NOTICE

Purpose: This manual provides detailed technical information on the internals of the HP 150 hardware, firmware, and software. It is designed to aid in the development of hardware and software products which will become part of, or work in conjunction with, the HP 150.

User Level: Use of this information assumes a background in digital logic and assembly-language programming.

Related Products: The HP 150 Programmer's Tool Kit (Product 45435A) contains the Assembler, Linker, and appropriate documentation. The HP 150 MS-DOS User's Guide (Product 45624A) provides details on MS-DOS.

Compatibility: A hardware/software product designed to be dependent on one specific version of the HP 150 hardware, firmware, or software may be incompatible with other versions of the HP 150 -- or such a product may be incompatible with future personal computer products from Hewlett-Packard, including models based on the HP 150 itself.

Support: Because of the specialized nature of this information, the many capabilities of the HP 150 when used at this level, and the importance of design strategy for successful implementation of hardware/firmware-based products, assistance in the use of this manual is provided through local consulting -- available by the hour (Product 45686A) or by the day (Product 45687A). Contact your local HP Sales and Service Office -- ask for Personal Computer Systems Engineering.

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

This document contains proprietary information which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another program language without the prior written consent of Hewlett-Packard Company.

Copyright (c) 1984 by HEWLETT-PACKARD COMPANY
The List of Effective Pages gives the date of the current edition, and lists the dates of all pages of that edition and all updates. Within the manual, any page changed since the last edition is indicated by printing the date the changes were made on the bottom of the page. Changes are marked with a vertical bar in the margin. If an update is incorporated when an edition is reprinted, these bars and dates remain. No information is incorporated into a reprinting unless it appears as a prior update.

First Edition.............May 1984

<table>
<thead>
<tr>
<th>Effective Pages</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL................</td>
<td>May 1984</td>
</tr>
</tbody>
</table>
New editions are complete revisions of the manual. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by the customer. The date on the title page and back cover of the manual changes only when a new edition is published. When an edition is reprinted, all the prior updates to the edition are incorporated. No information is incorporated into a reprinting unless it appears as a prior update. The edition does not change.

First Edition.............May 1984
This document provides detailed technical information revealing internal details of the HP 150 hardware, firmware, and software.

The information presented allows development of hardware and software products for the HP 150 which may be incompatible with future personal computer products from Hewlett-Packard, including models based upon the HP 150 itself.

This manual presents technical information concerning the HP 150 Personal Computer and covers its hardware, firmware, and software subsystems.

The intent of the manual is to aid in the development of hardware and software products which will become part of, or work in conjunction with, the HP 150.

The reader is directed towards the HP 150 Programmer's Reference materials for complementary programming information including language and development tools.
MANUAL OVERVIEW

This manual consists of the following sections and appendices:

Section 1 - Introduction provides an overview of the HP 150.

Section 2 - Hardware Overview provides product specifications information and briefly describes each subsystem of the HP 150.

Section 3 - Hardware Subsystems contain detailed information on the HP 150's hardware subsystems and helpful design hints.

Section 4 - Memory and I/O Mapping provides information on the memory and input/output bit mappings of the system's processor.

Section 5 - System Software provides information on the operating systems, device drivers, MS-DOS calls, AGIOS, BIOS, configuration, and disc format structure.

Section 6 - System Firmware provides mapping information on system RAM memory.

Section 7 - Programming the HP 150 contains programming information on escape sequences, MS-DOS, AGIOS, alphanumeric and graphics displays, datacomm, keyboard, HPIB, and accessory card interfacing.

Section 8 - AGIOS Function Call Reference provides a reference list of AGIOS function calls.

Appendix A - Logic Diagrams contain schematic diagrams of the HP 150.

Appendix B - Further Reference Documents provides a list of reference documents to supplement this manual.
# TABLE OF CONTENTS

## Section 1 - INTRODUCTION
HP 150 System Overview .................................................. 1-1
System Architecture ....................................................... 1-1
Display ................................................................. 1-2
Keyboard ............................................................... 1-3
Communications ......................................................... 1-3

## Section 2 - HARDWARE OVERVIEW
Introduction ........................................................................ 2-1
Product Specifications ....................................................... 2-1
  General Description ..................................................... 2-1
  Physical Specifications ................................................ 2-2
  Environmental Specifications ....................................... 2-2
  Product Regulations ................................................... 2-3
  Power Requirements ................................................... 2-3
  Communications ........................................................ 2-3
  Subsystem Power Requirements .................................. 2-3
HP 150 Printed Circuit Assemblies ..................................... 2-5
  Analog Boards .......................................................... 2-6
  Digital Logic Boards .................................................. 2-6

## Section 3 - HARDWARE SUBSYSTEMS
Front Plane ........................................................................ 3-1
  Thermal (Integral) Printer Interface .................................. 3-1
  Front Plane Connectors and Signals ............................... 3-2
  Video Board Front Plane Connector ................................ 3-4
  Accessory Front Plane Connectors ................................ 3-5
  Sweep Board Connector ................................................ 3-6
  Touchscreen Connector ................................................ 3-6
  Integral Printer Connector ............................................ 3-7
  Power Supply Connector ................................................ 3-7
  Front Plane Signal Descriptions ................................... 3-7
Processor Subsystem ....................................................... 3-10
  Overview ........................................................................ 3-10
  Memory and I/O Mappings .......................................... 3-11
  Processor Board Block Diagrams ................................... 3-11
  Reset, CMOS Power, and Test Strap Logic ..................... 3-13
  Microprocessor System Architecture .............................. 3-13
  Bus Cycles ..................................................................... 3-14
  System Timing and Control Logic ................................... 3-14
  Interrupt Controller ..................................................... 3-18
  I/O Devices .................................................................... 3-19
Video Subsystem ............................................................. 3-23
  Video Technology and Display Format ......................... 3-23
  Displaying Alpha Characters ....................................... 3-27
  Video Board Overview ................................................ 3-32
# TABLE OF CONTENTS (Continued)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard and Touchscreen Subsystem</td>
<td>3-37</td>
</tr>
<tr>
<td>Keyboard</td>
<td>3-37</td>
</tr>
<tr>
<td>Touchscreen</td>
<td>3-38</td>
</tr>
<tr>
<td>Specifications</td>
<td>3-40</td>
</tr>
<tr>
<td>Timing</td>
<td>3-40</td>
</tr>
<tr>
<td>8041 Keyboard/Touchscreen Scanner</td>
<td>3-44</td>
</tr>
<tr>
<td>Datacomm Subsystem</td>
<td>3-50</td>
</tr>
<tr>
<td>General Description</td>
<td>3-50</td>
</tr>
<tr>
<td>RS232C/422 Datacomm Module Connector</td>
<td>3-50</td>
</tr>
<tr>
<td>Baud Rate Generator</td>
<td>3-51</td>
</tr>
<tr>
<td>Multi-Protocol Controller</td>
<td>3-52</td>
</tr>
<tr>
<td>Communications Interface Circuitry</td>
<td>3-53</td>
</tr>
<tr>
<td>Mezzanine Memory Subsystem</td>
<td>3-55</td>
</tr>
<tr>
<td>Mezzanine Memory PCA</td>
<td>3-55</td>
</tr>
<tr>
<td>PCA Overview</td>
<td>3-55</td>
</tr>
<tr>
<td>Connector Signals</td>
<td>3-57</td>
</tr>
<tr>
<td>ROM</td>
<td>3-59</td>
</tr>
<tr>
<td>Slot Selection Generation</td>
<td>3-60</td>
</tr>
<tr>
<td>CMOS RAM</td>
<td>3-61</td>
</tr>
<tr>
<td>LEDs</td>
<td>3-62</td>
</tr>
<tr>
<td>Dynamic RAM</td>
<td>3-62</td>
</tr>
<tr>
<td>Accessories Subsystem</td>
<td>3-64</td>
</tr>
<tr>
<td>Accessory Hardware Design Guidelines</td>
<td>3-64</td>
</tr>
<tr>
<td>Mechanical Specifications</td>
<td>3-64</td>
</tr>
<tr>
<td>Power Requirements</td>
<td>3-64</td>
</tr>
<tr>
<td>Thermal Limits</td>
<td>3-65</td>
</tr>
<tr>
<td>Accessory Signal Loading Restrictions</td>
<td>3-65</td>
</tr>
<tr>
<td>Signal Timing Diagrams</td>
<td>3-67</td>
</tr>
<tr>
<td>Accessory Front Plane Connector</td>
<td>3-70</td>
</tr>
<tr>
<td>Accessory Connector Signal Descriptions</td>
<td>3-71</td>
</tr>
<tr>
<td>Accessory Card Hardware and Electrical</td>
<td>3-73</td>
</tr>
<tr>
<td>Electrical Design</td>
<td>3-73</td>
</tr>
<tr>
<td>Mechanical Design</td>
<td>3-76</td>
</tr>
<tr>
<td>Drawings</td>
<td>3-80</td>
</tr>
</tbody>
</table>

## Section 4 - MEMORY AND I/O MAPPING

<table>
<thead>
<tr>
<th>Memory Mapped Devices</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HP 150 Memory Map</td>
<td>4-1</td>
</tr>
<tr>
<td>CRT Controller Registers</td>
<td>4-3</td>
</tr>
<tr>
<td>Horizontal Timing Registers (R0, R1, R2 and R3)</td>
<td>4-4</td>
</tr>
<tr>
<td>Vertical Timing Registers (R4, R5, R7, R8 and R9)</td>
<td>4-5</td>
</tr>
<tr>
<td>Pin Configuration/Skew Bits Register (R6)</td>
<td>4-5</td>
</tr>
<tr>
<td>DMA Control Register (RA)</td>
<td>4-5</td>
</tr>
<tr>
<td>Control Register (RB)</td>
<td>4-6</td>
</tr>
<tr>
<td>Table Start Register (RC and RD)</td>
<td>4-6</td>
</tr>
<tr>
<td>Auxiliary Register 1 (RE and RF)</td>
<td>4-6</td>
</tr>
<tr>
<td>Sequential Break Register 1 (R10)</td>
<td>4-6</td>
</tr>
</tbody>
</table>
### TABLE OF CONTENTS (Continued)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Row Register (R1)</td>
<td>4-6</td>
</tr>
<tr>
<td>Data Row End Register (R12)</td>
<td>4-6</td>
</tr>
<tr>
<td>Auxiliary Address Register 2 (R13 and R14)</td>
<td>4-7</td>
</tr>
<tr>
<td>Start Command (R15)</td>
<td>4-7</td>
</tr>
<tr>
<td>Reset Command (R16)</td>
<td>4-7</td>
</tr>
<tr>
<td>Smooth Scroll Offset Register (R17)</td>
<td>4-7</td>
</tr>
<tr>
<td>Vertical Cursor Control Register (R18)</td>
<td>4-7</td>
</tr>
<tr>
<td>Horizontal Cursor Register (R19)</td>
<td>4-7</td>
</tr>
<tr>
<td>Cursor Registers R38 and R39 (READ)</td>
<td>4-7</td>
</tr>
<tr>
<td>Status Register (R3A)</td>
<td>4-8</td>
</tr>
<tr>
<td>Vertical Light Pen Register (R3B)</td>
<td>4-8</td>
</tr>
<tr>
<td>Horizontal Light Pen Register (R3C)</td>
<td>4-8</td>
</tr>
<tr>
<td>Video Attribute Latch</td>
<td>4-8</td>
</tr>
<tr>
<td>I/O Mapped Devices</td>
<td>4-9</td>
</tr>
<tr>
<td>HP 150 Input/Output Map</td>
<td>4-9</td>
</tr>
<tr>
<td>Real Time Clock (MM58167A)</td>
<td>4-10</td>
</tr>
<tr>
<td>Integral Printer Interface</td>
<td>4-11</td>
</tr>
<tr>
<td>Keyboard/Touchscreen Controller (8041A)</td>
<td>4-11</td>
</tr>
<tr>
<td>Datacomm Port 1 Control Lines/Manuf Test Repeat</td>
<td>4-12</td>
</tr>
<tr>
<td>Datacomm Port 2 Control Lines/Clock Source Select</td>
<td>4-12</td>
</tr>
<tr>
<td>Interrupt Controller (8259)</td>
<td>4-13</td>
</tr>
<tr>
<td>Baud Rate Generator (8116T)</td>
<td>4-13</td>
</tr>
<tr>
<td>HPIB Controller (9914)</td>
<td>4-14</td>
</tr>
<tr>
<td>MPSC - Datacomm Controller (7201/8274)</td>
<td>4-14</td>
</tr>
</tbody>
</table>

### Section 5 - SYSTEM SOFTWARE
- Operating System Structure | 5-1
- The Command Processor | 5-2
- The Personal Applications Manager (P.A.M.) | 5-2
- Applications Programs | 5-2
- Basic Disc Operating System (BDOS) | 5-3
- Basic Input/Output System (BIOS) | 5-3
- Operating System Memory Usage | 5-4
- Operating System Memory Map | 5-4
- Interrupt Vectors | 5-5
- MS-DOS Interrupts | 5-7
- HP 150 Hardware Interrupts | 5-7
- Firmware Variables | 5-7
- BIOS and BDOS | 5-7
- Disc Buffer Cache | 5-7
- File Control Blocks (FCBs) | 5-8
- Fields of the FCB | 5-8
- Installable Device Drivers | 5-10
- PAMCODE.EXE or Resident Portion of COMMAND.COM | 5-10
- Application Program Area (Program Segment) | 5-11
- Program Segment Prefix (PSP) Control Block | 5-12
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>How a Program Terminates</td>
<td>5-13</td>
</tr>
<tr>
<td>Conditions in Effect When a Program Receives Control</td>
<td>5-13</td>
</tr>
<tr>
<td>Transient Portion of COMMAND.COM</td>
<td>5-15</td>
</tr>
<tr>
<td>HP 150 Devices</td>
<td>5-16</td>
</tr>
<tr>
<td>Logical (Mappable) Devices</td>
<td>5-16</td>
</tr>
<tr>
<td>Physical Devices</td>
<td>5-16</td>
</tr>
<tr>
<td>Mapping Logical to Physical Devices</td>
<td>5-16</td>
</tr>
<tr>
<td>The Device Configuration Utility</td>
<td>5-17</td>
</tr>
<tr>
<td>Installable Devices</td>
<td>5-18</td>
</tr>
<tr>
<td>Character Devices and Block Devices</td>
<td>5-20</td>
</tr>
<tr>
<td>How Application Programs Can Get to Devices</td>
<td>5-20</td>
</tr>
<tr>
<td>Device Driver Structure</td>
<td>5-21</td>
</tr>
<tr>
<td>Pointer to Next Device Field</td>
<td>5-22</td>
</tr>
<tr>
<td>Attribute Field</td>
<td>5-22</td>
</tr>
<tr>
<td>Strategy and Interrupt Routines</td>
<td>5-23</td>
</tr>
<tr>
<td>Name Field</td>
<td>5-23</td>
</tr>
<tr>
<td>Device List</td>
<td>5-24</td>
</tr>
<tr>
<td>How to Create a Device Driver</td>
<td>5-26</td>
</tr>
<tr>
<td>How MS-DOS Calls a Device Driver</td>
<td>5-27</td>
</tr>
<tr>
<td>Request Header</td>
<td>5-27</td>
</tr>
<tr>
<td>Unit Code</td>
<td>5-27</td>
</tr>
<tr>
<td>Command Code Field</td>
<td>5-28</td>
</tr>
<tr>
<td>Status Word</td>
<td>5-28</td>
</tr>
<tr>
<td>Device Driver Functions and Parameters</td>
<td>5-31</td>
</tr>
<tr>
<td>Init</td>
<td>5-31</td>
</tr>
<tr>
<td>Media Check</td>
<td>5-32</td>
</tr>
<tr>
<td>Build BPB (BIOS Parameter Block)</td>
<td>5-33</td>
</tr>
<tr>
<td>Read or Write</td>
<td>5-35</td>
</tr>
<tr>
<td>Non-Destructive Read No Wait</td>
<td>5-36</td>
</tr>
<tr>
<td>Status</td>
<td>5-37</td>
</tr>
<tr>
<td>Flush</td>
<td>5-38</td>
</tr>
<tr>
<td>HP 150 Installable Device Driver Example</td>
<td>5-39</td>
</tr>
<tr>
<td>AGIOS: I/O Control of the Con Device</td>
<td>5-49</td>
</tr>
<tr>
<td>The Alpha/Graphic Input/Output System (AGIOS)</td>
<td>5-49</td>
</tr>
<tr>
<td>Accessing the AGIOS</td>
<td>5-49</td>
</tr>
<tr>
<td>BIOS and Its Devices</td>
<td>5-51</td>
</tr>
<tr>
<td>Introduction</td>
<td>5-51</td>
</tr>
<tr>
<td>The CONFIG.SYS File</td>
<td>5-55</td>
</tr>
<tr>
<td>Disc Format and Directory Structure</td>
<td>5-57</td>
</tr>
<tr>
<td>Physical Disc Format</td>
<td>5-57</td>
</tr>
<tr>
<td>Disc Media Storage Capacity</td>
<td>5-57</td>
</tr>
<tr>
<td>Disc Sector Allocation</td>
<td>5-58</td>
</tr>
<tr>
<td>Header Record</td>
<td>5-59</td>
</tr>
<tr>
<td>Boot Sector</td>
<td>5-61</td>
</tr>
<tr>
<td>File Allocation Table (FAT)</td>
<td>5-62</td>
</tr>
<tr>
<td>Disc Clusters</td>
<td>5-62</td>
</tr>
<tr>
<td>FAT Structure</td>
<td>5-62</td>
</tr>
<tr>
<td>How to Use the File Allocation Table</td>
<td>5-62</td>
</tr>
<tr>
<td>MS-DOS Disc Directory</td>
<td>5-63</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS (Continued)

## Section 6 - SYSTEM FIRMWARE
- Firmware Memory Map .................................................. 6-1
- RAM ................................................................................. 6-1
- Firmware Entry Point Jump Vectors .................................. 6-1

## Section 7 - PROGRAMMING THE HP 150
- Generating Escape Sequences ........................................... 7-1
- Performing MS-DOS System Function Calls ....................... 7-2
- Making AGIOS Function Calls .......................................... 7-5
- Alphanumeric Display Interfacing ...................................... 7-8
  - Alphanumeric Video RAM Structure ................................ 7-8
  - Alpha Video Buffer Format .......................................... 7-8
  - Video Row Pointer Table Format ................................ 7-9
  - Row Pointer Formulation ........................................... 7-9
  - Line Buffer Format .................................................. 7-10
  - Enhancement/Character Set Byte Structure .................... 7-10
- Character Code Structure .................................................. 7-10
- Finding Row Pointer Table Origin ..................................... 7-11
- Fetching Row Pointers from the Table ............................ 7-12
- Placing a Character in Alpha Memory .............................. 7-13
- Writing to Lines 25, 26, and 27 ..................................... 7-15
- Graphics Display Interfacing ............................................ 7-18
- Keyboard Interfacing .................................................... 7-22
  - Keycode Mode ............................................................ 7-22
  - Console RAW/COOKED Mode ....................................... 7-23
  - Key Characteristics .................................................. 7-24
  - Sample Keyboard Driver ............................................ 7-25
  - Flushing the Keyboard Buffer .................................... 7-26
  - Keycode Tables .......................................................... 7-29
- Programming Data Communications .................................. 7-34
  - Data Comm Using COM1 and COM2 Logical Devices ........ 7-34
  - Assigning COM1 and COM2 to Physical Ports ................ 7-34
  - Opening the COM Devices ......................................... 7-35
  - Input from the COM Devices ...................................... 7-35
  - Output to the COM Devices ....................................... 7-36
  - Closing the COM Devices ......................................... 7-37
  - COM Device I/O Example - A Terminal Emulator ............ 7-37
  - Programmatic Configuration of Data Comm ................... 7-39
  - Reading the Current Configuration ............................. 7-39
  - Changing the Data Comm Configuration ......................... 7-43
  - Restoring the Original Configuration .......................... 7-46
  - Data Comm Control Functions ................................... 7-48
  - IOCTL Reads for Input ............................................. 7-48
  - Special COM Functions Through IOCTL Write Request .... 7-48
  - Fast Buffer Send for Output ..................................... 7-50
- HPIB Interfacing ............................................................. 7-52
  - Limited HPIB Driver Functionality .............................. 7-52
  - Opening the HPIBDEV Device ..................................... 7-52
# TABLE OF CONTENTS (Continued)

- HPIB Control Calls .................................................. 7-53
- MS-DOS IOCTRL Function Call ........................................ 7-53
- Control Block Format .................................................. 7-53
- Control Template Format .............................................. 7-54
- Sample Identify Templates ............................................ 7-56
- Sample Read/Write Buffer Templates ................................ 7-57
- HPIB Interface Example (9111A Graphics Tablet) .................. 7-58
- Accessory Card Interfacing .......................................... 7-65
- Memory (Slot) Address Identification ................................ 7-65

## Section 8 - AGIOS FUNCTION CALL REFERENCE

- Syntax Used in the AGIOS Function Calls ......................... 8-1
- Batch Function Call ................................................... 8-2
- Video Intrinsics ....................................................... 8-4
  - Define Area ......................................................... 8-5
  - Write Area .......................................................... 8-5
  - Clear Area ........................................................... 8-6
  - Enhance Area ....................................................... 8-6
  - Read Area ............................................................ 8-6
  - Shift Area ............................................................ 8-7
  - Write Line ............................................................ 8-8
- Application Softkeys .................................................. 8-9
  - Update Softkey Label ............................................... 8-9
  - Read Softkey Label ................................................ 8-9
  - Display Softkey Labels ............................................ 8-10
- Control Functions .................................................... 8-11
  - Execute Two Character Sequence (ESC Char) .................... 8-11
  - Position Cursor (ESC & a) ........................................ 8-12
  - Define Enhancements (ESC & d) .................................. 8-13
  - Cursor Sense Absolute (ESC a) .................................. 8-13
  - Cursor Sense Relative (ESC ') ................................... 8-13
  - Set Cursor Type .................................................... 8-14
  - Read Cursor Type .................................................. 8-14
  - Read Terminal Configuration .................................... 8-14
- Touchscreen Functions .............................................. 8-15
  - Field Operations .................................................. 8-15
  - Row Column Operations ........................................... 8-16
  - Define Touch Field (ESC - z g) .................................. 8-17
  - Define Softkey Field (ESC - z s) ................................ 8-18
  - Delete Touch Field (ESC - z d) .................................. 8-18
  - Touchscreen Reset (ESC - z j) ................................... 8-18
  - Set Touch Reporting Mode (ESC - z n) ........................... 8-19
- Keyboard Intercept .................................................. 8-20
  - Define Key Characteristics ....................................... 8-21
  - Get Key Characteristics .......................................... 8-21
  - Put Key .............................................................. 8-22
  - Keycode On/Off ..................................................... 8-22
  - Keycode Status .................................................... 8-23
TABLE OF CONTENTS (Continued)

Read Keypad Status........................................... 8-23
Display Control (ESC * d)..................................... 8-24
Clear Graphics Memory (ESC * d a)........................... 8-24
Set Graphics Memory (ESC * d b).............................. 8-24
Turn On Graphics Display (ESC * d c)......................... 8-24
Turn Off Graphics Display (ESC * d d).......................... 8-25
Turn On Alphanumeric Display (ESC * d e)....................... 8-25
Turn Off Alphanumeric Display (ESC * d f)...................... 8-25
Turn On Graphics Cursor (ESC * d k)........................... 8-25
Turn Off Graphics Cursor (ESC * d l)........................... 8-26
Turn On Rubber Band Line (ESC * d m).......................... 8-26
Turn Off Rubber Band Line (ESC * d n).......................... 8-26
Move Graphics Cursor Absolute (ESC * d <x,y> 0).............. 8-26
Move Graphics Cursor Incremental (ESC * d <x,y> p)........... 8-27
Turn On Alphanumeric Cursor (ESC * d q)........................ 8-27
Turn Off Alphanumeric Cursor (ESC * d r)....................... 8-27
Turn On Graphics Text Mode (ESC * d s)........................ 8-28
Turn Off Graphics Text Mode (ESC * d t)....................... 8-28
Vector Drawing Mode (ESC * m)................................. 8-29
  Select Drawing Mode (ESC * m <mode> a)...................... 8-29
  Select Line Type (ESC * m <type> b).......................... 8-29
Define Line Pattern and Scale (ESC * m<pattern><scale>c). 8-30
Define Area Fill Pattern (ESC * m <pattern> d)............... 8-30
Fill Rectangular Area, Absolute (ESC * m <x1,y1,x2,y2> e). 8-31
Fill Rectangular Area, Absolute Relocatable
  (ESC * m <x1,y1,x2,y2> f)................................. 8-31
Select Polygonal Fill Pattern (ESC * m <pattern> g).......... 8-31
Select Boundary Pen (ESC * m <pen> h).......................... 8-32
No Polygon Boundary (ESC * m h)............................... 8-32
Set relocatable Origin (ESC * m <x,y> j)....................... 8-33
Set Relocatable Origin to Pen Position (ESC * m k).......... 8-33
Set Relocatable Origin to Cursor Position (ESC * m l)....... 8-33
Graphics Text (ESC *)............................................ 8-34
  Set Graphics Text Size Text Size (ESC * m <size> m)..... 8-34
  Set Graphics Text Orientation ((ESC * m <orientation> m)n). 8-34
Turn On Text Slant (ESC * m o)................................ 8-35
Turn Off Text Slant ((ESC * m p)............................... 8-35
Set Graphics Text Origin (ESC * m <0-9> q).................... 8-36
Graphics Text Label (ESC * l <text>).......................... 8-36
Define User Character Set........................................ 8-37
Select Default Character Set.................................... 8-37
Output Single Text Character................................... 8-37
Set Graphics Default (ESC * m r)............................... 8-38
Set Picture Definition Defaults (ESC * m l r)................. 8-39
Graphics Hard Reset (ESC * w r)............................... 8-39
Graphics Plotting (ESC * p)................................... 8-40
  Lift Pen (ESC * mp a)......................................... 8-40
  Vector Move (ESC * pa <x,y>)................................ 8-40
  Lower Pen (ESC * p b)....................................... 8-41
Vector Draw (ESC * p b <x,y>) ........................................ 8-41
Plot to Cursor Position (ESC * p c) .................................. 8-42
Point Plot (ESC * p d) ................................................. 8-42
Set Relocatable Origin to Pen Position (ESC * p e) ........... 8-42
Start Polygonal Area Fill (ESC * p s) ................................. 8-43
Terminate Polygonal Area Fill (ESC * p t) ....................... 8-43
Polygon Move .............................................................. 8-44
Polygon Draw ............................................................... 8-44
Lift Boundary Pen (ESC * p u) ........................................ 8-44
Lower Boundary Pen (ESC * p v) ..................................... 8-45
Graphics Status (ESC * s) ............................................. 8-46
Read Device ID (ESC * s 1) ........................................... 8-46
Read Pen Position (ESC * s 2) ....................................... 8-46
Read Cursor Position (ESC * s 3) ................................. 8-47
Read Cursor Position, Wait for Key (ESC * s 4) ............. 8-47
Read Display Size (ESC * s 5) ....................................... 8-48
Read Graphics Settings (ESC * s 6) ............................... 8-48
Read Graphics Text Status (ESC * s 7) ......................... 8-49
Read Zoom Status (ESC * s 8) ....................................... 8-49
Read Relocatable Origin (ESC * s 9) ............................... 8-50
Read Reset Status (ESC * s 10) .................................... 8-50
Read Area Shading (ESC * s 11) .................................... 8-51
Read Dynamics (ESC * s 12) ........................................ 8-51
Read Extended Screen Dimensions .................................. 8-52

Appendix A - LOGIC DIAGRAMS

Touchscreen PCA ......................................................... A-1
Keyboard PCA ............................................................ A-3
Processor PCA ............................................................ A-5
Processor Front Plane Interface ..................................... A-7
Processor I/O Bus ....................................................... A-9
Processor Datacomm Port .............................................. A-11
Video Alpha RAM Subsystem .......................................... A-13
Video Alpha Display Subsystem ...................................... A-15
Video Graphics Display Subsystem .................................. A-17
Thermal Printer Interface (Part of Front Plane PCA) ....... A-19
Mezzanine Memory PCA ............................................... A-21
Mezzanine Datacomm PCA ............................................. A-23
Sweep PCA ............................................................... A-25
RAM (Memory Extender) PCA ......................................... A-27
Language PCA ............................................................ A-29

Appendix B - FURTHER REFERENCE DOCUMENTS

xiv
This section provides an overview of the HP 150's system architecture, display, keyboard, data communications, and peripherals.
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HP 150 System Overview</td>
<td>1-1</td>
</tr>
<tr>
<td>System Architecture</td>
<td>1-1</td>
</tr>
<tr>
<td>Display</td>
<td>1-2</td>
</tr>
<tr>
<td>Keyboard</td>
<td>1-3</td>
</tr>
<tr>
<td>Communications/Peripherals</td>
<td>1-3</td>
</tr>
</tbody>
</table>
HP 150 SYSTEM OVERVIEW

The HP 150 Personal Office Computer, the third member of the Series 100 family, offers a complete business solution and gives the customer more power and memory space in a small, efficient package. The entire system (display, processor, keyboard, flexible and Winchester disc drives, and integral printer) occupies only 2.1 square feet of space, about the same "footprint" as an open looseleaf notebook. Several features have been added to decrease the amount of time required to learn and use the system such as a touchscreen interface and an advanced "shell" called the Personal Applications Manager (P.A.M.) which shields the user from the "computerese" associated with many computer systems.

The HP 150 system features are summarized below.

System Architecture
- Intel 8088 microprocessor (operating at 8 MHz)
- MS-DOS 2.0 operating system
- Built-in HPTouch
- 256K bytes of main memory standard; 640K bytes maximum
- Battery back-up for system configuration and real time clock

The HP 150 uses an Intel 8088 microprocessor running at 8 MHz. The standard system contains 256K bytes of RAM memory for the operating system, applications and user workspace, and can be expanded to 640K. Touchscreen and graphics are standard with the system.

In addition to the above capabilities, the HP 150 is also a customer expandable system. There are two expansion slots, accessible through the back of the unit, which allow the customer to add accessories (such as additional memory) to the system without requiring the assistance of an HP representative or dealer. Installation of accessories is a simple operation, requiring an average of 5 to 10 minutes.

The HP 150 Personal Computer uses MS-DOS 2.0 from Microsoft Corporation as the standard operating system. MS-DOS 2.0 is a single-user, single-task operating system for which many third-party software packages have been developed. The operating system resides on disc and upon initialization MS-DOS is loaded into the processor's main memory.
Introduction

A unique enhancement by Hewlett-Packard to the MS-DOS operating system has been the addition of an easy-to-understand facility to help the user execute commands. PAM (Personal Applications Manager) provides simple intuitive menus for the most frequently used system commands. Unlike other systems which require the user to learn the system "computerese", PAM through menus and HPTouch, guides the user through commands. With PAM starting applications, creating directories, deleting files and listing existing files can be as easy as touching the screen. For more advanced users the standard MS-DOS command facility is also available.

Display

- Built-in high-resolution on-screen graphics display
  (512 x 390)
- High-resolution character display; 9 x 14 dot character cells; upper and lower case
- Display enhancements: inverse video, underline, blinking, half-bright, security and all combinations
- Up to two pages of 24 lines x 80 characters of display memory

The HP 150 can display both alphanumeric and graphics on the 9 inch diagonal screen. The alphanumeric display consists of a 27 line by 80 column format. The 25th and 26th lines are used for the screen labeling of function keys (and all are automatically "touchable" through touchscreen), and the 27th line is for system status and error messages. The screen memory stores 2 pages of text, which allows off-screen storage of the display. High resolution characters with true descenders are generated in a 9 x 14 dot cell with half-dot shift. The standard display is green character against a black background.

The graphics display has a resolution of 512 dots horizontally by 390 dots vertically. This gives a 1:1 aspect ratio guaranteeing symmetry (that is, circles look like circles). The numeric keypad also serves as the graphics keypad, allowing the customer to turn on and off the alpha display, turn on and off the graphics display or transfer the graphics display to one of the HP graphics printers. It also displays the graphics cursor and allows it to be moved around the screen.
Keyboard

- Detachable, typewriter-style
- Special editing keys
- Numeric/Graphics pad
- Eight screen-labeled function keys

The HP 150 keyboard is designed to provide a familiar interface to the system and minimize training time. The low-profile keyboard shape, the sculptured keycaps and the dished "home" keys help to make the keyboard comfortable to use. The 107-key keyboard contains the full local editing keys such as cursor control keys, display scrolling keys, "next" and "prev" keys for scrolling by pages and "insert" and "delete" keys for inserting or deleting characters or entire lines.

Series 100 function keys are screen labels used by the system and by application programs to increase the ease of use of the system. With the HP 150, this capability is enhanced by the use of HPTouch. Now, all function keys can be selected by pressing the key itself or by touching the key label on the screen.

Communications/Peripherals

- One RS-232/RS-422 communication port
- One RS-232 communication port
- One HP-IB port
- Full block mode graphics terminal support

Two RS-232-C ports (one of which is capable of RS-422 communication) may be used to connect the system to a remote computer or to serial devices (such as printers or plotters). Flexible protocols allow the use of either hardware or software handshaking and communication speeds can range from 110 to 19,200 baud.

The HP 150 contains the HP 2623 Graphics terminal feature set and can run any HP 3000 software which currently runs on that terminal. This includes block mode for V/3000 software and graphics applications such as HPEasychart and HPDraw as well as line-drawing and math character sets, "security" fields, transmit-only fields, edit checks and Tektronix 4010/4014 emulation.
This section provides product specifications information and briefly describes each module of the HP 150.
CONTENTS

Introduction 2-1
Product Specifications 2-1
  General Description 2-1
  Physical Specifications 2-2
  Environmental Conditions 2-2
  Product Regulations 2-3
  Power Requirements 2-3
  Communications 2-3
  Subsystem Power Requirements 2-4
HP 150 Printed Circuit Assemblies 2-5
  Analog Boards 2-6
  Power Supply PCA. 2-6
  Sweep PCA. 2-6
  Digital Logic Boards 2-6
  Processor PCA. 2-6
  Keyboard PCA. 2-7
  Mezzanine Memory PCA. 2-7
  RS232C/422 Datacomm PCA. 2-7
  Front Plane PCA. 2-7
  Video PCA. 2-7
  Touchscreen PCA. 2-7
INTRODUCTION

The HP 150 digital hardware performs the logic functions of a 16 bit personal computer with screen graphics. It consists of a Processor PCA, a Memory PCA, a Video subsystem PCA, a Touchscreen PCA, a Front Plane PCA, and a Keyboard PCA within the keyboard itself. Its operation is based on the 8 MHz 8088 microprocessor.

The Processor PCA provides control signals, input/output and data processing functions. The ROM/RAM PCA provides 256K bytes of dynamic RAM for system and user memory and up to 160K bytes of ROM. The Video subsystem PCA controls the display RAM and provides video display data and timing signals for driving the sweep circuitry. The Touchscreen PCA provides an easy user interface to the system beyond the standard keyboard. Two accessory slots are provided allowing for memory expansion and additional processing and I/O capability.

PRODUCT SPECIFICATIONS

General Description

<table>
<thead>
<tr>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Processor</td>
<td>Intel 8088 microprocessor operating at 8 MHz</td>
</tr>
<tr>
<td>Main Memory</td>
<td>256K bytes of RAM memory</td>
</tr>
<tr>
<td>Screen Size</td>
<td>9 inch diagonal</td>
</tr>
<tr>
<td>Alphanumeric Graphics</td>
<td>116 X 150 mm (4.5 X 5.9 inches)</td>
</tr>
<tr>
<td></td>
<td>120 X 160 mm (4.7 X 6.3 inches)</td>
</tr>
<tr>
<td>Screen Capacity</td>
<td>24 lines X 80 columns, 25th and 26th lines for labeling of function keys, 27th line for system status/error messages.</td>
</tr>
<tr>
<td>Character Generation</td>
<td>7 X 10 enhanced dot matrix with 1/2 dot shifting, 9 X 14 dot character cell, noninterlaced raster scan.</td>
</tr>
<tr>
<td>Character Size</td>
<td>1.3 X 2.8 mm (0.05 X 0.11 inches)</td>
</tr>
<tr>
<td>Character Set</td>
<td>Roman8, line drawing, math standard (also bold and italic usable by applications only)</td>
</tr>
<tr>
<td>Cursor</td>
<td>Blinking underline or blinking square</td>
</tr>
</tbody>
</table>
Hardware Overview

Display Enhancements: Inverse video, underline, blinking, half-bright, security, and all combinations.

Refresh Rate: 60 Hz

Tube Phosphor: P31 (green)

Implosion Protection: Tension band

Keyboard: Full ASCII code keyboard, eight screen-labeled function keys, auto-repeat, N-key rollover, cursor controls, 18 key numeric pad, detachable with 2.43 m (8 ft.) coiled cable.

Physical Specifications

System Processor Weight: 10.15 kg (22.34 lbs.)

Keyboard Weight: 2.14 kg (4.71 lbs.)

Display Monitor Dimensions: 305 mm (W) x 305 mm (D) x 287 mm (H)

Keyboard Dimensions:

- Flat: 456 mm (W) x 225 mm (D) x 35 mm (H)
- Standing: 456 mm (W) x 225 mm (D) x 35 mm (H)

Environmental Conditions

Temperature (Free Space Ambient):

- Non-operating: -40 to +75 °C (-40 to +167 °F)
- Operating: 0 to +50 °C (+32 to +121 °F)
- With 2674A (Integral Thermal Printer): 0 to +50 °C (+32 to +121 °F)

Humidity: 5 to 95% noncondensing

*Vibration: 5-55 Hz @ 0.015" displacement

*Shock: 30g, 11 ms, 1/2 sine
Hardware Overview

*Type tested to qualify for normal shipping and handling original shipping carton.

Product Regulations

This product when used with HP approved options and peripherals meets the requirements of the following agencies/standards for EDP equipment or office equipment in the following countries:

Safety: Canada - CSA Certification
International - IEC 380/435 Compliance
United States - U.L. Listing
Finland - FEI (pending)

RFI: Germany - VDE Class B
United States - FCC Level B

Datacomm: CCITT V.24 interchange V.28 electrical
Australia - Telecom (pending)
Belgium - PTT (pending)
Finland - PTT (pending)
Germany - PTZ (pending)
Sweden - PTT (pending)
U. K. - BT (pending)

Power Requirements

Input Voltage: 115 V (+10%, -25%) at 50/60 Hz (+-5%)
230 V (+10%, -25%) at 50/60 Hz (+-5%)

Power Consumption:

45610A 240 Volt Amp
45650A 356 Volt Amp
45655A 356 Volt Amp
45660A 356 Volt Amp

Communications

Data Channels: 1 HP-IB, 1 RS-232/RS-422, 1 RS-232

HP-IB Channel: Bus used only for specified HP peripherals

RS-232 Channel: General asynchronous communications

Data Rate (RS-232): 110, 150, 300, 600, 1200, 2400,
4800, 9600, and 19200 baud
Hardware Overview

Port 1 and 2: EIA standard RS-232-C and CCITT V.24; hardware and XON/XOFF handshaking available.

Port 1, only: RS-422 communication capability

Subsystem Power Requirements

The figures below are the worst case power consumption figures for the main system boards. The values given for accessory slots are obtained by subtracting the main system board power consumption figures from the power available from the power supply printed specifications. A more realistic set of figures for accessory hardware board designers to use in determining power available for accessories is given in Section 3.

<table>
<thead>
<tr>
<th>Component</th>
<th>+5 Volt</th>
<th>+12 Volt</th>
<th>-12 Volt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sweep Board:</td>
<td>125 mA</td>
<td>1.9 A</td>
<td>60 mA</td>
</tr>
<tr>
<td>Processor:</td>
<td>2.3 A</td>
<td>140 mA</td>
<td>80 mA</td>
</tr>
<tr>
<td>ROM/RAM:</td>
<td>1.0 A</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Video subsystem:</td>
<td>3.4 A</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Front plane:</td>
<td>75 mA</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Thermal Printer:</td>
<td>800 mA</td>
<td>2.2 A *</td>
<td>0</td>
</tr>
<tr>
<td>Touch Screen:</td>
<td>0</td>
<td>150 mA</td>
<td>20 mA</td>
</tr>
<tr>
<td>Mezzanine Datacomm:</td>
<td>200 mA</td>
<td>50 mA</td>
<td>50 mA</td>
</tr>
<tr>
<td>Accessory Slots:</td>
<td>1.75 A</td>
<td>280 mA</td>
<td>270 mA</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td>9.6 A</td>
<td>2.5 A **</td>
<td>480 mA</td>
</tr>
</tbody>
</table>

* This current at +12v for the TPM comes from a separate winding on the power supply.

** This total does not include the current supplied for the TPM on the separate winding.
HP 150 PRINTED CIRCUIT ASSEMBLIES

The HP 150, in its standard configuration consists of seven modules. They are the Processor, Video, Sweep/CRT, Touchscreen, Keyboard, Power Supply, and Front Plane. Below is a block diagram of the HP 150 system showing each of the modules and their associated PCAs.

![Block Diagram of the HP 150 System]

Figure 2-1. HP 150 Block Diagram

A brief description of each PCA in the HP 150 system is given below. Each board is described in detail in Section 3.
Analog Boards

POWER SUPPLY PCA. The power supply used in the HP 150 system is a 120 watt switching supply that provides +5, +12, and -12 volts to the system components. The supply is mounted vertically to the side of the metal chassis and supplies power to the system through a cable which connects to the front plane board and a separate connector on the power supply connects to a ribbon cable to provide +12V to the optional thermal printer (TPM). An overview of the supply is provided in Section 3.

SWEEP PCA. The sweep electronics (assembly 45600-60004) used for HP 150 is the same sweep used in the HP 120 computer system. This board also mounts vertically to the side of the metal chassis and interfaces to the digital logic on the video board via a ribbon cable to the front plane board. This sweep provides focus, brightness, vertical and horizontal centering controls at the rear of the unit. Coarse brightness, horizontal width and vertical size controls are accessible after removal of the shroud of the system. Details about this board as well as the yoke and CRT are found in Section 3.

Digital Logic Boards

There are six PCAs containing primarily digital logic which create the HP 150 hardware personality.

PROCESSOR PCA. The processor board (45611-60002) houses the 8088 microprocessor, the heart of the system. The bulk of the I/O components such as HP-IB controller, keyboard and touchscreen controller, real-time clock, and datacomm are on this board. The processor board interfaces to the rest of the system boards through 96 pin, 60 pin, and 30 pin connectors. The processor board connects to the front plane through the 96 pin connector. The 60 and 30 pin connectors are physically mounted on top of the processor board, and connect the mezzanine memory and datacomm PCAs, respectively. See Section 3 for a discussion on the processor board.
KEYBOARD PCA. The keyboard is detached from the HP 150 unit and a cable connects it to a 6 pin phone jack on the processor board. See Section 3 for more keyboard information.

MEZZANINE MEMORY PCA. The mezzanine memory board (45611-60006) is one of two boards mounted on the processor board in a mezzanine position beneath the video board. The memory board contains ROM, dynamic RAM, CMOS RAM, and indicator LEDs. A 60 pin connector is used to interface the memory and processor boards. See Section 3 for detailed memory board information.

RS232C/422 DATACOMM PCA. The RS232C/422 board (45611-60015) is the other board in a mezzanine position. It connects to the processor board through a 30 pin connector. The board has provision for both asynchronous and synchronous communications and can meet RS232C or RS422 communications standards. Details on this board are in Section 3.

FRONT PLANE PCA. The front plane board (45611-60005) provides the interconnect for the processor, video, touchscreen, sweep, TPM, and accessory boards. The front plane also serves as a conduit for power to the various PCAs from the power supply. Section 3 describes the front plane in more detail.

VIDEO PCA. The alphanumeric and graphics displays are generated by the video subsystem, the core of which is the video board (45611-60003). The video board design is based on an SMC9007 display controller and a custom 40 pin gate array for generation of the alpha and graphics displays respectively. This board is covered in Section 3.

TOUCHSCREEN PCA. The touchscreen board (45611-60001) is actually a mixture of analog and digital circuitry. The 8088 interfaces to the touchscreen via the 8041A keyboard/touchscreen controller. A 10 conductor ribbon cable connects the touchscreen to the front plane where information is in turn exchanged with the processor board. Details on the touchscreen are in Section 3.
This section provides detailed information on the hardware subsystems of the HP 150. Design hints on the optional accessory card are discussed here also. Subsystem discussion includes the front plane, processor, video, keyboard, touchscreen, datacomm, and memory.
CONTENTS

Front Plane
   Thermal (Integral) Printer Interface 3-1
   Front Plane Connectors and Signals 3-2
   Video Board Front Plane Connector 3-4
   Accessory Front Plane Connectors 3-5
   Sweep Board Connector 3-6
   Touchscreen Connector 3-6
   Integral Printer Connector 3-7
   Power Supply Connector 3-7
   Front Plane Signal Descriptions 3-7

Processor Subsystem 3-10
   Overview 3-10
   Memory and I/O Mappings 3-11
   Processor Board Block Diagram 3-11
   Reset, CMOS Power, and Test Strap Logic 3-13
   Reset Logic. 3-13
   CMOS Power Circuit. 3-13
   Test Strap Logic. 3-13
   Microprocessor System Architecture 3-13
   Bus Cycles 3-14
   System Timing and Control Logic 3-14
   Clock Generator. 3-14
   Wait State Generation. 3-15
   GO Generator. 3-16
   Interrupt Controller 3-18
   I/O Devices 3-19
   I/O Decoding. 3-20
   Keyboard and Touchscreen Controller. 3-20
   Datacomm. 3-21
   HP-IB Controller and Interface. 3-21
   Real Time Clock. 3-21

Video Subsystem 3-23
   Video Technology and Display Format 3-23
   Raster Scan. 3-23
   Video Frame Format. 3-24
   Alpha Screen Format. 3-26
   Displaying Alpha Characters 3-27
   Alpha Character Cell Format. 3-28
   Alpha Video Enhancements. 3-30
   Graphics Display. 3-31
   Video Board Overview 3-32

Keyboard and Touchscreen Subsystem 3-37
   Keyboard 3-37
   Electrical Interface. 3-37
   Keyboard Operation. 3-37
   Touchscreen 3-38
   Mechanical Description. 3-38
   Specifications 3-40
   Timing 3-40
   Interface Description. 3-41
Resolution Versus Number of Pairs. 3-42
Pair Address to Row, Column No. Conversion. 3-42
8041 Keyboard/Touchscreen Scanner 3-44
Block Diagram. 3-44
Status Register (I/O Port 0019H). 3-45
Initialization. 3-45
8041 Commands (I/O Port 0019H). 3-46
Keyboard and Touchscreen Data Input (I/O Port 0018H). 3-49

Datacomm Subsystem
General Description 3-50
RS232C/422 Datacomm Module Connector 3-50
Baud Rate Generator 3-51
Multi-Protocol Controller 3-52
Communications Interface Circuitry 3-53

Mezzanine Memory Subsystem
Mezzanine Memory PCA 3-55
PCA Overview 3-55
Connector Signals 3-57
ROM 3-59
ROM Decoding. 3-59
Wait State Disable. 3-59
ROM Timing. 3-60
Slot Selection Generation 3-60
CMOS RAM 3-61
CMOS Decoding and Access. 3-62
CMOS Power. 3-62
LEDs 3-62
LED Decoding. 3-62
LED Register Reset. 3-62
Dynamic RAM 3-63
Decoding. 3-63
Dynamic RAM Refresh. 3-63
PCA Configuration. 3-63

Accessories Subsystem
Accessory Hardware Design Guidelines 3-64
Mechanical Specifications 3-64
Power Requirements 3-64
Thermal Limits 3-65
Accessory Signal Loading Restrictions 3-65
Signal Timing Diagrams 3-67
Accessory Front Plane Connector 3-70
Accessory Connector Signal Descriptions 3-71

Accessory Card Hardware and Electrical
Electrical Design 3-73
Helpful Design Hints 3-73
General Schematic Discussion 3-73
Tranceiver Schematic Discussion 3-74
Mechanical Design 3-76
Helpful Design Hints 3-76
I/O Panel Design 3-76
List of Vendors 3-77
I/O Panel Paint Specifications 3-78
Drawings 3-80
FRONT PLANE

The front plane, or mother board, of the HP 150 system is the board that provides the interconnection of the various components of the system. The processor, video, and two optional accessory boards plug into connectors on the front plane. Four cable connectors interface the power supply, sweep, touchscreen, and optional TPM to the rest of the system via the front plane.

The front plane is used primarily to provide a communications path from the processor board to the video and accessory boards. There is some digital logic on the front plane which is used for transferring bytes from the processor to the TPM. This circuitry is described below. Power for each of the boards comes from the power supply via the front plane. Video signals are transmitted to the sweep via the front plane and timing signals for the touchscreen are transmitted or received via the front plane.

Thermal (Integral) Printer Interface

The optional thermal printer mechanism (TPM) interfaces to the rest of the system via logic on the front plane and processor boards. The processor sends a data byte to the TPM by writing the byte into a transparent latch on the front plane. A negative going pulse on NTPMWRT is sent causing the latch to receive the data byte. On the positive edge of NTPMWRT (the deassertion of NTPMWRT) the byte is latched into the latch. Also, a D flip-flop on the front plane is clocked which results in NSTR (JF7-15) being asserted (goes low). The assertion of NSTR tells the TPM that a byte is available for it to read. After the TPM reads the byte in the latch, it asserts NACK (JF7-6) by bringing it low. This results in the flip-flop preset inputs going low presetting the two D flip-flops. This makes NSTR go back high in preparation for the next byte to be sent to the TPM by the processor. It also makes the output of the D flip-flop go low which in turn makes NOCINT go low. NOCINT is one of the open collector interrupt lines on the front plane. Asserting NOCINT causes an interrupt to the processor. The interrupt service routine will poll the devices which assert NOCINT to determine which device to service. After receiving an interrupt from the TPM interface logic, the 8088 can send another byte to the TPM. (Note that an interrupt service routine must poll the TPM interface logic last because the poll consists of reading the TPM status which also clears the interrupt at the end of the poll.)

The TPM status can be determined by reading the status register. The NTPMRD signal is asserted (goes low) by the processor board which enables the bus driver on the front plane to drive the data bus. Bit 0=1 if acknowledge has been sent by the TPM in response to a byte sent to the TPM. Bit 1=1 if the TPM is installed and bit 2=1 if paper is present in the TPM. (Bits 3-7 are not used.) The deassertion of NTPMRD (its rising edge) clocks the flip-flop which clears the interrupt generated on NOCINT.
Hardware Subsystems

A read or write to the TPM will cause the processor board signal NTPMSL to go low which results in 6 wait states to be added to the bus cycle.

The TPM connects via a ribbon cable to a connector on the front plane. The 8088 accesses the TPM at I/O address XX30 as described in the Memory and I/O Mapping section of this manual. Note that "XX" can be any (address) value.

Summarizing, a byte of data is sent to the TPM by writing the data at port XX30. The TPM acknowledges the receipt of the data and readiness to accept the next byte of data to be sent by asserting the NOCINT interrupt line. The interrupt handling routine can poll the TPM by reading port XX30 and inspecting the least significant bit. If the bit is set, the TPM has generated an interrupt. The end of the read cycle clears the interrupt if caused by the TPM. PAPER OUT and ONLINE status are also available at the same address.

Front Plane Connectors and Signals

The following paragraphs provide a description of the connectors and the connector signals on the front plane.

