TANGLE. Text styles are selected using TSTYLE.

Example

CODE:

```
ASCII: TCR 1
HEX: 8A 01 00
```

RESULT: Graph Text characters will be rotated to the current baseline rotation angle.

Error

None

See Also

TANGLE, TEXT, TEXTP, TSTYLE

TDEFIN Text Define

Command

- Long Form: TDEFIN n x y array
- Short Form: TD n x y array
- Hex Form: 84 n x y array

Parameter Type

- n = Char
- x = Char
- y = Char
- array = x columns by y rows of Chars (ASCII mode) or x bits packed left justified in y byte sets (Hex mode)

Description

The TDEFIN command defines the character given by n to be an array with character cell size x by y and contents array. In ASCII mode, each pixel in the character cell is represented by either the character "0" or the character "1". Where a pixel is set to "0", the character will be transparent, or the current background color (BCOLOR), depending on the current state of COLMOD. Where the pixel is set to "1", the pixel will be the color index last specified by the COLOR command. In Hex mode, each row of the character cell is represented by a packed string of bits, each bit representing one pixel. These bits are left justified so that the first bit is in the highest bit position.

NOTE: If you specify a value of 0 for either the x or the y parameter, you will delete the character definition.

Example

CODE:

```
ASCII: TD 65 5 7  0 1 1 1 0
                  1 0 0 0 1
                  1 0 0 0 1
                  1 1 1 1 1
                  1 0 0 0 1
                  1 0 0 0 1
                  1 0 0 0 1
HEX: 84 41 05 07 70 88 88 F8 88 88 88
```

RESULT: The letter "A" is defined.

Error

Not enough memory.

See Also

TEXTP, COLMOD

Text TEXT

Command

- Long Form: TEXT 'string' or "string"
- Short Form: T 'string' or "string"
- Hex Form: 80 'string' or "string"

Parameter Type

- string = any number of Chars (up to 640)

Description

The TEXT command writes a text string to the screen, justified about the current point as specified in the last TJUST command. The string may be delimited by either double or single quotes. If no quotes are used, the string will be terminated by the first delimiter encountered. The text will be in the size and style specified by the last TSIZE and TSTYLE commands. When TSTYLE has been set to 0, fat text will be produced; when TSTYLE has been set to 1, thin rotatable text will be produced. If COLMOD = Replace, the character cell will be drawn according to the current LINFUN and BCOLOR parameters.

Note: The fastest character drawing speed is attained when fat text of size 16 (size 8 if in QG-640 mode) is selected, with the left side of the beginning of the string located on 16-pixel multiples (0, 16, 32, ...) along the x-axis.

Example CODE:

ASCII: T 'Hello'
HEX: 80 22 48 65 6C 6C 6F 22

RESULT: **Hello** is printed on the screen.

Error String too long, arithmetic overflow.

See Also TANGLE, TASPCT, TCHROT, TEXTP, TJUST, TSIZE, TSTYLE

TEXTC Fixed Length Text

Command

- Long Form: None
- Short Form: None
- Hex Form: 8C count char char ... char

Parameter Type

- count = Unsigned Int [0..640]
- char = Char

Description

The TEXTC command displays a text string of up to 640 characters. The count parameter specifies the number of characters in the string that follows it. Note that this command is restricted to Hex mode.

Example

CODE:

ASCII: None

HEX: 8C 05 00 41 42 43 44 45

RESULT: The text string "ABCDE" is displayed at the current point.

Error

Out of range.

See Also

TEXT, TANGLE, TSIZE

Text with Programmable Font **TEXTP**

Command

- Long Form: TEXTP 'string' or "string"
- Short Form: TP 'string' or "string"
- Hex Form: 83 'string' or "string"

Parameter Type

- string = any number of Chars (up to 640)

Description

The TEXTP command writes a text string to the screen using programmable fonts. The text will be justified about the current point as specified in the last TJUST command, and be in the style specified in the last TSTYLE command. When TSTYLE is set to zero, the text font defined by TDEFIN is used; when TSTYLE is set to 1, the text defined by GTDEF is used. The string may be delimited by either double or single quotes. If no quotes are used, the string will be terminated by the first delimiter encountered.

Example

CODE:

```
ASCII: TP 'Hello'
HEX: 83 22 48 65 6C 6C 6F 22
```

RESULT: **Hello** is printed on the screen.

Error

String too long, arithmetic overflow.

See Also

TASPCT, TANGLE, TCHROT, TDEFIN, TEXT, TJUST, TSIZE, TSTYLE

TEXTPC Fixed Length Programmable Text

Command

- Long Form: None
- Short Form: None
- Hex Form: 8D count char ... char

Parameter Type

- count = Unsigned Int [0..640]
- char = Char

Description

This command displays a programmable text string at the current point. The count parameter specifies the number of characters in the string that follows. This command is identical to the TEXTC command. Note that this command is restricted to Hex mode.

Example

CODE:

ASCII: None

HEX: 8D 05 00 41 42 43 44 45

RESULT: The programmable text string "ABCDE" is displayed at the current point.

Error

Range

See Also

TEXTP, TANGLE, TSTYLE, TDEFIN, GTDEF

Text Justify TJUST

Command

- Long Form: TJUST **horiz vert**
- Short Form: TJ **horiz vert**
- Hex Form: 85 **horiz vert**

Parameter Type

- **horiz** = Char [1..3]
- **vert** = Char [1..3]

Description

The TJUST command sets horizontal and vertical justification as specified in the table below. The default values are: **horiz** = 1 and **vert** = 1.

TEXT JUSTIFICATION	
VALUE	ACTION
1	Justify on left or bottom
2	Center
3	Justify on top or right

Example

CODE:

ASCII: TJ 2 1

HEX: 85 02 01

RESULT: Output text is centered horizontally about the current point with its bottom on the current point.

Error

Range error

See Also

TEXT, TEXTP

TSIZE Text Size

Command

- Long Form: TSIZE size
- Short Form: TS size
- Hex Form: 81 size

Parameter Type

- size = Real

Description

The TSIZE command sets the text size by specifying the virtual distance from one character to the next. The default value is 8. TSIZE directly sets the width of each character and the height is set using TASPCT (height = width × aspect ratio). The size of fat text will be rounded off to a multiple of eight pixels.

Example

CODE:

```
ASCII: TS 16
HEX: 81 10 00 00 00
```

RESULT: Text size is doubled from default.

Error

Arithmetic overflow

See Also

TASPCT, TEXT, TEXTP, TSTYLE

Text Style TSTYLE

Command

- Long Form: TSTYLE flag
- Short Form: TSTY flag
- Hex Form: 88 flag

Parameter Type

- flag = Char [0..1]

Description

The TSTYLE command sets the style of the text drawn with TEXT or TEXTP commands. When flag is 0, characters will be fat – that is to say the lines forming the characters will become wider as their size is increased by a TSIZE command. When flag is 1, the characters will always be constructed with lines one pixel wide. The default is style 0. The effect of this command is only noticeable when characters are drawn in sizes larger than normal.

Example

CODE:

```
ASCII: TSTY 1
HEX: 88 01
```

RESULT: Thin rotatable text is selected.

Error

None

See Also

TEXT, TEXTP, TSIZE

VDISP — Video Display

Command

- Long Form: VDISP display
- Short Form: VD display
- Hex Form: D5 display

Parameter Type

- display = Char [0..1]

Description

The VDISP command selects the lookup table that will be affected by the following commands:

BLINK	BLINKX	LUT	LUTINT
LUTRD	LUTSAV	LUTX	LUTXRD
PMASK			

The current value of display can be read using a FLAGRD 36 command.

Example CODE:

ASCII: VD 00

HEX: D5 00

RESULT: LUT 0 is selected.

Error Value out of range.

See Also BLINK, BLINKX, LUT, LUTINT, LUTRD, LUTSAV, LUTX, LUTXRD, PMASK

Vertical Frequency VFREQ

Command

- Long Form: VFREQ flag
- Short Form: VFREQ flag
- Hex Form: 61 flag

Parameter Type

- flag = Char [0..1]

Description

The VFREQ command selects a 50 Hz (flag = 1) or 60 Hz (flag = 0) vertical refresh rate. The default upon start-up and cold reset is 60 Hz.

Example CODE:

```
ASII: VFREQ 1
HEX: 61 01
```

RESULT: 50 Hz vertical refresh rate is selected.

Error Value out of range.

See Also FLAGRD

VWIDEN Viewing Identity

Command

- Long Form: VWIDEN
- Short Form: VWI
- Hex Form: A0

Parameter Type None

Description

The VWIDEN command sets the viewing transformation matrix to the identity matrix.

Example CODE:

ASCII: VWI
HEX: A0

RESULT: Viewing matrix is set to the identity matrix.

Error None

Viewing Matrix VWMATX

Command

- Long Form: VWMATX array
- Short Form: VWM array
- Hex Form: A7 array

Parameter Type

- array = 16 Reals

Description

The VWMATX command loads the viewing matrix with the data in array.

Example

CODE:

```
ASCII: VWM  36.25   12.00  128  2
             0      36.75  100  0
            72.5     0     2.5  0
           100.25    0      0   0
HEX: A7  24 00 00 40 0C 00 00 00 80 00 00 00
         02 00 00 00 00 00 00 00 24 00 00 C0
         64 00 00 00 00 00 00 00 52 00 00 80
         00 00 00 00 02 00 00 80 00 00 00 00
         64 00 00 40 00 00 00 00 00 00 00 00
         00 00 00 00
```

RESULT: The viewing matrix is set to the above data.

Error

Arithmetic overflow

VWPORT View Port

Command

- Long Form: VWPORT x_1 x_2 y_1 y_2
- Short Form: VWP x_1 x_2 y_1 y_2
- Hex Form: B2 x_1 x_2 y_1 y_2

Parameter Type

- x_1 = Unsigned Int [0..639]
- x_2 = Unsigned Int [0..639]
- y_1 = Unsigned Int [0..479]
- y_2 = Unsigned Int [0..479]

Description

VWPORT defines a viewport on the screen where drawing can take place. The viewport is measured in pixels from the bottom left corner and clipping is always enabled. x_1 must be less than x_2, and y_1 less than y_2 or else a warning will be generated. The pair that generated the warning will be swapped. A warning is also produced when any coordinate falls outside of the screen boundary.

Example

CODE:

```
ASCII: VWP 0 300 0 100
HEX: B2 00 00 2C 01 00 00 64 00
```

RESULT: Viewport is defined to be from the lower left corner of the screen to (300, 100).

Error

Arithmetic overflow

See Also

WINDOW

Viewing Rotate X Axis VWROTX

Command

- Long Form: VWROTX angle
- Short Form: VWX angle
- Hex Form: A3 angle

Parameter Type angle = Int

Description

The VWROTX command rotates the x component of the viewing matrix by angle.

Example CODE:

```
ASCII: VWX 45
HEX: A3 1D 00
```

RESULT: The x component is rotated by 45°.

Error Arithmetic overflow

See Also VWMATX, VWROTY, VWROTZ

VWROTY Viewing Rotate Y Axis

Command

- Long Form: VWROTY angle
- Short Form: VWY angle
- Hex Form: A4 angle

Parameter Type

- angle = Int

Description

The VWROTY command rotates the y component of the viewing matrix by angle.

Example

CODE:

```
ASCII: VWY 45
HEX: A4 1D 00
```

RESULT: The y component is rotated by 45°.

Error

Arithmetic overflow

See Also

VWMATX, VWROTX, VWROTZ

Viewing Rotate Z Axis **VWROTZ**

Command

- Long Form: VWROTZ angle
- Short Form: VWZ angle
- Hex Form: A5 angle

Parameter Type

- angle = Int

Description

The VWROTZ command rotates the z component of the viewing matrix by angle.

Example CODE:

ASCII: VWZ 45

HEX: A5 1D 00

RESULT: The z component is rotated by 45°.

Error Arithmetic overflow

See Also VWMATX, VWROTX, VWROTY

VWRPT Viewing Reference Point

Command

- Long Form: VWRPT x y z
- Short Form: VWR x y z
- Hex Form: A1 x y z

Parameter Type

- x = Real
- y = Real
- z = Real

Description

The VWRPT command sets the viewing reference point to be (x, y, z). The viewing reference point is the point that the user is looking at.

Example

CODE:

```
ASCII: VWR 100 -25 50
HEX: A1 64 00 00 00 E7 FF 00 00 32 00 00 00
```

RESULT: Viewing reference point is defined to be (100, -25, 50).

Error

Arithmetic overflow

Wait WAIT

Command

- Long Form: WAIT frames
- Short Form: W frames
- Hex Form: 05 frames

Parameter Type

- frames = Unsigned Int

Description

The WAIT command produces a delay of frames frames. The value of frames is expressed in $\frac{1}{60}$ seconds (the maximum value of frames 65535 produces a delay of 18 minutes).

Example

CODE:

```
ASCII: W 60
HEX: 05 3C 00
```

RESULT: A 1 second delay is produced.

Error

None

WINDOW Window

Command

- Long Form: WINDOW x_1 x_2 y_1 y_2
- Short Form: WI x_1 x_2 y_1 y_2
- Hex Form: B3 x_1 x_2 y_1 y_2

Parameter Type

- x_1 = Real
- x_2 = Real
- y_1 = Real
- y_2 = Real

Description

The WINDOW command defines the coordinates of the corners of the window. The window is the section of the virtual workspace that is mapped to the screen viewport area, which is set by the most recent VWPORT command.

Example

CODE:

```
ASCII: WI -25 50 75 100
HEX: B3  E7 FF 00 00 32 00 00 00 96 00
         00 00 400 00 00
```

RESULT: The x and y coordinates are both defined to be from 0 to 64.

Error

Arithmetic overflow, range error.

See Also

VWPORT

Enable Cross Hair **XHAIR**

Command

- Long Form: XHAIR **flag** or XHAIR **flag x_size y_size**
- Short Form: XH **flag** or XH **flag x_size y_size**
- Hex Form: E2 **flag** or E2 **flag x_size y_size**

Parameter Type

- **flag** = Char [0..8]
- **x_size** = Int [0..32767]
- **y_size** = Int [0..32767]

Description

The XHAIR command enables or disables the graphics cursor. When the graphics cursor is enabled, the two parameters **x_size** and **y_size** must be used in order to define the size of the graphics cursor. The graphics cursor will have a horizontal length of **x_size** coordinate units and a vertical length of **y_size** coordinate units. The graphics cursor is displayed in complement form with its center on the position specified by the last XMOVE command. When the graphics cursor is disabled, the **x_size** and **y_size** parameters are not specified – the graphics cursor will no longer be displayed. The **flag** parameter options are shown in the following table:

Flag	Action
0	Disable graphics cursor.
1	Enable cross hair cursor, clipped on screen.
2	Not supported.
3	Enable cross hair cursor, clipped on viewport.
4	Not supported.
5	Enable box outline cursor, clipped on screen.
6	Enable box outline cursor, clipped on viewport.
7	Enable filled box cursor, clipped on screen.
8	Enable filled box cursor, clipped on viewport.

XHAIR Enable Cross Hair

Example CODE:

```
ASCII: XH 1 100 100
HEX: E2 01 64 00 64 00
```

RESULT: The cross hair is enabled and defined to be 100 × 100 with full screen clipping mode.

Error Value out of range.

See Also RBAND, VWPORT, XMOVE

Cross Hair Move XMOVE

Command

- Long Form: XMOVE x y
- Short Form: XM x y
- Hex Form: E3 x y

Parameter Type

- x = Int [0..639]
- y = Int [0..479]

Description

The XMOVE command changes the cross hair or the filled rectangle cursor coordinates to (x, y). The coordinates are specified in screen coordinates.

