GRAFIN2 USER'S MANUAL

(004-02704-01)

5510 N.E. Elam Young Parkway, Hillsboro, OR 97123

DISCLAIMER

Revision Number	Revision History	Date
-00	First Issue. Supports version 2.5 of the Ω500 microcode and version 3.6 of the Ω400 and Ω300 microcode.	7/84
-01	Adds two new commands for tablet-to-screen conversions. Adds a new chapter on writing commands in FORTH. Adds an appendix on Metheus FORTH.	10/84

This equipment complies with the requirements in part 15 of FCC rules for a class A computing device. Operation of this equipment in a residential area may cause unacceptable interference to radio and TV reception requiring the operator to take whatever steps are necessary to correct interference.

SERVICE INFORMATION

Contact the Graphics System Service Center (GSSC) when you need any Metheus graphics system repaired, replaced, or upgraded. This includes both hardware and firmware products. Following these simple instructions when you call or write will insure the quickest possible response to your request.

(1) You must provide these three items when you call or write to the service center:

Model Number
Serial Number
Purchase Order Number

Model numbers and serial numbers are marked on the outside of the chassis packaged products, on the board artwork, or on the firmware. The purchase order number authorizes the service center to charge for services. They can also provide the latest upgrade and service contract costs.

(2) Obtain a Return Authorization (RA) number from the service center BEFORE sending any equipment. Use this number in all correspondence.

Contact the Service Center at this address:

Graphics System Service Center
Metheus Corporation
5510 N. E. Elam Young Parkway
Hillsboro, Oregon 97123

(503) 640-8000

PREFACE

This manual is for new users of GRAFIN2. GRAFIN2 replaces the earlier GRAFIN option. This manual describes the commands and features of GRAFIN2. It assumes that you know how to use the Omega Display Controller and your graphics input device. You do not need to know how to use GRAFIN.

GRAFIN2 is an optional hardware interface package for Omega display controllers. GRAFIN2 is compatible with all host interfaces that support the Ω300, Ω400 and Ω500 series Display Controllers.

GRAFIN2 commands are used in programs to allow input from a graphics tablet or mouse. GRAFIN2 supports the Summagraphics[1] tablet and mouse, and GTCO[2] tablet.

GRAFIN2 is not compatible with GRAFIN. The differences between them and the installation procedures for GRAFIN2 are discussed in Appendix A and B.

MANUAL OVERVIEW

This manual contains three chapters and three appendixes:

- o Chapter 1 defines terms and discusses the GRAFIN2 commands by function.
- o Chapter 2 discusses the GRAFIN2 commands in detail.
- o Chapter 3 illustrates how to write customized GRAFIN2 commands using the Metheus version of FORTH[3].
- o Appendix A contains the installation procedure for GRAFIN2.
- o Appendix B describes the differences between GRAFIN2 and GRAFIN.
- o Appendix C summarizes Metheus FORTH.

---

[1]Summagraphics is a registered tradename of Summagraphics Corporation.

[2]GTCO is a registered tradename of GTCO Corporation.

[3]FORTH is a registered trademark of FORTH, Inc.

NOTE

If you are installing GRAFIN2 or upgrading GRAFIN to GRAFIN2, you should read Appendix A first. Installation should be performed by a qualified service person only.

RELATED PUBLICATIONS

AXIA Graphics Package User's Manual, *Order Number 004-01086*

FORTRAN Opcode Library User's Manual, *Order Number 004-01705*

Ω400 User's Manual, *Order Number 004-01085*

Ω500 User's Manual, *Order Number 004-02207*

The MicroFORTH Primer, FORTH, Inc., Manhattan Beach, CA, 1978

FORTH Fundamentals, Dilitheum Press, Beaverton, OR, 1983.

CONTENTS

CHAPTER 1
OVERVIEW

CHAPTER 2
GRAFIN2 COMMAND DICTIONARY

CHAPTER 3
WRITING CUSTOMIZED GRAFIN2 COMMANDS

APPENDIX A
GRAFIN2 INSTALLATION

APPENDIX B
GRAFIN AND GRAFIN2 DIFFERENCES

APPENDIX C
METHEUS FORTH

FIGURES

TABLES

TABLES

Chapter 1

OVERVIEW

This chapter contains four sections:

- o Definition of terms used in this manual
- o The GRAFIN2 commands listed by function
- o A discussion of the GRAFIN2 initialization commands
- o An example of GRAFIN2 use.

DEFINITION OF TERMS

The GRAFIN2 interface accepts graphics input from a bit-pad, tablet, or mouse. In this manual, the input device is referred to as the ***tablet***. The tracking part of the tablet (the stylus, puck, or mouse) is referred to in this manual as the ***mouse***. The mouse's position is indicated on the screen by the ***cursor***.

GRAFIN2 keeps track of ***button events***. Button event information includes which button on the mouse was pressed (the ***button** ID*), and the coordinates when the button was pressed. Button events are stored in the ***button event queue***. Events are read from the queue in first-in, first-out order. A button event can be defined as the push of a button (leading edge mode), the release of a button (trailing edge mode), the button held down (level mode), or the push and release of the button (both edge mode -- two events are recorded).

The button ID byte is identical to the data format byte sent from the tablet. The button ID is formed from the button information bits in the Summagraphics Bit Pad One Binary Data Format (bits F0 through F3), the SummaMouse Bit Pad One Data Format (bits L, M, and R), or the GTCO DIGI-PAD 5 High-Resolution Format (bits PB1, PB2, PB4, PB8, and PBA). Figure 1-1 illustrates these formats. Refer to your tablet manual for information on interpreting the button ID.

Summagraphics Bit Pad One Binary Data Format	P	1	F3	F2	F1	F0	0	0
Summa Mouse Bit Pad One Data Format	P	1	0	L	M	R	0	0
GTCO DIGI - PAD 5 High-Resolution Format	1	PBA	PB8	PB4	PB2	PB1	X15	X14

F-0081

Figure 1-1. Button ID Formats

The origin (0,0) point on the Omega screen is the upper left corner while on most tablets it is the lower left corner. Several commands map the tablet to the screen and adjust the coordinate systems. (A default mapping is performed by the INIT GRAFIN2 command.) Coordinates can be reported to or from the host as either *tablet coordinates* or *screen coordinates*. Coordinates are given in the format (X,Y). Each coordinate is two bytes: low-x, high-x; low-y, high-y.

The borders of the mapped area on the screen form a *bounding box*. This is usually the edge of the screen, but you can also specify a different bounding box with the SET CLIP MODE and SET WRAP MODE commands. The coordinates of the corners of the bounding box are always given in screen coordinates.

When the mouse reaches the edge of the area mapped to the bounding box, the cursor either *clips* or *wraps*. Figure 1-2 illustrates the behavior of the cursor in clip and wrap modes. In clip mode, the cursor always remains within the bounding box. In wrap mode, when the mouse moves beyond the mapped area, the cursor reappears on the opposite side of the bounding box.

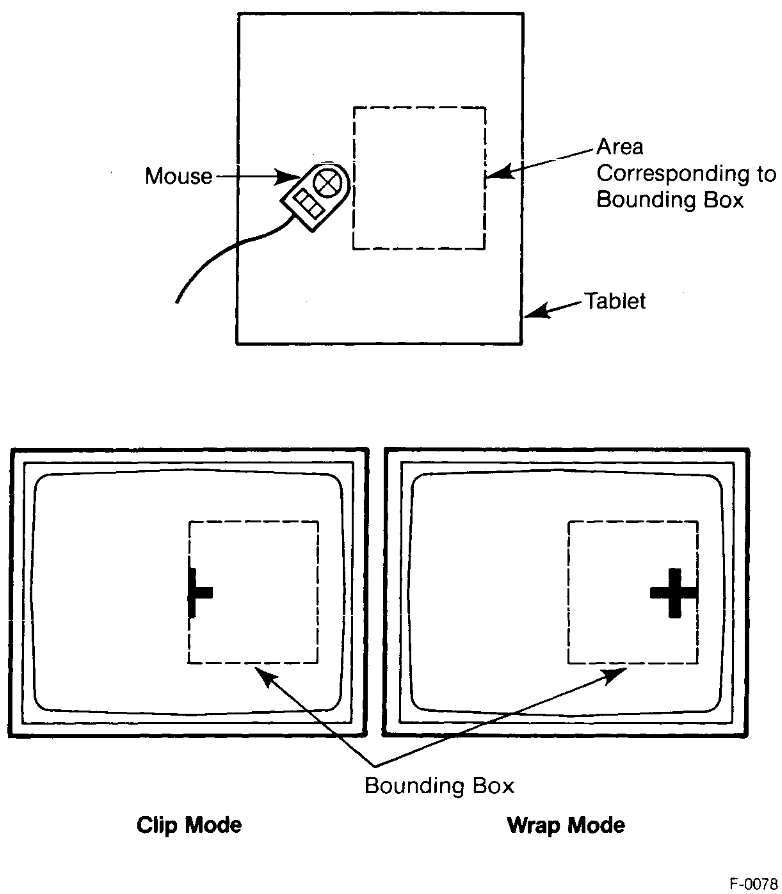

Figure 1-2. Clip Mode and Wrap Mode

GRAFIN2 COMMANDS

GRAFIN2 commands fall into three categories:

- o Environment commands
- o Cursor-Tracking commands
- o Event-Queue commands

Environment commands initialize the system, inquire the version number and inquire the error status. The last section of this chapter discusses the default initialization conditions and how to change them.

Cursor-Tracking commands allow you to select a style of cursor and to control when it appears on the screen. When the cursor is displayed, it tracks the mouse.

Event-Queue commands keep track of button events so that button input is sent to the host in an orderly manner.

Table 1-1 lists the GRAFIN2 commands. Following the table, each category of commands is discussed separately.

TABLE 1-1. GRAFIN2 Commands

Environment	*Cursor-Tracking*	*Event-Queue*
INIT GRAFIN2	CROSSHAIR CURSOR	EVENT COUNT
INQUIRE ERROR	CURSOR OFF	FLUSH Q
INQUIRE VERSION	CURSOR ON	READ CURRENT POSITION
SET CLIP MODE	RUBBERBAND BOX	READ Q
SET OFFSET/SCALE	RUBBERBAND LINE	READ Q AND WAIT
SET SCREEN SIZE	SET CURSOR SIZE	SET Q MODE
SET TABLET SIZE	SKETCH	
SET WRAP MODE		
WRITE TABLE		

NOTE

Every GRAFIN2 opcode must begin with these two bytes: the first is 4Ah (74 decimal) and the second is the specific command opcode.

Environment Commands

The environment commands allow you to:

- o Map the tablet to the screen (INIT GRAFIN2; SET OFFSET/SCALE; SET SCREEN SIZE; SET TABLET SIZE).
- o Set up a bounding box on the screen (SET CLIP MODE; SET WRAP MODE).
- o Inquire the error status of the system (INQUIRE ERROR).
- o Inquire the version of the firmware (INQUIRE VERSION).
- o Initialize the tablet (WRITE TABLET).

The environment commands are summarized in Table 1-2.

TABLE 1-2. Environment Commands

Name	*Hex Opcode*	*Decimal Opcode*	*Arguments*	*Returns*
INIT GRAFIN2	10	16	none	none
INQUIRE ERROR	25	37	none	2: (*a.)
INQUIRE VERSION	26	38	none	2: (*b.)
SET CLIP MODE	22	34	8:(X_1,Y_1,X_2,Y_2)	none
SET OFFSET/SCALE	20	32	8: (*c.)	none
SET SCREEN SIZE	11	17	4: (*d.)	none
SET TABLET SIZE	12	18	4: (*d.)	none
SET WRAP MODE	21	33	8:(X_1,Y_1,X_2,Y_2)	none
WRITE TABLET	24	36	(*e.)	none

(*a.) first byte = error code; second byte = error count
(*b.) first byte = version code; second byte = reserved
(*c.) 2 bytes each (16-bit, 2's complement): X-Offset, X-Scale, Y-Offset, Y-Scale
(*d.) 2 bytes each of width and height of screen or tablet.
(*e.) variable number of bytes (device dependent)

The *Arguments* and *Returns* columns in Table 1-2 indicate the number of bytes (if any) required as input or returned as output. For example, "8:(X_1,Y_1,X_2,Y_2)" means that the command opcode is followed by eight bytes of information, in this case, the coordinates (four bytes) of one corner of the bounding box and the coordinates (four bytes) of the opposite corner of the bounding box.

