

INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.
2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.
3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of "sectioning" the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.
4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.
5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.

**University
Microfilms
International**

300 N. Zeeb Road
Ann Arbor, MI 48106

Order Number 1332237

Graphics terminal emulation on the PC

Noll, Noland LeRoy, M.S.
The University of Arizona, 1987

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

GRAPHICS TERMINAL EMULATION ON THE PC

by

Noland LeRoy Noll

A Thesis Submitted to the Faculty of the
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

MASTER OF SCIENCE
WITH A MAJOR IN ELECTRICAL ENGINEERING

In the Graduate College

THE UNIVERSITY OF ARIZONA

1 9 8 7

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: Noland L. Noll

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

Theodore L. Williams

T. L. Williams

Associate Professor of
Electrical and Computer Engineering

December 7, 1987

Date

ACKNOWLEDGMENT

I would like to thank Dr. D. L. Shirer and Dr. T. L. Williams for being my project advisors. I would also like to thank the faculty of the University of Arizona who have taught the courses I have taken. I am thankful for the suggestions offered by W. R. Stearns and I would like to thank H. Yu for providing me with an example thesis and loaning me his "C" compiler. I extend my gratitude to Hughes Aircraft Company for their support through the Hughes Masters Fellowship Program. I am also very grateful to my supervision at Hughes for their understanding and patience throughout the course of my graduate work and for allowing me to use departmental computers. I give special thanks to God for His help in getting me through this phase of my life.

TABLE OF CONTENTS

	<u>PAGE</u>
LIST OF ILLUSTRATIONS.....	6
LIST OF TABLES.....	7
ABSTRACT.....	8
1 INTRODUCTION.....	9
2 APPROACH.....	12
2.1 Introduction.....	12
2.2 Programming guidelines.....	12
2.3 Alternative I/O methods.....	13
2.4 Development.....	14
2.5 Debugging.....	17
2.6 Demonstration.....	19

TABLE OF CONTENTS CONTINUED

	<u>PAGE</u>
3 IMPLEMENTATION.....	20
3.1 Introduction.....	20
3.2 Main control level.....	23
3.3 Command interpretation level.....	24
3.4 String processing level.....	25
3.5 High level I/O.....	27
3.6 Mid level I/O.....	28
3.7 Low level I/O.....	29
3.8 High level screen functions.....	33
3.9 Mid level screen functions.....	36
3.10 Low level screen functions.....	37
3.11 BIOS interface level.....	44
3.12 Header files.....	45
4 STARBASE DEMONSTRATION.....	47
5 SUMMARY.....	60
APPENDICES.....	61
A Users' Manual.....	61
B Programmers' Reference.....	68
LIST OF REFERENCES.....	77

LIST OF ILLUSTRATIONS

<u>FIGURE</u>		<u>PAGE</u>
1	Plot Program Output.....	52
2	Picture Program Output.....	53
3	Interactive Program Output.....	54
4	Debug File Output for Interactive Program.....	55

LIST OF TABLES

<u>TABLE</u>		<u>PAGE</u>
1	Organization of Modules.....	21
2	Header Files.....	22
3	Graphics Control Sequences.....	71
4	Display Control.....	72
5	Vector Drawing Mode.....	73
6	Plotting Commands.....	75
7	Graphics Status.....	76

ABSTRACT

The HP2623 graphics terminal emulator is implemented on the PC for use with the Starbase graphics package provided on the departmental HP9000 series 500 computer system. This paper discusses the development and implementation of this emulator. A demonstration of its compatibility with Starbase is also provided along with a users' manual and a programmers' reference.

CHAPTER 1

INTRODUCTION

The ECE department has on hand a Hewlett-Packard HP9000 series 500 computer running an HP-UX Unix operating system [1]. This system includes several graphics software packages but there are no graphics terminals connected to it. However, there are several IBM PCs and PC compatibles (namely, the AT&T) available. The thought comes to mind that it should be possible to connect a PC to the HP9000 and emulate a graphics terminal on the PC under the DOS operating system [2]. The thesis project culminating in this paper develops an emulator to do just that. This paper discusses the approach to the emulator development as well as its implementation.

A preliminary investigation reveals three graphics software packages available on the HP9000: Starbase [3], AGP (Advanced Graphics Package) [4], and DGL (Device-independent Graphics Library) [5]. These packages support both raster devices and tty devices. The raster devices all require a special interface card be plugged into the HP9000 backplane. However, the tty devices can be emulated by interfacing an HP9000 tty line to the serial port on the PC, a much simpler approach. The only tty devices supported are the HP2623 Monochrome Graphics Terminal [6] and the HP2627 Color

Graphics Terminal [7]. Since it is simpler to emulate a monochrome device, the HP2623 is the initial target for emulation. The capabilities of the emulator can later be expanded to emulate the HP2627.

Hewlett-Packard sales engineer Larry Littlefield was contacted for additional information. He indicated that development around the Starbase graphics package is preferable since the other two packages are being phased out at HP. He also suggested a department at Hughes where the reference manuals for the HP2623 [6] and the HP2627 [7] could be borrowed. These manuals specify the control sequences a graphics emulator must implement.

Larry also revealed the existence of third party software written by Reflection which emulates graphics terminals on the PC; one of these is probably for the HP2623. Of course, purchasing this software would make the thesis project trivial.

As an alternative to emulating the HP2623, Larry mentioned that any graphics terminal could be implemented if the appropriate Starbase terminal driver were written using the Starbase Driver Development Guide [8]. This approach is unappealing because of the additional effort required in writing a new terminal driver, assuming the effort in producing the alternative terminal emulator is comparable.

The next chapter discusses the approach used to develop the HP2623 Emulator. This includes discussions on the establishment of programming guidelines and on the use of simplifying assumptions. Chapter 3 delves into the implementation of the emulator, including organization and module by module descriptions. The module descriptions discuss the purpose and capabilities of each module, including comments on the data structures involved, test methods, and development histories. Chapter 4 deals with demonstrating Starbase compatibility with the emulator. This chapter includes several illustrations exhibiting Starbase output as displayed by the emulator on the PC. Chapter 5 provides the summary and the appendix presents the users' manual and the programmers' reference (a definition of all the control sequences understood by the emulator).

CHAPTER 2

APPROACH

2.1 Introduction

This chapter presents the "recipe" for building a graphics emulator. The programming guidelines and alternative I/O methods are discussed followed by discussions on the development, debug, and demonstration of the emulator. As indicated in Chapter 1, the HP2623 Monochrome Graphics Terminal is the target for emulation on an IBM PC or PC compatible. This emulator must perform correctly when driven by a Starbase application program on the HP9000 utilizing the HP2623 driver over a tty line.

2.2 Programming guidelines

The first consideration is establishing programming guidelines and choosing a programming language. The "C" programming language [9] is used because it is a high-level structured language which allows low level bit-fiddling when necessary. Bit-fiddling is required when accessing individual pixels in display memory, for example. The Datalight "C" Compiler [10] is used because it is readily available and has a reputation for producing fast code.

The use of modularity is paramount in all program development. All code concerning a single aspect of the emulator is collected in a single module. The design effort

can then be focused on this one aspect, making the relevant issues much clearer. Furthermore, the module can be tested and debugged independently, eliminating the code from suspicion when debugging other modules depending on this one. Finally, when debugging the complete program, the type of bug demonstrated will isolate suspicion to the module handling the related aspect of the program.

Good documentation practices are also utilized. The most important of these practices is starting each module with a short description followed by a function by function description. The function descriptions include descriptions and types of the arguments and the return values. Good documentation practices ensure effective integration of the modules by minimizing duplication and time wasted in deciphering what capabilities are offered by each module. Program modification and debugging are also made much easier by following these practices.

2.3 Alternative I/O methods

The next issue concerns the handling of I/O on the serial line. Interrupt based I/O provides the most throughput and the smoothest display. However, interrupt based I/O must be implemented in assembly language and depends on the PC never disabling interrupts for a time longer than it takes to respond to consecutive characters

arriving at 9600 baud. Polled I/O wastes a lot of time checking the serial line, especially when verifying response to XOFF. This leads to reduction of throughput and jerky development of the display. However, polled I/O can be implemented entirely in "C" and is not be subject to interrupts being disabled. Considering that an assembler for the PC is not readily available, that polled I/O is usually much easier to implement and debug, and that disk I/O (which is expected to be concurrent with the serial line I/O for spooling and debugging purposes) could conceivably disable the interrupts long enough for characters arriving at 9600 baud to be missed, polled I/O is the better choice. Polled I/O performance is acceptable for the target installation since the time required to verify response to XOFF, approximately the round trip time of a character, is small for a PC directly connected to a tty line from the HP9000. Performance degradation is more significant when a modem and/or the SYTEK network is included in the connection.