Example

CODE:

```
ASCII: XM 5 5
HEX: E3 05 00 05 00
```

RESULT: The cross hair coordinate is set to (5, 5).

Error

Value out of range.

See Also

RBAND, XHAIR, XRECT

Appendix A

Installation

A.1 Installation

This appendix provides the information necessary to install your QG–640. Section A.1 gives a suggested procedure to install the QG–640 on the Q-bus. Section A.2 provides brief descriptions of the circuit board straps. Information on the video connectors used is given in Section A.3, and Section A.4 describes the QG–640's status LED's.

For applications requiring modification to the board's straps, set the straps as specified in Section A.2, then return to this section for the installation procedure for the QG–640.

This section gives a suggested procedure to install your QG–640 board in a Q-bus system. It is assumed that you are familiar with the Q-bus, know where the straps are on the QG–640, and are installing a QG–640 with the "as shipped" strapping. For "as shipped" strap locations, refer to Appendix B.

1. Turn off the power on the Q-bus system.
2. Determine the correct slot for the QG–640, according to your DEC Q-bus configuration guide. The QG–640 generates interrupts, so its position in the backplane is important.
3. Determine the required base address in I/O space for the QG–640 in your system. The QG–640 is strapped at 160400 (octal) in the I/O page when shipped. If your system already has a device at this address, restrap the QG–640 to appear at a location suitable for your system.
4. The QG–640 is shipped with an interrupt vector of 240 (octal). If this is unsuitable for your system, find a free interrupt vector for the QG, and strap it accordingly.
5. The QG–640 is shipped to operate at interrupt level 4. If this is unsuitable for your system, choose the interrupt level at which the QG will operate, and strap it accordingly.
6. The QG–640 is shipped to provide video sync on the green video output. If this is unsuitable for your video monitor, remove the straps.
7. The QG–640 as shipped generates a non-interlaced 640×480 display. To obtain a 512×512 display, install strap 30-33. To obtain an interlaced display, remove straps 7-8 and 54-55, and install straps 8-9, 29-32, and 55-56.
8. Install the QG–640 in your backplane, ensuring that its slot has been granted all required signals – i.e. there are no empty slots between the QG and the CPU.
9. Do not connect your video monitor at this time, but power on the Q-bus system. The QG–640 should display one blinking LED, one LED on, and two off.
10. Use ODT to examine the address at which your QG is located. When read, this address should contain the value for the interrupt vector as it is strapped on the QG.
11. Boot your system. If it will not boot, check for empty slots in the backplane, address or interrupt vector conflicts between the QG–640 and any other card in your system.

12. If you have a utility to talk to the QG, use it to issue the FLAGRD 39 command, to make sure that the video format currently active is the one required for your video monitor. If you do not have such a utility, you may wish to use an oscilloscope to verify that the video output is correct for the video monitor.
13. Connect your video monitor.
14. Issue the STEST 0 command to test the video connections.

A.2 Configuration

A number of board parameters are set using straps. The straps must be installed in order for the board to operate. The straps are identified by the numbers of the two berg pins which are connected if the strap is designated IN and unconnected if the strap is OUT. The following pages describe the QG–640 strap configurations and their respective functions.

A.2.1 Sync Selection

These straps enable or disable, on each output channel, a composite sync signal on the green output signal.

Connection	Effect
113-115 IN	Composite sync on green channel (J1 output).
113-115 OUT	No sync on green (J1 output).
112-114 IN	Composite sync on green channel (J2 output).
112-114 OUT	No sync on green (J2 output).

A.2.2 Video Mode

This strap is used to select the pixel size of the output display; it also tells the firmware the size of the frame buffer being used.

Connection	Effect
30-33 IN	512×512 Mode
30-33 OUT	640×480 Mode

These straps select the video mode of the output signal.

Connection	Effect
8-9 IN, 55-56 IN, 29-32 IN	Interlace Mode
7-8 IN, 54-55 IN, 29-32 OUT	Non-interlace Mode

A.2.3 Interrupt Level and Vector

These straps set the interrupt level of the board according to the following table:

Interrupt Level	Pin 61 - Pin 58	Pin 59 - Pin 62
4	IN	IN
5	IN	OUT
6	OUT	IN
7	OUT	OUT

15			12			9			6			3			0
Ø	Ø	Ø	Ø	X	X	X	X	X	X	X	X	X	X	Ø	Ø

The Interrupt Vector.

These straps define the interrupt vector.

Bit	Connection
15	Always 0
14	Always 0
13	Always 0
12	Always 0
11	34-44 OUT = 1, IN = 0 (always in)
10	35-45 OUT = 1, IN = 0 (always in)
9	36-46 OUT = 1, IN = 0 (always in)
8	37-47 OUT = 1, IN = 0
7	38-48 OUT = 1, IN = 0
6	39-49 OUT = 1, IN = 0
5	40-50 OUT = 1, IN = 0
4	41-51 OUT = 1, IN = 0
3	42-52 OUT = 1, IN = 0
2	43-53 OUT = 1, IN = 0
1	Always 0
0	Always 0

A.2.4 I/O Base Address

These straps set the base address of the board.

15			12			9			6			3			0
1	1	1	X	X	X	X	X	X	X	X	Ø	V_2	V_1	Ø	Ø

The I/O Base Address.

Bit	Connection
12	11-20 OUT = 1, IN = 0
11	12-21 OUT = 1, IN = 0
10	13-22 OUT = 1, IN = 0
9	14-23 OUT = 1, IN = 0
8	15-24 OUT = 1, IN = 0
7	16-25 OUT = 1, IN = 0
6	17-26 OUT = 1, IN = 0
5	18-27 OUT = 1, IN = 0

The offsets from the base address for the QG–640 registers are listed below.

V_2	V_1	Write Register	Read Register
0	0	Command Register	Interrupt Vector Strapping Read
0	1	"Don't care"	Status Register
1	0	FIFO	"Don't care"
1	1	"Don't care"	Port Register

A.2.5 Micro-VAX Slot Option

If the QG–640 is to be used in slot 1, 2, or 3 of a Micro-VAX, connections 1-2 and 3-4 must be left OUT. These straps connect the grant in and grant out of the DMA and interrupt of slots C and D.

A.2.6 Factory-Set Straps

The straps which are connected to pins 123 through 140 are factory-configured as follows, and should not be changed:

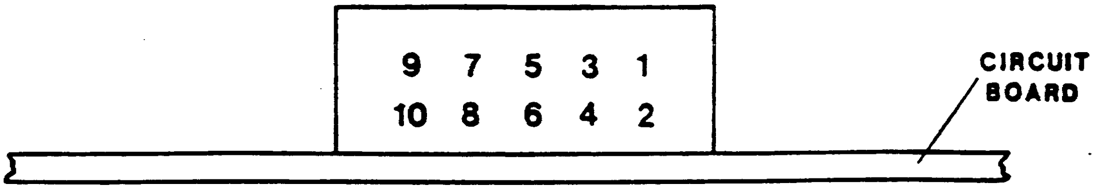
QG-640	QG-640/EMC	QG-640/F
123–124 IN	123–124 IN	123–124 IN
126–127 IN	126–127 IN	126–127 IN
130–131 IN	130–131 IN	129–130 IN
132–133 IN	132–133 IN	133–134 IN

A.3 Connectors

A.3.1 Video Connectors

The video connector is an AMP 1-87382-0. The mating connector is an AMP 87922-1. The pin orientation is shown below.

Pin	Function	Pin	Function
1	+5V (Pull-Up)	6	Gnd
2	No connection	7	Blue
3	Red	8	Gnd (Blue)
4	Gnd (Red)	9	Composite Sync/
5	Green	10	Gnd (Sync)



Looking into the Video Connector

A.4 LEDs

There are four LEDs on the QG–640 which provide information about the board's status. The LEDs are located on the board's component side and are described below from left to right:

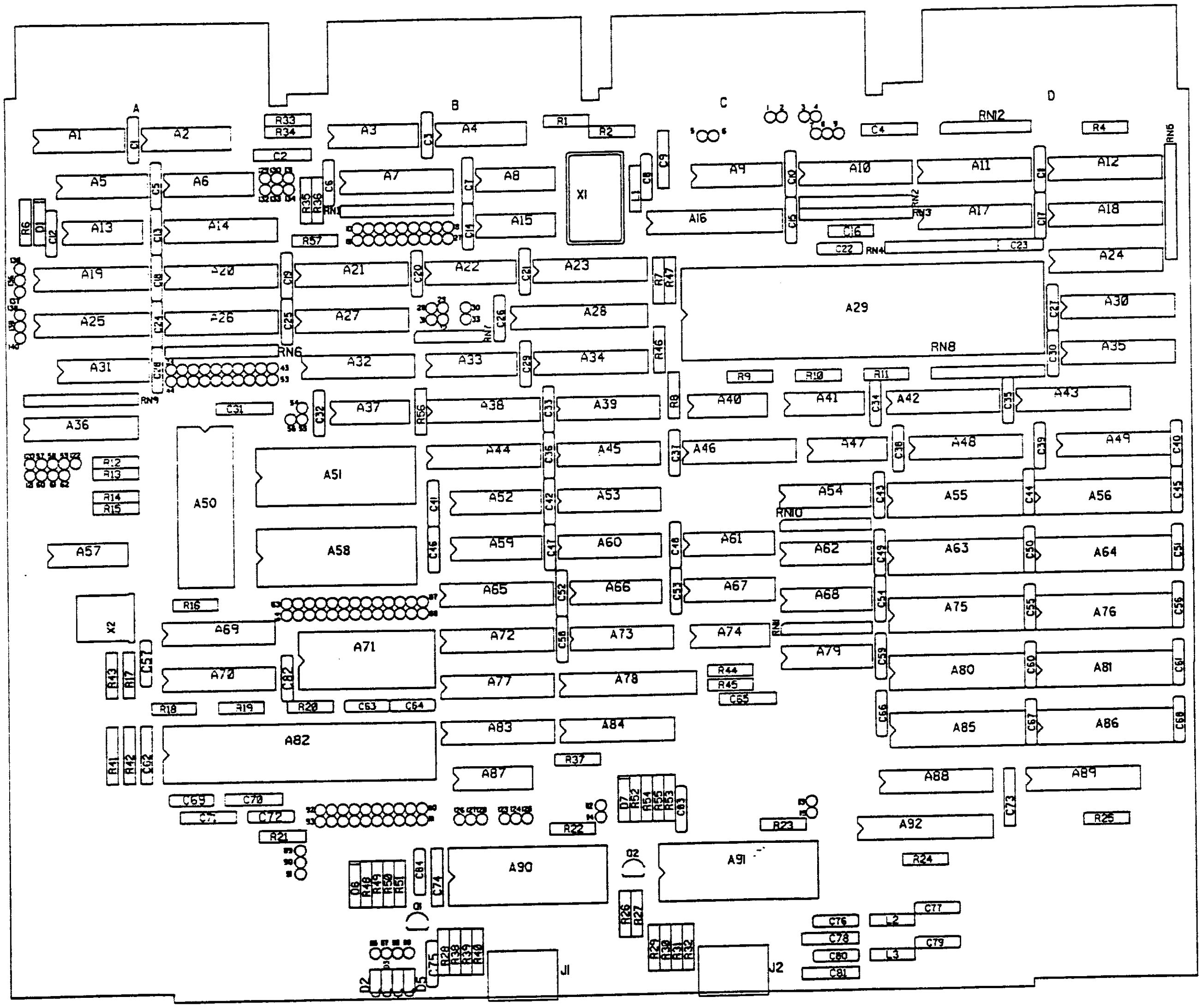
1. Heartbeat: the light blinks on and off to indicate that the board is functioning properly.
2. Data Out Port Full: this light indicates that there is more than one byte of data in the Data Out Register. The QG–640 will cease processing until this data is read.
3. Input Data FIFO Empty: this light indicates that the Input Data FIFO is empty and the board is waiting for input.
4. Error Register Full: this light indicates that there is more than one byte of error data in the Data Out Register. The QG–640 will cease processing until this data is read.

Appendix B

Board Layout

SECTION B.1 Board Layout Schematic Diagram

B.1 Board Layout Schematic Diagram



Appendix C

Lookup Table Data

SECTION C.1 Lookup Table Data

C.1 Lookup Table Data

This chapter contains the lookup table data that is provided in ROM on the QG–640. These tables contain three decimal numbers per entry. The entries are, from left to right: red, green, and blue. These values are given in the format used by the LUTX command (that is, as 8-bit values).