Cursor-Tracking Commands

The cursor tracking commands allow you to select different cursor types. The cursor types are:

- o Crosshair cursor (default) (CROSSHAIR CURSOR, SET CURSOR SIZE).
- o Rubberband box cursor (RUBBERBAND BOX).
- o Rubberband line cursor (RUBBERBAND LINE).
- o Sketching cursor (SKETCH).

When you select a cursor type, it becomes the currently selected cursor and is displayed on the screen. The cursor remains on the screen until explicitly removed with INIT GRAFIN2 or CURSOR OFF. Generally, you will want to remove the cursor before drawing over its location. If you draw over the cursor, a shadow image of the cursor's pixels will remain on the screen.

The CURSOR ON command displays the currently selected cursor. To change cursors, enter one of the four cursor commands.

The cursor tracking commands are summarized in Table 1-3. The *Arguments* and *Returns* columns indicate the number of bytes (if any) required as input or returned as output. (X,Y) indicates coordinates.

TABLE 1-3. Cursor-Tracking Commands

Name	*Hex Opcode*	*Decimal Opcode*	*Arguments*	*Returns*
CROSSHAIR CURSOR	31	49	none	none
CURSOR OFF	3F	63	none	none
CURSOR ON	30	48	none	none
RUBBERBAND BOX	33	51	4: (X,Y)	none
RUBBERBAND LINE	32	50	4: (X,Y)	none
SET CURSOR SIZE	23	35	4: (*a.)	none
SKETCH	34	52	none	none

(*a.) 2 bytes each: *halfwidth* and *halfheight* of the crosshair cursor in pixels

Event-Queue Commands

The event queue stores up to 100 button events from the mouse. There are two types of event queue commands:

- o Commands to manage the event queue itself (EVENT COUNT; FLUSH Q; SET Q MODE).
- o Commands that return event information to the host (READ CURRENT POSITION; READ Q; READ Q AND WAIT).

EVENT COUNT keeps track of the number of events in the event queue. If the queue count exceeds 100, new events are ignored and a queue overflow error will be indicated by the INQUIRE ERROR command. FLUSH Q clears the event queue. Use the SET Q MODE command to select which edge is recorded and whether tablet or screen coordinates are reported.

The event queue commands are summarized in Table 1-4. The *Arguments* and *Returns* columns indicates the number of bytes (if any) required as input or returned as output. (X,Y) indicates coordinates.

TABLE 1-4. Event-Queue Commands

Name	*Hex Opcode*	*Decimal Opcode*	*Arguments*	*Returns*
EVENT COUNT	40	64	none	2: (*a.)
FLUSH Q	45	69	none	none
READ CURRENT POSITION	44	68	none	4. (X,Y)
READ Q	42	66	none	6: (*b.)
READ Q AND WAIT	43	67	none	6: (*b.)
SET Q MODE	41	65	2: (*c.)	none

(*a.) first byte = event count; second byte = reserved
(*b.) first byte = button ID; second byte = reserved; third through sixth byte = (X,Y) coordinates
(*c.) first byte = type of button edge recorded; second byte = screen or tablet coordinates reported

INITIALIZING GRAFIN2

In general, when you use GRAFIN2 you will want to initialize the system and possibly change the default conditions. This section looks at the defaults of the INIT GRAFIN2 command and discusses the commands used to change the default conditions:

- o INIT GRAFIN2. Initializes the graphics input environment.
- o SET SCREEN SIZE or SET TABLET SIZE. Changes the default value of the screen or tablet size used in mapping.
- o SET OFFSET/SCALE. Adjusts the mapping of the tablet to the screen.
- o SET Q MODE. Selects the button event type and coordinate system.
- o Cursor-Tracking Commands. Selects and displays a cursor.
- o SET CLIP/WRAP MODE. Selects the mode and defines a bounding box.

INIT GRAFIN2 maps a 2200 by 2200 tablet to a 1024 by 768 pixel screen. If your tablet or screen is a different size, use SET SCREEN SIZE or SET TABLET SIZE to change these default values. You can also use SET OFFSET/SCALE.

INIT GRAFIN2 clears the button event queue, defines a button event as a leading edge, and specifies that screen coordinates are sent to the host. To select another button edge type or change to tablet coordinates, use SET Q MODE.

INIT GRAFIN2 selects a 33 by 33 pixel crosshair cursor and erases the cursor from the screen. To display the crosshair cursor, use CURSOR ON or CROSSHAIR CURSOR. To change the size of the crosshair cursor, use SET CURSOR SIZE. To select a different cursor and display it on the screen, use RUBBERBAND BOX, RUBBERBAND LINE, or SKETCH. CURSOR OFF erases the cursor from the screen. CURSOR ON displays the most recently selected cursor.

INIT GRAFIN2 selects clip mode, with a 1024 by 768 pixel screen as the bounding box. To set a different bounding box, use SET CLIP MODE. To change to wrap mode and set a bounding box, use SET WRAP MODE.

GRAFIN2 EXAMPLE

Figure 1-3 contains sample GRAFIN2 commands, followed by comments. The numbers in the left-most column refer to the comments and are not part of the GRAFIN2 commands.

(1) **4A 10** -- INIT GRAFIN2
(2) **4A 11 00 05 00 04** -- SET SCREEN SIZE
(3) **4A 41 02 00** -- SET Q MODE
(4) **4A 33 64 00 32 01** -- RUBBERBAND BOX
(5) **4A 44** -- READ CURRENT POSITION (4 bytes returned)

(Move the mouse around the tablet and press a few buttons.)

(6) **4A 42** -- READ Q (6 bytes returned)
(7) **4A 45** -- FLUSH Q
(8) **4A 3F** -- CURSOR OFF
(9) **4E 00 60** -- Set color 0 (black) and clear screen

Figure 1-3. Sample of GRAFIN2 Commands

The code in the above example performs these actions:

(1) Initializes GRAFIN2
(2) Adjusts the mapping to an 1280 by 1024 screen
(3) Sets the queue mode to trailing edge and tablet coordinates
(4) Enables and displays a rubberband box cursor
(5) Reads the current position of the mouse
(6) Reads information from the event queue
(7) Clears the event queue
(8) Removes the cursor from the screen
(9) Sets color to black and clears the screen

Chapter 2

GRAFIN2 COMMAND DICTIONARY

This chapter alphabetically lists the GRAFIN2 commands. Terms used in this chapter are defined in Chapter 1. The commands are summarized by function in tables in Chapter 1.

The command entries in this chapter follow a standard format which includes:

- o Name and short description of the command.
- o The functional group of the command.
- o Hex opcode and arguments (if any).
- o Decimal opcode and arguments (if any).
- o Definition of the argument(s).
- o Bytes returned (if any).
- o Description of the command.
- o Related commands.

In the command arguments, opcodes are in **boldface** while variables you must supply are in *italics*. Be sure to precede the opcode with **4Ah** to indicate a GRAFIN2 command. In the examples, bytes that you enter are given in **boldface**; bytes that the computer returns are given in regular (this style) type.

CROSSHAIR CURSOR -- Selects a crosshair cursor shape.
(Cursor-Tracking Command)

Hex: **31**

Decimal: **49**

Arguments: None

Bytes Returned: None

Description: CROSSHAIR CURSOR selects the crosshair shape for the cursor and displays the cursor on the screen. While displayed, the cursor tracks the mouse. The cursor is drawn in the complementary color of the existing pixels. A crosshair cursor is the default shape of the INIT GRAFIN2 and SKETCH commands. The default size of the cursor is 33 by 33 pixels, set by INIT GRAFIN2. Use the CURSOR SIZE command to change the size of the cursor. The cursor remains on the screen until removed by INIT GRAFIN2 or CURSOR OFF.

Related Command:

CURSOR OFF
CURSOR ON
INIT GRAFIN2
SET CURSOR SIZE

CURSOR OFF -- Removes the cursor from the screen.
(Cursor-Tracking Command)

Hex: **3F**

Decimal: **63**

Arguments: None

Bytes Returned: None

Description: CURSOR OFF removes the cursor from the screen. Generally, you will want to remove the cursor whenever you plan to draw a figure that will overlap the cursor. If you do not remove the cursor (or else move it out of the way), and draw over the cursor, a shadow image of the cursor's pixels will remain at that location.

Related Commands:

CROSSHAIR CURSOR
CURSOR ON
INIT GRAFIN2
RUBBERBAND BOX
RUBBERBAND LINE
SKETCH

CURSOR ON -- Displays the currently selected cursor.
(Cursor-Tracking Command)

Hex: **30**

Decimal: **48**

Arguments: None

Bytes Returned: None

Description: CURSOR ON displays the most recently selected cursor. While displayed, the cursor tracks the mouse. The cursor can be a crosshair cursor (the default), a rubberband line, a rubberband box or a sketching cursor. Select the cursor with the CROSSHAIR CURSOR, RUBBERBAND BOX, RUBBERBAND LINE, or SKETCH commands. The INIT GRAFIN2 command selects a 33 by 33 pixel crosshair cursor but does not display it on the screen. The cursor remains on the screen until removed by INIT GRAFIN2 or CURSOR OFF.

Related Commands:

CROSSHAIR CURSOR
CURSOR OFF
INIT GRAFIN2
RUBBERBAND BOX
RUBBERBAND LINE
SET CURSOR SIZE
SKETCH

EVENT COUNT -- Returns the number of events in the event queue. (Event-Queue Command)

Hex: **40**

Decimal: **64**

Arguments: None

Bytes Returned: 2 bytes -- (low, high) of event count

Description: EVENT COUNT tells you the number of items in the event queue. The event queue holds up to 100 events. Events that occur after the queue is full are ignored. (You can check if the queue has overflowed with the INQUIRE ERROR command.) Events are read from the queue in first-in, first-out order. EVENT COUNT returns a 16-bit, two's complement number: low byte, high byte.

Example:

4A 40	4Ah must precede all GRAFIN2 commands.
1A 00	The event queue contains 26 button hits.

Related Commands:

FLUSH Q
INQUIRE ERROR

FLUSH Q -- Clears the button event queue.
(Event-Queue Command)

Hex: **45**

Decimal: 69

Arguments: None

Bytes Returned: None

Description: FLUSH Q deletes all information in the button event queue and resets the count in EVENT COUNT to zero. Any pending requests (from the READ Q AND WAIT command) are also flushed. However, data generated by READ Q AND WAIT but not yet read by the host are not flushed. (See comments at READ Q AND WAIT.)

Related Commands:

EVENT COUNT
READ Q
READ Q AND WAIT

INIT GRAFIN2 -- Initialize GRAFIN2 to its power-up defaults. (Environment Command)

Hex: **10**

Decimal: **16**

Arguments: None

Bytes Returned: None

Description: The INIT GRAFIN2 command initializes the GRAFIN2 environment to the following default conditions:

o Offset/Scale = maps a 2200 by 2200 tablet to a 1024 by 768 screen
o Clip/Wrap mode = clip (bounding box is 1024 by 768 pixels)
o Cursor type = 33 by 33 pixel crosshair
o Cursor is erased from screen
o Button event recognition = leading edge mode
o Coordinates reported = screen coordinates
o Event queue = cleared
o Error status = cleared

INIT GRAFIN2 restores the graphics environment to its power-up conditions. The last section of Chapter 1, Initializing GRAFIN2, discusses the commands used to change the INIT GRAFIN2 default conditions.