Front Plane PCA Connector Layout:
## JF2 - Processor Board Front Plane Connector Pinouts and Signals

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>FPA 0</td>
<td>33. NTPMWR</td>
</tr>
<tr>
<td>2.</td>
<td>FPA 1</td>
<td>34. NPFAIL</td>
</tr>
<tr>
<td>3.</td>
<td>FPA 2</td>
<td>35. NSLOTSEL 2</td>
</tr>
<tr>
<td>4.</td>
<td>FPA 3</td>
<td>36. FPWRST</td>
</tr>
<tr>
<td>5.</td>
<td>FPA 4</td>
<td>37. NOCINT</td>
</tr>
<tr>
<td>6.</td>
<td>FPA 5</td>
<td>38. NOCWAIT</td>
</tr>
<tr>
<td>7.</td>
<td>FPA 6</td>
<td>39. +5V</td>
</tr>
<tr>
<td>8.</td>
<td>FPA 7</td>
<td>40. BIO/-M</td>
</tr>
<tr>
<td>9.</td>
<td>ABUS 8</td>
<td>41. -12V</td>
</tr>
<tr>
<td>10.</td>
<td>ABUS 9</td>
<td>42. NVIDINT</td>
</tr>
<tr>
<td>11.</td>
<td>ABUS 10</td>
<td>43. -12V</td>
</tr>
<tr>
<td>12.</td>
<td>ABUS 11</td>
<td>44. FPDO</td>
</tr>
<tr>
<td>13.</td>
<td>FPA 12</td>
<td>45. GND</td>
</tr>
<tr>
<td>14.</td>
<td>FPA 13</td>
<td>46. GND</td>
</tr>
<tr>
<td>15.</td>
<td>FPA 14</td>
<td>47. FPCLK</td>
</tr>
<tr>
<td>16.</td>
<td>FPA 15</td>
<td>48. GND</td>
</tr>
<tr>
<td>17.</td>
<td>FPA 16</td>
<td>49. GND</td>
</tr>
<tr>
<td>18.</td>
<td>FPA 17</td>
<td>50. BIO/-M</td>
</tr>
<tr>
<td>19.</td>
<td>FPA 18</td>
<td>51. +12V</td>
</tr>
<tr>
<td>20.</td>
<td>FPA 19</td>
<td>52. FPDT/-R</td>
</tr>
<tr>
<td>21.</td>
<td>FPD 0</td>
<td>53. +12V</td>
</tr>
<tr>
<td>22.</td>
<td>FPD 1</td>
<td>54. SHOLDA</td>
</tr>
<tr>
<td>23.</td>
<td>FPD 2</td>
<td>55. FPWRT</td>
</tr>
<tr>
<td>24.</td>
<td>FPD 3</td>
<td>56. N.C.</td>
</tr>
<tr>
<td>25.</td>
<td>FPD 4</td>
<td>57. FPWRD</td>
</tr>
<tr>
<td>26.</td>
<td>FPD 5</td>
<td>58. +5V</td>
</tr>
<tr>
<td>27.</td>
<td>FPD 6</td>
<td>59. N.C.</td>
</tr>
<tr>
<td>28.</td>
<td>FPD 7</td>
<td>60. GND</td>
</tr>
<tr>
<td>29.</td>
<td>FPNSSO</td>
<td>61. TSCLK</td>
</tr>
<tr>
<td>30.</td>
<td>NSLOTSEL1</td>
<td>62. GND</td>
</tr>
<tr>
<td>31.</td>
<td>BATV</td>
<td>63. N.C.</td>
</tr>
<tr>
<td>32.</td>
<td>BATV</td>
<td>64. FULLMEM</td>
</tr>
</tbody>
</table>

### Pictorial view of processor board front plane connector with pin assignment (as viewed from component side of front plane board)
## Hardware Subsystems

### Video Board Front Plane Connector

#### JF1 - Video Front Plane Connector Pinouts and Signals

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FPA 0</td>
<td>36</td>
<td>+5V</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>FPA 1</td>
<td>37</td>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>FPA 2</td>
<td>38</td>
<td>FPNRST</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>FPA 3</td>
<td>39</td>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>FPA 4</td>
<td>40</td>
<td>NOCWAITE</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>FPA 5</td>
<td>41</td>
<td>+5V</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>FPA 6</td>
<td>42</td>
<td>BIO/-M</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>FPA 7</td>
<td>43</td>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>ABUS 8</td>
<td>44</td>
<td>NVIDIAINT</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>ABUS 9</td>
<td>45</td>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>ABUS 10</td>
<td>46</td>
<td>FP0</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>ABUS 11</td>
<td>47</td>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>FPA 12</td>
<td>48</td>
<td>FPCLK</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>FPA 13</td>
<td>49</td>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>FPA 14</td>
<td>50</td>
<td>+5V</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>FPA 15</td>
<td>51</td>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>FPA 16</td>
<td>52</td>
<td>BIO/-M</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>FPA 17</td>
<td>53</td>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>FPA 18</td>
<td>54</td>
<td>FPDT/-R</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>FPA 19</td>
<td>55</td>
<td>N.C.</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>FPD 0</td>
<td>56</td>
<td>+5V</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>FPD 1</td>
<td>57</td>
<td>N.C.</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>FPD 2</td>
<td>58</td>
<td>FP0RD</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>FPD 3</td>
<td>59</td>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>FPD 4</td>
<td>60</td>
<td>(RESERVED)</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>FPD 5</td>
<td>61</td>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>FPD 6</td>
<td>62</td>
<td>+5V</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>FPD 7</td>
<td>63</td>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>+5V</td>
<td>64</td>
<td>NVSYNC</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>+5V</td>
<td>65</td>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>+12V</td>
<td>66</td>
<td>NHSYNC</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>+12V</td>
<td>67</td>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>+12V</td>
<td>68</td>
<td>NHB</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>BATV</td>
<td>69</td>
<td>NFB</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>BATV</td>
<td>70</td>
<td>GND</td>
<td></td>
</tr>
</tbody>
</table>

#### JF1

```
+-------------------------------------------------+
| 35 | + + ...                                      | ... + + | 1 |
| 70 | + + ...                                      | ... + + | 36 |
+-------------------------------------------------+
```

Pictorial view of processor board front plane connector with pin assignment (as viewed from component side of front plane board)
Accessory Front Plane Connectors

The accessory boards plug into the bottom slots of the card cage. The system signals available to the modules are listed below.

**JF3/4 Accessory Board Front Plane Connector Pinout and Signals**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
<th>Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>FPA 0</td>
<td>36.</td>
<td>(RESERVED) *</td>
</tr>
<tr>
<td>2.</td>
<td>FPA 1</td>
<td>37.</td>
<td>NPFAIL</td>
</tr>
<tr>
<td>3.</td>
<td>FPA 2</td>
<td>38.</td>
<td>(RESERVED) *</td>
</tr>
<tr>
<td>4.</td>
<td>FPA 3</td>
<td>39.</td>
<td>FPNRST</td>
</tr>
<tr>
<td>5.</td>
<td>FPA 4</td>
<td>40.</td>
<td>+5V</td>
</tr>
<tr>
<td>6.</td>
<td>FPA 5</td>
<td>41.</td>
<td>GND</td>
</tr>
<tr>
<td>7.</td>
<td>FPA 6</td>
<td>42.</td>
<td>GND</td>
</tr>
<tr>
<td>8.</td>
<td>FPA 7</td>
<td>43.</td>
<td>NOCINT</td>
</tr>
<tr>
<td>9.</td>
<td>ABUS 8</td>
<td>44.</td>
<td>GND</td>
</tr>
<tr>
<td>10.</td>
<td>ABUS 9</td>
<td>45.</td>
<td>NOCWAIT</td>
</tr>
<tr>
<td>11.</td>
<td>ABUS 10</td>
<td>46.</td>
<td>(RESERVED) *</td>
</tr>
<tr>
<td>12.</td>
<td>ABUS 11</td>
<td>47.</td>
<td>BIO/-M</td>
</tr>
<tr>
<td>13.</td>
<td>FPA 12</td>
<td>48.</td>
<td>+5V</td>
</tr>
<tr>
<td>14.</td>
<td>FPA 13</td>
<td>49.</td>
<td>-12V</td>
</tr>
<tr>
<td>15.</td>
<td>FPA 14</td>
<td>50.</td>
<td>NDCOCINT</td>
</tr>
<tr>
<td>16.</td>
<td>FPA 15</td>
<td>51.</td>
<td>GND</td>
</tr>
<tr>
<td>17.</td>
<td>FPA 16</td>
<td>52.</td>
<td>-12V</td>
</tr>
<tr>
<td>18.</td>
<td>FPA 17</td>
<td>53.</td>
<td>SHOLDA</td>
</tr>
<tr>
<td>19.</td>
<td>FPA 18</td>
<td>54.</td>
<td>FPGO</td>
</tr>
<tr>
<td>20.</td>
<td>FPA 19</td>
<td>55.</td>
<td>+5V</td>
</tr>
<tr>
<td>21.</td>
<td>GND</td>
<td>56.</td>
<td>GND</td>
</tr>
<tr>
<td>22.</td>
<td>GND</td>
<td>57.</td>
<td>FPCLK</td>
</tr>
<tr>
<td>23.</td>
<td>FPD 0</td>
<td>58.</td>
<td>(RESERVED) *</td>
</tr>
<tr>
<td>24.</td>
<td>FPD 1</td>
<td>59.</td>
<td>GND / FULLMEM *</td>
</tr>
<tr>
<td>25.</td>
<td>FPD 2</td>
<td>60.</td>
<td>+12V</td>
</tr>
<tr>
<td>26.</td>
<td>FPD 3</td>
<td>61.</td>
<td>GND</td>
</tr>
<tr>
<td>27.</td>
<td>FPD 4</td>
<td>62.</td>
<td>BIO/-M</td>
</tr>
<tr>
<td>28.</td>
<td>FPD 5</td>
<td>63.</td>
<td>+12V</td>
</tr>
<tr>
<td>29.</td>
<td>FPD 6</td>
<td>64.</td>
<td>FPNWRT</td>
</tr>
<tr>
<td>30.</td>
<td>FPD 7</td>
<td>65.</td>
<td>+5V</td>
</tr>
<tr>
<td>31.</td>
<td>GND</td>
<td>66.</td>
<td>FPD!/-R</td>
</tr>
<tr>
<td>32.</td>
<td>FPNRD</td>
<td>67.</td>
<td>GND</td>
</tr>
<tr>
<td>33.</td>
<td>GND</td>
<td>68.</td>
<td>FPNSSO</td>
</tr>
<tr>
<td>34.</td>
<td>+5V</td>
<td>69.</td>
<td>GND</td>
</tr>
<tr>
<td>35.</td>
<td>BATV</td>
<td>70.</td>
<td>NSLOTSELx **</td>
</tr>
</tbody>
</table>

**JF3 and JF4**

```
+-------------------------------------------------+
35 | + + ...                                      70 | + + ...
   | ... + + | 1                                   ...
   | ... + + | 36
+-------------------------------------------------+
```

Pictorial view of option module front plane connector with pin assignment (as viewed from component side of front plane board)
Hardware Subsystems

* Pin 59 is FULLMEM is on JF3 only. Pin 59 is GND on JF4. Connections to this pin and all pins labeled RESERVED should not be made.

** NSLOTSELx will be either NSLOTSEL1 or NSLOTSEL2 depending upon which side of the front plane the module is plugged into. The left side of the card cage (from rear view of the package) holds module 1 and gets the NSLOTSEL1 signal. The right side of the card cage holds module 2 and gets NSLOTSEL2.

Sweep Board Connector

The Sweep board connects to the front plane via a 20 pin connector, JF5, whose signal pins are listed below:

```
+----------+  +----------+
| 20. | +   + | 10. 20. | +5V  10. NHSYNC |
|  | +   + |  | 19. GND  | 9.  +12V |
|  | +   + |  | 18. NHB    | 8.  GND |
|  | +   + |  | 17. GND    | 7.  +12V |
|  | +   + |  | 16. NFB     | 6.  GND |
|  | +   + |  | 15. GND    | 5.  +12V |
|  | +   + |  | 14. SWPNRST | 4.  GND |
|  | +   + |  | 13. -12V   | 3.  GND |
|  | +   + |  | 12. +12V   | 2.  +12V |
| 11. | +   + | 1. 11. NVSYNC | 1.  +5V |
+----------+  +----------+
                 JF5
```

Touchscreen Connector

The Touchscreen connects to the front plane via a 10 pin connector, JF6, whose signal pins are listed below:

```
+----------+
| 1. | +   + | 6. 1.  +12V  6.  +12V |
|  | +   + | 2. TSDATA 7.  +12V |
|  | +   + | 3.  -12V  8.  GND |
|  | +   + | 4. TSYNC    9.  TSCLK |
| 5. | +   + | 10. 5.  GND  10.  GND |
+----------+
                 JF6
```
Integral Printer Connector

The Integral Printer connects to the front plane via a 20 pin connector, JF7, whose signal pins are listed below.

+--------+
1. + + 11. D0 11. D1
   + +  17. NPOR 15. NSTR
   + +  18. NACK 19. POUT
   + +  19. N.C. 20. NONL
   + +  20. N.C.
+--------+

Power Supply Connector

Power to the system comes from the power supply through a 12 pin connector, JF8, on the front plane board. The signal definition is given below:

+-----+
1. NPFAIL
   (Pin removed for polarization)
3. GND
4. -12V
5. GND
6. (N.C.)
7. +12V
8. GND
9. +5V
10. +5V
11. +5V
12. GND
+-----+

Front Plane Signal Descriptions

The following description of the front plane signals reference the connector diagrams shown previously.

FPA 0-7 The lower 8 address bits of the 20 bit address generated by the 8088. These signals are buffered and demultiplexed.

ABUS 8-11 The upper 12 address bits of the 20 bit address.
Hardware Subsystems

FPA 12-19 These signals also are buffered and demultiplexed.

FPDBUS 0-7 Data bus signals from the external data bus. This data path is the means through which data is passed between the processor and the video board or option slot modules or the optional TPM.

FPCLK Buffered 8 Mhz system clock.

FPNRD Buffered -RD signal from the 8088 used to indicate a bus read cycle in progress.

FPNWRT Buffered -WRT signal from the 8088 used to indicate a bus write cycle in progress.

FPNSSO 8088 bus cycle status line. The combination of FPNSSO, BIO/-M, and FPDT/-R allow boards connected to the front plane to completely decode the current bus cycle.

BIO/-M Buffered IO/-M signal from the 8088 used to distinguish memory and I/O bus cycles.

FPDT/-R Buffered DT/-R signal from the 8088 used to indicate direction of data from the 8088 for a given bus cycle.

FPGO Signal which qualifies the address generated by the microprocessor. The address qualification is needed primarily for dynamic RAM circuits which cannot tolerate an assertion of RAS or CAS on a false address. FPGO goes high at the beginning of T2 and goes back low at the beginning of T4 of a bus cycle. FPGO can also be used to terminate a bus write cycle by having its falling edge used to clock data into a register or other device on an accessory device in an option slot. Terminating the writes in this way can provide better hold timing than using FPNWRT.

NOCWAIT This line can be asserted by the option modules or video board through an open collector gate to insert wait states into a bus cycle to provide sufficient time for a bus cycle access.

NVIDIAINT This line is asserted by the video subsystem to generate an interrupt to the processor. This interrupt occurs once every frame. The signal can be monitored by the option slot devices if desired.

NDCOCINT This open collector interrupt signal has the same interrupt priority as the datacomm controller chip on the processor board as it shares the same input to the interrupt controller as the datacomm controller chip does. This input can be asserted by accessories for interrupt servicing.

NOCINT System interrupt signal asserted by a device via an open collector gate. This line can be used by options slots to get processor service.
SHOLDA
Synchronized hold acknowledge signal from the 8088. HOLD is asserted by the mezzanine memory board.

FULLMEM
Indicates which mezzanine memory board option is installed. FULLMEM=0 if 128K RAM and FULLMEM=1 (it is pulled high by a pull up resistor on an installed extension memory board) if 256K RAM is on the board. An accessory should not connect to this signal pin.

NSLOTSEL1
Signal indicating an address within the 64K block allocated to option module 1 has been generated by the 8088.

NSLOTSEL2
Same as NSLOTSEL1 but asserted when address within option module 2 address space is generated.

NFB
Video Full-Bright signal to sweep board.

NHB
Video Half-Bright signal to sweep board.

NHSYNC
Horizontal sync signal from video board used by the processor and sweep boards.

NVSYNC
Vertical sync signal from video to sweep board.

TSCLK
Touchscreen clock signal.

TSDATA
Touchscreen data signal.

TSSYNC
Touchscreen sync signal.

NTPMRD
Signal from CPU to read TPM status.

NTPMWRT
Signal generated by the processor card to handshake bytes of data to the TPM.

FPWRST
Signal generated by processor board at power-on time to initialize logic circuitry.

NPFFAIL
Signal generated by the power supply indicating power supply output level stability.

BATV
These two connector lines carry current from the batteries located on the video board to circuitry requiring battery back-up power such as the CMOS RAM and the CMOS real time clock.
PROCESSOR SUBSYSTEM

Overview

The HP 150 processor board is based on the Intel 8088-2 microprocessor. The microprocessor runs at an 8 MHz clock speed and is configured for operation in the minimum mode. The general architecture of the processor, memory, and I/O is depicted in figure 3-1. As shown in the diagram, the majority of the circuitry is partitioned between three data busses: the memory data bus, the I/O data bus, and the external data bus.

Figure 3-1. Processor Board and Memory Board Block Diagram
Memory and I/O Mappings

The logic on the processor board and the other boards which exchange data with the microprocessor are mapped into the 8088’s memory or I/O space. Section 4 depicts the memory and I/O maps for the system. Note that I/O addresses XX80 through XXFF are available for option module use. Also note that "XX" can be any value.

Processor Board Block Diagram

Besides the 8088 microprocessor, the processor board contains a number of LSI and MSI components which are used to interface with other subsystems, provide communications with external devices, or perform specialized functions. The block diagram in figure 3-2 shows the basic architecture of the board. This section of the manual will detail the various subsections and components of the processor board.
Hardware Subsystems

Figure 3-2. Processor Board Block Diagram
Reset, CMOS Power, and Test Strap Logic

This set of miscellaneous circuits provide functions of an ancillary or support nature to the more visible functions of the board to be described later.

RESET LOGIC. The power-fail signal from the power supply (NPFAIL, J1-34) is used by the system as a power-on-reset and a CMOS device standby operation mode switch. NPFAIL is low when power is first applied to the system and goes high no sooner than 75 msec after the power supply output voltages have stabilized at their specified values. When NPFAIL goes low due to the power being disconnected from the power supply, it does so at least 500 usec before the power supply output voltages go out of regulation.

CMOS POWER CIRCUIT The CMOS power circuit regulates current use by real time clock and configuration CMOS memory between the batteries and the power supply while the unit is switched on.

TEST STRAP LOGIC. If the test strap input is held low during execution of manufacturing self-test, a failing test will be iterated until the failure ceases or the input is brought high. This is intended to be a convenience for servicing purposes.

Microprocessor System Architecture

The system microprocessor is an 8088. The clock frequency the 8088 is run at is 8 Mhz. The 8088 is configured in the "minimum mode." As it is not the intent of this document to review microprocessor basics, please refer to the Intel 8088 data sheets and application notes for fundamentals on 8088 operation.
Hardware Subsystems

BUS CYCLES

Address Generation. At the beginning of a bus cycle (memory or I/O access), the 8088 places the memory or I/O address on its address and address/data pins. The address is multiplexed with the data and some of the status information so the processor board uses transparent latches to latch and buffer the address generated at the beginning of the cycle. Address bits A8-A11 are not multiplexed so a non-latching buffer is used for buffering those four signals. ALE (U211 pin 25) is the signal used to latch the address into the latches.

Data Transactions. The 8088 does all data transactions through one of three bus transceivers. All the mezzanine memory board components are isolated by a transceiver located on the memory board. All the I/O devices (real time clock, datacomm, baud rate generator, keyboard/touchscreen controller, HP-IB controller, etc.) are on the I/O bus which is separated from the 8088 by another transceiver on the processor board. Any access not to the mezzanine memory or the I/O bus will default to the front plane.

Status Generation. The 8088 generates status information indicating the type of bus cycle in progress. Control signals based upon these are generated by the processor and are buffered and sent throughout the system to allow circuitry to respond properly to read, write, memory, or I/O cycles.

HOLD State. The 8088 will be put into a HOLD state by the mezzanine memory approximately every 56 usec during dynamic RAM refresh.

System Timing and Control Logic

CLOCK GENERATOR. The 8284A clock generator is used to provide the system clock for the HP 150 hardware. A 24 MHz crystal provides the reference frequency used by the 8284. The 8284 creates an 8 MHz clock which nominally is high 33% and low 66% of the 125 nsec period. The 8284 also divides the 8 MHz clock by 2 to form the 4 MHz signal used as the system clock for the 7201 datacomm controller, the 8116T baud rate generator, the 9914A HP-IB controller, and the 8041A keyboard and touchscreen controller.

The 8284 also provides a reset output pin which is used as a basis for the reset signals throughout the system hardware.

The -F/C input (pin 13) of the 8284 is at a logic 0 state which selects the crystal oscillator as the source frequency for the 8284.

The 8284 is a part of the wait state insertion logic. The 8284 is configured in the synchronous mode which affects some of the timing constraints the wait state generator logic has. Wait state generation is described in the next section.
WAIT STATE GENERATION. The basic bus cycle for the 8088 consists of four 125 nsec clock periods during which the memory of I/O address is generated by the CPU followed by a transmission of data to or the reception of data from a memory or I/O device. The cycle can be extended by adding "wait states" to the basic 4 clock cycle. The standard bus cycle is shown in figure 3-3.

```plaintext
CLK  (8 Mhz)  |   |   |   |____|
ALE  |   |____|____|
READY  

Figure 3-3. Standard Bus Cycle with no Wait States
```

The four clock periods are labeled T1 through T4 for reference purposes. The 8088 samples the READY input during the low clock period of T3. If it is high at that time, no extra wait states are added. If it is low, a wait state is added as shown in figure 3-4. READY is sampled during the low clock period of every wait state. Whenever it is sampled high, the wait state is exited and the bus cycle goes into the T4 state.

```plaintext
-- T2 --|--- T3 ---|--- TW ---|--- TW ---|--- T4 ---|
CLK  (8 Mhz)  |   |   |____|____|
READY  

Figure 3-4. Bus Cycle with Two Wait States
```

The READY input to the 8088 is driven by the RDY output (pin 5) of the 8284. The 8284 has two inputs, RDY1 and RDY2 (pins 4 and 6), by which wait states are controlled. Normally, wait states are added by RDY1 (NWAIT). RDY1 is sampled at the falling edge of each clock, so effectively, the READY input of the 8088 is controlled by what the 8284 samples at its RDY1 input just prior to the beginning of T3 or just prior to the beginning of each wait state.

Some processor board logic exists to add a specific number of wait states to certain bus cycles. Two wait states are added to the bus cycle when the 8814 or 8041 or 8116 are accessed by the 8088 and 9 wait states are added when the real-time clock is accessed.
A minimum of one wait state is added to every bus cycle when jumper W1 is installed. (Access of devices requiring more wait states get the additional wait states needed but one wait state appears in bus cycles on accesses of devices whose decoding logic isn't tied into the wait state generator.) Installing W2 causes no wait states to be added for ROM access only.

Wait states may be inserted by other logic boards by asserting the NOCWAIT line that is routed to the front plane option slots and video board connector. As long as NOCWAIT is low, wait states will be executed by the 8088. Since the NOCWAIT input goes into a synchronizing flip-flop the wait states added by an option or the video board must be anticipated in a timely fashion. The timing diagram in figure 3-5 illustrates wait state insertion and deletion by an option card.

\[ \text{Figure 3-5. Option Wait State Insertion} \]

To put in an additional wait state (there is one automatic wait state for all over the front plane bus cycles) the option must assert NOCWAIT no later than 2 clocks prior to the time the wait state is desired. Wait states are exited 2 clocks after the next falling edge of CLK from the point where NOCWAIT is deasserted. If an option is designed whose enable is qualified by FPPO (described in the next section) wishes to put in a wait state, it must assert NOCWAIT within: 

\[
T_{\text{clk}} - T_{\text{fpgo}} - T_{\text{s74}} - 2T_{\text{fp}} = 125 - 38 - 3 - 20 = 64 \text{ nsec}
\]

of the falling edge or beginning of T3. (\(T_{\text{clk}} = \) clock period, 125 nsec; \(T_{\text{fpgo}} = \) front plane GO delay; \(T_{\text{s74}} = 74S74\) data setup time, \(T_{\text{fp}} = \) front plane propagation delay.)

**GO GENERATOR.** The purpose of the GO generator is to provide a signal to qualify the address generated by the 8088 on its address pins. The GO generator indicates the beginning of T2, end of T1 within a bus cycle and the beginning of T4. The address qualification is needed primarily for dynamic RAM circuits which cannot tolerate an assertion of RAS or CAS on a false address. Another feature of the GO signal is that its deassertion is useful for providing better hold margin timing on bus write cycles. A state machine is used to implement the GO generator. Basically, the machine inspects the NWAIT signal and ALELCH. When ALELCH goes high, the GO generator brings GO high at the next negative edge of the clock (beginning of T2). NWAIT is monitored until NWAIT goes high. Two clocks later, GO is brought low (at the beginning of T4) indicating the end of the bus cycle.
The machine algorithm is:

![Diagram](image-url)

**Figure 3-6. GO State Machine Algorithm**

The HP 150 utilizes an 8284 clock generator (READY generator) operating in synchronous mode. Therefore, the RDY1 input of the 8284 (pin 4) must be stable 35 nsec before each falling edge of the clock. The GO generator circuit requires NWAIT to be stable 20 nsec prior to the falling edge. Thus, meeting the 8284 specification meets the GO generator NWAIT timing requirement.
The timing relationship between GO and the other system timing is portrayed in figures 3-7 and 3-8.

![Figure 3-7. Standard Bus Cycle with no Wait States with GO Timing.](image)

![Figure 3-8. Bus Cycle with One Wait State With GO Timing.](image)

The skew between GO and the falling edge of the clock should not exceed 20 nsec.

**Interrupt Controller**

The 8088 is capable of processing two types of hardware interrupts. One type is the non-maskable interrupt (NMI) which is processed when the NMI input to the 8088 makes a low to high transition. This occurs when a hard reset key sequence (CTRL-SHIFT-RESET) is issued by the user. The 8088, after getting an NMI, will use the values in memory locations 8,9,A,B for new code segment and instruction pointer values. This will cause the 8088 to vector to an interrupt service routine.
The other type of interrupt processed by the 8088 is standard or maskable interrupts. This type of interrupt is invoked by the INTR input of the 8088 going high. In response the 8088 will lower the NINTA or interrupt acknowledge output twice. The second time NINTA goes low, the 8259A interrupt controller will place an 8 bit vector on the data bus. This vector will be a value from 0 - OFFH. The 8088 will start at the memory location equal to 4 times the vector and load the content of that location and the next three locations into the instruction pointer and code segment registers and subsequently branch to the interrupt routine.

The 8259 has 8 inputs through which it accepts incoming interrupts from various sources. When an interrupt request occurs by one or more of the inputs IR0 - IR7 going high, it raises the INTR input to the 8088 high. The 8088 in turn asserts NINTA two times. The 8259 will select one of the eight interrupt input sources for processing and when the 8088 sends the second pulse on NINTA, the 8259 will place a vector on the bus that corresponds to the interrupting device that is to be serviced. Should several interrupt requests come into the 8259 simultaneously, the 8259 chooses which one is to be processed first and which ones subsequently.

The interrupts coming into the 8259 are prioritized as:

Highest priority IR0 Video
   IR1 Datacomm (7201 and front plane interrupts)
   IR2 +5V (Video second level tasks)
   IR3 Keyboard and Touch Screen
   IR4 Front plane interrupts (lower level)
   IR5 HP-IB
   IR6 (not used)

Lowest priority IR7 Real-Time Clock

The 8259 can be programmed to handle incoming interrupts in a number of different ways. The system firmware puts it in the iAPX 86 mode, level-triggered and non-buffered modes. The 8088 programs the 8259 by writing various data bytes to it over the I/O bus. The 8259 can be programmed to ignore certain interrupts, handle incoming interrupts in a particular order, or even to work as a polling device if desired.

The various ways the 8259 can be used are described in the 8259 data sheet and application note available from Intel.

I/O Devices

The I/O bus has eight functional modules which exchange data with the microprocessor. These modules are the 8259A interrupt controller, the MM58167A real-time clock, the 8116T baud rate generator, the 8041A microcomputer which controls the keyboard and touchscreen, the 9914A HP-IB controller, 7201 datacomm controller, and logic pertaining to datacomm on the fixed port (port 2) as well as the mezzanine datacomm board (port 1). Figure 3-9 illustrates the I/O bus modules.
Hardware Subsystems

I/O DECODING. One or more I/O ports are assigned to each device or module on the I/O bus to provide the required communication path with the CPU. The I/O map (presented in Section 4) lists the port addresses assigned to each device. An I/O bus cycle (one that is not an interrupt acknowledge cycle) is executed by the 8088 with the IO/-M signal high, the NSS0 signal high and the DT/-R signal low, or with IO/-M high, NSS0 low and DT/-R high. During the I/O cycle the lower 16 bits of the address bus carry the port address being accessed. An OR and a NAND gate monitor BIO/-M, NSS0, and DT/-R to see if an I/O cycle is occurring. A 3-line-to-8-line decoder with a NOR gate decode the address on the address bus and enable the 7201, the 9914, the 8041, the 8259, the 8116, or logic in one of the datacomm ports when an I/O bus cycle is issued by the 8088 to access one of these devices. A NAND gate, a NOR and an inverter are used to decode the addresses used by the real-time clock chip.

KEYBOARD AND TOUCHSCREEN CONTROLLER. The interface between the 8088 and the keyboard is provided by a single chip microcomputer, an Intel 8041. Additionally, the 8041 provides the interface to the touchscreen.

The 8041 interfaces to the 8088 over the data bus. Two I/O ports are devoted to it, ports 18H and 19H. The 8088 can send various commands to the 8041 while the 8041 can send information such as a key number or a touchscreen report.
Keyboard: The interface between the 8041 and the keyboard depends on the synchronization of a hardware counter on the keyboard, and a software counter in the 8041. With each key clock the 8041 sends to the keyboard, it increments the software counter. It then checks the data line for status of the currently addressed key. The contents of the software counter specify the correct key address.

The KBDSYNC line provides a reset mechanism allowing the 8041 to put the keyboard counter into a known state (count = 0). Thus the interface to the keyboard is provided by three lines: KBDCLK from 8041, KBDSYNC from 8041, and KBDDATA from keyboard.

Touchscreen: The interface to the touchscreen is essentially identical to that for the keyboard. The most significant difference is that the touchscreen drives the TSSYNC line. The 8041 must send clocks until the touchscreen activates the TSSYNC line to determine the state of the touchscreen counter. This provides a positive indication as to whether the touchscreen is present. Again, the interface is over three lines: TSCLK from the 8041, TSSYNC from the touchscreen, and TSDATA from the touchscreen.

Bell: The 8041 also provides the drive for the system bell. The on-board timer circuit is used to select the frequency of the bell tone.

Circuitry between the 8041 and the keyboard converts the TTL levels from the 8041 to 12 volt reference levels used on the keyboard. This is also true for the touchscreen.

For a more detailed explanation of how the 8041 interfaces to the keyboard and touchscreen see "8041 Keyboard/Touchscreen Scanner", which is discussed later in this manual.

DATACOMM. The HP 150 hardware provides two independent datacomm ports. These ports provide a full complement of signals and a large degree of flexibility for the system firmware to take advantage of. A description of these datacomm facilities is discussed later in this manual under "Datacomm Subsystem".

HP-IB CONTROLLER AND INTERFACE. The HP-IB controller used in HP 150 is the TMS9914A which is capable of handling the talker and listener functions, parallel and serial poll functions, and data handshaking functions required to implement the HP-IB standard protocol. SN75160 and SN75161 bus transceivers are used to interface the controller to the HP-IB bus.

The 9914 system clock is the 4 Mhz square wave generated by the 8284 (PCLK). The 9914 has 8 I/O addresses assigned to it by which the CPU accesses the controller’s internal 14 registers to do HP-IB transactions.

The 9914 can also generate an interrupt to the system at interrupt level 5 to receive CPU attention if the processor has enabled the interrupt facilities. -INT, the interrupt output signal is an open-drain active low signal which is pulled up and inverted for 8259 interrupt handling.
REAL TIME CLOCK. The HP 150 uses the MH58167 CMOS real time clock (RTC). This is a battery backed up component as is the CMOS RAM on the mezzanine memory board. When the main system power is turned off, the 3V battery and the associated circuitry maintains approximately 2.5V at the power supply pins of the CMOS parts.

The '58167 is a versatile chip. It interfaces to the system with an 8 bit bus. The RTC keeps track of the month, day of month, and day of the week as well as the time in hours, minutes, seconds, tenth seconds, hundredths and milliseconds. It is capable of producing repeated interrupts at the rates of 1/month, 1/week, 1/day, 1/hour, 1/minute, 1/second and 10/second. An alarm feature allows an interrupt to be sent to the system at a designated time.

Several components external to the chip are used in the oscillator circuit for the RTC. These include a variable capacitor which must be tuned for accurate time keeping with this circuit. The suggested way to calibrate the variable capacitor is to connect a frequency counter to the interrupt output of the RTC and program the chip to produce a repetitive 1 Hz interrupt. Adjust C28, the variable capacitor, until the period of the pulse is as close to 1 second as it will adjust to.

A number of components are used to create the final signals used to drive the -RD and -WR inputs of the RTC. The '58167 has some rather unusual address and data setup and hold times that require the creation of special -RD and -WR signals to meet the RTC's timing constraints.

NOTE

The millisecond register of the RTC is not very accurate. It is not recommended for use in critical time applications.

The RTC interrupt status register cannot be polled! The status register must be interrogated after receiving an interrupt from the RTC. This is due to a race condition that exists in the chip that can cause an interrupt to be permanently masked if the status register is interrogated at the time the interrupt was about to occur. This implies that the RTC must not have its interrupt output shared in an open collector fashion with other devices unless it is always the last device to be polled in an interrupt service routine.

The RTC can continue to drive the data bus as long as 250 nsec after the -RD input is brought back high. A restriction that we have in the HP 150 system is that a read from the RTC cannot be followed immediately by a write to any device on the I/O bus.
VIDEO SUBSYSTEM

The HP 150 Video Board provides the alphanumeric and graphics displays. This section of the manual describes in detail the workings of the board. The video board interfaces to the rest of the system through the front plane board. The interface signals and connector pinouts are described earlier in this section.

Video Technology and Display Format

RASTER SCAN. The HP 150 uses a RASTER SCAN display technology in a non-interlaced mode at a 60Hz refresh rate. The video is generated by scanning the CRT (cathode ray tube) with an electron beam. Electrons striking the phosphorous coating on the inside of the tube cause it to glow. Characters are displayed using dot patterns traced out by the electron beam.

The HP 150's display system consists of:

1. Video Board
2. Sweep Board
3. CRT

The Video Board stores alphanumeric and graphics information in display RAMs. It then converts this information into a serial dot stream along with the necessary control signals and sends them to the Sweep Board.

The Sweep Board is responsible for controlling the CRT. It transforms the digital signals it receives from the Video Board into analog signals necessary for manipulating the electron beam.

The CRT acts as a transparent screen, onto which information is displayed. Electrons are generated in the back of the tube called the "electron gun." They are accelerated towards the front by the high voltage placed on the inside of the tube by the Sweep Board. The horizontal and vertical direction of the beam is regulated by a set of magnetic coils known as the "Yoke." The Yoke receives analog control signals from the Sweep Board also. See figure 3-10 for a block diagram description of the video system.
VIDEO FRAME FORMAT. Figure 3-11 shows the CRT pattern for the ALPHA display for one frame time. Part a) of the figure shows the beginning of the frame when the electron beam is traced from the upper left portion of the screen to the lower right. Part b) shows the Vertical Retrace, when the beam is returned to the upper left of the screen, thus getting ready for the start of the next frame. For the sake of clarity, the following section refers to ALPHA display only. The relationship between Graphics and Alpha will be explained later.

Looking at figure 3-11, the following events take place during an ALPHA display in one frame time:

1. The beam starts out at the top of the tube in the "Start Scan Lines" area. The beam will not be allowed to turn on for the first 12 scan lines. This area is blanked so that the beam control signals coming from the Sweep Board have a chance to settle down before active video display.

2. On the 13th scan line, the beam reaches the active display area and video data is displayed. Note that horizontal retraces are necessary to bring the beam back to the start of the next scan line. During this time, however, the beam is turned off.

3. After all of the 378 ALPHA scan lines have been displayed we enter the 6 "Extra Scan Line" region, where the beam is blanked.
4. We reach the end of the "Extra Scan Lines" and the Sweep Board receives a Vertical Drive pulse, from the Video Board, telling it to do a vertical retrace. In part b), we see the beam returning to the top of the screen. The beam is blanked during the vertical retrace, so that we won't see the zig-zag lines on the screen. (Note that horizontal retraces continue while this is taking place).

One complete frame includes the following:

- 12 Start Scan Lines
- 378 Active Scan Lines
- 6 Extra Scan Lines
- 19 Vertical Retrace Scan lines

---------------------------------------------
415 = scan lines per frame

Each scan line is divided into its active (displayable) part, and the blanked (horizontal retrace) part as follows:

- 80 displayable character times
- 35 horizontal retrace character times

---------------------------------------------
115 = visible and non-visible characters per scan line
ALPHA SCREEN FORMAT. Observing part a) of figure 3-12 we see that the alpha screen consists of a matrix of characters 27 rows X 80 columns. From here on, we will refer to them as Character Rows and Character Columns. Following is a table describing the different areas of the screen:

<table>
<thead>
<tr>
<th>Description</th>
<th>Row Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>24 user rows</td>
<td>1-24</td>
</tr>
<tr>
<td>2 soft key rows</td>
<td>25-26</td>
</tr>
<tr>
<td>1 status row</td>
<td>27</td>
</tr>
</tbody>
</table>

27 rows = total displayable rows in one frame
Figure 3-12. Alpha Character Matrix and Cell

DISPLAYING ALPHA CHARACTERS

- The dot pattern for every character is stored in a Character ROM (Read Only Memory).

- Characters are displayed one scan line at a time. For example, in order to display one character row (80 characters), the following must be done:

  1. Fetch the 1st scan line pattern of char #1 from the Character ROM, then send it out to be displayed.

  2. Fetch the 1st scan line of the 2nd character, and send to the display.

  3. Repeat this until scan line 1 of all the 80 characters has been displayed.
4. Now perform 1, 2, 3 again, but with scan lines 2 and again with 3, etc., up to scan line 14 which completes the character row.

o To display the entire screen of 27 character rows the above steps 1-4 would be performed for each character row, a total of 27 times.

ALPHA CHARACTER CELL FORMAT. Part b) of figure 3-12 indicates the dot matrix which makes up each character, known as the character cell. There are 14 scan lines of 9 dot columns in each cell.

The HP 150 character cell has the following components:

<table>
<thead>
<tr>
<th>Row</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x x x x x x x x</td>
</tr>
<tr>
<td>2</td>
<td>x o o o o o o o x</td>
</tr>
<tr>
<td>3</td>
<td>x o o o o o o o x</td>
</tr>
<tr>
<td>4</td>
<td>x o o o o o o o x</td>
</tr>
<tr>
<td>5</td>
<td>x o o o o o o o x</td>
</tr>
<tr>
<td>6</td>
<td>x o o o o o o o x</td>
</tr>
<tr>
<td>7</td>
<td>x o o o o o o o x</td>
</tr>
<tr>
<td>8</td>
<td>x o o o o o o o x</td>
</tr>
<tr>
<td>9</td>
<td>x o o o o o o o x</td>
</tr>
<tr>
<td>10</td>
<td>x o o o o o o o x</td>
</tr>
<tr>
<td>11</td>
<td>x o o o o o o o x</td>
</tr>
<tr>
<td>12</td>
<td>x s s s s s s s x</td>
</tr>
<tr>
<td>13</td>
<td>c u c u c u c c</td>
</tr>
<tr>
<td>14</td>
<td>c c c c c c c c</td>
</tr>
</tbody>
</table>

Note:
- x = inter row/column spacing
- o = character dots
- s = space
- cu = top half of double cursor, underline or underhang
- c = bottom half of double cursor or inter-row spacing.

Basic character cell contains the main body of the character called the Character Font. (All HP 150 characters are in a 7x10 font.)

Inter-row spacing is accomplished by blanking the first and last columns of the character cell.

Scan line 11 is called the Print Line, on which lower and upper case characters are positioned.

Inter-column spacing is ensured by not putting character information in rows 1 and 14. (Except on certain Roman Extention characters where ascenders include row 1.)

Underline occupies row 13

Descenders reach down to row 13
There are two types of cursors on the HP 150:

- The double scan line cursor occupies rows 13 and 14. It is a positive cursor, which means that it is logical OR-ed with whatever dots that may try to occupy the same row.

- The "Blob" cursor takes up the entire character cell. It is an invertible cursor, which means that it will invert whatever dot pattern that may be inside the cell.

- Both cursors are blinking types.

Refer to figure 3-13 for a character cell example. Here we see the letter "A." As stated above, it takes up only a 7x10 space, leaving room for inter-character spacing. A few of the dots do not start at the beginning of the designated dot position. These are shifted over by a half-dot position to make the slanted lines look much smoother.

Summary of Video Rates

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame Rate</td>
<td>60 Frames/Second</td>
</tr>
<tr>
<td>Scan line Rate</td>
<td>415 Scan Lines/Frame</td>
</tr>
<tr>
<td>Character Rate</td>
<td>115 Characters/Scan line</td>
</tr>
<tr>
<td>Dot Rate</td>
<td>9 Dots/Character</td>
</tr>
</tbody>
</table>

Multiplying the above numbers together will give a dots/frame rate of 25,771,500 (or 25.7715 MHz).
**Figure 3-13. Character Cell Example**

**ALPHA VIDEO ENHANCEMENTS.** The HP 150 supports the following types of enhancements:

- inverse
- underlined
- blinking
- half-bright
- security

(Only alpha information is blanked out. It will not have any effect on other enhancements in the same character cell.)

Any combination of the above enhancements may be defined for a character cell.
GRAPHICS DISPLAY. Graphics display is stored on the HP 150 using a RAM array, with each bit representing a graphics dot. The 8088 microprocessor writes the dot information into RAM, while the GDC3091 (U38) Graphics Display Controller fetches this information in a continuous manner. Every fetch involves the reading of one word (2 bytes) from RAM, and once the data enters the GDC3091, it is serialized and shifted out at graphics clock rate. In this manner, all graphics scan lines are displayed.

Graphics has a resolution of 390 vertical \times 512 horizontal dots. Graphics dots are 1 1/2 times as thick horizontally as alpha dots. Since the Alpha Display only has 378 scan lines, the Graphics Display area overlaps it by 6 scan lines on top and bottom of the screen. Because they are wider, the 512 graphics dots take up more room than the 720 alpha dots, and graphics overlaps alpha by about 16 graphics dots in the horizontal direction. Although the two displays are concentric in the vertical direction, they are slightly mismatched horizontally, as shown in figure 3-14.

The physical size of the display is:

<table>
<thead>
<tr>
<th>Type</th>
<th>Horizontal (mm)</th>
<th>Vertical (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHA</td>
<td>150</td>
<td>116</td>
</tr>
<tr>
<td>GRAPHICS</td>
<td>160</td>
<td>120</td>
</tr>
</tbody>
</table>
Video Board Overview

The video board contains the digital logic used to display alpha and graphics information on the screen. Alpha information is addressed from a plane separate from graphics information. To do this, it performs the following functions:

1. Stores the information to be displayed in alpha and graphics RAM.

2. Produces an alpha dot stream.
3. Produces a graphics dot stream.

4. Mixes the two dot streams and sends the resulting control signals to the sweep:

- NFB not full bright
- NHB not half bright

5. Provides deflection control signals used by the sweep:

- NVSYNC vertical sync
- HSYNC horizontal sync

Figure 3-15 shows the major components of the video board and the flow of data and control information between them. The following list provides a brief description of each of these components.

1. Processor Request and Wait Generation:
   Decodes 8088 access to the alpha or graphics RAM producing signals used by the RAM controllers. Also inserts necessary processor wait states.

2. Graphics RAM interface:
   Interfaces the 8088 and the Graphics Display Controller Chip (GDC-3091) to the graphics RAM.

3. Graphics RAM:
   32K x 8 block of dynamic RAM which is used to store graphics information. Because the graphics is a bit map display the information is stored in that format. A portion of the RAM is also used by the alpha video firmware to store variables.

4. Graphics Display Controller Chip:
   Custom gate array graphics controller which retrieves data from the graphics RAM and forms the graphics dot stream. Also provides graphics RAM timing signals.

5. Alpha RAM controller:
   Interfaces 8088 to the alpha RAM and to the SMC 9007 video controller. Also interfaces the SMC 9007 to the alpha RAM.

6. Alpha RAM:
   12K x 8 block of static RAM which stores the alpha information to be displayed.

7. SMC 9007 Alpha Video Controller:
   VLSI video controller which retrieves data from the alpha RAM and provides display signals needed to generate the alpha dot stream.

8. Character ROM:
   16K x 8 ROM which contains template for each alpha character. The ROM contains eight character sets. Each set contains 128 characters.
9. Alpha character display hardware:
This hardware uses information from alpha RAM and the character ROM along
with signals from the SMC 9007 and generates the alpha dot character stream.
This stream does not include any enhancement information.

10. Enhancement decoding logic:
This hardware decodes the enhancement information. It produces signals used
by the mixing hardware.

11. Dot stream mixing hardware:
Mixes the alpha and graphics dot streams with the enhancement information to
obtain the signals NFB and NHB which are sent to the sweep.

12. Clock generation:
 Produces various clock signals which are used by the other sections of the
board.

The following discussion outlines the events which must take place to obtain
the video dot stream and other signals sent to the sweep. It also highlights
the key points of implementation.

To obtain an alpha dot stream the following happens:

1. The 8088 writes control information to the SMC 9007 to initialize the
controller. This establishes the VSYNC and HSYNC signals sent to the sweep
as well as the other control signals produced by the SMC 9007.

2. The 8088 writes the characters and enhancements to be displayed to the alpha
RAM along with information needed by the SMC 9007 to retrieve the characters.

3. SMC 9007 retrieves data from the alpha RAM and latches it into discrete TTL
latches.

4. This data is used to address the character ROM and generate the enhancements
associated with the character.

5. The dot stream generated from the character ROM information and the
enhancement information is combined to form the alpha dot stream.

NOTES: 1. Accesses to the SMC 9007 and the alpha RAM by the 8088 are
synchronized to the SMC 9007 character clock. Also accesses by the
SMC 9007 to the alpha RAM are synchronized to the character clock.

2. 8088 and SMC 9007 alpha RAM accesses are time multiplexed within the
character clock. In other words, part of the character clock is
allotted to 8088 accesses and part is allotted to the SMC 9007.
To obtain the graphics display:

1. The 8088 writes information to be displayed into graphics RAM.

2. The Graphics Display Controller Chip retrieves the information and forms the graphics dot stream.

NOTES: 1. The signal which initializes the Graphics Display Controller Chip is derived from signals generated by the SMC 9007. This means that the SMC 9007 must be initialized before the graphics display is enabled or the graphics RAM is accessed.

2. Accesses to the graphics RAM by the 8088 and the Graphics display controller are synchronized to the graphics dot clock. The accesses are also time multiplexed within each graphics cycle, where one graphics cycle is 16 graphics dot clocks in length.

To obtain the sweep signals:

1. The alpha and graphics dot streams are combined using combinatorial logic. This logic produces the NFB and NHB signals which are sent to the sweep.

2. The NHSYNC and NVSYNC signals are produced by the SMC 9007 and are buffered before being sent to the sweep.

Notes on clock generation:

1. The 8 MHz system clock is used by the request and wait generation circuitry to aid in synchronizing the 8088 RAM accesses to the video clocks.

2. All clocks on the video board are derived from a 25.7715 MHz crystal.
KEYBOARD AND TOUCHSCREEN SUBSYSTEM

Keyboard

ELECTRICAL INTERFACE. The keyboard interfaces with the 8088 via an 8041 microcomputer that creates the timing signals needed for communications. Power for the keyboard is delivered by the processor board through the keyboard cable. The 8041 delivers a clock and sync signal to the keyboard while monitoring a key acknowledge return signal from the keyboard on one of the 8041's timer inputs.

The following signals are connected to the connector jack on the keyboard:

<table>
<thead>
<tr>
<th>Pin</th>
<th>Function</th>
<th>Pin Numbering</th>
<th>TOP VIEW</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sync</td>
<td>3 2 1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>N.C.</td>
<td>* * *</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>+12V</td>
<td>6 5 4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Ground</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>CLOCK (return)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3-16. Keyboard Jack Detail

The 8041 creates a CLOCK signal that is non-periodic that averages just under 5 KHz. The shortest time period is around 190 microseconds with a duty cycle near 50%. The SYNC signal (active high) is designed to reset the keyboard to a known state with respect to the 8041. The KEY line is a return signal from the keyboard telling the 8041 a key was depressed.

KEYBOARD OPERATION. The 8041 clocks the keyboard continually by pulsing the CLOCK line and sends out a sync pulse on the SYNC line after all the keys have been scanned. The sync pulse keeps the 8041 in sync with the keyboard.

After the 8041 delivers a sync pulse to a seven bit counter on the keyboard, it resets it to zero. On the very next negative edge of the clock this counter begins to count. The three least significant bits of the counter control the inputs to an 8 to 1 multiplexer which monitors the 8 rows of the keyboard matrix (each clock pulse causes the multiplexer to scan a different row). The four most significant bits are controlling two BCD to decimal decoders which strobe each column of the matrix. The inputs to the 8 to 1 multiplexer are pulled down via a resistor. For a particular column address, an output of the decoders will be high. If a key in that particular column is depressed, one of the eight inputs to the 8 to 1 multiplexer will go high. As the least significant bits of the counter count through all eight rows of that column, the output of the multiplexer will go high when the row of depressed key is scanned. This logic one signal is returned to the 8041 via a NAND buffer. The 8041 keeps track of how many clocks were sent to the keyboard and can determine which key was depressed when the logic one was sent on the KEY line since there is a
Hardware Subsystems

one-to-one correspondence to the number of clock pulses sent and each key switch.

Touchscreen

NOTE

The information presented here may be covered by one or more Hewlett-Packard patents.

The touchscreen allows the determination of the X-Y coordinate of an object touching the screen. This is accomplished by placing infrared emitters along two sides of the grid, and photo-detectors along the other two sides, opposite the emitters. By turning on each emitter and checking the output of the opposite detector, it can be determined which pairs are blocked. A complete scan of the pairs results in an image of the object, allowing the determination of an X-Y coordinate.

MECHANICAL DESCRIPTION. The touchscreen consists of a printed circuit board with a center cut-out. This mounts slightly in front of the CRT face, inside the bezel. The infrared beams pass through holes in the bezel and across the CRT face.

Connecting the touchscreen p.c. board with the terminal is one 10 conductor cable. It plugs into the touchscreen board at the upper right hand corner, and into the frontplane next to the sweep board connector.

For ESD protection, the inside of the bezel insert is coated with a conductive paint. Contacting this paint is a spring clip mounted on the back of the upper right corner of the board. This clip is bolted to the PCA and connects through the pad to a quick disconnect on the front of the board. A ground cable runs from the quick disconnect to the chassis.
Figure 3-17. Touchscreen Block Diagram
Hardware Subsystems

SPECIFICATIONS

Resolution: 27 rows by 40 columns

Power Consumption: +12V @ 70ma

Interface: 6 signals:

1. +12
2. ground
3. -12
4. TSCLOCK: should be driven with an open collector buffer pulled up on the T.S. board to 10 volts.
   (Period of 1.4 milliseconds or greater.
   active low, low time to be > 1.5 microseconds
   and less than 10 microseconds.)

5. TSDATA: high output is 10 volts.
   (Data out, low indicates that the currently
   addressed pair is interrupted. High
   indicates that the pair is unblocked.)

6. TSSYNC: high output is 10 volts.
   (Synch out, active low, indicates that the
   ts counter is currently on 37 (43 base 8)
   and will roll over to zero on the next
   clock.)

Connector: J1 10 pin connector with key.
TIMING

| 1.4 ms min | | <= 1.5 micro sec min |
| no max | | <= 10 micro sec max |

tsclock

count 33 34 35 36 0 0

tssync

tsdata uuu
  unblocked blocked bad emitter undefined unblocked

o Delay from falling edge clock to data valid = 0.8 msecs maximum.

o No pair exists at decimal count 21, thus data will always be high at this count.

INTERFACE DESCRIPTION. In the HP 150, a single chip microcomputer, an 8041, provides the interface to the touchscreen. Within the 8041, a software count is synchronous with the touchscreen hardware counter. As the 8041 sends clocks to the touchscreen, it increments this software count and checks the data line. When the data shows a blocked pair, the software count specifies which pair is blocked. By keeping track of which pairs were interrupted in a scan cycle, the 8041 can determine if a valid hit has occurred. If so, the hit is reported as one of 27 rows and one of 41 columns. (0-26, 0-40).

The general scheme for this is as follows:

1. While sync not active send clock (*synchronize count*)
2. Set count = 00
3. For count = 0 to 36
   send clock
   wait 1.4 milliseconds
   check data
   if data = block store count
   end
4. Check synch
5. If synch is not active then error - GOTO step 1
   else check blocked count numbers for a valid hit
6. If valid hit then report to system processor
7. GOTO step 2.
Valid Hits: At least one interrupted pair in the column range and at least one interrupted pair in the row range. More than one pair, in row or column space, is okay as long as the multiple, interrupted pairs are adjacent. (Example: pair 3 interrupted, and pair 5 interrupted is an error condition.)

No Hit: This would be a scan cycle which finds no pair interrupted.

Error Conditions:

1. An interrupted row pair, but no interrupted column pair.

2. An interrupted column pair, but no interrupted row pair.

3. Multiple hits: If any two interrupted row pairs are separated by one or more uninterrupted pairs. Likewise for columns (i.e., two distinct objects are touching the screen).

4. Out of synch: At the end of a scan cycle, the synch should be active. If this does not occur, a reset should be done.

If an error condition is found, no hit is reported.

RESOLUTION VERSUS NUMBER OF PAIRS. The resolution of the touchscreen is greater than the number of pairs used. This is accomplished by averaging the data. That is, if two adjacent pairs are interrupted, the row or column between the two is the one desired. Thus the resolution is effectively doubled, except for edge effects.

This requires that the distance between the adjacent pairs be less than the minimum stylus diameter.

Thus, if any single row or column is interrupted, that is the one desired. If an even number of adjacent beams are interrupted, the result is the center, 'half', address. If an odd number of pairs are interrupted, the result is the center beam.

PAIR ADDRESS TO ROW, COLUMN NO. CONVERSION. The pairs are positioned above and below the CRT as follows:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>32101234</td>
<td>5678901234</td>
<td>5678901234</td>
<td>5678901234</td>
<td>5678901234</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Column No. | 6 | 7 | 8 |
|---|---|---|
| 012345678901234567890 |
| x | x | x | x | x |

<table>
<thead>
<tr>
<th>Pair No.</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
</table>

Thus, across the top, the pairs are positioned every fourth column. This leads to a resolution of 40 vertical divisions, each division = 2 columns.
The pairs are actually offset to the space between 2 adjacent columns, starting at the space between columns -2, -1 and repeating every fourth column until the space between column 78, 79. This causes an offset which should be corrected at the system processor level. The effect is that the 8041 will report column pairs in the range 0 to 40. This is one more division (two columns) than on the display. To correct for this, the following should be done:

Multiply the column report by 2 => changes range from 0-40 -> 0-80
Subtract 2 => changes range from 0-80 -> -2-78
If negative then set result=0 => changes range from -2-78 -> 0-78

This corrects for the offset of the touchscreen hardware with respect to the display.