LOOKUP TABLE DATA

```
 State 0 : red, green, blue intensity

Entry 0 :   0,   0,   0
Entry 1 :  16,  16,  16
Entry 2 :  32,  32,  32
Entry 3 :  48,  48,  48
Entry 4 :  64,  64,  64
Entry 5 :  80,  80,  80
Entry 6 :  96,  96,  96
Entry 7 : 112, 112, 112
Entry 8 : 128, 128, 128
Entry 9 : 144, 144, 144
Entry 10 : 160, 160, 160
Entry 11 : 176, 176, 176
Entry 12 : 192, 192, 192
Entry 13 : 208, 208, 208
Entry 14 : 224, 224, 224
Entry 15 : 240, 240, 240
Entry 16 :   0,   0,   0
Entry 17 :  32,   0,   0
Entry 18 :  64,   0,   0
Entry 19 :  96,   0,   0
Entry 20 : 128,   0,   0
Entry 21 : 160,   0,   0
Entry 22 : 192,   0,   0
Entry 23 : 224,   0,   0
Entry 24 : 240,   0,   0
Entry 25 : 240,  32,  32
Entry 26 : 240,  64,  64
Entry 27 : 240,  96,  96
Entry 28 : 240, 128, 128
Entry 29 : 240, 160, 160
Entry 30 : 240, 192, 192
Entry 31 : 240, 224, 224
Entry 32 :   0,   0,   0
Entry 33 :  32,   0,  16
Entry 34 :  64,   0,  32
Entry 35 :  96,   0,  48
Entry 36 : 128,   0,  64
Entry 37 : 160,   0,  80
Entry 38 : 192,   0,  96
Entry 39 : 224,   0, 112
Entry 40 : 240,   0, 128
Entry 41 : 240,  32, 144
Entry 42 : 240,  64, 160
Entry 43 : 240,  96, 176
Entry 44 : 240, 128, 192
Entry 45 : 240, 160, 208
Entry 46 : 240, 192, 224
Entry 47 : 240, 224, 240
Entry 48 :   0,   0,   0
Entry 49 :  32,   0,  32
Entry 50 :  64,   0,  64
Entry 51 :  96,   0,  96
Entry 52 : 128,   0, 128
Entry 53 : 160,   0, 160
Entry 54 : 192,   0, 192
Entry 55 : 224,   0, 224
Entry 56 : 240,   0, 240
Entry 57 : 240,  32, 240
Entry 58 : 240,  64, 240
Entry 59 : 240,  96, 240
Entry 60 : 240, 128, 240
Entry 61 : 240, 160, 240
Entry 62 : 240, 192, 240
Entry 63 : 240, 224, 240
Entry 64 :   0,   0,   0
Entry 65 :  16,   0,  32
Entry 66 :  32,   0,  64
Entry 67 :  48,   0,  96
Entry 68 :  64,   0, 128
Entry 69 :  80,   0, 160
Entry 70 :  96,   0, 192
Entry 71 : 112,   0, 224
Entry 72 : 128,   0, 240
Entry 73 : 144,  32, 240
Entry 74 : 160,  64, 240
Entry 75 : 176,  96, 240
Entry 76 : 192, 128, 240
Entry 77 : 208, 160, 240
Entry 78 : 224, 192, 240
Entry 79 : 240, 224, 240
Entry 80 :   0,   0,   0
Entry 81 :   0,   0,  32
Entry 82 :   0,   0,  64
Entry 83 :   0,   0,  96
Entry 84 :   0,   0, 128
Entry 85 :   0,   0, 160
Entry 86 :   0,   0, 192
Entry 87 :   0,   0, 224
Entry 88 :   0,   0, 240
Entry 89 :  32,  32, 240
Entry 90 :  64,  64, 240
Entry 91 :  96,  96, 240
Entry 92 : 128, 128, 240
Entry 93 : 160, 160, 240
Entry 94 : 192, 192, 240
Entry 95 : 224, 224, 240
Entry 96 :   0,   0,   0
Entry 97 :   0,  16,  32
Entry 98 :   0,  32,  64
Entry 99 :   0,  48,  96
Entry 100 :   0,  64, 128
Entry 101 :   0,  80, 160
Entry 102 :   0,  96, 192
Entry 103 :   0, 112, 224
Entry 104 :   0, 128, 240
Entry 105 :  32, 144, 240
Entry 106 :  64, 160, 240
Entry 107 :  96, 176, 240
Entry 108 : 128, 192, 240
Entry 109 : 160, 208, 240
Entry 110 : 192, 224, 240
Entry 111 : 224, 240, 240
Entry 112 :   0,   0,   0
Entry 113 :   0,  32,  32
Entry 114 :   0,  64,  64
Entry 115 :   0,  96,  96
Entry 116 :   0, 128, 128
Entry 117 :   0, 160, 160
Entry 118 :   0, 192, 192
Entry 119 :   0, 224, 224
Entry 120 :   0, 240, 240
Entry 121 :  32, 240, 240
Entry 122 :  64, 240, 240
Entry 123 :  96, 240, 240
Entry 124 : 128, 240, 240
Entry 125 : 160, 240, 240
Entry 126 : 192, 240, 240
Entry 127 : 224, 240, 240
Entry 128 :   0,   0,   0
Entry 129 :   0,  32,  16
Entry 130 :   0,  64,  32
Entry 131 :   0,  96,  48
Entry 132 :   0, 128,  64
Entry 133 :   0, 160,  80
Entry 134 :   0, 192,  96
Entry 135 :   0, 224, 112
Entry 136 :   0, 240, 128
Entry 137 :  32, 240, 144
Entry 138 :  64, 240, 160
Entry 139 :  96, 240, 176
Entry 140 : 128, 240, 192
Entry 141 : 160, 240, 208
Entry 142 : 192, 240, 224
Entry 143 : 224, 240, 240
Entry 144 :   0,   0,   0
Entry 145 :   0,  32,   0
Entry 146 :   0,  64,   0
Entry 147 :   0,  96,   0
Entry 148 :   0, 128,   0
Entry 149 :   0, 160,   0
Entry 150 :   0, 192,   0
Entry 151 :   0, 224,   0
Entry 152 :   0, 240,   0
Entry 153 :  32, 240,  32
Entry 154 :  64, 240,  64
Entry 155 :  96, 240,  96
Entry 156 : 128, 240, 128
Entry 157 : 160, 240, 160
Entry 158 : 192, 240, 192
Entry 159 : 224, 240, 224
Entry 160 :   0,   0,   0
Entry 161 :  16,  32,   0
Entry 162 :  32,  64,   0
Entry 163 :  48,  96,   0
Entry 164 :  64, 128,   0
```

```
Entry 165 :  80, 160,   0
Entry 166 :  96, 192,   0
Entry 167 : 112, 224,   0
Entry 168 : 128, 240,   0
Entry 169 : 144, 240,  32
Entry 170 : 160, 240,  64
Entry 171 : 176, 240,  96
Entry 172 : 192, 240, 128
Entry 173 : 208, 240, 160
Entry 174 : 224, 240, 192
Entry 175 : 240, 240, 224
Entry 176 :   0,   0,   0
Entry 177 :  32,  32,   0
Entry 178 :  64,  64,   0
Entry 179 :  96,  96,   0
Entry 180 : 128, 128,   0
Entry 181 : 160, 160,   0
Entry 182 : 192, 192,   0
Entry 183 : 224, 224,   0
Entry 184 : 240, 240,   0
Entry 185 : 240, 240,  32
Entry 186 : 240, 240,  64
Entry 187 : 240, 240,  96
Entry 188 : 240, 240, 128
Entry 189 : 240, 240, 160
Entry 190 : 240, 240, 192
Entry 191 : 240, 240, 224
Entry 192 :   0,   0,   0
Entry 193 :  32,  16,   0
Entry 194 :  64,  32,   0
Entry 195 :  96,  48,   0
Entry 196 : 128,  64,   0
Entry 197 : 160,  80,   0
Entry 198 : 192,  96,   0
Entry 199 : 224, 112,   0
Entry 200 : 240, 128,   0
Entry 201 : 240, 144,  32
Entry 202 : 240, 160,  64
Entry 203 : 240, 176,  96
Entry 204 : 240, 192, 128
Entry 205 : 240, 208, 160
Entry 206 : 240, 224, 192
Entry 207 : 240, 240, 224
Entry 208 :   0,   0,   0
Entry 209 :  16,   0,   0
Entry 210 :  48,  16,  16
Entry 211 :  64,  16,  16
Entry 212 :  96,  32,  32
Entry 213 : 112,  32,  32
Entry 214 : 144,  48,  48
Entry 215 : 160,  48,  48
Entry 216 : 192,  64,  64
Entry 217 : 192,  80,  80
Entry 218 : 208, 112, 112
Entry 219 : 208, 128, 128
Entry 220 : 224, 160, 160
Entry 221 : 224, 176, 176
Entry 222 : 240, 208, 208
Entry 223 : 240, 224, 224
Entry 224 :   0,   0,   0
Entry 225 :   0,  16,   0
Entry 226 :  16,  48,  16
Entry 227 :  16,  64,  16
Entry 228 :  32,  96,  32
Entry 229 :  32, 112,  32
Entry 230 :  48, 144,  48
Entry 231 :  48, 160,  48
Entry 232 :  64, 192,  64
Entry 233 :  80, 192,  80
Entry 234 : 112, 208, 112
Entry 235 : 128, 208, 128
Entry 236 : 160, 224, 160
Entry 237 : 176, 224, 176
Entry 238 : 208, 240, 208
Entry 239 : 224, 240, 224
Entry 240 :   0,   0,   0
Entry 241 :   0,   0,  16
Entry 242 :  16,  16,  48
Entry 243 :  16,  16,  64
Entry 244 :  32,  32,  96
Entry 245 :  32,  32, 112
Entry 246 :  48,  48, 144
Entry 247 :  48,  48, 160
Entry 248 :  64,  64, 192
Entry 249 :  80,  80, 192
Entry 250 : 112, 112, 208
Entry 251 : 128, 128, 208
Entry 252 : 160, 160, 224
Entry 253 : 176, 176, 224
Entry 254 : 208, 208, 240
Entry 255 : 224, 224, 240


State 1 : red, green, blue intensity

Entry 0 :  96, 128, 208
Entry 1 :   0,   0,   0
Entry 2 : 112,  64,  32
Entry 3 : 160, 112,  64
Entry 4 : 112,   0,   0
Entry 5 : 240,   0,   0
Entry 6 : 240, 112,   0
Entry 7 : 240, 240,   0
Entry 8 : 160, 240,   0
Entry 9 :   0, 240,   0
Entry 10 :   0, 112,   0
Entry 11 :   0, 112, 112
Entry 12 :   0,   0, 112
Entry 13 : 224, 144,  96
Entry 14 : 112, 112, 112
Entry 15 : 240, 240, 240
Entry 16 :   0,   0,   0
```

```
Entry 17 :   0,   0,   0
Entry 18 :   0,   0,   0
Entry 19 :   0,   0,   0
Entry 20 :   0,   0,   0
Entry 21 :   0,   0,   0
Entry 22 :   0,   0,   0
Entry 23 :   0,   0,   0
Entry 24 :   0,   0,   0
Entry 25 :   0,   0,   0
Entry 26 :   0,   0,   0
Entry 27 :   0,   0,   0
Entry 28 :   0,   0,   0
Entry 29 :   0,   0,   0
Entry 30 :   0,   0,   0
Entry 31 :   0,   0,   0
Entry 32 : 112,  64,  32
Entry 33 : 112,  64,  32
Entry 34 : 112,  64,  32
Entry 35 : 112,  64,  32
Entry 36 : 112,  64,  32
Entry 37 : 112,  64,  32
Entry 38 : 112,  64,  32
Entry 39 : 112,  64,  32
Entry 40 : 112,  64,  32
Entry 41 : 112,  64,  32
Entry 42 : 112,  64,  32
Entry 43 : 112,  64,  32
Entry 44 : 112,  64,  32
Entry 45 : 112,  64,  32
Entry 46 : 112,  64,  32
Entry 47 : 112,  64,  32
Entry 48 : 160, 112,  64
Entry 49 : 160, 112,  64
Entry 50 : 160, 112,  64
Entry 51 : 160, 112,  64
Entry 52 : 160, 112,  64
Entry 53 : 160, 112,  64
Entry 54 : 160, 112,  64
Entry 55 : 160, 112,  64
Entry 56 : 160, 112,  64
Entry 57 : 160, 112,  64
Entry 58 : 160, 112,  64
Entry 59 : 160, 112,  64
Entry 60 : 160, 112,  64
Entry 61 : 160, 112,  64
Entry 62 : 160, 112,  64
Entry 63 : 160, 112,  64
Entry 64 : 112,   0,   0
Entry 65 : 112,   0,   0
Entry 66 : 112,   0,   0
Entry 67 : 112,   0,   0
Entry 68 : 112,   0,   0
Entry 69 : 112,   0,   0
Entry 70 : 112,   0,   0
Entry 71 : 112,   0,   0
Entry 72 : 112,   0,   0
Entry 73 : 112,   0,  0
Entry 74 : 112,   0,  0
Entry 75 : 112,   0,  0
Entry 76 : 112,   0,  0
Entry 77 : 112,   0,  0
Entry 78 : 112,   0,  0
Entry 79 : 112,   0,  0
Entry 80 : 240,   0,  0
Entry 81 : 240,   0,  0
Entry 82 : 240,   0,  0
Entry 83 : 240,   0,  0
Entry 84 : 240,   0,  0
Entry 85 : 240,   0,  0
Entry 86 : 240,   0,  0
Entry 87 : 240,   0,  0
Entry 88 : 240,   0,  0
Entry 89 : 240,   0,  0
Entry 90 : 240,   0,  0
Entry 91 : 240,   0,  0
Entry 92 : 240,   0,  0
Entry 93 : 240,   0,  0
Entry 94 : 240,   0,  0
Entry 95 : 240,   0,  0
Entry 96 : 240, 112,  0
Entry 97 : 240, 112,  0
Entry 98 : 240, 112,  0
Entry 99 : 240, 112,  0
Entry 100 : 240, 112,  0
Entry 101 : 240, 112,  0
Entry 102 : 240, 112,  0
Entry 103 : 240, 112,  0
Entry 104 : 240, 112,  0
Entry 105 : 240, 112,  0
Entry 106 : 240, 112,  0
Entry 107 : 240, 112,  0
Entry 108 : 240, 112,  0
Entry 109 : 240, 112,  0
Entry 110 : 240, 112,  0
Entry 111 : 240, 112,  0
Entry 112 : 240, 240,  0
Entry 113 : 240, 240,  0
Entry 114 : 240, 240,  0
Entry 115 : 240, 240,  0
Entry 116 : 240, 240,  0
Entry 117 : 240, 240,  0
Entry 118 : 240, 240,  0
Entry 119 : 240, 240,  0
Entry 120 : 240, 240,  0
Entry 121 : 240, 240,  0
Entry 122 : 240, 240,  0
Entry 123 : 240, 240,  0
Entry 124 : 240, 240,  0
Entry 125 : 240, 240,  0
Entry 126 : 240, 240,  0
Entry 127 : 240, 240,  0
Entry 128 : 160, 240,  0
```

```
Entry 129 : 160, 240,   0
Entry 130 : 160, 240,   0
Entry 131 : 160, 240,   0
Entry 132 : 160, 240,   0
Entry 133 : 160, 240,   0
Entry 134 : 160, 240,   0
Entry 135 : 160, 240,   0
Entry 136 : 160, 240,   0
Entry 137 : 160, 240,   0
Entry 138 : 160, 240,   0
Entry 139 : 160, 240,   0
Entry 140 : 160, 240,   0
Entry 141 : 160, 240,   0
Entry 142 : 160, 240,   0
Entry 143 : 160, 240,   0
Entry 144 :   0, 240,   0
Entry 145 :   0, 240,   0
Entry 146 :   0, 240,   0
Entry 147 :   0, 240,   0
Entry 148 :   0, 240,   0
Entry 149 :   0, 240,   0
Entry 150 :   0, 240,   0
Entry 151 :   0, 240,   0
Entry 152 :   0, 240,   0
Entry 153 :   0, 240,   0
Entry 154 :   0, 240,   0
Entry 155 :   0, 240,   0
Entry 156 :   0, 240,   0
Entry 157 :   0, 240,   0
Entry 158 :   0, 240,   0
Entry 159 :   0, 240,   0
Entry 160 :   0, 112,   0
Entry 161 :   0, 112,   0
Entry 162 :   0, 112,   0
Entry 163 :   0, 112,   0
Entry 164 :   0, 112,   0
Entry 165 :   0, 112,   0
Entry 166 :   0, 112,   0
Entry 167 :   0, 112,   0
Entry 168 :   0, 112,   0
Entry 169 :   0, 112,   0
Entry 170 :   0, 112,   0
Entry 171 :   0, 112,   0
Entry 172 :   0, 112,   0
Entry 173 :   0, 112,   0
Entry 174 :   0, 112,   0
Entry 175 :   0, 112,   0
Entry 176 :   0, 112, 112
Entry 177 :   0, 112, 112
Entry 178 :   0, 112, 112
Entry 179 :   0, 112, 112
Entry 180 :   0, 112, 112
Entry 181 :   0, 112, 112
Entry 182 :   0, 112, 112
Entry 183 :   0, 112, 112
Entry 184 :   0, 112, 112
Entry 185 :   0, 112, 112
Entry 186 :   0, 112, 112
Entry 187 :   0, 112, 112
Entry 188 :   0, 112, 112
Entry 189 :   0, 112, 112
Entry 190 :   0, 112, 112
Entry 191 :   0, 112, 112
Entry 192 :   0,   0, 112
Entry 193 :   0,   0, 112
Entry 194 :   0,   0, 112
Entry 195 :   0,   0, 112
Entry 196 :   0,   0, 112
Entry 197 :   0,   0, 112
Entry 198 :   0,   0, 112
Entry 199 :   0,   0, 112
Entry 200 :   0,   0, 112
Entry 201 :   0,   0, 112
Entry 202 :   0,   0, 112
Entry 203 :   0,   0, 112
Entry 204 :   0,   0, 112
Entry 205 :   0,   0, 112
Entry 206 :   0,   0, 112
Entry 207 :   0,   0, 112
Entry 208 : 224, 144,  96
Entry 209 : 224, 144,  96
Entry 210 : 224, 144,  96
Entry 211 : 224, 144,  96
Entry 212 : 224, 144,  96
Entry 213 : 224, 144,  96
Entry 214 : 224, 144,  96
Entry 215 : 224, 144,  96
Entry 216 : 224, 144,  96
Entry 217 : 224, 144,  96
Entry 218 : 224, 144,  96
Entry 219 : 224, 144,  96
Entry 220 : 224, 144,  96
Entry 221 : 224, 144,  96
Entry 222 : 224, 144,  96
Entry 223 : 224, 144,  96
Entry 224 : 112, 112, 112
Entry 225 : 112, 112, 112
Entry 226 : 112, 112, 112
Entry 227 : 112, 112, 112
Entry 228 : 112, 112, 112
Entry 229 : 112, 112, 112
Entry 230 : 112, 112, 112
Entry 231 : 112, 112, 112
Entry 232 : 112, 112, 112
Entry 233 : 112, 112, 112
Entry 234 : 112, 112, 112
Entry 235 : 112, 112, 112
Entry 236 : 112, 112, 112
Entry 237 : 112, 112, 112
Entry 238 : 112, 112, 112
Entry 239 : 112, 112, 112
Entry 240 : 240, 240, 240
```

```
Entry 241 : 240, 240, 240
Entry 242 : 240, 240, 240
Entry 243 : 240, 240, 240
Entry 244 : 240, 240, 240
Entry 245 : 240, 240, 240
Entry 246 : 240, 240, 240
Entry 247 : 240, 240, 240
Entry 248 : 240, 240, 240
Entry 249 : 240, 240, 240
Entry 250 : 240, 240, 240
Entry 251 : 240, 240, 240
Entry 252 : 240, 240, 240
Entry 253 : 240, 240, 240
Entry 254 : 240, 240, 240
Entry 255 : 240, 240, 240
```

```
 State 2 : red, green, blue intensity