2.4 Development

Considering that the key to the success of the emulator is the ability of its lowest level functions to perform, these modules containing these functions are developed first. These modules provide the keyboard interface, the serial line interface, the ability to set the screen mode,

and the line drawing capability. The keyboard and serial line modules can be linked together to produce a "dumb" terminal, a device capable of sending typed characters and displaying received characters on the screen. Testing this dumb terminal provides a test for both modules. The first step in designing these modules is to study the capabilities of the Datalight "C" Library and the interface it provides to DOS functions [11] and the BIOS routines [12].

Examination of the display memory layout as related to screen modes described in the IBM Technical Reference [12] is also required. Additionally, it is necessary to study the DOS functions and the BIOS routines to determine which ones are applicable. BIOS "INT 10" provides the video interface while BIOS "INT 14" provides the serial line interface. DOS functions 3 and 4 provide input and output to and from the serial line and DOS function 44H (hex) otherwise known as IOCTL can be used as a non-blocking check to see if a character is present on the serial line. See the appropriate implementation section for details.

The next phase of development is determining the set of capabilities implemented by the emulator. This involves comparing the capabilities of the HP2623 terminal and the expectations of the Starbase HP2623 driver. This allows elimination of unnecessary or redundant capabilities of the

HP2623 terminal from the emulator. For example, it is possible to eliminate graphics character generation at the emulator level since Starbase is capable of generating characters in terms of plotting commands. Not only does this save a lot of programming effort, but the characters generated by Starbase do not have the size and orientation limitations of the HP2623 terminal. Since user defined area patterns are not supported by Starbase, these are also eliminated. On the other hand, some HP2623 terminal capabilities which are not Starbase supported are easily implemented and significantly enhance the capabilities of the emulator. These capabilities can be accessed from Starbase by providing the necessary control sequence to the "gescape" (graphics escape) function. Such capabilities include drawing modes and line types in addition to those normally supported by Starbase.

In addition to the emulator capabilities required to support Starbase, the capabilities of the operator interface must be determined. This includes a preliminary definition of the function keys, a definition of the display screens, cursor control, file spooling, and debug support. As will be pointed out later, capabilities which facilitate testing or circumvent deficiencies of the HP2623 driver are also required. These capabilities are accessed through control sequences not normally supported by the HP2623 terminal.

The screens to be implemented are the alpha screen, the graphics screen, and the local screen. These screens must be swapped since there is no hardware available to combine them. The alpha screen is dedicated to displaying alphanumeric input as a dumb terminal. The graphics screen, however, displays the cumulative result of incoming graphics sequences. Function key control of the graphics cursor and display of the cursor position in HP2623 coordinates is also provided. The local screen is reserved for emulator control functions and error reporting. A function key is provided for switching to the alpha or graphics screens.

Program completion now consists of the following steps. First, the interpreter is pseudo coded to provide an overall idea of how the program will fit together. Next, a library of low level functions are developed for the interpreter to draw upon. Related functions are contained in a single module. The function key definitions are then finalized with user friendliness being the major factor in their implementation. Finally, the interpreter is implemented based on these other functions.

2.5 Debugging

Debug support is provided by interspersing the incoming control sequences with descriptions of the emulator response. This output is saved in a file determined by the

operator. The control sequences must be presented in human readable form; therefore, special visible characters are substituted for escape characters, spaces, and unexpected characters. The unexpected control characters are preceded with "^". These debug files have proven extremely useful in the final phase of program development.

A preliminary form of this debug support is an individual program which converts Starbase output into the readable form just described. This is instrumental in determining the set of control sequences the emulator is expected to handle without reporting any errors. It is interesting to note that many HP2627 Color Terminal sequences are included by the HP2623 Monochrome Terminal driver. However, all these sequences produce null output if actually given to an HP2627.

Test and debug are performed on two levels. The low level tests consist of small driver programs which test the low level library functions. These tests are implemented as the modules are completed so that these modules are free of bugs before other modules are based upon them. The high level test tests the emulator as a whole by generating a test file for spooled input by the emulator. This file fully exercises the interpreter and tests any aspects of library functions not tested by the low level tests. Both debug file output and operator observations determine the

outcome of this test. The capabilities of interpreting pause and wait control sequences are included in the emulator since these capabilities expedite testing by adding delays to allow observation of rapidly changing features.

2.6 Demonstration

The final step of the thesis project is to massage Starbase demo programs until they perform with the emulator. This demonstrates compatibility of the emulator with Starbase. The major difficulty is that most of these programs are intended for raster display color terminals involving special circuit cards plugged into the HP9000 backplane. This difficulty is surmounted by implementing a module on the HP9000 which, when linked to the Starbase demo program, produces meaningful output on the emulator. A similar module would still have been required if using a regular HP2623 terminal. To simulate color with a shading technique, it is necessary to provide a capability to the emulator which causes it to ignore changes in drawing mode or line type commanded by Starbase. This capability can be turned on and off via the control sequence supplied to the Starbase "gescape" function. Refer to chapter 4 for more details and sample output from these demonstration programs.

CHAPTER 3

IMPLEMENTATION

3.1 Introduction

This chapter describes the implementation of the HP2623 graphics emulator. First the modular organization is presented followed by a module by module description. These descriptions include the purpose and capabilities of the module along with test methods and nontrivial histories. A description of the header files follow. The configuration files are described in conjunction with the comio.c module. For capabilities of the emulator as a whole, refer to the users' manual and the programmers' reference in the appendix.

The twenty-five modules comprising the graphics emulator can be divided into ten different categories. These categories and modules are tabulated in Table 1. Eight header files are also utilized by the emulator. These files are presented in Table 2 along with the list of modules which include them. Two configuration files, comio.con and comio.wat, are used at run time by the comio.c module. The descriptions follow the order presented in the tables.

TABLE 1
ORGANIZATION OF MODULES

<u>PARAGRAPH</u>	<u>CATEGORY</u>	<u>MODULE</u>
3.2	main control level	grafterm.c init.c reset.c
3.3	command interpretation level	dispcntl.c fkey.c modecmd.c plot.c status.c
3.4	string processing level	format.c interprrt.c
3.5	high level I/O	monitor.c pause.c
3.6	mid level I/O	channel.c error.c
3.7	low level I/O	comio.c kbdio.c
3.8	high level screen functions	cursor.c pen.c relative.c
3.9	mid level screen functions	clip_map.c typemode.c
3.10	low level screen functions	dispmem.c linefill.c
3.11	BIOS interface level	comport.c screen.c

TABLE 2
HEADER FILES

<u>HEADER FILE</u>	<u>MODULES INCLUDING IT</u>
command.h	dispcntl.c grafterm.c modcmd.c status.c
coord.h	clip_map.c dispmem.c linefill.c
grafctyp.h	channel.c format.c grafterm.c interpr.c plot.c status.c
grafstd.h	channel.c comio.c comport.c cursor.c dispcntl.c format.c grafterm.c interpr.c modcmd.c monitor.c pause.c plot.c status.c typemode.c
linebody.h	linefill.c
linefill.h	cursor.c linefill.c typemode.c
relative.h	cursor.c dispcntl.c modcmd.c pen.c plot.c typemode.c
typemode.h	typemode.c

NOTE: See paragraph 3.12 for header file descriptions.

3.2 Main control level

The main control level is the highest level of the emulator. `Grafterm.c` contains main and, after initialization, performs the sorting of control sequences from ordinary characters to be displayed on the alpha screen. These control sequences are then sorted into the types handled by the interpretation level modules. This module is tested with the spooled input test on the completed emulator. The development process is a straightforward implementation of the upper level of the pseudo-coded interpreter.

`Init.c` calls upon the initialization routines within the various modules requiring them, namely `clip_map.c` and `linefill.c`. The primary purpose of this module is to provide a place to collect all the initialization calls. Testing is also performed at the complete emulator level.

`Reset.c` handles two kinds of reset. The first reset calls upon routines which reinstate the graphics default conditions within the appropriate modules. The second reset restores the emulator to start-up conditions. The first reset is intended to be invoked by a standard HP2623 control sequence while the second by a function key. These resets simply call upon functions within the applicable modules. These functions were woven in at the time `reset.c` was implemented. Testing is performed at the complete emulator

level with spooled input as well as with the appropriate function key.

3.3 Command interpretation level

Each module of the command interpretation level interprets its own type of command. Three of the modules, `dispcntl.c` (display control), `modecmd.c` (mode commands), and `status.c`, are based on a template. This template consists of processing a command character and its associated parameter string. A switch is performed on the command character and the appropriate screen function is called with parameters which are derived from the parameter string. The command character and parameter string are determined by `interpret.c`. `Status.c` also performs the following I/O function. The "wait for key press then return graphics cursor position" command requires `status.c` to take over the function of `monitor.c` (discussed later) until a keyboard character is given, thus allowing the operator to position the cursor while allowing the host computer to cancel the request with a new control sequence. Testing of these modules is conducted with spooled input.