Related Commands:

CROSSHAIR CURSOR
CURSOR ON
FLUSH Q
INQUIRE ERROR
SET CLIP MODE
SET CURSOR SIZE
SET OFFSET/SCALE
SET Q MODE
SET SCREEN SIZE
SET TABLET SIZE

INQUIRE ERROR -- Returns code indicating most recent error. (Environment Command)

Hex: 25

Decimal: 37

Arguments: None

Bytes Returned: 2 bytes – error code and error count

Description: INQUIRE ERROR returns two bytes. The first byte contains the code of the most recent error and the second byte contains the number of errors (up to 256) since the last INQUIRE ERROR command. The error codes are:

0 = no error when error count byte is 0; macro compilation error when error count byte is non-0
1 = cold start error
2 = warm start error
3 = stack error
4 = unknown FORTH command
5 = reserved
6 = event-queue overflow (too many button hits)
7 = unknown GRAFIN2 command

These errors are informational and non-fatal with the exception of errors 1, 2, and 3. However, it is unlikely you will ever see those three errors.

Example:

4A 25	4Ah must precede all GRAFIN2 commands.
06 0C	Indicates 12 errors; "event-queue overflow" was most recent.

Related Commands:

EVENT COUNT

INQUIRE VERSION -- Returns the GRAFIN2 firmware version. (Environment Command)

Hex: **26**

Decimal: **38**

Arguments: None

Bytes Returned: 2 bytes -- version code and reserved byte

Description: INQUIRE VERSION reports the current implementation version of the GRAFIN2 firmware. The first byte returns the GRAFIN2 firmware version number. The second byte is reserved for future use.

Example:

4A 26	4Ah must precede all GRAFIN2 commands.
10 00	Indicates version 1.0 of the GRAFIN2 firmware.

Related Commands: None

READ CURRENT POSITION -- Reports the position of the mouse. (Event-Queue Command)

Hex: **44**

Decimal: **68**

Arguments: None

Bytes Returned: 4 bytes -- low-x, high-x, low-y, high-y

Description: READ CURRENT POSITION reads and reports the current location of the mouse in either tablet or screen coordinates. The SET Q MODE command determines whether the location of the mouse is reported in tablet or screen coordinates. READ CURRENT POSITION does not affect the event queue.

Related Commands:

SET Q MODE

READ Q -- Reads next entry from the button-event queue.
(Event-Queue Command)

Hex: **42**

Decimal: **66**

Arguments: None

Bytes Returned: 6 bytes -- button ID, reserved byte, low-x, high-x, low-y, high-y

Description: READ Q reads the next entry from the button-event queue and removes the entry from the queue. Events are read in first-in, first-out order. READ Q returns the button ID, a byte reserved for future use, and the coordinates when the button was hit. READ Q reports either tablet or screen coordinates, depending on the setting of the SET Q MODE command. When the queue is empty, the button ID is 00 and the current X and Y coordinates are returned, as in READ CURRENT POSITION.

Example:

4A 42	4Ah must precede all GRAFIN2 commands
02 00	button 2 was hit; reserved byte
2C 01	X coordinate = 12Ch
C8 00	Y coordinate = C8h

Related Commands:

READ CURRENT POSITION
READ Q AND WAIT
SET Q MODE

READ Q AND WAIT -- Reads next entry from button-event queue. (Event-Queue Command)

Hex: **43**

Decimal: **67**

Arguments: None

Bytes Returned: 6 bytes -- button ID, reserved byte, low-x, high-x, low-y, high-y

Description: READ Q AND WAIT reads the next entry from the button-event queue and removes the entry from the queue. Events are read in first-in, first-out order. READ Q AND WAIT reports the button ID, a byte reserved for future use, and the coordinates when the button was hit. The coordinates are in screen or tablet coordinates, depending on the SET Q MODE command.

READ Q AND WAIT is the same as READ Q except when the queue is empty. When the queue is empty, nothing is reported until an event occurs. (When the queue is empty, READ Q reports a button ID of 00 and the current coordinates.)

You can have more than one READ Q AND WAIT commands pending by issuing several in a row. Be sure to read as many 6-byte groups as READ Q AND WAIT commands as issued. Otherwise, you may get inappropriate data from the Omega. The FLUSH Q command clears any pending READ Q AND WAIT commands but not the data that may have been generated but not yet read.

Example:

4A 43	4Ah must precede all GRAFIN2 commands
01 00	button 1 was hit; reserved byte
2C 01	X coordinate = 12Ch
C8 00	Y coordinate = C8h

Related Commands:

READ Q
SET Q MODE

RUBBERBAND BOX -- Selects rubberband box style cursor. (Cursor-Tracking Command)

Hex: **33** *anchor*

Decimal: 51 *anchor*

Arguments: 4 bytes -- *anchor* = low-x, high-x, low-y, high-y

Bytes Returned: None

Description: RUBBERBAND BOX generates a rectangle between the anchor point and the cursor. The anchor point is given in screen coordinates. While displayed, the cursor tracks the mouse. The cursor remains on the screen until removed by INIT GRAFIN2 or CURSOR OFF. The rectangle is drawn in the complement color of the existing pixels.

Note that when the cursor is exactly on the X-axis or Y-axis (reducing the box to a single horizontal or vertical line) the line will disappear due to complementing the pixels twice. Placing the anchor point just outside a bounding box eliminates this problem. (See SET CLIP MODE or SET WRAP MODE.)

Example:

4A 33	4Ah must precede the GRAFIN2 command
64 00	X coordinate of anchor = 64h
32 01	Y coordinate of anchor = 132h

Related Commands:

CURSOR OFF
CURSOR ON
RUBBERBAND LINE

RUBBERBAND LINE -- Selects rubber line style cursor.
(Cursor-Tracking Command)

Hex: **32** *anchor*

Decimal: **50** *anchor*

Arguments: 4 bytes -- ***anchor*** = low-x, high-x, low-y, high-y

Bytes Returned: None

Description: RUBBERBAND LINE generates a line between the anchor point and the cursor. The anchor point is given in screen coordinates. While displayed, the cursor tracks the mouse. The cursor remains on the screen until removed by INIT GRAFIN2 or CURSOR OFF. The line is drawn in the complement color of the existing pixels.

Note that when the cursor is exactly on the anchor point, the cursor will disappear due to complementing the pixels twice. Placing the anchor point just outside a bounding box eliminates this problem. (See SET CLIP MODE or SET WRAP MODE.)

Example:

4A 32	4Ah must precede the GRAFIN2 command
20 02	X coordinate of anchor = 220h
32 01	Y coordinate of anchor = 132h

Related Command:

CURSOR OFF
CURSOR ON
RUBBERBAND BOX

SET CLIP MODE -- Sets clip mode for cursor tracking.
(Environment Command)

Hex: **22** *corners*

Decimal: **34** *corners*

Arguments: 8 bytes -- *corners* = low-x_1, high-x_1, low-y_1, high-y_1; low-x_2, high-x_2, low-y_2, high-y_2

Bytes Returned: None

Description: SET CLIP MODE turns on clip mode and defines the diagonal corners of the bounding box. The coordinates of the bounding box are always screen coordinates. In clip mode, when you move the mouse beyond the area defined by the bounding box, the cursor is clipped at the boundary. The cursor will still move along the other axis as long it is within the boundary. Refer to Figure 1-2.

Example:

4A 22	4Ah must precede all GRAFIN2 commands
64 00	X_1 = 64h
32 00	Y_1 = 32h
2C 01	X_2 = 12Ch
C8 00	Y_2 = C8h

Related Commands:

SET WRAP MODE

SET CURSOR SIZE -- Sets the size of the crosshair cursor. (Cursor-Tracking Command)

Hex: **23** *halfwidth* *halfheight*

Decimal: **35** *halfwidth* *halfheight*

Arguments: 4 bytes -- ***halfwidth*** = low-halfwidth, high-halfwidth; ***halfheight*** = low-halfheight, high-halfheight. Range is 1 to 2047 (1h to 7FFh) pixels.

Bytes Returned: None

Description: SET CURSOR SIZE allows you to select the size of the crosshair cursor. The actual cursor size is:

2 x (***halfwidth*** or ***halfheight***) + 1.

The default size of the crosshair cursor is 33 by 33 pixels.

Example: To make a 45 by 45 (2Dh by 2Dh) pixel cursor:

4A 23	4Ah must precede all GRAFIN2 commands
16 00	halfwidth = 16h
16 00	halfheight = 16h

Related Commands:

CROSSHAIR CURSOR
SKETCH

SET OFFSET/SCALE -- Maps the tablet to the screen.
(Environment Command)

Hex: **20** *X-Offset X-Scale Y-Offset Y-Scale*

Decimal: **32** *X-Offset X-Scale Y-Offset Y-Scale*

Arguments: 8 bytes -- *X-Offset* = low-X-offset, high-X-offset, *X-Scale* = low-X-scale, high-X-scale; *Y-Offset* = low-Y-offset, high-Y-offset, *Y-Scale* = low-Y-scale, high-Y-scale

Bytes Returned: None

Description: The SET OFFSET/SCALE command sets values used to map tablet coordinates to screen coordinates. The tablet-to-screen conversion equations are:

Screen X = X-Offset + (X-Scale x Tablet X)
Screen Y = Y-Offset + (Y-Scale x Tablet Y)

The offsets move the coordinates a constant amount along each axis. The offsets are each two bytes of data in 16-bit, two's complement form:

X-Offset, *Y-Offset* = -32768 through +32767
(8000h through 7FFFh)

The scale factor shrinks or expands the tablet coordinates to fit the screen. The scale factors are each two bytes of data in 16-bit, two's complement form. The most-significant 4 bits form a signed integer and the least-significant 12 bits form the fractional part:

X-Scale, *Y-Scale* = -8.00000 through +7.99976
(8000h through 7FFFh)

To convert a decimal scale factor to hex, multiply the decimal number by 4096, convert to hex and truncate to 16 bits. Table 2-1 contains some decimal scale factors and their hex equivalents.

Table 2-1. Decimal and Hex Scale Factors

Decimal	*Hex*	*Decimal*	*Hex*
1.00000	1000	1.50000	1800
0.50000	0800	-1.50000	E800
-0.50000	F800	3.00000	3000
0.75000	0C00	-3.00000	D000
-0.75000	F400	-7.50000	8800

The (0, 0) point on the Omega screen is the upper left corner while on most tablets it is the lower left corner. Therefore, you will usually want to reverse the Y coordinate system so that cursor movement on the screen is the same as the mouse. You can do this by setting a negative Y-Scale factor and a positive full screen Y-Offset. Tables 2-2 and 2-3 contain the scale and offset values (in hex and decimal) to map a 2200 by 2200 Summagraphics tablet to the Ω300, Ω400 and Ω500.

Table 2-2. Tablet-To-Screen Conversions (Hexadecimal)

Screen Size	*X-Offset*	*X-Scale*	*Y-Offset*	*Y-Scale*
1024 by 768	0000	0072	02FF	FA6C
736 by 552	0000	055A	0227	F6FF
1280 by 1024	0000	094E	03FF	F890
640 by 512	0000	04A7	01FF	FC49

Table 2-3. Tablet-To-Screen Conversions (Decimal)

Screen Size	*X-Offset*	*X-Scale*	*Y-Offset*	*Y-Scale*
1024 by 768	0000	1906	767	-1424
736 by 552	0000	1370	511	-1025
1280 by 1024	0000	2382	1023	-1904
640 by 512	0000	1191	511	-951

Example: To map a 2200 by 2200 tablet to a 736 by 552 screen:

4A 20	4Ah must precede all GRAFIN2 commands
00 00	X-Offset = 0000h
5A 05	X-Scale = 055Ah
27 02	Y-Offset = 0227h
FF F6	Y-Scale = F6FFh

Related Commands:

INIT GRAFIN2
SET SCREEN SIZE
SET TABLET SIZE

SET Q MODE -- Sets button-detect and data-reporting modes. (Event-Queue Command)

Hex: **41** *detect-byte* *report-byte*

Decimal: **65** *detect-byte* *report-byte*

Arguments: 2 bytes -- ***detect-byte*** = type of button edge that constitutes an event; ***report-byte*** = whether tablet or screen coordinates are reported to the host.

Bytes Returned: None

Description: SET Q MODE determines which button edge(s) are detected and selects whether coordinates are reported as screen or tablet coordinates. Only the lower two bits of the detect byte are significant; only the lowest bit of the report byte is significant. All other bits are reserved for future use and should be set to zero.