Entry 0 :   0,   0,   0
Entry 1 :   0,   0,  48
Entry 2 :   0,   0,  80
Entry 3 :   0,   0, 112
Entry 4 :   0,   0, 144
Entry 5 :   0,   0, 176
Entry 6 :   0,   0, 208
Entry 7 :   0,   0, 240
Entry 8 :   0,  48,   0
Entry 9 :   0,  48,  48
Entry 10 :   0,  48,  80
Entry 11 :   0,  48, 112
Entry 12 :   0,  48, 144
Entry 13 :   0,  48, 176
Entry 14 :   0,  48, 208
Entry 15 :   0,  48, 240
Entry 16 :   0,  80,   0
Entry 17 :   0,  80,  48
Entry 18 :   0,  80,  80
Entry 19 :   0,  80, 112
Entry 20 :   0,  80, 144
Entry 21 :   0,  80, 176
Entry 22 :   0,  80, 208
Entry 23 :   0,  80, 240
Entry 24 :   0, 112,   0
Entry 25 :   0, 112,  48
Entry 26 :   0, 112,  80
Entry 27 :   0, 112, 112
Entry 28 :   0, 112, 144
Entry 29 :   0, 112, 176
Entry 30 :   0, 112, 208
Entry 31 :   0, 112, 240
Entry 32 :   0, 144,   0
Entry 33 :   0, 144,  48
Entry 34 :   0, 144,  80
Entry 35 :   0, 144, 112
Entry 36 :   0, 144, 144
Entry 37 :   0, 144, 176
Entry 38 :   0, 144, 208
Entry 39 :   0, 144, 240
Entry 40 :   0, 176,   0
Entry 41 :   0, 176,  48
Entry 42 :   0, 176,  80
Entry 43 :   0, 176, 112
Entry 44 :   0, 176, 144
Entry 45 :   0, 176, 176
Entry 46 :   0, 176, 208
Entry 47 :   0, 176, 240
Entry 48 :   0, 208,   0
Entry 49 :   0, 208,  48
Entry 50 :   0, 208,  80
Entry 51 :   0, 208, 112
Entry 52 :   0, 208, 144
Entry 53 :   0, 208, 176
Entry 54 :   0, 208, 208
Entry 55 :   0, 208, 240
Entry 56 :   0, 240,   0
Entry 57 :   0, 240,  48
Entry 58 :   0, 240,  80
Entry 59 :   0, 240, 112
Entry 60 :   0, 240, 144
Entry 61 :   0, 240, 176
Entry 62 :   0, 240, 208
Entry 63 :   0, 240, 240
Entry 64 :  80,   0,   0
Entry 65 :  80,   0,  48
Entry 66 :  80,   0,  80
Entry 67 :  80,   0, 112
Entry 68 :  80,   0, 144
Entry 69 :  80,   0, 176
Entry 70 :  80,   0, 208
Entry 71 :  80,   0, 240
Entry 72 :  80,  48,   0
Entry 73 :  80,  48,  48
Entry 74 :  80,  48,  80
Entry 75 :  80,  48, 112
Entry 76 :  80,  48, 144
Entry 77 :  80,  48, 176
Entry 78 :  80,  48, 208
Entry 79 :  80,  48, 240
Entry 80 :  80,  80,   0
Entry 81 :  80,  80,  48
Entry 82 :  80,  80,  80
Entry 83 :  80,  80, 112
Entry 84 :  80,  80, 144
Entry 85 :  80,  80, 176
Entry 86 :  80,  80, 208
Entry 87 :  80,  80, 240
Entry 88 :  80, 112,   0
Entry 89 :  80, 112,  48
Entry 90 :  80, 112,  80
Entry 91 :  80, 112, 112
Entry 92 :  80, 112, 144
```

```
Entry 93 :  80, 112, 176
Entry 94 :  80, 112, 208
Entry 95 :  80, 112, 240
Entry 96 :  80, 144,   0
Entry 97 :  80, 144,  48
Entry 98 :  80, 144,  80
Entry 99 :  80, 144, 112
Entry 100 :  80, 144, 144
Entry 101 :  80, 144, 176
Entry 102 :  80, 144, 208
Entry 103 :  80, 144, 240
Entry 104 :  80, 176,   0
Entry 105 :  80, 176,  48
Entry 106 :  80, 176,  80
Entry 107 :  80, 176, 112
Entry 108 :  80, 176, 144
Entry 109 :  80, 176, 176
Entry 110 :  80, 176, 208
Entry 111 :  80, 176, 240
Entry 112 :  80, 208,   0
Entry 113 :  80, 208,  48
Entry 114 :  80, 208,  80
Entry 115 :  80, 208, 112
Entry 116 :  80, 208, 144
Entry 117 :  80, 208, 176
Entry 118 :  80, 208, 208
Entry 119 :  80, 208, 240
Entry 120 :  80, 240,   0
Entry 121 :  80, 240,  48
Entry 122 :  80, 240,  80
Entry 123 :  80, 240, 112
Entry 124 :  80, 240, 144
Entry 125 :  80, 240, 176
Entry 126 :  80, 240, 208
Entry 127 :  80, 240, 240
Entry 128 : 160,   0,   0
Entry 129 : 160,   0,  48
Entry 130 : 160,   0,  80
Entry 131 : 160,   0, 112
Entry 132 : 160,   0, 144
Entry 133 : 160,   0, 176
Entry 134 : 160,   0, 208
Entry 135 : 160,   0, 240
Entry 136 : 160,  48,   0
Entry 137 : 160,  48,  48
Entry 138 : 160,  48,  80
Entry 139 : 160,  48, 112
Entry 140 : 160,  48, 144
Entry 141 : 160,  48, 176
Entry 142 : 160,  48, 208
Entry 143 : 160,  48, 240
Entry 144 : 160,  80,   0
Entry 145 : 160,  80,  48
Entry 146 : 160,  80,  80
Entry 147 : 160,  80, 112
Entry 148 : 160,  80, 144
Entry 149 : 160,  80, 176
Entry 150 : 160,  80, 208
Entry 151 : 160,  80, 240
Entry 152 : 160, 112,   0
Entry 153 : 160, 112,  48
Entry 154 : 160, 112,  80
Entry 155 : 160, 112, 112
Entry 156 : 160, 112, 144
Entry 157 : 160, 112, 176
Entry 158 : 160, 112, 208
Entry 159 : 160, 112, 240
Entry 160 : 160, 144,   0
Entry 161 : 160, 144,  48
Entry 162 : 160, 144,  80
Entry 163 : 160, 144, 112
Entry 164 : 160, 144, 144
Entry 165 : 160, 144, 176
Entry 166 : 160, 144, 208
Entry 167 : 160, 144, 240
Entry 168 : 160, 176,   0
Entry 169 : 160, 176,  48
Entry 170 : 160, 176,  80
Entry 171 : 160, 176, 112
Entry 172 : 160, 176, 144
Entry 173 : 160, 176, 176
Entry 174 : 160, 176, 208
Entry 175 : 160, 176, 240
Entry 176 : 160, 208,   0
Entry 177 : 160, 208,  48
Entry 178 : 160, 208,  80
Entry 179 : 160, 208, 112
Entry 180 : 160, 208, 144
Entry 181 : 160, 208, 176
Entry 182 : 160, 208, 208
Entry 183 : 160, 208, 240
Entry 184 : 160, 240,   0
Entry 185 : 160, 240,  48
Entry 186 : 160, 240,  80
Entry 187 : 160, 240, 112
Entry 188 : 160, 240, 144
Entry 189 : 160, 240, 176
Entry 190 : 160, 240, 208
Entry 191 : 160, 240, 224
Entry 192 : 240,   0,   0
Entry 193 : 240,   0,  48
Entry 194 : 240,   0,  80
Entry 195 : 240,   0, 112
Entry 196 : 240,   0, 144
Entry 197 : 240,   0, 176
Entry 198 : 240,   0, 208
Entry 199 : 240,   0, 240
Entry 200 : 240,  48,   0
Entry 201 : 240,  48,  48
Entry 202 : 240,  48,  80
Entry 203 : 240,  48, 112
Entry 204 : 240,  48, 144
```

```
Entry 205 : 240,  48, 176
Entry 206 : 240,  48, 208
Entry 207 : 240,  48, 240
Entry 208 : 240,  80,   0
Entry 209 : 240,  80,  48
Entry 210 : 240,  80,  80
Entry 211 : 240,  80, 112
Entry 212 : 240,  80, 144
Entry 213 : 240,  80, 176
Entry 214 : 240,  80, 208
Entry 215 : 240,  80, 240
Entry 216 : 240, 112,   0
Entry 217 : 240, 112,  48
Entry 218 : 240, 112,  80
Entry 219 : 240, 112, 112
Entry 220 : 240, 112, 144
Entry 221 : 240, 112, 176
Entry 222 : 240, 112, 208
Entry 223 : 240, 112, 240
Entry 224 : 240, 144,   0
Entry 225 : 240, 144,  48
Entry 226 : 240, 144,  80
Entry 227 : 240, 144, 112
Entry 228 : 240, 144, 144
Entry 229 : 240, 144, 176
Entry 230 : 240, 144, 208
Entry 231 : 240, 144, 240
Entry 232 : 240, 176,   0
Entry 233 : 240, 176,  48
Entry 234 : 240, 176,  80
Entry 235 : 240, 176, 112
Entry 236 : 240, 176, 144
Entry 237 : 240, 176, 176
Entry 238 : 240, 176, 208
Entry 239 : 240, 176, 240
Entry 240 : 240, 208,   0
Entry 241 : 240, 208,  48
Entry 242 : 240, 208,  80
Entry 243 : 240, 208, 112
Entry 244 : 240, 208, 144
Entry 245 : 240, 208, 176
Entry 246 : 240, 208, 208
Entry 247 : 240, 208, 240
Entry 248 : 240, 240,   0
Entry 249 : 240, 240,  48
Entry 250 : 240, 240,  80
Entry 251 : 240, 240, 112
Entry 252 : 240, 240, 144
Entry 253 : 240, 240, 176
Entry 254 : 240, 240, 208
Entry 255 : 240, 240, 240


 State 3 : red, green, blue intensity

Entry 0 :   0,   0,   0
Entry 1 :   0,   0,  48
Entry 2 :   0,   0,  80
Entry 3 :   0,   0, 112
Entry 4 :   0,   0, 144
Entry 5 :   0,   0, 176
Entry 6 :   0,   0, 208
Entry 7 :   0,   0, 240
Entry 8 :   0,  80,   0
Entry 9 :   0,  80,  48
Entry 10 :   0,  80,  80
Entry 11 :   0,  80, 112
Entry 12 :   0,  80, 144
Entry 13 :   0,  80, 176
Entry 14 :   0,  80, 208
Entry 15 :   0,  80, 240
Entry 16 :   0, 160,   0
Entry 17 :   0, 160,  48
Entry 18 :   0, 160,  80
Entry 19 :   0, 160, 112
Entry 20 :   0, 160, 144
Entry 21 :   0, 160, 176
Entry 22 :   0, 160, 208
Entry 23 :   0, 160, 240
Entry 24 :   0, 240,   0
Entry 25 :   0, 240,  48
Entry 26 :   0, 240,  80
Entry 27 :   0, 240, 112
Entry 28 :   0, 240, 144
Entry 29 :   0, 240, 176
Entry 30 :   0, 240, 208
Entry 31 :   0, 240, 240
Entry 32 :  48,   0,   0
Entry 33 :  48,   0,  48
Entry 34 :  48,   0,  80
Entry 35 :  48,   0, 112
Entry 36 :  48,   0, 144
Entry 37 :  48,   0, 176
Entry 38 :  48,   0, 208
Entry 39 :  48,   0, 240
Entry 40 :  48,  80,   0
Entry 41 :  48,  80,  48
Entry 42 :  48,  80,  80
Entry 43 :  48,  80, 112
Entry 44 :  48,  80, 144
Entry 45 :  48,  80, 176
Entry 46 :  48,  80, 208
Entry 47 :  48,  80, 240
Entry 48 :  48, 160,   0
Entry 49 :  48, 160,  48
Entry 50 :  48, 160,  80
Entry 51 :  48, 160, 112
Entry 52 :  48, 160, 144
Entry 53 :  48, 160, 176
Entry 54 :  48, 160, 208
Entry 55 :  48, 160, 240
Entry 56 :  48, 240,   0
Entry 57 :  48, 240,  48
Entry 58 :  48, 240,  80
Entry 59 :  48, 240, 112
Entry 60 :  48, 240, 144
Entry 61 :  48, 240, 176
Entry 62 :  48, 240, 208
Entry 63 :  48, 240, 240
Entry 64 :  80,   0,   0
Entry 65 :  80,   0,  48
Entry 66 :  80,   0,  80
Entry 67 :  80,   0, 112
Entry 68 :  80,   0, 144
Entry 69 :  80,   0, 176
Entry 70 :  80,   0, 208
Entry 71 :  80,   0, 240
Entry 72 :  80,  80,   0
Entry 73 :  80,  80,  48
Entry 74 :  80,  80,  80
Entry 75 :  80,  80, 112
Entry 76 :  80,  80, 144
Entry 77 :  80,  80, 176
Entry 78 :  80,  80, 208
Entry 79 :  80,  80, 240
Entry 80 :  80, 160,   0
Entry 81 :  80, 160,  48
Entry 82 :  80, 160,  80
Entry 83 :  80, 160, 112
Entry 84 :  80, 160, 144
Entry 85 :  80, 160, 176
Entry 86 :  80, 160, 208
Entry 87 :  80, 160, 240
Entry 88 :  80, 240,   0
Entry 89 :  80, 240,  48
Entry 90 :  80, 240,  80
Entry 91 :  80, 240, 112
Entry 92 :  80, 240, 144
Entry 93 :  80, 240, 176
Entry 94 :  80, 240, 208
Entry 95 :  80, 240, 240
Entry 96 : 112,   0,   0
Entry 97 : 112,   0,  48
Entry 98 : 112,   0,  80
Entry 99 : 112,   0, 112
Entry 100 : 112,   0, 144
Entry 101 : 112,   0, 176
Entry 102 : 112,   0, 208
Entry 103 : 112,   0, 240
Entry 104 : 112,  80,   0
Entry 105 : 112,  80,  48
Entry 106 : 112,  80,  80
Entry 107 : 112,  80, 112
Entry 108 : 112,  80, 144
Entry 109 : 112,  80, 176
Entry 110 : 112,  80, 208
Entry 111 : 112,  80, 240
Entry 112 : 112, 160,   0
Entry 113 : 112, 160,  48
Entry 114 : 112, 160,  80
Entry 115 : 112, 160, 112
Entry 116 : 112, 160, 144
Entry 117 : 112, 160, 176
Entry 118 : 112, 160, 208
Entry 119 : 112, 160, 240
Entry 120 : 112, 240,   0
Entry 121 : 112, 240,  48
Entry 122 : 112, 240,  80
Entry 123 : 112, 240, 112
Entry 124 : 112, 240, 144
Entry 125 : 112, 240, 176
Entry 126 : 112, 240, 208
Entry 127 : 112, 240, 240
Entry 128 : 144,   0,   0
Entry 129 : 144,   0,  48
Entry 130 : 144,   0,  80
Entry 131 : 144,   0, 112
Entry 132 : 144,   0, 144
Entry 133 : 144,   0, 176
Entry 134 : 144,   0, 208
Entry 135 : 144,   0, 240
Entry 136 : 144,  80,   0
Entry 137 : 144,  80,  48
Entry 138 : 144,  80,  80
Entry 139 : 144,  80, 112
Entry 140 : 144,  80, 144
Entry 141 : 144,  80, 176
Entry 142 : 144,  80, 208
Entry 143 : 144,  80, 240
Entry 144 : 144, 160,   0
Entry 145 : 144, 160,  48
Entry 146 : 144, 160,  80
Entry 147 : 144, 160, 112
Entry 148 : 144, 160, 144
Entry 149 : 144, 160, 176
Entry 150 : 144, 160, 208
Entry 151 : 144, 160, 240
Entry 152 : 144, 240,   0
Entry 153 : 144, 240,  48
Entry 154 : 144, 240,  80
Entry 155 : 144, 240, 112
Entry 156 : 144, 240, 144
Entry 157 : 144, 240, 176
Entry 158 : 144, 240, 208
Entry 159 : 144, 240, 240
Entry 160 : 176,   0,   0
Entry 161 : 176,   0,  48
Entry 162 : 176,   0,  80
Entry 163 : 176,   0, 112
Entry 164 : 176,   0, 144
Entry 165 : 176,   0, 176
Entry 166 : 176,   0, 208
Entry 167 : 176,   0, 240
Entry 168 : 176,  80,   0
```

```
Entry 169 : 176,  80,  48
Entry 170 : 176,  80,  80
Entry 171 : 176,  80, 112
Entry 172 : 176,  80, 144
Entry 173 : 176,  80, 176
Entry 174 : 176,  80, 208
Entry 175 : 176,  80, 240
Entry 176 : 176, 160,   0
Entry 177 : 176, 160,  48
Entry 178 : 176, 160,  80
Entry 179 : 176, 160, 112
Entry 180 : 176, 160, 144
Entry 181 : 176, 160, 176
Entry 182 : 176, 160, 208
Entry 183 : 176, 160, 240
Entry 184 : 176, 240,   0
Entry 185 : 176, 240,  48
Entry 186 : 176, 240,  80
Entry 187 : 176, 240, 112
Entry 188 : 176, 240, 144
Entry 189 : 176, 240, 176
Entry 190 : 176, 240, 208
Entry 191 : 176, 240, 240
Entry 192 : 208,   0,   0
Entry 193 : 208,   0,  48
Entry 194 : 208,   0,  80
Entry 195 : 208,   0, 112
Entry 196 : 208,   0, 144
Entry 197 : 208,   0, 176
Entry 198 : 208,   0, 208
Entry 199 : 208,   0, 240
Entry 200 : 208,  80,   0
Entry 201 : 208,  80,  48
Entry 202 : 208,  80,  80
Entry 203 : 208,  80, 112
Entry 204 : 208,  80, 144
Entry 205 : 208,  80, 176
Entry 206 : 208,  80, 208
Entry 207 : 208,  80, 240
Entry 208 : 208, 160,   0
Entry 209 : 208, 160,  48
Entry 210 : 208, 160,  80
Entry 211 : 208, 160, 112
Entry 212 : 208, 160, 144
Entry 213 : 208, 160, 176
Entry 214 : 208, 160, 208
Entry 215 : 208, 160, 240
Entry 216 : 208, 240,   0
Entry 217 : 208, 240,  48
Entry 218 : 208, 240,  80
Entry 219 : 208, 240, 112
Entry 220 : 208, 240, 144
Entry 221 : 208, 240, 176
Entry 222 : 208, 240, 208
Entry 223 : 208, 240, 240
Entry 224 : 240,   0,   0
Entry 225 : 240,   0,  48
Entry 226 : 240,   0,  80
Entry 227 : 240,   0, 112
Entry 228 : 240,   0, 144
Entry 229 : 240,   0, 176
Entry 230 : 240,   0, 208
Entry 231 : 240,   0, 240
Entry 232 : 240,  80,   0
Entry 233 : 240,  80,  48
Entry 234 : 240,  80,  80
Entry 235 : 240,  80, 112
Entry 236 : 240,  80, 144
Entry 237 : 240,  80, 176
Entry 238 : 240,  80, 208
Entry 239 : 240,  80, 240
Entry 240 : 240, 160,   0
Entry 241 : 240, 160,  48
Entry 242 : 240, 160,  80
Entry 243 : 240, 160, 112
Entry 244 : 240, 160, 144
Entry 245 : 240, 160, 176
Entry 246 : 240, 160, 208
Entry 247 : 240, 160, 240
Entry 248 : 240, 240,   0
Entry 249 : 240, 240,  48
Entry 250 : 240, 240,  80
Entry 251 : 240, 240, 112
Entry 252 : 240, 240, 144
Entry 253 : 240, 240, 176
Entry 254 : 240, 240, 208
Entry 255 : 240, 240, 240


 State 4 : red, green, blue intensity

Entry 0 :    0,   0,   0