The pairs are positioned along the sides of the CRT as follows:

<table>
<thead>
<tr>
<th>T.S. Pair</th>
<th>Display Row</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>9</td>
<td>18</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>11</td>
<td>22</td>
</tr>
<tr>
<td>12</td>
<td>24</td>
</tr>
<tr>
<td>13</td>
<td>26</td>
</tr>
</tbody>
</table>

The rows are a straightforward mapping of the row number reported by the 8041 into the rows of the display. The report need only be multiplied by 2.
8041 Keyboard/Touchscreen Scanner

A description of the operation of the 8041 peripheral processor used to scan the keyboard and touchscreen follows. A complete description of the commands used to control the 8041 is included.

**NOTE**

The information presented in this manual with respect to the touchscreen may be covered by one or more Hewlett-Packard patents.

BLOCK DIAGRAM. The keyboard and touchscreen are both scanned by an 8041 peripheral processor.
STATUS REGISTER (I/O PORT 0019H). The 8041 status register contents are obtained by reading from the 8041 at I/O port address 0019H. It has the following format:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 - 4</td>
<td>Four status bits controlled by the 8041 program. These are used to qualify the data values sent to the HP 150 processor by the 8041.</td>
</tr>
<tr>
<td>3</td>
<td>F1 flag. Used internally by the 8041.</td>
</tr>
<tr>
<td>2</td>
<td>F0 flag. Used internally by the 8041.</td>
</tr>
<tr>
<td>1</td>
<td>IBF flag. Set when the HP 150 processor writes to the 8041 and cleared when the 8041 accepts the data.</td>
</tr>
<tr>
<td>0</td>
<td>OBF flag. Set when the 8041 has data available for the HP 150 processor and cleared when the host reads from the 8041 with AO = 0.</td>
</tr>
</tbody>
</table>

INITIALIZATION. The following actions will leave the 8041 in its initialized state:

- Power on
- Hard reset from keyboard
- Hard reset command from HP 150 processor
- Initialization command from HP 150 processor

After initialization, the 8041 will not scan the keyboard or touchscreen until an "enable scanning" command is given. About 200us after initialization commences, status bits 7-4 will be 0001 if the reset was from power-on or 0010 otherwise. The initialization continues and attempts to synchronize with the touchscreen. The entire initialization takes up to 100 ms.

The recommended start-up procedure for the 8041 after any of the four initialization actions listed above is:

1. Wait 100 milliseconds for the 8041 to initialize itself and check the touchscreen.
2. Send self-test command and wait for results to come back.
3. Send identify keyboard command and wait for results to come back.
4. Enable scanning.
8041 COMMANDS (I/O PORT 0019H). These commands must only be written to the command buffer of the 8041 (I/O port 0019H).

<table>
<thead>
<tr>
<th>Command Code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>08H</td>
<td>Stop bell immediately. Next bell will function normally.</td>
</tr>
<tr>
<td>10H</td>
<td>Release interrupt line. 8041 will negate its interrupt request to the host.</td>
</tr>
<tr>
<td>20H</td>
<td>Hard reset. 8041 will assert its reset output to the HP 150 processor and initialize itself to power-on state with the exception of 8041 status flags (see preceding discussion on 8041 status register). Scanning must be re-enabled for keyboard or touchscreen input.</td>
</tr>
<tr>
<td>21H</td>
<td>Identify keyboard. 8041 will respond with either 7FH or FFH depending on whether the keyboard identification diode is present or not. The 8041 generates an interrupt to the host when the ID is ready. Status bits 7-4 will be 0000 if touchscreen is defective and non-zero if the touchscreen is working.</td>
</tr>
<tr>
<td>22H</td>
<td>Keyclick off. 8041 will not make keyclick sound when keys are pressed.</td>
</tr>
<tr>
<td>23H</td>
<td>Keyclick on. 8041 will make keyclick sound.</td>
</tr>
<tr>
<td>24H</td>
<td>Enable Scanning. After power-on or hard reset, 8041 will not scan keyboard or touchscreen until command is given by the host.</td>
</tr>
<tr>
<td>25H</td>
<td>Initialize 8041 to power-on state except for status flags. Scanning must be re-enabled by HP 150 processor for keyboard or touchscreen input. This command does not reset the HP 150 processor.</td>
</tr>
<tr>
<td>26H</td>
<td>Self-test 8041. The 8041 will test its RAM and ROM and respond with F9H if it passes or F8H if it fails. No HP 150 processor interrupt will be generated; the status register must be polled until data is available.</td>
</tr>
<tr>
<td>28H</td>
<td>Stop key repeat. Any key on the keyboard will auto repeat after 500 ms if held down. If the HP 150 processor does not want the current key to auto repeat, it must send this command to stop it.</td>
</tr>
<tr>
<td>2AH</td>
<td>Disable hard reset. This command will disable the hard reset that is initiated by the control/shift/reset combination.</td>
</tr>
<tr>
<td>2BH</td>
<td>Enable hard reset. This command enables the keyboard hard reset. (Default)</td>
</tr>
</tbody>
</table>
30H - 3FH Beep bell. The bell duration is about 100 ms and its period is determined by the lower four bits of the command code. With a 5 MHz 8041 clock, the frequency is about 290 Hz for 30H and about 1.7 KHz for 3FH.

40H Do one keyclick immediately. Ignored if keyclick is disabled.

60H Touchscreen detector pairs report. This command is used to determine if any LED/Transistor pairs appeared to be blocked or bad during initialization. Two data bytes are always returned after this command is given. If either or both are not OFFH, then their values are the addresses of the blocked pairs. If both are OFFH, then there are no blocked pairs. No interrupt is generated for these two bytes; the 8041 must be polled for them. This command must be given after initialization and before scanning is enabled because it will interfere with the interrupt system if the 8041 is attempting to report keycodes.
<table>
<thead>
<tr>
<th>Decimal Count</th>
<th>Address Hex</th>
<th>Address Octal</th>
<th>Ired CR</th>
<th>Photo-Transistor Q-</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>00</td>
<td>36</td>
<td>21</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>01</td>
<td>35</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>02</td>
<td>34</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>03</td>
<td>33</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>04</td>
<td>32</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>05</td>
<td>05</td>
<td>31</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>06</td>
<td>06</td>
<td>30</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>07</td>
<td>07</td>
<td>29</td>
<td>14</td>
</tr>
<tr>
<td>8</td>
<td>08</td>
<td>10</td>
<td>28</td>
<td>13</td>
</tr>
<tr>
<td>9</td>
<td>09</td>
<td>11</td>
<td>27</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>0A</td>
<td>12</td>
<td>26</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>0B</td>
<td>13</td>
<td>25</td>
<td>10</td>
</tr>
<tr>
<td>12</td>
<td>0C</td>
<td>14</td>
<td>24</td>
<td>9</td>
</tr>
<tr>
<td>13</td>
<td>0D</td>
<td>15</td>
<td>23</td>
<td>8</td>
</tr>
<tr>
<td>14</td>
<td>0E</td>
<td>16</td>
<td>22</td>
<td>7</td>
</tr>
<tr>
<td>15</td>
<td>0F</td>
<td>17</td>
<td>21</td>
<td>6</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>20</td>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>21</td>
<td>19</td>
<td>4</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>22</td>
<td>18</td>
<td>3</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>23</td>
<td>17</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>24</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>25</td>
<td>blank</td>
<td>blank</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>26</td>
<td>2</td>
<td>22</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>27</td>
<td>3</td>
<td>23</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>30</td>
<td>4</td>
<td>24</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>31</td>
<td>5</td>
<td>25</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>32</td>
<td>6</td>
<td>26</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>33</td>
<td>7</td>
<td>27</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>34</td>
<td>8</td>
<td>28</td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>35</td>
<td>9</td>
<td>29</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>36</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>37</td>
<td>11</td>
<td>31</td>
</tr>
<tr>
<td>32</td>
<td>20</td>
<td>40</td>
<td>12</td>
<td>32</td>
</tr>
<tr>
<td>33</td>
<td>21</td>
<td>41</td>
<td>13</td>
<td>33</td>
</tr>
<tr>
<td>34</td>
<td>22</td>
<td>42</td>
<td>14</td>
<td>34</td>
</tr>
<tr>
<td>35</td>
<td>23</td>
<td>43</td>
<td>15</td>
<td>35</td>
</tr>
<tr>
<td>36</td>
<td>24</td>
<td>44</td>
<td>reset</td>
<td>reset</td>
</tr>
</tbody>
</table>
KEYBOARD AND TOUCHSCREEN DATA INPUT (I/O PORT 0018H). When data is available as a result of key presses or touchscreen touches, the 8041 will write to its output register and signal an interrupt request to the HP 150 processor. Data is buffered one byte deep in the 8041. If the previous byte written by the 8041 to its output register has not been read by the HP 150 processor, it will wait. Otherwise, it will resume scanning. Bits 7-4 of the status register are valid when the data register is valid. They are used to indicate the type of data available. The status values are:

0000 - Key address. If bit 7 of the data byte is 0, the key was depressed, else the key was released.

  Exception: If the data is in response to an Identify Keyboard command, then 0000 means the touchscreen is not connected.

0001 - Power on (valid until first write).

0010 - Reset (valid until first write).

0011 - Touchscreen release code. Data is 0.

0100 - Touchscreen row address. Same as screen row.

0101 - Touchscreen column address. Same as screen column/2.

1000 - Calculator command is complete. Data is 0 if command was not read from calculator.

1111 - Only occurs after Identify Keyboard command; indicates that touchscreen is connected.

NOTE

These status bits are valid only when data is available (OBF flag = 1). The only exception to this is the power-on/reset status which become valid 200 microseconds after reset or power-on and remains valid until the first write to the 8041.

After reading the 8041 data register, the host MUST send a release interrupt command to the 8041 to acknowledge receipt of the data. (Except for self-test results and touchscreen bad pair addresses, which are reported without setting the interrupt lines.)

Data from the touchscreen always comes in pairs of bytes, each row address will be followed by a column address. As soon as the touchscreen scanner completes a scan without any touches, the 8041 will report a touchscreen release code. Touchscreen row/column reports will be generated each time a new touched position is detected. The only way that the same position can be reported twice in a row, is when the screen is released between touches. In this case, a release code will be sent between touch reports.
DATACOMM SUBSYSTEM

General Description

The data communication (datacomm) electronics of the HP 100 Series Personal Computer allows the computer to communicate with other computers, mainframes, or peripherals via a serial data stream. The HP 150 has two serial EIA Standard RS232C (CCITT V.22) datacomm ports; Port 1 which is designed for communication with other computers either directly or using a modem, and Port 2 which is designed primarily for use with peripherals such as RS232C printers, but also can be used for computer communications.

The principal components in the datacomm circuit are the baud rate generator chip and the serial controller chip, both of which are on the I/O bus of the 8088. The baud rate generator chip supplies the timing needed for data serialization for both ports independently and the Multiprotocol Serial Controller chip (MPSC) controls both of the datacomm ports. During communications, the 8088 writes and reads 8-bit data to and from the MPSC which serializes or deserializes the data from the proper port. The rate of data serialization is governed by the timing sent to the MPSC by the baud rate generator.

RS232C/422 Datacomm Module Connector

A 36 pin connector, J3 on the Processor PCA, interfaces the RS232C/422 datacomm module to the system. The connector signals are:

| 1.  | +12V | 13. | N.C. | 25. | DATA 7 |
| 2.  | +5V  | 14. | NBRD | 26. | DATA 6 |
| 3.  | NRxC(1) | 15. | GND | 27. | DATA 5 |
| 4.  | NRR(1) | 16. | --- | 28. | DATA 3 |
| 5.  | GND | 17. | --- | 29. | DATA 1 |
| 6.  | RD(1) | 18. | --- | 30. | x16 CLK(1) |
| 7.  | NTR(1) | 19. | -12V | 31. | NSELB |
| 8.  | GND | 20. | +5V | 32. | NWRT |
| 9.  | DATA 4 | 21. | NTxC(1) | 33. | NRST |
| 10. | DATA 2 | 22. | NCS(1) | 34. | --- |
| 11. | DATA 0 | 23. | NRS(1) | 35. | --- |
| 12. | GND | 24. | SD(1) | 36. | --- |
The numbering scheme for the connector goes from 1 through 36 with pins 16, 17, 18, 34, 35, and 36 undefined and physically absent on the 36 pin connector. The signals are defined as:

**DATA 0** - The eight bit interconnect to the 8088 I/O bus.
**DATA 7**

NRxC(1) Receive timing input to the 7201 datacomm controller
NTxC(1) Transmit timing input to the 7201 controller
x16 CLK(1) Baud rate generator output

NRR(1) Receiver Ready
NTR(1) Terminal Ready
NCS(1) Clear to Send
SD(1) Send Data
RD(1) Receive Data
NRS(1) Ready to Send

NRST System Reset
NBRD 8088 read cycle
NWRT 8088 write cycle
NSELB (not used)

**Baud Rate Generator**

The baud rate generator used in the HP 150 is the SMC 8116T Dual Baud Rate Generator which can supply two independent baud rates based on a single input clock. It consists essentially of a pair of programmable dividers using an input frequency of 4 MHz. The user defines a baud rate which the 8088 translates into an 8-bit value which is written into the 8116T at I/O address XX0C hex, the most significant 4 bits of which set the baud rate for Port 2 and the least significant 4 bits set the baud rate for Port 1. The timing outputs then connect to the proper port or channel timing inputs on the MPSC.

The 8116T, in this application, generates clock frequencies for asynchronous data communications only. Asynchronous communication requires that the input clock frequency be 16 times the transmission baud rate, e.g. for a baud rate of 300 baud, a clock frequency of 4.8 KHz is required to serialize and deserialize the data. See the "Memory and I/O Mapping" section of this manual for 8116T programming information.

The clock frequencies for Port 2 tie directly into the Transmit and Receive clock inputs of the MPSC and are always the baud rate times 16.
Hardware Subsystems

The clock frequencies for Port 1 go to the datacomm interface for Port 1 and into a multiplexer, the outputs of which tie to the Transmit and Receive clock inputs of the MPSC. The multiplexer allows the selection of the 8116T clock, the times-one clock, or the Send timing clock as the Transmit clock input to the MPSC. It also selects the 8116T clock, the times-one clock, or the Receive timing clock, respectively as the Receive clock input to the MPSC. A latch drives the multiplexer select inputs and is accessed by writing the bit patterns as described in the "Memory and I/O Mapping" section of this manual.

The times-one clock is derived by dividing the 8116T clock output by 16 giving a clock of the same frequency as the transmission baud rate. This clock is available to support some modems which require a times-one clock for data transmission. Even though the Transmit and Receive clocks may be configured to use the times-one clock, the communication protocol always assumes asynchronous communication.

Multi-Protocol Controller

All of the data communications in the HP 150 are controlled by the MPSC, which is a dual-port communications chip. Channel A on the chip controls communication to Port 1 while channel B controls Port 2. The chip is capable of being configured to do all of the data serialization and deserialization, stop/start bit insertion and deletion and parity bit insertion and deletion. The chip also controls standard communications status lines on each port through internal registers such as Terminal Ready (TR) (CD), Clear to Send (CS)(CB), Receiver Ready (RR)(CB), and Request to Send (RS)(CA). The data is transmitted from each port on the Send Data line (SD)(BA) and received on the Receive Data line (RD)(BB). The data received or transmitted is accessed by the system CPU by addressing the data register for the proper port. The chip also provides a FIFO three-byte receive buffer to prevent incoming data overrun.

The chip is also capable of interrupting the CPU upon receipt or transmission of a character. The Interrupt Request output is tied to the second priority interrupt input of the interrupt controller. This is an open-collector output and is shared with the open-collector datacomm request line coming from the accessory slots (NDCOCINT). In addition to interrupts on character transmissions, the MPSC also generates interrupts on parity errors and changes in status of the communication control line inputs, freeing the CPU from polling these on each character transmission.

The internal logic of the MPSC chip (aside from transmit and receive logic) runs on the system 4 MHz clock (PCLK) and the chip is reset by system resets (NRST). The Transmit and Receive clock signal inputs used for channel B (Port 2) are tied together and are always derived from the baud rate generator times-16 clock. The Transmit and Receive clocks for channel A are driven independently from the Port 2 interface circuit (see Baud Rate Generator). The chip's Ready outputs for synchronization with the CPU are not used since this function is accomplished by always asserting an extra cycle in I/O accesses which should satisfy access times for this part. The MPSC also has Interrupt Priority inputs and outputs which are not used in this system.
Communications Interface Circuitry

The circuitry interfacing the MPSC to the communications circuitry consists primarily of line drivers (MC1488) and receivers (MC1489) which convert the MPSC TTL levels to RS232C transmission levels (+12V/-12V). The RS232C standard requires that driver outputs must not exceed a slew rate of 30 volts per microsecond. All driver outputs are tied to a 470 pf capacitor which provides an effective slew rate of 21.3 volts per microsecond maximum (assuming short circuit current). The receiver gates (MC1489) have "response control" inputs which allow filtering of high frequency noise pulses. The 470 pf capacitor used should filter noise faster than 100 ns in the 12V range. Both Port 1 and Port 2 use these parts for interfacing to the communications channel. The signal interconnection for both ports is shown in table 3-2.

Table 3-2. Datacomm Interconnect List

<table>
<thead>
<tr>
<th>Connector pin</th>
<th>Port 1 Signals</th>
<th>Port 2 Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Shield(AA)</td>
<td>Shield(AA)</td>
</tr>
<tr>
<td>2</td>
<td>SD(BA)</td>
<td>SD(BA)</td>
</tr>
<tr>
<td>3</td>
<td>RD(BB)/RD.A*</td>
<td>RD(BB)</td>
</tr>
<tr>
<td>4</td>
<td>RS(CA)</td>
<td>RS(CA)</td>
</tr>
<tr>
<td>5</td>
<td>CS(CB)</td>
<td>CS(CB)</td>
</tr>
<tr>
<td>6</td>
<td>DM(CC)</td>
<td>DM(CC)</td>
</tr>
<tr>
<td>7</td>
<td>SG(AB)</td>
<td>SG(AB)</td>
</tr>
<tr>
<td>8</td>
<td>RR(CF)</td>
<td>RR(CF)</td>
</tr>
<tr>
<td>9</td>
<td>SD.A*</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>SD.B*</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>OCR2(SCF)</td>
<td>OCR2(SCF)</td>
</tr>
<tr>
<td>15</td>
<td>ST(DB)</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>RT(DD)</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>RD.B*</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>OCD2(SCA)</td>
<td>OCD2(SCA)</td>
</tr>
<tr>
<td>20</td>
<td>TR(CD)</td>
<td>TR(CD)</td>
</tr>
<tr>
<td>22</td>
<td>OCR1(CE)</td>
<td>OCR1(CE)</td>
</tr>
<tr>
<td>23</td>
<td>OCD1(CH)</td>
<td>OCD1(CH)</td>
</tr>
<tr>
<td>24</td>
<td>TT(DA)</td>
<td></td>
</tr>
</tbody>
</table>

* Represents RS422 signal.
Hardware Subsystems

The control lines OCD1, OCD2, OCR1, OCR2 and DM are not connected directly to the MPSC, but are tied to the CPU data bus through handshake capability with certain kinds of equipment that require them (e.g. printers, modems, etc.) and are all available on both ports. I/O ports 14H and 16H access these control lines on Ports 1 and 2 respectively as described in the "Memory and I/O Mapping" section of this manual.

Port 1 differs from Port 2 in two major respects; Port 1 contains timing signals on the RS232C interface as well as RS422 signals on pins that are not used by RS232C in most HP applications (those pins are reserved for Data Set Testing in the RS232C definition). The timing signals added are Transmit Timing (TT)(DA), Send Timing (ST)(DB), and Receive Timing (RT)(DD). The Send and Receive timing signals allow an external device to drive the MPSC clock inputs for serialization. The Transmit Timing signal is always the same as the MPSC input transmission clock.

The RS422 interface consists essentially of a differential driver and receiver (75179) in place of the RS232C drivers. These drivers transmit and receive differential voltages with respect to the opposite signal line as opposed to referencing ground. Note that the RS232C signal RD(BB) and the RS422 signal RD.A are shared in this application. This is accomplished by pulling down the opposite driver input to create the proper translation to TTL level signals when the RS232C cable is connected. The selection between RS232C and RS422 is simply a matter of connecting the cable which connects to the proper signals. Since the receive signal comes into the same gate and the transmit signal drives both interfaces in parallel, the difference is transparent to the MPSC.
MEZZANINE MEMORY SUBSYSTEM

Mezzanine Memory PCA

PCA Overview

The HP 150 mezzanine memory PCA (assembly 45611-60006) houses ROM, dynamic RAM, CMOS RAM, and system status LEDs on a PCA in a mezzanine position between the processor and video PCAs. The PCA is supported by four standoffs mounted on the processor PCA and electrically interfaces to the processor PCA through a 60 pin connector. The block diagram of the PCA is shown in figure 3-18.
Figure 3-18. Mezzanine Memory PCA Block Diagram
Connector Signals

A 60 pin connector, J2 on the Processor PCA, connects the ROM and user RAM to the rest of the system. The signals on the interface are listed below:

**Mezzanine Memory Connector Signals**

<table>
<thead>
<tr>
<th>Signal</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSLOTSEL1</td>
<td>+5V</td>
</tr>
<tr>
<td>ABUS 11</td>
<td>ABUS 19</td>
</tr>
<tr>
<td>NSLOTSEL2</td>
<td>ABUS 9</td>
</tr>
<tr>
<td>ASCHOLD</td>
<td>NPWRDN</td>
</tr>
<tr>
<td>AD1</td>
<td>AD0</td>
</tr>
<tr>
<td>AD3</td>
<td>AD2</td>
</tr>
<tr>
<td>AD5</td>
<td>AD4</td>
</tr>
<tr>
<td>AD6</td>
<td>DT/-R</td>
</tr>
<tr>
<td>ABUS 5</td>
<td>AD7</td>
</tr>
<tr>
<td>ROMSEL</td>
<td>ABUS 16</td>
</tr>
<tr>
<td>ABUS 17</td>
<td>ABUS 14</td>
</tr>
<tr>
<td>ABUS 18</td>
<td>BCLK</td>
</tr>
<tr>
<td>ABUS 12</td>
<td>ABUS 13</td>
</tr>
<tr>
<td>NCMOSSL</td>
<td>NCMSWRT</td>
</tr>
<tr>
<td>ABUS 4</td>
<td>NXR1</td>
</tr>
<tr>
<td>+5V</td>
<td>GO</td>
</tr>
<tr>
<td>FULLMEM</td>
<td>+5V</td>
</tr>
<tr>
<td>+5V</td>
<td>EDEN</td>
</tr>
<tr>
<td>ABUS 7</td>
<td>CMOSPWR</td>
</tr>
<tr>
<td>GND</td>
<td>GND</td>
</tr>
<tr>
<td>+5V</td>
<td>GND</td>
</tr>
<tr>
<td>GND</td>
<td>NWRWT</td>
</tr>
<tr>
<td>NCMSOE</td>
<td>SLODA</td>
</tr>
<tr>
<td>ABUS 5</td>
<td>ABUS 6</td>
</tr>
<tr>
<td>BIO/-M</td>
<td>NERD</td>
</tr>
<tr>
<td>ABUS 8</td>
<td>ABUS 10</td>
</tr>
<tr>
<td>+5V</td>
<td>ABUS 3</td>
</tr>
<tr>
<td>ABUS 1</td>
<td>GND</td>
</tr>
<tr>
<td>NRST</td>
<td>NMSEL</td>
</tr>
<tr>
<td>ABUS2</td>
<td>ABUS0</td>
</tr>
</tbody>
</table>

The mezzanine memory PCA connector signals are described below:

**ABUS 0-19** The demultiplexed 20 bit address generated by the 8088 during a bus cycle.

**AD 0-7** Least significant 8 bits of the multiplexed data/address bus. Data is transferred to and from the memory PCA over these lines.
Signal indicating an address within the 64K block allocated to accessory module 1 (90000H - 9FFFFH) has been generated by the 8088.

Same as NSLOTSEL1 but is asserted when the address is in the address space allocated to accessory module 2 (A0000H - AFFFH).

Signal which qualifies the address generated by the 8088. GO goes high at the beginning of T2 and goes back low at the beginning of T4 of a bus cycle.

Buffered 8 Mhz system clock.

Buffered IO/-M signal from the 8088.

DT/-R from the 8088.

Inverted and buffered -DEN signal from the 8088.

Buffered -RD signal from the 8088.

Buffered -WRT signal from the 8088.

Asynchronous HOLD request generated by the memory PCA and sent to the 8088.

Synchronized (with the 8 Mhz clock) HOLD acknowledge from the 8088.

Signal if high prevents write to dynamic RAM.

Transceiver control signal generated by memory PCA.

ROM access indicator which causes the processor PCA to eliminate wait state on the bus cycle.

When high indicates 256K RAM loaded on the memory PCA. When low indicates 128K RAM loaded. (A pull-up resistor external to the memory PCA is used to pull high.)

System reset line for the memory PCA.

CMOS RAM select line active low.

CMOS RAM output enable active low.

When low causes a write into CMOS RAM.

Battery backed-up power source for the CMOS RAM.
NPWRDN Goes low to indicate an imminent power interruption to the system and puts the CMOS RAM into the standby mode.

ROM

ROM DECODING. Five 32K x 8 ROMs contain the system firmware used by the PC. These ROMs are located in the memory map as follows:

<table>
<thead>
<tr>
<th>ROM</th>
<th>Memory Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>(ROM 5)</td>
<td>OF8000H - 0FFFFFH</td>
</tr>
<tr>
<td>(ROM 4)</td>
<td>OF0000H - 0F7FFFH</td>
</tr>
<tr>
<td>(ROM 3)</td>
<td>OE8000H - 0EFFFH</td>
</tr>
<tr>
<td>(ROM 2)</td>
<td>OE0000H - 0E7FFFH</td>
</tr>
<tr>
<td>(ROM 1)</td>
<td>OB0000H - 0B7FFFH</td>
</tr>
</tbody>
</table>

For a particular ROM to be selected, an address within the memory space allocated to it must be generated by the 8088, the 8088 control signal IO/-M (becomes BIO/-M after buffering for the memory PCA) must be low, and NBRD (buffered read strobe control signal fromn the 8088) must be asserted (active low). The decode circuitry for ROMs 2, 3, 4, and 5 decode address lines ABUS 19, 18, 18, 17, 16, and 15 when BIO/-M is low and assert the chip select input of the ROM whose contents is at the address generated by the processor. The 8088 receives data from the memory (or writes to RAM or the LED register) through a transceiver on the memory PCA. This transceiver is enabled when ROM, CMOS RAM, dynamic RAM, or the LEDs are accessed.

WAIT STATE DISABLE. Every bus cycle (memory or I/O operation) of an 8088 consists of four system clocks unless logic external to the 8088 dictates that the cycle be extended by some integer number of additional clocks. The wait state generation circuitry on the processor PCA inserts a minimum of 1 additional clock period into all bus cycles unless a ROM access is being made.
Hardware Subsystems

ROM TIMING. The memory address from the 8088 becomes available to the memory PCA 85 nsec after the beginning of T1. The propagation delays until data is available to the processor is as follows:

<table>
<thead>
<tr>
<th>ROM 1 Access</th>
<th>ROMs 2-5 Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>address valid</td>
<td>address valid</td>
</tr>
<tr>
<td>chip select delay</td>
<td>chip select delay</td>
</tr>
<tr>
<td>ROM access time</td>
<td>ROM access time</td>
</tr>
<tr>
<td>buffer delay</td>
<td>buffer delay</td>
</tr>
<tr>
<td>connector delay</td>
<td>connector delay</td>
</tr>
<tr>
<td>total delay</td>
<td>total delay</td>
</tr>
<tr>
<td>85 nsec</td>
<td>85 nsec</td>
</tr>
<tr>
<td>31 nsec</td>
<td>29 nsec</td>
</tr>
<tr>
<td>200 nsec</td>
<td>200 nsec</td>
</tr>
<tr>
<td>12 nsec</td>
<td>12 nsec</td>
</tr>
<tr>
<td>2 nsec</td>
<td>2 nsec</td>
</tr>
<tr>
<td>330 nsec</td>
<td>328 nsec</td>
</tr>
</tbody>
</table>

The cycle is 375 nsec long (T1 + T2 + T3). The data setup time for the 8088 is 20 nsec. Therefore, the data read margin is 375 - 20 - 330 = 25 nsec. Note that the output enable inputs to the ROMs have been asserted worst case 120 nsec after the beginning of T1 so that ROM address access time, not ROM output enable time, is the limiting factor in ROM access timing.

Slot Selection Generation

The HP 150 package provides two PCA slots for handling optional module PCA devices. The HP 150 architecture provides a flexible interface to the accessory slots. Accessories can be accessed through either memory or I/O access (see memory and I/O maps).

I/O addresses XX80 through XXFF have been reserved for accessory module use. This provides 128 I/O ports that can be decoded and used for processor and auxiliary device intercommunication. The most flexible arrangement that can be used for accessory module device interfacing to the 8088 is the memory mapped interface with slot select. A memory address within the 90000 - 9FFFF range causes Slot Select #1 on the front plane connecting to the external module slot #1 to be asserted and an address within A0000 - AFFFF causes Slot Select #2 to be asserted.

**CAUTION**

It is highly recommended that only 16K of address space be used for either of the accessory cards. The address ranges 90000-93FFF and A0000-A3FFF for slot 1 and slot 2, respectively, should be exclusively decoded and used even though the slot select lines provide 64K of address space. Accessory cards using memory space outside this range may not be compatible with any enhanced future versions of the HP 150.
What does the slot select scheme provide? An accessory PCA may choose to use or not to use the slot select line. If a device does not utilize slot select, a decode of at least the most significant four address bits must be done to detect a CPU access. More decoding of the least significant 16 address bits must be done depending on the nature of the PCA and its circuitry. In addition, the user must never plug PCAs into Slots #1 and #2 which become selected on the same address range. When fully decoding the address space, boards must be designed for a specific accessory slot.

Using the slot select lines, a more limited decode is required on the accessory PCA since the four most significant bits are decoded by the CPU in asserting the slot select line. Therefore, an accessory PCA can detect an access by decoding slot select and the appropriate subset of the least significant 16 address bits. The decode of slot select ensures that no contention will occur between accessory modules #1 and #2 since only one slot select signal will be active at one time.

The slot select thereby eliminates the need for a designer of an accessory module to be aware of all the addresses used by other existing or future accessory modules. It also limits the hardware required for decoding.

Since the accessory modules are in the 8088 address space, firmware ROMs can be placed on the accessory modules to be executed by the 8088. This way drivers for each accessory can be located in ROM on the accessory PCA and at power-on, when the terminal operating system does a logical system generation, the drivers for the modules will be used when needed for module stimulation.

Another benefit of having the accessory modules in memory space is that block transfers of data between CPU and accessory module can take place quickly with little CPU software overhead and with the enhanced flexibility of the memory access instruction set of the 8088 over the I/O instructions. Thus, a softcard CPU can be added cleanly to the system without the hardware and particularly the processor intercommunication path hampering performance.

To summarize, if a memory address between 090000H - 09FFFFH is generated by the CPU, NSLOTSEL1 is asserted. If an address between 0A0000H - 0AFFFFFH is generated, NSLOTSEL2 is asserted. These signals are routed to the accessory slots with NSLOTSEL1 going to one accessory slot and NSLOTSEL2 going to the other.

**CMOS RAM**

256 nybbles of battery backed up CMOS RAM is located at address 0BC000H - 0BFFFFH. The 32K address space allocated to CMOS RAM images the 256 nybbles found at 0BC000H - 0BCOFFH. The nybble wide data is accessed on the lower four bits of the data bus.
CMOS DECODING AND ACCESS. Interfacing the CMOS RAM to the system required circuitry to accommodate some of the unusual address and data setup and hold timing parameters associated with the part. An address generated by the processor within CMOS memory space causes decoding circuitry to assert a CMOS address space select signal. When this signal is asserted, the processor PCA wait state generation circuitry adds 6 clock periods to the bus cycle.

CMOS POWER. The power delivered to the CMOS comes from the same power source as that used for the real time clock. The Vcc pin for the chip is maintained at approximately 5 volts when power is applied to the system and falls to about 2.5 volts when the power is turned off.

LEDs

Six LEDs are positioned on the memory PCA to report power-on test results. The LEDs are memory mapped at 0B8000H - 0BBFFFH. Any memory write within this range will access the LEDs. The LEDs are accessed on the lower six bits of the data bus.

LED DECODING. When the LED address is present, a memory write cycle is in progress, and when GO is deasserted, the LED register will be clocked with data from the CPU. The LEDs will in turn display the complement of the register contents (i.e., a "zero" turns an LED on, a "one" turns an LED off).

LED REGISTER RESET. When the power is turned on or the NRESET test point on the processor PCA is grounded, the system reset signal will reset the LED register, making all outputs go to the zero state. This will turn all the LEDs on and will indicate to the observer that +5 volts is functioning at least well enough to light the LEDs. The self-test code will then proceed to turn off the LEDs if the system tests pass; or writes an error code into the LEDs signifying the failing subsystem or component.

Dynamic RAM

The 256K of dynamic RAM requires a considerable amount of support circuitry. This section will describe the decoding, control signal generation, and RAM timing. The next section will cover the refresh mechanism used on this PCA.
DECODING. The 256K of RAM is organized as four banks of 64K. The memory space allocation is:

<table>
<thead>
<tr>
<th>Bank</th>
<th>Address Range</th>
<th>Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00000H - 0FFFFF</td>
<td>U42, U49</td>
</tr>
<tr>
<td>1</td>
<td>10000H - 1FFFFF</td>
<td>U32, U39</td>
</tr>
<tr>
<td>2</td>
<td>20000H - 2FFFFF</td>
<td>U22, U29</td>
</tr>
<tr>
<td>3</td>
<td>30000H - 3FFFFF</td>
<td>U12, U19</td>
</tr>
</tbody>
</table>

When a memory cycle is initiated by the 8088 (BIO/-M is low) and an address within one of the four memory regions listed above is generated, a 3-to-8 line decoder asserts one of four outputs designated for RAM bank selection. When GO becomes valid at the beginning of T2, the RAS inputs to the selected RAM bank will all go low.

DYNAMIC RAM REFRESH. In order to retain the data that has been written into a dynamic RAM, a periodic refresh cycle must occur. The refresh cycle consists of placing a row address on the address input pins of the dynamic RAM (DRAM) and then bringing the RAS input low for a specified amount of time and then raising it back high. Each row address within the RAM must be refreshed as described within a time period specified by the RAM manufacturer in order for the data to be retained in the RAM. This memory PCA uses a dynamic RAM which requires each of its 128 row addresses to be refreshed no less than every 2 milliseconds. This specification is typical for most DRAMs. Some DRAMs have 256 row addresses and require all row addresses to be refreshed no less than every 4 msec. The refresh controller on this memory PCA is capable of meeting the requirements of both types of DRAM.

The refresh scheme used on this PCA refreshes 4 row addresses every 56 usec. Thus, all the rows are refreshed within 1.8 msec. The entire refresh process takes about 2.5 usec including latencies due to the synchronizing flip-flops. Therefore, the refresh process consumes about \( (2.5 \text{ usec}/56 \text{ usec}) \times 100\% = 4.6\% \) of the system bandwidth.

PCA CONFIGURATION. Normally, this PCA will be shipped with 256K of DRAM. This requires jumper W3 to be installed. If this PCA is configured with only 128K of DRAM jumpers W1 and W2 should be installed and jumper W3 should not be installed. This will allow this PCA to function properly as a 128K RAM PCA with banks 0 and 1 of RAM installed and allows an extended memory PCA in an accessory slot to have a contiguous mapping of its RAM with the RAM on this PCA starting at memory address 20000H.
ACCESSORIES SUBSYSTEM

Accessory Hardware Design Guidelines

The two accessory slots in the card cage provide the means of increasing the overall system capability by adding hardware to meet needs specific to that of certain users. The following paragraphs provide guidelines and specifications needed by designers to interface with the system hardware.

Mechanical Specifications

The accessory PCA electrically interfaces to the system through a 70 pin connector located at the front of the board. The PCA can be of the type that fits within the card cage such as the Expansion Memory PCAs. Or it may be of the type which requires interfacing to external devices and requires a special rear panel to accommodate a connector or connectors.

Mechanical drawings for the accessory card slots and rear panel are included in this section.

Power Requirements

Each of the two accessory slots have the following available power:

<table>
<thead>
<tr>
<th>VOLTAGE</th>
<th>CURRENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>+5V</td>
<td>1.9A</td>
</tr>
<tr>
<td>+12V</td>
<td>0.45A</td>
</tr>
<tr>
<td>-12V</td>
<td>0.1A</td>
</tr>
</tbody>
</table>

If powering an external device through the power supply, the designer must use a filtering network and fuse between the external device and the HP 100 Series Personal Computer. Note that the power consumed by an accessory is limited by thermal considerations described next.
Thermal Limits

The maximum thermal dissipation allowed per accessory board is 10 watts. If there is an accessory device consuming 10W power in one slot and your board is to go in the other, you should expect temperatures on your accessory board of 20 to 30 degrees C above ambient if your board consumes 5W. If your board consumes 8.5W, you should expect to see a 20 to 40 degrees C temperature rise above ambient on your accessory board. It is best to measure the case temperatures of components on the accessory board if one wants to verify the operating temperature of a particular device is within the manufacturer’s specifications.

Accessory Signal Loading Restrictions

The accessory boards have access to the signals listed in the front plane accessory slot pinout detail (table 3-5). Proper adherence to the AC and DC loading restrictions are required for per specification performance of the system. Table 3-3 lists the loading restrictions per accessory PCA.
Table 3-3. Accessory Loading Restrictions

<table>
<thead>
<tr>
<th>Signal</th>
<th>Available Source Current</th>
<th>Available Sink Current</th>
<th>Maximum Capacitive Loading****</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPA 0-7, 12-19, ABUS 8-11</td>
<td>800 uA</td>
<td>3.5 mA</td>
<td>35 pF</td>
</tr>
<tr>
<td>FPD 0-7</td>
<td>800 uA</td>
<td>3.5 mA</td>
<td>35 pF</td>
</tr>
<tr>
<td>BIO/-M*</td>
<td>800 uA</td>
<td>3.5 mA</td>
<td>25 pF</td>
</tr>
<tr>
<td>FPNRD, FPNWRT, FPNSSO, FPDT/-R, FPCLK, FPGO</td>
<td>500 uA</td>
<td>5.0 mA</td>
<td>25 pF</td>
</tr>
<tr>
<td>NSLOTSEL n</td>
<td>500 uA</td>
<td>5.0 mA</td>
<td>25 pF</td>
</tr>
<tr>
<td>FPNRST</td>
<td>800 uA</td>
<td>5.0 mA</td>
<td>50 pF</td>
</tr>
<tr>
<td>NPPFAIL</td>
<td>500 uA</td>
<td>2.0 mA</td>
<td>25 pF</td>
</tr>
<tr>
<td>BATV</td>
<td>40 uA</td>
<td>0</td>
<td>---</td>
</tr>
<tr>
<td>SHOLDA</td>
<td>250 uA</td>
<td>5.0 mA</td>
<td>25 pF</td>
</tr>
<tr>
<td>FULLMEM**</td>
<td>0</td>
<td>10 mA</td>
<td>---</td>
</tr>
<tr>
<td>NOCINT***</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDCOCINT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOCWAIT</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTE**

* The BIO/-M signal comes from one driver and is found on pins 47 and 62 of the connector. The aggregate of the loading on the two pins must not exceed the specification in the table.

** FULLMEM, if used, must be pulled high by a pullup resistor on the accessory board.

*** These signals must be driven by an open collector driver. The pullup resistor is on the processor board and need not be on the option board. NOCWAIT requires a driver capable of sinking 18 mA (Schottky driver). The NOCINT and NDCOCINT drivers must be capable of sinking 5 mA.

**** To estimate capacitive loading, one can use 2 pF per connector pin, 5 pF/inch of PC trace, and 5-10 pF per gate terminal.
Signal Timing Diagrams

Timing diagrams in figures 3-19 thru 3-21 are based on a system with two accessory boards installed which conform to the loading restrictions stated in the previous paragraph.

Table 3-4. Accessory Slot Timing Characteristics*

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Nom.</th>
<th>Min.</th>
<th>Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tab</td>
<td>Clock high time</td>
<td>43</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tbc</td>
<td>Clock low time</td>
<td>68</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tac</td>
<td>Clock period</td>
<td>125</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tal, Tbm</td>
<td>FPCLK to 8088 clock skew</td>
<td></td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>Tan</td>
<td>BIO/-M delay from clock rising</td>
<td>14</td>
<td>88</td>
<td></td>
</tr>
<tr>
<td>Tao</td>
<td>FPNSS0 delay from clock rising</td>
<td>14</td>
<td>83</td>
<td></td>
</tr>
<tr>
<td>Tbp</td>
<td>FPA n delay from T1 beginning</td>
<td></td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>Tbc</td>
<td>ABUS n delay from T1 beginning</td>
<td></td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>Tkr</td>
<td>ABUS hold time from T4 rising</td>
<td></td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Tbs</td>
<td>NSLOTSEL delay from T1 beginning</td>
<td></td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>Tdt</td>
<td>FPGO delay from T2 beginning</td>
<td></td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>Tju</td>
<td>FPGO delay from T4 beginning</td>
<td></td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>Tdv</td>
<td>FPNRD delay from T2 beginning</td>
<td>14</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>Tjw</td>
<td>FPNRD delay from T4 beginning</td>
<td>14</td>
<td>108</td>
<td></td>
</tr>
<tr>
<td>Twv</td>
<td>FPNRD width</td>
<td></td>
<td>325</td>
<td></td>
</tr>
<tr>
<td>Txj</td>
<td>Read data setup time</td>
<td></td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>Tjy</td>
<td>Read data hold time</td>
<td>10</td>
<td>225</td>
<td></td>
</tr>
<tr>
<td>Taz</td>
<td>FPDT/-R delay from clock rising</td>
<td></td>
<td>14</td>
<td>83</td>
</tr>
<tr>
<td>TkAA</td>
<td>FPDT/-R delay from clock rising</td>
<td></td>
<td>14</td>
<td>83</td>
</tr>
<tr>
<td>TdBB</td>
<td>FPNWRT delay from T2 beginning</td>
<td>14</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>TjCC</td>
<td>FPNWRT delay from T4 beginning</td>
<td>14</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>TBBCC</td>
<td>FPNWRT width</td>
<td></td>
<td>335</td>
<td></td>
</tr>
<tr>
<td>TdDD</td>
<td>Data valid delay from T2 beginning</td>
<td></td>
<td>84</td>
<td></td>
</tr>
<tr>
<td>TkEE</td>
<td>Data hold time from rising</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TuEE</td>
<td>Data hold time from FPPO deassertion</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TFFHH,</td>
<td>SHOLDA delay from falling</td>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TGGII</td>
<td>edge of clock</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TTHII</td>
<td>SHOLDA width (in usec)</td>
<td>2.5</td>
<td>2.25</td>
<td>2.63</td>
</tr>
<tr>
<td>TLLJJ,</td>
<td>NOCWAIT setup time before</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TMMKK</td>
<td>falling edge of clock</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLLMM</td>
<td>NOCWAIT width (in usec)</td>
<td>0.125</td>
<td>40</td>
<td></td>
</tr>
</tbody>
</table>

*All time specifications are in nsec unless stated otherwise.
Figure 3-19. Accessory Slot Bus Cycle Timing
Hardware Subsystems

---

Figure 3-20. SHOLDA Timing

---

Figure 3-21. Accessory Wait State Insertion

---
Accessory Front Plane Connector

The accessory PCAs plug into the bottom slots of the card cage. The system signals available to the modules are listed in table 3-5.

Table 3-5. Connector Pinouts

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FPA 0</td>
<td>26</td>
<td>FPD 3</td>
<td>51</td>
</tr>
<tr>
<td>2</td>
<td>FPA 1</td>
<td>27</td>
<td>FPD 4</td>
<td>52</td>
</tr>
<tr>
<td>3</td>
<td>FPA 2</td>
<td>28</td>
<td>FPD 5</td>
<td>53</td>
</tr>
<tr>
<td>4</td>
<td>FPA 3</td>
<td>29</td>
<td>FPD 6</td>
<td>54</td>
</tr>
<tr>
<td>5</td>
<td>FPA 4</td>
<td>30</td>
<td>FPD 7</td>
<td>55</td>
</tr>
<tr>
<td>6</td>
<td>FPA 5</td>
<td>31</td>
<td>GND</td>
<td>56</td>
</tr>
<tr>
<td>7</td>
<td>FPA 6</td>
<td>32</td>
<td>FPNRD</td>
<td>57</td>
</tr>
<tr>
<td>8</td>
<td>FPA 7</td>
<td>33</td>
<td>GND</td>
<td>58</td>
</tr>
<tr>
<td>9</td>
<td>ABUS 8</td>
<td>34</td>
<td>+5V</td>
<td>59</td>
</tr>
<tr>
<td>10</td>
<td>ABUS 9</td>
<td>35</td>
<td>BATV</td>
<td>60</td>
</tr>
<tr>
<td>11</td>
<td>ABUS 10</td>
<td>36</td>
<td>(RESERVED) *</td>
<td>61</td>
</tr>
<tr>
<td>12</td>
<td>ABUS 11</td>
<td>37</td>
<td>NPFAIL</td>
<td>62</td>
</tr>
<tr>
<td>13</td>
<td>FPA 12</td>
<td>38</td>
<td>(RESERVED) *</td>
<td>63</td>
</tr>
<tr>
<td>14</td>
<td>FPA 13</td>
<td>39</td>
<td>FPNRST</td>
<td>64</td>
</tr>
<tr>
<td>15</td>
<td>FPA 14</td>
<td>40</td>
<td>+5V</td>
<td>65</td>
</tr>
<tr>
<td>16</td>
<td>FPA 15</td>
<td>41</td>
<td>GND</td>
<td>66</td>
</tr>
<tr>
<td>17</td>
<td>FPA 16</td>
<td>42</td>
<td>GND</td>
<td>67</td>
</tr>
<tr>
<td>18</td>
<td>FPA 17</td>
<td>43</td>
<td>NOCINT</td>
<td>68</td>
</tr>
<tr>
<td>19</td>
<td>FPA 18</td>
<td>44</td>
<td>GND</td>
<td>69</td>
</tr>
<tr>
<td>20</td>
<td>FPA 19</td>
<td>45</td>
<td>NOCWAIT</td>
<td>70</td>
</tr>
<tr>
<td>21</td>
<td>GND</td>
<td>46</td>
<td>(RESERVED) *</td>
<td>71</td>
</tr>
<tr>
<td>22</td>
<td>GND</td>
<td>47</td>
<td>BIO/-M</td>
<td>72</td>
</tr>
<tr>
<td>23</td>
<td>FPD 0</td>
<td>48</td>
<td>+5V</td>
<td>73</td>
</tr>
<tr>
<td>24</td>
<td>FPD 1</td>
<td>49</td>
<td>-12V</td>
<td>74</td>
</tr>
<tr>
<td>25</td>
<td>FPD 2</td>
<td>50</td>
<td>NDCOCINT</td>
<td>75</td>
</tr>
</tbody>
</table>

**JF3 and JF4**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>+ + ...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>+ + ...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pictorial view of option module front plane connectors (JF3 and JF4) with pin assignment (as viewed from component side of Front Plane PCA).

* Pin 59 is FULLMEM is on JF3 only. Pin 59 is GND on JF4. Connections to this pin and all pins labeled RESERVED should not be made.

** NNSLOTSELx will be either NNSLOTSEL1 or NNSLOTSEL2 depending upon which side of the front plane the module is plugged into. The left side of the card cage (from rear view of the package) holds module 1 and gets the NNSLOTSEL1 signal. The right side of the card cage holds module 2 and gets NNSLOTSEL2.
Accessory Connector Signal Descriptions

The accessory connector signals described in table 3-6 reference the timing diagrams in figures 3-19 through 3-21.

Table 3-6. Signal Descriptions

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPA 0-7</td>
<td>The lower 8 address bits of the 20 bit address generated by the 8088. These signals are buffered and demultiplexed.</td>
</tr>
<tr>
<td>ABUS 8-11</td>
<td>The upper 12 address bits of the 20 bit address.</td>
</tr>
<tr>
<td>FPA 12-19</td>
<td>These signals also are buffered and demultiplexed.</td>
</tr>
<tr>
<td>FPD 0-7</td>
<td>Data bus signals from the external data bus. This data path is the means through which data is passed between the processor and the video board or option slot modules or the optional TPM.</td>
</tr>
<tr>
<td>FPCLK</td>
<td>Buffered 8 MHz system clock.</td>
</tr>
<tr>
<td>FPWRD</td>
<td>Buffered -RD signal from the 8088 used to indicate a bus read cycle in progress.</td>
</tr>
<tr>
<td>FPWRT</td>
<td>Buffered -WRT signal from the 8088 used to indicate a bus write cycle in progress.</td>
</tr>
<tr>
<td>FPNSSO</td>
<td>8088 bus cycle status line. The combination of FPNSSO, BIO/-M, and FPDT/-R allow boards connected to the front plane to completely decode the current bus cycle.</td>
</tr>
<tr>
<td>BIO/-M</td>
<td>Buffered I0/-M signal from the 8088 used to distinguish memory and I/O bus cycles.</td>
</tr>
<tr>
<td>FPDT/-R</td>
<td>Buffered DT/-R signal from the 8088 used to indicate direction of data from the 8088 for a given bus cycle.</td>
</tr>
<tr>
<td>FPGO</td>
<td>Signal which qualifies the address generated by the microprocessor. The address qualification is needed primarily for dynamic RAM circuits which cannot tolerate an assertion of RAS or CAS on a false address. FPGO goes high at the beginning of T2 and goes back low at the beginning of T4 of a bus cycle. FPGO can also be used to terminate a bus write cycle by having its falling edge used to clock data into a register or other device on an accessory device in an option slot. Terminating the writes in this way can provide better hold timing than using FPNWRT.</td>
</tr>
</tbody>
</table>
Hardware Subsystems

NOCWAIT: This line can be asserted by the option modules or video board through an open collector gate to insert wait states into a bus cycle to provide sufficient time for a bus cycle access.

NDCOCINT: This open collector interrupt signal has the same interrupt priority as the datacomm controller chip on the processor board as it shares the same input to the interrupt controller as the datacomm controller chip does. This input can be asserted by accessories for interrupt servicing.

NOCINT: System interrupt signal asserted by a device via an open collector gate. This line can be used by options slots to get processor service.

SHOLDA: Synchronized hold acknowledge signal from the 8088. HOLD is asserted by the mezzanine memory board.

FULLMEM: Indicates which mezzanine memory board option is installed. FULLMEM=0 if 128K RAM and FULLMEM=1 (it is pulled high by a pull up resistor on an installed extension memory board) if 256K RAM is on the board. An accessory should not connect to this signal pin.

NSLOTSEL1: Signal indicating an address within the 64K block allocated to option module 1 has been generated by the 8088.

NSLOTSEL2: Same as NSLOTSEL1 but asserted when address within option module 2 address space is generated.

FPNRST: Signal generated by processor board at power-on time to initialize logic circuitry.

NPFFAIL: Signal generated by the power supply indicating power supply output level stability.

BATV: These two connector lines carry current from the batteries located on the video board to circuitry requiring battery back-up power such as the CMOS RAM and the CMOS real time clock.
ACCESSORY CARD HARDWARE AND ELECTRICAL

Electrical Design

HELPFUL DESIGN HINTS. Because the 8088 in the HP 150 runs in "Min" mode, the write line (FPWWR) may not rise soon enough to lock your data in. The signal FPGO is designed to allow this timing deficiency to be overcome. One way this may be implemented is by combining DT/-R with FPGO to create a write signal. The sample schematic (figure 3-31) at the end of this section shows a typical method of doing this.

Note that the signal NSLOTSEL is not guaranteed to be valid until 110 ns into the "T1" cycle. Prior to that time, glitches may be present. Edge sensitive chips should not be directly connected to this line. In general, NSLOTSEL should be combined with another of the control lines to create a valid signal. Note how NSLOTSEL is combined with FPGO on the sample schematic to create the signal NQUAL which is low only when the board is truly selected.

Based on the two points above, you have probably noticed what a handy little signal FPGO can be. FPGO is the first signal you should turn to if you come up with any timing problems in the interface to the HP 150 bus.

The HP 150 firmware has provisions to deal with options cards. To take full advantage of these features, an option card should provide an "ID BYTE" at the first address in the memory space. In the sample schematic enclosed, the "ID BYTE" has been loaded into the first address of the ROM. If a ROM is not present on the accessory card, an "ID Byte" may be generated via an octal buffer (like a LS244). Use a dip switch to set the "ID BYTE" in this case.

Be aware of the capacitive and dc loading specifications that an accessory card must meet. It doesn't take too many gates and trace lengths to exceed the specifications. Note, for example, that most control lines will not drive two Schottky gates because there would be a violation of the DC loading specification. The DC and AC loading specifications are contained in table 3-3.