`Plot.c` is different mainly because it has to execute pen movement commands as soon as enough parameter characters are received to represent a coordinate pair; therefore, the function of `interpret.c` must be embedded inside the module.

The other difference is that none of the command characters take parameters; all parameters are devoted to pen movement. Control returns to the main control level when a termination character is received. Testing is also conducted with spooled input.

Fkey.c (function key) does not deal with control sequences; instead, it provides a place to tie function keys to the command functions they represent. Function keys are numbered 1 to 10 and shifted function keys are numbered 11 to 20. Kbdio.c is responsible for converting the actual character code generated by the key press into the appropriate one of these numbers. Fkey.c also provides the help screen which is primarily a listing of the function key definitions. Testing is initially performed before control sequence interpretation is added to the emulator. At this level, the emulator is just an enhanced dumb terminal. The reset key is not implemented until after the interpreter is added, so final testing is performed on the complete emulator in conjunction with the spooled input test. Refer to the users' manual for the definition and use of the function keys.

3.4 String processing level

The string processing level provides parsing and format conversion services. Format.c converts characters encoded in binary format into integer lists and converts integers

represented in ASCII into integer lists. Basically, these functions are implemented as state machines which are called with one character at a time. When enough characters have been received to fill the integer list, the function returns true; otherwise, the function returns false. Another function processes a whole string at once and returns true if and only if the integer list was filled. The number of integers in the list and the number of characters used in the binary format to represent one integer are given as parameters. Consult the programmers' reference for the details of the character formats. Testing is conducted independently on this module using keyboard input and display of the results on the screen. Correct performance of the emulator as a whole also verifies this module.

Interprt.c (interpret) interprets incoming characters by catenating parameter characters to the parameter string, ignoring control characters, and, when the command character is received, calling the appropriate command interpretation level function with the parameter string and the command character. This parsing process is repeated until a termination character is received. The address of the appropriate command interpretation level function is passed to interprt.c. Testing is performed with spooled input to the complete emulator.

3.5 High level I/O

The high level I/O modules provide the primary link between the I/O section and the interpretation section of the emulator. Except when paused, inside a special status function, or "ungetting" a character, all I/O is funneled through monitor.c. Therefore, for each request for a character by the interpreter section, the keyboard is serviced. While no characters are available for returning to the interpreter, the keyboard is continually checked for characters to transmit. Graphic cursor blinking is also commanded inside this loop. Testing is conducted at the enhanced dumb terminal level before the interpreter is added.

Pause.c is responsible for holding the emulator in a paused state, only checking the keyboard so that function key presses will be serviced. (No characters are transmitted.) A function key press releases pause and allows the emulator to continue character interpretation. Pause.c will block only once so that the stack cannot overflow. However, this forces the responsibility on pause.c to return to the display screen (graphics, alpha, or local) which was in use at the time of the blocking invocation. Therefore, the screen to return to is passed as a parameter so that the blocking invocation will return to it. Testing is also initially conducted at the enhanced

dumb terminal level.

3.6 Mid level I/O

The mid level I/O modules deal with more than one interface but are not overall I/O managers like the high level I/O modules. Channel.c determines the input channel for the emulator. Input can be taken from the serial line, a file, or a string buffer, with the string buffer taking the highest precedence and the serial line the lowest. An end of file condition is returned when no characters are currently available. Input can also be spooled to a file for later input. Channel.c also provides the debug facilities for the emulator. When this feature is activated, input is visibly reproduced in the debug file along with the debug comments generated elsewhere in the emulator. These comments are usually generated by the command interpretation level modules, by the main control level modules, or by an error. Testing is conducted independently of the other modules using stubs for the interface to the comio.c module. Testing at the enhanced dumb terminal level and at the complete level provides further verification of the module. An example of debug output is provided in figure 4. This output is generated while producing figure 3.

Error.c provides a uniform error reporting mechanism.

The error is reported on local screen and interpretation is paused. A debug comment is also generated. This allows the operator the opportunity to examine the display screens and then choose to continue or quit the emulator based on the severity of the error. Before implementation of error.c, error reporting was performed through print and abort statements. Testing is conducted at the enhanced dumb terminal level.

3.7 Low level I/O

The low level I/O modules are interface specific. Comio.c (communication line I/O) provides the interface to the serial communication line, otherwise known as the com port. Functions equivalent to the standard "C" library functions "getc", "ungetc", "putc", and "puts" are provided to interface the serial line. The "getc" equivalent, however, does not block but instead gives an end of file indication if no characters are currently available. Initialization routines are also provided. Two configuration files are utilized. Comio.con contains the baud rate and the number of the com port to be used. Comio.wat contains a wait factor used to determine the width of the window within which a character is accepted. The existence of comio.con is mandatory but if comio.wat does not exist, the wait factor is calculated and, at the discretion of the operator, stored in a new comio.wat file.

The heart of `comio.c` is the polled I/O algorithm employed to transmit and receive characters. The serial line is maintained in XOFF whenever execution transpires outside of `comio.c`. Since sending a character is apparently equivalent to sending XON, the main difference between sending and receiving a character is that XON is sent for receiving if a character is not already waiting in the buffer. After sending XON or a character, the serial line is continuously checked wait factor times for an incoming character which is placed in the buffer on arrival. This establishes the input window. While characters are received and the buffer is not full, this process is repeated. When full or if no characters arrive within the window, XOFF is sent. While characters continue to arrive within the window, they are stored in the reserve portion of the buffer. From this discussion it is apparent that too small a wait factor allows characters to be missed while too large a wait factor unnecessarily slows down the emulator.

Calculation of the wait factor used in `comio.c` involves several steps. First all incoming characters are thrown out until the serial line is quiet for the duration of an arbitrarily large window. Then a character is sent immediately followed by XOFF. Next the response time of the HP9000 to echo the character after an XON is sent is

measured in terms of the number of times required to check the serial line before it arrives. This measurement is the wait factor. Since the HP9000 ignores XOFF when requesting a password, the carriage return character is used as the test character and the response time to this character after sending the XOFF and before the XON is also checked. This response time is used instead as the wait factor if a response occurs within an arbitrarily large window. Next the wait factor is corrected with a safety margin and finally the echoed character is discarded. If a response is never obtained, the operator can either provide a wait factor or cancel the connection process.

Unfortunately, this calculation procedure does not work when the Sytek network is used to interface the PC running the emulator to the HP9000 directly without a modem. It appears the Sytek tries to build a block of characters before transmitting them if the interval between characters is small enough. If the last character received is XOFF then the block is not transmitted until another character arrives, regardless of the time delay. It seems that the decision by Sytek to block is not only based on the frequency of character arrival but on the length of the character burst at that frequency. Since it appears to be an intractable task to determine the algorithm to beat the Sytek blocking system, a wait factor known to work when

interfacing Sytek through a modem using an 8MHz PC is installed in the comio.wat file for use in this application.

Development of comio.c has been the most difficult of any module comprising the emulator. Originally, the BIOS interface was included in the module and there was no configuration file mechanism. The module in this form together with kbdio.c made up the dumb terminal program implemented to test serial line and keyboard functions as an early phase of emulator development. Much experimentation was required to determine the design of comio.c. Use of the standard "C" library function `getc(stdaux)` was inappropriate since it either waits indefinitely for a character or produces a DOS error when it times out. Use of DOS function 3, serial line input, in conjunction with DOS IOCTL function 44H (hex), a non-blocking check for character presence, proved inadequate since characters were missed at 9600 baud. Finally, it was discovered that using BIOS "INT 14", the low level serial interface, provided sufficient response time to capture characters arriving at 9600 baud. After further experimentation and critical thinking, the sending and receiving algorithms were developed. The fact that characters may still arrive after sending an XOFF in response to no characters arriving in the input window was not anticipated but was demonstrated by experimentation.

More experimentation led to the development of the wait factor calculation algorithm. Fine tuning took place throughout the development of the emulator, and the Sytek problem was not discovered until after the emulator was believed to be completely finished. From the preceding it is apparent that testing is conducted at all test levels for the emulator.

Kbdio.c (keyboard I/O) provides the interface to the keyboard. Functions equivalent to the standard "C" library functions "getc" and "ungetc" are provided for this interface. The "getc" equivalent, however, does not block waiting for a carriage return but instead gives any available characters or an end of file indication if no characters are currently available. While the "getc" equivalent function detects extended keys, such as function keys or cursor movement keys, calls to the fkey.c or the cursor.c module are made as appropriate, and then the next character is returned if available. Testing is initially performed at the dumb terminal level with the calls to cursor.c and fkey.c stubbed in. Testing at the enhanced dumb terminal level verifies the function key and cursor movement key interface.