Entry 1 :    0,   0,  80
Entry 2 :    0,   0, 160
Entry 3 :    0,   0, 240
Entry 4 :    0,  48,   0
Entry 5 :    0,  48,  80
Entry 6 :    0,  48, 160
Entry 7 :    0,  48, 240
Entry 8 :    0,  80,   0
Entry 9 :    0,  80,  80
Entry 10 :    0,  80, 160
Entry 11 :    0,  80, 240
Entry 12 :    0, 112,   0
Entry 13 :    0, 112,  80
Entry 14 :    0, 112, 160
Entry 15 :    0, 112, 240
Entry 16 :    0, 144,   0
Entry 17 :    0, 144,  80
Entry 18 :    0, 144, 160
Entry 19 :    0, 144, 240
Entry 20 :    0, 176,   0
```

```
Entry 21 :    0, 176,  80
Entry 22 :    0, 176, 160
Entry 23 :    0, 176, 240
Entry 24 :    0, 208,   0
Entry 25 :    0, 208,  80
Entry 26 :    0, 208, 160
Entry 27 :    0, 208, 240
Entry 28 :    0, 240,   0
Entry 29 :    0, 240,  80
Entry 30 :    0, 240, 160
Entry 31 :    0, 240, 240
Entry 32 :   48,   0,   0
Entry 33 :   48,   0,  80
Entry 34 :   48,   0, 160
Entry 35 :   48,   0, 240
Entry 36 :   48,  48,   0
Entry 37 :   48,  48,  80
Entry 38 :   48,  48, 160
Entry 39 :   48,  48, 240
Entry 40 :   48,  80,   0
Entry 41 :   48,  80,  80
Entry 42 :   48,  80, 160
Entry 43 :   48,  80, 240
Entry 44 :   48, 112,   0
Entry 45 :   48, 112,  80
Entry 46 :   48, 112, 160
Entry 47 :   48, 112, 240
Entry 48 :   48, 144,   0
Entry 49 :   48, 144,  80
Entry 50 :   48, 144, 160
Entry 51 :   48, 144, 240
Entry 52 :   48, 176,   0
Entry 53 :   48, 176,  80
Entry 54 :   48, 176, 160
Entry 55 :   48, 176, 240
Entry 56 :   48, 208,   0
Entry 57 :   48, 208,  80
Entry 58 :   48, 208, 160
Entry 59 :   48, 208, 240
Entry 60 :   48, 240,   0
Entry 61 :   48, 240,  80
Entry 62 :   48, 240, 160
Entry 63 :   48, 240, 240
Entry 64 :   80,   0,   0
Entry 65 :   80,   0,  80
Entry 66 :   80,   0, 160
Entry 67 :   80,   0, 240
Entry 68 :   80,  48,   0
Entry 69 :   80,  48,  80
Entry 70 :   80,  48, 160
Entry 71 :   80,  48, 240
Entry 72 :   80,  80,   0
Entry 73 :   80,  80,  80
Entry 74 :   80,  80, 160
Entry 75 :   80,  80, 240
Entry 76 :   80, 112,   0
Entry 77 :  80, 112,  80
Entry 78 :  80, 112, 160
Entry 79 :  80, 112, 240
Entry 80 :  80, 144,   0
Entry 81 :  80, 144,  80
Entry 82 :  80, 144, 160
Entry 83 :  80, 144, 240
Entry 84 :  80, 176,   0
Entry 85 :  80, 176,  80
Entry 86 :  80, 176, 160
Entry 87 :  80, 176, 240
Entry 88 :  80, 208,   0
Entry 89 :  80, 208,  80
Entry 90 :  80, 208, 160
Entry 91 :  80, 208, 240
Entry 92 :  80, 240,   0
Entry 93 :  80, 240,  80
Entry 94 :  80, 240, 160
Entry 95 :  80, 240, 240
Entry 96 : 112,   0,   0
Entry 97 : 112,   0,  80
Entry 98 : 112,   0, 160
Entry 99 : 112,   0, 240
Entry 100 : 112,  48,   0
Entry 101 : 112,  48,  80
Entry 102 : 112,  48, 160
Entry 103 : 112,  48, 240
Entry 104 : 112,  80,   0
Entry 105 : 112,  80,  80
Entry 106 : 112,  80, 160
Entry 107 : 112,  80, 240
Entry 108 : 112, 112,   0
Entry 109 : 112, 112,  80
Entry 110 : 112, 112, 160
Entry 111 : 112, 112, 240
Entry 112 : 112, 144,   0
Entry 113 : 112, 144,  80
Entry 114 : 112, 144, 160
Entry 115 : 112, 144, 240
Entry 116 : 112, 176,   0
Entry 117 : 112, 176,  80
Entry 118 : 112, 176, 160
Entry 119 : 112, 176, 240
Entry 120 : 112, 208,   0
Entry 121 : 112, 208,  80
Entry 122 : 112, 208, 160
Entry 123 : 112, 208, 240
Entry 124 : 112, 240,   0
Entry 125 : 112, 240,  80
Entry 126 : 112, 240, 160
Entry 127 : 112, 240, 240
Entry 128 : 144,   0,   0
Entry 129 : 144,   0,  80
Entry 130 : 144,   0, 160
Entry 131 : 144,   0, 240
Entry 132 : 144,  48,   0
```

```
Entry 133 : 144,  48,  80
Entry 134 : 144,  48, 160
Entry 135 : 144,  48, 240
Entry 136 : 144,  80,   0
Entry 137 : 144,  80,  80
Entry 138 : 144,  80, 160
Entry 139 : 144,  80, 240
Entry 140 : 144, 112,   0
Entry 141 : 144, 112,  80
Entry 142 : 144, 112, 160
Entry 143 : 144, 112, 240
Entry 144 : 144, 144,   0
Entry 145 : 144, 144,  80
Entry 146 : 144, 144, 160
Entry 147 : 144, 144, 240
Entry 148 : 144, 176,   0
Entry 149 : 144, 176,  80
Entry 150 : 144, 176, 160
Entry 151 : 144, 176, 240
Entry 152 : 144, 208,   0
Entry 153 : 144, 208,  80
Entry 154 : 144, 208, 160
Entry 155 : 144, 208, 240
Entry 156 : 144, 240,   0
Entry 157 : 144, 240,  80
Entry 158 : 144, 240, 160
Entry 159 : 144, 240, 240
Entry 160 : 176,   0,   0
Entry 161 : 176,   0,  80
Entry 162 : 176,   0, 160
Entry 163 : 176,   0, 240
Entry 164 : 176,  48,   0
Entry 165 : 176,  48,  80
Entry 166 : 176,  48, 160
Entry 167 : 176,  48, 240
Entry 168 : 176,  80,   0
Entry 169 : 176,  80,  80
Entry 170 : 176,  80, 160
Entry 171 : 176,  80, 240
Entry 172 : 176, 112,   0
Entry 173 : 176, 112,  80
Entry 174 : 176, 112, 160
Entry 175 : 176, 112, 240
Entry 176 : 176, 144,   0
Entry 177 : 176, 144,  80
Entry 178 : 176, 144, 160
Entry 179 : 176, 144, 240
Entry 180 : 176, 176,   0
Entry 181 : 176, 176,  80
Entry 182 : 176, 176, 160
Entry 183 : 176, 176, 240
Entry 184 : 176, 208,   0
Entry 185 : 176, 208,  80
Entry 186 : 176, 208, 160
Entry 187 : 176, 208, 240
Entry 188 : 176, 240,   0
Entry 189 : 176, 240,  80
Entry 190 : 176, 240, 160
Entry 191 : 176, 240, 240
Entry 192 : 208,   0,   0
Entry 193 : 208,   0,  80
Entry 194 : 208,   0, 160
Entry 195 : 208,   0, 240
Entry 196 : 208,  48,   0
Entry 197 : 208,  48,  80
Entry 198 : 208,  48, 160
Entry 199 : 208,  48, 240
Entry 200 : 208,  80,   0
Entry 201 : 208,  80,  80
Entry 202 : 208,  80, 160
Entry 203 : 208,  80, 240
Entry 204 : 208, 112,   0
Entry 205 : 208, 112,  80
Entry 206 : 208, 112, 160
Entry 207 : 208, 112, 240
Entry 208 : 208, 144,   0
Entry 209 : 208, 144,  80
Entry 210 : 208, 144, 160
Entry 211 : 208, 144, 240
Entry 212 : 208, 176,   0
Entry 213 : 208, 176,  80
Entry 214 : 208, 176, 160
Entry 215 : 208, 176, 240
Entry 216 : 208, 208,   0
Entry 217 : 208, 208,  80
Entry 218 : 208, 208, 160
Entry 219 : 208, 208, 240
Entry 220 : 208, 240,   0
Entry 221 : 208, 240,  80
Entry 222 : 208, 240, 160
Entry 223 : 208, 240, 240
Entry 224 : 240,   0,   0
Entry 225 : 240,   0,  80
Entry 226 : 240,   0, 160
Entry 227 : 240,   0, 240
Entry 228 : 240,  48,   0
Entry 229 : 240,  48,  80
Entry 230 : 240,  48, 160
Entry 231 : 240,  48, 240
Entry 232 : 240,  80,   0
Entry 233 : 240,  80,  80
Entry 234 : 240,  80, 160
Entry 235 : 240,  80, 240
Entry 236 : 240, 112,   0
Entry 237 : 240, 112,  80
Entry 238 : 240, 112, 160
Entry 239 : 240, 112, 240
Entry 240 : 240, 144,   0
Entry 241 : 240, 144,  80
Entry 242 : 240, 144, 160
Entry 243 : 240, 144, 240
Entry 244 : 240, 176,   0
```

```
Entry 245 : 240, 176,  80
Entry 246 : 240, 176, 160
Entry 247 : 240, 176, 240
Entry 248 : 240, 208,   0
Entry 249 : 240, 208,  80
Entry 250 : 240, 208, 160
Entry 251 : 240, 208, 240
Entry 252 : 240, 240,   0
Entry 253 : 240, 240,  80
Entry 254 : 240, 240, 160
Entry 255 : 240, 240, 240
```

```
 State 5 : red, green, blue intensity

Entry 0 :   0,   0,   0
Entry 1 :   0,   0,  48
Entry 2 :   0,   0,  96
Entry 3 :   0,   0, 144
Entry 4 :   0,   0, 192
Entry 5 :   0,   0, 240
Entry 6 :   0,  48,   0
Entry 7 :   0,  48,  48
Entry 8 :   0,  48,  96
Entry 9 :   0,  48, 144
Entry 10 :   0,  48, 192
Entry 11 :   0,  48, 240
Entry 12 :   0,  96,   0
Entry 13 :   0,  96,  48
Entry 14 :   0,  96,  96
Entry 15 :   0,  96, 144
Entry 16 :   0,  96, 192
Entry 17 :   0,  96, 240
Entry 18 :   0, 144,   0
Entry 19 :   0, 144,  48
Entry 20 :   0, 144,  96
Entry 21 :   0, 144, 144
Entry 22 :   0, 144, 192
Entry 23 :   0, 144, 240
Entry 24 :   0, 192,   0
Entry 25 :   0, 192,  48
Entry 26 :   0, 192,  96
Entry 27 :   0, 192, 144
Entry 28 :   0, 192, 192
Entry 29 :   0, 192, 240
Entry 30 :   0, 240,   0
Entry 31 :   0, 240,  48
Entry 32 :   0, 240,  96
Entry 33 :   0, 240, 144
Entry 34 :   0, 240, 192
Entry 35 :   0, 240, 240
Entry 36 :  48,   0,   0
Entry 37 :  48,   0,  48
Entry 38 :  48,   0,  96
Entry 39 :  48,   0, 144
Entry 40 :  48,   0, 192
Entry 41 :  48,   0, 240
Entry 42 :  48,  48,   0
Entry 43 :  48,  48,  48
Entry 44 :  48,  48,  96
Entry 45 :  48,  48, 144
Entry 46 :  48,  48, 192
Entry 47 :  48,  48, 240
Entry 48 :  48,  96,   0
Entry 49 :  48,  96,  48
Entry 50 :  48,  96,  96
Entry 51 :  48,  96, 144
Entry 52 :  48,  96, 192
Entry 53 :  48,  96, 240
Entry 54 :  48, 144,   0
Entry 55 :  48, 144,  48
Entry 56 :  48, 144,  96
Entry 57 :  48, 144, 144
Entry 58 :  48, 144, 192
Entry 59 :  48, 144, 240
Entry 60 :  48, 192,   0
Entry 61 :  48, 192,  48
Entry 62 :  48, 192,  96
Entry 63 :  48, 192, 144
Entry 64 :  48, 192, 192
Entry 65 :  48, 192, 240
Entry 66 :  48, 240,   0
Entry 67 :  48, 240,  48
Entry 68 :  48, 240,  96
Entry 69 :  48, 240, 144
Entry 70 :  48, 240, 192
Entry 71 :  48, 240, 240
Entry 72 :  96,   0,   0
Entry 73 :  96,   0,  48
Entry 74 :  96,   0,  96
Entry 75 :  96,   0, 144
Entry 76 :  96,   0, 192
Entry 77 :  96,   0, 240
Entry 78 :  96,  48,   0
Entry 79 :  96,  48,  48
Entry 80 :  96,  48,  96
Entry 81 :  96,  48, 144
Entry 82 :  96,  48, 192
Entry 83 :  96,  48, 240
Entry 84 :  96,  96,   0
Entry 85 :  96,  96,  48
Entry 86 :  96,  96,  96
Entry 87 :  96,  96, 144
Entry 88 :  96,  96, 192
Entry 89 :  96,  96, 240
Entry 90 :  96, 144,   0
Entry 91 :  96, 144,  48
Entry 92 :  96, 144,  96
Entry 93 :  96, 144, 144
Entry 94 :  96, 144, 192
Entry 95 :  96, 144, 240
Entry 96 :  96, 192,   0
```

```
Entry 97 :  96, 192,  48
Entry 98 :  96, 192,  96
Entry 99 :  96, 192, 144
Entry 100 :  96, 192, 192
Entry 101 :  96, 192, 240
Entry 102 :  96, 240,   0
Entry 103 :  96, 240,  48
Entry 104 :  96, 240,  96
Entry 105 :  96, 240, 144
Entry 106 :  96, 240, 192
Entry 107 :  96, 240, 240
Entry 108 : 144,   0,   0
Entry 109 : 144,   0,  48
Entry 110 : 144,   0,  96
Entry 111 : 144,   0, 144
Entry 112 : 144,   0, 192
Entry 113 : 144,   0, 240
Entry 114 : 144,  48,   0
Entry 115 : 144,  48,  48
Entry 116 : 144,  48,  96
Entry 117 : 144,  48, 144
Entry 118 : 144,  48, 192
Entry 119 : 144,  48, 240
Entry 120 : 144,  96,   0
Entry 121 : 144,  96,  48
Entry 122 : 144,  96,  96
Entry 123 : 144,  96, 144
Entry 124 : 144,  96, 192
Entry 125 : 144,  96, 240
Entry 126 : 144, 144,   0
Entry 127 : 144, 144,  48
Entry 128 : 144, 144,  96
Entry 129 : 144, 144, 144
Entry 130 : 144, 144, 192
Entry 131 : 144, 144, 240
Entry 132 : 144, 192,   0
Entry 133 : 144, 192,  48
Entry 134 : 144, 192,  96
Entry 135 : 144, 192, 144
Entry 136 : 144, 192, 192
Entry 137 : 144, 192, 240
Entry 138 : 144, 240,   0
Entry 139 : 144, 240,  48
Entry 140 : 144, 240,  96
Entry 141 : 144, 240, 144
Entry 142 : 144, 240, 192
Entry 143 : 144, 240, 240
Entry 144 : 192,   0,   0
Entry 145 : 192,   0,  48
Entry 146 : 192,   0,  96
Entry 147 : 192,   0, 144
Entry 148 : 192,   0, 192
Entry 149 : 192,   0, 240
Entry 150 : 192,  48,   0
Entry 151 : 192,  48,  48
Entry 152 : 192,  48,  96
Entry 153 : 192,  48, 144
Entry 154 : 192,  48, 192
Entry 155 : 192,  48, 240
Entry 156 : 192,  96,   0
Entry 157 : 192,  96,  48
Entry 158 : 192,  96,  96
Entry 159 : 192,  96, 144
Entry 160 : 192,  96, 192
Entry 161 : 192,  96, 240
Entry 162 : 192, 144,   0
Entry 163 : 192, 144,  48
Entry 164 : 192, 144,  96
Entry 165 : 192, 144, 144
Entry 166 : 192, 144, 192
Entry 167 : 192, 144, 240
Entry 168 : 192, 192,   0
Entry 169 : 192, 192,  48
Entry 170 : 192, 192,  96
Entry 171 : 192, 192, 144
Entry 172 : 192, 192, 192
Entry 173 : 192, 192, 240
Entry 174 : 192, 240,   0
Entry 175 : 192, 240,  48
Entry 176 : 192, 240,  96
Entry 177 : 192, 240, 144
Entry 178 : 192, 240, 192
Entry 179 : 192, 240, 240
Entry 180 : 240,   0,   0
Entry 181 : 240,   0,  48
Entry 182 : 240,   0,  96
Entry 183 : 240,   0, 144
Entry 184 : 240,   0, 192
Entry 185 : 240,   0, 240
Entry 186 : 240,  48,   0
Entry 187 : 240,  48,  48
Entry 188 : 240,  48,  96
Entry 189 : 240,  48, 144
Entry 190 : 240,  48, 192
Entry 191 : 240,  48, 240
Entry 192 : 240,  96,   0
Entry 193 : 240,  96,  48
Entry 194 : 240,  96,  96
Entry 195 : 240,  96, 144
Entry 196 : 240,  96, 192
Entry 197 : 240,  96, 240
Entry 198 : 240, 144,   0
Entry 199 : 240, 144,  48
Entry 200 : 240, 144,  96
Entry 201 : 240, 144, 144
Entry 202 : 240, 144, 192
Entry 203 : 240, 144, 240
Entry 204 : 240, 192,   0
Entry 205 : 240, 192,  48
Entry 206 : 240, 192,  96
Entry 207 : 240, 192, 144
Entry 208 : 240, 192, 192
```

```
Entry 209 : 240, 192, 240
Entry 210 : 240, 240,   0
Entry 211 : 240, 240,  48
Entry 212 : 240, 240,  96
Entry 213 : 240, 240, 144
Entry 214 : 240, 240, 192
Entry 215 : 240, 240, 240
Entry 216 :   0,   0,   0
Entry 217 :   0,   0,   0
Entry 218 :   0,   0,   0
Entry 219 :   0,   0,   0
Entry 220 :   0,   0,   0
Entry 221 :   0,   0,   0
Entry 222 :   0,   0,   0
Entry 223 :   0,   0,   0
Entry 224 :   0,   0,   0
Entry 225 :   0,   0,   0
Entry 226 :   0,   0,   0
Entry 227 :   0,   0,   0
Entry 228 :   0,   0,   0
Entry 229 :   0,   0,   0
Entry 230 :   0,   0,   0
Entry 231 :   0,   0,   0
Entry 232 :   0,   0,   0
Entry 233 :   0,   0,   0
Entry 234 :   0,   0,   0
Entry 235 :   0,   0,   0
Entry 236 :   0,   0,   0
Entry 237 :   0,   0,   0
Entry 238 :   0,   0,   0
Entry 239 :   0,   0,   0
Entry 240 :   0,   0,   0
Entry 241 :   0,   0,   0
Entry 242 :   0,   0,   0
Entry 243 :   0,   0,   0
Entry 244 :   0,   0,   0
Entry 245 :   0,   0,   0
Entry 246 :   0,   0,   0
Entry 247 :   0,   0,   0
Entry 248 :   0,   0,   0
Entry 249 :   0,   0,   0
Entry 250 :   0,   0,   0
Entry 251 :   0,   0,   0
Entry 252 :   0,   0,   0
Entry 253 :   0,   0,   0
Entry 254 :   0,   0,   0
Entry 255 :   0,   0,   0
```