GENERAL SCHEMATIC DISCUSSION. There is a "generic" accessory card schematic in figure 3-31. It includes a tranceiver interface to the data bus, buffering of the address lines, buffering and filtering of the reset line, memory and I/O space accesses, and a ROM. The memory and address reads are separated with the BIO/-M signal. When high, this signal disables the S138. The S138 determines which of the the main blocks on the accessory card are selected. If an I/O access occurs to the address set by the dip switches, then the state of the F74 flip flop is toggled. This will select one of the two LEDs. The "controller" is a piece of LSI logic which takes care of the external peripheral device.
Another feature that is nice to have, although not present on the enclosed schematic, is a software hard reset of the accessory card. This allows the card to be hard reset without a hard reset of the system.

**TRANSCEIVER SCHEMATIC DISCUSSION.** The schematic in figure 3-22 includes a tranceiver on the data bus. This circuit illustrates how a "general purpose" tranceiver might be put on the data bus.

There are a number of problems in putting a tranceiver on the data bus including:

1. Creating a write pulse which will clock data into its destination and allow sufficient hold time before the tranceiver is disabled.

2. Ensuring that the tranceiver is not disabled prior to the data hold time on a read.

3. Ensuring that the tranceiver is disabled prior to DT/-R changing state.

The circuit proposed should solve these problems without causing contention on the HP 150's data bus or on the accessory card's data bus. In addition, it does not rely on unspecified minimum delay times. Check the timing to see if this circuit will work for a particular application.

The circuit works by first latching (with the rising edge of FPCLK) a board select signal (NQUAL) as the tranceiver enable signal. This delays the tranceiver enable signal, allowing read/write cycles to finish prior to disabling the tranceiver.

Since the tranceiver enable signal has been delayed, DT/-R must also be delayed. The second flip-flop latches the state of DT/-R with the falling edge of FPCLK. This delay allows the tranceiver to be disabled prior to a direction change (at the end of an instruction cycle) while still allowing the direction to be established prior to tranceiver enable (at the start of an instruction cycle).

Of course, the circuits above will not be useful for every design but they may be helpful in saving a little time in the design process. Remember that the ultimate responsibility is yours.

**Parts List**

- 1 FAST '00
- 1 FAST '74
- 1 ALS '245
- 1 LS '04
DC Loading (For Transceiver Circuit Only)

- **FPDT/-R**: 40 μA High, 1.2 mA Low
- **FPGO**: 40 μA High, 1.2 mA Low
- **FPCLK**: 40 μA High, 1.0 mA Low
- **NSLOTSEL**: 20 μA High, .4 mA Low

References

1982 *Fast Data Book* (Fairchild)
1981 *TTL Data Book* (Texas Instruments)
1983 *ALS/AS Data Book* (Texas Instruments)

Figure 3-22. Transceiver Schematic
HELPFUL DESIGN HINTS. Consider the height restriction drawing since the tabs where the I/O panel thumbscrews attach can interfere with components on the accessory card.

The I/O Panel should be of a sturdy design so it will transfer sufficient force to cause the frontplane connector to seat. When testing the mechanical design, remove the metal chassis and watch the connector as the thumbscrews are tightened. It is possible to design a panel such that the thumbscrews will go all the way in, yet the frontplane connector will not move.

Make sure that the I/O panel has as many holes for airflow as possible. The air will flow along the long dimension of the accessory board. For most effective use of this airflow, lay the chips out with their long dimension perpendicular to this airflow. Also, lay your "hot" chips as close to the I/O panel as possible.

Be sure to mask off the annular ring around the thumbscrew's hole and to use an inside star washer between the thumbscrew and the I/O panel. This will ensure that there is a solid ground path between the I/O panel and the chassis of the HP 150.

I/O Panels with a connector should have the ground plane split near the I/O panel to bypass the connector shield to the I/O Panel ground. This will prevent high frequency noise (like ESD) from entering the system.

I/O PANEL DESIGN. Accessory cards not requiring an I/O Panel should be based on the board blank drawing titled "Board Blank - no I/O Panel" with two extractors and two extractor retainer pins to remove the accessory card from the HP 150. Drawings for the extractors as well as the extractor retainer pins are provided in this manual.

If an I/O panel is required, all the required dimensions may be obtained from the drawings enclosed. Please note that the accessory card fits under the L-bracket for the design presented. The I/O Panel should be solid and sturdy since the force from the thumbscrews must be transmitted to the frontplane connector in order to fully insert the connector.
LIST OF VENDORS

**CAUTION**

Parts may be available from the following vendors. HP does not guarantee or warrant any of the following parts or vendors.

- **Frontplane Connector**
  
  Vendor: Amp
  Harrisburg, Pa.
  
  Vendor Part Number: 1-102584-0

- **Thumbscrew**
  
  Vendor: Precision Metal
  3402 Enterprise Ave.
  Hayward, Ca 94545
  (415) 887-1401
  
  Vendor Part Number: 45641-0001-2 (also HP Part No.)
  
  Use the drawing in figure 3-24 to set this part up with its own part number. This will prevent any possible changes to the part from affecting you. The drawing will also allow the part to be set up with any vendor you wish.

- **E-Ring Retainer**
  
  Vendor: Truarc Retaining Rings Division
  Waldes Kohinoor, Inc.
  47-16 Austel Place
  Long Island City, NY 11101
  (212) 392-3100
  
  Contact the factory for the distributor near you.
  
  Vendor part number: Truarc X-5133-11 H.
Hardware Subsystems

- **L-Bracket**

  Vendor: Pike Tool and Grinding Inc.
  4205 High Country Road
  Colorado Springs, Colo. 80907
  (303) 598-9611

  Vendor Part Number: 45641-00001-1 (also HP Part No.)

  Use the drawing in figure 3-25 to set this part up with its own part number. This will prevent any possible changes to the part from affecting you. Again, the drawing is provided for this part if you wish to set it up yourself.

- **Accessory Card Extractor (figure 3-26)**

  Vendor: Trend Plastic
  1245 Laurelwood Rd.
  Santa Clara, CA. 95050
  (408) 727-5797

  Vendor Part Number: 0403-0473 (also HP Part No.)

- **Accessory Card Extractor Retainer Pin (figure 3-27)**

  Vendor: Drivelock, Inc.
  Sycamore, IL.
  (815) 895-8161

  Vendor Part Number: 9945

  One possible distributor for this part is:

  Bearing Engineers, Inc.
  1250 Space Park Way
  Mountain View, CA. 94040
  (415) 969-1155

  The Bearing part number is 3-0.330-R-SP.

**I/O PANEL PAINT SPECIFICATIONS**

**Scope**

This specification establishes the requirements for pigment composition, to achieve a color in HP's Corporate Color Program.
Requirements

- Pigments used to manufacture a color should be from the approved Hewlett-Packard List, "Pigments Used to Achieve the Corporate Color".

- Pigment substitutes may be required for specific type of material that uses higher temperature, (such as plastic) to be produced in a Corporate Color.
  
  a. Use pigment with equivalent physical and performance properties as recommended on the pigment list.

  b. Required physical and performance properties of pigments:

     1. Heat Resistance
     2. Bleeding Characteristics
     3. Chemical Resistance
     4. Exposure Performance (Ultra-violet, etc.)

Quantitative

- Pigment Composition (approximation - adjust % for tinting purposes)

  a. Paint Pigments for Parchment White Case, Visual Color Standard No. 6009-0130:

     | Component        | Amount  |
     |------------------|---------|
     | Lampblack        | Trace   |
     | Titanium Dioxide White | 18.55%  |
     | Yellow Iron Oxide | .31%    |
     | Red Iron Oxide   | Trace   |

     Gloss Standard: 18+-3 Gloss Units at 85 degree /

  b. Paint Pigments for Cobblestone Gray Accent, Visual Color Standard No. 6009-0133:

     | Component        | Amount  |
     |------------------|---------|
     | Lampblack        | 8.0%    |
     | Titanium Dioxide White | 28.4%   |
     | Yellow Iron Oxide | 10.9%   |
     | Red Iron Oxide   | 4.5%    |
     | Other            | 48.3%   |

     Gloss Standard: 18+-3 Gloss Units at 85 degree /
Hardware Subsystems

Drawings

Included in this section are the following drawings:

- I/O Panel
- Retainer Thumbscrew
- L-Bracket
- Accessory Card Extractor
- Accessory Card Extractor Pin
- Board Blank With I/O Panel (Accessory Card Details)
- Height Restriction (Accessory Card Clearances)
- Board Blank - No I/O Panel
- Accessory Card Schematic
NOTES
1. FINISH: SAND TO DEBURR. ZINC CHROMATE CONVERSION .005MM THICK.
2. RIVET ITEM 7 TO ITEM 1 WITH ITEM 8 (2 PLCS.)
3. MASK OFF TWO AREAS OF 8MM DIA. AT THE TWO 3.5 DIA. HOLES AND SPRAY PAINT THE FRONT SIDE AND EDGES OF ITEM 1.
4. INSTALL ITEMS 2, 4, 5 AS SHOWN ON SIDE VIEW

Hardware Subsystems
Figure 3-24. Retainer Thumbscrew

NOTES:
1. FINISH: SAND TO DEBURR,
   ZINC CHROMATE, 0.005 MM THICK
2. ITEM 2 INSTALLED AS SHOWN.
   USE PEM SELF-CLINCHING
   FASTENER OR EQUIVALENT.

Figure 3-25. L-Bracket
Hardware Subsystems

Figure 3-26. Accessory Card Extractor

Figure 3-27. Accessory Card Extractor Retainer Pin
Hardware Subsystems

Figure 3-28. Board Blank Drawing With I/O Panel (Accessory Card Details)

Figure 3-29. Height Restriction Drawing (Accessory Card Clearances)
NOTE

No components allowed in shaded area. Also do not run any traces in the shaded region on the circuit or component side.

Figure 3-30. Board Blank - No I/O Panel
Hardware Subsystems

This is page is blank.
Figure 3-31. Typical Accessory Card Schematic
This section contains memory and input/output maps for the HP 150 memory and input/output address spaces. Included is information describing the register and bit mappings of all devices which consume a portion of the 8088 processor's address space for both memory and input/output instruction types. More information regarding the functionality of the devices themselves may be found in Section 3, Hardware Subsystems.
Memory Mapped Devices

HP 150 Memory Map 4-1
HP 150 Memory Map continued 4-2

CRT Controller Registers

Horizontal Timing Registers (R0, R1, R2 and R3) 4-4
Vertical Timing Registers (R4, R5, R7, R8 and R9) 4-5
Pin Configuration/Skew Bits Register (R6) 4-5
DMA Control Register (RA) 4-5
Control Register (RB) 4-6
Table Start Register (RC and RD) 4-6
Auxiliary Register 1 (RE and RF) 4-6
Sequential Break Register 1 (R10) 4-6
Data Row Start Register (R11) 4-6
Data Row End Register (R12) 4-6
Auxiliary Address Register 2 (R13 and R14) 4-7
Start Command (R15) 4-7
Reset Command (R16) 4-7
Smooth Scroll Offset Register (R17) 4-7
Vertical Cursor Register (R18) 4-7
Horizontal Cursor Register (R19) 4-7
Cursor Registers R38 and R39 (READ) 4-7
Interrupt Enable Register (R1A) 4-7
Status Register (R3A) 4-8
Vertical Light Pen Register (R3B) 4-8
Horizontal Light Pen Register (R3C) 4-8
Video Attribute Latch 4-8

I/O Mapped Devices

HP 150 Input / Output Map 4-9
Real Time Clock (MM58167A) 4-10
Integral Printer Interface 4-11
Keyboard / Touchscreen Controller (8041A) 4-11
Datacomm Port 1 Control Lines / Manuf Test Repeat 4-12
Datacomm Port 2 Control Lines / Clock Source Select 4-12
Interrupt Controller (8259A) 4-13
Baud Rate Generator (8116T) 4-13
HPIB Controller (9914) 4-14
MPSC - Datacomm Controller (7201/8274) 4-14
MEMORY MAPPED DEVICES

HP 150 Memory Map

+-------------------------------------------+  ..... continued on next page
HP 150 Memory Map continued

<table>
<thead>
<tr>
<th>Reservation</th>
<th>Accessory Slot 2 Address Space (16K)</th>
<th>Accessory Slot 1 Address Space (16K)</th>
<th>ROM Expansion</th>
<th>RAM Expansion</th>
</tr>
</thead>
<tbody>
<tr>
<td>B0000</td>
<td>Reserved for future accessory slots</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A4000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>94000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>90000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30000</td>
<td>Operating System and User RAM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTE**

For a more detailed map of Operating System, User, and Firmware RAM usage, refer to System Software (Section 5), Operating System Memory Map.
## CRT Controller Registers

<table>
<thead>
<tr>
<th>Register Type</th>
<th>Bit Definition</th>
<th>Initialization Register</th>
<th>Data 8088-Addr (hex) (hex)</th>
<th>Read/Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7 D6 D5 D4 D3 D2 D1 D0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSB</td>
<td>CHARs./HORIZONTAL PERIOD</td>
<td>LSB</td>
<td>R0</td>
<td>73</td>
</tr>
<tr>
<td>MSB</td>
<td>CHARACTERS/ DATA ROW</td>
<td>LSB</td>
<td>R1</td>
<td>#14F</td>
</tr>
<tr>
<td>MSB</td>
<td>HORIZONTAL DELAY</td>
<td>LSB</td>
<td>R2</td>
<td>2E</td>
</tr>
<tr>
<td>MSB</td>
<td>HORIZONTAL SYNC WIDTH</td>
<td>LSB</td>
<td>R3</td>
<td>07</td>
</tr>
<tr>
<td>MSB</td>
<td>VERTICAL SYNC WIDTH</td>
<td>LSB</td>
<td>R4</td>
<td>13</td>
</tr>
<tr>
<td>MSB</td>
<td>VERTICAL DELAY</td>
<td>LSB</td>
<td>R5</td>
<td>20</td>
</tr>
<tr>
<td>PIN CONFIG</td>
<td>CURSOR SKEW</td>
<td>BLANK SKEW</td>
<td>LSB</td>
<td>R6</td>
</tr>
<tr>
<td>MSB</td>
<td>VISIBLE DATA ROWS/F RAME</td>
<td>LSB</td>
<td>R7</td>
<td>#1A</td>
</tr>
<tr>
<td>(B10) (B8) (MSB)</td>
<td>SCAN LINES/DATA ROW</td>
<td>(LSB)</td>
<td>R8</td>
<td>2D</td>
</tr>
<tr>
<td>(B7)</td>
<td>SCAN LINES/F RAME</td>
<td>(B0)</td>
<td>R9</td>
<td>9F</td>
</tr>
<tr>
<td>DMA</td>
<td>DMA BURST DLY</td>
<td>DMA BURST COUNT</td>
<td></td>
<td>RA</td>
</tr>
<tr>
<td>ABL</td>
<td>MSB</td>
<td>LSB</td>
<td>LSB</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>PB/</td>
<td>INTERLACE</td>
<td>OPERATION</td>
<td>2X/C</td>
</tr>
<tr>
<td>SS</td>
<td>MODES</td>
<td>MODES</td>
<td>1X</td>
<td></td>
</tr>
<tr>
<td>MSB</td>
<td>TABLE START REGISTER (LS BYTE)</td>
<td>LSB</td>
<td>RC</td>
<td>91</td>
</tr>
<tr>
<td>ADDRESS</td>
<td>TABLE START REG (MS BYTE)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODE</td>
<td>(MSB)</td>
<td>(LSB)</td>
<td>RD</td>
<td>97</td>
</tr>
<tr>
<td>MSB</td>
<td>AUX. ADDRESS REG.1 (LS BYTE)</td>
<td>LSB</td>
<td>RE</td>
<td>#FF</td>
</tr>
<tr>
<td>ROW</td>
<td>AUX ADDR REG 1 (MS BYTE)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATTR'S</td>
<td>(MSB)</td>
<td>(LSB)</td>
<td>RF</td>
<td>#3F</td>
</tr>
<tr>
<td>MSB</td>
<td>SEQUENTIAL BREAK REG. 1</td>
<td>LSB</td>
<td>R10</td>
<td>#FF</td>
</tr>
<tr>
<td>MSB</td>
<td>DATA ROW START REGISTER</td>
<td>LSB</td>
<td>R11</td>
<td>FF</td>
</tr>
<tr>
<td>MSB</td>
<td>DATA ROW END/SEQU. BRK REG 2</td>
<td>LSB</td>
<td>R12</td>
<td>FF</td>
</tr>
<tr>
<td>MSB</td>
<td>AUX. ADDRESS REG.2 (LS BYTE)</td>
<td>LSB</td>
<td>R13</td>
<td>+0</td>
</tr>
<tr>
<td>ROW</td>
<td>AUX ADDRESS REG 2 (MS BYTE)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATTR'S</td>
<td>(MSB)</td>
<td>(LSB)</td>
<td>R14</td>
<td>+0</td>
</tr>
<tr>
<td>OFST</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OVR-</td>
<td>OFFSET VALUE</td>
<td></td>
<td>R17</td>
<td>00</td>
</tr>
<tr>
<td>FLOW</td>
<td>(MSB)</td>
<td>(LSB)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>MSB</td>
<td>VERTICAL CURSOR REGISTER (ROW COORD.)</td>
<td>(LSB)</td>
<td>R18</td>
<td>FF</td>
</tr>
<tr>
<td>(MSB)</td>
<td></td>
<td></td>
<td>R19</td>
<td>00</td>
</tr>
<tr>
<td>MSB</td>
<td>HORIZONTAL CURSOR REG. (COL. COORD.)</td>
<td></td>
<td>R19</td>
<td>00</td>
</tr>
<tr>
<td>(LSB)</td>
<td></td>
<td></td>
<td>R39</td>
<td>+--</td>
</tr>
<tr>
<td>VERT</td>
<td>INTERRUPT ENABLE REG</td>
<td>FRM</td>
<td></td>
<td>R1A</td>
</tr>
<tr>
<td>RE-LIGHT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIM-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>TRCE</td>
<td>PEN</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>INT VERT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PEND</td>
<td>RE-LIGHT</td>
<td></td>
<td>ODD/</td>
<td></td>
</tr>
<tr>
<td>DING</td>
<td>TRCE</td>
<td>PEN</td>
<td></td>
<td>EVEN</td>
</tr>
<tr>
<td>MSB</td>
<td>VERT LIGHT PEN REG (ROW COORD)</td>
<td>LSB</td>
<td>R3B</td>
<td>+--</td>
</tr>
<tr>
<td>MSB</td>
<td>HOR LIGHT PEN REG (COL COORD)</td>
<td>LSB</td>
<td>R3C</td>
<td>+--</td>
</tr>
<tr>
<td>[VIDEO ATTRIBUTES REGISTER]</td>
<td>[DISP]</td>
<td>[CURS]</td>
<td></td>
<td>&quot;VATT&quot;</td>
</tr>
</tbody>
</table>
* These registers are programmed with N -1.
♯ These registers must have these initialization values.
+ These registers may not be accessible on future HP 150 revisions and must not be written to or read from.

NOTE

For a full description of the CRT Controller registers, refer to the SMC9007 data in Standard Microsystems Corporation 1982 Data Book. Use of registers noted as being not accessible on future HP 150 revisions is not recommended and such access may result in software incompatibilities with certain HP 150 units. All addresses not specifically shown are reserved and should not be used.

A description of CRT controller registers as those registers are used and supported by the HP 150 follows.

HORIZONTAL TIMING REGISTERS (R0, R1, R2 AND R3)

These registers define the horizontal scan line timing. They must contain the (hexadecimal) values shown.

```
! <------- one scan line ----------> !

! <---------- R0 ----------------> !
! 73 !
! <-------- R1 -------> !<--(R0-R1)--->!
! 4F !
! 2E

+-----------------------+ +--------ACTIVE DISPLAY +--------+

VLT +-----------------------+ +-----------------------+

HOR. SYNC !<--- R3 --->!
+-----------------------+ 07 +---
VERTICAL TIMING REGISTERS (R4, R5, R7, R8 AND R9)

These registers define the vertical frame timing. They must contain the (hexadecimal) values shown.

! <----- one complete frame -------> !
! <---------- R9 ----------------> !
! 9F !
! <------ (R7xR8)------- > !
! 1A 2D !
! ! ! <-R5-> !

VERTICAL BLANKING + DATA ROW DISPLAY TIME! +
(NO SKEW) +---------------------- +

VERT. SYNC !
++------------------------+

PIN CONFIGURATION/SKEW BITS REGISTER (R6)

- Contains pin configuration information ( bit 7,6 )
- Cursor skew ( bits 5,4,3 ) define the number of character
clocks the cursor signal is delayed from VLT
- Blank skew ( bits 2,1,0 ) define the number of character
clocks the horizontal blank component of the CBLANK signal
is delayed from VLT.
- Both cursor skew and blank skew are the value "001" for
the 1 character skew.
- Must contain C9 Hex.

DMA CONTROL REGISTER (RA)

- DMA disable ( bit 7 ). A logic "1" on this pin forces the
SMC9007 DMA request into the inactive state, and the
address bus will enter its high impedance state.
- DMA burst delay (bits 6,5,4 ). This register is loaded
with "111" for zero delay, allowing all characters to be
retrieved from video RAM in one burst.
- DMA Burst Count ( bits 3,2,1,0 ). Not Used.
- Must contain 70 Hex.
CONTROL REGISTER (RB)
- 7 bit register
- Smooth scroll mechanism is enabled by writing a "1" to bit 6.
- Interlace, ( bits 5,4 ). "00" - non interlaced mode.
- Operation mode ( bits 3,2,1 ). "000" for repetitive memory addressing.
- Single/double height cursor ( bit 0 ). "1" = single.
- Must contain 41 Hex.

TABLE START REGISTER (RC AND RD)
- These registers point to the address where the row table begins.
- The registers are set up the following way for contiguous row table mode:
  * register D ( bits 7,6 ) = "10"
  * register D ( bits 5-0 ) = upper 6 bits of the 14 bit address
  * register C ( bits 7-0 ) = lower 8 bits of the 14 bit address.
- RC must contain 91 Hex, RD must contain 97 Hex.

AUXILIARY REGISTER 1 (RE AND RF)
- Not used, except for bits 7,6 in register F which must be "00" for single height, single width characters.
- RE must contain FF Hex, RF must contain 3FH.

SEQUENTIAL BREAK REGISTER 1 (R10)
- This register may not be accessible on certain HP 150 revisions and must not be used.

DATA ROW START REGISTER (R11)
- Defines the first data row number at which a smooth scroll operation begins.
- Is initialized by the system firmware to FF Hex.

DATA ROW END REGISTER (R12)
- The row numerically one less than the row defined by this register is the last data row on which a smooth scroll will occur.
- Is initialized by the system firmware to FF Hex.
AUXILIARY ADDRESS REGISTER 2 (R13 AND R14)

- This register may not be accessible on certain HP 150 revisions and must not be used.

START COMMAND (R15)

- During initialization of the SMC9007, after all vital screen parameters are loaded, a start command can be initiated by addressing this dummy register location.

RESET COMMAND (R16)

- This register may not be accessible on certain HP 150 revisions and must not be used.

SMOOTH SCROLL OFFSET REGISTER (R17)

- This register is loaded with the scan line offset number to allow a smooth scroll operation to occur. The offset register causes the scan line counter to start at the programmed value rather than zero for the data row that starts the smooth scroll interval.
- Must have data bit 7 (most significant) cleared (0).

VERTICAL CURSOR REGISTER (R18)

- This register specifies the data row in which the cursor appears.

HORIZONTAL CURSOR REGISTER (R19)

- This register specifies the character position in which the cursor appears.

CURSOR Registers R38 AND R39 (READ)

- These registers may not be accessible on some revisions of the HP 150 and must not be used.

INTERRUPT ENABLE REGISTER (R1A)

- This 3 bit write only register allows each of the 3 SMC9007 interrupts to be enabled or disabled.
- Bit 6 is set to "1" to enable vertical retrace interrupts.
- Must contain 40 Hex.
Memory and I/O Mapping

STATUS REGISTER (R3A)

- This register is only used to clear the interrupt bit. The value read is irrelevant.

VERTICAL LIGHT PEN REGISTER (R3B)

- This register may not be accessible on certain revisions of the HP 150 and must not be used.

HORIZONTAL LIGHT PEN REGISTER (R3C)

- This register may not be accessible on certain revisions of the HP 150 and must not be used.

VIDEO ATTRIBUTE LATCH

This latch stores 5 bits of video data, and can be updated during the blank portions of the video frame. It is a memory mapped I/O register with the 8088 address D30BE hexadecimal. The register is actually decoded as a non-existent SMC9007 register.

VIDEO ATTRIBUTE LATCH PIN DEFINITION

+----------+-----------+----------+----------+----------+
| DISPON   | BLRT      | CBLRT    | BLOB     | GREW     |
+----------+-----------+----------+----------+----------+
| D4       | D3        | D2       | D1       | D0       |

note: - positive logic is used ( 1 = on, 0 = off )
- when writing to this register, D5-D7 are don't cares

DISPON : Alpha Display On
BLRT : Character Blink Rate
    Characters with BL ( blink ) enhancement set will be blanked on the screen while BLRT is a "1."
CBLRT : Cursor Blink Rate
    The character for which CURS from the SMC9007 is active high, the cursor will be blinked when CBLRT is a "1."
BLOB : Blob Cursor Select
    (otherwise double scan line cursor)
GREW : Graphics Display Enable
### I/O MAPPED DEVICES

#### HP 150 I/O Map

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00FF</td>
<td>Reserved for Accessory Slots</td>
</tr>
<tr>
<td>0080</td>
<td>I/O Image (DO NOT USE)</td>
</tr>
<tr>
<td>0060</td>
<td>Real Time Clock (MM58167A)</td>
</tr>
<tr>
<td>0040</td>
<td>Integral Printer Interface</td>
</tr>
<tr>
<td>0030</td>
<td>Reserved (DO NOT USE)</td>
</tr>
<tr>
<td>0018</td>
<td>Keyboard/Touchscreen Controller (8041A)</td>
</tr>
<tr>
<td>0016</td>
<td>Datacomm Port 2 Control Lines</td>
</tr>
<tr>
<td></td>
<td>Manufacturing Test Repeat</td>
</tr>
<tr>
<td>0014</td>
<td>Datacomm Port 1 Control Lines</td>
</tr>
<tr>
<td></td>
<td>Datacomm Clock Source Select</td>
</tr>
<tr>
<td>0010</td>
<td>Interrupt Controller (8259)</td>
</tr>
<tr>
<td>000C</td>
<td>Baud Rate Generator (8116T)</td>
</tr>
<tr>
<td>0004</td>
<td>HPIB Controller (9914)</td>
</tr>
<tr>
<td>0000</td>
<td>MPSC - Datacomm Controller (7201)</td>
</tr>
</tbody>
</table>
Memory and I/O Mapping

NOTE

I/O mapped devices at 0000 Hex through 0080 Hex are not exclusively decoded to 16 bits. Rather they may be accessed through any combination of high order byte bits. For future hardware compatibility it is highly recommended that programs which access I/O mapped devices directly at these addresses do so with a high order byte address of 00 Hex. Accessory cards may exclusively decode a 16 bit address. See the accessory card specifications for more information.

Real Time Clock (MM58167A)

<table>
<thead>
<tr>
<th>Address</th>
<th>Register</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0056-005FH</td>
<td>Not Used</td>
<td>reserved</td>
</tr>
<tr>
<td>0055H</td>
<td>&quot;GO&quot; Command Register</td>
<td>write</td>
</tr>
<tr>
<td>0054H</td>
<td>Status Bit</td>
<td>read</td>
</tr>
<tr>
<td>0053H</td>
<td>RAM Register Reset</td>
<td>write</td>
</tr>
<tr>
<td>0052H</td>
<td>Counter Register Reset</td>
<td>write</td>
</tr>
<tr>
<td>0051H</td>
<td>Interrupt Control Register</td>
<td>write</td>
</tr>
<tr>
<td>0050H</td>
<td>Interrupt Status Register</td>
<td>read</td>
</tr>
<tr>
<td>004FH</td>
<td>Month RAM Register</td>
<td>read/write</td>
</tr>
<tr>
<td>004EH</td>
<td>Date RAM Register</td>
<td>read/write</td>
</tr>
<tr>
<td>004DH</td>
<td>Day RAM Register</td>
<td>read/write</td>
</tr>
<tr>
<td>004CH</td>
<td>Hour RAM Register</td>
<td>read/write</td>
</tr>
<tr>
<td>004BH</td>
<td>Minutes RAM Register</td>
<td>read/write</td>
</tr>
<tr>
<td>004AH</td>
<td>Seconds RAM Register</td>
<td>read/write</td>
</tr>
<tr>
<td>0049H</td>
<td>Tenth/Hundreth Sec RAM Reg</td>
<td>read/write</td>
</tr>
<tr>
<td>0048H</td>
<td>Millisecond RAM Register</td>
<td>read/write</td>
</tr>
<tr>
<td>0047H</td>
<td>Month Counter Register</td>
<td>read/write</td>
</tr>
<tr>
<td>0046H</td>
<td>Date Counter Register</td>
<td>read/write</td>
</tr>
<tr>
<td>0045H</td>
<td>Day Counter Register</td>
<td>read/write</td>
</tr>
<tr>
<td>0044H</td>
<td>Hour Counter Register</td>
<td>read/write</td>
</tr>
<tr>
<td>0043H</td>
<td>Minutes Counter Register</td>
<td>read/write</td>
</tr>
<tr>
<td>0042H</td>
<td>Seconds Counter Register</td>
<td>read/write</td>
</tr>
<tr>
<td>0041H</td>
<td>Tenth/Hundreth Sec Counter</td>
<td>read/write</td>
</tr>
<tr>
<td>0040H</td>
<td>Millisecond Counter Register</td>
<td>read/write</td>
</tr>
</tbody>
</table>

For more information on this part, see the MM58167A Data Sheet, National Semiconductor Corporation.
Integral Printer Interface

<table>
<thead>
<tr>
<th>Address</th>
<th>Register</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0030H</td>
<td>Status/Data</td>
<td>read/write</td>
</tr>
</tbody>
</table>

Bit: 7 6 5 4 3 2 1 0

READ | - | - | - | - | - | NPAPER | ONLINE | ACK |
|-----|---|---|---|---|---|--------|--------|-----|

WRITE | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|------|----|----|----|----|----|----|----|----|

Keyboard / Touchscreen Controller (8041A)

<table>
<thead>
<tr>
<th>Address</th>
<th>Register</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0019H</td>
<td>Status / Command Register</td>
<td>read/write</td>
</tr>
<tr>
<td>0018H</td>
<td>Data Register</td>
<td>read/write</td>
</tr>
</tbody>
</table>

For more information on the use of these registers, see the "Hardware Subsystems" section, "Keyboard and Touchscreen".
## Memory and I/O Mapping

### Datacomm Port 2 Control Lines / Manuf Test Repeat

<table>
<thead>
<tr>
<th>Address</th>
<th>Register</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0016H</td>
<td>OCR1/OCD1, OCR2/OCD2, DM Ctrl. Lines, Port 1 detect, Manufac. test bit</td>
<td></td>
</tr>
</tbody>
</table>

**Bit:**

<table>
<thead>
<tr>
<th>READ</th>
<th>WRITE</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----</td>
<td>-----</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>POD</td>
<td>DM</td>
<td>MTST</td>
<td>OCR2</td>
<td>OCR1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the above:

- **POD** = 0 if Port 1 datacomm PCA is not present.
- **POD** = 1 if Port 1 datacomm PCA is in place.
- **MTST** = 0 if jumper wire is not grounding U62 pin 14.
- **MTST** = 1 if U62 pin 14 is grounded causing repetition of manufacturing test segment that failed.

### Datacomm Port 1 Control Lines / Clock Source Select

<table>
<thead>
<tr>
<th>Address</th>
<th>Register</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0014H</td>
<td>OCR1/OCD1, OCR2/OCD2, DM Ctrl. Lines, Clock Source Select</td>
<td></td>
</tr>
</tbody>
</table>

**Bit:**

<table>
<thead>
<tr>
<th>READ</th>
<th>WRITE</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Clock Source</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>DM</td>
<td>0</td>
<td>OCR2</td>
<td>OCR1</td>
</tr>
</tbody>
</table>

Clock source select bits are defined as follows:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Receive Clock</th>
<th>Transmit Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x16</td>
<td>x16</td>
</tr>
<tr>
<td>0</td>
<td>x1</td>
<td>x1</td>
</tr>
<tr>
<td>1</td>
<td>RT</td>
<td>ST</td>
</tr>
<tr>
<td>1</td>
<td>RT</td>
<td>ST</td>
</tr>
</tbody>
</table>
Interrupt Controller (8259A)

<table>
<thead>
<tr>
<th>Address</th>
<th>Register</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010,0011H</td>
<td>Interrupt Controller Registers</td>
<td>read/write</td>
</tr>
</tbody>
</table>

For more information on this part, see the 8259A Data Sheet, Intel Corporation.

Baud Rate Generator (8116T)

<table>
<thead>
<tr>
<th>Address</th>
<th>Register</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>000CH</td>
<td>Baud Rate Select</td>
<td>write</td>
</tr>
</tbody>
</table>

Baud Rate Select Coding:

<table>
<thead>
<tr>
<th>Port 2 Control</th>
<th>Port 1 Control</th>
<th>Clock Output (Hz.)</th>
<th>Baud Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7 D6 D5 D4</td>
<td>D3 D2 D1 D0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 0 0</td>
<td>0 0 0 0 0</td>
<td>800</td>
<td>50</td>
</tr>
<tr>
<td>0 0 0 1 0</td>
<td>0 0 0 1 0</td>
<td>1200</td>
<td>75</td>
</tr>
<tr>
<td>0 0 1 1 1</td>
<td>0 0 1 1 1</td>
<td>1760</td>
<td>110</td>
</tr>
<tr>
<td>0 1 0 0 0</td>
<td>0 1 0 0 0</td>
<td>2152</td>
<td>134.5</td>
</tr>
<tr>
<td>0 1 0 1 0</td>
<td>0 1 0 1 0</td>
<td>2400</td>
<td>150</td>
</tr>
<tr>
<td>0 1 1 0 0</td>
<td>0 1 1 0 0</td>
<td>4800</td>
<td>300</td>
</tr>
<tr>
<td>0 1 1 1 0</td>
<td>0 1 1 1 0</td>
<td>9600</td>
<td>600</td>
</tr>
<tr>
<td>1 0 0 0 0</td>
<td>1 0 0 0 0</td>
<td>1200</td>
<td>120</td>
</tr>
<tr>
<td>1 0 0 1 0</td>
<td>1 0 0 1 0</td>
<td>28800</td>
<td>1800</td>
</tr>
<tr>
<td>1 0 1 0 0</td>
<td>1 0 1 0 0</td>
<td>32000</td>
<td>2000</td>
</tr>
<tr>
<td>1 0 1 1 0</td>
<td>1 0 1 1 0</td>
<td>38400</td>
<td>2400</td>
</tr>
<tr>
<td>1 1 0 0 0</td>
<td>1 1 0 0 0</td>
<td>57600</td>
<td>3600</td>
</tr>
<tr>
<td>1 1 0 1 0</td>
<td>1 1 0 1 0</td>
<td>76800</td>
<td>4800</td>
</tr>
<tr>
<td>1 1 1 0 0</td>
<td>1 1 1 0 0</td>
<td>115200</td>
<td>7200</td>
</tr>
<tr>
<td>1 1 1 0 1</td>
<td>1 1 1 0 1</td>
<td>153600</td>
<td>9600</td>
</tr>
<tr>
<td>1 1 1 1 0</td>
<td>1 1 1 1 0</td>
<td>307200</td>
<td>19200</td>
</tr>
</tbody>
</table>

* Denotes baud rates not configurable through the HP 150 Config menus.
## HPIB Controller (9914)

<table>
<thead>
<tr>
<th>Address</th>
<th>Register</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000H</td>
<td>Command Pass Thru / Parallel Pol</td>
<td>read/write</td>
</tr>
<tr>
<td>000AH</td>
<td>Address Status Register</td>
<td>read</td>
</tr>
<tr>
<td>0009H</td>
<td>Address Switch / Address Reg</td>
<td>read/write</td>
</tr>
<tr>
<td>0008H</td>
<td>Interrupt Status 0 / Interrupt M</td>
<td>read/write</td>
</tr>
<tr>
<td>0007H</td>
<td>Data In / Data Out</td>
<td>read/write</td>
</tr>
<tr>
<td>0006H</td>
<td>Bus Status / Auxiliary Command</td>
<td>read/write</td>
</tr>
<tr>
<td>0005H</td>
<td>Serial Poll register</td>
<td>read</td>
</tr>
<tr>
<td>0004H</td>
<td>Interrupt Status 1 / Interrupt M</td>
<td>read/write</td>
</tr>
</tbody>
</table>

For more information on this part, see the TMS9914 Data Sheet, Texas Instruments Incorporated.

## MPSC - Datacomm Controller (7201/8274)

<table>
<thead>
<tr>
<th>Address</th>
<th>Register</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0003H</td>
<td>Channel B Control Reg</td>
<td>read/write</td>
</tr>
<tr>
<td>0002H</td>
<td>Channel B Data Reg</td>
<td>read/write</td>
</tr>
<tr>
<td>0001H</td>
<td>Channel A Control Reg</td>
<td>read/write</td>
</tr>
<tr>
<td>0000H</td>
<td>Channel A Data Reg</td>
<td>read/write</td>
</tr>
</tbody>
</table>

For more information on programming this part, see the "PD7201 Multiprotocol Serial Communications Controller Technical Manual", NEC Electronics U.S.A. Incorporated, or 8274 Data Sheet, Intel Corporation.
The software environment of the HP 150 is covered in this section. Included is a discussion of the operating system, MS-DOS, and HP's enhancements to it, installable and BIOS devices, and the disc format and directory structure including drive specific capacity and organizational information.
## CONTENTS

Operating System Structure  
- The Command Processor  
- The Personal Applications Manager (P.A.M.)  
- Application Programs  
- Basic Disc Operating System (RDOS)  
- Basic Input/Output System (BIOS)  

Operating System Memory Usage  
- Operating System Memory Map  
- Interrupt Vectors  
- MS-DOS Interrupts  
- HP 150 Hardware Interrupts  
- Firmware Variables  
- BIOS and BDOS  
- Disc Buffer Cache  
- File Control Blocks (FCBs)  
- Fields of the FCB.  
- Installable Device Drivers  
- PAMCODE.EXE or Resident Portion of COMMAND.COM  
- Application Program Area (Program Segment)  
- Program Segment Prefix (PSP) Control Block  
- How A Program Terminates  
- Conditions In Effect When A Program Receives Control  
- Transient Portion of COMMAND.COM

### HP 150 Devices
- Logical (Mappable) Devices  
- Physical Devices  
- Mapping Logical to Physical Devices  
- The Device Configuration Utility  
- Installable Devices  
- Character Devices and Block Devices  
- How Application Programs Can Get to Devices  
- Device Driver Structure  
- Pointer to Next Device Field  
- Attribute Field  
- Strategy and Interrupt Routines  
- Name Field  
- Device List  
- How to Create a Device Driver  

### How MS-DOS Calls a Device Driver
- Request Header  
- Unit Code  
- Command Code Field  
- Status Word  
- Device Driver Functions and Parameters  
- Init  
- Media Check  
- Build BPP (BIOS Parameter Block)  
- Read or Write  
- Non Destructive Read No Wait  
- Status
<table>
<thead>
<tr>
<th>CONTENTS (Cont.)</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flush</td>
<td>5-38</td>
</tr>
<tr>
<td>HP 150 Installable Device Driver Example</td>
<td>5-39</td>
</tr>
<tr>
<td>AGIOS: I/O Control of the Con Device</td>
<td>5-49</td>
</tr>
<tr>
<td>The Alpha/Graphic Input/Output System (AGIOS)</td>
<td>5-49</td>
</tr>
<tr>
<td>Accessing the AGIOS</td>
<td>5-49</td>
</tr>
<tr>
<td>BIOS and its Devices</td>
<td>5-51</td>
</tr>
<tr>
<td>Introduction</td>
<td>5-51</td>
</tr>
<tr>
<td>The CONFIG.SYS File</td>
<td>5-55</td>
</tr>
<tr>
<td>Disc Format and Directory Structure</td>
<td>5-57</td>
</tr>
<tr>
<td>Physical Disc Format</td>
<td>5-57</td>
</tr>
<tr>
<td>Disc Media Storage Capacity</td>
<td>5-57</td>
</tr>
<tr>
<td>Disc Sector Allocation</td>
<td>5-58</td>
</tr>
<tr>
<td>Header Record</td>
<td>5-59</td>
</tr>
<tr>
<td>Boot Sector</td>
<td>5-61</td>
</tr>
<tr>
<td>File Allocation Table (FAT)</td>
<td>5-62</td>
</tr>
<tr>
<td>Disc Clusters</td>
<td>5-62</td>
</tr>
<tr>
<td>FAT Structure</td>
<td>5-62</td>
</tr>
<tr>
<td>How to Use the File Allocation Table</td>
<td>5-63</td>
</tr>
<tr>
<td>MS-DOS Disc Directory</td>
<td>5-63</td>
</tr>
</tbody>
</table>
OPERATING SYSTEM STRUCTURE

This chapter deals with the operating system on the HP 150, MS-DOS from Microsoft Corporation, and its interface with the HP 150 hardware and firmware.

The operating system can be partitioned into several layers:

---
| Command | Invokes | Application | Invokes | Personal |
| Processor | ------- | Program | ------- | Applications |
| (COMMAND.COM) | (xxxxxxxx.EXE) | | | (PAMCODE.EXE) |
---

| Basic Disc Operating System (BDOS) |
---

| Basic Input/Output System (BIOS) |
---

| Firmware (Read Only Memory) |
---

| Hardware |
The Command Processor

The Command Processor is a program file called COMMAND.COM. When running under MS-DOS COMMANDS mode, this program acts as the user interface to the system. Its function is to read, parse, and execute command line inputs such as "COPY A:*.* B:". Command lines may come from the keyboard or from a batch file on disc.

The Command Processor contains a number of built-in commands such as COPY, DIR, DEL, and REN. That is, these commands are recognized and acted upon directly by the COMMAND.COM program. Given an input line the command processor first checks for a built-in command. If the input line was not a directive for one of the built-in commands, the first word of the command line is checked against filenames with the extensions .COM, .EXE, and .BAT (in that order) on the default disc drive. Files with extensions .COM or .EXE are executable programs. .COM files are single group, non-relocatable programs. .EXE files are relocatable. .BAT files are text files containing one or more command lines which are processed sequentially by the Command Processor.

The Command Processor consists of three parts: the resident, initialization, and transient parts. The resident part handles all standard MSDOS errors including the CONTROL-C and Fatal Error Abort interrupts, plus it contains the code necessary to reload its transient part if necessary. The initialization portion contains the code necessary to process the AUTOEXEC file and determine the segment at which programs are to be loaded. This part of the command module is overlaid by the first program it loads in. The transient part of the command module contains all of the internal command processors and the batch file processor. This portion of the command module displays the disc prompt, reads commands from the keyboard or batchfile, and processes them.

The Personal Applications Manager (P.A.M.)

P.A.M. is similar in structure to the MS-DOS Command Processor, COMMAND.COM. However, rather than being driven by MS-DOS command line-type inputs, the Personal Applications Manager presents a friendly face to the user through touchscreen menus. Together with the File Manager, it offers essentially all the functionality of the generic MS-DOS command processor, however it is much easier for the novice user to operate. The CONFIG.SYS file indicates to MS-DOS that it should load and execute PAMCODE.EXE rather than COMMAND.COM on the HP 150. See the CONFIG.SYS File and System Booting sections for more information.

Application Programs

Application programs are generally program files with the extension .EXE. Both the Command Processor and Personal Applications Manager can invoke application programs.
Basic Disc Operating System (BDOS)

The Basic Disc Operating System is the heart of the MS-DOS operating system. The BDOS is supplied by Microsoft Corporation and is standard across a wide variety of equipment from many vendors. BDOS presents a system-independent hardware interface for the Command Processor and applications programs. It is for this reason that applications programs written to run under MS-DOS are to a certain extent easily transported from machine to machine.

The BDOS exists as a file called MSDOS.SYS residing on all MS-DOS system discs. It is a collection of system management and input/output functions termed System Functions.

Application programs interface with the BDOS through System Function calls. These calls are made by loading 8088 registers on the HP 150 with parameters including a function number identifier, and then issuing a software interrupt.

An application program written to perform all I/O and System Management functions through the documented BDOS System Functions will be truly machine independent, and capable of being run on other machines utilizing a compatible version of MS-DOS. However many I/O intensive applications bypass BDOS for the sake of efficiency, commonly accessing lower-levels of the operating system including the Basic Input/Output System (BIOS) and even the hardware. At this point, machine dependencies and non-portability is introduced.

For details of System Functions see the HP 150 Programmers Reference Manual.

Basic Input/Output System (BIOS)

The Basic Input/Output System (BIOS) processes BDOS I/O requests. While the BDOS knows nothing of the details of the specific piece of the hardware upon which it runs, the BIOS is very knowledgeable. MS-DOS is customized to run on the HP 150 through the BIOS.

Just as the BDOS exists on system discs as a file named MSDOS.SYS, the BIOS is contained in a file called IO.SYS. These two files are read into HP 150 RAM memory during system initialization (booting) and much of them remains in memory to perform I/O and System Management functions on behalf of applications programs.

MS-DOS (the BDOS) invokes the BIOS for each I/O request by passing the BIOS a long pointer (20 bit address) to an I/O packet. This is true for all requests whether they are from MS-DOS itself or on behalf of its applications. The packet is a self-contained structure which contains all the information required to fully execute the I/O request. For more information on this interface, see "Calling a Device Driver" in this section.

The BIOS is discussed in more detail later in this chapter.
OPERATING SYSTEM MEMORY USAGE

Operating System Memory Map

High Memory

---

Transient Portion of COMMAND.COM

---

Application Program
(User) Area
(Program Segment)

---

User Stack for COMMAND.COM (256 bytes)

---

PAMCODE.EXE or resident portion of COMMAND.COM

---

Installable Device Drivers

---

File Control Blocks (FCBs)

---

Disc Buffer Cache

---

MS-DOS (BDOS)
(File MSDOS.SYS)

---

BIOS
(File IO.SYS)

---

Firmware
Variables
(RESERVED)

---

Interrupt Vectors

---

06800H

---

00400H

---

00000H

---
Interrupt Vectors

Interrupts within the HP 150 are triggered by hardware attached to the 8088 microprocessor, software interrupt instructions performed by MS-DOS, software interrupt instructions from application (user) programs, or under some special circumstances, by the 8088 itself.

Every interrupt is assigned a type code that identifies it to the 8088. Interrupts are identified as "INT n" where n (the type code) is a number between 0 and 255 inclusive. The type code is used by the 8088 to calculate a location in the memory based interrupt vector table containing the four byte address of the interrupt routine. The interrupt vector for INT 0 is at address 00000H, the vector for INT 1 is at address 00004H, and so on. The interrupt vector table can contain up to 256 vectors, one for each interrupt type.

Each entry in the table is a doubleword pointer containing the address of the procedure that is to service interrupts of that type. The higher addressed word of the pointer contains the base address of the segment containing the procedure. The lower addressed word contains the procedure's offset from the beginning of the segment. Since each entry is four bytes long, the 8088 can calculate the location of the correct entry for a given interrupt type by simply multiplying the type by four. For more information on how the 8088 processor treats interrupts and the conditions which cause Intel-reserved interrupts, see the IAPX86/88, 186/188 User's Manual, Programmer's Reference, Intel Corporation, May 1983.

The 256 interrupt types are pre-allocated for the HP 150. The following table describes that allocation.
<table>
<thead>
<tr>
<th>Starting Address</th>
<th>Interrupt Vector Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>003FCH</td>
<td>Type 255 Pointer - (**** not used ****)</td>
<td>/</td>
</tr>
<tr>
<td>00200H</td>
<td>Type 128 Pointer - (**** not used ****)</td>
<td>| /</td>
</tr>
<tr>
<td>001FCH</td>
<td>Type 127 Pointer - (RESERVED)</td>
<td>Reserved</td>
</tr>
<tr>
<td>00120H</td>
<td>Type 72 Pointer - (RESERVED)</td>
<td></td>
</tr>
<tr>
<td>0011CH</td>
<td>Type 71 Pointer - HP 150 Hardware (IR7)</td>
<td>/</td>
</tr>
<tr>
<td>0010CH</td>
<td>Type 67 Pointer - HP 150 Hardware (IR3)</td>
<td>used by HP</td>
</tr>
<tr>
<td>0010AH</td>
<td>Type 66 Pointer - HP 150 Hardware (IR2)</td>
<td>hardware.</td>
</tr>
<tr>
<td>0010AH</td>
<td>Type 65 Pointer - HP 150 Hardware (IR1)</td>
<td></td>
</tr>
<tr>
<td>00100H</td>
<td>Type 64 Pointer - HP 150 Hardware (IR0)</td>
<td>/</td>
</tr>
<tr>
<td>00100H</td>
<td>Type 63 Pointer - (**** not used ****)</td>
<td>/</td>
</tr>
<tr>
<td>000A8H</td>
<td>Type 42 Pointer - (**** not used ****)</td>
<td></td>
</tr>
<tr>
<td>000A8H</td>
<td>Type 41 Pointer - MSDOS Interrupt 29</td>
<td>Reserved</td>
</tr>
<tr>
<td>000A0H</td>
<td>Type 40 Pointer - MSDOS Interrupt 28</td>
<td>for</td>
</tr>
<tr>
<td>00098H</td>
<td>Type 39 Pointer - MSDOS Interrupt 27</td>
<td>MicroSoft</td>
</tr>
<tr>
<td>00088H</td>
<td>Type 34 Pointer - MSDOS Interrupt 22</td>
<td></td>
</tr>
<tr>
<td>00088H</td>
<td>Type 33 Pointer - MSDOS Interrupt 21</td>
<td></td>
</tr>
<tr>
<td>00080H</td>
<td>Type 32 Pointer - MSDOS Interrupt 20</td>
<td>/</td>
</tr>
<tr>
<td>0007CH</td>
<td>Type 31 Pointer - (**** not used ****)</td>
<td>/</td>
</tr>
<tr>
<td>00010H</td>
<td>Type 5 Pointer - (**** not used ****)</td>
<td>Reserved</td>
</tr>
<tr>
<td>00000H</td>
<td>Type 0 Pointer - Intel Dedicated</td>
<td>for</td>
</tr>
</tbody>
</table>

5-6
MS-DOS INTERRUPTS

Interrupt 27H: Terminate But Stay Resident  
Interrupt 26H: Absolute Disc Write  
Interrupt 25H: Absolute Disc Read  
Interrupt 24H: Fatal Error Abort Address  
Interrupt 23H: CONTROL-C Exit Address  
Interrupt 22H: Terminate Address  
Interrupt 21H: Function Request  
Interrupt 20H: Program Terminate

HP 150 HARDWARE INTERRUPTS

IR7: Real Time Clock (MM58167A)  
IR6: Not Used (Tied High)  
IR5: HPIB Controller (9914)  
IR4: *Integral Printer, Accessory Slot NOCINT (Low Priority Open Collector)  
IR3: Keyboard and Touchscreen  
IR2: Not Used (Tied High)  
IR1: *MPSC (Datacomm Controller), Accy. Slot NDCOCINT (High Priority O/C)  
IR0: Video Controller (9007)

* These interrupts may be initiated from an accessory board by using the NOCINT and NDCOCINT open collector lines. Either may be used, IR1 has higher priority than IR4 (IR1 will be serviced prior to IR2-IR7). See Accessory Board Subsystem in the Hardware Subsystems section of this manual.

Firmware Variables

The firmware uses the RAM space between 00400H and 067FFH for jump vectors and other working data storage. See Section 6, "System Firmware" for more information.

BIOS and BDOS

The Basic Input/Output System (BIOS) and the Basic Disc Operating System (BDOS) are loaded from disc during initialization. BDOS is loaded following BIOS loading and initialization, and overlays the initialization portion of BIOS. The origin address of MS-DOS is BIOS revision dependent.

Disc Buffer Cache

The Disc Buffer Cache is an area used for buffering of disc data for all drives. It includes portions or all of the File Allocation Tables (depending upon cache size), the root directory, and data for non-sector oriented reads and writes. The size of the Disc Buffer Cache may be
altered from the default by the "BUFFERS=<number>" command in the CONFIG.SYS file. See the CONFIG.SYS file information in this section for a description of the "BUFFERS" command.

File Control Blocks (FCBs)

File control blocks contain file information taken from the directory, and include pointers marking an access position.

Two types of FCBs exist. An unopened FCB is one that contains only a drive specifier and a filename, which can contain wild card characters (* and ?). An opened FCB contains all fields filled by the MS-DOS 'Open File' system function call.

Space for two FCBs is allocated in the Program Segment Prefix (in the application program area). Space is allocated above the Disc Buffer Cache for a number of additional FCBs. The "FILES=<number>" command in the CONFIG.SYS file can be used to alter the amount of space allocated for FCB's in this area. See the CONFIG.SYS file information in this section for a description of the "FILES" command. Note that each open file requires its own FCB.

FIELDS OF THE FCB. The FCB is structured as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Offset</th>
<th>Hex</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drive number</td>
<td>1</td>
<td>00H</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Filename</td>
<td>8</td>
<td>01-08H</td>
<td>1-8</td>
<td></td>
</tr>
<tr>
<td>Extension</td>
<td>3</td>
<td>09-0BH</td>
<td>9-11</td>
<td></td>
</tr>
<tr>
<td>Current block</td>
<td>2</td>
<td>0CH,0DH</td>
<td>12,13</td>
<td></td>
</tr>
<tr>
<td>Record size</td>
<td>2</td>
<td>0EH,0FH</td>
<td>14,15</td>
<td></td>
</tr>
<tr>
<td>File size</td>
<td>4</td>
<td>10-13H</td>
<td>16-19</td>
<td></td>
</tr>
<tr>
<td>Date of last write</td>
<td>2</td>
<td>14H,15H</td>
<td>20,21</td>
<td></td>
</tr>
<tr>
<td>Time of last write</td>
<td>2</td>
<td>16H,17H</td>
<td>22,23</td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td>8</td>
<td>18-1FH</td>
<td>24-31</td>
<td></td>
</tr>
<tr>
<td>Current record</td>
<td>1</td>
<td>20H</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>Relative record</td>
<td>4</td>
<td>21-24H</td>
<td>33-36</td>
<td></td>
</tr>
</tbody>
</table>
Drive Number.
Specifies the disk drive; 1 means drive A: and 2 means drive B: and so on.
If the FCB is to be used to create or open a file, this field can be set to 0 to specify the default drive; the Open File system call Function (0FH) sets the field to the number of the default drive.

Filename.
Eight characters, left-aligned and padded (if necessary) with blanks. If you specify a reserved device name (such as PRW), do not put a colon at the end.

Extension.
Three characters, left-aligned and padded (if necessary) with blanks. This field can be all blanks (no extension).

Current Block.
Points to the block (group of 128 records) that contains the current record. This field and the Current Record field (offset 20H) make up the record pointer. This field is set to 0 by the Open File system call.

Record Size.
The size of a logical record, in bytes. Set to 128 by the Open File system call. If the record size is not 128 bytes, you must set this field after opening the file.

FileSize.
The size of the file, in bytes. The first word of this 4-byte field is the low-order part of the size.

Date of Last Write.
The date the file was created or last updated. The year, month, and day are mapped into two bytes as follows:

<table>
<thead>
<tr>
<th>offset 15H</th>
<th>offset 14H</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td></td>
</tr>
<tr>
<td>Y Y Y Y Y Y Y M M M M D D D D D</td>
<td></td>
</tr>
<tr>
<td>year</td>
<td>month</td>
</tr>
</tbody>
</table>
Time of Last Write.
The time the file was created or last updated. The hour, minutes, and
seconds are mapped into two bytes as follows:

+---------------------------------+---------------------------------+
<table>
<thead>
<tr>
<th>offset 17H</th>
<th>offset 16H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1      14 13 12 11 10 9 8</td>
<td>7      6 5 4 3 2 1 0</td>
</tr>
<tr>
<td>-------+-------+-------+-------+</td>
<td>-------+-------+-------+-------+</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------</td>
<td></td>
</tr>
<tr>
<td>hour</td>
<td>minute</td>
</tr>
</tbody>
</table>

Reserved. These fields are reserved for use by MS-DOS.

Current Record.
Points to one of the 128 records in the current block. This field and the
Current Block field (offset 0CH) make up the record pointer. This field is
not initialized by the Open File system call. You must set it before doing
a sequential read or write to the file.

Relative Record.
Points to the currently selected record, counting from the beginning of the
file (starting with 0). This field is not initialized by the Open File
system call. You must set it before doing a random read or write to the
file. If the record size is less than 64 bytes, both words of this field
are used; if the record size is 64 bytes or more, only the first three
bytes are used.

NOTE

If you use the FCB at offset 05CH of the Program Segment Prefix, the last
byte of the Relative Record field is the first byte of the unformatted
parameter area that starts at offset 80H. This is the default Disk
Transfer Address.

Installable Device Drivers

The code for devices additional to those implemented in the BIOS is loaded above
the Disc Buffer Cache. The CONFIG.SYS file contains declarations of device
driver files, and these files are loaded from disc and sequentially linked at system
initialization time. For more information on this process and device drivers in
general, see the "HP 150 Devices", the "CONFIG.SYS File" and "Operating System
Initialization/Booting" discussions later in this section.
PAMCODE.EXE or Resident Portion of COMMAND.COM

The resident portion of COMMAND.COM contains the code necessary to determine if the transient portion needs to be reloaded (a checksum test), and contains the code to do the reloading of that transient portion. It also contains the interrupt 22H, 23H, and 24H handlers. PAMCODE.EXE is fully resident, that is it does not contain a transient portion in high memory as does COMMAND.COM. As such, if PAMCODE.EXE is declared as the SHELL in the CONFIG.SYS file, quite a deal of system memory will be consumed by this user interface.

Application Program Area (Program Segment)

When an external command is typed, or when a program is executed through P.A.M. or through the EXEC system call, MS-DOS determines the lowest available free memory address to use as the start of the program. This area is called the Program Segment.

The first 256 bytes of the Program Segment are set up by the EXEC system call to use as the Program Segment Prefix (PSP) Control Block. The program is then loaded following this block. An .EXE file with minalloc and maxalloc both set to zero is loaded as high as possible.
PROGRAM SEGMENT PREFIX (PSP) CONTROL BLOCK At offset 0 within the program segment, MS-DOS builds the Program Segment Prefix (PSP) control block. The format of this block is as follows:

```
| 000H+---------------------------------------+ 008H+---------------------------------------+ 010H+---------------------------------------|
| INT 20H  | End of INT 20H alloc. | Reserved | Long call to MS-DOS function dispatcher (5 bytes) |
| 008H+---------------------------------------| Terminates address (IP, CS) | CTRL-C exit address (IP) |
| 010H+---------------------------------------| CTRL-C exit address (CS) | Hard error exit address (IP, CS) |

Used by MS-DOS

02CH
05CH

Formatted Parameter Area 1 formatted as standard unopened FCB 06CH

Formatted Parameter Area 2 formatted as standard unopened FCB (overlaid if FCB 05CH is opened)

Unformatted Parameter Area
(default Disk Transfer Area)

```

NOTE

Programs must not alter any part of the Program Segment Prefix control block below offset 05CH.
HOW A PROGRAM TERMINATES

A user (application) program may return to EXEC by one of four methods:

1. A long jump to offset 0 in the Program Segment
   Prefix (PSP)

2. By issuing an INT 20H with CS:0 pointing to the PSP

3. By issuing an INT 21H with register AH=0 with CS:0
   pointing at the PSP, or 4CH and no restrictions on CS

4. By a long call to location 50H in the Program Segment
   Prefix with AH=0 or Function Request 4CH

NOTE

It is the responsibility of all programs to ensure that the CS register contains
the segment address of the Program Segment Prefix when terminating via any of
these methods, except Function Request 4CH. For this reason, using Function
Request 4CH is the preferred method.

All four methods result in transferring control to the program that issued the
EXEC. During this returning process, Interrupts 22H, 23H, and 24H (Terminate
Address, CONTROL-C Exit Address, and Fatal Error Abort Address) addresses are
restored from the values saved in the Program Segment Prefix of the terminating
program. Control is then given to the terminate address. If this is a program
returning to COMMAND.COM, control transfers to its resident portion. If a batch
file was in process, it is continued; otherwise, COMMAND.COM performs a checksum
on the transient part, reloads it if necessary, then issues the system prompt
and waits for you to type the next command.

CONDITIONS IN EFFECT WHEN A PROGRAM RECEIVES CONTROL

For All Programs.

The segment address of the passed environment is contained at offset 2CH in the
Program Segment Prefix.

The environment is a series of ASCII strings (totaling less than 32K) in the
form:

    NAME=parameter

Each string is terminated by a byte of zeros, and the set of strings is
terminated by another byte of zeros. The environment built by the command
processor contains at least a COMSPEC=string (the parameters on COMSPEC define
the path used by MS-DOS to locate COMMAND.COM on disk). The last PATH and
PROMPT commands issued will also be in the environment, along with any
environment strings defined with the MS-DOS SET command.
The environment that is passed is a copy of the invoking process environment. If your application uses a "keep process" concept, you should be aware that the copy of the environment passed to you is static. That is, it will not change even if subsequent SET, PATH, or PROMPT commands are issued.

Offset 50H in the Program Segment Prefix contains code to call the MS-DOS function dispatcher. By placing the desired function request number in AH, a program can issue a far call to offset 50H to invoke an MS-DOS function, rather than issuing an Interrupt 21H. Since this is a call and not an interrupt, MS-DOS may place any code appropriate to making a system call at this position. This makes the process of calling the system portable.

The Disk Transfer Address (DTA) is set to 80H (default DTA in the Program Segment Prefix).

File control blocks at 5CH and 6CH are formatted from the first two parameters typed when the command was entered. If either parameter contained a pathname, then the corresponding FCB contains only the valid drive number. The filename field will not be valid.

An unformatted parameter area at 81H contains all the characters typed after the command (including leading and imbedded delimiters), with the byte at 80H set to the number of characters. If the <, >, or parameters were typed on the command line, they (and the filenames associated with them) will not appear in this area; redirection of standard input and output is transparent to applications.

Offset 6 (one word) contains the number of bytes available in the segment.

Register AX indicates whether or not the drive specifiers (entered with the first two parameters) are valid, as follows:

AL=FFH if the first parameter contained an invalid drive specifier (otherwise AL=00H)

AH=FFH if the second parameter contained an invalid drive specifier (otherwise AH=00H)

Offset 2 (one word) contains the segment address of the first byte of unavailable memory. Programs must not modify addresses beyond this point unless they were obtained by allocating memory via the Allocate Memory system call (Function Request 48H).

For .EXE Executable Programs.

DS and ES registers are set to point to the Program Segment Prefix. CS,IP,SS, and SP registers are set to the values passed by MS-LINK.

For .COM Executable Programs.

All four segment registers contain the segment address of the initial allocation block that starts with the Program Segment Prefix control block.
All of user memory is allocated to the program. If the program invokes another program through Function Request 4BH, it must first free some memory through the Set Block (4AH) function call, to provide space for the program being executed.

The Instruction Pointer (IP) is set to 100H.

The Stack Pointer register is set to the end of the program's segment. The segment size at offset 6 is reduced by 100H to allow for a stack of that size.

A word of zeros is placed on top of the stack. This is to allow a user program to exit to COMMAND.COM by doing a RET instruction last. This assumes, however, that the user has maintained his stack and code segments.

**Transient Portion of COMMAND.COM**

These are the command interpreter, internal commands and batch processing portions of COMMAND.COM. This area may be legally destroyed (overlayed) by an application program, and if it is then it is reloaded from disc at application program termination time.
HP 150 DEVICES

There are two classes of "devices" associated with the HP 150. These are logical devices and physical devices.

Logical (Mappable) Devices

Logical devices can be thought of as generic types of input/output channels. Logical devices are named in a symbolic manner. For example, the device name "PRN" refers to printer, the name "CON" refers to console. The following logical device names are supported by the operating system BIOS Version A.01.02:

* "CON" - Console Device
  "COM1" - Primary Communication Device
  "COM2" - Secondary Communication Device
  "AUX" - Auxiliary Device
  "PRN" - Printer Device
* "CLOCK" - Clock Device
  "LST" - List Device
  "PLT" - Plotter Device
* "HPIBDEV" - HPIB Device
* "INT" - Internal Printer Device
  "LPT1" - Parallel Printer Device 1 (another name for PRN)
  "LPT2" - Parallel Printer Device 2 (another name for LST)
  "LPT3" - Parallel Printer Device 3 (another name for AUX)
"A", "B", "C"... - Disc Drives

Note: * These named devices have actual physical devices associated directly with them and thus are not mappable. All of the other named devices are mappable.

Physical Devices

The physical class of devices on the HP 150 are the hardware input/output entities.

These include the

- Serial Communications Port 1
- Serial Communications Port 2
- Real Time Clock
- Keyboard
- Display Screen
- Integral Printer
- HPIB Port.

Mapping Logical to Physical Devices

The HP 150 personal computer is uniquely flexible when it comes to the assignment of logical to physical devices. The logical devices may be "mapped"
to physical devices. In this respect, the HP 150 differs from most common personal computers. Some other common personal computers for example have fixed logical to physical device mapping. The serial communications devices COM1 and COM2 are permanently mapped to specific hardware I/O addresses carried on internal board connectors. While this presents a simple system architecture, it also carries with it a rather rigid one.

**The Device Configuration Utility**

The HP 150 has a piece of non-volatile memory which contains active logical to physical mapping information. An applications program called "DEVCONFG.EXE" allows you to change the mapping stored here. DEVCONFG is a menu driven application and its usage is described in the HP 150 Owner's Guide. Conceptually, the mapping scheme looks like this:
Installable Devices

There is really a third class of devices on the HP 150. These devices are termed "installable devices". They combine the attributes of both logical and physical devices. Installable devices can be created by the system programmer to implement I/O. Installable devices are given names just as the logical devices are however installable devices are implemented outside the BIOS. An example of an installable device might be "NET" -- a network.

Installable devices are really entities unto themselves. They are not mappable (through the CMOS logical to physical device mapping tables) in current HP 150 implementations nor are they recognized by the device configuration utility. Upcoming versions of the operating system will allow installable devices to be mapped into the system device configuration however.

An installable device consists of some executable object code termed a "device driver". The code is stored on the operating system disc as a file with the extension .COM (for example NETWORK.COM). The device driver generally implements all levels of the I/O interface from its BDOS interface down to the hardware itself.

Installable devices can be thought of as extensions to the BIOS. They allow for I/O expansion of an MS-DOS based system. The device architecture is described in the following diagram.
Installable devices share the same calling conventions as the BIOS devices. A discussion of how to implement installable device drivers appears later in this, the system software section.
Character Devices and Block Devices

There are two kinds of devices:

Character devices
Block devices

Character devices are designed to perform serial character I/O like CON, AUX, and PRN. These devices are named (i.e., CON, AUX, CLOCK, etc.), and users may open channels (handles or FCBs) to do I/O to them.

Block devices are the "disk drives" on the system. They can perform random I/O in pieces called blocks (usually the physical sector size). These devices are not named as the character devices are, and therefore cannot be opened directly. Instead they are identified via the drive letters (A:, B:, C:, etc.).

Block devices also have units. A single driver may be responsible for one or more disk drives. For example, block device driver ALPHA may be responsible for drives A:, B:, C: and D:. This means that it has four units (0-3) defined and, therefore, takes up four drive letters. If driver ALPHA is the first block driver in the device list, and it defines 4 units (0-3), then they will be A:, B:, C: and D:. If BETA is then the second block driver and defines three units (0-2), then they will be E:, F: and G:, and so on. MS-DOS 2.0 is not limited to 16 block device units, as previous versions were. The theoretical limit is 63 (26 - 1), but it should be noted that after 26 the drive letters are unconventional (such as ], \, and .).

NOTE

Character devices cannot define multiple units because they have only one name.

How Application Programs Can Get to Devices

The BDOS (the heart of MS-DOS) serves as an interface for dealing with devices. Many of the MS-DOS System Functions access devices implemented in the BIOS. These System Functions include functions to input and output characters to the CON, AUX, and PRN devices, functions to perform block I/O (reads and writes) to character (CON, AUX, PRN) and block (A,B,C...etc.) devices, and functions to perform block device file directory (create, open, close) operations.

A special I/O control function allows control information to be passed to devices. This may be used for example to perform special CON device functions including the whole AGIOS function library. MS-DOS also allows access to installed devices through many of its system functions.

Application programs for the sake of portability should gain access to devices through the MS-DOS (BDOS) System Function Calls. See "Accessing Devices through MS-DOS" later in this chapter for more information.
Device Driver Structure

A device driver is a binary file with all of the code in it to manipulate the hardware and provide a consistent interface to MS-DOS. In addition, it has a special header at the beginning that identifies it as a device, defines the strategy and interrupt entry points, and describes various attributes of the device.

**NOTE**

For device drivers, the file must not use the ORG 100H (like .COM files). Because it does not use the Program Segment Prefix, the device driver is simply loaded; therefore, the file must have an origin of zero (ORG 0 or no ORG statement).

A device header is required at the beginning of a device driver. A device header looks like this:

```
<table>
<thead>
<tr>
<th>DWORD pointer to next device</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Must be set to -1)</td>
</tr>
<tr>
<td>----------------------------</td>
</tr>
<tr>
<td>WORD attributes</td>
</tr>
<tr>
<td>Bit 15 = 1 if char device 0 is blk</td>
</tr>
<tr>
<td>if bit 15 is 1</td>
</tr>
<tr>
<td>Bit 0 = 1 if current sti device</td>
</tr>
<tr>
<td>Bit 1 = 1 if current sto output</td>
</tr>
<tr>
<td>Bit 2 = 1 if current NUL device</td>
</tr>
<tr>
<td>Bit 3 = 1 if current CLOCK dev</td>
</tr>
<tr>
<td>Bit 4 = 1 if special</td>
</tr>
<tr>
<td>Bits 15-12 Reserved, must be set to 0</td>
</tr>
<tr>
<td>Bit 14 is the IOCTL bit</td>
</tr>
<tr>
<td>Bit 13 is the NON IBM FORMAT bit</td>
</tr>
<tr>
<td>----------------------------</td>
</tr>
<tr>
<td>WORD pointer to device’s strategy entry point</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>WORD pointer to device’s interrupt entry point</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>8-BYTE character device name field</td>
</tr>
<tr>
<td>Character devices set a device by name.</td>
</tr>
<tr>
<td>For block devices the first byte is</td>
</tr>
<tr>
<td>the number of units, the other 7 are not</td>
</tr>
<tr>
<td>used</td>
</tr>
</tbody>
</table>
```

Note that the device entry points are words. They must be offsets from the same segment number used to point to this table. For example, if XXX:YYY points to the start of this table, then XXX:strategy and XXX:interrupt are the entry points.
POINTER TO NEXT DEVICE FIELD The pointer to the next device header field is a double word field (offset followed by segment) that is set by MS-DOS to point at the next driver in the system list at the time the device driver is loaded. See the "Device List" discussion which appears later in this chapter. It is important that this field be set to -1 prior to load (when it is on the disk as a file) unless there is more than one device driver in the file. If there is more than one driver in the file, the first word of the double word pointer should be the offset of the next driver’s Device Header.

NOTE

If there is more than one device driver in the installable device driver file, the last driver in the file must have the pointer the next Device Header field set to -1.

ATTRIBUTE FIELD

The attribute field is used to tell the system whether this device is a block or character device (bit 15). Most other bits are used to give selected character devices certain special treatment. (Note that these bits mean nothing on a block device). For example, assume that a user has a new device driver that he wants to be the standard input and output. Besides installing the driver, he must tell MS-DOS that he wants his new driver to override the current standard input and standard output (the CON device). This is accomplished by setting the attributes to the desired characteristics, so he would set bits 0 and 1 to 1 (note that they are separate!). Similarly, a new CLOCK device could be installed by setting that attribute. (Refer to "The CLOCK Device", in this chapter for more information.) Although there is a NUL device attribute, the NUL device cannot be reassigned. This attribute exists so that MS-DOS can determine if the NUL device is being used.

The NON IBM FORMAT bit applies only to block devices and affects the operation of the BUILD BPB (Bios Parameter Block) device call. (Refer to "Media Check" and "Build BPB" in the "Calling a Device Driver" section later in this chapter for further information on this call.)

The other bit of interest is the IOCTL bit, which has meaning on character and block devices. This bit tells MS-DOS whether the device can handle control strings (via the IOCTL system call, Function 44H).

If a driver cannot process control strings, it should initially set this bit to 0. This tells MS-DOS to return an error if an attempt is made (via Function 44H) to send or receive control strings to this device. A device which can process control strings should initialize the IOCTL bit to 1. For drivers of this type, MS-DOS will make calls to the IOCTL INPUT and OUTPUT device functions to send and receive IOCTL strings.

The IOCTL functions allow data to be sent and received by the device for its own use (for example, to set baud rate, stop bits, and form length), instead of passing data over the device channel as does a normal read or write. The interpretation of the passed information is up to the device, but it must not be treated as a normal I/O request.
STRATEGY AND INTERRUPT ROUTINES
These two fields are the pointers to the entry points of the strategy and interrupt routines. The strategy and interrupt routines are the actual device driver code. They are word values so they must be in the same segment as the Device Header.

The 2.0 DOS does not really make use of two entry points (it simply calls strategy, then immediately calls interrupt). This dual entry point scheme is designed to facilitate future multi-tasking versions of MS-DOS. In multi-tasking environments I/O must be asynchronous, to accomplish this the strategy routine will be called to queue (internally) a request and return quickly. It is then the responsibility of the interrupt routine to perform the actual I/O at interrupt time by picking requests off the internal queue (set up by the strategy routine), and process them. When a request is complete, it is flagged as "done" by the interrupt routine. The DOS periodically scans the list of requests looking for ones flagged as done, and "wakes up" the process waiting for the completion of the request.

In order for requests to be queued as above it is no longer sufficient to pass I/O information in registers, as was the case in earlier versions, since many requests may be pending at any one time. Therefore the new device interface uses data "packets" to pass request information. A device is called with a pointer to a packet called a Request Header, this packet is linked into a global chain of all pending I/O requests maintained by the DOS. The device then links the packet into its own local chain of requests for this particular device. The device interrupt routine picks requests of the local chain for processing. The DOS scans the global chain looking for completed requests. These packets are composed of two pieces, a static piece which has the same format for all requests (called the static request header), which is followed by information specific to the request. Thus packets have a variable size and format.

At this point it should be emphasized that MS-DOS 2.0 does not implement most of these features, as future versions will. There is no global or local queue. Only one request is pending at any one time, and the DOS waits for this current request to be completed. For 2.0 it is sufficient for the strategy routine to simply store the address of the packet at a fixed location, and for the interrupt routine to then process this packet by doing the request and returning.

Remember: The DOS just calls the strategy routine and then immediately calls the interrupt routine, it is assumed that the request is completed when the interrupt routine returns.

NAME FIELD
This is an 8-byte field that contains the name of a character device or the number of units of a block device. If it is a block device, the number of units can be put in the first byte. This is optional, because MS-DOS will fill in this location with the value returned by the driver's INIT code. Refer to "Installation of Device Drivers" in this chapter for more information.
Device List

A list of all devices accessible by the operating system is maintained in RAM memory on the HP 150. The list includes each device supported by the BIOS together with any installable device drivers loaded from disc during initialization of the operating system.

During operating system initialization, the SYSINIT routine provided by Microsoft the BIOS reads a file called CONFIG.SYS located on the operating system disc. If any installable device drivers are to be installed, the CONFIG.SYS file should contain an entry such as

```
DEVICE = NETWORK.SYS
```

for each installable device. SYSINIT will load the device driver from disc (the file "NET.COM" in the above example) and link it into the list of existing device drivers. This device list contains both the boot (default BIOS) drivers and any installable device drivers found referenced in the CONFIG.SYS file. The system always processes the installable device drivers first and links them in ahead of the default ones, thus allowing the user to override default devices if he so chooses.

The format of the entries in the system's device list is the same as that of the device header required at the beginning of an installable device driver file. The details of this structure have been described, but in general it consists of the device specific information MSDOS needs to utilize each device. Such entries as the device type, and pointers to the device driver's strategy and interrupt routines are found in the system's device list structure. MSDOS requires that there be at least four devices defined by this list at boot time. They must be both the first four entries in this list and they must implement the CON, AUX, PRN, and CLOCK devices. The minimum system device list would then appear as follows:
An installable device driver with the same name as one of these BIOS resident devices will be linked in at that position, thus preempting the default device of the same name. As such, BIOS devices may be replaced by installable devices of the same name. Additional devices of either type (block or character) will be added to the system (whether they reside in BIOS or on disc) by updating the "next device driver" field of the last entry in this list to the point to the additional drivers. The SYSINIT module of the BIOS is responsible for bringing in the installable device drivers from the disc and inserting them in this list as pictured above. Once all of the installable drivers have been linked in, MS-DOS scans the preset list of default or BIOS resident device drivers, and links them into this list. The logical device names for the block devices (the labels A, B, C, etcetera) are determined by the position of the block device in the device list, and by how many units each device driver supports. The first unit of the first block device driver in the device list is assigned the label A, the second B, and so on.

How to Create a Device Driver

In order to create a device driver that MS-DOS can install, you must write a binary file with a Device Header at the beginning of the file. Note that for device drivers, the code should not be originated at 100H, but rather at 0. The link field (pointer to next Device Header) should be -1, unless there is more than one device driver in the file. The attribute field and entry points must be set correctly.

If it is a character device, the name field should be filled in with the name of that character device. The name can be any legal 8-character filename.

MS-DOS always processes installable device drivers before handling the default devices, so to install a new CON device, simply name the device CON. Remember to set the standard input device and standard output device bits in the attribute word on a new CON device. The scan of the device list stops on the first match, so the installable device driver takes precedence.

NOTE

Because MS-DOS can install the driver anywhere in memory, care must be taken in any far memory references. You should not expect that your driver will always be loaded in the same place every time.
HOW MS-DOS CALLS A DEVICE DRIVER

Request Header

When MS-DOS receives a reference to a device it scans the device list to find the first match of the referenced device. From this list it pulls the pointer to the device’s strategy routine and calls the code with a device request header. The strategy routine simply saves a pointer to the segment and offset of the start of the device request header, and returns to the DOS. The DOS then immediately calls the device’s interrupt routine, where the request header is examined and acted upon as requested. It is the responsibility of the device’s interrupt routine to examine the command code in the request header, transfer control of the routine to process that command, set the status word in the request header appropriately, fill in any command specific parameters, and return to the DOS.

A pointer to the Request header is passed to the strategy entry point in ES:BX. is made up of a piece of fixed-length data (the Static Request Header) followed by data pertinent to the operation being performed. Note that is is the device driver’s responsibility to preserve the machine state (for example, save all registers on entry and restore them on exit). There is enough room on the stack when strategy or interrupt is called to do about 20 pushes. If more stack is needed, the driver should set up its own stack.

The following figure illustrates the static portion of a Request Header.

```
<table>
<thead>
<tr>
<th>STATIC REQUEST HEADER -&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE length of record</td>
</tr>
<tr>
<td>Length in bytes of this</td>
</tr>
<tr>
<td>Request Header</td>
</tr>
<tr>
<td>-------------------------</td>
</tr>
<tr>
<td>BYTE unit code</td>
</tr>
<tr>
<td>The subunit the operation</td>
</tr>
<tr>
<td>is for (minor device)</td>
</tr>
<tr>
<td>(no meaning on character</td>
</tr>
<tr>
<td>devices)</td>
</tr>
<tr>
<td>-------------------------</td>
</tr>
<tr>
<td>BYTE command code</td>
</tr>
<tr>
<td>-------------------------</td>
</tr>
<tr>
<td>WORD status</td>
</tr>
<tr>
<td>8 bytes RESERVED</td>
</tr>
</tbody>
</table>
```

UNIT CODE The unit code field identifies which unit of a block device in your
device driver the request is for. For example, if your device driver has 3 units defined, then the possible values of the unit code field would be 0, 1, and 2.

**COMMAND CODE FIELD** The command code field in the Static Request Header specifies the function to be performed and can have the following values:

<table>
<thead>
<tr>
<th>Command Code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>INIT</td>
</tr>
<tr>
<td>1</td>
<td>MEDIA CHECK (Block only, NOP for character)</td>
</tr>
<tr>
<td>2</td>
<td>BUILD BFB (Block only, NOP for character)</td>
</tr>
<tr>
<td>3</td>
<td>IOCTL INPUT (Only called if device has IOCTL)</td>
</tr>
<tr>
<td>4</td>
<td>INPUT (read)</td>
</tr>
<tr>
<td>5</td>
<td>NON-DESTRUCTIVE INPUT NO WAIT (Char devs only)</td>
</tr>
<tr>
<td>6</td>
<td>INPUT STATUS (Char devs only)</td>
</tr>
<tr>
<td>7</td>
<td>INPUT FLUSH (Char devs only)</td>
</tr>
<tr>
<td>8</td>
<td>OUTPUT (write)</td>
</tr>
<tr>
<td>9</td>
<td>OUTPUT (write) with verify</td>
</tr>
<tr>
<td>10</td>
<td>OUTPUT STATUS (Char devs only)</td>
</tr>
<tr>
<td>11</td>
<td>OUTPUT FLUSH (Char devs only)</td>
</tr>
<tr>
<td>12</td>
<td>IOCTL OUTPUT (Only called if device has IOCTL)</td>
</tr>
</tbody>
</table>

These functions are described in detail later in the section.

**STATUS WORD** The following figure illustrates the status word in the Request Header.

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

<table>
<thead>
<tr>
<th>E</th>
<th>B</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>RESERVED</td>
<td>U</td>
</tr>
<tr>
<td>R</td>
<td>S</td>
<td>N</td>
</tr>
</tbody>
</table>

The status word is zero on entry and is set by the driver interrupt routine on return.

Bit 8 is the one bit. When set, it means the operation is complete. For MS-DOS 2.0, the driver sets it to 1 when it exits.

Bit 15 is the error bit. If it is set, then the low 8 bits indicate the error. The errors are:
0 Write protect violation
1 Unknown Unit
2 Drive not ready
3 Unknown command
4 CRC error
5 Bad drive request structure length
6 Seek error
7 Unknown media
8 Sector not found
9 Printer out of paper
A Write fault
B Read fault
C General failure

Bit 9 is the busy bit, which is set only by status calls.

For output on character devices: If bit 9 is 1 on return, a write request (if made) would wait for completion of a current request. If it is 0, there is no current request, and a write request (if made) would start immediately.

For input on character devices with a buffer: If bit 9 is 1 on return, a read request (if made) would go to the physical device. If it is 0 on return, then there are characters in the device buffer and a read would return quickly. It also indicates that something has been typed. MS-DOS assumes all character devices have an input type-ahead buffer. Devices that do not have a type-ahead buffer should always return busy=0 so that MS-DOS will not continuously wait for something to get into a buffer that does not exist.

One of the functions defined for each device is INIT. This routine is called only once when the device is installed. The INIT routine returns a location (DS:DX), which is a pointer to the first free byte of memory after the device driver (similar to "Keep Process"). This pointer method can be used to delete initialization code that is only needed once, saving on space.

Block devices are installed the same way and also return a first free byte pointer as described above. Additional information is also returned:

The number of units is returned. This determines logical device names. If the current maximum logical device letter is F at the time of the install call, and the INIT routine returns 4 as the number of units, then they will have logical names G, H, I, and J. This mapping is determined by the position of the driver in the device list, and by the number of units on the device (stored in the first byte of the device name field).

A pointer to a BPB (BIOS Parameter Block) pointer array is also returned. There is one table for each unit defined. These blocks will be used to build an internal DOS data structure for each of the units. The pointer passed to the DOS from the driver points to an array of n word pointers to BPBs, where n is the number of units defined. In this way, if all units are the same, all of the pointers can point to the same BPB, saving space. Note that this array must be protected (below the free pointer set by the return) since an internal DOS structure will be built starting at the byte
pointed to by the free pointer. The sector size defined must be less than or equal to the maximum sector size defined at default BIOS INIT time. If it isn’t, the install will fail.

The last thing that INIT of a block device must pass back is the media descriptor byte. This byte means nothing to MS-DOS, but is passed to devices so that they know what parameters MS-DOS is currently using for a particular drive unit.

Block devices may take several approaches; they may be dumb or smart. A dumb device defines a unit (and therefore an internal DOS structure) for each possible media drive combination. For example, unit = drive 0 single side, unit 1 = drive 0 double side. For this approach, media descriptor bytes do not mean anything. A smart device allows multiple media per unit. In this case, the BPB table returned at INIT must define space large enough to accommodate the largest possible media descriptor byte to pass information about what media is currently in a unit.
Device Driver Functions and Parameters

All strategy routines are called with ES:BX pointing to the Request Header. The interrupt routines get the pointers to the Request Header from the queue that the strategy routines store them in. The command code in the Request Header tells the driver which function to perform.

NOTE

All DWORD pointers are stored offset first, then segment.

INIT

Command code = 0

INIT - ES:BX ->

<table>
<thead>
<tr>
<th>13-BYTE Static Request Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE # of units</td>
</tr>
<tr>
<td>DWORD break address</td>
</tr>
<tr>
<td>DWORD pointer to BPB array</td>
</tr>
<tr>
<td>(Not set by character devices)</td>
</tr>
</tbody>
</table>

The number of units, break address, and BPB pointer are set by the driver. On entry, the DWORD that is to be set to the BPB array (on block devices) points to the character after the '=' on the line in CONFIG.SYS that loaded this device. This allows drivers to scan the CONFIG.SYS invocation line for arguments.

NOTE

If there are multiple device drivers in a single device driver file, the ending address returned by the last INIT called will be the one MS-DOS uses. It is recommended that all of the device drivers in a single device driver file return the same ending address.
MEDIA CHECK  Command Code = 1

MS-DOS calls MEDIA CHECK first for a drive unit. MS-DOS passes its current media descriptor byte (refer to the section "Media Descriptor Byte" later in this chapter).

MEDIA CHECK - ES:BX ->

--- Request Header ---
<table>
<thead>
<tr>
<th>13-BYTE Request Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE media descriptor from DPB</td>
</tr>
<tr>
<td>-----------------------------</td>
</tr>
<tr>
<td>BYTE returned</td>
</tr>
</tbody>
</table>

In addition to setting the status word, the driver must set the return byte to one of the following:

Media Not Changed - current DPB and media byte are OK.

Media Changed - current DPB and media are wrong. MS-DOS invalidates any buffers for this unit and calls the device driver to build the BPB with media byte and buffer.

Not Sure - If there are dirty buffers (buffers with changed data, not yet written to disk) for this unit, MS-DOS assumes the DPB and media byte are OK (media not changed). If nothing is dirty, MS-DOS assumes the media has changed. It invalidates any buffers for the unit, and calls the device driver to build the BPB with media byte and buffer.

Error - If an error occurs, MS-DOS sets the error code accordingly.

-1 Media has been changed
0 Don't know if media has been changed
1 Media has not been changed

If the driver can return -1 or 1 (by having a door-lock or other interlock mechanism) MS-DOS performance is enhanced because MS-DOS does not need to reread the FAT for each directory access.
BUILD BPP (BIOS PARAMETER BLOCK)

Command code = 2

MS-DOS will call BUILD BPP under the following conditions:

If Media Changed is returned

If Not Sure is returned, and there are no dirty buffers.

The BUILD BPP call also gets a pointer to a one-sector buffer. What this buffer contains is determined by the NON IBM FORMAT bit in the attribute field. If the bit is zero (device is IBM format-compatible), then the buffer contains the first sector of the first File Allocation Table (FAT). The FAT IB byte is the first byte of this buffer.

NOTE

The BPB must be the same, as far as location of the FAT is concerned, for all possible media because this first FAT sector must be read before the actual BPB is returned. If the NON IBM FORMAT bit is set, then the pointer points to one sector of scratch space (which may be used for anything).

BUILD BPP - ES:BX ->

--------------- 13-BYTE Request Header ---------------
| BYTE media descriptor from DPB |
| DWORD transfer address |
| (Points to one sector worth of scratch space or first sector of FAT depending on the value of the NON IBM FORMAT bit) |
| DWORD pointer to BPB |

If the NON IBM FORMAT bit of the device is set, then the DWORD transfer address points to a one sector buffer, which can be used for any purpose. If the NON IBM FORMAT bit is 0, then this buffer contains the first sector of the first FAT and the driver must not alter this buffer.

If IBM compatible format is used (NON IBM FORMAT BIT = 0), then the first sector of the first FAT must be located at the same sector on all possible media. This is because the FAT sector will be read before the media is actually determined. Use this mode if all you want is to read the FAT ID byte.

In addition to setting status word, the driver must set the Pointer to the BPB on return.
In order to allow for many different OEMs to read each other's disks, the following standard is suggested. The information relating to the BPB for a particular piece of media is kept in the boot sector for the media. In particular, the format of the boot sector is:

```
BUILD BPP - ES:BX ->
```

```
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3 BYTE near JUMP to boot code</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>8 BYTES OEM name and version</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>B</td>
<td>WORD bytes per sector</td>
</tr>
<tr>
<td>P</td>
<td>----------------------------</td>
</tr>
<tr>
<td>B</td>
<td>BYTE sectors per allocation unit</td>
</tr>
<tr>
<td>v</td>
<td>----------------------------</td>
</tr>
<tr>
<td>v</td>
<td>WORD reserved sectors</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>BYTE number of FATs</td>
<td></td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>WORD number of root dir entries</td>
<td></td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>WORD number of sectors in logical image</td>
<td></td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>B</td>
<td>BYTE media descriptor</td>
</tr>
<tr>
<td>P</td>
<td>----------------------------</td>
</tr>
<tr>
<td>B</td>
<td>WORD number of FAT sectors</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>WORD sectors per track</td>
<td></td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>WORD number of heads</td>
<td></td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>WORD number of hidden sectors</td>
<td></td>
</tr>
</tbody>
</table>
```

The three words at the end (sectors per track, number of heads, and number of hidden sectors) are optional. They are intended to help the BIOS understand the media. Sectors per track may be redundant (could be calculated from total size of the disk). Number of heads is useful for supporting different multi-head drives which have the same storage capacity, but different numbers of surfaces. Number of hidden sectors may be used to support drive-partitioning schemes.

Media Descriptor Byte

The last two digits of the FAT ID byte are called the media descriptor byte. Currently, the media descriptor byte has been defined for a few media types, including 5-1/4" and 8" standard disks.

Although these media bytes map directly to FAT ID bytes (which are constrained to the 8 values F8H-FFH), media bytes can, in general, be any value in the range 00H-FFH.
System Software

READ OR WRITE  \textit{Command codes} = 3,4,8,9, and 12

READ or WRITE - ES:BX (Including IOCTL) ->

<table>
<thead>
<tr>
<th>13-BYTE Request Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE media descriptor from DPB</td>
</tr>
<tr>
<td>-----------------------------</td>
</tr>
<tr>
<td>DWORD transfer address</td>
</tr>
<tr>
<td>-----------------------------</td>
</tr>
<tr>
<td>WORD byte/sector count</td>
</tr>
<tr>
<td>-----------------------------</td>
</tr>
<tr>
<td>WORD starting sector number</td>
</tr>
<tr>
<td>(Ignored on character devices)</td>
</tr>
</tbody>
</table>

In addition to setting the status word, the driver must set the sector count to the actual number of sectors (or bytes) transferred. No error check is performed on an IOCTL I/O call. The driver must correctly set the return sector (byte) count to the actual number of bytes transferred.

The Following Applies to Block Device Drivers:
Under certain circumstances the BIOS may be asked to perform a write operation of 64K bytes, which seems to be a "wrap around" of the transfer address in the BIOS I/O packet. This request arises due to an optimization added to the write code in MS-DOS. It will only manifest on user writes that are within a sector size of 64K bytes on files "growing" past the current EOF. It is allowable for the BIOS to ignore the balance of the write that "wraps around" if it so chooses. For example, a write of 10000H bytes worth of sectors with a transfer address of XXX:1 could ignore the last two bytes. A user program can never request an I/O of more than FFFFH bytes and cannot wrap around (even to 0) in the transfer segment. Therefore, in this case, the last two bytes can be ignored.
NON DESTRUCTIVE READ NO WAIT

Command code = 5

NON DESTRUCTIVE READ NO WAIT - ES:BX ->

<table>
<thead>
<tr>
<th>13-BYTE Request Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE read from device</td>
</tr>
</tbody>
</table>

If the character device returns bit = 0 (characters in buffer), then the next character that would be read is returned. This character is not removed from the input buffer (hence the term "Non Destructive Read"). Basically, this call allows MS-DOS to look ahead one input character.
STATUS

Command codes = 6 and 10

STATUS - ES:BX ->

| 13-BYTE Request Header |

All the driver must do is set the status word and the busy bit as follows:

For output on character devices: If bit 9 is 1 on return, a write request (if made) would wait for completion of a current request. If it is 0, there is no current request and a write request (if made) would start immediately.

For input on character devices with a buffer: A return of 1 means a read request (if made) would go to the physical device. If it is 0 on return, then there are characters in the device's buffer and read would return quickly. A return of 0 also indicates that the user has typed something. MS-DOS assumes that all character devices have an input type-ahead buffer. Devices that do not have a type-ahead buffer should always return busy = 0 so that the DOS will not hang waiting for something to get into a buffer which doesn't exist.
**FLUSH**  
*Command codes = 7 and 11*

FLUSH - ES:BX ->

```
| 13-BYTE Request Header |
```

The **FLUSH** call tells the driver to flush (terminate) all pending requests. This call is used to flush the input queue on character devices.
HP 150 INSTALLABLE DEVICE DRIVER EXAMPLE

This is an example of an MSDOS installable device driver for the HP 150. As an MSDOS installable device driver, this example does nothing more than support the 12 MSDOS command codes in name only, and does not attempt to implement their functions. The intention of the example is to outline the structure of an installable device driver. It is then a matter for the programmer to "fill in the blanks" to implement a functional driver. All device functions exist in name only, returning done status immediately.

<table>
<thead>
<tr>
<th>Driver Routine's Label</th>
<th>Function of the particular routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSDOS COMMAND CODES:</td>
<td>Supported MSDOS Command Codes</td>
</tr>
<tr>
<td>MSDOS_INITIALIZE</td>
<td>0 - initializes the option into the system</td>
</tr>
<tr>
<td>MSDOS_MEDIA_CHECK</td>
<td>1 - performs media check on block devices</td>
</tr>
<tr>
<td>MSDOS_BUILD_BPB</td>
<td>2 - builds the device's BIOS Parameter Block</td>
</tr>
<tr>
<td>MSDOS_IOCTL_INPUT</td>
<td>3 - performs an I/O control read from device</td>
</tr>
<tr>
<td>MSDOS_INPUT</td>
<td>4 - performs a normal destructive read</td>
</tr>
<tr>
<td>MSDOS_NON_DTSRV_INPUT</td>
<td>5 - performs a non-destructive read, no wait</td>
</tr>
<tr>
<td>MSDOS_INPUT_STATUS</td>
<td>6 - returns current input status of device</td>
</tr>
<tr>
<td>MSDOS_INPUT_FLUSH</td>
<td>7 - flushes the device's input buffers</td>
</tr>
<tr>
<td>MSDOS_OUTPUT</td>
<td>8 - performs a normal output to the device</td>
</tr>
<tr>
<td>MSDOS_OUTPUT_WITH_VERIFY</td>
<td>9 - performs output with verify to the device</td>
</tr>
<tr>
<td>MSDOS_OUTPUT_STATUS</td>
<td>10 - returns current output status of device</td>
</tr>
<tr>
<td>MSDOS_OUTPUT_FLUSH</td>
<td>11 - flushes the device's output buffers</td>
</tr>
<tr>
<td>MSDOS_IOCTL_OUTPUT</td>
<td>12 - performs I/O control output to the device</td>
</tr>
</tbody>
</table>

NOTE: All command code routines listed above exist in name only and simply return status equal to done when called.

******************************************************************************
System Software

;******************************************************************************
; ; Start of the Installable Device Driver Routine
; ; This next section starts the code segment for the rest of this module.
; ; CODE SEGMENT PUBLIC 'CODE'
; DRIVER PROC FAR
; ASSUME CS:CODE, ES:CODE, DS:CODE
; ORG 0
; BEGIN:
; START EQU $ ;
;******************************************************************************
;******************************************************************************
; ; MS DOS Installable Device Driver Header Format
; ; The following is the MS DOS installable device driver's device header.
; ; This header must be at the beginning of each installable device driver. It
; ; is used by the SYSINIT module provided by Microsoft to link this installable
; ; device driver into the system's device list. The format of this device header
; ; is as follows:
; ; | 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |
; |---------------------------------------------|
; | offset of the NEXT DEVICE DRIVER in the device list (-1 if last entry) |
; |---------------------------------------------|
; | segment of the NEXT DEVICE DRIVER in the device list (-1 if last entry) |
; |---------------------------------------------|
; | ATTRIBUTES word describing device specific characteristics |
; |---------------------------------------------|
; | offset to the device driver's STRATEGY routine |
; |---------------------------------------------|
; | offset to the device driver's INTERRUPT routine |
; |---------------------------------------------|
; | 1st 2 bytes of a character device NAME, or # of UNITS for a block device |
; |---------------------------------------------|
; | 2nd 2 bytes of a character device NAME, or nothing for a block device |
; |---------------------------------------------|
; | 3rd 2 bytes of a character device NAME, or nothing for a block device |
; |---------------------------------------------|
; | 4th 2 bytes of a character device NAME, or nothing for a block device |
; |---------------------------------------------|
;******************************************************************************

5-40
The device implemented by the driver is a simple character device, with an MS DOS strategy routine at DRIVER STRATEGY and an MS DOS interrupt routine at DRIVER INTERRUPT. The name of this device is "DRIVER". The installable device driver header for this device is as follows:

```
; The following section contains a dispatch table that describes the entry points for the routines that implement the various 12 MS DOS command codes for this example device. This table is indexed into based on the MS DOS command code in the request header, and a jump is made to the address of the routine listed in this table. The dispatch table of MS DOS command codes for this example driver is as follows:

MS DOS COMMAND CODES:
```

```
; Support MS DOS Command Codes (dispatch table)

DW MS DOS INITIALIZE ; 0 - initializes the driver into the system
DW MS DOS MEDIA CHECK ; 1 - performs media check on block devices
DW MS DOS BUILD BPB ; 2 - builds the device's BIOS Parameter Block
DW MS DOS IOCTL INPUT ; 3 - performs an I/O control read from device
DW MS DOS INPUT ; 4 - performs a normal destructive read
DW MS DOS NON DESTW INPUT ; 5 - performs a non-destructive read, no wait
DW MS DOS INPUT STATUS ; 6 - returns current input status of device
DW MS DOS INPUT FLUSH ; 7 - flushes the device's input buffers
DW MS DOS OUTPUT ; 8 - performs a normal output to the device
DW MS DOS OUTPUT WITH VERIFY ; 9 - performs output with verify to the device
DW MS DOS OUTPUT STATUS ; 10 - returns current output status of device
DW MS DOS OUTPUT FLUSH ; 11 - flushes the device's output buffers
DW MS DOS_IOCTL_OUTPUT ; 12 - performs I/O control output to the device
```
This next section contains a template for the MSDOS request header. This template is used to extract and update the necessary request header entries. The MSDOS request header's format is as shown below:

```
7 6 5 4 3 2 1 0
+---------------------------------------+
<table>
<thead>
<tr>
<th>LENGTH of the request HEADER in bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIT that this call is intended for</td>
</tr>
<tr>
<td>---------------------------------------</td>
</tr>
<tr>
<td>COMMAND CODE or function number</td>
</tr>
<tr>
<td>---------------------------------------</td>
</tr>
<tr>
<td>driver's RETURN STATUS, first of two</td>
</tr>
<tr>
<td>---------------------------------------</td>
</tr>
<tr>
<td>driver's RETURN STATUS, second of two</td>
</tr>
<tr>
<td>---------------------------------------</td>
</tr>
<tr>
<td>**********  RESERVED  **********</td>
</tr>
<tr>
<td>---------------------------------------</td>
</tr>
<tr>
<td>**********  RESERVED  **********</td>
</tr>
<tr>
<td>---------------------------------------</td>
</tr>
<tr>
<td>**********  RESERVED  **********</td>
</tr>
<tr>
<td>---------------------------------------</td>
</tr>
<tr>
<td>**********  RESERVED  **********</td>
</tr>
<tr>
<td>---------------------------------------</td>
</tr>
<tr>
<td>**********  RESERVED  **********</td>
</tr>
<tr>
<td>---------------------------------------</td>
</tr>
<tr>
<td>**********  RESERVED  **********</td>
</tr>
<tr>
<td>---------------------------------------</td>
</tr>
<tr>
<td>**********  RESERVED  **********</td>
</tr>
<tr>
<td>---------------------------------------</td>
</tr>
<tr>
<td>**********  RESERVED  **********</td>
</tr>
<tr>
<td>---------------------------------------</td>
</tr>
<tr>
<td>** Command Specific Data **</td>
</tr>
<tr>
<td>---------------------------------------</td>
</tr>
<tr>
<td>** Command Specific Data **</td>
</tr>
<tr>
<td>---------------------------------------</td>
</tr>
<tr>
<td>****************************************</td>
</tr>
</tbody>
</table>
```
Driver's MSDOS Request Header Structure

The REQUEST_HEADER structure that this module will use to access the MSDOS request header packet is defined as follows:

```assembly
REQUEST_HEADER STRUC ;request header template structure
    HEADER_LENGTH DB ? ;length of the request header
    UNIT_NUMBER DB ? ;unit number for this request
    COMMAND_CODE DB ? ;request header's command code
    RETURN_STATUS DW ? ;driver's return status word
    RESERVED DB 8 DUP (?) ;***** RESERVED *****
    MEDIA_DESCRIPTOR DB ? ;media descriptor byte
    TRANSFER_ADDRESS DD ? ;transfer address
    COUNT DW ? ;byte/sector count value
    STARTING_SECTOR DW ? ;starting sector value
REQUEST_HEADER ENDS ;end of request header template
```

Driver's Local Variables

```assembly
HEADER_POINTER ED ? ;pointer to the request header
```

Driver's Local Constants

```assembly
MAX_COMMAND_CODE EQU 012 ;MSDOS command code number maximum
```

This next section contains this driver's strategy routine. This routine is entered each time a request is made for this device.

All requests for this device first call its strategy routine and then its interrupt routine. The strategy routine is passed a pointer (in ES:BX) to the request header, which it saves in a local variable called HEADER_POINTER and then it returns to the BDOS. This example device driver's strategy routine is defined as follows:

**Inputs** - ES:BX points to the request header

**Outputs** - HEADER_POINTER points to the active request header

```
; beginning of strategy procedure

DRIVER_STRATEGY:
    MOV WORD PTR CS:[HEADER_POINTER], BX ; save offset of request header
    MOV WORD PTR CS:[HEADER_POINTER+2], ES ; save segment of request header
    RET

; end of the strategy procedure
```

This next section contains this device's MSDOS interrupt routine. This is the entry point that the BDOS will call immediately after it has called this driver's strategy routine. It is this routine's responsibility to transfer control to the appropriate routine within this module based on the command code found in the current request header. It uses this command code as an index into its MSDOS command codes dispatch table, to extract the address of the routine that can service the incoming request.

**Inputs** - HEADER_POINTER points to the request header

**Outputs** - REQUEST_HEADER [RETURN_STATUS] set appropriately
- command code specific data where specified
; device driver interrupt routine
; save the caller's AX register
; save the caller's BX register
; save the caller's CX register
; save the caller's DX register
; save the caller's DS register
; save the caller's ES register
; save the caller's DI register
; save the caller's SI register
; save the caller's BP register

; These sections of code prepare the system's registers with the values
; found in the request header before transferring control to the appropriate
; routine. The registers are prepared as follows:

; AX = AH->MEDIA_DESCRIPTOR/AL->UNIT NUMBER
; BX = OFFSET within ES segment to the request header
; CX = COUNT (bytes or sectors) of the data to be read or written
; DX = STARTING SECTOR to be read or written
; DI = TRANSFER ADDRESS of user's buffer or data
; SI = address of the start of the servicing routine
; DS = data segment address of this driver (same as CS)
; ES = segment address of the request header

; point DS:BX to the request header
; load AL with the UNIT NUMBER
; load AH with the MEDIA_DESCRIPTOR
; load CX with the COUNT value
; load DX with the STARTING SECTOR
; save AX in DI temporarily

; load AX with the COMMAND CODE
; clear off the high byte of AX
; is this a legal command code?
; if no. then error out here
; convert byte to word pointer
; point SI to MSDOS dispatch table
; index SI to the desired routine
; restore AX from DI temp. storage
; point DI to transfer addr. offset
; place code segment on the stack...
; then point DS to the code segment
; jump to the appropriate routine

; point DS:BX to the request header
; load AL with the UNIT NUMBER
; load AH with the MEDIA_DESCRIPTOR
; load CX with the COUNT value
; load AX with the COMMAND CODE
These are a set of general purpose routines that are used by all of the MSDOS command code routines.

**COMMAND CODE ERROR:**

```
        MOV   AL,3
        MOV   AH,10000001B
        JMP   EXIT_DRIVER
```

**EXIT:**

```
        MOV   AH,00000001B
```

**Driver's MSDOS Command Code Exit Routine**

This procedure is used to generate a long return to the caller. The value passed to this routine in AX is placed in the RETURN_STATUS word entry of the pending request header. Then this routine restores the stack back to the state it was when this driver was originally called, and then returns to the caller.

**INPUTS** - AX contains the RETURN_STATUS value for the pending I/O request.