Detect-byte values:
0 = level mode (events are recorded while a button is held down and the coordinates change)
1 = leading edge mode (button pushed)
2 = trailing edge mode (button released)
3 = both edge mode (two events -- button pushed and released)

The default for INIT GRAFIN2 is leading edge detection (value 1).

Report-byte values:
0 = reports in tablet coordinates
1 = reports in screen coordinates

The default for INIT GRAFIN2 is screen coordinates (value 1).

Example:

4A 41	4Ah must precede all GRAFIN2 commands
00 01	Selects level button input and screen coordinates.

Related Commands:

INIT GRAFIN2
READ CURRENT POSITION
READ Q
READ Q AND WAIT

SET SCREEN SIZE -- Sets screen size for mapping.
(Environment Command)

Hex: 11 *screenwidth screenheight*

Decimal: 17 *screenwidth screenheight*

Arguments: 4 bytes -- *screenwidth* = low-width, high-width; *screenheight* = low-height, high-height

Bytes Returned: None

Description: SET SCREEN SIZE maps the default tablet size to the screen size specified in the arguments. The default tablet size is 2200 by 2200 pixels. (The tablet size can be changed with SET TABLET SIZE. If you need to change both tablet and screen sizes, issue both commands.)

The bounding box for clip or wrap mode is set to the full screen

SET SCREEN SIZE does not change the default values used by INIT GRAFIN2.

Example:

4A 11	4A must precede all GRAFIN2 commands
00 05	*screenwidth* = 1280 (500h)
00 04	*screenheight* = 1024 (400h)

Related Commands:

INIT GRAFIN2
SET OFFSET/SCALE
SET TABLET SIZE

SET TABLET SIZE -- Sets tablet size for mapping.
(Environment Command)

Hex: **12** *tabletwidth tabletheight*

Decimal: **18** *tabletwidth tabletheight*

Arguments: 4 bytes -- ***tabletwidth*** = low-width, high-width; ***tabletheight*** = low-height, high-height

Bytes Returned: None

Description: SET TABLET SIZE maps the tablet size specified in the arguments to the default screen size. The default screen size is 1024 by 768 pixels. (The screen size can be changed with SET SCREEN SIZE. If you need to change both tablet and screen sizes, issue both commands.)

SET TABLET SIZE does not change the default values used by INIT GRAFIN2.

Example:

4A 12	4A must precede all GRAFIN2 commands
00 04	*tabletwidth* = 1024 (400h)
00 04	*tabletheight* = 1024 (400h)

Related Commands:

INIT GRAFIN2
SET OFFSET/SCALE
SET SCREEN SIZE

SET WRAP MODE -- Sets wrap-around mode for cursor tracking. (Environment Command)

Hex: 21 *corners*

Decimal: 33 *corners*

Arguments: 8 bytes -- *corners* = low-x_1, high-x_1, low-y_1, high-y_1; low-x_2, high-x_2, low-y_2, high-y_2

Bytes Returned: None

Description: SET WRAP MODE turns on wrap mode and defines the diagonal corners of the bounding box. The coordinates of the bounding box are always screen coordinates. If you move the mouse beyond the area defined by the bounding box, the cursor "wraps around" the boundary and reappears on the opposite side of the bounding box. Refer to Figure 1-2.

Example:

4A 22	4Ah must precede all GRAFIN2 commands
64 00	X_1 = 64h
32 00	Y_1 = 32h
2C 01	X_2 = 12Ch
C8 00	Y_2 = C8h

Related Command:

SET CLIP MODE

SKETCH -- Turns on sketching mode.
(Cursor-Tracking Command)

Hex: **34**

Decimal: **52**

Arguments: None

Bytes Returned: None

Description: SKETCH is used to make freeform drawings on the screen. SKETCH draws in the currently-selected Omega color while any button is held down on the mouse. Use the Omega SETCOL command to change color while drawing.

SKETCH uses the crosshair cursor. While displayed, the cursor tracks the mouse. The cursor remains on the screen until removed by INIT GRAFIN2 or CURSOR OFF.

Button events are entered in the event queue as defined by the SET Q MODE command.

Related Commands:

CURSOR OFF
CURSOR ON
SET CURSOR SIZE
SET Q MODE

WRITE TABLET -- Sends initialization bytes to tablet or mouse. (Environment Command)

Hex: **24** *init-bytes* **1B**

Decimal: **36** *init-bytes* **27**

Arguments: *init-bytes* = device-dependent bytes to initialize special functions of the tablet. Argument is terminated by an ASCII escape (1Bh).

Bytes Returned: None

Description: WRITE TABLET allows you to send initialization bytes to the tablet, for example, to change the sampling rate. (Refer to the manual of your tablet for the appropriate bytes.) In most cases, you will not need this command. The command is terminated by an ASCII "escape" code (1Bh).

Related Commands: None

Chapter 3

WRITING CUSTOMIZED GRAFIN2 COMMANDS

This chapter shows by example how to write customized GRAFIN2 commands using the Metheus version of FORTH[1]. In this chapter we assume that you are an experienced programmer and that you want to expand the functionality provided by GRAFIN2. If the existing GRAFIN2 commands meet your needs, you do not have to read this chapter.

The chapter contains three sections:

- o FORTH Fundamentals
- o Command Examples
- o GRAFIN2 Subroutines, Variables, and Pointers

The first part, "FORTH Fundamentals", introduces FORTH with some simple examples. The second section, "Command Examples", contains examples with comments. The section shows you the implementation of the Rubberband Line cursor, the Sketch cursor a "TV" cursor, and some grid-drawing routines. The examples are intended as a model for writing your own commands. The last section, "GRAFIN2 Subroutines, Variables, and Pointers", lists the predefined FORTH routines and variables used by GRAFIN2.

GRAFIN2 is written in the Metheus version of FORTH. Appendix C is a summary of Metheus FORTH. You should refer to the FORTH reference manuals listed in the Preface for detailed information about FORTH.

FORTH FUNDAMENTALS

FORTH is a stack based, interpreted language. FORTH evaluates expressions by placing values on the evaluation stack, applying operators to these values, then leaving the result on the stack or saving the result away. (The evaluation stack is very similar to the stack used by "Reverse Polish Notation" calculators.)

[1]FORTH is a registered trademark of FORTH, Inc.

Commands are built up from subroutines which are, in turn, built from the FORTH core words and Omega instructions. Data is pushed on the stack in the reverse order of execution; the stack is "last in, first out".

Definition of Terms

Table 3-1 contains terms used in the discussions and examples that follow:

Table 3-1. FORTH Terminology

stack	The FORTH evaluation stack. FORTH uses this stack for storing intermediate results. (FORTH also has a ***return stack*** used for subroutine nesting which GRAFIN2 does not use.)
TOS	Top Of Stack. The top word on the stack.
n1 n2 n3 n4...	One way to describe the stack: n1 is the TOS, n2 is the second word on the stack, n3 the word below that, et
"abcd..."	Another way to describe the stack: "abcd..." represents words on the stack. The right-most character (in this case, "d") is the TOS.
coordinate	Two numbers, X and Y, that represent a point on the screen. These must always be positive numbers. When pushed on the stack, X is always on top of Y.
LSB	Least Significant Bit (in some contexts, Byte).

Using the Stack

Figure 3-1 illustrates typical use of the stack. In this case, do A = B + C, defining the variables and leaving the result in A.

NOTE

In this and all examples, the numbers appearing in the left-most column are ONLY for the discussion of the example and are NOT part of the FORTH input.

```
1)      0 Var A     ( declare variable A )
2)      2 Var B     ( declare B )
3)      3 Var C     ( declare C )
4)      B @         ( move B to the stack )
5)      C @         ( move C to the stack )
6)      +           ( add B and C together )
7)      A !         ( save result in A )
```

Figure 3-1. Typical Stack Usage

Figure 3-1 illustrates a very simple stack operation. The first three lines allocate three variables and give them initial values of 0, 2, and 3. Lines 4 and 5 move B and C to the stack (C is the TOS). Line 6 adds them together, leaving the result on the stack, and line 7 saves the result in A.

NOTE

All keywords (tokens) are separated by spaces or tabs. In Metheus FORTH, tokens differ by length and by the first three characters. Case is not considered; lower case is converted to upper case. ("There" and "theory" differ in length, but "treat" and "trees" are considered identical.) Comments are within parentheses. Be sure to include the right parenthesis.

Extending FORTH With Subroutines

Because FORTH is a threaded, interpreted language, it is very easy to extend the language by defining new subroutines. Metheus provides some subroutines for use with GRAFIN2; these are discussed in the last section of this chapter. Once a routine is defined, it can be called by other routines the same way FORTH core words or previously-defined routines are called. (The Metheus FORTH "core words" are covered in Appendix C.)

Figure 3-2 shows a subroutine that finds the sum of the numbers 1 through 7.

```
1)   decimal    ( set base 10 )
2)   : sum7     ( define a routine named 'sum7' )
3)     0        ( init the accumulator )
4)     8 1 do   ( start at 1, count till we reach 8 )
5)       I      ( get the current iteration count )
6)        +     ( add it to our accumulator )
7)     loop     ( bump the iteration count, exit if 8 )
8)   ;          ( end definition of routine 'sum7' )
```

Figure 3-2. "Sum7" Subroutine

This summing example illustrates the construction of a simple subroutine. The first line sets the radix of the numbers, in this case, decimal. Line 2 begins the definition of the routine "sum7". Line 3 pushes a 0 on the stack; this will be our accumulator. Line 4 pushes the parameters of the loop onto the stack and begins the *Do ... Loop*. The *Do ... Loop* construction is used to iterate through a sequence of numbers. The parameters of the loop (1 and 8) are popped from the stack. The "I" and "+" in the fifth and sixth lines get the current iteration count and add it to the accumulator. Line 7 ends the *Do ... Loop*, and line 8 ends the definition of "sum7".

The routine "sum7" exits with the result left on the stack. We allocated no variables since the result would be left on the stack. The code in Figure 3-3 executes "sum7" and saves its result in the variable A (which was defined in a previous example):

```
sum7        ( execute the routine )
A !         ( save the result in A )
```

Figure 3-3. Executing the "Sum7" Routine

By defining this routine, we have now extended the language to include a function that returns the sum of the first seven numbers.

Implementing the MINMAX Routine

Figure 3-4 shows the implementation of a GRAFIN2 subroutine, **minmax**. **Minmax** sorts the top two values on the stack by size

NOTE

> In discussing this and the remaining examples, the GRAFIN2 subroutines are referenced in **boldface**, and the variables and pointers are referenced in *italics*. The last section of this chapter lists the GRAFIN2 subroutines, variables, and pointers.

```
1)  : minmax    ( define the routine )
2)    over over ( copy the two top items on the stack )
3)    >         ( compare the 2 numbers, replace with T/F )
4)    if        ( test if n2 is greater than n1 )
5)      swap    ( reverse the order )
6)    then      ( end of If statement )
7)  ;           ( exit with larger number on TOS)
```

Figure 3-4. Implementing the "Minmax" Routine

Minmax expects two numbers on the stack, and when it exits, the larger number is on the top of stack. The first line starts the definition of **minmax**. Line 2 copies the two numbers on the stack so that we now have four numbers on the stack. (If the values were "ab", the stack now contains "abab".) The ">" in line 3 compares the top two numbers and replaces them with a True/False flag: a 0 if n2 was less than n1, else a 1. (In FORTH, 0 is false, non-zero is true.) The stack now contains either "ab0" or "ab1". The "if" statement in line 4 pops the T/F flag left by the ">" operator. If the second number was bigger, line 5 swaps the two numbers left on the stack. Line 6 ends the "if" statement, and line 7 ends the definition of **minmax**.

Learning FORTH

Now that you have seen some simple examples of FORTH, you may want to try it yourself. There are two ways to learn FORTH on your Omega system. The first method disables the GRAFIN2 firmware and allows you to directly access the FORTH interpreter on the GRAFIN2 card. Routines developed this way are lost when you enable GRAFIN2 again. This method is intended only for learning FORTH.