Appendix D

Command List Sample

SECTION D.1 Command List

D.1 Command List

The following are the ASCII commands for the command list that draws the house illustrated in Figure 4.5 in Section 4.4.

```
CA
RF
CB 100
P3 4  -100,      0,    100
       100,      0,    100
       100,      0,   -100
      -100,      0,   -100
P3 4  -100,    100,    100
       100,    100,    100
       100,    100,   -100
      -100,    100,   -100
M3    -100,    100,    100
D3    -100,      0,    100
M3     100,    100,    100
D3     100,      0,    100
M3     100,    100,   -100
D3     100,      0,   -100
M3    -100,    100,   -100
D3    -100,      0,   -100
M3    -100,    100,    100
D3       0,    135,    100
D3     100,    100,    100
M3     100,    100,   -100
D3       0,    135,   -100
D3    -100,    100,   -100
M3       0,    135,   -100
D3       0,    135,    100
M3     -15,      0,    100
D3     -15,     66,    100
D3      15,     66,    100
D3      15,      0,    100
P3 4    -5,     33,    100
         5,     33,    100
         5,     55,    100
        -5,     55,    100
P3 4   -75,     33,    100
       -40,     33,    100
       -40,     66,    100
       -75,     66,    100
P3 4    40,     33,    100
```

```
          75,       33,     100
          75,       66,     100
          40,       66,     100
P3 4      40,       33,    -100
          75,       33,    -100
          75,       66,    -100
          40,       66,    -100
P3 4     -75,       33,    -100
         -40,       33,    -100
         -40,       66,    -100
         -75,       66,    -100
P3 4    -100,       33,     -70
        -100,       33,     -35
        -100,       66,     -35
        -100,       66,     -70
P3 4    -100,       33,      35
        -100,       33,      70
        -100,       66,      70
        -100,       66,      35
P3 4       8,       25,     100
          13,       25,     100
          13,       30,     100
           8,       30,     100
P3 4     100,        0,     100
         175,        0,     100
         175,        0,       0
         100,        0,       0
P3 4     100,       75,     100
         175,       75,     100
         175,       75,       0
         100,       75,       0
M3       100,       75,       0
D3       100,        0,       0
M3       175,       75,       0
D3       175,        0,       0
M3       175,       75,     100
D3       175,        0,     100
M3       105,        0,     100
D3       105,       66,     100
D3       170,       66,     100
D3       170,        0,     100
M3       105,       22,     100
D3       170,       22,     100
M3       105,       44,     100
D3       170,       44,     100
P3 4     120,       50,     100
```

COMMAND LIST SAMPLE

```
       130,     50,    100
       130,     60,    100
       120,     60,    100
P3 4   145,     50,    100
       155,     50,    100
       155,     60,    100
       145,     60,    100
CE
CLS 0
C 25
CR 100
```

Appendix E

Default Parameters

E.1 Default Parameters

The following table represents the default values after a cold reset of the various matrices, flags and patterns used in the QG–640.

Flag	Name	Default Value	Description
1	AREAPT	65535 16 times	Solid area
2	CLIPH	0	Disabled
3	CLIPY	0	Disabled
4	COLOR	255	
6	DISTAN	500	
7	DISTH	-30000	
8	DISTY	30000	
9	FILMSK	255	All planes used
10	LINFUN	0	Set mode
11	LINPAT	65535	Solid lines
12	MASK	255	All planes on
13	MDORG	(0, 0, 0)	
14	2D current point	(0, 0)	
15	3D current point	(0, 0)	
16	PRMFIL	0	Off
17	PROJCT	60	
18	TANGLE	0	Horizontal
19	TJUST	1,1	Left, bottom
20	TSIZE	8	8×12 cells
21	VWPORT	0,639*,0,479*	Entire screen
22	VWRPT	(0, 0, 0)	
23	WINDOW	-320,319,-240,239*	
24	Transformed 3D point	(0, 0, 0)	
25	none	none	Used in FLAGRD
26	XHAIR - current pt on screen	(320,240)*	
27	XHAIR - current pt in 2D	(0,0)	
28	Screen Current Pt	(320,240)*	
29	none	none	Used in FLAGRD
30	none	none	Used in FLAGRD
31	none	none	Used in FLAGRD
32	TSTYLE	0	'fat' text
33	TASPCT	1.5	
34	TCHROT	0	
35	none	none	Used in FLAGRD
36	VDISP	0	
37	PMASK	255**	All LUT bits enabled
38	none	none	Used in FLAGRD
39	Display Format	640*,480*,60**,0*	
41	COLMOD	1	Transparent
42	BCOLOR	0	Transparent

* These values are determined by straps on the QG–640 circuit board.

* These values are set only on reset and power up.