```
EXIT_PROC PROC FAR

EXIT_DRIVER:
    LDS   BX,CS:[HEADER_POINTER]
    MOV   DS:[BX.RETURN_STATUS],AX
    POP   BP
    POP   SI
    POP   DI
    POP   ES
    POP   DS
    POP   DX
    POP   CX
    POP   BX
    POP   AX
    RET
```

EXIT_PROC ENDP
Driver's MSDOS Command Code Routines

The MSDOS Command Code Routines do nothing other than exit with the
RETURN_STATUS word in the request header set to done without errors.

; MSDOS MEDIA_CHECK:
NOP
JMP EXIT

; MSDOS BUILD BPB:
NOP
JMP EXIT

; MSDOS IOCTL_INPUT:
NOP
JMP EXIT

; MSDOS_NON_DSTRV_INPUT:
NOP
JMP EXIT

; MSDOS_INPUT:
NOP
JMP EXIT

; MSDOS_INPUT_STATUS:
NOP
JMP EXIT

; MSDOS_INPUT_FLUSH:
NOP
JMP EXIT

; MSDOS_OUTPUT:
NOP
JMP EXIT

; MSDOS_OUTPUT_WITH_VERIFY:
NOP
JMP EXIT

; MSDOS_OUTPUT_STATUS:
NOP
JMP EXIT

; MSDOS_OUTPUT_FLUSH:
NOP
JMP EXIT

; MSDOS Command Code # 1
;code to process MEDIA CHECK
;jump to the EXIT routine

; MSDOS Command Code # 2
;code to process BUILD BPB
;jump to the EXIT routine

; MSDOS Command Code # 3
;code to process IOCTL_INPUT
;jump to the EXIT routine

; MSDOS Command Code # 4
;code to process INPUT
;jump to the EXIT routine

; MSDOS Command Code # 5
;code to process NON_DSTRV_INPUT
;jump to the EXIT routine

; MSDOS Command Code # 6
;code to process INPUT_STATUS
;jump to the EXIT routine

; MSDOS Command Code # 7
;code to process INPUT_FLUSH
;jump to the EXIT routine

; MSDOS Command Code # 8
;code to process OUTPUT
;jump to the EXIT routine

; MSDOS Command Code # 9
;code to process OUTPUT_WITH_VERIFY
;jump to the EXIT routine

; MSDOS Command Code #10
;code to process OUTPUT_STATUS
;jump to the EXIT routine

; MSDOS Command Code #11
;code to process OUTPUT_FLUSH
;jump to the EXIT routine

5-47
MSDOS_IOCTL_OUTPUT:
    MOV     EXIT
    ;MSDOS Command Code #12
    ;code to process IOCTL OUTPUT
    ;jump to the EXIT routine

MSDOS_INITIALIZE:
    ;MSDOS Command Code #0
    LDS     BX, CS:[HEADER_POINTER]     ;point DS:BX to the request header
    MOV     WORD PTR DS:[BX.TRANSFER_ADDRESS], OFFSET MSDOS_INITIALIZE
    MOV     WORD PTR DS:[BX.TRANSFER_ADDRESS+2], CS
    JMP     EXIT

; DRIVER ENDP
;
; CODE ENDS
; START
AGIOS: I/O CONTROL OF THE CON DEVICE

The Alpha/Graphic Input/Output System (AGIOS)

The Alpha/Graphic I/O system is a set of functions which can be called from application programs. The functions help programmers manipulate the HP 150 video display, control the touchscreen, and handle the keyboard. The AGIOS functions provide a high level interface to these entities sparing the programmer of much tedium which may otherwise be necessary. The price which is sometimes paid for this convenience is performance. For this reason, Section 7, Programming the HP 150, includes some information allowing programmers access to the alpha and graphics memories in a direct manner.

Accessing the AGIOS

The Alpha/Graphic I/O System is implemented as a set of functions which are really MS-DOS System Function Call sub-functions. MS-DOS includes a function, I/O control for devices (function 44H). This function enables an IOCTL device command to an open device to be performed. The IOCTL function can be implemented by a device driver to allow control-type manipulation of the device, rather than read/write oriented operations. The HP 150 console device "CON" implements the AGIOS function set through its IOCTL call.

Most high level programming languages on the HP 150 do not allow direct manipulation of the 8088 registers. To perform an MS-DOS System Function and hence, an AGIOS function, the registers must be loaded with function-dependent data. As a result, AGIOS function calls from high level languages are usually made through assembly language subroutines combined with the higher level code at link time.

The following code is an example of an AGIOS function call written in assembly language. It is the "Execute Two Character Escape Sequence" AGIOS function.

This AGIOS function call requires a three byte buffer which contains the function code followed by the "J" parameter.

```
BUFF  | 16 | 0 | J |
```

The program segment to accomplish the clear display would look something like this in assembler:
The above example lacks elegance and structure typical of good programming practice. For example, the buffer could reside in the data segment and be accessible such that other escape sequences could be programmed. Or the function dependent argument ("J" in this case) could be passed on the stack. Nor does the above routine do any error checking. If an MS-DOS error occurs when you make an AGIOS function call, the "carry flag" is set according to the standard MS-DOS procedures. If an AGIOS error occurs, the AX register contains a non-zero value. AGIOS errors can occur when MS-DOS errors do not occur. This means that the carry flag is not set but the AX register contains a non-zero character. Therefore, you need to check both of these indicators after a function call to determine if the operation was completed successfully.

For more information on making AGIOS and MS-DOS calls from high level languages, together with some more generalized examples, refer to Section 7, Programming the HP 150.
Introduction

The HP 150's Basic I/O System (BIOS) consists of a set of twelve modules that are responsible for interacting with Microsoft's Basic Disc Operating System (BDOS) on one side, and with the HP 150 firmware routines on the other. The interface between the BDOS and the BIOS is specified by Microsoft, and the HP 150's BIOS modules implement this interface. (See "Calling a Device Driver" earlier in this section.) However, to perform the actual I/O operations they rely heavily on the services provided by the firmware. Much of what these modules do is to take function calls handed down from the BDOS, translate them into appropriate firmware calls, and call the firmware routines to execute the I/O function.

CONSOLE DEVICE

Device Names: CON

Device Functions Implemented:

0  INIT ............................. Not implemented
1  MEDIA CHECK ..................... Not implemented
2  BUILD BPP ......................... Not implemented
3  IOCTL INPUT ...................... Reserved - returns error
4  INPUT (read) ...................... Valid
5  NON-DESTRUCTIVE INPUT NO WAIT Valid
6  INPUT STATUS ..................... Not implemented
7  INPUT FLUSH ....................... Valid
8  OUTPUT (write) .................... Valid
9  OUTPUT (write) WITH VERIFY ..... Valid
10 OUTPUT STATUS .................... Not required by firmware
11 OUTPUT FLUSH ..................... Valid
12 IOCTL OUTPUT .................... Valid - AGIOS
SERIAL DATA COMMUNICATIONS DEVICES

Device Names:

COM, COM1, COM2, PRN, LST, AUX, PLT, LPT1, LPT2, LPT3

Device Functions Implemented:

<table>
<thead>
<tr>
<th>Device Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 INIT</td>
<td>Not implemented</td>
</tr>
<tr>
<td>1 MEDIA CHECK</td>
<td>Not implemented</td>
</tr>
<tr>
<td>2 BUILD BPP</td>
<td>Not implemented</td>
</tr>
<tr>
<td>3 IOCTL INPUT</td>
<td>Valid - returns on buffer empty</td>
</tr>
<tr>
<td>4 INPUT (read)</td>
<td>Valid</td>
</tr>
<tr>
<td>5 NON-DESTRUCTIVE INPUT NO WAIT</td>
<td>Valid</td>
</tr>
<tr>
<td>6 INPUT STATUS</td>
<td>Not implemented</td>
</tr>
<tr>
<td>7 INPUT FLUSH</td>
<td>Valid</td>
</tr>
<tr>
<td>8 OUTPUT (write)</td>
<td>Valid</td>
</tr>
<tr>
<td>9 OUTPUT (write) WITH VERIFY</td>
<td>Valid</td>
</tr>
<tr>
<td>10 OUTPUT STATUS</td>
<td>Valid</td>
</tr>
<tr>
<td>11 OUTPUT FLUSH</td>
<td>Not implemented</td>
</tr>
<tr>
<td>12 IOCTL OUTPUT</td>
<td>Valid</td>
</tr>
</tbody>
</table>

TIME DEVICE

Device Name:

CLOCK

Device Functions Implemented:

<table>
<thead>
<tr>
<th>Device Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 INIT</td>
<td>Not implemented</td>
</tr>
<tr>
<td>1 MEDIA CHECK</td>
<td>Not implemented</td>
</tr>
<tr>
<td>2 BUILD BPP</td>
<td>Not implemented</td>
</tr>
<tr>
<td>3 IOCTL INPUT</td>
<td>Reserved - returns an error</td>
</tr>
<tr>
<td>4 INPUT (read)</td>
<td>Valid</td>
</tr>
<tr>
<td>5 NON-DESTRUCTIVE INPUT NO WAIT</td>
<td>Not implemented - returns busy flag</td>
</tr>
<tr>
<td>6 INPUT STATUS</td>
<td>Not implemented</td>
</tr>
<tr>
<td>7 INPUT FLUSH</td>
<td>Not implemented</td>
</tr>
<tr>
<td>8 OUTPUT (write)</td>
<td>Valid</td>
</tr>
<tr>
<td>9 OUTPUT (write) WITH VERIFY</td>
<td>Valid</td>
</tr>
<tr>
<td>10 OUTPUT STATUS</td>
<td>Not implemented</td>
</tr>
<tr>
<td>11 OUTPUT FLUSH</td>
<td>Not implemented</td>
</tr>
<tr>
<td>12 IOCTL OUTPUT</td>
<td>Not implemented</td>
</tr>
</tbody>
</table>
INTEGRAL PRINTER DEVICE

Device Names:

INT, PRN, AUX, LST, LPT1, LPT2, LPT3

Driver Functions Implemented:

0  INIT ......................... Not implemented
1  MEDIA CHECK ................. Not implemented
2  BUILD BPP .................... Not implemented
3  IOCTL INPUT ................ Reserved - returns an error
4  INPUT (read) ................. Valid
5  NON-DESTRUCTIVE INPUT NO WAIT Not implemented - returns an error
6  INPUT STATUS ............... Not implemented
7  INPUT FLUSH ................. Not implemented
8  OUTPUT (write) ............. Valid
9  OUTPUT (write) WITH VERIFY ... Valid
10 OUTPUT STATUS ............. Valid
11 OUTPUT FLUSH ............... Not implemented
12 IOCTL OUTPUT ............. Not implemented

DISC, HPIB PRINTER, HPIB PLOTTER DEVICES

Device Names:

A:......L:, AUX, PRN, LST, PLT, LPT1, LPT2, LPT3

Driver Functions Implemented:

0  INIT ........................ Not implemented
1  MEDIA CHECK ................. Not implemented
2  BUILD BPP .................... Not implemented
3  IOCTL INPUT ................ Reserved - returns an error
4  INPUT (read) ................. Valid
5  NON-DESTRUCTIVE INPUT NO WAIT Not implemented - returns busy flag
6  INPUT STATUS ............... Not implemented
7  INPUT FLUSH ................. Not implemented
8  OUTPUT (write) ............. Valid
9  OUTPUT (write) WITH VERIFY ... Valid
10 OUTPUT STATUS ............. Valid
11 OUTPUT FLUSH ............... Not implemented
12 IOCTL OUTPUT ............. Not implemented
HPIBDEV DEVICE

Device Name:

HPIBDEV

Driver Functions Implemented:

0  INIT ........................... Not implemented
1  MEDIA CHECK .................... Not implemented
2  BUILD BPP ........................ Not implemented
3  IOCTL INPUT ..................... Special HP data returns
4  INPUT (read) ...................... Not implemented
5  NON-DESTRUCTIVE INPUT NO WAIT Not implemented - returns busy flag
6  INPUT STATUS ...................... Not implemented
7  INPUT FLUSH ...................... Not implemented
8  OUTPUT (write) ................... Valid
9  OUTPUT (write) WITH VERIFY ... Valid
10  OUTPUT STATUS .................. Valid
11  OUTPUT FLUSH .................... Not implemented
12  IOCTL OUTPUT .................. Valid - pass through of template & data
THE CONFIG.SYS FILE

In many cases, there are installation-specific configurations of the operating system that are needed to be set up at boot time. The user need not re-build the DOS (BDOS) to include special drivers or to include a particular number of device drivers. Drivers for additional devices may be included as separate entities. See the discussion of devices earlier in this chapter. The configuration file allows a user to configure his system without extra work.

The configuration file is simply an ASCII file that has certain commands for the operating system initialization task.

During operating system initialization (booting) a long jump to the SYSINIT routine in the BIOS SYSINIT module is made. This module (supplied with the operating system from Microsoft) will initialize the DOS and read the configuration file CONFIG.SYS, if it exists, to perform device installation and various other user settable things.

The following are a list of commands for the configuration file CONFIG.SYS:

BUFFERS = <number>
This is the number of additional sector buffers to add to the system list. The effect of several BUFFERS commands is to allocate a series of buffers. The default is BIOS specific, equal to 16 in versions A.01.02 and A.01.06. The minimum value allowed for BUFFERS is one. With Winchester disc based systems, performance may be improved by increasing the buffer or size so as to allow the entire disc FAT (File Allocation Table) into buffer memory. As such, the operating system can access the entire FAT without going to disc.

FILES = <number>
This is the number of open files that the XENIX system calls can access. The default is BIOS specific, equal to 20 in versions A.01.02 and A.01.06. If a number less than or equal to five is specified, the command is ignored.

DEVICE = <filename>
This installs the device driver in <filename> into the system list.

BREAK = <ON or OFF>
If ON is specified (the default is OFF), a check for 'C at the console input will be made every time the system is called. ON improves the ability to abort programs over previous versions of the DOS.

SWITCHCHAR = <char>
Causes the DOS to return <char> as the current switch designator character when the DOS call to return the switch character is made. Default is '/'. Note that the setting of SWITCHCHAR may effect characters used on the SHELL line (this is true of COMMAND.COM).

AVAILDEV = <TRUE or FALSE>
The default is TRUE which means both /dev/<dev> and <dev> will reference the device <dev>. If FALSE is selected, only /dev/<dev> refers to device <dev>,

5-55
<dev> by itself means a file in the current directory with the same name as one of the devices.

**SHELL = <filename>**
This begins execution of the shell (top-level command processor) from <filename>. Note that the parameters on this line are shell dependent.

A typical configuration file might look like this:

```
BUFFERS = 10
FILES = 10
DEVICE = /bin/network.sys
BREAK = ON
SWITCHAR = -
SHELL = pamcode.exe root
```
DISC FORMAT AND DIRECTORY STRUCTURE

Physical Disc Format

HP 150 disc media are partitioned physically into "tracks" which in turn are each partitioned into "sectors". Each sector is a physical portion of a track commonly containing 256 bytes of information.

Sectors are numbered 0, 1, 2, 3, .... and so on. This is not to say that sector 1 follows sector 0 physically on the disc. Sector "staggering" is employed to improve disc read and write efficiency.

The disc drive takes care of logical to physical mapping and as such the programmer need only be concerned with logical sector numbering, above.

Disc Media Storage Capacity

The following table shows the storage capacity in sectors and number of kilobytes for different types of disc drives available on the HP 150.

<table>
<thead>
<tr>
<th>Media Type</th>
<th>Total Sectors</th>
<th>Total KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1/2&quot; Single Sided Microfloppy</td>
<td>1,056</td>
<td>264</td>
</tr>
<tr>
<td>3-1/2&quot; Double Sided Microfloppy</td>
<td>1,385 *</td>
<td>709</td>
</tr>
<tr>
<td>5-1/4&quot; Double Sided Minifloppy</td>
<td>1,056</td>
<td>264</td>
</tr>
<tr>
<td>8&quot; Floppy (HP Format)</td>
<td>4,500</td>
<td>1,125</td>
</tr>
<tr>
<td>8&quot; Floppy (IBM Format)</td>
<td>2,002 **</td>
<td>250</td>
</tr>
<tr>
<td>&quot;5MB&quot; Winchester</td>
<td>18,848</td>
<td>4,712</td>
</tr>
<tr>
<td>&quot;10MB&quot; Winchester</td>
<td>37,820</td>
<td>9,455</td>
</tr>
<tr>
<td>&quot;15MB&quot; Winchester</td>
<td>56,730</td>
<td>14,182</td>
</tr>
</tbody>
</table>

* Sector size is 512 bytes.
** Sector size is 128 bytes.
   All other media have 256 byte sectors.
Disc Sector Allocation

Beginning with logical sector number 0, several contiguous sectors are reserved for system use, that is they do not contain file data. The sector map of a disc looks like this:

<table>
<thead>
<tr>
<th>Sector Number</th>
<th>Contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Header Record</td>
</tr>
<tr>
<td>1</td>
<td>Boot sector for operating system disc. FF’s for non-O.S. disc.</td>
</tr>
<tr>
<td>2-XX</td>
<td>File Allocation Table (FAT) Number 1</td>
</tr>
<tr>
<td>XX-XX</td>
<td>File Allocation Table (FAT) Number 2</td>
</tr>
<tr>
<td>XX-XX</td>
<td>Disc Directory</td>
</tr>
<tr>
<td>XX - Last</td>
<td>Disc Data Area</td>
</tr>
</tbody>
</table>

NOTE: The number of FAT, DIRECTORY and DATA sectors varies from media type to media type.
Header Record

The header record contains disc dependent data. It is always the first logical disc record. The header record is structured as follows.

<table>
<thead>
<tr>
<th>ENTRY REFERS TO</th>
<th>DISC TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>3&quot; &amp; 5&quot;</td>
<td>3&quot; dbl.</td>
</tr>
<tr>
<td>single sided</td>
<td>disc</td>
</tr>
<tr>
<td>RAM</td>
<td>8&quot;</td>
</tr>
</tbody>
</table>

| EBH, 1CH,90H    | (3 Bytes) | All discs contain this information |
|-----------------|-----------|
| "HP150"         | (8 Bytes) | All discs contain this information |

<table>
<thead>
<tr>
<th>bytes per sector</th>
<th>(W)</th>
<th>256</th>
<th>512</th>
<th>512</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>sectors per cluster*</td>
<td>(B)</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>reserved sectors</td>
<td>(W)</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>number of FATs</td>
<td>(B)</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>number of DIR entries</td>
<td>(W)</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>256</td>
</tr>
<tr>
<td>total sectors</td>
<td>(W)</td>
<td>1,056</td>
<td>1,385</td>
<td>**</td>
<td>4,500</td>
</tr>
<tr>
<td>media type = FAH</td>
<td>(B)</td>
<td>All discs use this value</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sectors per FAT</td>
<td>(W)</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>sectors per track</td>
<td>(W)</td>
<td>16</td>
<td>8</td>
<td>NA</td>
<td>30</td>
</tr>
<tr>
<td>number of heads</td>
<td>(W)</td>
<td>2</td>
<td>2</td>
<td>NA</td>
<td>2</td>
</tr>
<tr>
<td># of hidden sectors</td>
<td>(W)</td>
<td>All discs use this value</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>= 00H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## System Software

### ENTRY REFERS TO  DISC TYPE

| EBH, 1CH,90H  | 5 MB | 10 MB | 15 MB | IBM 8" |
| "HP150 "      |      |       |       |        |

| All discs contain this information |

<table>
<thead>
<tr>
<th>bytes per sector</th>
<th>sectors per cluster*</th>
<th>reserved sectors</th>
<th>number of FATs</th>
<th>number of DIR entries</th>
<th>total sectors</th>
<th>media type = FAH</th>
<th>sectors per FAT</th>
<th>sectors per track</th>
<th>number of heads</th>
<th>% of hidden sectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>(W)</td>
<td>(B)</td>
<td>(W)</td>
<td>(B)</td>
<td>(W)</td>
<td>(W)</td>
<td>(B)</td>
<td>(W)</td>
<td>(W)</td>
<td>(W)</td>
<td>(W)</td>
</tr>
<tr>
<td>256</td>
<td>16</td>
<td>2</td>
<td>2</td>
<td>1,024</td>
<td>8,848</td>
<td>All discs use this value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>16</td>
<td>2</td>
<td>2</td>
<td>1,024</td>
<td>37,820</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>16</td>
<td>2</td>
<td>2</td>
<td>1,024</td>
<td>56,730</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>8</td>
<td>1</td>
<td>2</td>
<td>68</td>
<td>2,002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Also referred to as an allocation unit.

** Size specified in the DEVCONFG utility.
**Boot Sector**

The boot sector holds an index to other sectors which contain the operating system code (MS-DOS). During system initialization, the firmware reads in the code from the sectors indexed by the boot sector. If the disc is a non-operating system disc, this sector contains FF's.

The format of the boot sector of a disc containing an operating system is as follows.

```
+--------------------------------------------------+
| Number of sectors to Read                          |
| (WORD)                                             |
+--------------------------------------------------+
<p>| Absolute Disc Sector Number to Load From           |</p>
<table>
<thead>
<tr>
<th>(WORD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Address to Load Into (Offset)</td>
</tr>
<tr>
<td>(WORD)</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>Memory Address to Load Into (Segment)</td>
</tr>
<tr>
<td>(WORD)</td>
</tr>
</tbody>
</table>
+--------------------------------------------------|

(contiguous)

+--------------------------------------------------+
| Number of Sectors to Read                          |
| (WORD)                                             |
+--------------------------------------------------+
| Absolute Disc Sector Number to Load From           |
| (WORD)                                             |
+--------------------------------------------------+
| Memory Address to Load Into (Offset)               |
| (WORD)                                             |
+--------------------------------------------------+
| Memory Address to Load Into (Segment)              |
| (WORD)                                             |
+--------------------------------------------------|

(contiguous)

+--------------------------------------------------+
| FFFFH (Extent Terminator)                        |
| (WORD)                                             |
+--------------------------------------------------+
| Execution Starting Address (Offset)               |
| (WORD)                                             |
+--------------------------------------------------+
| Execution Starting Address (Segment)              |
| (WORD)                                             |
+--------------------------------------------------+
```

**Note:** The second through n’th extents may not exist.
File Allocation Table (FAT)

DISC CLUSTERS
Disc sectors are grouped together into what are known as "clusters". On a 3 1/2" flexible disc, there are four sectors per cluster. Files are assigned space on the disc in increments of 1 cluster. That is for a 3 1/2" disc files are built in one kilobyte pieces.

The File Allocation Table is an area partitioned into many one and one half byte (12 bit) entries, each representative of a particular cluster. A file's directory entry contains a starting cluster number. The FAT entry corresponding to this cluster will point to another cluster, being the second cluster of the file. The FAT entry corresponding to this second cluster will point to a third cluster and so on. That is, the FAT entries describe a linked list of clusters containing the file.

FAT STRUCTURE The File Allocation Table always begins on the first section after the reserved sectors. If the FAT is larger than one sector, the sectors are contiguous. Two copies of the FAT are usually written for data integrity. The FAT is read into one of the MS-DOS buffers whenever needed (open, read, write, etc.). The File Allocation Table is an array of 12-bit entries (1.5 bytes) for each cluster on the disk. The first two FAT entries map a portion of the directory; these FAT entries indicate the size and format of the disk.

The second and third bytes currently always contain FFH.

The third FAT entry, which starts at byte offset 4, begins the mapping of the data area (cluster 002). Files in the data area are not always written sequentially on the disk. The data area is allocated one cluster at a time, skipping over clusters already allocated. The first free cluster found will be the next cluster allocated, regardless of its physical location on the disk. This permits the most efficient utilization of disk space because clusters made available by erasing files can be allocated for new files.

Each FAT entry contains three hexadecimal characters:

000 If the cluster is unused and available.

FF7 The cluster has a bad sector in it. MS-DOS will not allocate such a cluster. CHKDSK counts the number of bad clusters for its report. These bad clusters are not part of any allocation chain.

FF8-FFF Indicates the last cluster of a file.

XXX Any other characters that are the cluster number of the next cluster in the file. The cluster number of the first cluster in the file is kept in the file's directory entry.
HOW TO USE THE FILE ALLOCATION TABLE

The following information is included for system programmers who wish to write installable block device drivers. This section explains how MS-DOS uses the File Allocation Table to convert the clusters of a file to logical sector numbers. The driver is then responsible for locating the logical sector on disk. Programs must use the MS-DOS file management function calls for accessing files; programs that access the FAT are not guaranteed to be upwardly-compatible with future releases of MS-DOS.

Use the directory entry to find the starting cluster of the file. Next, to locate each subsequent cluster of the file:

1. Multiple the cluster number just used by 1.5 (each FAT entry is 1.5 bytes long).

2. The whole part of the product is an offset into the FAT pointing to the entry that maps the cluster just used. That entry contains the cluster number of the next cluster of the file.

3. Use a MOV instruction to move the word at the calculated FAT offset into a register.

4. If the last cluster used was an even number, keep the low-order 12 bits of the register by ANDing it with FFF; otherwise, keep the high-order 12 bits by shifting the register right 4 bits with a SHR instruction.

5. If the resultant 12 bits are FF8H-FFFH, the file contains no more clusters. Otherwise, the 12 bits contain the cluster number of the next cluster in the file.

To convert the cluster to a logical sector number (relative sector, such as that used by Interrupts 25H and 26H and by DEBUG):

1. Subtract 2 from the cluster number.

2. Multiply the result by the number of sectors per cluster.

3. Add to this result the logical sector number of the beginning of the data area.

MS-DOS Disc Directory

The disc directory is a list of all files residing on the disc. The FORMAT program builds the root directory for all disks. Its location on disk and the maximum number of entries are dependent on the media.

Since directories other than the root directory are regarded as file by MS-DOS, there is no limit to the number of files they may contain.

All directory entries are 32 bytes in length, and are in the following format (note that byte offsets are in hexadecimal):
Filename. Eight characters, left aligned and padded, if necessary, with blanks. The first byte of this field indicates the file status as follows:

00H The directory entry has never been used. This is used to limit the length of directory searches, for performance reasons.

2EH The entry is for a directory. If the second byte is also 2EH, then the cluster field contains the cluster number of this directory's parent directory (0000H if the parent directory is the root directory). Otherwise, bytes 01H through 0AH are all spaces, and the cluster field contains the cluster number of this directory.

E5H The file was used, but it has been erased.

Any other character is the first character of a filename.

Filename extension

0B File attribute. The attribute byte is mapped as follows (values are in hexadecimal):

01 File is marked read-only. An attempt to open the file for writing using the Open File system call (Function Request 3DH) results in an error code being returned. This value can be used along with other values below. Attempts to delete the file with the Delete File system call (13H) or Delete a Directory Entry (41H) will also fail.

02 Hidden file. The file is excluded from normal directory searches.

04 System file. The file is excluded from normal directory searches.

08 The entry contains the volume label in the first 11 bytes. The entry contains no other usable information (except date and time of creation), and may exist only in the root directory.

10 The entry defines a sub-directory, and is excluded from normal directory searches.

20 Archive bit. The bit is set to "on" whenever the file has been written to and closed.

Note: The system files (IO.SYS and MSDOS.SYS) are marked as read only, hidden, and system
files. Files can be marked hidden when they are created. Also, the read-only, hidden, system, and archive attributes may be changed through the Change Attributes system call (Function Request 43H).

0C-15  Reserved.

16-17  Time the file was created or last updated. The hour, minutes, and seconds are mapped into two bytes as follows:

<table>
<thead>
<tr>
<th>offset 15H</th>
<th>offset 14H</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td></td>
</tr>
</tbody>
</table>

where:

H is the binary number of hours (0-23)
M is the binary number of minutes (0-59)
S is the binary number of two second increments

18-19  Date the file was created or last updated. The year, month, and day are mapped into two bytes as follows:

<table>
<thead>
<tr>
<th>offset 17H</th>
<th>offset 16H</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td></td>
</tr>
</tbody>
</table>

where:

Y is 0-119 (1980-2099)
M is 1-12
D is 1-31

1A-1B  Starting cluster; the cluster number of the first cluster in the file.

Note that the first cluster for data space on all disks is cluster 002.

The cluster Number is stored with the least significant byte first.
NOTE

Refer to section "How to Use the File Allocation Table," for details about converting cluster numbers to logical sector numbers.

1C-1F  File size in bytes. The first word of this four-byte field is the low order part of the size.
The HP 150 system contains a large amount of system management code located in Read Only Memory. In general, the firmware is not designed to be utilized by system programmers. The operating system BIOS serves as the interface to firmware located functionality.

The HP 150 firmware is a rather complex multi-tasking operating system in itself. MS-DOS runs on the firmware as a single task from the firmwares perspective. Essentially all of the terminal functionality of the HP 150 is implemented in the firmware. The firmware implements much of the BIOS device driver functionality in addition to the terminal personality.

This section details the firmware’s usage of the 26 K Byte of system RAM which it claims, and includes a table of the firmware entry points, vectors located within a portion of that memory.
<table>
<thead>
<tr>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Firmware Memory Map</strong></td>
</tr>
<tr>
<td><strong>RAM</strong></td>
</tr>
<tr>
<td><strong>Firmware Entry Point Jump Vectors</strong></td>
</tr>
</tbody>
</table>
FIRMWARE MEMORY MAP

RAM

Portion of the system RAM memory is reserved for the use of the firmware. The following shows how that address space is allocated.

Start of MS-DOS BIOS
06800H
|---------------------------------------------|
| System Messages

01560H
|---------------------------------------------|
| Firmware Data Area
| (RESERVED - DO NOT MODIFY)

00580H
|---------------------------------------------|
| Firmware Entry Point Jump Vectors

00400H
|---------------------------------------------|
| Top of Interrupt Vector Table

FIRMWARE ENTRY POINT JUMP VECTORS

The firmware is highly structured, consisting of many tasks, drivers, and software processors. Long calls may be made to these entities through vectors located in an area of firmware RAM. In general, each entry point detailed in the table below has a defined set of functions and parameter passing conventions for those functions. Programs should not call these functions, except when implementing operations as described in section 7 of this manual, Programming the HP 150. Access to the firmware entry points (located in ROM) should at all times be made through the following jump vectors.
System Firmware

<table>
<thead>
<tr>
<th>Address</th>
<th>Vector to:</th>
<th>Length (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00400H</td>
<td>System timer for MS-DOS and applications</td>
<td>4</td>
</tr>
<tr>
<td>00425H</td>
<td>DHP ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>0042AH</td>
<td>DHP DRIVER</td>
<td>5</td>
</tr>
<tr>
<td>0042FH</td>
<td>MMP ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>00434H</td>
<td>STP ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>00439H</td>
<td>IWP ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>0043EH</td>
<td>PHP ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>00443H</td>
<td>PAP ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>00448H</td>
<td>PAPPARSE</td>
<td>5</td>
</tr>
<tr>
<td>0044DH</td>
<td>PAP_PERFORM</td>
<td>5</td>
</tr>
<tr>
<td>00452H</td>
<td>DTP ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>00457H</td>
<td>VCP_ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>0045CH</td>
<td>CNP_ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>00461H</td>
<td>DMP_ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>00466H</td>
<td>WMP_ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>00468H</td>
<td>KBF_ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>00470H</td>
<td>DCP_ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>00475H</td>
<td>RSP_ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>0047AH</td>
<td>UTF_ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>0047FH</td>
<td>NVRP_ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>00484H</td>
<td>GSP_ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>00489H</td>
<td>BMF_ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>0048EH</td>
<td>STGP_ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>00493H</td>
<td>HPSP_ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>00498H</td>
<td>SKP_ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>0049DH</td>
<td>SWF_ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>004A2H</td>
<td>SLP_ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>004A7H</td>
<td>GRP_ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>004ACH</td>
<td>GRP_TKPARSE</td>
<td>5</td>
</tr>
<tr>
<td>004B1H</td>
<td>TSP_ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>004B6H</td>
<td>TPM_SKP_ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>004BBH</td>
<td>IHF_ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>004COH</td>
<td>OHP_ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>004C5H</td>
<td>OHP_OPTION</td>
<td>5</td>
</tr>
<tr>
<td>004CAH</td>
<td>VAU_EXTERNAL_DRIVER</td>
<td>5</td>
</tr>
<tr>
<td>004CFH</td>
<td>WMP_ENTRY</td>
<td>5</td>
</tr>
<tr>
<td>004D4H</td>
<td>KBT_ENTRY1</td>
<td>5</td>
</tr>
<tr>
<td>004D9H</td>
<td>KBT_ENTRY2</td>
<td>5</td>
</tr>
</tbody>
</table>
This section is intended as an aid to programmers wishing to write applications programs to run on the HP 150. The HP 150 includes many system level functions intended to ease the complexity of programs running on the machine. Some of the examples included in this section demonstrate the use of these facilities.

In some cases the functions provided, particularly in the area of video interfacing, may not provide acceptable performance levels for your application. For this reason the internal video structure of the HP 150 is described. The HP 150's built-in capabilities in the area of data communications are extensive. Information is included in this section to allow a programmer to take advantage of some of these capabilities. Finally, the HPIB interface on the HP 150 is described from an applications program standpoint. The discussion and corresponding examples show how to drive this interface and as such extend the capabilities of the HP 150 even further.
CONTENTS

Generating Escape Sequences 7-1
Performing MS-DOS System Function Calls 7-2
Making AGIOS Function Calls 7-5
Alphanumeric Display Interfacing 7-8
  Alphanumeric Video RAM Structure 7-8
  Alpha Video Buffer Format 7-8
  Video Row Pointer Table Format 7-9
  Row Pointer Formulation from Above Table 7-9
  Line Buffer Format 7-10
  Enhancement/Character Set Byte Structure 7-10
  Character Code Structure 7-10
  Finding the Row Pointer Table Origin 7-11
  Fetching Row Pointers from the Table 7-12
  Placing a Character in Alpha Memory 7-13
  Writing to Lines 25, 26, and 27 7-15
Graphics Display Interfacing 7-18
Keyboard Interfacing 7-22
  Keycode Mode 7-22
  Console RAW/COOKED Mode 7-23
  Key Characteristics 7-24
  Sample Keyboard Driver 7-25
  Flushing the Keyboard Buffer 7-26
  Keycode Tables 7-29
Programming Data Communications 7-34
  Data Comm Using COM1 and COM2 Logical Devices 7-34
  Assigning COM1 and COM2 to Physical Ports 7-34
  Opening the COM Devices 7-35
  Input From the COM Devices 7-35
  Output to the COM Devices 7-36
  Closing the COM Devices 7-37
  COM Device I/O Example - A Terminal Emulator 7-37
Programmatic Configuration of Data Comm 7-39
  Reading the Current Configuration 7-39
  Changing the Data Comm Configuration 7-43
  Restoring the Original Configuration 7-47
Data Comm Control Functions 7-48
  IOCTL reads for Input 7-48
  Special COM Functions through IOCTL Write Request 7-48
  Fast Buffer Send for Output 7-50
HPIB Interfacing 7-52
  Limited HPIB Driver Functionality 7-52
  Opening the HPIBDEV Device 7-52
  HPIB Control Calls 7-53
  MS-DOS IOCTL Function Call 7-53
  Control Block Format 7-53
  Control Template Format 7-54
  Sample Identify Templates 7-56
  Sample Read/Write Buffer Templates 7-57
  HPIB Interface Example (9111A Graphics Tablet) 7-58
Accessory Card Interfacing
Memory (Slot) Address Identification
GENERATING ESCAPE SEQUENCES

From the Microsoft C Language for example, a formatted print function (from the Microsoft C Library) may be used to send an escape sequence to the console escape sequence processor. Note that in the following example the string is terminated by cariage return and linefeed characters to cause the console output buffer to be "flushed", that is have the information actually processed.

```c
#define ESC 0x1B

main()
{
    printf ( "%c\a12c2\r\n", ESC );
}
```

The above example moves the cursor to row 2, column 12 in the currently displayed screen window.

[CAUTION]

Because AGIOS calls are processed immediatly the call is issued and escape sequences are not processed until the console buffer is flushed, mixing the two without flushing the console buffer prior to making the AGIOS call may result in the AGIOS call being processed first.
PERFORMING MS-DOS SYSTEM FUNCTION CALLS

From the Microsoft C Language for example, MS-DOS System Function calls are most easily made through an assembly language routine combined with the C object at link time. The following is an example of a general purpose System Function call written in assembly language. It is actually an assembly language coding of a Microsoft C compatible function. The function, when called with the appropriate parameters, can invoke almost any MS-DOS System Function. Parameters are passed on the stack. Note that since parameter passing conventions between routines vary from language to language, this function is particular to Microsoft C.

```
; doscall ( ax, bx, cx, dx )
    int ax, bx, cx, dx;

; Issues an MS-DOS System Function Call with registers passed in using
; Microsoft C parameter passing convention and registers passed out in
; global four word variable.

; Parameters in:
;    ax - Value to put in register AX
;    bx - Value to put in register BX
;    cx - Value to put in register CX
;    dx - Value to put in register DX

; Return value:
;    1 (boolean TRUE) - Carry flag was set by function (error for some f'ns)
;    0 (boolean FALSE) - Carry flag not set by function (no error)

; Parameters out:
;    Registers AX, BX, CX, and DX are returned as four contiguous words
; in global variable 'dc_ret' (which is defined by this routine)
;    Word 1 - AX register returned by function
;    Word 2 - BX register returned by function
;    Word 3 - CX register returned by function
;    Word 4 - DX register returned by function

TRUE EQU 1 ;Boolean true for return value
FALSE EQU 0 ;Boolean false for return value
do_args STRUC ;Stack structure for passed parameters
    DW ? ;Callers BP
    DW ? ;Return address
ax_in DW ? ;'ax'
bx_in DW ? ;'bx'
cx_in DW ? ;'cx'
dx_in DW ? ;'dx'
do_args ENDS
```
Programming the HP 150

ret_args   STRUC
ax_out     DW  
bx_out     DW  
cc_out     DW  
ret_args   ENDS

DGROUP     GROUP DATA
DATA       SEGMENT WORD PUBLIC 'DATA'
ASSUME DS:GROUP

PUBLIC dc_ret
DC_ret    ret_args <0,0,0,0> ;Global variable 'dc_ret' declaration
DATA       ENDS

PGROUP     GROUP PROG
PROG       SEGMENT WORD PUBLIC 'PROG'
ASSUME CS:PGROUP, DS:DGROUP

PUBLIC doscall
PROC       NEAR
PUSH BP    ;Save BP and set it up as a pointer into
MOV BP,SP  ;the stack to retrieve passed arguments
MOV AX,[BP].ax_in ;Copy AX parameter from stack
MOV BX,[BP].bx_in ;Copy BX parameter from stack
MOV CX,[BP].cx_in ;Copy CX parameter from stack
MOV DX,[BP].dx_in ;Copy DX parameter from stack
INT 21H    ;Invoke MS-DOS System Function
MOV DGROUP:dc_ret.ax_out,AX   ;AX word for return
MOV DGROUP:dc_ret.bx_out,BX   ;BX word for return
MOV DGROUP:dc_ret.cx_out,CX   ;CX word for return
MOV DGROUP:dc_ret.dx_out,DX   ;DX word for return
MOV AX,TRUE ;Assume no error
JNC dcall_end
MOV AX,FALSE ;Assume no error
JNC dcall_end

dcall_end  LABEL NEAR
POP BP      ;Restore BP
RET

doscall    ENDP
PROG       ENDS
END

As an example of a C program performing an MS-DOS Function Call consider the following:
Programming the HP 150

#define DUMMY 0

struct dc_ret_tem { /* 'doscall' return parameter structure */
    char al;
    char ah;
    char bl;
    char bh;
    char cl;
    char ch;
    char dl;
    char dh;
};

extern struct dc_ret_tem dc_ret;

main () /* Prints the drive letter for the current default disc */
{
    char drive;

doscall ( 0x1900, DUMMY, DUMMY, DUMMY ); /* Current Disc System P'n */
if ( dc_ret.al < 12 )
    drive = dc_ret.al + 'A';
else
    drive = '?';
printf( "Current Disc is %c\n", drive );
}

This program invokes MS-DOS System Function Call 19H (Current Disc) which returns a number in AL indicating the current default disc drive. The appropriate drive letter is printed.
MAKING AGIOS FUNCTION CALLS

From the Microsoft C Language for example, AGIOS Function calls are most easily made through an assembly language routine combined with the C object at link time. AGIOS calls, being a special case of the doscall function previously described, could be performed using that assembly language routine. However for the sake of efficiency another routine is described. It is actually an assembly language coding of a Microsoft C compatible function. The function, when called with the appropriate parameters, can be used to invoke any AGIOS function. Parameters are passed on the stack. Note that since parameter passing conventions between routines vary from language to language, this function is particular to Microsoft C. Refer to Section 8, AGIOS Function Call Reference, for a directory of AGIOS functions.

```assembly
; int agios ( buf_add, length )
; int buf_add, length
;
; Performs an AGIOS function call with parameters passed using Microsoft
; C language parameter passing conventions. Calls MSDOS function number 044H
; (I/O Control For Devices) with sub-function 003H (Write to Device Control
; Channel) using CON (console) handle.
;
; Parameters in:
;   buf_add - The address (offset - assumes same data segment) of the AGIOS
;   parameter buffer for the call. Specifies the particular
;   AGIOS function to be performed and other pertinent data for
;   the function.
;   length - Specifies the length of the above parameter buffer in bytes.
;
; Return value:
;   0 - AGIOS function executed with no error.
;   Non-0 - AGIOS function executed with error. Value returned is the
;   status code returned from the function specified in the
;   parameter buffer (see AGIOS function documentation).
;
; Parameters out: none

PGROUP GROUP PROG
PROG SEGMENT BYTE PUBLIC 'PROG'
ASSUME CS:PGROUP

CONIN EQU 0 ;Standard MS-DOS console handle

agios_args STRUC
   DW  ?
   DW  ?
buf_add  DW  ?
buf_len  DW  ?
agios_args ENDS
```

;Stack structure for passed args.
;Callers BP
;Return address
;Parameter buffer address offset
;Number of bytes in parm buffer
Since the above assembly language function can be used to perform any AGIOS function, you may wish to put a shell around it for any particular AGIOS functions called. This will be done for Function 17, Position Cursor in the following example.

First, declare the parameter buffer structure and data area:

```
struct agios_pc_tem {
    unsigned int funct_num;  /* function number */
    char mode;               /* mode */
    unsigned int column;    /* column number */
    unsigned int row;       /* row number */
};
```

The above serves as a template for the parameter buffer particular to the Position Cursor function.

Now two constants, the parameter buffer length and the Position Cursor function number are defined:

```
#define PC_FLEN  sizeof(struct agios_pc_tem)  /* Function length */
#define PC_FNUM  17                               /* Function number */
```

The following statement actually reserves a data storage area for the parameter buffer.

```
struct agios_pc_tem agios_pc_par = { PC_FNUM, 0, 0, 0, };
```

The variable 'ret code' is used for the integer value returned by an AGIOS function call. The value indicates success or failure of the function call.
int ret_code;            /* Function call return value */

Now the cursor positioning function is defined:

cursor ( row, column )

unsigned int row;
unsigned int column;

{
    agios_pc_par.row = row;
    agios_pc_par.column = column;
    ret_code = agios ( &agios_pc_par, PC_FLEN ); /* perform the AGIOS call */
}

Note that the mode parameter has not been defined. This would be done in most cases only once outside the above routine. The mode parameter is defined and the above function called with the following statements:

    agios_pc_par.mode = 0x99;  /* row and column parameters are both valid positive absolute window values */
    cursor (2, 12);
ALPHANUMERIC DISPLAY INTERFACING

Alphanumeric Video RAM Structure

The following information describes the internal architecture of the HP 150 alphanumeric video structure. It provides programmers with a means of bypassing the console driver (CON: device) and its AGIOS function subset for the purpose of putting alphanumeric information to the screen.

The AGIOS function set contains some powerful screen manipulation capabilities and is the preferred method of accessing the alpha (and graphics) video memory. In situations where these functions provide adequate functionality and performance they should be used.

The HP 150 has 48 lines of display normally available to applications through console outputs and the AGIOS function calls. The following information applies only to the 27 lines (24 text plus two softkey plus one status) forming the physical screen as it appears at any given time.

Each of the 27 lines of alphanumeric text displayed on the screen are represented by 80 consecutive words (pairs of bytes) in RAM. A table (video row pointer table) indexes the address of the first byte in each line. This table is read by the HP 150 video control hardware directly, and the video control chip DMAs into the appropriate area of memory accordingly. Note that consecutively displayed lines on the screen do not necessarily have their storage areas in a corresponding order in RAM.

ALPHA VIDEO BUFFER FORMAT

```
       +-------------------------------------------------------+
       | 160 character buffer, line "a"                        |
       +-------------------------------------------------------+
       | 160 character buffer, line "b"                        |
       +-------------------------------------------------------+
       | 160 character buffer, line "c"                        |
       +-------------------------------------------------------+
       |                                                        |
       +-------------------------------------------------------+
       | 160 character buffer, line "x"                        |
       +-------------------------------------------------------+
       | 160 character buffer, line "y"                        |
       +-------------------------------------------------------+
```

Note: the ordering of line buffers here in RAM is not indicative of their position on the screen.
VIDEO ROW POINTER TABLE FORMAT

The row pointer table also located in alphanumeric video RAM contains 27 consecutive pointers indexing the first character of each line in the alpha line buffer. That is each pointer indexes the first byte in the above buffer. Note that the pointers are arranged line 1 through line 27 as described in the following diagram.

Byte # 1  | low byte of pointer |
2        | ----- not used ----- | forms row pointer for line 1
3        | high byte of pointer | (top of screen)
4        | ----- not used ----- |

Byte # 5  | low byte of pointer |
6        | ----- not used ----- | forms row pointer for line 2
7        | high byte of pointer | forms row pointer for line 1
8        | ----- not used ----- |

| +=================================================+ |
| \forms row pointer for line 1 | \forms row pointer for line 2 | \forms row pointer for line 27 |
| +=================================================+ |

| +=================================================+ |
| \represents | 14 bit word offset from alpha RAM origin (D0000H) |
| +=================================================+ |

**ROW POINTER FORMULATION FROM ABOVE TABLE**

<table>
<thead>
<tr>
<th>high byte of pointer</th>
<th>low byte of pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit number</td>
<td>represents</td>
</tr>
<tr>
<td>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td>14 bit word offset from alpha RAM origin (D0000H)</td>
</tr>
</tbody>
</table>

* The least significant bit (A0) must be a zero (0).

*** These bits define the double height and width characteristics of the referenced row. Note that double high/double wide characters are not supported by the HP 150 firmware, software or hardware.

<table>
<thead>
<tr>
<th>D15</th>
<th>D14</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
LINE BUFFER FORMAT

The diagram below describes the format of each display line's RAM buffer.

---

<table>
<thead>
<tr>
<th>first column position in line</th>
<th>last column position in line</th>
</tr>
</thead>
<tbody>
<tr>
<td>character</td>
<td>attribute</td>
</tr>
<tr>
<td>code</td>
<td>/char set</td>
</tr>
</tbody>
</table>

---

ENHANCEMENT/CHARACTER SET BYTE STRUCTURE

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>secur-</td>
<td>half</td>
<td>under-</td>
<td>inver-</td>
<td>blink</td>
<td>character set</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ity</td>
<td>bright</td>
<td>line</td>
<td>se</td>
<td>code</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| NORMAL ROMAN | 0 | ... | 0 | ... | 0 |
| ROMAN EXTENSION | 0 | ... | 0 | ... | 1 |
| LINE DRAWING | 0 | ... | 1 | ... | 0 |
| MATH | 0 | ... | 1 | ... | 1 |
| BOLD NORMAL ROMAN | 1 | ... | 0 | ... | 0 |
| BOLD ROMAN EXTENSION | 1 | ... | 0 | ... | 1 |
| ITALICISED NORMAL ROMAN | 1 | ... | 1 | ... | 0 |
| ITALICISED NORMAL ROMAN | 1 | ... | 1 | ... | 1 |

CHARACTER CODE STRUCTURE

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>end of line *</td>
<td>7 bit character code **</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* The end of line bit, when set, will cause the remainder of the line to be blanked by the HP 150 video hardware.

** The character sets may be seen by performing "SYSTEM TEST" from the HP 150 Service Keys softkey level.
Finding the Row Pointer Table Origin

In order to find the base address of the row pointer table a call must be made to the firmware "memory manager". The firmware manages the alphanumeric video memory and different versions of the firmware may place the pointer table in different places in RAM. For this reason it is necessary to call the memory manager at run time.

The following assembly language routine will return a pointer which is an offset from the alpha RAM segment at 0D0000H. It is compatible with Microsoft C language function calling conventions.

```assembly
ALPHA_SEG EQU 0D000H ;Alpha video RAM segment

DGROUP GROUP DATA
DATA SEGMENT WORD PUBLIC 'DATA'
 ASSUME DS:DGROUP
DATA ENDS

PGROUP GROUP PROG
row_jumps SEGMENT AT 0000H
 ORG 042FH
mem_mgr LABEL FAR
row_jumps ENDS

PROG SEGMENT BYTE PUBLIC 'PROG'
 ASSUME CS:PGROUP,DS:DGROUP,ES:NOTHING

; unsigned int vid_taddr()
;
; /* Returns start address offset of video pointer table. Segment is
;    'alpha_seg'. video memory is addressed through a table of pointers
; to absolute display locations. This function calls the firmware
; memory manager and returns the table origin */
;
Programming the HP 150

PUBLIC vid_tadd

vid_tadd PROC NEAR

FUSH BP ;Save BP
FUSH DS ;Save ES
FUSH ES ;Save ES
MOV DX,7 ;Call the firmware memory manager with
FUSH DX ; function 7: return segment of row
CALL mem mgr ; pointer table in BX.
SUB BX,ALPHA_SEG ;Convert to ALPHA_SEG relative address
MOV CL,4 ;Multiply by 16
SHL BX,CL
ADD BX,12H ;Fixed offset to row table
MOV AX,BX
POP ES ;Restore ES
POP DS ;Restore DS
POP BP ;Restore BP
RET

vid_tadd ENDP

PROG ENDS

END

Fetching Row Pointers from the Table

Given the row pointer table origin, the following Microsoft C compatible assembly language function will return the pointer for any alphanumeric video row, zero through 26. Note that the pointer returned from this routine is a word pointer and that it must be multiplied by two to get the correct byte offset for the row origin. This is done in the qik_char function which follows. The function uses an external variable row_tab which is assumed to contain the result of the vid_tad function above. The following C language statement would make that assignment:

```c
int row_tab

row_tab = vid_tadd();
```

Note that the variable row_tab is declared external in the following assembly language code and defined in the C module above.

```assembly
EXTERN row_tab:WORD ;Base address of alpha video row pointer
                   ; table. Must be initialized from
                   ; 'vid_tadd' function prior to use.
```

```assembly
; unsigned int row_ptr ( row_num )
;
; /* Returns start offset address of video row number 'row_num'.
; Segment is 'alpha_seg'. Pointer comes from table whose base
; is 'row_tab' is returned by function 'vid_tadd'. */
```
Programming the HP 150

```
rptr_args  STRUC
   DW    ? ; Stack structure for passed arguments
   DW    ? ; Callers BP
row_num    DW    ? ; 'row_num' argument
rptr_args  ENDS

row_ptr    PROC  NEAR
   PUSH  BP
   MOV   BP,SP
   MOV   BX,[BP].row_num ; 'row_num' in BX
   SHL   BX,1 ; 4 bytes
   SHL   BX,1 ; per entry
   MOV   AX,DGROUP:row_tab
   ADD   BX,AX ; Now have offset address of video row ptr
   PUSH  DS
   MOV   DX,ALPHA_SEG
   MOV   DS,DX
   MOV   AL,[BX] ; Low order row pointer
   MOV   AH,[BX+2] ; High order row pointer
   POP   DS
   POP   BP
   RET
row_ptr    ENDP
```

**Placing a Character in Alpha Memory**

The Microsoft C compatible function `qik_char` described below is an example of a routine which uses the row pointer returned by the above function to place a character onto the display in any row and column position.

```
; qik_char ( row, col, dispword )
;
/* Writes dispword (character code and attribute character set code into alpha video memory at display row number 'row' and column number 'col'.
Uses row origin address returned by the 'row_ptr' function.

Arguments in:
  'row' - screen position row number (0 <= row <= 26)
  'col' - screen position column number (0 <= col <= 79)
  'disp_word' - high order byte is attribute/character set byte
                and low order byte is character code

Return values: none

Registers preserved: BP, SP, SI

```

7-13
It is necessary to initialize the video memory prior to using the above function for random character reads and writes. The firmware sets bit 7 of the character code to blank trailing portions of lines. Any character placed into the line buffer will not be displayed unless all preceding character codes in the line have their most significant bits cleared (0). The following Microsoft C code uses the qik_char routine above to clear all those bits, writing a space with no attributes to display lines 1 through 24.
qik_init()

/* Initializes alpha RAM for random use of the 'qik_char' function. Writes spaces to all character positions on lines zero through 23. This is required to remove any end-of-line markers in the memory. Should be called once prior to using the 'qik_char' function in a random manner */

{
    int row;
    int col;

    for (row = 0; row <= 23; ++row) {
        for (col = 0; col <= 79; ++col) {
            qik_char (row, col, 0x0020);
        }
    }
}

Writing to Lines 25, 26, and 27

The HP 150 uses screen lines 25 and 26 for softkey labels, and line 27 as a status line. It periodically updates softkey labels and the cursor position counters on lines 25 and 26, and the time and status messages on the 27th line from variables it maintains in other places in memory. The previously described character to screen routine qik_char is capable of writing to any screen line, including the last three. If that routine is to be used to place text on lines 25 through 27 the firmware should be disabled from overwriting those lines with softkey and status text. The first of the two routines which follow will disable firmware modification of the last three lines of the screen. The second should be called prior to program termination to reenable that processing.

PGROUP   GROUP   PROG

firm_jumps SEGMENT AT 0000H

;hp_entry LABEL FAR ;Terminal handler process entry
ORG 0420H

php_entry LABEL FAR ;Personality handler process
ORG 043EH

firm_jumps ENDS

PROG SEGMENT BYTE PUBLIC 'PROG'
ASSUME CS:PGROUP,DS:DGROUP,ES:NOTHING
To use the status line on the HP150, one must first disable the portion of the firmware (the status line process) that would otherwise continually update the status line. After disabling the status line process, a program can write into the video memory representing the status line.