The second method downloads routines to the FORTH interpreter with one of the GRAFIN2 FORTH commands: opcodes 50h, 51h, 52h, or 53h. (The GRAFIN2 FORTH commands, like all GRAFIN2 commands, must be preceded by opcode 4Ah.) Your routines are stored in the "user-dictionary".

NOTE

> The stack and the user-dictionary share approximately 1200 bytes of memory on the GRAFIN2 card. The stack and dictionary are located at opposite ends of memory and each "grows" toward the middle. If you define too many routines, the stack area may become so small that the system crashes.

METHOD ONE - HARDWARE DIRECT

The first method of learning FORTH requires you to remove the cover of your Omega and turn off switch 7 of the data-transfer switch on the GRAFIN2 card. (See Appendix A for instructions. Refer to Figure A-3 for the location of the switch.) Switch 7 OFF disables the normal GRAFIN2 firmware and allows you to communicate directly with the FORTH interpreter on the GRAFIN2 card. You can plug your terminal into tablet port J4 on the back of the Omega. Be sure the baud rate of your terminal matches the baud rate of the Omega.

With switch 7 OFF, all input from J4 goes directly to the FORTH interpreter. A "control C" will usually break you out of a FORTH program and return control to the interpreter. If your program completely crashes, you can regain control by turning the Omega off and back on. Be sure to turn switch 7 ON when you are done playing with FORTH, otherwise the Omega will not boot up.

CAUTION

> Be sure to replace the cover of the Omega after setting the switch or you may damage the circuit board. The Omega's cooling system requires the cover to be in place.

METHOD TWO - DOWNLOADING

You can download routines to the FORTH interpreter with the GRAFIN2 firmware. User-defined routines execute before the routines located in the firmware. Therefore, if you write a new version of an existing command, your command executes instead of the original. (The original is not affected).

Entering opcodes 50h, 51h, or 52h performs three functions: removes the cursor (if it was visible), sets the cursor type to crosshair, and opens a channel to the host. FORTH input is terminated by an escape (1Bh). Opcode 53h, Forget Words, is used to delete all user-defined entries from the FORTH dictionary. You can also use the FORTH "forget" routine to delete specific routines.

Table 3-2 summarizes the GRAFIN2 commands that call FORTH.

TABLE 3-2. – GRAFIN2 FORTH COMMANDS

Opcode	*Meaning*
50h	Comm mode. Used to download debugged code. The error count is reset to zero when FORTH is invoked. Nothing is displayed on the Omega screen.
51h	Error mode. Used to download code that may contain errors. The error count is reset to zero when FORTH is invoked. Errors are displayed on the Omega screen. When leaving FORTH, the error count is displayed.
52h	Text mode. Used to download code that may contain errors. The error count is reset to zero when FORTH is invoked. All FORTH code (except comments) is displayed on the Omega screen. When leaving FORTH, the error count is displayed.
53h	Forget Words. Used to delete all user-defined commands from the dictionary. (See also the FORTH "forget" routine.)

COMMAND EXAMPLES

This section contains examples that show the implementation of three cursors and some grid-drawing routines.

Writing Custom Cursors

In general, you need to create three routines to draw a custom cursor:

- o a cursor-erasing routine
- o a cursor-drawing routine
- o an installation routine

The three routines are linked by the pointer variables *ccdown* and *ccup*. The installation routine puts the address of the cursor-erasing routine in *ccdown* and the address of the cursor-drawing routine in *ccup*.

The routine **remove** uses the erasing routine pointed to by *ccdown* to erase the cursor. In addition, **remove** pushes the current cursor coordinates on the stack before using *ccdown*.

The routine **new** uses the drawing routine pointed to by *ccup* to draw the new cursor. **New** pushes the current cursor coordinates on the stack before using *ccup*.

The installation routine needs to erase any cursor that may currently be visible by calling **remove**, set *ccdown* and *ccup*, and draw the new cursor with the **new** routine.

```
1)   : U35                  ( GRAFIN2 names begin with 'U' )
2)     remove               ( removes a cursor if visible )
3)     " Draw ccup !        ( installs Draw )
4)     " Erase ccdown !     ( installs Erase )
5)     new                  ( draws the new cursor )
6)   ;                      ( end definition of U35 )
```

Figure 3-5. An Installation Routine

Figure 3-5 illustrates an installation routine that defines a new cursor as opcode 35h. (Assume that we have previously defined the "Draw" and "Erase" routines.) The first line starts the definition of the routine. By convention, all GRAFIN2 commands are named "U*nn*", where *nn* is a hex number. (To execute the command, you would enter "**4A***nn*".) **Remove**, in line 2, pushes the cursor's coordinates on the stack and removes the current cursor if one is visible. (At this point, *ccdown* contains the last cursor-erasing routine used.) Line 3 takes the address of Draw and stores it in the pointer variable *ccup*. Likewise, line 4 puts the address of Erase in *ccdown*.

New, in line 5 causes the new cursor to be drawn. When **new** executes, it places the current coordinates on the stack (X on top, Y as the second word), then calls the Draw routine. Likewise, **remove** places the current coordinates on the stack then calls the Erase routine.

After the Draw or Erase routine executes, the stack contains the Omega instructions placed on it by the routines. The instructions on the stack are executed ("unloaded") by the **sexec** (Stack EXECution) routine. The GRAFIN2 firmware automatically calls **new**, **remove**, and **sexec** at the 60Hz refresh rate until it receives another command.

Implementing the Rubberband Line Cursor

The Rubberband Line cursor is supplied in the GRAFIN2 firmware. The Rubberband Line cursor draws a complement vector between an anchor point and the cursor's location. Figure 3-6 shows the implementation of the Rubberband Line cursor.

The Rubberband Line command contains three subroutines: **anchor**, **rlcomp**, and **U32**. Before defining the subroutines, we set hex as the radix and define two variables, *xanchor* and *yanchor*. These variables hold the location of the anchor point for the complement vector. The routine **anchor** in line 4 reads an XY coordinate from the host and saves it in *xanchor* and *yanchor*.

Line 8 begins the definition of the **rlcomp** (Rubber Line COMPute) routine. **Rlcomp** pushes the data and opcodes needed to draw a complement mode vector from the anchor location to the current tablet position. **Rlcomp** is entered with the current XY cursor

```
1)  hex                  ( set hexadecimal format for numbers )

2)  0 var xanchor        ( define anchor points )
3)  0 var yanchor

4)  : anchor             ( routine to set anchor point from host data )
5)    wgethost xanchor !
6)    wgethost yanchor !
7)  ;

       ( routine to push the complement vector data to the stack )
8)  : rlcomp             ( begin definition of rubberline compute )
9)    0172               ( push compdr )
10)   rrot               ( get X to TOS, then Y )
11)   0752               ( push mov P1 + flag )
12)   yanchor @ xanchor @        ( get the anchor points )
13)   0753               ( push mov P2 + flag )
14)   68 0250            ( sets a solid line )
15) ;                          ( end definition of rlcomp )

                 ( rubber line cursor )
16) : U32
17)   remove
18)   " rlcomp dup ccdown ! ccup !  ( install new routine )
19)   anchor
20)   new
21) ;
```

Figure 3-6. Implementing the Rubberband Line Cursor

position on the stack.

Since we are pushing Omega instructions on the stack, the first actions are the last items pushed on the stack. For **rlcomp**, the last action is the complement draw. Thus the first Omega opcode that we push is the COMPDR instruction in line 9.

After pushing the COMPDR opcode, we move the XY on top of it using the RROT command. RROT rotates the top three stack entries so that n1 becomes n3. (If the stack contained "abc", it now contains "cab".) This leaves X on top of the stack with Y beneath it. Next we push the MOV P1 opcode (0752h) to the stack--its parameters are the X and Y already on the stack. In line 12, we retrieve the anchor points from *yanchor* and *xanchor* (the X coordinate will be on top), then push the MOV P2 instruction in line 13. Finally we push the Omega PATTERN instruction and data in line 14 to set a solid line.

The routine **U32** installs the rubber line cursor. In line 17 it calls **remove** to erase any currently drawn cursor. In line 18 it gets the address of **rlcomp** to the stack, duplicates it, then saves it in *ccup* and *ccdown*. It then calls **anchor** in line 19 to get the XY coordinates of the anchor location from the host computer. Finally, it calls **new** in line 20 to draw the new cursor.

Once the stack contains the instructions and data, the routines **sexec**, **new**, and **remove** are automatically invoked (at the 60Hz refresh rate). After this, the routine **rlcomp** is called automatically whenever the system needs to update the cursor position.

Stack Management With SEXEC

Since the stack uses 16-bit numbers, the upper byte of each Omega opcode on the stack tells **sexec** the location and size of the opcode's parameters. Note that only the low byte of the opcode is sent to the Omega. The upper byte is used for stack management.

If the upper byte is 0, the next word on the stack points to a block of data to be sent to the Omega. The first word of the block is the byte count (16 bits), and the following bytes are data.

If the upper byte is non-zero, the parameters for the opcode are on the stack. The upper byte specifies how many parameters are on the stack, which of them are bytes, and which of them are words. Only seven parameters can be passed in this manner.

The algorithm (in "C") for making the upper byte is:

```
int upper_byte ;
upper_byte = 1 ;
/* check all parameters for size */
for (i = number_params; i > 0; i--) {
      /* we have a parameter, shift upper_byte up 1 */
      upper_byte *= 2 ;
      /* now set the LSB to 1 if a word parameter */
      if (param[i] == WORD)
           upper_byte += 1 ;
}
```

Here is how **sexec** interprets the upper byte. When the upper byte equals 1, there are no parameters left on the stack. When the upper byte is greater than 1, the LSB of the upper byte represents the next parameter of the opcode. If it is 0, the parameter is a byte, if a 1 the parameter is a word. The firmware then performs a right shift and tests if the upper byte is greater than 1. When the upper byte is equal to 1, there are no more parameters for that opcode on the stack. Some typical upper bytes are:

Opcode	*Upper_Byte*	*Composite Opcode*
61h - Draw	01h	0161h
72h - Compdr	01h	0172h
52h - Mov P1	07h	0752h
4Eh - Mov Color	02h	024Eh
70h - Pixblt	0Bh	0B70h

The byte/word distinction is important. Since the stack is allocated in words, **sexec** must know which stack entries are byte parameters, which are words, and which are opcodes. The stack always pops words; when a byte is sent, the upper byte is thrown away. **Sexec** stops when it sees a 0 opcode. This 0 is preloaded at the bottom of the stack by the GRAFIN2 firmware.

Implementing the Sketch Cursor

The Sketch cursor (opcode 34h) leaves a trail behind the cursor during the time a button is pushed on the mouse. Figure 3-7 illustrates the implementation of the Sketch cursor.

Sketch contains two routines: **inkup** and **U34**. **U34** uses the **xhair_set** routine instead of **remove** to erase the previous cursor. To implement Sketch, we first set hexadecimal as the current radix and define the variables *xanchor* and *yanchor*.

Next we create a routine **inkup** that draws the cursor's trail in the current Omega drawing color while a button is pressed. **Inkup** is

```
hex                  ( set hexadecimal format for numbers )

0 var xanchor        ( initialize our variables )
0 var yanchor

        ( compute cursor routine - ink lines if button pressed )
: inkup
  cxhair             ( compute the Xhair cursor )
  cpos toscreen      ( get tablet and convert to screen coords )
  button if          ( get the button status )
                     ( if a button is pressed )
        0161 rrot    ( push draw; rotate XY to TOS )
        0753         ( mov P2 )
        2over yanchor @ swap yanchor !
                     ( store Y in yanchor )
        2over xanchor @ swap xanchor !
                     ( store X in xanchor )
        0752         ( mov P1 )
        68 0250      ( set solid line )
  else
        xanchor ! yanchor !     ( update last position )
  then
;

   ( sketch in current drawing color while a button is pressed )
: U34
  xhair_set          ( set crosshair cursor )
  " inkup ccup !     ( install new routine )
  new
;
```

Figure 3-7. Implementing the Sketch Cursor called with XY coordinates on the stack. Line 5 calls the routine **cxhair**. **Cxhair** replaces the coordinates with the appropriate Omega moving and drawing instructions to create a complemented crosshair cursor. The "cpos toscreen" in line 6 takes the current tablet position and converts it to screen coordinates. Line 7 checks the button status. If a button is being pressed, line 8 ("0161 rrot") pushes an Omega DRAW instruction, then rotates the top three stack entries so that the X and Y are back on the TOS.