Appendix F

VMS Macro Code Example

SECTION F.1 VMS Macro Code Example

F.1 VMS Macro Code Example

The following program reads the Status Register of the QG–640.

```
.title readstatus
;
; Sample program to read the status register of the QG640/QG1280
;
$iodef
.psect data,noexe,wrt
;
; This is the name of the QG-640 on your system
;
qgname: .ascid /QGAO/
;
OUTPUT_LENGTH = 80
; bit definititions for the QG Status Register
EMPTY = 0
FULL = 1
HALF_EMPTY = 2
PORT_FULL = 3
ERROR = 4

hello: .ascid "!/Reading status register of !AS"
qg_empty: .ascid /  -- FIFO empty/
qg_full: .ascid /  -- FIFO full/
qg_half_empty: .ascid /  -- FIFO half empty/
qg_port_full: .ascid /  -- Port register full/
```

```
qg_error: .ascid /  -- Error flag set/
error_message: .ascid /An error occurred./
status_value: .ascid /Status register value: !XB/
port_contents: .ascid /Port contents: !XB '!AF'/
channel: .blkw 1
output_string: .long OUTPUT_LENGTH
.address output_buffer
output_buffer: .blkb OUTPUT_LENGTH
input_buffer: .blkb 1

;
; this is the format of the I/O status block as returned by the
; QG driver.
;
status_block: .blkw 1 ; status
.blkw 1 ; byte count
qgstat: .blkb 1 ; qgstatus register
.blkb 3 ; reserved
;
; Actual code starts here
;
.psect code,exe,nowrt
.entry readstatus,^m<>
;
; make the announcement
;
movl #OUTPUT_LENGTH,output_string
$fao_s hello, output_string, output_string, #qgname
pushab output_string
calls #1,g^lib$put_output
;
; assign a channel to the QG
;
$assign_s qgname,channel
blbs r0,1$
brw exit_error
1$:
;
;
;
```

```
; null read from the QG (this will load the status register into
; the IOSB (status_block) for us) Note: status_block for p1 here
; is used as a buffer but is not modified since the read length
; is 0 bytes.
;
$qiow_s ,channel,#IO$_READVBLK,status_block,,,status_block,#0
blbs r0,2$
brw exit_error

2$:
;
; show results
;
movl #OUTPUT_LENGTH,output_string
$fao_s status_value,output_string,output_string,qgstat
pushab output_string
calls #1,g^lib$put_output
;
; check the empty flag
;
bbc #EMPTY,qgstat,10$
pushab qg_empty
calls #1,g^lib$put_output
;
; check the FIFO full flag
;
10$:
bbc #FULL,qgstat,20$
pushab qg_full
calls #1,g^lib$put_output
;
; check the FIFO half empty flag
;
20$:
bbc #HALF_EMPTY,qgstat,30$
pushab qg_half_empty
calls #1,g^lib$put_output
```

```
;
; check the port full flag
;
30$:
bbc #PORT_FULL,qgstat,40$
pushab qg_port_full
calls #1,g^lib$put_output
;
; check the error flag
;
40$:
bbs #ERROR,qgstat,50$
pushab qg_error
calls #1,g^lib$put_output
;
; if the port is full, dump its contents one byte at a time until
; the status register says there are no more bytes (port empty)
;
50$:
bbc #PORT_FULL,qgstat,60$
$qiow_s ,channel,#IO$_READVBLK,status_block,,,input_buffer,#1
movl #OUTPUT_LENGTH,output_string
$fao_s port_contents,output_string,output_string,input_buffer,-
     #1,#input_buffer
pushab output_string
calls #1,g^lib$put_output
brb 50$
60$:
$dassgn_s channel
blbc r0,exit_error
ret

exit_error:
pushab error_message
calls #1,g^lib$put_output
ret

.end readstatus
```

```
$! testread.com
$!
$! use the readstatus program to test the driver IOSB status register
$! return
$!
$ run := RUN
$ wait := WAIT
$ copy := COPY
$!
$! show the starting state of the QG640
$!
$ run readstatus
$!
$! copy something to the QG640 that will keep the FIFO non-empty for 10 seconds
$!
$ copy sys$input qga0:
wait 600
move 0 0
flagrd 14
$ run readstatus
$!
$! wait 10 seconds and then show the status register again
$!
$ wait 00:00:10.00
$ run readstatus
```

```
;
;
; Reading status register of QGA0
;
Status register value: F9
  -- FIFO empty
  -- Port register full
Port contents: 20 ' '
Reading status register of QGA0
Status register value: F0
Reading status register of QGA0
Status register value: F9
  -- FIFO empty
  -- Port register full
Port contents: 30 '0'
Port contents: 2C ','
Port contents: 30 '0'
Port contents: 0D '.'
```

Appendix G

Fast Execution "Local Pipes"

This chapter describes the fast execution families of graphic commands, optimized to work together as a group, in the firmware for the QG–640. These families of graphic commands use local command decoders to offer greatly increased command decoding speed. Section G.1 explains the concept of "local pipes" and Section G.2 describes the "local pipe" Command Sets.

G.1 Description of Local Pipes

The QG–640 contains fast execution "local pipes" in its firmware. The term "pipe" is used here to describe a subset of the board's full set of graphic commands which has been optimized to work as a group. Special areas of the firmware contain local command decoders which bypass the normal, lengthy highlevel decoding overhead. These local command decoders are, therefore, capable of decoding a small, fixed number of commands very quickly. If only graphic commands which are part of the local pipe's command set are issued to the board, decoding stays within the pipe and executes much faster than would normally be possible.

Entry to a local pipe is automatically achieved by sending the QG–640 one of a local pipe's *Entry Point Commands*. As soon as a command outside of the local pipe's command set is issued to the board, the local pipe is exited and decoding of commands through the highlevel command decoder resumes.

NOTE:

— Local pipes are accessed through *Entry Point Commands* only.

— Commands in a local pipe's command set are not all *Entry Point Commands*.

— Certain local pipes are not accessible from command lists.

— No local pipes are accessible from ASCII input mode.

G.2 Local Pipe Command Set Descriptions

Screen Coordinate Drawing Command Pipe

Command Set:

SMOVE *En.*

SMOVER *En.*

SDRAW *En.*

SDRAWR *En.*

COLOR

En. denotes an *Entry Point Command* in the Pipe Command Set.

Access from:

Hex Input Mode

Command Lists

User Definable Raster Text Command Pipe

Command Set:

TEXTP $^{En.}$

TEXTPC $^{En.}$

SMOVE †

SMOVER †

COLOR

BCOLOR

RFONT

$^{En.}$ denotes an ***Entry Point Command*** in the Pipe Command Set.

† denotes an ***Entry Point Command*** for the Screen Coordinate Drawing Command Pipe.

Access from:

Hex Input Mode **only**

NOTE:

The User Definable Raster Text Command Pipe is a two-level local pipe in which two of the commands in the command set, SMOVE and SMOVER, are also part of the Screen Coordinate Drawing Command Pipe. The following shows the process flow when either one of these commands is invoked. Note the two-level pipelining in Example 1.

COMMAND	COURSE OF ACTION
	Example 1
TEXTP	Enters User Definable Raster Text Command Pipe.
SMOVE	Enters Screen Coordinate Drawing Command Pipe.
SDRAW	Remains in Screen Coordinate Drawing Command Pipe.
TEXTP	Exits back to User Definable Raster Text Command Pipe.
	Example 2
TEXTP	Enters User Definable Raster Text Command Pipe.
SMOVE	Enters Screen Coordinate Drawing Command Pipe.
TEXTP	Exits back to User Definable Raster Text Command Pipe.

Any number of the commands from the Screen Coordinate Drawing Command Pipe command set may be used **directly** following the SMOVE or SMOVER commands. Once the flow has exited the Screen Coordinate Drawing Command Pipe, invoking any of the Screen commands will cause the program to exit the User Definable Raster Text Command Pipe and return to highlevel command decoding.

FAST EXECUTION "LOCAL PIPES"

World Coordinate 2D Drawing Command Pipe

Command Set:

MOVE $^{En.}$

MOVER $^{En.}$

DRAW $^{En.}$

DRAWR $^{En.}$

COLOR

$^{En.}$ denotes an ***Entry Point Command*** in the Pipe Command Set.

Access from:

Hex Input Mode

Command Lists

World Coordinate 3D Drawing Command Pipe

Command Set:

MOVE3 $^{En.}$

MOVER3 $^{En.}$

DRAW3 $^{En.}$

DRAWR3 $^{En.}$

$^{En.}$ denotes an ***Entry Point Command*** in the Pipe Command Set.

Access from:

Hex Input Mode

Command Lists

IMAGEW Command Pipe

Command Set:

IMAGEW *En.*

En. denotes an *Entry Point Command* in the Pipe Command Set.

Access from:

Hex Input Mode **only**

PDRAW Command Pipe

Command Set:

PDRAW *En.*

COLOR

NOP

En. denotes an *Entry Point Command* in the Pipe Command Set.

Access from:

Hex Input Mode

Command Lists

Appendix H

Warranty

H.1 Warranty

Matrox products are warranted against defects in materials and workmanship for a period of 180 days from date of delivery. We will repair or replace products which prove to be defective during the warranty period provided they are returned, at the user's expense, to Matrox Electronic Systems Limited. No other warranty is expressed or implied. We are not liable for consequential damages.

If you experience any difficulties with your Matrox product, please contact the Matrox representative where you purchased the product for service. Do not return any product to Matrox without authorization.

If, for some reason, you must return your product directly to Matrox, please follow these steps:

1. Contact the Matrox Customer Support Group.

   - U.S. customers call 1-800-4MATROX.
   - Canadian and international customers call (514) 685-2630.

   The Customer Support Group will issue a Return Merchandise Authorization (RMA) number.

2. Complete the Product Maintenance Report at the back of this manual. Write the RMA number in the space provided.

3. Do not change the hardware configuration. Leave all straps as you were using them.

4. Pack the product in its original box and return it with the completed Product Maintenance Report.

U. S. customers must return their products to our U. S. warehouse:

Matrox International Corp.
Trimex Building
Mooers, N.Y.
12958

Canadian and other international customers may return their products directly to our Canadian facility:

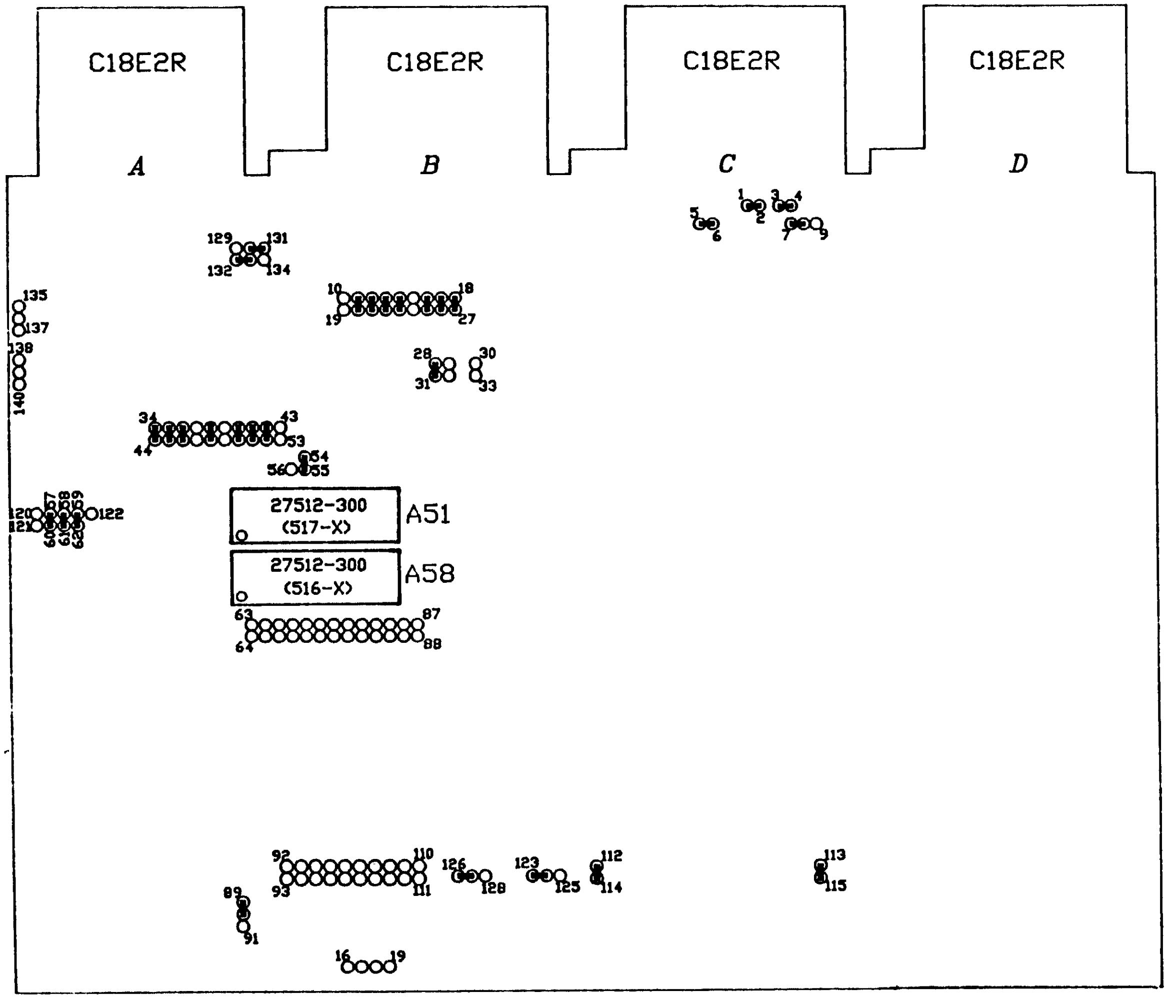
Matrox Electronic Systems Ltd.
1055 St. Régis Blvd.
Dorval, Québec, Canada
H9P 2T4

Appendix I

As-Shipped Straps

The following pages provide the QG-640 as-shipped strap locations. These circuit board layout drawings will assist you when modifying the configuration setting of your QG-640 board. Refer to Section A.2 for the configuration procedures.