Arguments: none

Returned values: none

Registers preserved: none */

PUBLIC stat_init

stat_init PROC NEAR

MOV DX,1
PUSH DX
MOV DX,2
PUSH DX
CALL php_entry

MOV DS,DX ;Get DS register value for the ;global context area for the ;terminal personality

MOV DX,0
PUSH DX
MOV DX,0
PUSH DX
MOV DX,3
PUSH DX
MOV DX,[3AH] ;Using the DS value obtained above ;get the token for the status ;line process

PUSH DX
MOV DX,6
PUSH DX
CALL thp_entry ;Call TASK HANDLER PROCESS to ;disable the status line process

MOV DX,0
PUSH DX
CALL yield ;The yield routine is described ;elsewhere

stat_init ENDP
; stat_clnu()
; /
; /* This routine renegotiates the firmware status line process causing it
to resume updating of softkey labels, cursor position information,
time, and status messages on the last three lines of the display.
; */

Arguments in: none

Returned values: none

Registers preserved: none */

PUBLIC stat_clnu

stat_clnu PROC NEAR

MOV DX,1
PUSH DX
MOV DX,2
PUSH DX
CALL php_entry ;Get DS register value for the
MOV DS,DX ; global context area for the
; terminal personality

MOV DX,0
PUSH DX
MOV DX,1
PUSH DX
MOV DX,3
PUSH DX
MOV DX, [3AH] ;Code to enable status line process

; get the token for the status
; line process

PUSH DX
MOV DX, 6
PUSH DX
CALL thp_entry ;Call TASK HANDLER PROCESS to

MOV DX,0
PUSH DX
CALL yield ; enable the status line process

;The yield routine is described

stat_clnu ENDP

PROG ENDS

END
Programming the HP 150

GRAPHICS DISPLAY INTERFACING

The display system on the HP 150 consists of two separate "planes", a character based alphanumeric plane and a bit mapped graphics plane 512 dots wide by 390 dots high. The Alphanumeric and Graphics planes overlay each other on the display.

A comprehensive set of high level functions are provided to access the graphics plane. The AGIOS function set (see System Software section) provides functions to perform vector draws, area fills, implement graphics-plane based character sets, etcetera. This is the preferred interface for programmers wishing to access the graphics bit-map. However there are situations where, for performance or ease of software transportation reasons, the programmer may wish to access the graphics bit-map directly. The following information describes how that may be done.

Portion of the 8088 address space is allocated to a 32 kilobyte block of memory, the Graphics RAM (see Memory and I/O Mapping section). Of this 32 kilobyte block, a little over 24 kilobytes forms the graphics bit-map. The remainder is used by the firmware and should not be modified in any way by the programmer. The bit-map is organized as described in the following diagram:

```
'y'

'x' BIT POSITION

<table>
<thead>
<tr>
<th>BIT POSITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>:0</td>
</tr>
<tr>
<td>:8</td>
</tr>
<tr>
<td>:16</td>
</tr>
<tr>
<td>:24</td>
</tr>
<tr>
<td>:32</td>
</tr>
<tr>
<td>:40</td>
</tr>
<tr>
<td>:504</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ION</th>
</tr>
</thead>
<tbody>
<tr>
<td>:0</td>
</tr>
<tr>
<td>:7</td>
</tr>
<tr>
<td>:15</td>
</tr>
<tr>
<td>:23</td>
</tr>
<tr>
<td>:31</td>
</tr>
<tr>
<td>:39</td>
</tr>
<tr>
<td>:503</td>
</tr>
<tr>
<td>:511</td>
</tr>
</tbody>
</table>

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>+---------------+-------+-------+-------+-------+</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>+---------------+-------+-------+-------+-------+</td>
</tr>
<tr>
<td>.</td>
</tr>
<tr>
<td>.</td>
</tr>
<tr>
<td>.</td>
</tr>
<tr>
<td>.</td>
</tr>
<tr>
<td>389</td>
</tr>
<tr>
<td>+---------------+-------+-------+-------+-------+</td>
</tr>
</tbody>
</table>
```

A bit set ("1") will turn on the graphics dot corresponding to that bit position.

An example of an assembly language procedure which accesses the graphics bit map follows. The routine is implemented as a Microsoft C Language compatible function which sets a bit (turns the graphics dot on) given the x and y coordinates.
; bit_set (x, y)
; /* sets graphics bit map memory location for bit position 'x' on
; scan line 'y' */
; int x, y;

TRUE EQU 1 ; Boolean true for return value
FALSE EQU 0 ; Boolean false for return value
dc args STRUC ; Stack structure for passed parameters
  DW ? ; Callers BP
  DW ? ; Return address
ax in DW ? ; 'ax'
bx in DW ? ; 'bx'
cx in DW ? ; 'cx'
dx in DW ? ; 'dx'
dc args ENDS

ret args STRUC ; Structure for returned parameters
  DW ? ; Returned AX word
  DW ? ; Returned BX word
  DW 1 ; Returned CX word
  DW ? ; Returned DX word
ret args ENDS

DGROUP GROUP DATA
DATA SEGMENT WORD PUBLIC 'DATA'
ASSUME DS: DGROUP

PUBLIC dc ret
dc ret ret_args <0,0,0,0> ; Global variable 'dc_ret' declaration

DATA ENDS
As an example of a C program using the above function, consider the following. This program takes 9111A Graphics Template stylus position inputs and scales the x and y coordinates down to the HP 150 graphics bit-map matrix size. The corresponding dot in the graphics bit-map is then turned on. In this way, the graphics template stylus acts as an "electronic pencil" for the HP 150 graphics display. Note that the stylus status check inhibits bit-map modification unless the stylus is depressed.

The following main calling routine requires the following support code which is described elsewhere in this section:
- `doscall` function, see "MS-DOS System Function Calls from C Language".

- All routines (except `main`), structures, variables and constants from "HPIB Interface Example (911A Graphics Tablet)".

```c
/*****************************************************************************/
main()
{
  int x, y, status;

  if ( tab_init() == 1 ) { /* initialize graphics tablet */
    printf( "%c*d
\r", ESC ); /* clear graphics memory */
    printf( "%c*d
\r", ESC ); /* turn on graphics memory */
    while ( kbhit() == 0 ) {
      get_x_y_status(&x, &y, &status, tab_handle); /* get data, */
      /* if stylus within bounds and */
      /* is depressed, put dot on display*/
      if ( (x <= 0x2de9) && (y <= 0x216e) &&
           (status == 0x519) || (status == 0x719) ) {
        x = x/23; /* convert stylus position to */
        y = 389 - (y/22); /* display bit map position */
        bit_set(x, y); /* turn on graphics memory bit */
      }
    }
  }
  xenixclose(tab_handle); /* close the tablet */
}
```
KEYBOARD INTERFACING

The HP 150 keyboard includes many special function keys such as "Clear display". These keys are processed locally by the HP 150 firmware rather than being sent to an application requesting its information through the console input function and its associated high-level language implementations (for example BASIC "INPUT"). Applications may obtain additional control over the HP 150 keyboard by placing the HP 150 keyboard into what is termed "keycode mode". This mode allows such keys as Clear display to be passed to the application program rather than resulting in the display being cleared.

Keycode Mode

Through the use of keycode mode, applications can detect activation of almost any HP 150 keyboard key as well as the state of the SHIFT, EXTEND CHAR, and CTRL keys in conjunction with other keys. Keycode mode also allows programs to receive touchscreen input and distinguish it from keyboard input.

The following routines are examples showing how the HP 150 might be put into keycode mode. The AGIOS calls are more thoroughly described in the HP 150 Programmer’s Reference Manual. The first function uses the AGIOS Keycode ON/OFF function to read and save the keycode mode (on or off). It could be used so that the following function (km_rest) could restore the entry mode prior to program termination. The other two functions set keycode mode on and off respectively. Note that all functions are written in the C language and use the agios and data_seg functions described elsewhere in this section.

```c
#define ON 1
#define OFF 0

struct agios_ko_tem {               /* AGIOS Keycode Mode ON/OFF parameter temp*/
  unsigned int funct_num;
  char mode;
};

#define KO_FLEN sizeof(struct agios_ko_tem)
#define KO_FNUM 43              /* Keycode mode ON/OFF function */

struct agios_ks_tem {              /* AGIOS Key Status parameter template */
  unsigned int funct_num;
  unsigned int buf_offset;
  unsigned int buf_segment;
};

#define KS_FLEN sizeof(struct agios_ks_tem)
#define KS_FNUM 44              /* Read keycode mode status function */

struct agios_ko_tem agios_ko_par = (KO_FNUM, 0); /* AGIOS set keycode mode command buffer */
struct agios_ks_tem agios_ks_par; /* AGIOS read keycode mode command buffer */

unsigned int entry_kcm;
int dseg;
```
Programming the HP 150

#include <doscall.h>

#define RAW_MASK 0x20 /* MSDOS ioctrl - raw mode bit mask */
#define IOC_GET 0x4400 /* MSDOS ioctrl - get device information */
#define IOC_SET 0x4401 /* MSDOS ioctrl - set device information */
#define STDIN 0 /* standard input MSDOS file handle */

char entry_ioc;

```

km_get()
{
    dseg = data_seg();
    agios_ks_par.funct_num = KS_FNUM; /* Read and save keycode mode status */
    agios_ks_par.buf_offset = &entry_kcm;
    agios_ks_par.buf_segment = dseg;
    agios(&agios_ks_par, KS_FLEN);
}

km_rest()
{
    agios_ks_par.mode = entry_kcm; /* Restore keycode mode state */
    agios(&agios_ks_par, KO_FLEN);
}

km_on()
{
    agios_ks_par.mode = ON; /* Turn on keycode reporting mode */
    agios(&agios_ks_par, KO_FLEN);
}

km_off()
{
    agios_ks_par.mode = OFF; /* Turn off keycode reporting mode */
    agios(&agios_ks_par, KO_FLEN);
}

Console RAW/COOKED Mode

It is advisable that MS-DOS be placed in RAW mode when doing keycode mode inputs from the keyboard, otherwise MS-DOS will process the key information looking for special control sequences such as Control-C (terminate program) and Control-P (toggle printer logging). The following routines show how the RAW/COOKED mode for the console may be manipulated. They are written in the C language and use the doscall function described earlier in this section. The first two functions are intended for initialization and cleanup respectively. They can be used to save the RAW/COOKED mode as at program entry, and restore it prior to termination. See the Programmer's Reference Manual for more information on the MS-DOS I/O Control for Devices System Function Call.
```
Programming the HP 150

```c
raw_get()
{
    doscall(IOC_GET, STDIN);         /* Read and save raw/cooked mode */
    entry_ioc = (`dc_ret.dl`) & 0x00FF; /* MSB of entry_ioc must be zero
before it is used to set mode */
}
```

```c
raw_rest()
{
    doscall(IOC_SET, STDIN, 0, entry_ioc); /* Restore raw/cooked mode state */
}
```

```c
raw_on()
{
    doscall(IOC_SET, STDIN, 0, (entry_ioc | RAW_MASK)); /* Set raw mode on */
}
```

```c
raw_off()
{
    doscall(IOC_SET, STDIN, 0, (entry_ioc & ~RAW_MASK)); /* Turn off raw mode */
}
```

**Key Characteristics**

While in keycode mode, the characteristics of each key may be set individually. Specifically each key may be:

- Processed normally (for example Clear display clears the display)
- Intercepted and passed to the application for processing
- Ignored
- Any of the above accompanied by a bell beep

The following C language functions turn the interception mode on and off for all keys on the keyboard. The programmer could modify these routines to selectively define individual keys with any of the above characteristics. Note that it is not possible to read and save the key characteristics prior to changing them so that they could be restored prior to program termination. See the HP 150 Programmer's Reference Manual for more information on the AGIOS Define Key Characteristics function.

```c
struct agios_dk_tem { /* AGIOS Define Key Characteristics */
    unsigned int funct_num;
    unsigned int characteristics;
    unsigned int keycode;
};
#define DK_FLEN sizeof(struct agios_dk_tem)
#define DK_FNUM 40        /* Define Key characteristics function */
```

```c
struct agios_dk_tem agios_dk_par = {DK_FNUM, 0x0002, 0x00FE}; /* AGIOS
define key characteristics command buffer */
```
int_on()
{
    agios dk_par.characteristics = 0x0002; /* Turn on key intercept mode */
    agios(&agios dk_par, DK_FLEN);        /* for all keys */
}

int_off()
{
    agios dk_par.characteristics = 0x0000; /* Turn off key intercept mode */
    agios(&agios dk_par, DK_FLEN);
}

Sample Keyboard Driver

The following program and its support function 'key_hit' demonstrate how one might use keycode mode. The main calling routine initializes the HP 150 by saving the state of RAW/COOKED and keycode modes prior to setting them on. It then loops printing keyboard statistics for keys pressed and terminates following depression of the "Stop" key. Prior to termination, RAW/COOKED and keycode modes are restored as at entry.

#define KBD_STATUS 0x4406  /* MS-DOS ioctl - get input status */
#define READ_FC 0x3F00  /* MS-DOS read file/device function number */
#define TRUE 1
#define FALSE 0

struct key_buf_tem { /* keycode buffer template */
    char flags;
    char id;
    char keycode;
    char unused;
};

struct key_buf_tem key_buf; /* Holds a single entry read from
                                the keyboard buffer while in keycode mode */
Programming the HP 150

main()
{
    raw_get(); /* remember current console RAW/COOKED mode */
    km_get(); /* remember current keycode mode */
    raw_on(); /* turn on console RAW mode */
    int_on(); /* turn on key intercept mode for all keys */
    km_on(); /* turn on keycode mode */

    do {
        if ( key_hit() ) {
            printf("Data from keyboard: FLAGS = %x; ID = %x; KEYCODE = %x\n",
                key_buf.flags, key_buf.id, key_buf.keycode);
        }
    } while ( key_buf.keycode != 0x58 );
    km_rest(); /* restore keycode mode as at entry */
    int_off(); /* turn intercept mode off again */
    raw_rest(); /* restore raw mode as at entry */
}

key_hit()

/* Checks for data in the HP-150 keyboard type-ahead buffer. This routine checks
to see if there are any keys in the keyboard type-ahead buffer. If so the data for the first key is read from the buffer and placed in
a structure variable.

Arguments in: None.
Return value:
TRUE if a key was read from the buffer
FALSE if nothing was read from the buffer */
{
    doscall(KBD_STATUS, STDIN, 0, 0); /* If there are any keys in the */
    if ( dc_ret.al ) {
        /* keyboard type-ahead buffer */
        doscall(READ_FILE, STDIN, 4, &key_buf); /* then read the data bytes */
        return(TRUE); /* for the first key */
    }
    return(FALSE);
}

Flushing the Keyboard Buffer

The following routine flushes the keyboard type-ahead buffer by reading all the
keys from it until none are left. Note that the keyboard type-ahead buffer is a
firmware buffer, not an MS-DOS buffer. The only way to flush the firmware
buffer is to read keys from the MS-DOS device until the buffer is empty. After
reading a key from the MS-DOS device the MS-DOS task must yield to the firmware
to allow it to remove another key from its buffer and pass it to the MS-DOS
task. An assembly language routine which will yield control to the firmware
follows the flush_kbuf routine.

7-26
flush_kbuf()

/* Flush HP-150 keyboard type ahead buffer */
{
  struct key_buf_tem trash_can; /* temporary buffer to deposit keys */
  doscall ( KBD_STATUS, STDIN, 0, 0 ); /* Read everthing in the type */
  while ( dc_ret.al ) { /* ahead buffer and ignore it */
    doscall ( READ_PC, STDIN, 4, &trash_can );
    yield ( 1 ); /* Yield to allow the firmware to*/
    doscall ( KBD_STATUS, STDIN, 0, 0 ); /* Put any pending keys in the */
  } /* MS DOS key buffer */
}

YIELD_FCN EQU 009H ;Task handler yield function code

DGROUP GROUP DATA
DATA SEGMENT WORD PUBLIC 'DATA'
ASSUME DS:DGROUP
THP_ENTRY LABEL DWORD ;Task handler entry point
THP_CSET DW 00420H ;Offset
THP_SEG DW 00000H ;Segment
DATA ENDS

PGROUP GROUP PROG
PROG SEGMENT BYTE PUBLIC 'PROG'
ASSUME CS:PGROUP, DS:DGROUP, ES:NOTHING

;------------------------------------------------------------------------------
; yield ( time ) /* Yield control to firmware */
; int time; /* Time parameter to firmware */
;
; Calls the firmware task handler with a function code indicating that
; any pending firmware tasks should be allowed to process for a specified
; period of time.
;
; Arguments in:
; time minimum number of milliseconds that the firmware will have
; control up to a maximum value determined by the firmware. A
; value of zero will simply cause the MS DOS task to be placed
; on the end of the firmware ready task list.
;
; Return value: None.
;
; Registers preserved: BP, ES
;
Programming the HP 150

YIELD_ARGS STRUC
  DW ?
  DW ?
  DW ?
YIELD_TIME DW ?
YIELD_ARGS ENDS

PUBLIC YIELD

YIELD PROC NEAR
  PUSH ES
  PUSH BP
  MOV AX,[BP].YIELD_TIME
  PUSH AX
  MOV AX,YIELD_FCN
  PUSH AX
  CALL DGROUP:THP_ENTRY
  POP BP
  POP ES
  RET
YIELD ENDP

PROG ENDS

END
### Keycode Tables

<table>
<thead>
<tr>
<th>Key</th>
<th>Physical</th>
<th>Normal</th>
<th>Control</th>
<th>Shift</th>
<th>Extended Character</th>
<th>Special</th>
<th>8-bit yes</th>
<th>8-bit no</th>
<th>flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>Key addr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>057</td>
<td>054</td>
<td>054</td>
<td>FFF</td>
<td>054</td>
<td>054</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>002</td>
<td>03A</td>
<td>058</td>
<td>FFF</td>
<td>05A</td>
<td>058</td>
<td>058</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>003</td>
<td>03D</td>
<td>000</td>
<td>000</td>
<td>001</td>
<td>001</td>
<td>001</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>004</td>
<td>03E</td>
<td>001</td>
<td>002</td>
<td>002</td>
<td>002</td>
<td>002</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>005</td>
<td>03F</td>
<td>002</td>
<td>003</td>
<td>003</td>
<td>003</td>
<td>003</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>006</td>
<td>038</td>
<td>004</td>
<td>005</td>
<td>006</td>
<td>006</td>
<td>006</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>007</td>
<td>008</td>
<td>053</td>
<td>FFF</td>
<td>059</td>
<td>053</td>
<td>053</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>008</td>
<td>00F</td>
<td>051</td>
<td>FFF</td>
<td>051</td>
<td>051</td>
<td>051</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>009</td>
<td>00E</td>
<td>004</td>
<td>004</td>
<td>004</td>
<td>004</td>
<td>004</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>00D</td>
<td>005</td>
<td>005</td>
<td>005</td>
<td>005</td>
<td>005</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>00C</td>
<td>006</td>
<td>006</td>
<td>006</td>
<td>006</td>
<td>006</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>012</td>
<td>00B</td>
<td>007</td>
<td>007</td>
<td>007</td>
<td>007</td>
<td>007</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>013</td>
<td>00A</td>
<td>043</td>
<td>043</td>
<td>043</td>
<td>043</td>
<td>043</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>014</td>
<td>009</td>
<td>042</td>
<td>069</td>
<td>048</td>
<td>042</td>
<td>042</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>015</td>
<td>03C</td>
<td>060</td>
<td>000</td>
<td>07E</td>
<td>0FB</td>
<td>060</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>016</td>
<td>02C</td>
<td>031</td>
<td>FFF</td>
<td>021</td>
<td>08B</td>
<td>031</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>017</td>
<td>051</td>
<td>032</td>
<td>FFF</td>
<td>040</td>
<td>040</td>
<td>040</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>018</td>
<td>044</td>
<td>033</td>
<td>FFF</td>
<td>023</td>
<td>023</td>
<td>023</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>019</td>
<td>045</td>
<td>034</td>
<td>FFF</td>
<td>024</td>
<td>0F7</td>
<td>034</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>020</td>
<td>046</td>
<td>035</td>
<td>FFF</td>
<td>025</td>
<td>0F8</td>
<td>035</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>021</td>
<td>047</td>
<td>036</td>
<td>FFF</td>
<td>05E</td>
<td>05E</td>
<td>036</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>022</td>
<td>040</td>
<td>037</td>
<td>FFF</td>
<td>026</td>
<td>05C</td>
<td>037</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>023</td>
<td>010</td>
<td>038</td>
<td>FFF</td>
<td>02A</td>
<td>05B</td>
<td>038</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>024</td>
<td>017</td>
<td>039</td>
<td>FFF</td>
<td>028</td>
<td>05D</td>
<td>039</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>025</td>
<td>016</td>
<td>030</td>
<td>FFF</td>
<td>029</td>
<td>0B9</td>
<td>030</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>026</td>
<td>015</td>
<td>02D</td>
<td>02D</td>
<td>05F</td>
<td>0F6</td>
<td>02D</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>027</td>
<td>014</td>
<td>03D</td>
<td>03D</td>
<td>02B</td>
<td>0FE</td>
<td>03D</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>028</td>
<td>013</td>
<td>027</td>
<td>027</td>
<td>027</td>
<td>027</td>
<td>027</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>029</td>
<td>012</td>
<td>044</td>
<td>044</td>
<td>044</td>
<td>044</td>
<td>044</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>030</td>
<td>011</td>
<td>045</td>
<td>045</td>
<td>045</td>
<td>045</td>
<td>045</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>031</td>
<td>049</td>
<td>024</td>
<td>024</td>
<td>026</td>
<td>024</td>
<td>024</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>032</td>
<td>04A</td>
<td>071</td>
<td>011</td>
<td>051</td>
<td>FFF</td>
<td>071</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>033</td>
<td>04B</td>
<td>077</td>
<td>017</td>
<td>057</td>
<td>07E</td>
<td>077</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>034</td>
<td>04C</td>
<td>065</td>
<td>005</td>
<td>045</td>
<td>0D7</td>
<td>065</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>035</td>
<td>04D</td>
<td>072</td>
<td>012</td>
<td>052</td>
<td>FFF</td>
<td>072</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>036</td>
<td>04E</td>
<td>074</td>
<td>014</td>
<td>054</td>
<td>FFF</td>
<td>074</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>037</td>
<td>04F</td>
<td>079</td>
<td>019</td>
<td>059</td>
<td>FFF</td>
<td>079</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>038</td>
<td>048</td>
<td>075</td>
<td>015</td>
<td>055</td>
<td>FFF</td>
<td>075</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>039</td>
<td>018</td>
<td>069</td>
<td>009</td>
<td>049</td>
<td>FFF</td>
<td>069</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>040</td>
<td>01F</td>
<td>06F</td>
<td>00F</td>
<td>04F</td>
<td>0D6</td>
<td>06F</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7-29
Programming the HP 150

<table>
<thead>
<tr>
<th>Key</th>
<th>Physical</th>
<th>Normal</th>
<th>Control</th>
<th>Shift</th>
<th>Extended Character</th>
<th>Special</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>Key addr</td>
<td></td>
<td></td>
<td></td>
<td>8-bit yes</td>
<td>8-bit no</td>
</tr>
<tr>
<td>041</td>
<td>01E</td>
<td>070</td>
<td>010</td>
<td>050</td>
<td>0F1</td>
<td>070</td>
</tr>
<tr>
<td>042</td>
<td>01D</td>
<td>05B</td>
<td>01B</td>
<td>07B</td>
<td>0B3</td>
<td>05B</td>
</tr>
<tr>
<td>043</td>
<td>01C</td>
<td>05D</td>
<td>01D</td>
<td>07D</td>
<td>07C</td>
<td>05D</td>
</tr>
<tr>
<td>044</td>
<td>01B</td>
<td>05C</td>
<td>01C</td>
<td>07C</td>
<td>FFF</td>
<td>05C</td>
</tr>
<tr>
<td>045</td>
<td>01A</td>
<td>040</td>
<td>06B</td>
<td>046</td>
<td>040</td>
<td>040</td>
</tr>
<tr>
<td>046</td>
<td>019</td>
<td>041</td>
<td>06D</td>
<td>047</td>
<td>041</td>
<td>041</td>
</tr>
<tr>
<td>047</td>
<td>043</td>
<td>056</td>
<td>056</td>
<td>056</td>
<td>056</td>
<td>056</td>
</tr>
<tr>
<td>048</td>
<td>03B</td>
<td>FFF</td>
<td>FFF</td>
<td>FFF</td>
<td>FFF</td>
<td>FFF</td>
</tr>
<tr>
<td>049</td>
<td>053</td>
<td>061</td>
<td>001</td>
<td>041</td>
<td>0D4</td>
<td>061</td>
</tr>
<tr>
<td>050</td>
<td>054</td>
<td>073</td>
<td>013</td>
<td>053</td>
<td>0DE</td>
<td>073</td>
</tr>
<tr>
<td>051</td>
<td>055</td>
<td>064</td>
<td>004</td>
<td>044</td>
<td>0E4</td>
<td>064</td>
</tr>
<tr>
<td>052</td>
<td>056</td>
<td>066</td>
<td>006</td>
<td>046</td>
<td>0BE</td>
<td>066</td>
</tr>
<tr>
<td>053</td>
<td>039</td>
<td>067</td>
<td>007</td>
<td>047</td>
<td>0BA</td>
<td>067</td>
</tr>
<tr>
<td>054</td>
<td>050</td>
<td>068</td>
<td>008</td>
<td>048</td>
<td>0BC</td>
<td>068</td>
</tr>
<tr>
<td>055</td>
<td>020</td>
<td>06A</td>
<td>00A</td>
<td>04A</td>
<td>024</td>
<td>06A</td>
</tr>
<tr>
<td>056</td>
<td>027</td>
<td>06B</td>
<td>00B</td>
<td>04B</td>
<td>0BF</td>
<td>06B</td>
</tr>
<tr>
<td>057</td>
<td>026</td>
<td>06C</td>
<td>00C</td>
<td>04C</td>
<td>0BB</td>
<td>06C</td>
</tr>
<tr>
<td>058</td>
<td>025</td>
<td>03B</td>
<td>03B</td>
<td>03A</td>
<td>0AF</td>
<td>03B</td>
</tr>
<tr>
<td>059</td>
<td>024</td>
<td>027</td>
<td>027</td>
<td>022</td>
<td>060</td>
<td>027</td>
</tr>
<tr>
<td>060</td>
<td>023</td>
<td>025</td>
<td>025</td>
<td>025</td>
<td>025</td>
<td>025</td>
</tr>
<tr>
<td>061</td>
<td>022</td>
<td>02C</td>
<td>06E</td>
<td>02D</td>
<td>02C</td>
<td>02C</td>
</tr>
<tr>
<td>062</td>
<td>021</td>
<td>02F</td>
<td>02F</td>
<td>02F</td>
<td>02F</td>
<td>02F</td>
</tr>
<tr>
<td>063</td>
<td>071</td>
<td>055</td>
<td>055</td>
<td>057</td>
<td>055</td>
<td>055</td>
</tr>
<tr>
<td>064</td>
<td>041</td>
<td>FFF</td>
<td>FFF</td>
<td>FFF</td>
<td>FFF</td>
<td>FFF</td>
</tr>
<tr>
<td>065</td>
<td>073</td>
<td>07A</td>
<td>01A</td>
<td>05A</td>
<td>FFF</td>
<td>07A</td>
</tr>
<tr>
<td>066</td>
<td>074</td>
<td>07B</td>
<td>01B</td>
<td>05B</td>
<td>0EC</td>
<td>078</td>
</tr>
<tr>
<td>067</td>
<td>075</td>
<td>063</td>
<td>003</td>
<td>043</td>
<td>0B5</td>
<td>063</td>
</tr>
<tr>
<td>068</td>
<td>076</td>
<td>076</td>
<td>016</td>
<td>056</td>
<td>0BD</td>
<td>076</td>
</tr>
<tr>
<td>069</td>
<td>077</td>
<td>062</td>
<td>002</td>
<td>042</td>
<td>0FC</td>
<td>062</td>
</tr>
<tr>
<td>070</td>
<td>070</td>
<td>06E</td>
<td>00E</td>
<td>04E</td>
<td>0F9</td>
<td>06E</td>
</tr>
<tr>
<td>071</td>
<td>028</td>
<td>06D</td>
<td>00D</td>
<td>04D</td>
<td>0FA</td>
<td>06D</td>
</tr>
<tr>
<td>072</td>
<td>02F</td>
<td>02C</td>
<td>02C</td>
<td>03C</td>
<td>03C</td>
<td>02C</td>
</tr>
<tr>
<td>073</td>
<td>02E</td>
<td>02E</td>
<td>02E</td>
<td>03E</td>
<td>03E</td>
<td>02E</td>
</tr>
<tr>
<td>074</td>
<td>02D</td>
<td>02F</td>
<td>02F</td>
<td>03F</td>
<td>05F</td>
<td>02F</td>
</tr>
<tr>
<td>075</td>
<td>042</td>
<td>FFF</td>
<td>FFF</td>
<td>FFF</td>
<td>FFF</td>
<td>FFF</td>
</tr>
<tr>
<td>076</td>
<td>02B</td>
<td>052</td>
<td>052</td>
<td>052</td>
<td>052</td>
<td>052</td>
</tr>
<tr>
<td>077</td>
<td>02A</td>
<td>020</td>
<td>065</td>
<td>029</td>
<td>020</td>
<td>020</td>
</tr>
<tr>
<td>078</td>
<td>029</td>
<td>02E</td>
<td>06C</td>
<td>02E</td>
<td>02E</td>
<td>02E</td>
</tr>
<tr>
<td>079</td>
<td>072</td>
<td>050</td>
<td>05C</td>
<td>05B</td>
<td>050</td>
<td>050</td>
</tr>
<tr>
<td>080</td>
<td>036</td>
<td>FFF</td>
<td>FFF</td>
<td>FFF</td>
<td>FFF</td>
<td>FFF</td>
</tr>
</tbody>
</table>
### USASCII KEYCODES

<table>
<thead>
<tr>
<th>Key #</th>
<th>Key addr</th>
<th>Physical</th>
<th>Normal</th>
<th>Control</th>
<th>Shift</th>
<th>Extended Character</th>
<th>8-bit yes</th>
<th>8-bit no</th>
<th>Special flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>081</td>
<td>037</td>
<td>020</td>
<td>020</td>
<td>020</td>
<td>020</td>
<td>020</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>082</td>
<td>035</td>
<td>FFF</td>
<td>FFF</td>
<td>FFF</td>
<td>FFF</td>
<td>FFF</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>083</td>
<td>033</td>
<td>023</td>
<td>061</td>
<td>02A</td>
<td>023</td>
<td>023</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>084</td>
<td>032</td>
<td>021</td>
<td>062</td>
<td>028</td>
<td>021</td>
<td>021</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>085</td>
<td>031</td>
<td>022</td>
<td>063</td>
<td>02B</td>
<td>022</td>
<td>022</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>086</td>
<td>06F</td>
<td>008</td>
<td>008</td>
<td>008</td>
<td>008</td>
<td>008</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>087</td>
<td>06D</td>
<td>009</td>
<td>009</td>
<td>009</td>
<td>009</td>
<td>009</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>088</td>
<td>06B</td>
<td>00A</td>
<td>00A</td>
<td>00A</td>
<td>00A</td>
<td>00A</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>089</td>
<td>069</td>
<td>00B</td>
<td>00B</td>
<td>00B</td>
<td>00B</td>
<td>00B</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>090</td>
<td>067</td>
<td>071</td>
<td>071</td>
<td>071</td>
<td>071</td>
<td>071</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>091</td>
<td>065</td>
<td>073</td>
<td>073</td>
<td>073</td>
<td>073</td>
<td>073</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>092</td>
<td>063</td>
<td>072</td>
<td>072</td>
<td>072</td>
<td>072</td>
<td>072</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>093</td>
<td>061</td>
<td>070</td>
<td>06A</td>
<td>070</td>
<td>070</td>
<td>070</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>094</td>
<td>05F</td>
<td>037</td>
<td>037</td>
<td>037</td>
<td>037</td>
<td>037</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>095</td>
<td>05D</td>
<td>038</td>
<td>038</td>
<td>038</td>
<td>038</td>
<td>038</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>096</td>
<td>05B</td>
<td>039</td>
<td>039</td>
<td>039</td>
<td>039</td>
<td>039</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>097</td>
<td>059</td>
<td>075</td>
<td>075</td>
<td>075</td>
<td>075</td>
<td>075</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>098</td>
<td>05E</td>
<td>034</td>
<td>034</td>
<td>034</td>
<td>034</td>
<td>034</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>099</td>
<td>05C</td>
<td>035</td>
<td>035</td>
<td>035</td>
<td>035</td>
<td>035</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>05A</td>
<td>036</td>
<td>036</td>
<td>036</td>
<td>036</td>
<td>036</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>058</td>
<td>074</td>
<td>074</td>
<td>074</td>
<td>074</td>
<td>074</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>066</td>
<td>031</td>
<td>031</td>
<td>031</td>
<td>031</td>
<td>031</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>064</td>
<td>032</td>
<td>032</td>
<td>032</td>
<td>032</td>
<td>032</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>062</td>
<td>033</td>
<td>033</td>
<td>033</td>
<td>033</td>
<td>033</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>06E</td>
<td>030</td>
<td>030</td>
<td>030</td>
<td>030</td>
<td>030</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>06A</td>
<td>077</td>
<td>077</td>
<td>077</td>
<td>077</td>
<td>077</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>060</td>
<td>076</td>
<td>076</td>
<td>076</td>
<td>076</td>
<td>076</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTE**

1. The keyboard key numbers are specified in decimal, all other entries are specified in hexadecimal.

2. A keycode entry of 'FFF' means that no keycode is returned for that key sequence.
3. The keycodes were obtained using Rev. B of the firmware ROM's:
   1818-3321, 1818-3322, 1818-3323, 1818-3324, 1818-3325

   Keycode reporting differences in Rev. A of the firmware ROM's:
   CTRL <key 072> does not report a keycode
   CTRL <key 073> does not report a keycode
   CTRL <key 097> reports keycode 05C

4. The keycodes were recorded with the keyboard state set as follows:
   Keycode mode on.
   Console device in raw mode.
   Intercept mode set for all keys.
   Control-C checking off.
   Caps mode off (keyboard key function).
   Caps lock off (configuration menu function).
   Language USASCII (configuration menu function).
   Application softkey labels displayed.

5. Every keycode returned to an application is made up of four bytes:

   
   |=======================================|
   | SP | XX | LE | RE | CT | LS | RS | RP |
   |=======================================|

   SP - Special key
   XX - Reserved
   LE - Left extended character key down
   RE - Right extended character key down
   CT - Control key down
   LS - Left shift key down
   RS - Right shift key down
   RP - Repeating key

   Keycode source ID byte:
   0C0H - Keyboard data
   080H - Touchscreen data

6. Keycodes vary with the keyboard (language) field of the terminal configuration menu.

7. CTRL-numeric keys on main keyboard section (not numeric) report keycodes in an erratic manner. These key sequences should not be used for this reason.
Keyboard Key Position Numbers
PROGRAMMING DATA COMMUNICATIONS

The HP 150 personal computer has comprehensive data communications facilities which are accessible to the programmer. There are two data communications ports, both of which are capable of running in synchronous or asynchronous mode. Only asynchronous mode is supported by the system firmware and software. If you wish to program the HP 150 to use synchronous mode it will be necessary to write your own hardware level I/O, configuration, and interrupt drivers. For persons wishing to do such things, see the "Data Comm at the Hardware I/O Level" discussion later in this section.

The following information should make your task of interfacing to the HP 150's powerful data communications capabilities relatively simple, especially for asynchronous operation. The discussion covers logical (COM1 and COM2) to physical (Port1 and Port2) configuration, input and output of data through these devices, programmatic reconfiguration of the port asynchronous operating parameters, and port hardware interfacing. Included is an example of a simple terminal emulator application program.

Data Comm Using COM1 and COM2 Logical Devices

Input/Output to the COM1 and COM2 devices can be performed using the MS-DOS "Read from a File/Device" and "Write to a File/Device" System Function Calls. The device in question must be "opened" prior to accessing it in this manner, and should be "closed" when access is complete, usually just prior to application program termination. It may be necessary to have the user assign COM1 and/or COM2 to one of the physical serial I/O ports on the HP 150 prior to running your application.

NOTE

Serial communications I/O through the MS-DOS COM devices is somewhat slow because of operating system and firmware overhead. In practice throughput rates of around 2400 baud (half duplex) can be attained. For higher throughput it may be necessary to use the MS-DOS I/O Control for Devices System Function Call or deal directly with the hardware. See "Data Comm Control Functions" later in this section.

ASSIGNING COM1 AND COM2 TO PHYSICAL PORTS

COM1 and COM2 are recognized as device names by MS-DOS. They may be mapped (assigned) to either of the physical hardware ports, "Port1" and "Port2". There is an RS232 connector for Port1 and an RS232/422 connector for Port2 at the rear of the HP 150. Both these ports are standard on the HP 150. The mapping between COM1/COM2 and Port1/Port2 is stored in non-volatile memory on the HP 150. Currently the only trivial way to change this assignment is to run the "DEVCONFIG" utility program shipped with the HP 150 operating system. The default mapping has COM1 assigned to Port1 and COM2 assigned to Port2. This means that MS-DOS references to COM1 cause Port1 hardware to be accessed, and references to COM2 cause Port2 hardware to be accessed. The assignment of Port1
and Port2 as the "Remote/Serial" devices in the DEVCORFG utility has no meaning here and will not alter the logical to physical device mapping.

OPENING THE COM DEVICES

The MS-DOS "Open a File" System Function Call (3DH) provides a "handle" or device identifier used for input and output of data and control information to the device. The function call is made passing a pointer to a character string being the device name, COM1 or COM2 in this case. The character string must be terminated by a null byte. The following is a C language example of a call to open the COM1 device. It uses the doscall function previously described.

```c
#define OPEN RW 0x3D02 /* MS-DOS Open File/Device System Function */
#define DUMMY 0 /* dummy 'doscall' argument */

struct dc_ret_tem { /* 'doscall' return parameter structure */
  char al;
  char ah;
  char bl;
  char bh;
  char cl;
  char ch;
  char dl;
  char dh;
};

extern struct dc_ret_tem dc_ret;

char com1_name[5] = { "COM1" };
char comhandle;

{
  com1_name[4] = 0; /* terminate device name string with NULL */
  doscall ( OPEN_RW, DUMMY, DUMMY, &com1_name[0] ); /* open COM1: device */
  comhandle = dc_ret.al;
}
```

The above code associates the variable comhandle with a handle used for all future references to the COM1 device while it remains open.

INPUT FROM THE COM DEVICES

As mentioned, data input from the serial communications devices is most easily performed using the MS-DOS "Read from a File/Device" System Function Call (3FH). This function allows single or multiple character input from the device. The Read from a File/Device function will wait on a character if none exists when called. The firmware level drivers for Port1 and Port2 buffer incoming and outgoing data. If the port incoming buffer is empty when the read function is called, the function will not return with a character until a byte is received.
at the serial interface for the port. Otherwise the function will return a
character from the buffer and adjust the buffer accordingly.

The MS-DOS I/O Control for Devices System Function Call can be used to check the
Port1 and Port2 incoming buffer status. The function is called using the "get
input status" request as in the following C language example.

```
#define IOC_INSTAT 0x4406          /* MS-DOS IOCTL Function - get input status */
{
    doscall ( IOC_INSTAT, comhandle );        /* check device input stat */
    if ( dc_ret.al == 0xFF )                 /* buffer contains data */;
    else                                     /* buffer is empty */;
}
```

The above I/O Control for Devices function with the get input status request
would normally be called prior to reading a character from the device. Note
that this may not be a valid function call for devices other than COM1 and COM2.
The following is a C language example of a single character input from an open
device with the handle comhandle. It uses the doscall function previously
described. The input character is placed in variable inchar.

```
#define READ 0x3F00             /* MS-DOS Read From File/Device System Function */
char inchar;
{
    doscall ( READ, comhandle, 1, &inchar );        /* char in from device */
}
```

OUTPUT TO THE COM DEVICES

The MS-DOS "Write to a File/Device" System Function Call allows single or
multiple character output to the serial communication devices among others.
Once again, it is advisable to check status prior to making the call in case the
device outgoing buffer is full. The following example shows how this may be
done in the C language.

```
#define IOC_OUTSTAT 0x4407         /* MS-DOS IOCTL Function - get output stat */
{
    doscall ( IOC_OUTSTAT, comhandle );        /* check device output stat */
    if ( dc_ret.al == 0xFF )                 /* buffer can accept data */;
    else                                     /* buffer is full */;
```
Given that ready is returned by the check output status call above, the MS-DOS "Write to a File/Device" System Function Call is used in the following example to send the character variable outchar.

```c
#define WRITE 0x4000 /* MS-DOS Write To File/Device System Function */
char outchar;
{
    doscall (WRITE, comhandle, 1, &outchar); /* write char. to device */
}
```

CLOSING THE COM DEVICES

An application which has opened devices or files should close them prior to termination. If either of the MS-DOS COM devices are opened, as COM1 is above, they should be closed using the MS-DOS "Close a File Handle" System Function Call (3EH). The following example shows how that device may be closed in C language. comhandle and com1_name variables are as defined in the device open example as previously described.

```c
{
    doscall (CLOSE, comhandle ); /* close the COM1: device */
}
```

COM DEVICE I/O EXAMPLE - A TERMINAL EMULATOR

The following example implements a simple terminal emulator for the HP 150. It will use the COM1 device and sends all escape and control sequences as well as special keyboard keys to the firmware for processing in a manner consistent with "terminal" mode on the HP 150. Pressing ! on the keyboard terminates the application. To use this application on a HP3000 computer, the Port to which COM1 is assigned should be configured as for HP3000 Point-to-Point (system defaults).
#define TRUE 1
#define FALSE 0
#define RAW_MASK 0x20 /* MS-DOS ioctl - raw mode bit mask */
#define OPEN_RW 0x3D02 /* MS-DOS Open File/Device System Function */
#define CLOSE 0x3E00 /* MS-DOS Close File/Device System Function */
#define READ 0x3F00 /* MS-DOS Read From File/Device System Function */
#define WRITE 0x4000 /* MS-DOS Write To File/Device System Function */
#define IOC_GET 0x4400 /* MS-DOS IOCTL Function - get device information */
#define IOC_SET 0x4401 /* MS-DOS IOCTL Function - set device information */
#define IOC_INSTAT 0x4406 /* MS-DOS IOCTL Function - get input status */
#define IOC_OUTSTAT 0x4407 /* MS-DOS IOCTL Function - get input status */
#define STDOUT 1 /* standard output MS-DOS file handle (console) */
#define DUMMY 0 /* dummy 'doscall' argument */

struct dc_ret_tem { /* 'doscall' return parameter structure */
  char al;
  char ah;
  char bl;
  char bh;
  char cl;
  char ch;
  char dl;
  char dh;
};

char dcent_ioc;
char comhandle;
char outchar;
char inchar;
char terminate;
char coml_name[5] = { "COM1" };

extern struct dc_ret_tem dc_ret;

main()
{
  coml_name[4] = 0; /* terminate device name string with NULL */
  doscall(OPEN_RW, DUMMY, DUMMY, &coml_name[0]); /* open COM1: device */
  comhandle = dc_ret.al;
  doscall(IOC_GET, comhandle); /* Read and save console raw/cooked mode */
  dcnt_ioc = (dcnt_ioc | RAW_MASK); /* Set raw mode on */
  terminate = FALSE;
}
Programmatic Configuration of Data Comm

The current configuration of each data communications port may be determined by making a call to the HP 150 data communications driver firmware. The calls are made by referencing a particular physical port - that is, Port1 or Port2. Note that either of these ports are configurable as MS-DOS devices COM1 or COM2. Currently there is no trivial way of determining the mapping in a programmatic manner. This information is stored in non-volatile CMOS memory and is managed by the DEVCONFG utility. The location of the mapping data which the DEVCONFG utility manages is dynamic, that is it varies from version to version of the utility. Unlike the firmware which may be called to return configuration information which it manages, the DEVCONFG utility has no such facility. Accessing the CMOS memory directly to manipulate or read the Port1/Port2 to COM1/COM2 mapping data will introduce DEVCONFG version dependencies. This should be avoided at all costs. Stated in simple terms, if your application must reconfigure either of the serial ports using the methodology to be described, your application will require the user to have fixed Port1/Port2 to COM1/COM2 mapping.

READING THE CURRENT CONFIGURATION

The firmware call to be described will read configuration menu items into a data structure defined by the caller. The structure has a fixed format and is 7 bytes long. Three additional header bytes and one additional trailer word are defined as part of the structure. These bytes are used for firmware use during the read configuration described here and the verify and reconfigure datacomm functions to be described later.

The configuration data structure elements (ignoring the bytes used by the firmware) are defined as follows:
Programming the HP 150

<table>
<thead>
<tr>
<th>misc_1:</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1 Stop</th>
<th>CS(CB)</th>
<th>SRR</th>
<th>RR(CF)</th>
<th>SRR</th>
<th>SR(CH)</th>
<th>Chk</th>
<th>ENQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>Xmit</td>
<td>Invert</td>
<td>Recv</td>
<td>Xmit</td>
<td>Parity</td>
<td>ACK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>2 Stop</th>
<th>0</th>
<th>No</th>
<th>0</th>
<th>No</th>
<th>0</th>
<th>Lo</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 Stop</td>
<td>1 Yes</td>
<td>1</td>
<td>Yes</td>
<td>1</td>
<td>Yes</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>misc_2:</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>TR(CD)</th>
<th>DM(CC)</th>
<th></th>
<th></th>
<th></th>
<th>7 Data</th>
<th>Xmit</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Xmit</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>Bits</th>
<th>Pace</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>baud_rate:</th>
<th>02H</th>
<th>04H</th>
<th>05H</th>
<th>06H</th>
<th>07H</th>
<th>0AH</th>
<th>0CH</th>
<th>0EH</th>
<th>0FH</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>110 Baud</td>
<td>150 Baud</td>
<td>300 Baud</td>
<td>600 Baud</td>
<td>1200 Baud</td>
<td>2400 Baud</td>
<td>4800 Baud</td>
<td>9600 Baud</td>
<td>19200 Baud</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>parity:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Zeros</td>
<td>Ones</td>
<td>Even</td>
<td>Odd</td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>clock:</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>INT CLOCK</td>
<td>EXT X1 CLOCK</td>
<td>EXT X16 CLOCK</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>asterisk:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No Asterisk</td>
<td>DM Asterisk</td>
<td>RR Asterisk</td>
<td>Line Asterisk</td>
<td>CS Asterisk</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>recv_pace:</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No Recv Pace</td>
<td>Xon/Xoff Recv Pace</td>
<td>TR(CD) Recv Pace</td>
</tr>
</tbody>
</table>

7-40
The above structure elements plus some firmware variables are represented in an assembly language structure as follows:

```
config_data STHUC
predef_val DB ? ;Structure defining config data area
barrel_idx DB ? ;Used by firmware for config calls
port DB ? ;Used by firmware for config calls
misc_1 DB ? ;Miscellaneous configuration bits
misc_2 DB ? ;Miscellaneous configuration bits
baud_rate DB ? ;Baud rate
parity DB ? ;Parity and checking
clock DB ? ;Clock source
asterisk DB ? ;Control line asterisk
recv_rate DB ? ;Receive pacing
dev_token DW ? ;Used by firmware for config calls
config_data ENDS
```

A Microsoft C language compatible call is now described. The function is passed two parameters:

- The port number (1 = Port1, 2 = Port2)
- A pointer to a data structure defined as above

The function will fill in the structure provided by the caller with the configuration for the appropriate port. Note that the configuration information will reflect that as it currently exists in non-volatile memory, as is seen in the configuration menu.

```
TRUE EQU 1
FALSE EQU 0
SUCCESS EQU 0
DHP_RELEASEDEVICE EQU 3
DHP_CDEV EQU 1
DHP_CDTOK EQU 14
VERIFY_CODE EQU 14
CONFIG_CODE EQU 1
START_DRIVER_CODE EQU 13
```

```
DGROUP GROUP DATA
DATA SEGMENT WORD PUBLIC 'DATA'
ASSUME DS:DGROUP

hpibdev DB 'HPIBDEV',0 ;HPIB device name string
PUBLIC fwa_es
fwa_es DW ? ;Firmware ES saved here

DATA ENDS
```
Programming the HP 150

PGROUP GROUP PROG
ASSUME CS:firm_jumps,DS:NOTHING

firm_jumps SEGMENT BYTE AT 0000H
ORG 0425H

hp_entry LABEL FAR
ORG 042AH

hp_driver LABEL FAR
ORG 045CH

cnp_entry LABEL FAR

firm_jumps ENDS

PROG SEGMENT BYTE PUBLIC 'PROG'
ASSUME CS:PGROUP,DS:DGROUP,ES:NOTHING

;--------------------------------------------------------------------------

; char get_conf ( port_num, scratch )
;
; /* Reads current daqcomm port number 'port_num' configuration
; information from CMOS (configuration) memory into the scratch
; structure at DS:'scratch'. The configuration data area at 'scratch'
; will have all the values needed to specify the daqcomm
; configuration. These can be modified and passed to the firmware
; daqcomm driver to set a new configuration. Returns TRUE if
; successful in performing the operation, false otherwise.
;
getcf_args STRUC
DW ? ;Stack structure for passed arguments
DW ? ;Callers BP
DW ? ;Callers return address
port_num DW ? ;Port number (1 = Port1, 2 = Port2)
scratch DW ? ;'scratch' argument
getcf_args ENDS
Programming the UP 150

get_conf PROC NEAR
PUSH BP
MOV BP,SP
MOV BX, [BP]. port_num
MOV DI, [BP]. scratch ; 'scratch' argument from stack
POP BP
MOV ES, fs.es ; Initialize firmware ES register
MOV [DI]. predefined_var, 3 ; Initialize the variables of the
MOV [DI]. barrel_idx, 6 ; config data area required by
SHL BL, 1 ; config processing
CMP BL, 2 ; Port ID is 2 for port 1
JE port_one
MOV BL, 11 ; Port ID is 11 for port 2

port_one LABEL NEAR
MOV [DI]. port, BL
MOV AH, BL ; Port identification
MOV AL, 04H ; 04 for full duplex hardwired form
PUSH AX
MOV AX, 0201H ; 01 for personality ID, 02 for config ; values are required
PUSH AX
PUSH DS ; Segment of scratch data area
PUSH DI ; Offset of scratch data area
LEA AX, [DI]. port ; Pointer to start of config data area
PUSH AX
MOV AX, 3 ; Function code for CNP_GET_DEV_CONFIG
PUSH AX
CALL cnp_entry ; Returns: AX = status, BX = fail reason
CMP AX, SUCCESS ; Call error if failures
MOV AX, TRUE ; Firmware call executed without errors
JE get_cf_ok
MOV AX, FALSE ; Error during firmware call

get_cf_ok LABEL NEAR
RET

get_conf ENDP

CHANGING THE DATA COMM CONFIGURATION

The configuration data structure initialized by the above function is used when reconfiguring the port with different baud rate, parity, handshaking and other parameters. The process of reconfiguration is as follows:

1. Save a copy of this data area for later restoration of the original configuration.

2. Change the appropriate structure parameters to reflect the desired new configuration.

3. Call ver_conf to "obtain the firmware device token", "get the firmware device", and verify that the new configuration data area is valid.
4. If the data is a valid configuration for the port (\texttt{ver_conf} returns \texttt{TRUE}), call \texttt{config} to "attach the firmware datacomm driver", reconfigure the hardware, "start the firmware driver", and "release the firmware datacomm device".

The following function performs the verification process, returning \texttt{TRUE} if the configuration data structure represents a valid port configuration.

```asm
PUBLIC ver_conf

ver_conf PROC NEAR
    PUSH BP
    MOV BP,SP
    MOV DI, [BP].vc_scratch ; 'scratch' argument from stack
    POP BP
    CALL get_dtok ; Obtain the firmware datacomm device token
    CALL get_dev ; Get the firmware datacomm device
    MOV ES, [DI].va_es ; Initialize firmware ES register
    PUSH DS ; 'vc_scratch' segment
    PUSH DI ; 'vc_scratch' offset
    MOV AX, VERIFY CODE ; Firmware driver function code
    PUSH AX
    PUSH [DI].dev_token ; Device token - set up using 'get_conf'
    CALL dhcp_driver ; Call firmware datacomm driver - returns
    CMP AX, SUCCESS
    JZ conf_ok
    MOV AX, FALSE ; Configuration was bad
    JMP vconf_done
    JMP vconf_done
    conf_ok LABEL NEAR -
    MOV AX, TRUE ; Configuration was valid
    RET
    vconf_done LABEL NEAR
    RET
    ver_conf ENDP
```

7-44
Programming the HP 150

get_dtok PROC NEAR

; Obtain datacomm device token. Saves
; the token in the config data area.
MOW ES,fs_es
XOR AX,AX
MOW AL,[DI].port
PUSH AX
MOW AX,DHP_GDTOK
PUSH AX
CALL dhp_entry
MOW [DI].dev_token,BX
RET

get_dtok ENDP

get_ddev PROC NEAR

; Get the datacomm device allowing for
; exclusive access.
MOW ES,fs_es
PUSH [DI].dev_token
MOW AX,DHP_GDEV
PUSH AX
CALL dhp_entry
RET

get_ddev ENDP

The following function performs the actual port configuration and leaves the
port operational from the firmware standpoint. It should be preceded by the
ver_conf call and only called if the ver_conf call returns TRUE.

;------------------------------------------------------------------------------
; config ( cf_scratch )
;
; /* Attaches the firmware datacomm driver, configures the datacomm port
; using configuration data structure at DS:'cf_scratch', then starts
; the firmware datacomm device driver for the port.
; The attach call initializes the driver so that the configuration is
; active. It is necessary to start the datacomm device driver
; following a configuration call in order to have the configuration
; changes take effect. The 'get_conf' function should be used to
; initialize the configuration data area prior to modification of
; variables in this area for the new configuration.
; Returns TRUE if no error was encountered during calls to the
; firmware, FALSE otherwise */
;
config_args STRUC
   DW ?
   DW ?
   cf_scratch DW ?
config_args ENDS

7-45
Programming the HP 150

PUBLIC config

config PROC NEAR
PUSH BP
MOV BP, SP
MOV DI,[BP].cf_scratch ;'cf_scratch' argument from stack
POP BP

;Attach the firmware datacomm driver -
; initializes the driver so that the
; configuration is active.