3.8 High level screen functions

The modules containing the high level screen functions are each concerned with a specific location in HP2623

coordinate space, namely the graphics cursor location, the pen location, and the relative origin. `Cursor.c` provides functions for moving the cursor and displaying the cursor. Display of the current cursor position is also provided by this module and is controlled with a print on/off function. The cursor is drawn with lines which complement the background so when it is redrawn, the cursor disappears and the background is unaffected. This means that the blink command blocks while the cursor is visible so that the screen cannot be disturbed. The cursor is to be blinked when commanded as long as the graphics screen is on, the cursor is on, and the emulator is not in the middle of a plotting sequence unless it is paused (eliminates jerky plotting). The blink command is issued in `monitor.c`, in `pause.c`, and in the `status.c` function which waits for the operator to position the cursor and signal completion with a key press. To implement this logic there is an on/off function, an enable/disable function, and a suspend/resume function. Additionally, a parameter to the blink command can override the suspend function. The need for the suspend/resume function became apparent when the interpreter was implemented and so these functions were added at that time. The cursor movement functions corresponding to the cursor movement keys are also complicated. These functions

normally step by one but will step by ten if called frequently, as when the key is held down. Testing of the cursor.c module is initially conducted at the enhanced dumb terminal level. Final testing is conducted with the spooled input test on the complete emulator.

The pen.c module controls the position and state of the logical pen, drawing lines from the saved pen position to the new pen position given by the move pen function if the pen is in the down state. The pen can also be moved incrementally. Pen up/down functions are provided and an implicit pen down is given for each pen movement. Testing is performed at the enhanced dumb terminal level with calls from a driver function.

The relative.c module controls the position of the relative origin and, through the relative function, provides a means for offsetting parameters to be provided to functions in other modules by the relative origin. The relative function accepts the address of the function along with the four parameters representing the two coordinate pairs to be offset with the relative origin. No harm results if relative is called with only one coordinate pair and the target function expects only one coordinate pair. Testing is performed independently using dummy functions and calls from a driver function.

3.9 Mid level screen functions

The modules containing mid level screen functions interface the low level screen functions of the linefill.c module to the rest of the emulator. Clip_map.c (clipping and mapping) constrains lines and fill areas to screen boundaries and then converts the resulting coordinates from HP2623 coordinate space to PC coordinate space. The appropriate linefill.c function whose address is passed as an argument is then called with the converted coordinates if the line or area exists on the screen. The coordinate conversion is accomplished with a lookup table for speed. For lines, the intersection of the line as defined by its endpoints with the screen boundary replaces the endpoint lying outside the boundary or both endpoints are so replaced if there are two intersections. If the line lies entirely outside the boundary then no action is performed. Fill areas are defined by the endpoints of a diagonal line between the lower left and the upper right corners of a rectangle. The intersection of this rectangle with the screen is used to define new endpoints. No action is taken if there is no intersection. Testing is performed with calls from a driver function when linked with the linefill.c and the screen.c modules. The area fill clip and map function was added when area fills were added to line.c to make linefill.c.

`Typemode.c` (line types and drawing modes) invokes the proper line or fill function from `linefill.c` with the draw or the fill function, respectively. The standard and the user line types along with the current drawing mode are also managed with the `typemode.c` module. For the sake of speed, the selection of the proper `linefill.c` function is accomplished with a lookup table of function addresses indexed by the current HP2623 drawing mode in conjunction with the class of line type currently in use. This address along with the coordinates are passed on to the `clip_map.c` module. Line types can be of either solid, normal, or dot class where normal line types depend on the set line type function of `linefill.c` for their implementation. For more details on line type classes, refer to the discussion of `linefill.c` below. Testing is accomplished with calls from a driver function when linked with the `screen.c`, `clip_map.c`, and `linefill.c` modules.

3.10 Low level screen functions

The modules containing the low level screen functions directly access the screen memory. The `dispmem.c` (display memory) module deals with screen memory as a whole, responsible for swapping the display between the graphics, alpha, and local screens, for clearing the current screen, and for setting and clearing all pixels of the graphics

screen. Functions are also provided to specify the current screen and switch to the screen that was previously specified. This enables the error function, for example, to pause, allow the operator to switch screens with a function key, then return to the screen in effect before the error. The contents of the alpha display and the graphics display are saved before switching from that respective screen. Then the appropriate contents are copied back in to display memory after the screen mode is changed via the screen.c module. The local screen is always cleared so it never needs to be saved. A separate save area is required for both the alpha and the graphics screen because of the existence of the local screen and because changing screen mode clears display memory. Otherwise a simple swap between display memory and a screen buffer could be employed to save memory usage. Currently static memory is used for storing the alpha and graphics screens. If the screen resolution is increased from the 640 X 200 mode to the 640 X 400 mode, then allocated memory must be used since static memory would be exhausted. Testing is conducted with calls from a driver function when linked with the screen.c module. Originally, the screen.c module was one with the dispmem.c module and this combined module was known as screen.c, but the size and capabilities of this module grew to the point where it seemed more appropriate for the BIOS interface to be .

contained in its own module.

The `linefill.c` module contains all the line drawing and rectangle filling algorithms for the emulator. Functions are also provided for initializing lookup tables and setting the line pattern. There is a line drawing function and a rectangular fill function for each unique combination of drawing mode and line type class. Drawing modes are clear, set, complement, and jam. Clear, set, and complement modes affect each pixel along the line being drawn corresponding to bits which are set in the line pattern while the jam mode sets pixels along the line corresponding to bits which are set in the line pattern and clears the other bits along the line. Rectangular fills are simply made up of successive horizontal lines filling the rectangle with the corresponding line pattern. The line type classes enable simplifications to the line drawing algorithm to save time. The solid class eliminates the check to see if the current pixel is affected by the line pattern. The dot class simply affects the endpoints of the line. Only the normal class utilizes the line pattern.

The implementation of `linefill.c` essentially involves solving two problems, where to draw dots and how to draw dots. The "where" problem is independent of line type and drawing mode and concerns itself with approximating a line

on a raster display made up of pixels. The "where" problem is simplified for rectangular fills since all the lines involved are strictly horizontal. It is the "how" problem that is directed toward drawing modes and line types. The "how" problem is also concerned with locating the dots as pixels in the display memory. Since the "how to draw dots" algorithm is utilized at the innermost loop of the "where to draw dots" algorithm, the "how" algorithm is implemented as a macro called `dot(x,y)`. This saves execution time by eliminating the overhead of making function calls. Unfortunately, implementing `dot(x,y)` as a macro forces duplication of the "where" algorithm for each combination of drawing mode and line type class. This would result in a huge amount of source code to manage if this were all contained within the module. Therefore, the body of the "where" algorithm is formed into the `linebody.h` file and simply included after every definition of `dot(x,y)` and declaration of the particular mode and class of line drawing function. The fill functions consist solely of two nested "for" loops over `dot(x,y)`, so the include facility is not utilized for them.

A line is defined with its endpoints in an `x,y` coordinate system made up of discrete points (pixels). At first one might consider a line drawing algorithm that places dots by selecting consecutive `x` values and using the

nearest discrete y value that lies on the given line segment. Unfortunately, lines drawn in this way which are more nearly vertical than horizontal have gaps. This suggests using a different approach for such lines which involves selecting consecutive y values instead. The algorithm utilizing both approaches is based on the following two equations:

$$\begin{aligned}\text{next } y &= (\text{next } x - \text{initial } x) * \text{delta } y / \text{delta } x + \text{initial } y \\ \text{next } x &= (\text{next } y - \text{initial } y) * \text{delta } x / \text{delta } y + \text{initial } x\end{aligned}$$

The first equation uses consecutive x values while the second uses consecutive y values. To be consistent with the preceding analysis, it is assumed that delta y is less than delta x in the first equation and vice versa for the second.

Since these equations are executed for every dot along a line, using a more efficient algorithm should greatly improve performance of the emulator. One way to improve efficiency is to eliminate the integer division and multiplication from the above equations. For the following discussion assume delta y is less than delta x and that consecutive x values are being taken as in the first equation above. Keep a remainder which is initialized to half of delta x. Each time x is incremented add delta y to the remainder. Repeat until the remainder is larger than delta x. At this time increment y and subtract delta x from

the remainder. Repeat this entire process for each x value in the range of the line segment, drawing a dot each time x is incremented. It should be apparent that this algorithm generates the same x and y values as the first equation above, except the y values are rounded by starting with half of delta x instead of truncated by using integer division. Timing tests reveal that eliminating integer division results in a time savings of twenty-four percent! The "C" representation of this algorithm is as follows:

```

for (x = x1, y = y1, rem = delx>>1; x <= x2; x++) {
    dot(x,y);
    if ( (rem += dely) > delx) {
        rem -= delx;
        y++;
    }
}

```

In this representation, x1 and y1 are the initial x and y, x2 is the final x, rem is the remainder, and delx and dely are delta x and delta y. The loop in this representation can only handle the forty-five degree sector above the positive x axis. Seven other loops are required with a different combination of x direction, y direction, and whether the loop is taken over x or over y. If-then-else constructs nested three deep choose the correct loop for the given line segment.