Lines 8 through 13 draw a vector from the last cursor position to the current position. The "0753" in line 9 pushes MOV P2 to the stack; it will use the X and Y on the stack as data. Lines 10 and 11 move the previous cursor position to the stack and store the current position in *xanchor* and *yanchor*. Then in line 12, MOV P1 is pushed to the stack. The last line of the "if" statement pushes an Omega PATTERN instruction and data to set a solid line.

The "else" statement in line 14 updates the cursor's position when no button was pressed.

Routine **U34** installs the SKETCH cursor. **Xhair_set** removes the old cursor and sets *ccup* and *ccdown* to the address of **cxhair**. Line 20, " **"** inkup ccup **!** ", places the address of **inkup** in *ccup*. **New** pushes the current coordinates to the stack and calls the routine whose address is in *ccup* (**inkup**).

Implementing the TV Cursor

The "TV" cursor (opcode 35h) is not supplied with the standard GRAFIN2 firmware. The command draws a small rectangle on the screen to be the "TV receiver", and the cursor acts as a roving camera. The command performs a block transfer of the information the cursor "sees" to the "receiver" area. (It is helpful to have something on the Omega screen for the cursor to "see" when you invoke the TV cursor.)

The "TV" cursor defines two routines: **bltup** and **U35**. But first we set the current radix to hex and define two variables, *xsave* and *ysave*.

The **bltup** routine performs a block transfer of the information

```
1)  hex                      ( set hexadecimal format for numbers )

2)  0 var xsave              ( define our variables )
3)  0 var ysave

     ( routine to compute the cursor-drawing commands )
4)  : bltup
5)    over over              ( copy the X Y )
6)    xsave ! ysave !        ( save the X Y away for now )
7)    cxhair                 ( compute the cross hair )
8)    0 80 80 0B70           ( data for the pixblt )
9)    1FF dup 0753           ( mov P2, pixblt destination )
10)   ysave @ xsave @ 0752        ( mov P1, pixblt source )
11) ;

     ( install this as a new cursor, name it U35 )
12) : U35
13)   remove                 ( remove a cursor if visible )
14)   " bltup ccup !         ( install cursor draw routine )
15)   " cxhair ccdown !      ( install cursor erase routine )
16)   0163                   ( draw rectangle around destination )
17)   0FF 024E               ( set color 255 - white )
18)   1FE dup 0752           ( mov P1 )
19)   280 dup 0753           ( mov P2 )
20)   new                    ( draw a new cursor )
21) ;
```

Figure 3-8. Implementing the TV Cursor

"seen" by the cursor to the "receiver" area on the Omega screen. **Bltup** is called with an XY coordinate on the stack. Lines 5 and 6, ("over over" and "xsave ! ysave !"), copy the coordinates and save them away in the variables *xsave* and *ysave* for temporary storage. Line 7 calls **cxhair**, which replaces the XY with the commands needed to draw a complement mode cross-hair cursor on the stack. In line 8, we push the Omega PIXBLT instruction to the stack: the opcode and three parameters. Note the order that we push the parameters and opcode. We now push the MOV P2 instruction to the stack in line 9 with the fixed address of (1FFh,1FFh). This is the destination of PIXBLT instruction. Finally, in line 10, we retrieve the cursor position from *xsave* and *ysave* and use it with MOV P1 (the source of the PIXBLT).

When the stack is unloaded (via **sexec**), the commands sent to the Omega are:

1) MOV P1 to current tablet position
2) MOV P2 to (1FFh,1FFh)
3) PIXBLT a rectangle from P1 to P2
4) Draw a cross-hair cursor.

U35 is the installation routine. First, the cursor is removed from the screen by **remove** in line 13. We then install **bltup** as the cursor-drawing routine by placing the address of **bltup** in the variable *ccup*. When we examine **bltup** we see that it uses the cross-hair cursor routine **cxhair**. Because of this, we install **cxhair** as the cursor-erasing routine. Next, in line 16, we draw a box around the "TV receiver." Since the stack is unloaded in reverse order, the first thing we push is the Omega RECT1 instruction (outline box). We then set the drawing color to white (color 255). Next we push coordinates for the two corners of the box with the appropriate MOV P1 and MOV P2 instructions in lines 18 and 19. One corner is at (1FEh,1FEh) and the other is at (280h,280h). Finally, we call **new** in line 20 to draw the cursor the first time. **New** will call **bltup** to draw the cursor.

Before the next cursor is drawn, the previous cursor must be removed. Since the routine **cxhair** was installed as the erase routine, the GRAFIN2 firmware will automatically invoke **cxhair** after **bltup** so that the crosshair-cursor-erase instructions and data are on the TOS. Thus the previous cursor is erased before the next one is drawn since the stack is unloaded top to bottom.

Implementing Grid and Grid-Drawing Routines

The final command example, shown in Figure 3-9, contains three grid routines:

U18 -- Followed by one byte each of X spacing and Y spacing to set grid size. The cursor is snapped to the grid.

U19 -- Turns off snapping the cursor to the grid.

U81 -- Followed by one byte each of X spacing and Y spacing to set grid size. Draws a grid on the screen in the current Omega drawing color. (Cursor is not snapped to this grid).

The size of the grid (for either **U18** or **U81**) is given as one byte of X spacing and one byte of Y spacing, and stored in *xgrid* and *ygrid*, defined in lines 2 and 3.

"Snapping" the cursor to the grid means that the cursor moves in discrete jumps. The jump size is specified by the XY spacing.

Snapping uses a feature called filtering. One example of filtering is shown in the GRID routine in Figure 3-9. This routine intercepts the tablet data and modifies it before the GRAFIN2 firmware draws the cursor.

More importantly, you can write routines that are unrelated to graphics input! The routine U81, also shown in Figure 3-9, simply draws a grid on the screen in the currently selected drawing color The ability to download general-purpose routines to the FORTH interpreter offers an efficient alternative to host resident software.

```
hex                ( set hexadecimal format for numbers )

0 var xgrid        ( declare 2 variables to hold the grid spacing )
0 var ygrid


752 con p1         ( constants for use in grid drawing routine )
753 con p2
161 con draw


  ( filter the XY on the stack so that it is snapped to the grid )
: grid
  xgrid @ 2/ +          ( get 1/2 the xgrid value, add to X )
  dup xgrid @ mod -
  swap                  ( do the same for Y )
  ygrid @ 2/ +
  dup ygrid @ mod -
  swap                  ( put X on TOS )
;


  ( turn grid on )
: U18
  remove                ( remove the cursor )
  hget xgrid !          ( init the X and Y grid spacing )
  hget ygrid !
  '' grid uscaleat !
  ifon
;


  ( turn grid off )
: U19
  remove
  '' null uscaleat !
  ifon
;
```

Figure 3-9. Implementing Grid-Drawing Routines

```
     ( draw a grid on the screen in current color )

: U81
  hget xgrid !        ( init the X and Y grid spacing )
  hget ygrid !

  swidth @ 0 do          ( draw vertical lines )
       draw
       0 I p1
       sheight @ I p2
       sexec drop
  xgrid @ +loop

  sheight @ 0 do         ( draw horizontal lines )
       draw
       I 0 p1
       I swidth @ p2
       sexec drop
  ygrid @ +loop
;
```

Figure 3-9. (Cont.) Implementing Grid-Drawing Routines

GRAFIN2 SUBROUTINES, VARIABLES, AND POINTERS

This section contains subroutines, variables, and pointers used by GRAFIN2. Entries are arranged alphabetically within each grouping. Variables and pointers are referenced by *italics* and subroutines by **bold** text.

GRAFIN2 Subroutines

anchor

Reads an XY coordinate from the host and initializes the variables *xanchor* and *yanchor* with the values.

bget

Gets a byte from the Omega, fills leading zeros to 16 bits and returns that word as the TOS.

bsend
Sends the lower byte of the TOS word to the Omega.

button
Returns the current button ID on the TOS. This word is 0 if no buttons are pressed. The button ID format is tablet dependent.

clip
Clips the XY coordinates on the TOS so that it is within the current bounding box defined by *hiXclip*, *loXclip*, *hiYclip*, and *loYclip*.

cpos
Pushes the current tablet coordinates on the stack (X is TOS).

csend
Gets a byte from the Omega and puts it on the stack.

cxhair
Uses the XY coordinates on the TOS to produce the appropriate Omega move and draw instructions needed to create a complemented crosshair cursor. The XY are replaced on the stack by the Omega instructions and data.

event
Moves a button from the event queue to the stack. If the event queue is empty, **event** puts a -1 on the stack, otherwise **event** pushes a word with the button in the low byte (top of stack), followed by a word of X coordinate, then a word of Y coordinate. (The X and Y are in tablet coordinates).

hcoord
Gets an XY coordinate from the host computer and places it on the stack. The host must send four bytes: low_x, hi_x, low_y, hi_y. The Omega must be ready to accept an opcode.

hget
Gets a byte from the host computer, fills leading zeros to 16 bits, and returns that word as the TOS. The Omega must be ready to accept an opcode.

hput
Sends the low byte of the TOS to the host computer, discarding the high byte. The Omega must be ready to accept an opcode.

ifon

Redraws a cursor on the screen if **remove** erased one. **Ifon** calls **new** to do this.

init

Removes the cursor, then resets all variables to their power-up state. (See INIT GRAFIN2, opcode 10h, in Chapter 2.) User-defined routines are not affected.

minmax

Sorts the top two numbers on the stack so that the larger is on the TOS and the smaller is beneath it.

new

Checks if a cursor is displayed on the screen. If not, **new** pushes the current XY coordinates on the stack. It then computes the new cursor-drawing data and instructions by calling the routine whose address is in *ccup*. **New** leaves the Omega data and instructions on the stack. The stack contents are later automatically sent to the Omega by **sexec**.

null

Does nothing and is the default routine pointed to by *uscaleat*.

ppad

Sends the low byte from the TOS to the tablet devices.

qflush

Discards any data in the event queue.

remove

Checks if a cursor is displayed on the screen. If so, **remove** pushes the cursor's XY position to the stack. It then removes the cursor by calling the routine whose address is in *ccdown*. **Remove** then invokes **sexec** to send the instructions and data stored on the stack to the Omega.

sexec

Pops instructions and data from the stack and sends them to the Omega until a zero opcode is found. **Sexec** leaves the zero on the stack and also returns a garbage word on TOS. **Sexec** is invoked automatically at 60Hz.

size_init

Uses the top two words on the stack to reset values used in tablet-to-screen transformations and screen clipping. The TOS is the screen width, the word beneath it is the screen height.

Size_init sets these variables:

yoffset	*twidth*
xoffset	*theight*
yscale	*hiXclip*
xscale	*loXclip*
swidth	*hiYclip*
sheight	*loYclip*

toscreen
Converts the XY coordinates on the TOS from tablet coordinates to screen coordinates, using the current offsets and scales. After conversion, a user-defined filter can be applied by the routine pointed to by *uscaleat*. Finally, the position is clipped or wrapped by invoking the routine whose address is in *clpwrp*.

wgethost
Gets a word from the host computer and puts it on the stack. The Omega must be ready to accept an opcode. If the host does not send a word, the Omega will hang and need to be reset.

wputhost
Sends the word on the TOS to the host computer. The Omega must be ready to accept an opcode.

wrap
Wraps the XY coordinates on the TOS so that the point is within the current bounding box defined by *hiXclip*, *loXclip*, *hiYclip*, and *loYclip*.

wsend
Sends the top word on the stack to the Omega.

xfinit
Uses the variables *swidth* and *twidth* to calculate *xscale* and *xoffset*, and *sheight* and *theight* to calculate *yoffset* and *yscale*. The scale and offset values are used to convert tablet-to-screen coordinates so that the full tablet size is mapped to the full screen. (See SET OFFSET/SCALE, opcode 20h, in Chapter 2.)

xhair_set
Removes the cursor and installs the crosshair cursor as the current cursor type by setting *ccup* and *ccdown* to the address of **cxhair**.