AS-SHIPPED STRAPS



"STRAP FUNCTIONS" :

1-2 : INTERRUPT GRANT CONTINUITY

3-4 : DMA GRANT CONTINUITY

5-6 : MUST BE IN

7-8 : NON-INTERLACE MODE

10-19 : MUST BE OUT

11 ● 18 : BASE ADRESS (160400_8)
20 ● 27

28-31 : MUST BE IN

34 ● 53 : INTERRUPT VECTOR (504_8)

54-55 : NON-INTERLACE MODE

57-60 : INTERRUPT MODE (DISTRIBUTED ARBITRATION)

58-61 : INTERRUPT LEVEL (LEVEL 4)
59-62

63 ● 88 : MUST BE OUT

89-90 : MUST BE IN

92 ● 111 : MUST BE OUT

112-114 : COMPOSITE SYNC. ON GREEN CHANNEL

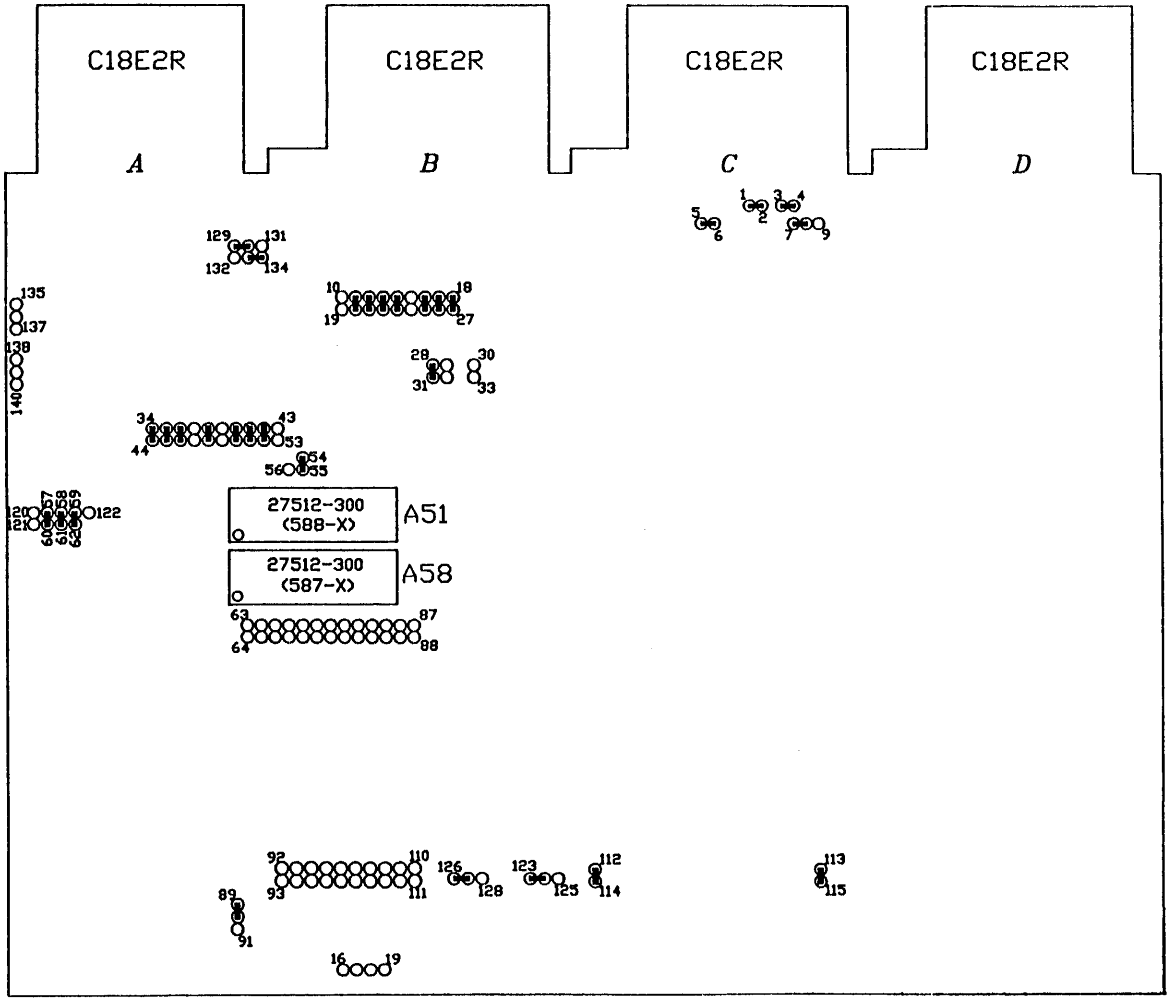
113-115 : COMPOSITE SYNC. ON GREEN CHANNEL

116 ● 121 : MUST BE OUT

29-32 : NON-INTERLACE MODE (OUT)

30-33 : 640x480 FORMAT (OUT)

QG-640



"STRAP FUNCTIONS" :

1-2 : INTERRUPT GRANT CONTINUITY

3-4 : DMA GRANT CONTINUITY

5-6 : MUST BE IN

7-8 : NON-INTERLACE MODE

10-19 : MUST BE OUT

11 ● 18 : BASE ADRESS (160400_8)
20 ● 27

28-31 : MUST BE IN

34 ● 53 : INTERRUPT VECTOR (504_8)

54-55 : NON-INTERLACE MODE

57-60 : INTERRUPT MODE (DISTRIBUTED ARBITRATION)

58-61 : INTERRUPT LEVEL (LEVEL 4)
59-62

63 ● 88 : MUST BE OUT

89-90 : MUST BE IN

92 ● 111 : MUST BE OUT

112-114 : COMPOSITE SYNC. ON GREEN CHANNEL

113-115 : COMPOSITE SYNC. ON GREEN CHANNEL

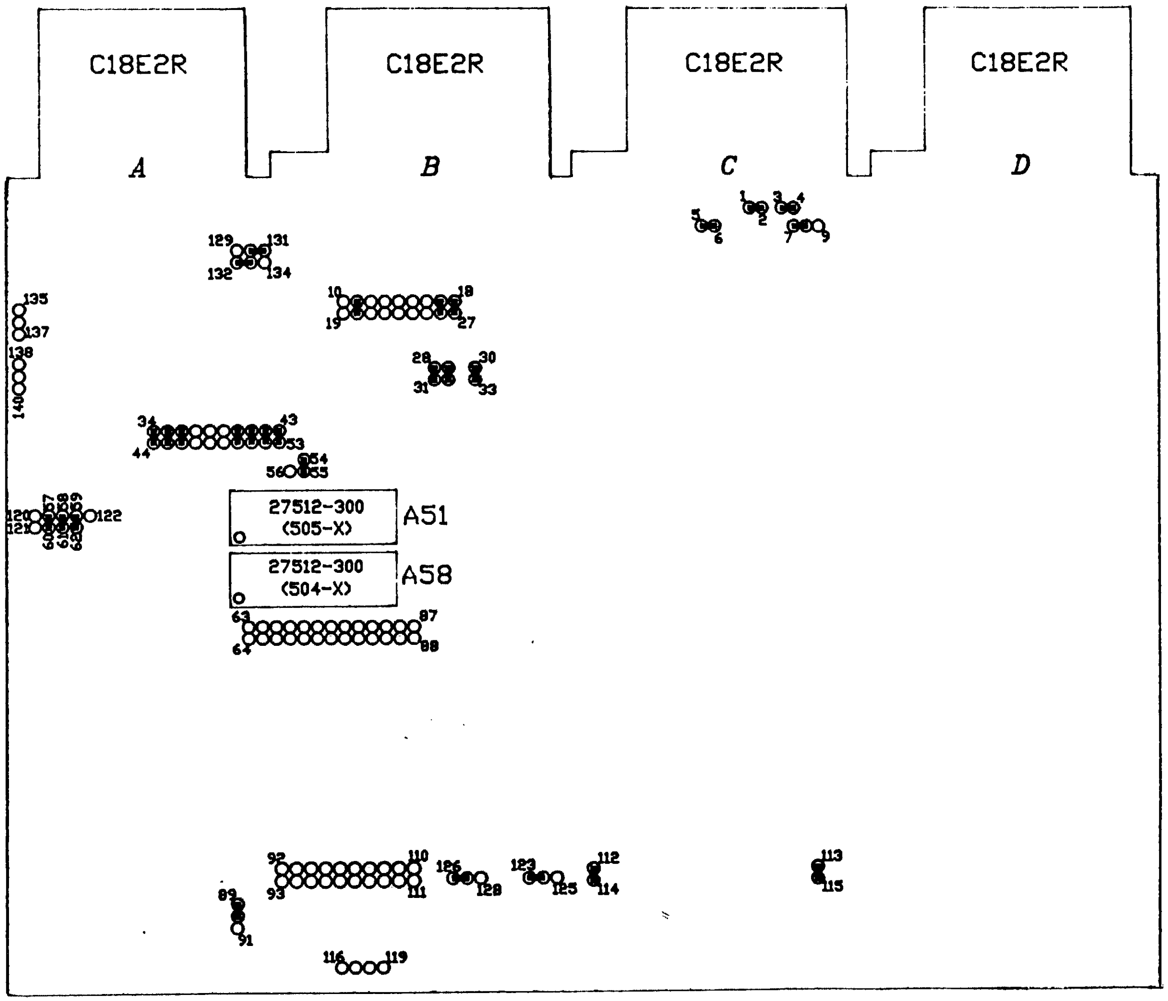
116 ● 121 : MUST BE OUT

29-32 : NON-INTERLACE MODE (OUT)

30-33 : 640x480 FORMAT (OUT)

QG-640/F REV.A

AS-SHIPPED STRAPS



"STRAP FUNCTIONS" :

1-2 : INTERRUPT GRANT CONTINUITY

3-4 : DMA GRANT CONTINUITY

5-6 : MUST BE IN

7-8 : NON-INTERLACE MODE

10-19 : MUST BE OUT

11 ● 18 : BASE ADDRESS (167 600_8)
20 ● 27

28-31 : MUST BE IN

34 ● 53 : INTERRUPT VECTOR (700_8)

54-55 : NON-INTERLACE MODE

57-60 : INTERRUPT MODE (DISTRIBUTED ARBITRATION)

58-61 : INTERRUPT LEVEL (LEVEL 4)
59-62

63 ● 88 : MUST BE OUT

89-90 : MUST BE IN

92 ● 111 : MUST BE OUT

112-114 : COMPOSITE SYNC. ON GREEN CHANNEL

113-115 : COMPOSITE SYNC. ON GREEN CHANNEL

116 ● 121 : MUST BE OUT

29-32 : EMC MODE

30-33 : 512x512 FORMAT

QG-640/EMC

J.1 Commands by Name

Name	Opcode
ARC	3C
AREA	C0
AREABC	C1
AREAPT	E7
BCOLOR	CB
BLINK	C8
BLINKX	E5
CA	43 41 20
CIRCLE	38
CLBEG	70
CLDEL	74
CLEARS	0F
CLEND	71
CLIPH	AA
CLIPY	AB
CLOOP	73
CLMOD	78
CLRD	75
CLRUN	72
COLMOD	CA
COLOR	06
CONVERT	AF
CX	43 58 20
DISTAN	B1
DISTH	A8
DISTY	A9
DRAW	28
DRAWR	29
DRAW3	2A
DRAWR3	2B
ELIPSE	39
ERROR	60
EXPAND	B4
FILMSK	EF
FLAGRD	51
FLOOD	07
GTDEF	89
IMAGER	D8
IMAGEW	D9
LINFUN	EB
LINPAT	EA
LUT	EE
LUTINT	EC
LUTRD	50
LUTSAV	ED
LUTSTO	C9
LUTX	E6
LUTXRD	53
MASK	E8
MATXRD	52
MDIDEN	90
MDMATX	97
MDORG	91
MDROTX	93
MDROTY	94
MDROTZ	95
MDSCAL	92
MDTRAN	96
MOVE	10
MOVER	11
MOVE3	12
MOVER3	13
NOOP	01
PDRAW	FF
POINT	08
POINT3	09
POLY	30
POLYR	31
POLY3	32
POLYR3	33
PMASK	D6
PRMFIL	E9
PROJCT	B0
RASTOP	DA
RASTRD	DB
RASTWR	DC
RBAND	E1
RDEFIN	54
RECT	34
RECTR	35
RESETF	04
RFONT	55
SARC	F4
SBLINK	E4
SCIRC	F2
SDRAW	FA
SDRAWR	FB
SECTOR	3D
SELIPSE	F3
SMOVE	F8
SMOVER	F9
SPOLY	FC
SPOLYR	FD
SRECT	F0
SRECTR	F1
SSECT	F5
STEST	62
TANGLE	82
TASPCT	8B
TCHROT	8A
TDEFIN	84
TEXT	80
TEXTC*	8C
TEXTP	83
TEXTPC*	8D
TJUST	85
TSIZE	81
TSTYLE	88
VDISP	D5
VFREQ	61
VWIDEN	A0
VWMATX	A7
VWPORT	B2
VWROTX	A3
VWROTY	A4
VWROTZ	A5
VWRPT	A1
WAIT	05
WINDOW	B3
XHAIR	E2
XMOVE	E3

*** Note: These commands are available only in Hex communications mode.**

J.2 Commands by Hex Opcode

Opcode	Name
01	NOOP
04	RESETF
05	WAIT
06	COLOR
07	FLOOD
08	POINT
09	POINT3
0F	CLEARS
10	MOVE
11	MOVER
12	MOVE3
13	MOVER3
28	DRAW
29	DRAWR
2A	DRAW3
2B	DRAWR3
30	POLY
31	POLYR
32	POLY3
33	POLYR3
34	RECT
35	RECTR
38	CIRCLE
39	ELIPSE
3C	ARC
3D	SECTOR
43 41 20	CA
43 58 20	CX
50	LUTRD
51	FLAGRD
52	MATXRD
53	LUTXRD
54	RDEFIN
55	RFONT
60	ERROR
61	VFREQ
62	STEST
70	CLBEG
71	CLEND
72	CLRUN
73	CLOOP
74	CLDEL
75	CLRD
78	CLMOD
80	TEXT
81	TSIZE
82	TANGLE
83	TEXTP
84	TDEFIN
85	TJUST
88	TSTYLE
89	GTDEF
8A	TCHROT
8B	TASPCT
8C	TEXTC*
8D	TEXTPC*
90	MDIDEN
91	MDORG
92	MDSCAL
93	MDROTX
94	MDROTY
95	MDROTZ
96	MDTRAN
97	MDMATX
A0	VWIDEN
A1	VWRPT
A3	VWROTX
A4	VWROTY
A5	VWROTZ
A7	VWMATX
A8	DISTH
A9	DISTY
AA	CLIPH
AB	CLIPY
AF	CONVERT
B0	PROJCT
B1	DISTAN
B2	VWPORT
B3	WINDOW
B4	EXPAND
C0	AREA
C1	AREABC
C8	BLINK
C9	LUTSTO
CA	COLMOD
CB	BCOLOR
D5	VDISP
D6	PMASK
D8	IMAGER
D9	IMAGEW
DA	RASTOP
DB	RASTRD
DC	RASTWR
E1	RBAND
E2	XHAIR
E3	XMOVE
E4	SBLINK
E5	BLINKX
E6	LUTX
E7	AREAPT
E8	MASK
E9	PRMFIL
EA	LINPAT
EB	LINFUN
EC	LUTINT
ED	LUTSAV
EE	LUT
EF	FILMSK
F0	SRECT
F1	SRECTR
F2	SCIRC
F3	SELIPSE
F4	SARC
F5	SSECT
F8	SMOVE
F9	SMOVER
FA	SDRAW
FB	SDRAWR
FC	SPOLY
FD	SPOLYR
FF	PDRAW

*** Note: These commands are available only in Hex communications mode.**

PRODUCT FAILURE REPORT

If you are returning one of our products for repair, you must fill out this form and return it with the defective unit. The information so provided is necessary for us to provide a high standard of service.

COMPANY NAME AND ADDRESS:

NAME OF UNIT:
MODEL NO.(on silkscreen):
SERIAL NO.(on label):
DATE UNIT RECEIVED: DATE UNIT FAILED:
OR DEAD ON ARRIVAL □

MEMORY BASE ADDRESS USED:
I/O BASE ADDRESS USED:
PLEASE DESCRIBE THE SYSTEM THAT THE UNIT IS USED IN (CPU, BUS, MEMORY, ETC.):

UNIT CONFIGURATION (50 or 60 Hz, attributes used, display resolution selected, etc.):

PLEASE DESCRIBE THE FAULT:

FAULT IS CONSTANT □ FAULT IS INTERMITTENT □

NOTE: No merchandise will be accepted by MATROX for replacement or repair unless accompanied by an RMA number obtained from our Application Engineering Department.

RMA Number:

THE FOLLOWING SPACE IS FOR FACTORY USE ONLY

CORRECTIVE STEPS TAKEN:

MATROX Electronic Systems Limited,
1055 St. Regis Boulevard,
Dorval, Quebec,
CANADA H9P 2T4

Telephone: (514)685-2630 Telex: 05-822798 FAX: (514)685-2853

matrox
electronic systems ltd.

1055 ST. RÉGIS BLVD., DORVAL, QUÉBEC H9P 2T4, CANADA
TEL.: (514) 685-2630 TELEX: 05-822798
FAX.: 514-685-2853

Matrox Electronic Systems Ltd reserves the right to make changes in specifications at any time and without notice. The information furnished by Matrox Electronic Systems Ltd in this publication is believed to be accurate and reliable. However, no responsibility is assumed by Matrox Electronic Systems Ltd for its use; nor for any infringements of patents or other rights of third parties resulting from its use. No license is granted under any patents or patent rights of Matrox Electronic Systems Ltd.