The dot(x,y) macro utilizes a table lookup to locate

the correct byte of display memory and then a bitwise logical assignment operator with a mask also obtained from a lookup table. Bitwise AND is used for clear, inclusive OR for set, and exclusive OR for complement. The line pattern is implemented as a circularly linked list of boolean values which determines which bitwise logical assignments are executed. For example, the set mode, normal class definition of dot(x,y) is as follows:

```
#define dot(x,y) ( (lintyp[ index = next[index] ]) && \
    ( disp[ col[x]+row[y] ] |= bit[x] ) )
```

The jam mode utilizes the conditional expression operator to select between the set or clear operation. The solid class simply eliminates the conditional part of dot which obviously reduces the execution time.

Earlier implementations of dot(x,y) utilized the BIOS "INT 10" video interface. This proved to be about ten times slower than using direct memory access. Using the peek and poke functions provided in the "C" library proved to be about three times slower. Use of peek and poke are alternatives to using the large memory model in compiling and linking the emulator. The large memory model is chosen with a compiler option and with use of the large memory model "C" library. According to the compiler manual [10], the use of the large memory model reduces efficiency in

pointer arithmetic and array indexing; however, the three times improvement in speed when drawing lines makes the use of the large memory model, which allows pointers into display memory, the logical choice.

Testing is conducted with calls from a driver function when the module is linked to `screen.c`. Tests are used to verify performance and make decisions between the various algorithms. Efficiency is paramount in development of this module since the emulator is expected to spend most of its time in drawing lines.

3.11 BIOS interface level

The BIOS interface level is the lowest level of the emulator. Functions in these modules set up the pseudo registers and utilize the "C" library function which loads the real registers from these registers and performs a software interrupt to access the BIOS routine. The `comport.c` module utilizes the BIOS "INT 14" serial line routines and the `screen.c` module utilizes the BIOS "INT 10" video routines. The serial line routines perform nonblocking input and output of a single character and also provide a check for a character waiting for input. The video routines initialize the alpha screen to color 80 mode and the graphics screen to 640 X 200 pixel monochrome mode. The foreground and background colors are also chosen by these initialization routines. Other video routines save

and restore the alpha cursor. Originally, `comport.c` was one with `comio.c` and `screen.c` was one with `dispmem.c`, which was then known as `screen.c`. However, the capabilities and size of these modules grew to a point where it seemed logical to separate out the BIOS interface. Testing is conducted in conjunction with the parent modules.

3.12 Header files

The header files provide a place to define global constants, declare functions whose addresses are to be passed as parameters, and define useful macros. They also reduce duplication of code. The `command.h` file provides declarations of command interpretation level functions for use with the `interpret.c` module. The `coord.h` file defines the dimensions of the HP2623 and the PC coordinate space as well as the start of odd and even rows in display memory. The `grafctyp.h` file expands the set of character type macros given by the standard `ctype.h` file. The `grafstd.h` file provides general purpose, frequently used macros and definitions. The `linebody.h` file contains the body of the line drawing algorithm utilized in the `linefill.c` module in conjunction with each definition of the `dot(x,y)` macro. The `linefill.h` file declares all the line drawing and rectangular fill functions for inclusion in the lookup table utilized in the `typemode.c` module. The `relative.h` file

declares all functions whose address can be passed to the `relative.c` module. The `typemode.h` file defines constants for all the drawing modes and line types and defines the bit patterns and scales representing the seven standard line types.

CHAPTER 4

STARBASE DEMONSTRATION

This chapter discusses the demonstration of Starbase compatibility with the HP2623 graphics emulator. There are basically three types of applications for the HP2623: generation of a two-axis plot of data complete with text labels, drawing a picture, and conducting an interactive session where the display represents the results of moving a locator (the cursor) and using a selector (pressing an alphanumeric key). Three Starbase programs developed on the HP9000 demonstrate these applications, and sample output from each of these is presented in Figures 1, 2, and 3. These figures are first drawn on the screen then dumped to the printer with the "Print Screen" key. The debug file corresponding to Figure 3 is also provided in Figure 4 as a further demonstration of the emulator.

The plot program was originally provided as a part of the Starbase documentation. Modification of the program is fairly straightforward; the major difficulty is properly scaling the text size so that it becomes readable without becoming so large that it overlaps other features. This demonstrates that utilizing the 400 line resolution mode would greatly improve the emulator. This plot is depicted in Figure 1.

The picture program was originally provided with the Starbase demonstration software, and modification of this program is much more difficult. This program, as most of the Starbase demos, was designed for use with a raster display color terminal involving a special circuit card plugged into the back of the HP9000. Many errors are generated when attempting to use this program with the HP2623 driver, mainly because the HP2623 only has two legal colors (black and white) and the picture program tries to access other colors. The solution to this problem was first attempted by implementing a new module, `help2623.c`, to link with the picture program which translates these colors into drawing modes and line types understood by both the HP2623 driver and the emulator. The "gescape" (graphics escape) function of Starbase is utilized to communicate drawing modes and line types not implemented by the HP2623 driver. These drawing modes and line types approximate the following shading function taken from the manual page for `fill_color()` [3]:

$$\text{Intensity} = 0.30 * \text{red} + 0.59 * \text{green} + 0.11 * \text{blue}.$$

However, this version of `help2623.c` does not work; somehow the colors are not represented on the screen. Subsequent examination of the debug file reveals the following problem. Every time the Starbase "fill" command

is executed, a drawing mode and a line type command is sent to the emulator, canceling the previous drawing mode and line type sent by the gescape function. The most straightforward solution is to extend the capabilities of the emulator to include a command which turns on and off the ignoring of drawing mode and line type changes. The help2623.c module was then changed to use this command, causing the emulator to ignore all drawing mode and line type changes except those executed by the help2623.c module. This command is simply embedded in the control sequence sent by the gescape which issues the drawing mode and line type commands. The resulting picture is presented in Figure 2.

Starbase provides three mechanisms for interactive input from the HP2623. The event mechanism reads a device as soon as input becomes available and stores the information in a queue for retrieval at the convenience of the application program. The request mechanism blocks the application program while waiting for input. The sampling mechanism simply returns whatever input is immediately available, indicating input as invalid if none is available. Except when sampling the cursor location, these mechanisms utilize the HP2623 command which return cursor location and key press when the operator presses a key.

Experimentation reveals that the event mechanism works as expected when the device is opened for both input and

output on the same file descriptor, except the Starbase disable event function leaves the emulator still waiting for a key press. This last problem can be worked around by opening a second file descriptor against the same device then closing the first file descriptor followed by the second. The request mechanism works well for obtaining a key press or a cursor location, but to obtain the key press corresponding to the one releasing the cursor location, one must use the sample mechanism for the key press after using the request mechanism for the cursor location. The sample mechanism, however, does not perform for returning the cursor location. Examination of the debug file for the cause of this problem reveals that Starbase does not allow sufficient time for the emulator to respond to the nonblocking cursor return command.

Evaluation of the above results lead to the generation of the interactive demonstration program, which utilizes the request mechanism to obtain the cursor location and the sample mechanism to obtain the key press. The key press indicates whether a rectangle is drawn at the new position, the pen is up and moved to the new position, the pen is down and moved to the new position, or the program is terminated. A sample of the output generated by this program is presented in Figure 3 and the corresponding debug file is

listed in Figure 4. The debug file is provided to illustrate the correlation of the display to the control sequences generated by the Starbase program.