Scaler Variables

cheight, cwidth

Two variables containing the height and width of the current bounding box as set by SET WRAP MODE and SET CLIP MODE (opcodes 21h and 22h). Also see **size_init**.

ecount

Tells how many button events are stored in the event queue (0 means empty).

hiXclip, hiYclip, loXclip, loYclip

Four variables that define the bounding box for the clip and wrap routines. *HiXclip* > *loXclip* and *hiYclip* > *loYclip*, or you will get unpredictable results. Also, *cwidth* and *cheight* must be set so that they agree with the bounding box. These variables are set by SET WRAP MODE and SET CLIP MODE (opcodes 21h and 22h), and are also set by **size_init**.

swidth, sheight

Two variables containing the screen's width and height. They are used in computing the tablet-to-screen conversion. They are initially set to 1024 by 768. If this is not your screen size, use SET SCREEN SIZE (opcode 11h) to put the correct values into these variables. See also routines **size_init** and **xfinit**.

syncon

Enables an Omega SYNC instruction between cursor draw and erase when this variable is set to 1. (Most cursors look better with this variable set to 1).

trig

Selects the trigger mode when a button is put in the event queue. See SET Q MODE, opcode 41h.

Trigger Flag	*Mode*
0	Level Mode
1	Leading Edge
2	Trailing Edge
3	Both Edges

twidth, theight

Two variables containing the tablet's width and height. They are used by **xfinit** in calculating the tablet-to-screen conversion. They are initially set to 2200 by 2200 (Summa bit pad size). If this is not your tablet size, you can use SET TABLET SIZE (opcode 12h) to put the correct values into these variables. If you change these values, you must call **xfinit**.

xanchor, *yanchor*

Two variables used only inside the cursor tracking routines. If you write a custom cursor, you can use these for temporary storage.

xscale, *xoffset*, *yscale*, *yoffset*

Four variables containing the scale and offset values used to transform the tablet coordinates to screen coordinates. (See the SET OFFSET/SCALE (20h) command in Chapter 2 for their format.) Also see **size_init**

Pointer Variables

These variables point to routines to execute.

ccup, *ccdown*

These variables are set with the addresses of the routines that draw and erase the cursor. Each routine is called with the current cursor position on the stack (X on top). Each routine replaces the coordinates with a set of Omega instructions. If you need to install a routine that does nothing in one of these variables, remember that the routine must drop two words (the X and Y) from the stack.

clpwrp

Contains the address of either the clip routine or the wrap routine. You would typically set it by:

```
'' wrap clpwrp !
    or
'' clip clpwrp !
```

scaleat

Points to the routine that converts tablet coordinates to screen coordinates for button-reporting. GRAFIN2 normally sets this to '' **null** (for tablet coordinates) or '' **toscreen** (for screen coordinates).

uscaleat

Points to a user-defined position-scaling routine. The routine is called after the current position is converted from tablet to screen coordinates. After the routine is called, the result is clipped or wrapped. This variable is normally set to '' **null** and can only be changed by a user-created GRAFIN2 command. The routine *uscaleat* points to is called with XY on the stack, and must return with an XY on the stack. Also see **toscreen**.

APPENDIX A

GRAFIN2 INSTALLATION

This chapter explains how to install GRAFIN2 in your Omega system. The GRAFIN2 update package includes the GRAFIN2 board and eight new PROMs for the Omega display controller board. Specifically, the instructions in this chapter describe:

- o Position of the GRAFIN2 board
- o Replacing the microcode PROMs in the Omega display controller board
- o Selecting GRAFIN2 data rates
- o Connecting the graphics tablet

CAUTION

Only qualified service personnel should attempt any procedure where the covers must be removed. Read and follow the installation instructions carefully. Failure to properly install the GRAFIN2 board or microcode PROMs could result in improper operation or equipment damage to your Omega system.

(1) Unplug the Omega from the AC power source.

(2) Remove the top cover by removing the screws that hold it in place.

(3) Remove the interface board (and if you are upgrading from GRAFIN, also remove the GRAFIN board.) Figure A-1 illustrates the position of the interface and GRAFIN2/GRAFIN boards.

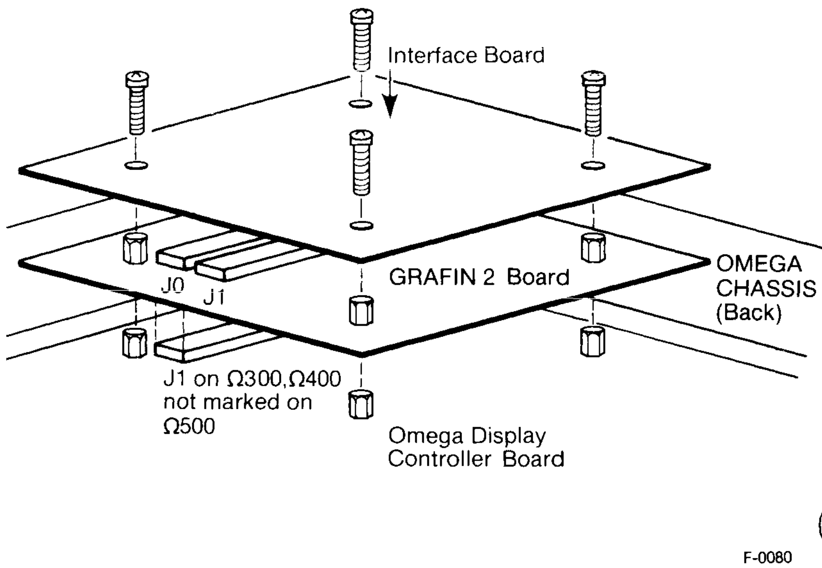

Figure A-1. Position of the Interface and GRAFIN2 Boards

(4) Replace the Omega microcode PROMs. Figure A-2 illustrates the location of the microcode PROMs for the Ω300/Ω400 and for the Ω500. Note the arrangement of the PROMs.

WARNING

Be sure that each PROM is in the correct socket and is correctly oriented. Failure to install the PROMs correctly may damage the PROMs and result in improper operation.

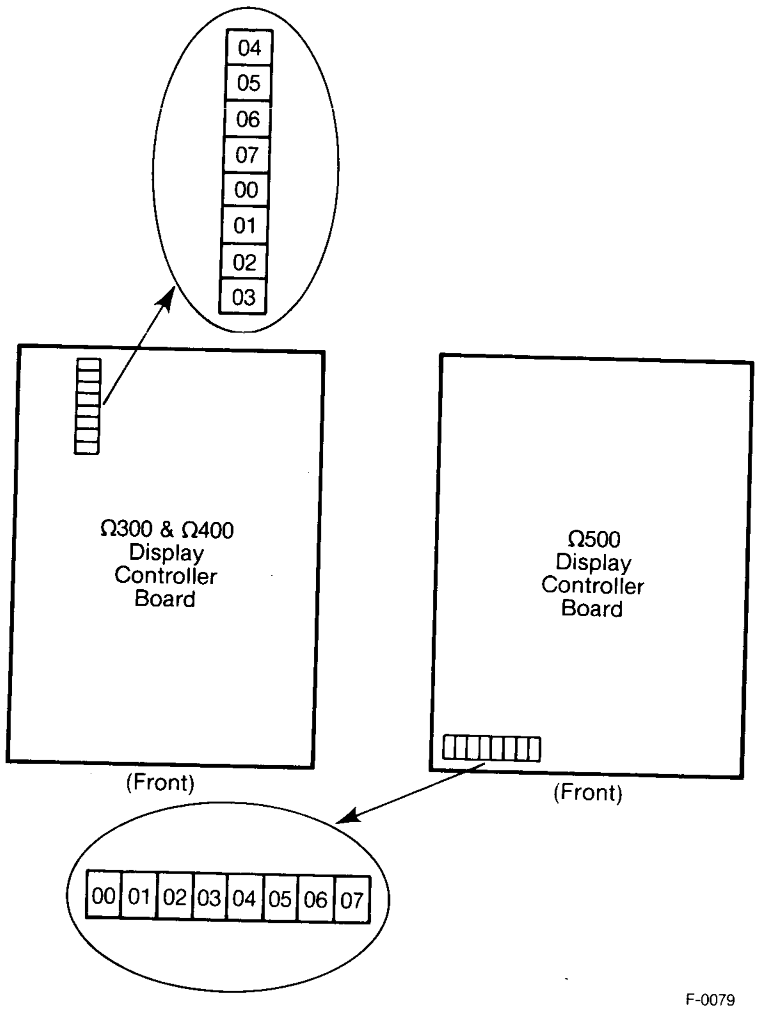

Figure A-2. Installing the Omega Microcode PROMs

(5) Select GRAFIN2 Data Rates. GRAFIN2 interface boards are set at the factory for 9600 baud. If you need to change this setting, follow these instructions. Find the 8-position switch near the center of the GRAFIN2 board. Four positions, numbered 1 through 4 in Figure A-3, select the transfer rates. This figure shows the default switch positions selecting 9600 baud.

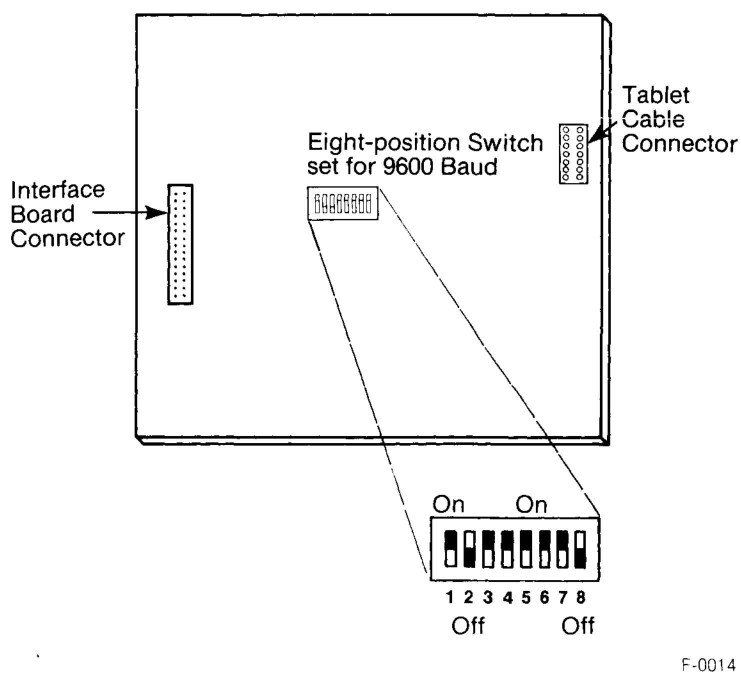

Figure A-3. The GRAFIN2 Data-Transfer Switch

(6) Select the new baud rate from Table A-1 and set the switches accordingly.

Table A-1. Data Rate Selection

Baud Rate	*Switch Settings* 1	2	3	4
unused	1	1	1	1
19.2K	0	1	1	1
9600	1	0	1	1
7200	0	0	1	1
4800	1	1	0	1
3600	0	1	0	1
2400	1	0	0	1
1800	0	0	0	1
1200	1	1	1	0
600	0	1	1	0
300	1	0	1	0
150	0	0	1	0
134.5	1	1	0	0
110	0	1	0	0
75	1	0	0	0
50	0	0	0	0

NOTE: 0=Off, 1=On

(7) Switch 5 determines resolution format. When set in the 0 (off) position, the GTCO binary high-resolution format is selected. The 1 (on) position selects the Summagraphics Bit-Pad format.

(Refer to the tablet manual for compatibility information.) The factory setting is 1, selecting the Summagraphics Bit-Pad format.

(8) Switches 6, 7, and 8 are reserved for future use and should remain as illustrated: switches 6 and 7 are on, and switch 8 is off.

(9) After changing switch positions, replace boards as in Figure A-1. Replace the top cover and the power cord.

(10)
Connect the input device to the Omega. Attach the input device to socket J4 on the back of the Omega. The pin arrangement of the RS-232 connector is illustrated in Figure A-4.