FIGURE 1
PLOT PROGRAM OUTPUT

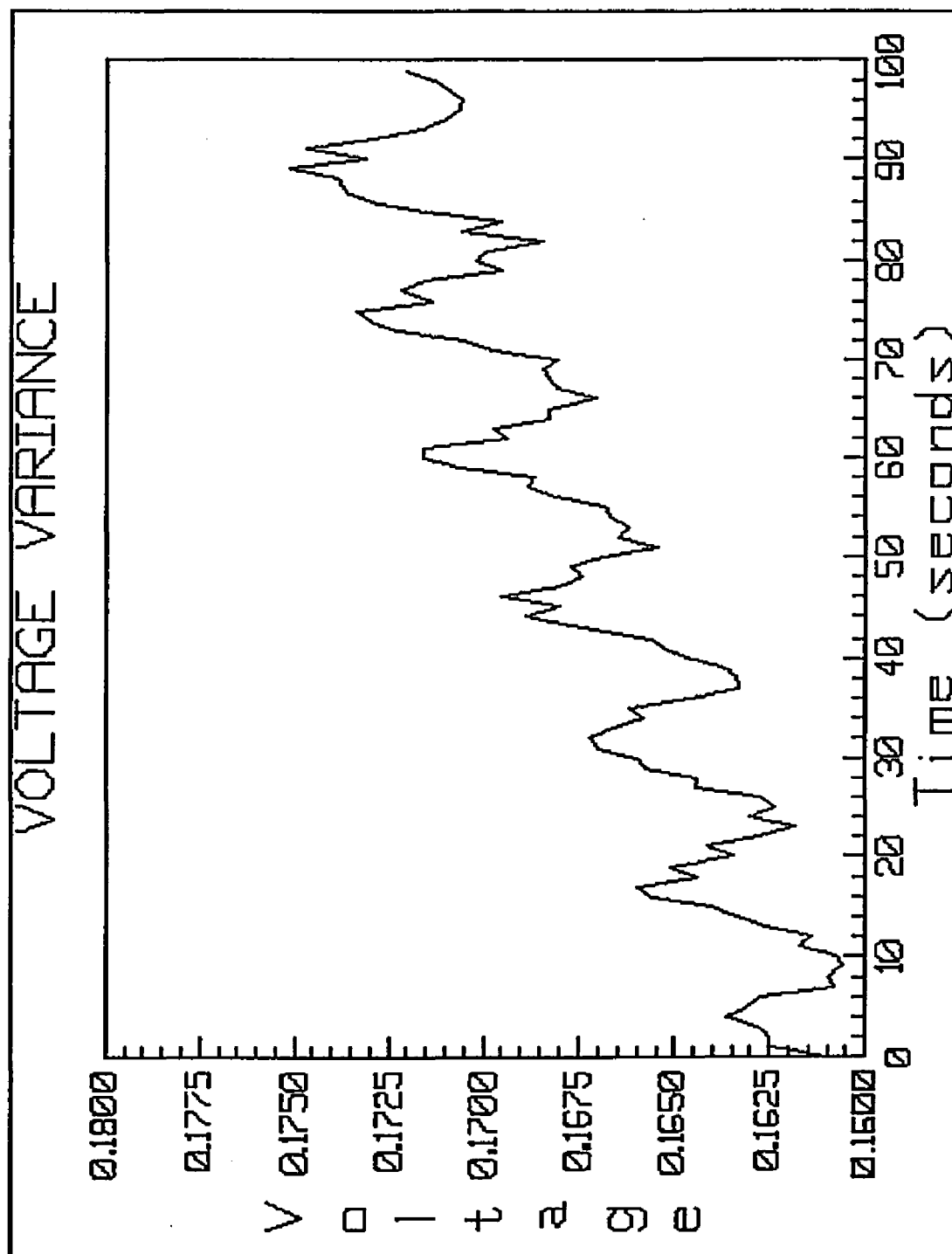


FIGURE 2
PICTURE PROGRAM OUTPUT

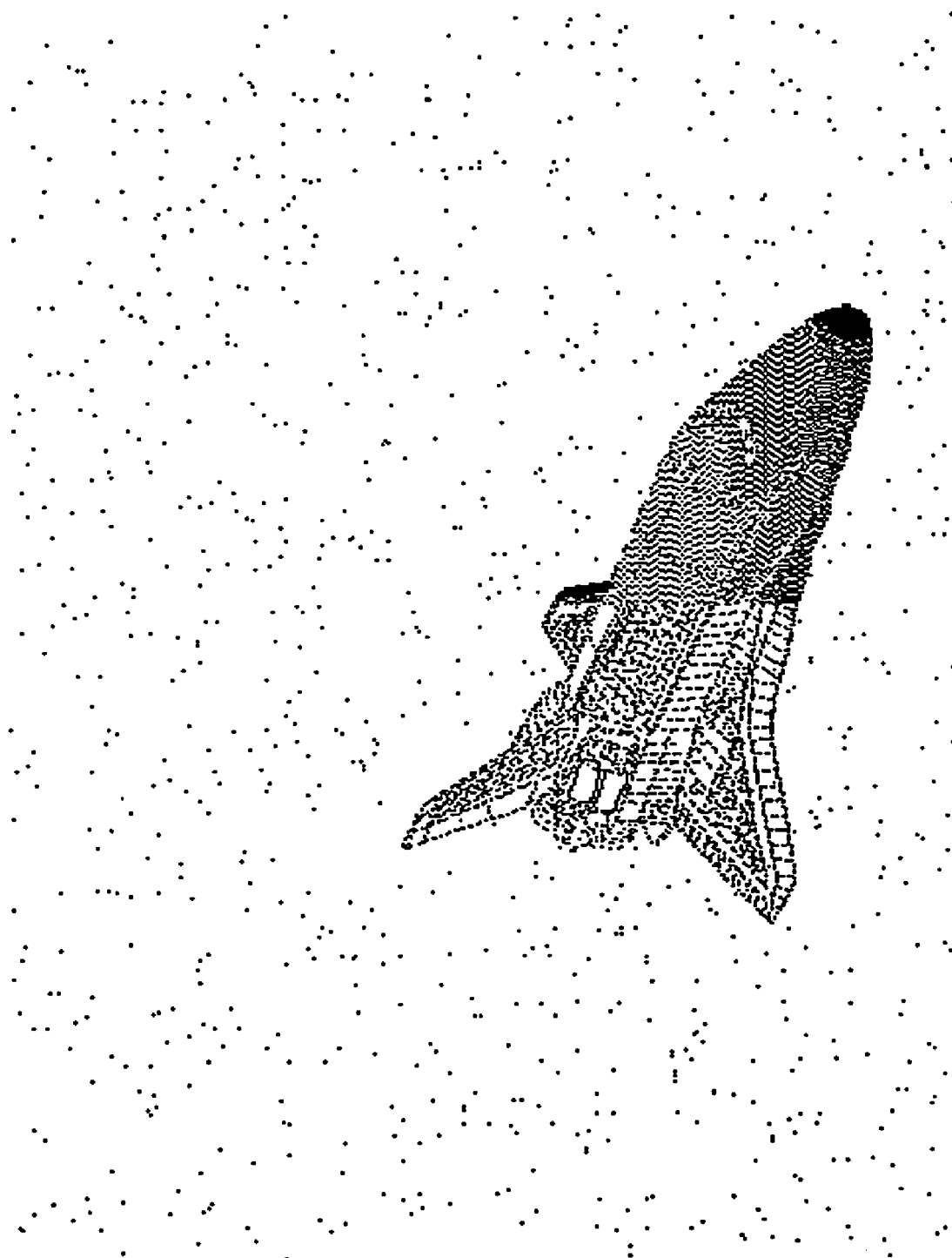


FIGURE 3
INTERACTIVE PROGRAM OUTPUT

201,351

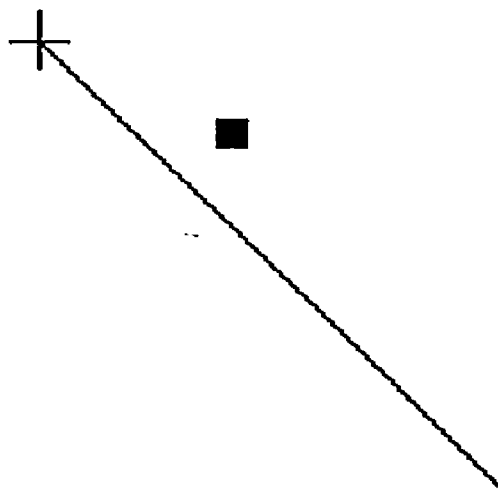


FIGURE 4

DEBUG FILE OUTPUT FOR THE INTERACTIVE PROGRAM

NOTE: escape = "@" blank = "~"
 messages enclosed with "()" unexpected chars
 enclosed with "[]"

```

m                      {out:  m)
                      {out:  y)
                      {out:  d)
yd                     {out:  e)
                      {out:  m)
                      {out:  o)
emo                    {out: ^M)
[ ^M][ ^J][ ^M]
@*s1^
                      {string output: 2623A)
                      {out: ^M)
2623A
[ ^M][ ^J]
@*dT
                      {dispctl: graphics text mode ended)

@*m1n
                      {modecmd: graphics text control)
1q
                      {modecmd: graphics text control)
255,255,255,255,255,255,255,255d
                      {modecmd: define area pattern)
7X
                      {modecmd: hp2627 area pattern or pen
                      control)

```

FIGURE 4 CONTINUED

DEBUG FILE OUTPUT FOR THE INTERACTIVE PROGRAM

```

@*da
C
    {dispcntl: turn off all dots}

    {dispcntl: graphic screen}

@*pA
    {plot: penup}

@*m0,0J
    {modecmd: set relocatable origin to 0,0}

@*dl
    {dispcntl: graph cursor off}

@*d0,0o
    {dispcntl: move graph cursor to 0,0}

@*pi
a
    {plot: binary absolute mode}
    {plot: penup}
    {plot: move pen to 0,0}
z
    {plot: NOP}

@*s4^
    {string output: +00168,+00282,114}
    {out: ^M}
+00168,+00282,114[^M][^J]

@*m1m
    {modecmd: graphics text control}

```

FIGURE 4 CONTINUED
DEBUG FILE OUTPUT FOR THE INTERACTIVE PROGRAM

```
@*mp                                {modecmd: graphics text control}

@*da                                {dispcntl: turn off all dots}

@*mlb                                {modecmd: set line type to 1}

@*m2a                                {modecmd: set drawmode to 2}
Z                                    {modecmd: NOP}

@*pi                                {plot: binary absolute mode}
s                                    {plot: hp2627 area fill starts}

@*ml63,277,173,287e                 {modecmd: fill with x1 = 163, y1 = 277, x2 =
                                     173, y2 = 287}

@*mlb                                {modecmd: set line type to 1}

@*pi                                {plot: binary absolute mode}
a                                    {plot: penup}
%#(5                                {plot: move pen to 163,277}
Z                                    {plot: NOP}
```


FIGURE 4 CONTINUED
 DEBUG FILE OUTPUT FOR THE INTERACTIVE PROGRAM

```

@*dl
    {dispcntl: graph cursor off}

@*d0,0o
    {dispcntl: move graph cursor to 0,0}

@*pi
    {plot: binary absolute mode}
a
    {plot: penup}
----
    {plot: move pen to 0,0}
Z
    {plot: NOP}

@*pi
    {plot: binary absolute mode}
Z
    {plot: NOP}

[ [ ]
34[ ] ]
%
```

CHAPTER 5

SUMMARY

The HP2623 graphics terminal emulator is implemented on the PC. When a PC running the emulator is connected to the HP9000 running various Starbase application programs, meaningful Starbase output can be displayed by the emulator on the screen. The screen can then be dumped to a printer by pressing the "Print Screen" key. With the introduction of this emulator, graphics can now be performed on the HP9000, thereby increasing its usefulness. The "C" modules comprising the emulator provide a platform for future development with such enhancements as high resolution color graphics. The interpreter could also be expanded to support alpha control sequences so use with screen editors such as "vi" could be supported. With the implementation of interrupt based serial I/O and the enhancements just described, the emulator becomes a nice commercial package.

APPENDIX A

USERS' MANUAL

This appendix discusses the installation and the operation of the HP2623 graphics terminal emulator. The name of the executable file containing the code for the emulator is "grafterm.exe". This file along with the configuration files comio.con (mandatory) and comio.wat (optional) should be located together in a separate directory on the IBM PC or compatible such as the AT&T. This directory will be the default location of all debug and spool files generated by the emulator. Connection to the HP9000 is made by connecting one of the serial ports on the PC to one of the standard terminal ports on the HP9000.

Any line in the configuration files beginning with a "#" is a comment and is ignored. The comio.con file must be present, and integers representing the serial port connected to the HP9000 (1 or 2) and the baud rate (300, 1200, or 9600) are the first nonblank characters on the first two non-comment lines, respectively. The comio.wat file, if present, contains an integer wait factor as the first nonblank characters on the first non-comment line. This wait factor adjusts the width of the serial input window during which characters are accepted. The emulator will generate this file if it is not present; however, its

presence expedites start of communication. Note that the emulator does not properly generate the wait factor in installations involving the Sytek network when a modem is not involved. In this case, the operator must provide the `comio.wat` file and experimentally determine the wait factor which is large enough such that no characters are missed but small enough that the emulator is not slowed down unnecessarily.

To run the emulator, simply switch to the directory containing `grafterm.exe` then execute "grafterm". The first screen displayed is the "help" screen which defines all the function keys. The emulator is completely controlled through the function keys and the arrow keys. A description of the use of each of these keys is provided below. Use of control-C or control-break is also permitted for emergency termination but the mode command would probably be required to return the screen to the normal state.

The other screens utilized by the emulator are the error screen, the file input screen, the alpha screen, and the graphics screen. The error screen displays any errors that occur and the emulator is paused while this screen is displayed. The file input screen is displayed while the operator is providing the name of the debug, spool to, or spool from files. The alpha screen provides the dumb

terminal interface to the HP9000 as well as a record of the preceding activities on the PC. The graphics screen shows the cumulative result of all the graphics control sequences provided by the HP9000 as well as cursor movements provided by the operator. Switching between alpha and graphics screens is performed automatically depending on the nature of the input being currently received. Except for the graphics and the alpha screens, the "HP 2623 EMULATOR" label is presented to indicate that the emulator is in "local" mode. If the emulator is paused, a note to press the CONTINUE key is also provided.

The rest of this appendix explains the use of the function keys and the arrow keys. The keys are arranged such that related functions are shift-unshift pairs.

HELP

This function key (F1) causes the "help" screen to be displayed and pauses the emulator.

CLEAR SCREEN

The shift-F1 key causes the current screen to be cleared.

GRAPHICS

The F2 key causes the graphics screen to be displayed. If further input is expected, use the PAUSE key so that examination of the graphics screen will not be interrupted. Be sure to press CONTINUE when ready for more input.

ALPHA

Shift-F2 displays the alpha screen.

CURSOR ON

Display of the graphics cursor is initiated with the F3 key. The cursor will not be seen if the graphics screen is not on or if a control sequence explicitly turning off the cursor is detected. The arrow keys can be used to position the cursor.

CURSOR OFF

Display of the graphics cursor is terminated with the shift-F3 key.

CURSOR REPORT ON

This function key (F4) causes the cursor position in HP2623 coordinates to be displayed in the upper left hand corner of the graphics screen while the cursor is displayed. This report is updated whenever the cursor is moved. Any graphics in the area of the cursor report are permanently obliterated.

CURSOR REPORT OFF

This function key (shift-F4) stops further cursor reporting and clears the area of the report.

READ SPOOL FILE

With the F5 key, the operator is prompted for the name of the spool file which is to be taken as input. An invalid

file name causes re-prompt and a null file name aborts file name input. This file will preempt the serial line until its contents are exhausted or the STOP READING SPOOL FILE key is pressed. If a spool file is already being read the F5 key reports the name of that file and the emulator is paused.

STOP READING SPOOL FILE

The shift-F5 key terminates the use of the spool file as input.

START SPOOL OF INPUT

With the F6 key, the operator is prompted for the name of the file in which to save a copy of all subsequent input received by the emulator. An invalid file name causes re-prompt and a null file name aborts file name input. This file can later be taken as input with the READ SPOOL FILE key. Until the STOP SPOOL OF INPUT key is pressed, any subsequent presses of the F6 key will cause the name of this file to be reported and pause the emulator.

STOP SPOOL OF INPUT

The shift-F6 key closes the spool file.

START DEBUG

With the F7 key, the operator is prompted for the name of the debug file. An invalid file name causes re-prompt and a null file name aborts file name input. This file contains a listing of all subsequent input in human readable

form interspersed with the response of the emulator to each control sequence as it is interpreted. Until the STOP DEBUG key is pressed, any subsequent presses of the F7 key will cause the name of this file to be reported and pause the emulator.

STOP DEBUG

The shift-F7 key closes the debug file.

CONTINUE

The F8 key causes the emulator to continue processing input and exit the paused state.

PAUSE

The shift-F8 key places the emulator in the paused state and processing of input is suspended. The function keys and the arrow keys are still functional, however.

START COM LINE

With the F9 key, connection to the serial line is established through use of the configuration files. The contents of these files are reported and the emulator is paused. If the comio.wat file is not present then the wait factor is calculated and the operator is given the option to generate the comio.wat file. Any errors in configuration are reported and cause the serial line to be disconnected. Until the STOP COM LINE key is pressed, any subsequent presses of F9 generate a message stating that the connection

has already been made and pause the emulator.

STOP COM LINE

The shift-F9 key breaks the connection to the serial line.

QUIT

The F10 key closes all files and returns the operator to the PC operating system.

RESET

The shift-F10 key returns the emulator to initial conditions. All files except the debug file are closed, and the serial line is disconnected. All internal parameters are set to the graphics defaults as in receiving a control sequence requesting a graphics reset.

GRAPHICS CURSOR MOVEMENT

The arrow keys move the cursor one unit in HP2623 coordinate space with each press. If the key is held down, the cursor is moved rapidly across the screen. These keys are functional even when the cursor is not currently displayed.