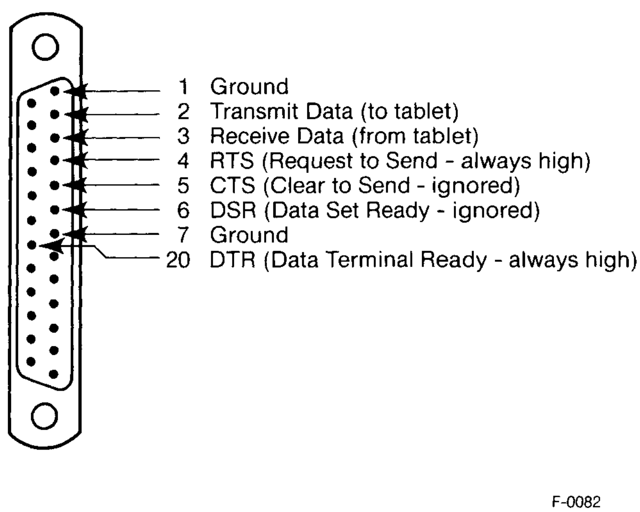

Figure A-4. RS-232 Pin Configuration for GRAFIN2

Appendix B

GRAFIN AND GRAFIN2 DIFFERENCES

GRAFIN2 differs from GRAFIN in the following ways:

(1) GRAFIN2 stores button hits in the button event queue. GRAFIN sends button hits to the host immediately (no event queue).

(2) GRAFIN2 returns six bytes of information about button hits; GRAFIN returns five.

(3) GRAFIN2 allows a choice of four styles of cursor, plus varying the size of the default crosshair cursor. GRAFIN supports only the crosshair cursor.

(4) GRAFIN2 keeps the cursor on the screen unless taken down explicitly with the INIT GRAFIN2 or CURSOR OFF commands. GRAFIN takes down the cursor whenever a command is executed.

(5) GRAFIN2 allows you to specify a bounding box for clip and wrap modes. GRAFIN sets a default bounding box according to the hardware.

APPENDIX C

METHEUS FORTH

This appendix summarizes the Metheus version of FORTH used in GRAFIN2. Refer to a FORTH reference manual for more information about FORTH.

The following terms are used throughout the appendix:

TOS	The top word on the FORTH Evaluation stack. Each word on the stack is 16 bits wide. When a byte is popped or pushed on the stack, only the lower 8 bits of the word are used.
n1 n2 n3 n4	These are used to refer to words on the stack: n1 is the TOS, n2 is just below it, etc.
LSB	Least Significant Bit (or Byte).
MSB	Most Significant Bit (or Byte).

SYNTAX CHANGES

The following are changes from the normal FORTH syntax.

VAR

FORTH uses *VARIABLE*. Metheus FORTH requires initialized variables. For example:

10 VAR XPOS

CVAR

Just like *VAR* except it allocates a byte of storage instead of a word. For example:

6 CVAR COUNT

CON

Standard FORTH uses *CONSTANT*, Metheus uses *CON*. For example:

752 CON P1

CCON

Just like CON except it allocates a byte of storage instead of a word. For example:

1 CON ONE

GENERAL PURPOSE ADDITIONS

These additions are of general interest and are not related to Omega I/O or GRAFIN2 functions.

CASE

Begin a case statement. The TOS is popped and used as an index into the case. Note that each case target must be EXACTLY one word, else it dies. This means you shouldn't use literals in the case statement.

Stack I/O: pop 1 word

CEND

End a case statement. After the selected case entry is executed, the program will resume after the CEND statement.

Stack I/O: none

LSHIFT

This word pops two items off the stack: TOS is a left shift count, n2 the word to shift. The second word is shifted left the number of places specified by the first word, then pushed back on the stack.

Stack I/O: pop two words, push 1 word

RSHIFT

This word pops two items off the stack: TOS is a right shift count, n2 the word to shift. The second word is shifted right the number of places specified by the first word, then pushed back on the stack.

Stack I/O: pop two words, push 1 word

D* A double precision signed multiply. The top word is multiplied by the second word and the 32-bit result is pushed back to the stack so that the low 16 bits are on the top of stack.

Stack I/O: pop two words, push two words.

ERROR

This command returns the address of the error flag word variable. The lower byte of the error flag contains the last error code, the upper byte contains the number of errors since the flag was reset. The user is responsible for resetting this flag

SUMMARY TABLES

The following 10 tables summarize most of the keywords, primitives, and operators found in Metheus FORTH.

Table C-1. Looping and Conditional Primitives

Structure	*Description*
BEGIN ... AGAIN	Infinite loop.
BEGIN ... (flag) UNTIL	Loop until flag on stack is true.
BEGIN ... (flag) WHILE ... REPEAT	Exit loop when flag is false.
IF ... THEN	If TOS is true, execute inner code.
IF ... ELSE ... THEN	If TOS is true, execute IF code, otherwise ELSE code.
DO ... LOOP	Iteration loop, uses words for counters.
DO ... (cnt) +LOOP	Ditto, increment by cnt.
CDO ... CLOOP	Iteration loop, uses bytes for counters.
CDO ... (cnt) C+LOOP	Ditto, increment by cnt.
DO ... LEAVE ... LOOP	Do loop, but LEAVE forces exit.
CDO ... CLEAVE ... CLOOP	CDO loop, but CLEAVE forces exit.

Table C-2. Arithmetic and Logical Words

Name	*Function*
*	Signed multiply n2 by n1, return 16-bit word on stack.
+	16-bit add n1 and n2, replace both with 16-bit result on stack.
-	16-bit subtract n2 from n1, replace both with 16-bit result on stack.
/	Divide n1 by n2, replace with 16-bit result on stack.
*/	Given n1, n2, and n3 on the stack, perform (n2 * n3) / n1 and leave the 16-bit result on the stack.
/MOD	Divide n2 by n1. Replace both by a 16-bit quotient as n2, and a 16-bit remainder as TOS.
1+	Increment TOS by 1.
2+	Increment TOS by 2.
1-	Decrement TOS by 1.
2-	Decrement TOS by 2.
2*	Multiply TOS by 2.
2/	Divide TOS by 2.
ABS	Absolute value of TOS.
AND	Bit AND.
D*	Signed multiply n2 by TOS. Replaces both with a 32-bit signed number; the 16 LSBs in TOS.
INTEGER	Convert from 32-bit fixed-point to integer. n2 contains the 16 MSBs, the TOS is the 16 LSBs. The 32-bit number is divided by 4096 and a 16-bit number is returned.
IOR	Bit inclusive-OR.
LSHIFT	Arithmetic left shift of n2 by the number of bits specified in the TOS word.
MAX	Compare n1 and n2; larger value becomes TOS.
MIN	Compare n1 and n2; smaller value becomes TOS.
MOD	Signed divide n2 by the low byte of the TOS. Replace both words by the 8-bit remainder expanded to 16 bits.
NEGATE	2's complement the TOS word.
RSHIFT	Arithmetic right shift n2 by the number of bits specified in the TOS word.
XOR	Bit exclusive-OR.

Table C-3. Compiler Directives*

Name	*Function*
+LOOP	Terminate a DO loop. The number on TOS is added to the current iteration count. If the result is >= the final value, stop.
AGAIN	End token for a BEGIN...AGAIN infinite loop.
BEGIN	Start of a BEGIN loop.
CASE	Start of CASE statement. The word on TOS indexes into the case-statement. Each word in the case-statement must be exactly 1 word, no literals allowed.
CEND	End of CASE statement.
CLOOP	Terminate a CDO loop. The low byte on TOS is added to the current iteration count. If the result is >= the final value, stop.
DO	Start of a DO loop.
ELSE	Middle of an IF...ELSE...THEN construct.
IF	Start of the IF construct.
LEAVE	Exit a DO loop.
LOOP	End of a DO loop. 1 is added to the current iteration count.
REPEAT	End of a BEGIN...WHILE...REPEAT loop.
THEN	End of a IF construct.
UNTIL	End of a BEGIN...UNTIL.
WHILE	Middle of a BEGIN...WHILE...REPEAT
[	Start a ASCII string.
]	End an ASCII string.
''	(two single quote marks together) Move dictionary address of next token to TOS.

*NOTE: These commands only work if you are in compile mode. They cannot be executed in command mode.

Table C-4. Defining Words

Name	*Function*
:	Begin dictionary definition.
;	End dictionary definition.
CON	Assign a constant (word).
CCON	Assign a constant (byte).
CVAR	Assign a variable (byte).
VAR	Assign a variable (word).

Table C-5. I/O Words (Not available to GRAFIN2 application code)

Name	*Function*
?	Prints the 16-bit number whose address is the TOS
C?	Prints the 8-bit number whose address is the TOS.
ASCII	Converts low byte of TOS to ASCII.
CRET	Send a carriage return/line-feed.
DISPLAY	Prints low bytes from successive TOSs until low byte is non-ASCII (>=80h).
ECHO	Prints low byte of TOS.
KEY	Pushes byte from terminal to TOS.
MSG	Type out a prompt based on the current error flag.

Table C-6. Memory References

Name	*Function*
!	Save n2 at address n1.
+!	Add n2 to variable whose address is n1.
0SET	Set variable whose address is n1 to 0.
1SET	Set variable whose address is n1 to 1.
@	Move variable whose address is n1 to n1.
C!	Set the character variable whose address is n1 to n2.
C+!	Add n2 to the character variable whose address is n1.
C@	Move the character variable whose address is n1 to n1.
C0SET	Set the character variable whose address is n1 to 0.
C1SET	Set the character variable whose address is n1 to 1.

Table C-7. Relational Operators

Name	*Function*
0<	Compare TOS to zero, <0 sets true on TOS, else false.
0=	Compare TOS to zero, =0 sets true on TOS, else false.
<	Compare top two words on stack, replaces words with true/false.
=	Compare top two words on stack, replaces words with true/false.
>	Compare top two words on stack, replaces words with true/false.

Table C-8. Stack Words

Name	*Function*
0	Push a 0 to the stack.
0STACK	Reset the stack pointer to power up value.
1	Push a 1 to the stack.
2	Push a 2 to the stack.
2DUP	Copy n1 to TOS twice.
2OVER	Copy n3 to TOS.
2SWAP	Swap TOS and n3.
ASPACE	Push a space character to the stack (20 hex).
CJOIN	Merge high byte of n2 and low byte of n1.
CSPLIT	Split TOS so high byte is in n2, low in n1.
DROP	Drop the TOS.
DUP	Duplicate the TOS.
H	Get value of previous iteration counter to stack.
I	Get current iteration counter value of inner loop to stack.
J	Get iteration counter value of second outer loop to stack.
K	Get iteration counter value of third outer loop to stack.
LROT	Rotate top three stack entries so that n3 becomes TOS.
OVER	Copy n2 to TOS.
RROT	Rotate top three stack entries so that n2 becomes TOS
SWAP	Swap n1 and n2.

Table C-9. System Words

Name	*Function*
'	(tick mark); In interpret mode, move dictionary address of next token to TOS.
,	(comma); Put 16-bit value into next two dictionary bytes.
ABORT	Warm start.
BINARY	Set input radix to binary.
DECIMAL	Set input radix to decimal.
DO.	Output a signed double-precision value to terminal.
EXECUTE	Jumps to address at TOS.
FORGET	Forget user definitions after this one, inclusive.
HERE	Push address of next free dictionary entry to TOS.
HEX	Set input radix to hexadecimal.

Table C-10. System Variables

Name	*Function*
BASE	Current input/output radix.
DP	Current dictionary pointer.

Deletions

These words are not used by GRAFIN2 and are unsupported. They may or may not work with Metheus FORTH. Refer to a FORTH reference book for their explanation.

<BUILDS
;CODE
CREATE
<R
C<R
#
#>
#S
+SP
-SP
?RS
?SP
CA!
END,
ENTRY
COMPILER
CONTEXT
CORE
DOES>
VOCABULARY
STATE
R>
CR>
<#
SIGN
TYPE
IMMEDIATE
INLINE
NUMBER
SEARCH
SINGLE
TOKEN
CURRENT
LBP
MODE