APPENDIX B

PROGRAMMERS' REFERENCE

This appendix describes the control sequences used to drive the emulator and produce graphics output. Much of the information in this appendix is taken directly from the HP2623 reference manual [6]. All input is made up of seven bit ASCII characters. Each graphic sequence is started with a three character preamble consisting of the ESCAPE character, "*", and one of "d", "m", "p", or "s", depending on the type of graphics sequence. This preamble is followed by characters which are divided into four classes based on the most significant two bits: control (00), parameter (01), command with sequence termination (10), and command with sequence continuation (11). Control characters within a control sequence are ignored excepting the ESCAPE character, which terminates the sequence and introduces the next sequence.

Parameters are usually provided as ASCII digits delimited with commas and spaces. However, plotting sequences can also use a binary format consisting of one, two, or three consecutive parameter class characters per parameter where the value is the integer word represented by the catenation of the least significant five bits from the characters involved with the first character received

providing the most significant five bits. The two character format represents an unsigned ten bit quantity while the one and three character formats represent signed, twos complement five and fifteen bit quantities, respectively. The binary format provides a much more compact representation of parameters than the ASCII representation.

Each command character causes the emulator to execute a command. Except for plotting sequences, parameters associated with the command precede the command (postfix notation). None of the commands in a plotting sequence are associated with the parameters; instead, each pair of parameters represent a point in HP2623 coordinate space to which the logical pen is moved, drawing a line if the pen is down. All commands which allow the sequence to continue are lower case letters. All the upper case equivalents perform the same command except the sequence is terminated and all subsequent input is directed to the alpha screen.

There are four types of control sequences supported by the emulator. The preambles introducing each type are listed in Table 3. Tables 4, 5, 6, and 7 describe the display control, mode control, plot control, and status request control sequences, respectively. Any HP2623 or HP2627 commands or sequence types not supported by the emulator are ignored, except graphics text is announced with an error and

redirected to the alpha screen marked as graphics text. Any other commands or sequence types generate an error.

Specifically, none of the graphics text controls have been implemented; generation of graphics text must be implemented in terms of appropriate plotting sequences. Another HP2623 feature eliminated from the emulator is use of the user area line type. Elimination of these features should not be a problem, however, since Starbase can still produce meaningful output without them.

TABLE 3
GRAPHICS CONTROL SEQUENCES
ESCAPE * <control sequence>

<u>CODE</u>	<u>FUNCTION</u>
d	Display control
m	Mode control
p	Plot control
s	Status

TABLE 4
 DISPLAY CONTROL
 ESCAPE * d <parameters>

<u>CODE</u>	<u>FUNCTION</u>
a	Clear graphics memory
b	Set graphics memory
c	Turn on graphics display
d	Turn on alpha display
e	Turn on alpha display
f	Turn on graphics display
k	Turn on graphics cursor
l	Turn off graphics cursor
<x,y> o	Position the graphics cursor, absolute (0,0 is initial location)
<x,y> p	Position the graphics cursor, relocatable
<t> x	Wait <t> centiseconds
y	Pause with "press CONTINUE" prompt on alpha screen
z	NOP

Example: Clear the graphics display, position the cursor
 at x=100, y=100, and turn the cursor on.

ESCAPE * d a 100,100o K

TABLE 5
VECTOR DRAWING MODE
ESCAPE * m <parameters>

<u>CODE</u>	<u>FUNCTION</u>
<mode> a	Select drawing mode (1-4 in Note 1)
<line type> b	Select line type (1-11 in Note 2)
<pattern> <scale> c	Define line pattern (set bit = on, scale is 1-16)
<x1,y1,x2,y2> e	Fill area, absolute (lower left and upper right coordinates)
<x1,y1,x2,y2> f	Fill area, relocatable (lower left and upper right coordinates)
<x,y> j	Select relocatable origin (0,0 is initial location)
k	Set relocatable origin to current pen position
l	Set relocatable origin to graphics cursor position
r	Set graphics defaults (reset)
s	Start ignoring select drawing mode and line type commands
t	Stop ignoring select drawing mode and line type commands
z	NOP

TABLE 5 CONTINUED

NOTE 1: 1 (clear), 2 (set), 3 (complement), 4 (jam)

NOTE 2: 1 (solid line) 5 (line #2) 9 (line #6)
2 (user line pattern) 6 (line #3) 10 (line #7)
3 (error -- no support) 7 (line #4) 11 (point plot)
4 (line #1) 8 (line #5)

Example: Select the set drawing mode, user line type of
alternating sets of 3 dots, and fill an area
from 0,0 on lower left to 80,90 on upper right.

ESCAPE * m 2a 2b 85 3c 0,0 80,90E

TABLE 6
PLOTING COMMANDS
ESCAPE * p <parameters>

<u>CODE</u>	<u>FUNCTION</u>
a	Lift the pen (initial state)
b	Lower the pen (implicit after each point)
c	Use graphics cursor as new point
d	Draw a point at the current pen position and lift the pen
e	Set relocatable origin to the current pen position
f	Data is ASCII absolute (default for each sequence)
g	Data is ASCII incremental
h	Data is ASCII relocatable
i	Data is binary absolute (10 bit unsigned)
j	Data is binary short incremental (5 bit)
k	Data is binary incremental (15 bit)
l	Data is binary relocatable (15 bit)
z	NOP
<x,y>	Move pen to <x,y> and perform an implicit pen down, drawing a line if pen is down (initial location is 0,0)

Example: Draw a box 25 units wide and 10 units high, beginning at x=100, y=50.

ESCAPE * p a f 100 50 g 25,0 0,10 -25,0 0,-10Z

TABLE 7

GRAPHICS STATUS

ESCAPE * s <parameter> ^

<u>CODE</u>	<u>FUNCTION</u>
1	Read device ID (response "2623A\r")
2	Read pen position (response "+xxxxx,+yyyyy,p\r" p = 0/1 for up/down)
3	Read graphics cursor position (response "+xxxxx,+yyyyy\r")
4	Read cursor position, wait for decimal ASCII key code (response "+xxxxx,+yyyyy,ddd\r")
5	Read display size (response "+00000,+00000,+00511,+00389,00002.,00002.\r")
6	Read graphics capabilities (response "3,1,0,0,1,0,0,1,1,1,1,2,0,0,0,0\r")
7	Read graphics text status (response "+00007,+00010,1\r")
8	Read zoom status (response "001.,0\r")
9	Read relocatable origin (response "sxxxxx,syyyyy\r")
10	Read reset status (response "b,0,0,0,0,0,0,0,0\r" b=1/0 for reset/no reset since last check)
11	Read area shading (response "1,8,8\r")
12	Read dynamics (response "1,1\r")
<any other>	Any other parameter maps to 1
Example:	Read text status. ESCAPE * s 7 ^

LIST OF REFERENCES

- [1] The Hewlett-Packard Company in conjunction with The Regents of the University of Colorado (a body corporate), The Regents of the University of California, and AT&T Technologies, "HP-UX Unix Operating System for the HP9000 Series 500 Computer", The Hewlett-Packard Company, 1985.
- [2] Microsoft Company, "Disk Operating System, Version 2.1", International Business Machines Corp., 1983.
- [3] The Hewlett-Packard Company, "Starbase graphics package", as described by the following three manuals: HP-UX Concepts and Tutorials Vol. 6: Graphics, Starbase Device Drivers Library for HP9000 Series 500 Computers, and Starbase Reference, The Hewlett-Packard Company, 1985.
- [4] The Hewlett-Packard Company, "Advanced Graphics Package", The Hewlett-Packard Company, 1985.
- [5] The Hewlett-Packard Company, "Device-independent Graphics Library", The Hewlett-Packard Company, 1985.
- [6] The Hewlett-Packard Company, "2622A/2623A Display Terminals Reference Manual", The Hewlett-Packard Company, 1982.
- [7] The Hewlett-Packard Company, "2627A Color Graphics Terminal Reference Manual", The Hewlett-Packard Company, 1982.
- [8] The Hewlett-Packard Company, "Starbase Driver Development Guide", The Hewlett-Packard Company, 1985.
- [9] Kernighan, B. W. and D. M. Ritchie, "The C Programming Language", Prentice-Hall, Englewood Cliffs, New Jersey, 1978.
- [10] Bright, Walter, "Datalight C Compiler", Datalight Company, Seattle, Washington, 1985.
- [11] International Business Machines Corp., "Disk Operating System Technical Reference, Version 3.00", International Business Machines Corp., 1985.

LIST OF REFERENCES CONTINUED

- [12] Microsoft Company, "Technical Reference, Personal Computer XT, Version 2.02", International Business Machines Corp., 1983.