MOV ES, fW_es
XOR AX, AX
MOV AL,[DI].port
PUSH AX
MOV AX, 2
PUSH AX
MOV AX, 10
PUSH AX
CALL dhcp_entry
CMP AX, SUCCESS
JNE conf_error

;Initialize firmware ES register

MOV ES, fW_es
PUSH DI
PUSH [DI].port
MOV AX, CONFIG_CODE
PUSH AX
PUSH AX
PUSH [DI].dev_token
CALL dhcp_driver

;Port id word (2 = Port1, 11 = Port2)

;Point-to-point firmware driver

;Firmware driver function code

; (DHP_ATTACH_DRIVER)

;Configure the datacomm port.

;Initialize firmware ES register

; 'cf_scratch' segment

; 'cf_scratch' offset

;Firmware driver function code

;Always returns success

;Start the firmware datacomm driver

;Initialize firmware ES register

;Firmware driver function code

;Device token from scratch area

;Always returns success

;No error during attachment

conf_error LABEL NEAR
MOV AX, FALSE
conf_done LABEL NEAR
RET
config ENDP

;Error while trying to attach driver
RESTORING THE ORIGINAL CONFIGURATION

It is important for applications which modify a port’s configuration to restore the original configuration prior to termination. Note that the above calls do not change the configuration reflected in the port configuration menus. If the ports are not restored to their originally configured state, their actual state will not correspond with that specified in the menus until the next system hard reset.

This applies to programs which access the data communication controller and associated hardware directly. Programs which modify configuration should include the following operations to ensure the hardware is reconfigured prior to program termination.

At program initialization:

1. Call get_conf to read the initial configured state.

2. If the program is going to use the config function to reconfigure the port, rather than going to the hardware directly, the configuration data area should be saved.

At program cleanup:

1. If the program used the config function to change the initial configuration, the configuration data area as saved at initialization should be restored.

2. The ver_conf function is called to obtain the firmware device token and get the firmware device. It also verifies the config data area as being valid (which it should be - it has not been changed).

3. The config function is called to actually restore the initial configuration as reflected by the config data area.
Data Comm Control Functions

The MS-DOS I/O Control For Devices System Function Call implements some special data comm control functions on the HP 150. Data communications data I/O using the MS-DOS Read and Write functions on COM1 or COM2 devices has been described earlier in this section. In addition to some special control functions, alternative and possibly preferable data I/O methods are described here.

IOCTL READS FOR INPUT

In addition to using the MS-DOS Read From a File/Device System Function (3FH) input may be accomplished through the I/O Control For Devices Function, request 2: "Read bytes from device control channel". Data channel bytes are returned for the COM devices. There are two principal advantages to input in this manner. First, the call gets any characters already received by the system without waiting for additional data (even if the requested number of bytes are not returned). Second, the call provides for optimal performance without having to put the device in "RAW" mode. The caller should check the AX register when MS-DOS returns (as specified for the function) to determine the number of bytes transferred.

An example of IOCTL data input is shown below. It uses the doscall function previously described in this section.

```c
#define IOC_READ Ox4402

int com_recv ( handle, buffer, length )

    /* Reads information from the datacomm device (COM1/COM2 previously 
       opened, file handle 'handle') into a character buffer at 'buffer'. 
       Will read a maximum of 'length' bytes. Returns number of bytes 
       actually read (0 for none read) */

    int handle;
    char *buffer;
    int length;

    {
        doscall ( IOC_READ, handle, length, buffer );
        return ( (dc_ret.ah << 8) + dc_ret.al );
    }

SPECIAL COM FUNCTIONS IMPLEMENTED THROUGH IOCTL WRITE REQUEST

The MS-DOS I/O Control for Devices System Function with Request 3: "Write to device control function" is used to implement a set of sub-functions for datacomm control. The defined sub-functions are:
(8,1) - Set 7-bit (normal) mode for both transmit and receive
(8,2) - Set 8-bit (binary) mode for both transmit and receive
(8,3) - Set transparent (monitor) mode
(8,4) - Disable transparent (monitor) mode
(8,5) - Modem disconnect
(8,6) - Send break
(8,7) - Fast buffer send
(8,8) - Reset data comm

The functions are specified with a two-byte identifier whose first byte is 8 to
differentiate them from control requests to other devices. They are initiated
by performing the MS-DOS I/O Control for Devices System Function with request 3.
The buffer specified by the MS-DOS function (DS:DX) should contain the function
code. For example, to send a break:

\[
\text{doscall ( IOC_WRITE, handle, length, &sub function );}
\]

where:

\[
\text{IOC WRITE is defined as 4403 Hex}
\]

and:

\[
\begin{align*}
\text{handle} & = \text{device handle from device open call} \\
\text{length} & = 2 \text{ (sub-function number length)} \\
\text{sub function} & = 0807 \text{ Hex}
\end{align*}
\]

Note that function (8,7) is a special case, described below.

The following definitions are in order:

**BINARY MODE** The eighth bit is not stripped from the data stream of received
characters. In non-binary mode, either 7 or 8 bits are received as configured
for the port (in the configuration menu). That is, when the port is configured
for 7 data bit operation (in the config menu), the eighth data bit (parity) is
masked. In binary mode, the 7/8 data bits configured state is effectively
overridden and a full 8 data bits are returned. The eighth bit will be the
parity bit if the port is configured for 7 data bits in the config menu.

**TRANSPARENT MODE** In transparent mode, control characters normally resulting
in some action being performed by the firmware datacomm driver are sent through
as data instead. For example, if ENQ/ACK or XON/XOFF hanshaking for the port is
enabled in the configuration menu for the port, placing the port in transparent
mode effectively disables the handshaking from taking place. The handshaking
settings in the configuration menus are not altered however. Both receive and
transmit handshaking are effected. In addition, transparent mode causes parity
checking to be disabled.

**SEND BREAK** The transmit data line is driven active (on) for 210 milliseconds.
This is commonly used to signal a special condition to the host.
DISCONNECT MODEM  Commonly used to cause a modem attached to the port to go
do-off-line. The Terminal Ready control line output is driven inactive for two
seconds.

RESET DATA COMM  This function disables transparent mode and sets 7 bit
(normal) mode. It has no other effect.

FAST BUFFER SEND FOR OUTPUT

For improved performance over the "Write to a File/Device" MS-DOS System
Function for data output to the communications ports, a "fast send" I/O Control
for Devices sub-function is offered. This sub-function is implemented in a
manner similar to the "send break" call above however the buffer pointed to by
DS:DX is slightly different. In addition to the sub-function number, three
words follow it. The buffer structure for fast send is:

```c
struct fcs_buf_tem {
    int fcode;  /* sub-function number 0807 Hex */
    int seq_offset; /* send data buffer offset address */
    int seq_segment; /* send data buffer segment address */
    int count;     /* length of send data buffer (bytes) */
};
```

So a Microsoft C language compatible function to implement the fast send
sub-function can be defined as follows:

```c
com_send ( handle, buffer, length )

/* Using "fast send buffer" function of IOCTL for datacomm device to
   send 'length' bytes from 'buffer' to datacomm output channel with a
   handle of 'handle'. */

int handle;
char *buffer;
int length;
{
    fcs_buf.fcode = 0x0807;
    fcs_buf.seq_offset = buffer;
    fcs_buf.seq_segment = dseg;
    fcs_buf.count = length;

doscall ( IOCTL_WRITE, handle, 8, &fcs_buf );
}
```
where:

 IOC_WRITE is defined as 4403 Hex

and:

\[\text{handle} = \text{device handle from device open call}\]
\[\text{buffer} = \text{address (offset) of buffer containing send data}\]
\[\text{length} = \text{number of bytes in buffer}\]

\[dseg\] is equal to the applications data segment register (DS)
HPIB INTERFACING

WARNING

This facility is meant for use with NON-DISC devices. Templates can interfere with the disc driver used by the MSDOS file system. In general, problems can occur if any action on the bus changes conditions at a disc drive. For example, since a UNIVERSAL DEVICE CLEAR resets the head on an 82905 drive, it should not be used. Let the user beware.

Limited HPIB Driver Functionality

It should be understood that the HP 150 does not implement the full set of HPIB functions through its firmware and as such has limited capabilities in an instrument-control environment. In general the firmware-based HPIB driver does not support and activity which may be initiated by a peripheral.

The computer peripherals on the HP 150 (discs, plotters, printers etcetera) are slave devices to the HP 150, which acts as a controller at all times. The HP 150 initiates all HPIB operations. The limitation is a firmware restriction, the HP 150 has a full hardware implementation of the HPIB standard. Therefore it may be necessary to write RAM-based drivers which deal directly with the HPIB controller hardware to support certain HPIB configurations.

Specifically the following conditions are not supported by the HP 150 firmware interface:

- SRQ (Service Request) Function
- TRIGGER Function
- Multiple controllers
- Transactions between other devices on the same bus

Opening the "HPIBDEV" Device

MSDOS functions are called by placing parameters in registers and executing a software interrupt 21 (hexadecimal). Controlling the HPIB device requires familiarity with two MSDOS functions. The "OPEN" function provides access to the HPIB driver. Once the driver is available, the "IOCTRL" function specifies the actions to be performed on the HPIB.

The OPEN function returns a "file handle" that is used by the IOCTRL function. To OPEN the HPIB device, set up the registers as follows and perform a software interrupt 21 hexadecimal.

\[ AH = \text{OPEN function number} = 61 \ (3Dh) \]
\[ AL = \text{access requested} = 0, 1, \text{or } 2 \]
\[ DS,DX \rightarrow \text{the ASCII name of the device} = \text{HPIBDEV} \]
HPIB Control Calls

MS-DOS IOCTRL FUNCTION CALL. Now that the HPIB device is opened, the IOCTRL function is used to control the HPIB. The function IOCTRL requires:

\[
\begin{align*}
\text{AH} & = \text{IOCTRL function number} = 68 (44h) \\
\text{AL} & = \text{subfunction number} = 3 \\
\text{BX} & = \text{file handle returned by OPEN function} \\
\text{CX} & = \text{count of bytes in the CONTROL BLOCK} \\
\text{DX,DX} & \rightarrow \text{the CONTROL BLOCK}
\end{align*}
\]

CONTROL BLOCK FORMAT. The CONTROL BLOCK has the following format:

<table>
<thead>
<tr>
<th>Parm/Ret</th>
<th>SIZE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parm</td>
<td>word</td>
<td>Function</td>
</tr>
<tr>
<td></td>
<td></td>
<td>high byte = 0 (numbers are in decimal)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>low byte = use of Control Block</td>
</tr>
<tr>
<td></td>
<td></td>
<td>17 = Perform this new template</td>
</tr>
<tr>
<td></td>
<td></td>
<td>18 = Continue this template</td>
</tr>
<tr>
<td></td>
<td></td>
<td>--used to recover from a --parallel poll</td>
</tr>
<tr>
<td>Return</td>
<td>word</td>
<td>Status of function</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Success</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 = Busy (try NEW function again)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 = Timeout Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 = EOI Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 = Bus Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9 = Poll Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 = Undefined Error - HPIB bus protocol error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 = Overall Timeout Error</td>
</tr>
<tr>
<td>Parm</td>
<td>word</td>
<td>Data Buffer Length</td>
</tr>
<tr>
<td>Return</td>
<td>word</td>
<td>Number of data bytes sent or received</td>
</tr>
<tr>
<td>Parm</td>
<td>word</td>
<td>Offset of Data Buffer</td>
</tr>
<tr>
<td>Parm</td>
<td>word</td>
<td>Segment of Data Buffer</td>
</tr>
<tr>
<td>Parm</td>
<td>word</td>
<td>Offset of Template</td>
</tr>
<tr>
<td>Parm</td>
<td>word</td>
<td>Segment of Template</td>
</tr>
<tr>
<td>Return</td>
<td>word</td>
<td>Return Template Pointer</td>
</tr>
<tr>
<td>Parm</td>
<td>word</td>
<td>Overall Time Limit in milliseconds</td>
</tr>
<tr>
<td>Parm</td>
<td>byte</td>
<td>HPIB Device address</td>
</tr>
<tr>
<td>Parm</td>
<td>byte</td>
<td>HPIB Unit address</td>
</tr>
</tbody>
</table>
CONTROL TEMPLATE FORMAT. The template is a series of commands, and optionally, data bytes to the HPIB driver. Four types of commands exist with the high two bits of the command determining the type. The low 6 bits are qualifiers that further identify the command.

```
<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>command type</td>
<td>SBUF</td>
<td>PPE</td>
<td>EOI</td>
<td>TERM</td>
<td>UNIT</td>
<td>ADDR</td>
<td></td>
</tr>
</tbody>
</table>
```

The following equates are used in the sample templates to denote command type.

**Template Commands** (bits 6,7)

- **SET VALUE** = 0 changes a configuration value in the HPIB driver.
- **RECEIVE** = 040h indicates data should be read. The number of bytes read and where they are placed is determined by the lower 6 bits of the command. See the qualifiers described below. If the SBUF bit is clear, then one data byte is read and it is placed at the next location in the template. Regardless of the state of SBUF, the next byte is not considered a template command.
- **SEND** ≠ 080h indicates data should be sent. The number of bytes transmitted and where they are coming from is determined by the qualifiers.
- **CONTROLLER** = 0COH indicates the following byte (the controller command) should be transmitted with the attention (ATN) line high.

Some of the controller commands are:

- **UNTALK** = 5Fh
- **UNLISTEN** = 3Fh
- **TALK** = 40h
- **LISTEN** = 20H

The value of a command is created by "OR"ing the equates of the command types mentioned above with the following qualifiers.

**NOTE**

Terminator and time unit are carried from one template to another.

**Qualifiers to Commands**

- **ADDRESS** = 1 (bit 0)
When the CONTROLLER command has the address bit set, the drive address is "OR"ed with the following byte (the controller command). The address bit can be used with a SET VALUE command to specify what HP/IB device address should be used for all the operations specified in the template, otherwise the address specified in the CONTROL BLOCK is used.

---- UNIT = 2h (bit 1)

When the CONTROLLER command has the unit bit set, the unit address is "OR"ed with the following byte (the controller command). The unit bit can be used with a SET VALUE command to change the HP/IB unit address used with all the following template commands. (The default unit address is the one specified in the CONTROL BLOCK.)

---- TERM = 4h (bit 2)

The TERM bit implies the terminator character is of relevance when receiving characters. When used with SET VALUE, the following byte is the terminator character. When used with RECEIVE, the terminator character previously specified will denote the last character to be read.

NOTE

SET_VALUE + EOI + TRM => Read until TRM character, or EOI is received.

---- EOI = 8h (bit 3)

The EOI bit is used with RECEIVE to specify that with the last data byte read, an EOI is expected. If EOI expectancy state is not set correctly, an error is reported. Set Value + EOI => next byte in template is ignored and EOI should be ignored for the rest of the template.

---- PPE = 10h (bit 4)

PPE is used with SET VALUE to specify what devices the parallel poll should be looking for a response from. The following byte indicates the devices. PPE is used with disc drives. The PPE qualifier with RECEIVE, SEND, or CONTROLLER indicates a parallel poll will be performed before the command is executed for the current device id.

---- SBUF = 20h (bit 5)

The SBUF bit is used with RECEIVE or SEND to specify that a buffer rather than a single byte is to be transferred. The buffer offset address and the buffer segment address are always taken from the CONTROL BLOCK. When SET VALUE and SBUF are used, the next byte is the amount of time the HP/IB driver waits for a single character before going into interrupt mode. The time unit is ~ 10 microseconds.
SAMPLE IDENTIFY TEMPLATES. On an identify command, two data bytes are expected with the EOI line asserted after the second data byte is received. The identify command is IDENTIFY_SECONDARY = 60h. The first template reads the data bytes into the buffer specified in the CONTROL BLOCK. The second template places the data bytes in the template itself with labels D1 and D2 allowing easy access to the data.

IDENTIFY TEMPLATE #1

```
DB CONTROLLER        -  C0
DB UNLISTEN          -  3F
DB CONTROLLER        -  C0
DB UNTALK            -  5F
DB CONTROLLER (or) ADDRESS -  C1
DB IDENTIFY_SECONDARY -  60
DB CONTROLLER        -  C0
DB LISTEN + POC      -  3E
DB RECEIVE (or) EOI (or) SBUF -  68
DB 0                 - dummy byte -  00
DB CONTROLLER        -  C0
DB UNTALK            -  5F
DB CONTROLLER        -  C0
DB UNLISTEN          -  3F
DB END_DATA          - end of template-  00
```

IDENTIFY TEMPLATE #2

```
DB CONTROLLER        -  C0
DB UNLISTEN          -  3F
DB CONTROLLER        -  C0
DB UNTALK            -  5F
DB CONTROLLER (or) ADDRESS -  C1
DB IDENTIFY_SECONDARY -  60
DB CONTROLLER        -  C0
DB LISTEN + POC      -  3E
DB RECEIVE           -  40
D1: DB ?              - received byte - ?
DB RECEIVE (or) EOI  -  68
D2: DB ?              - received byte - ?
DB CONTROLLER        -  C0
DB UNTALK            -  5F
DB CONTROLLER        -  C0
DB UNLISTEN          -  3F
DB END_DATA          - end of template-  00
```
SAMPLE READ/WRITE BUFFER TEMPLATES. The following templates assume the buffer specified in the CONTROL BLOCK contains/receives the data.

BLOCK TRANSFER "IN" TEMPLATE

<table>
<thead>
<tr>
<th>DB</th>
<th>SET_VALUE (or) TERM (or) EOI - 0C</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB</td>
<td>? -last data expected- FF</td>
</tr>
<tr>
<td>DB</td>
<td>CONTROLLER - C0</td>
</tr>
<tr>
<td>DB</td>
<td>UNTALK - 5F</td>
</tr>
<tr>
<td>DB</td>
<td>CONTROLLER - C0</td>
</tr>
<tr>
<td>DB</td>
<td>ULISTEN - 3F</td>
</tr>
<tr>
<td>DB</td>
<td>CONTROLLER (or) ADDRESS - C1</td>
</tr>
<tr>
<td>DB</td>
<td>TALK - 40</td>
</tr>
<tr>
<td>DB</td>
<td>CONTROLLER - C0</td>
</tr>
<tr>
<td>DB</td>
<td>LISTEN + POC - 3E</td>
</tr>
<tr>
<td>DB</td>
<td>RECEIVE (or) SBUF - 60</td>
</tr>
<tr>
<td>DB</td>
<td>x -dummy byte- x</td>
</tr>
<tr>
<td>DB</td>
<td>CONTROLLER - C0</td>
</tr>
<tr>
<td>DB</td>
<td>UNTALK - 5F</td>
</tr>
<tr>
<td>DB</td>
<td>CONTROLLER - C0</td>
</tr>
<tr>
<td>DB</td>
<td>ULISTEN - 3F</td>
</tr>
<tr>
<td>DB</td>
<td>END-DATA -end of template- 00</td>
</tr>
</tbody>
</table>

BLOCK TRANSFER "OUT" TEMPLATE

| DB     | CONTROLLER - C0                   |
| DB     | UNTALK - 5F                       |
| DB     | CONTROLLER - C0                   |
| DB     | ULISTEN - 3F                      |
| DB     | CONTROLLER (or) ADDRESS - C1      |
| DB     | LISTEN - 20                       |
| DB     | CONTROLLER - C0                   |
| DB     | TALK + POC - 5E                   |
| DB     | SEND (or) SBUF - A0               |
| DB     | x -dummy byte- x                  |
| DB     | CONTROLLER - C0                   |
| DB     | UNTALK - 5F                       |
| DB     | CONTROLLER - C0                   |
| DB     | ULISTEN - 3F                      |
| DB     | END_DATA -end of template- 00     |
Programming the HP 150

HPIB Interface Example (9111A Graphics Tablet)

The following is an example of using the MS-DOS I/O Control System Function accessing the "HPIBDEV" device to interface an HPIB peripheral. In this case it is a 9111A Graphics Tablet which is interfaced. The example is written in the C language.

The routines obtain and print stylus position and status information. The routines use the doscall function previously described (see "MS-DOS System Function Calls from the C Language"). The routine which returns the DS (data segment) register contents is described following the C code.

Note that this example for the sake of compactness contains minimal error checking. Error checking enhancements should be included for reliable functionality.

```c
/* ***********************************************************************/
#define IO_DEVICE_CTRL 0x4403 /* MS-DOS I/O Control for Devices*/
#define CTRL_FUNCTION 17  /* System Function code, sub-function*/
#define TIME_LIMIT 0x30    /* Write to Device Ctrl Channel */
#define DEV_AND_UNIT_ADDRESS 0x0006 /* HPIBDEV IOCTL Call time limit */
#define SET_VALUE 0x00 /* HPIBDEV IOCTL CALL */
#define RECEIVE 0x40
#define SEND 0x80
#define CONTROLLER 0xc0
#define UNTALK 0x5f  /* HPIB CONTROLLER COMMANDS */
#define UNLISTEN 0x3f
#define TALK 0x40
#define LISTEN 0x20
#define ADDRESS 0x01  /* HPIB CONTROLLER COMMAND QUALIFIERS */
#define UNIT 0x02
#define TERM 0x04
#define EOI 0x08
#define PPE 0x10
#define SBUF 0x20
#define END_DATA 0x00
#define END 0x00
#define MAXLENGTH 128
#define LINEFEED 0x0a
#define EOF 0xff
#define ESC 0x1b

int tab handle;
char buf[12];
```
struct dc_ret_tem { /* 'doscall' return parameter structure */
    char al;
    char ah;
    char bl;
    char bh;
    char cl;
    char ch;
    char dl;
    char dh;
};

extern struct dc_ret_tem dc_ret;

static int hpiib_ctrl_block[11] = { /* Control block used for MS-DOS */
    CTRL_FUNCTION,
    0, 0,
    0, 0,
    0, 0,
    TIME_LIMIT,
    DEV_AND_UNIT_ADDRESS
};

#define CTRL_BLOCK_COUNT 22 /* length of above block */

static char write_template[15] = { /* Template for ASCII writes (the */
    /* only kind) to the tablet */
    CONTROLLER,
    UNTALK,
    CONTROLLER,
    LISTEN,
    CONTROLLER,
    TALK + POC,
    SEND + SBUF,
    0,
    CONTROLLER,
    UNTALK,
    CONTROLLER,
    UNTALK,
    END_DATA
};
static char read_binary[15] = 
{ /* Template for binary reads from tablet */
  CONTROLLER,
  UNTALK,
  CONTROLLER,
  UNLISTEN,
  CONTROLLER + ADDRESS,
  TALK,
  CONTROLLER,
  LISTEN + POC,
  RECEIVE + SBUF + BOI,
  0,
  CONTROLLER,
  UNTALK,
  CONTROLLER,
  UNLISTEN,
  END_DATA
};

static char read_ascii[19] = 
{ /* Template for ASCII reads from tablet */
  SET_VALUE + BOI + TERM,
  LINEFEED,
  SET_VALUE + SBUF,
  EOF,
  CONTROLLER,
  UNTALK,
  CONTROLLER,
  UNLISTEN,
  CONTROLLER + ADDRESS,
  TALK,
  CONTROLLER,
  LISTEN + POC,
  RECEIVE + SBUF + TERM,
  0,
  CONTROLLER,
  UNTALK,
  CONTROLLER,
  UNLISTEN,
  END_DATA
};
/*****************************************************************************************/
main ()
{
    int x, y, status;

    if ( tab_init() == 1 ) { /* initialize graphics tablet */
        while ( kbhit() == 0 ) { /* Continuously request stylus position and status from tablet until keyboard is hit */
            get_xy_status (&x, &y, &status, tab_handle); /* get data */
            printf ("x: %x y: %x status: %s\n", x, y, status); /* display it */
        }
    }
    xenixclose (tab_handle); /* close the tablet */
}

 pasadoon:tab_init()

    /* Initializes the graphics tablet by opening the device and checking for the tablet's response to the Identify function. Returns 0 if the tablet did not identify correctly, returns 1 otherwise */
{
    tab_handle = open_tablet (); /* open the tablet and get handle */
    write_block (tab_handle, "OI;", 3); /* Perform tablet Output Identity */
    if (buf[0] != '9') /* Did tablet identify itself? */
        printf ("9111A Graphics Tablet did not identify\n");
        /* Note - HPIB appears to 'hang' when OI sent to a non-existent tablet */
        return (0);
    else
        return (1);
}

int open_tablet () /* Opens the device "HPIDEB7" and returns handle Note - no error checking! */
{
    return (xenixopen ("HPIDEB7", 2));
}
/***************************************************************************/
int xenixopen (file_name, access) /* open device "file_name" for mode
"access " (read/write/both) */
char *file_name;
int access;
{
    int ax, dummy;
    ax = 0x3d00 + access;
doscall (ax, dummy, dummy, file_name);
    return ((dc_ret.ah << 8) + dc_ret.al); /* return handle */
}

/***************************************************************
write_block (handle, buffer, count) /* Writes "count" ASCII bytes from
"buffer to device "handle" using
MS-DOS Write to a File/Device
System Function Call */
char *buffer;
{
    hpiib_ctrl_block[2] = count;
    /* Update the control block */
    hpiib_ctrl_block[4] = (int) buffer;
    /* with the passed parameters */
    hpiib_ctrl_block[6] = (int) write_template;
    /* Perform the Write to a File/Device System Function Call */
doscall (IO_DEVICE_CTRL, handle, CTRL_BLOCK_COUNT, hpiib_ctrl_block);
}

/***************************************************************
a_read (handle, buffer) /* Reads a block of ASCII data from the specified
device. */
char *buffer;
{
    hpiib_ctrl_block[2] = MAXLENGTH;
    /* Update the control block with */
    hpiib_ctrl_block[4] = (int) buffer;
    /* passed parameters */
    hpiib_ctrl_block[6] = (int) read_ascii;
    /* Perform the read */
doscall (IO_DEVICE_CTRL, handle, CTRL_BLOCK_COUNT, hpiib_ctrl_block);
}
/*****************************************************************************/
get_x_y_status (x, y, status, tab_handle)

/* Reads the stylus position and status from the tablet. Reads until six bytes of data are received. Parses the address and status variables from the return string. Note that due to timing considerations, only five bytes are received in some instances, followed by seven next time */

int *x, *y, *status;
int tab_handle;

{ int my_x, my_y, my_status;

while (b_read (&my_x, tab_handle) != 0); /* Read until valid data */
  *x = ((my_x >> 8) & 0xff) + (my_x << 8); /* Convert to 1byte, hbyte */
  *y = ((my_y >> 8) & 0xff) + (my_y << 8);
  *status = ((my_status >> 8) & 0xff) + (my_status << 8);
}

int b_read (buffer, handle) /* Performs a binary data read from the tablet until an EOI character is detected. Returns the number of bytes read from the tablet */
int *buffer;
int handle;

{ hplb_ctrl_block[2] = 6; /* Update the control block with */
  hplb_ctrl_block[4] = (int) buffer; /* passed parameters */
  hplb_ctrl_block[6] = (int) read_binary;

  /* Perform the read */

doscall (IO DEVICE_CTRL, handle, CTRL_BLOCK_COUNT, hplb_ctrl_block);
return (hpi5_ctrl_block[3]);
}

int close_device (handle) /* Close device with handle of "handle" */
int handle;

{ doscall (Ox3e00, handle);
}
Programming the HP 150

;-----------------------------------------------------------------------------
; unsigned int data_seg()
; /* returns current DS register contents */

PUBLIC data_seg

PROC NEAR
MOV AX,DS
RET

ENDP

PROG ENDS
ACCESSORY CARD INTERFACING

Memory (Slot) Address Identification

16K of contiguous memory space is allocated to each of the two accessory slots on the HP 150. Associated with each slot is a hardware accessory card select line which goes active when a memory read or write occurs to the memory space allocated to that particular slot. The memory blocks are allocated to each slot as follows:

| Accessory Slot 1 | 90000H - 93FFFH |
| Accessory Slot 2 | A0000H - A3FFFH |

Through utilization of the slot select line, accessory boards using memory mapped I/O space may be designed to be slot independent. That is, their memory mapped space will reside within the absolute bounds as defined for the particular slot in which the card is installed. A firmware routine called the Option Handling Processor (OHP) has a function which will return the slot ID (1 or 2) for a particular accessory, if that accessory is installed. Accessories with associated software wishing to utilize this function must have a unique id byte memory mapped to the base address of the 16K block. See the Accessories Subsystem discussion in Section 3 for more information.

The following example shows how to obtain the memory segment address for an installed card given its specific id.

```
ACC_ID EQU 000H ;Put your accessory board id number here
GET_TOKEN EQU 007H ;OHP get slot token function code
SLOT1 EQU 001H ;Slot one id token
SLOT2 EQU 002H ;Slot two id token
S1_SEG EQU 09000H ;Slot one RAM segment address
S2_SEG EQU 0A000H ;Slot two RAM segment address

;MSDOS entry interrupt and function codes
MSDOS EQU 021H ;MSDOS entry interrupt
OPEN_DEV EQU 03DH ;Open device function code
ICNTRL EQU 044H ;I/O control for devices function code

;NS-DOS System Function parameter equates
INPUT EQU 000H ;Open device for input
OUTPUT EQU 001H ;Open device for output
READ EQU 002H ;Read from device
```
Programming the HP 150

DGROUP  GROUP  DATA
DATA  SEGMENT  WORD  PUBLIC 'DATA'
ASSUME  DS:DGROUP

OHP_ENTRY LABEL  DWORD  ; Option handler processor entry point
OHP_OSET  DW  0000CH  ; Offset of option handler processor
OHP_SEG  DW  00040H  ; Segment of option handler processor

HPIBDEV DB  'HPIBDEV',0  ; Firmware device name string
FW_ES  DW  ?  ; Firmware ES value

DATA  ENDS
PAGE

PGROUP  GROUP  PROG
PROG  SEGMENT  BYTE  PUBLIC 'PROG'
ASSUME  CS:PGROUP,DS:DGROUP,ES:NOTHING

;-----------------------------------------------------------------------
;  GET_ACC_SEG  - Get accessory slot segment address for given id. The
;  firmware is called to locate which option slot contains
;  the option board and the accessory slot RAM address
;  segment pointer is set accordingly.
;
;  Registers in: None
;  Registers out: AX = 0 if accessory board with given id not found
;                  Address base pointer (segment) if found
;  Registers preserved: None

PUBLIC GET_ACC_SEG
GET_ACC_SEG PROC  NEAR
    CALL GET_ES
    JNZ INIT_ERR
    MOV AX,ACC_ID
    PUSH AX
    MOV AX,GET_TOKEN
    PUSH AX
    CALL FAR PTR OHP
    OR AX,AX
    JNZ INIT_ERR
    MOV AX,S1_SEG
    CMP BX,SLOT1
    JNZ CHK_S2
    RET

    CHK_S2 LABEL  NEAR
    MOV AX,S2_SEG
    CMP BX,SLOT2
    JNZ INIT_ERR
    RET

    INIT_ERR LABEL  NEAR
    SUB AX,AX
    JNZ ERR
    RET

GET_ACC_SEG ENDP
Programming the HP 150

-------------

; OHP - Far call to firmware option handler processor.
; This routine loads ES with the address of the firmware extra seg
; and then does a long jump to the firmware OHP routine. When
; the OHP does a far return, control is returned to the caller
; of this routine.
; NOTE - The call to this routine must be a far call (32 bit)
;
; Arguments in:
; All arguments to the OHP are pushed on the stack and are
; the callers responsibility.
; Return values:
; Values returned are specified by the OHP and depend on the
; OHP function executed.
; Registers preserved: None

PUBLIC OHP
OHP
PROC FAR ; Far call to firmware OHP
  MOV ES,FW ES ; Load ES for firmware
  JMP DWORD PTR DGROUP:OHP ENTRY ; Intersegment indirect jmp to OHP
  RET ; Dummy return

OHP
ENDP

-------------

; GET_ES - Load and save firmware extra segment address.
; This routine opens a firmware device 'HPIBDEV', reads four bytes
; from it which is a 32-bit pointer to a firmware information block,
; loads the firmware ES value from this information block, and
; stores the ES value for later firmware calls.
;
; Registers in:
; None.
; Registers out:
; FF - Z: everything O.K., firmware ES value stored
; NZ: error in function calls, ES value not stored
; Registers preserved:
; BP, ES

GET_VARS STRUC ; Stack structure for temp variables
FW_PTR DD ? ; Pointer to firmware info block
GET_VARS ENDS

PUBLIC GET_ES
GET_ES
PROC NEAR ; Get and save firmware ES value
  PUSH ES ; Save callers ES
  PUSH BP ; Save callers BP
  SUB SP,4 ; Allocate space for temp variables
  MOV BP,SP ; Initialize base pointer
  MOV AH,OPEN DEV ; Open device
  LEA DX, HPIBDEV ; name 'HPIBDEV'
  MOV AL, INPUT ; for input
  INT MSDOS ;
  JC GET_ERR ; Jump if error on open
Programming the HP 150

MOV BX, AX ; Get device handle for read
MOV AH, IOCNTRL ; I/O control for device
MOV AL, READ ; Read
MOV CX, 4 ; four bytes
LEA DX,[BP].FW_PTR ; into temporary storage
INT MSDOS ; from device handle in BX
JC GET_ERR ; Jump if error on read
CMP AX, 4 ; Jump if four bytes not read
JNZ GET_ERR ;
LES BX,[BP].FW_PTR ; Load pointer to fw info block
MOV AX, ES:[BX] ; Load firmware ES value
MOV FW ES, AX ; Store ES value
ADD SP, 4 ; Release allocated stack space
POP BP ; Restore callers BP
POP ES ; Restore callers ES
SUB AX, AX ; Flag everything O.K.
RET ; Return to caller

GET_ERR LABEL NEAR ; Error occurred
ADD SP, 4 ; Release allocated stack space
POP BP ; Restore callers BP
POP ES ; Restore callers ES
OR AX, OFFFFH ; Set error flag
RET ; Return to caller

GET_ES ENDP

PROG ENDS

END
This section is a reference to the Alpha Graphics Input/Output System (AGIOS) set of function calls on the HP 150.

AGIOS is a facility that lets you use system routines to perform tasks on the HP 150 keyboard and display. They let you perform text and graphics mode operations on your display, let you define and use softkeys (function keys), and let you perform all touch screen operations.

This manual contains information elsewhere pertinent to using the AGIOS function set. Section 5, System Software, provides an introduction to the Alpha Graphics I/O System and includes a simple example of an AGIOS function call. Section 7, Programming the HP 150, describes a general AGIOS caller function for the C language. This function provides a relatively easy way for C programs (and programs in other languages with suitable modifications to the agios function) to invoke any AGIOS function. Section 7 also includes many examples of AGIOS calls from the C language.
CONTENTS

Syntax Used in the AGIOS Function Calls 8-1
Batch Function Call 8-2
Video Intrinsics 8-4
Define Area 8-5
Write Area 8-5
Clear Area 8-6
Enhance Area 8-6
Read Area 8-6
Shift Area 8-7
Write Line 8-8
Application Softkeys 8-9
Update Softkey Label 8-9
Read Softkey Label 8-9
Display Softkey Labels 8-10
Control Functions 8-11
Execute Two Character Sequence (ESC char) 8-11
Position Cursor (ESC & a) 8-12
Define Enhancements (ESC & d) 8-13
Cursor Sense Absolute (ESC a) 8-13
Cursor Sense Relative (ESC `) 8-13
Set Cursor Type 8-14
Read Cursor Type 8-14
Read Terminal Configuration 8-14
Touch Screen Functions 8-15
Field Operations 8-15
Row Column Operations 8-16
Define Touch Field (ESC - z g) 8-17
Define Softkey Field (ESC - z s) 8-18
Delete Touch Field (ESC - z d) 8-18
Touch Screen Reset (ESC - z j) 8-18
Set Touch reporting Modes (ESC - z n) 8-19
Keyboard Intercept 8-20
Define Key Characteristics 8-21
Get Key Characteristics 8-21
Put Key 8-22
Keycode ON/OFF 8-22
Keycode Status 8-23
Read Keypad Status 8-23
Display Control (ESC * d) 8-24
Clear Graphics Memory (ESC * d a) 8-24
Set Graphics Memory (ESC * d b) 8-24
Turn On Graphics Display (ESC * d c) 8-24
Turn Off Graphics Display (ESC * d d) 8-25
Turn On Alphanumeric Display (ESC * d e) 8-25
Turn Off Alphanumeric Display (ESC * d f) 8-25
Turn On Graphics Cursor (ESC * d k) 8-25
Turn Off Graphics Cursor (ESC * d l) 8-26
Turn On Rubber Band Line (ESC * d m) 8-26
Turn Off Rubber Band Line (ESC * d n) 8-26
Move Graphics Cursor Absolute (ESC * d <x,y> o) 8-26
Move Graphics Cursor Incremental (ESC * d <x,y> p) 8-27
Turn On Alphanumeric Cursor (ESC * d q) 8-27
Turn Off Alphanumeric Cursor (ESC * d r) 8-28
Turn On Graphics Text Mode (ESC * d s) 8-28
Turn Off Graphics Text Mode (ESC * d t) 8-28
Vector Drawing Mode (ESC * m) 8-29
Select Drawing Mode (ESC * m <mode> a) 8-29
Select Line Type (ESC * m <type> b) 8-29
Define Line Pattern and Scale (ESC * m <pattern><scale> c) 8-30
Define Area Fill Pattern (ESC * m <pattern> d) 8-30
Fill Rectangular Area, Absolute (ESC * m <x1,y1,x2,y2> e) 8-31
Fill Rectangular Area, Relocatable (ESC * m <x1,y1,x2,y2> f) 8-31
Select Polygonal Fill Pattern (ESC * m <pattern> g) 8-31
Select Boundary Pen (ESC * m <pen> h) 8-32
No Polygon Boundary (ESC * m h) 8-32
Set Relocatable Origin (ESC * m <x,y> j) 8-33
Set Relocatable Origin to Pen Position (ESC * m k) 8-33
Set Relocatable Origin to Cursor Position (ESC * m l) 8-33
Graphics Text (ESC *) 8-34
Set Graphics Text Size (ESC * m <size> m) 8-34
Set Graphics Text Orientation (ESC * m <orientation> n) 8-34
Turn On Text Slant (ESC * m o) 8-35
Turn Off Text Slant (ESC * m p) 8-35
Set Graphics Text Origin (ESC * m <0-9> q) 8-36
Graphics Text Label (ESC * l <text>) 8-36
Define User Character Set 8-37
Select Default Character Set 8-37
Output Single Text Character 8-37
Set Graphics Default (ESC * m r) 8-38
Set Picture Definition Defaults (ESC * m l r) 8-39
Graphics Hard Reset (ESC * w r) 8-39
Graphics Plotting (ESC * p ) 8-40
Lift Pen (ESC * p a) 8-40
Vector Move (ESC * p a <x,y>) 8-40
Lower Pen (ESC * p b) 8-41
Vector Draw (ESC * p b <x,y>) 8-41
Plot to Cursor Position (ESC * p c) 8-42
Point Plot (ESC * p d) 8-42
Set Relocatable Origin to Pen Position (ESC * p e) 8-42
Start Polygonal Area Fill (ESC * p s) 8-43
Terminate Polygonal Area Fill (ESC * p t) 8-43
Polygon Move 8-43
Polygon Draw 8-44
Lift Boundary Pen (ESC * p u) 8-44
Lower Boundary Pen (ESC * p v) 8-45
Graphics Status (ESC * s ) 8-46
Read Device ID (ESC * s 1) 8-46
Read Pen Position (ESC * s 2) 8-46
Read Cursor Position (ESC * s 3) 8-47
Read Cursor Position, Wait For Key (ESC * s 4) 8-47
Read Display Size (ESC * s 5) 8-48
Read Graphics Settings (ESC * s 6) 8-48
Read Graphics Text Status (ESC * s 7) 8-49
Read Zoom Status (ESC * s 8) 8-49
Read Relocatable Origin (ESC * s 9) 8-50
Read Reset Status (ESC * s 10) 8-50
Read Area Shading (ESC * s 11) 8-51
Read Dynamics (ESC * s 12) 8-51
Read Extended Screen Dimensions 8-52
SYNTAX USED IN THE AGIOS FUNCTION CALLS

Each AGIOS function call is explained in detail on the succeeding pages. In order to clarify the information that you need to supply, a standard notation is used. Parentheses and positional notation is used as follows:

- **PARM** Indicates a single byte parameter.
- **(PARM1,PARM2)** Indicates two single byte parameters with parm1 in the high byte and parm2 in the low byte.
- **(,PARM)** Indicates a single byte parameter in the low order byte of the word. The high byte is ignored.
- **(PARM,)** Indicates a single byte parameter in the high order byte of the word. The low byte is ignored.
- **(PARM)** Indicates a word (16 bit) parameter.
- **((PARM))** Indicates a double word parameter. Usually the first word is a data segment address and the second word is an offset address.

Where applicable, the AGIOS call name is followed by the corresponding escape sequence. For example, one entry later in this section is:

```
DEFINE TOUCH FIELD (ESC - z g)
```

This indicates that the escape sequence which corresponds to the Define Touch Field AGIOS call is ESC - z g.
BATCH FUNCTION CALL

A special function call is available which lets you execute a sequence of function calls automatically. Using this function call is especially convenient when you frequently perform the same set of AGIOS function calls.

To "batch" function calls, you set up the sequence of AGIOS function calls in a command buffer. Then you issue the following batch function call using the command buffer as one of its parameters.

(0, 0) Function code
(BUFFER LENGTH) COMMAND BUFFER length (byte count).

(COMMAND BUFFER) A pointer to the buffer containing the AGIOS function calls. The function calls are defined consecutively. Use the same parameter format as specified in this section for the individual function calls.

A batch call is aborted when any of the function calls in the batch causes an error condition. Additionally, you cannot nest batch function calls (include them in the batch).

Example:

This example clears the alpha display by homeing up first, then clearing the display. Refer to the "H" and "J" options of the "Execute Two Character Sequence" function call.

The command buffer looks like this:

<table>
<thead>
<tr>
<th>Contents---&gt;</th>
<th>16</th>
<th>0</th>
<th>H</th>
<th>16</th>
<th>0</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte --&gt;&gt;</td>
<td>+0</td>
<td>+1</td>
<td>+2</td>
<td>+3</td>
<td>+4</td>
<td>+5</td>
</tr>
</tbody>
</table>
The assembler routine that sets up the command buffer and executes a batch call might look like this:

```
CLS  PUSH DS          ; Save DS on Stack
    POP  CMDSEG        ; And Store it in Batch Buffer

    MOV  AX,4403H      ; I/O Control Write
    MOV  BX,1          ; Console Handle
    MOV  CX,8          ; Batch Buffer Length
    MOV  DX,offset BATCH ; Batch Command Buffer
    INT  21H
    RET

; CMDBUF DB 16,0,'H',16,0,'J'

; BATCH DB 0,0        ; AGIOS Batch Command
; BUFLEN DW 6          ; CMDBUF Len 6 Here
; CMDOFF DW CMDBUF     ; CMDOFF equates CMDBUF
; CMDSEG DW 0          ; Data Segment Dummy Value
```
VIDEO INTRINSICS

The video intrinsics are a set of functions that may be used to update the state of the display. With the exception of the Write Line function, all intrinsics operate on a pre-defined subset area of the 24 by 80 character display. All row and column values are relative to zero. The upper left corner of the display is (0,0) and the lower right corner is (23,79).

A null data buffer pointer (segment = 0FFFFH) will suppress the update operation for that data type. There is a one to one correspondence between the position of a data byte in its buffer and the character position that it will affect in the pre-defined update area, starting at the upper left corner, incrementing column position first and then row position.

The ASCII data consists of the 8 bit HP Standard ASCII character code. The character set data consists of character set code characters as follows:

<table>
<thead>
<tr>
<th>CHARACTER CODE</th>
<th>CHARACTER SET SELECTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>@</td>
<td>Normal Roman</td>
</tr>
<tr>
<td>A</td>
<td>Line Drawing</td>
</tr>
<tr>
<td>B</td>
<td>Bold Face Roman</td>
</tr>
<tr>
<td>C</td>
<td>Italic Roman</td>
</tr>
<tr>
<td>D</td>
<td>Math</td>
</tr>
<tr>
<td>SPACE</td>
<td>No Change</td>
</tr>
</tbody>
</table>

The enhancement data consists of enhancement code characters as follows:

security off: @ A B C D E F G H I J K L M N O
security on : P Q R S T U V W X Y Z [ \ ]

-------------|-----------------------------------------------
half-bright | x x x x x x x x
underline    | x x x x x x x x x
inverse video| x x x x x x x x
blinking     | x x x x x x

(A space, 20H, indicates no change to the current state.)
DEFINE AREA

This function specifies the area to be operated upon by subsequent area update operations.

(0, 1) Function Code.

(LR-ROW,LR-COL) Defines the lower right corner of the area to be operated upon.

(UL-ROW,UL-COL) Defines the upper left corner of the area to be operated upon.

((PREV-COORD)) A pointer to a buffer where the previous area coordinates are returned. The format of the returned data is the same as the two input coordinates. This buffer pointer may be null.

WRITE AREA

This function writes data into the pre-defined display area.

(0, 2) Function Code

(DATA LENGTH) Length of the data buffers.

((ENH POINTER)) A double word pointer to a buffer of enhancement code characters to be used to change the enhancement state of the update area.

((CHAR SET)) A double word pointer to a buffer of character set code characters to be used to change the character set state of the update area.

((ASCII POINTER)) A double word pointer to the buffer of ASCII data to be written into the update area.
CLEAR AREA

This function clears the most recently defined area to ASCII blanks (20H), with no enhancements set.

(0, 3) Function Code

ENHANCE AREA

This function sets the enhancement state of the entire pre-defined display area to the specified enhancement.

(0, 4) Function Code

(,ENHANCEMENT) An enhancement code character.

READ AREA

This function reads data from the pre-defined display area.

(0, 5) Function Code

(DATA LENGTH) Length of the data buffers.

((ENH POINTER)) A double word pointer to the buffer that the enhancement data will be read into.

((CHAR SET)) A double word pointer to a buffer that the character set data will be read into.

((ASCII POINTER)) A double word pointer to the buffer that the ASCII data will be read into.
SHIFT AREA

This function shifts the data in the pre-defined display area. Enhancements and character set are shifted with the ASCII data. Data shifted off an edge of the update area is lost.

Function Code

(0, 6) Function Code

(DATA LENGTH) Length of the data buffer.

((ENH POINTER)) A double word pointer to a buffer of enhancement codes to be used to enhance the remaining unshifted area.

((CHAR SET)) A double word pointer to a buffer of character set code characters to be used to change the character set state of the remaining unshifted area.

((ASCII POINTER)) A double word pointer to a buffer of ASCII data to be written into the remaining unshifted area.

(DIRECTION, DIST) DIRECTION:

0 = up
1 = down
2 = left
3 = right

DIST:

The number of rows/columns to shift the current video data.
WRITE LINE

This function writes a single row (or part of a row) in the workspace. Unlike Write Area, this intrinsic ignores the area bounds set by Define Area. If the position and length of the data are defined such that the right workspace boundary is violated, that portion of the data exceeding the boundary is ignored. No line wrap occurs.

(0, 7)  Function Code

(WKSP-ROW,WKSP-COL) Defines the workspace relative position at which the data will be written.

(DATA LENGTH)  Length of the data buffers.

((ENH POINTER))  A double word pointer to the buffer of enhancement data to be written at the designated position.

((CHAR SET))  A double word pointer to a buffer of character set code characters to be used to change the character set state of the update area.

((ASCII POINTER))  A double word pointer to the buffer of ASCII data to be written at the designated position.
APPLICATION SOFTKEYS

This section gives the AGIOS function calls that support Application Softkeys. You can access these softkeys by typing [Shift] [User System]. You can use ASCII and Extended Roman characters within Application Softkey labels.

UPDATE SOFTKEY LABEL

This function will update a softkey label and enhancement.

(0, 8) Function Code

(,NUMBER) Softkey Number (the softkey number is from 1 to 8 inclusive)

((DATA)) A double word pointer to the buffer of ASCII data to be written into the label area. (16 bytes.)

(,TOP ENH) Enhancement code for the top half of the label.

(,BOT ENH) Enhancement code for the bottom half of the label.

READ SOFTKEY LABEL

This function gets the softkey number specified by the caller and then returns the softkey label and the enhancement code characters in two buffers.

(0, 9) Function Code

(,NUMBER) Softkey Number

((DATA)) A double word pointer to the buffer which is for the ASCII data.

((ENHANCEMENTS)) A double word pointer to the buffer which is for the enhancement code characters.
DISPLAY SOFTKEY LABELS

This function displays the application softkey labels in the softkey window.

(0,11) Function Code
CONTROL FUNCTIONS

EXECUTE TWO CHARACTER SEQUENCE (ESC CHAR)

(0,16) Function Code

OP-CHAR The character equivalent of a 2 character escape sequence. Any operation characters are valid except for those that return data. The following list defines some of the most common ones.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Dump Alpha to Printer</td>
</tr>
<tr>
<td>1</td>
<td>Set tab</td>
</tr>
<tr>
<td>2</td>
<td>Clear tab</td>
</tr>
<tr>
<td>3</td>
<td>Clear all tabs</td>
</tr>
<tr>
<td>4</td>
<td>Set left margin</td>
</tr>
<tr>
<td>5</td>
<td>Set right margin</td>
</tr>
<tr>
<td>9</td>
<td>Clear margins</td>
</tr>
<tr>
<td>@</td>
<td>Delay one second</td>
</tr>
<tr>
<td>A</td>
<td>Cursor up</td>
</tr>
<tr>
<td>B</td>
<td>Cursor down</td>
</tr>
<tr>
<td>C</td>
<td>Cursor right</td>
</tr>
<tr>
<td>D</td>
<td>Cursor left</td>
</tr>
<tr>
<td>E</td>
<td>Reset terminal</td>
</tr>
<tr>
<td>F</td>
<td>Home down</td>
</tr>
<tr>
<td>G</td>
<td>Return</td>
</tr>
<tr>
<td>H</td>
<td>Home up</td>
</tr>
<tr>
<td>I</td>
<td>Tab</td>
</tr>
<tr>
<td>J</td>
<td>Clear display</td>
</tr>
<tr>
<td>K</td>
<td>Clear line</td>
</tr>
<tr>
<td>L</td>
<td>Insert Line</td>
</tr>
<tr>
<td>L</td>
<td>Delete Line</td>
</tr>
<tr>
<td>P</td>
<td>Delete Character w/o Wrap</td>
</tr>
<tr>
<td>Q</td>
<td>Insert Character w/o Wrap</td>
</tr>
<tr>
<td>R</td>
<td>Insert Character Off</td>
</tr>
<tr>
<td>S</td>
<td>Roll Up</td>
</tr>
<tr>
<td>T</td>
<td>Roll Down</td>
</tr>
<tr>
<td>U</td>
<td>Next Page</td>
</tr>
<tr>
<td>V</td>
<td>Previous Page</td>
</tr>
<tr>
<td>W</td>
<td>Format Mode On</td>
</tr>
<tr>
<td>X</td>
<td>Format Mode Off</td>
</tr>
<tr>
<td>Y</td>
<td>Display Functions On</td>
</tr>
<tr>
<td>Z</td>
<td>Display Functions Off</td>
</tr>
<tr>
<td>[</td>
<td>Start Unprotected Field</td>
</tr>
<tr>
<td>]</td>
<td>End Protected Field</td>
</tr>
<tr>
<td>b</td>
<td>Enable Keyboard</td>
</tr>
<tr>
<td>c</td>
<td>Disable Keyboard</td>
</tr>
<tr>
<td>f</td>
<td>Modem Disconnect</td>
</tr>
<tr>
<td>g</td>
<td>Soft Reset Terminal</td>
</tr>
<tr>
<td>h</td>
<td>Home Up</td>
</tr>
<tr>
<td>i</td>
<td>Back Tab</td>
</tr>
<tr>
<td>j</td>
<td>Display Softkey Def Menu</td>
</tr>
<tr>
<td>k</td>
<td>Exit Softkey Def Menu</td>
</tr>
<tr>
<td>l</td>
<td>Memory Lock On</td>
</tr>
<tr>
<td>m</td>
<td>Memru Lock Off</td>
</tr>
<tr>
<td>p</td>
<td>Default [f1] Value</td>
</tr>
<tr>
<td>q</td>
<td>Default [f2] Value</td>
</tr>
<tr>
<td>r</td>
<td>Default [f3] Value</td>
</tr>
<tr>
<td>s</td>
<td>Default [f4] Value</td>
</tr>
<tr>
<td>t</td>
<td>Default [f5] Value</td>
</tr>
<tr>
<td>u</td>
<td>Default [f6] Value</td>
</tr>
<tr>
<td>v</td>
<td>Default [f7] Value</td>
</tr>
<tr>
<td>w</td>
<td>Default [f8] Value</td>
</tr>
</tbody>
</table>

8-11
POSITION CURSOR (ESC & a)

(0,17) Function Code

\textbf{MODE}

\textbf{Bit 0:}
1 = window row address
0 = workspace row address

\textbf{Bit 1:}
1 = relative row address
0 = absolute row address

\textbf{Bit 2:}
1 = negative row address
0 = positive row address

\textbf{Bit 3:}
1 = row address is valid
0 = row address is not valid

\textbf{Bit 4:}
1 = window column address
0 = workspace column address

\textbf{Bit 5:}
1 = relative column address
0 = absolute column address

\textbf{Bit 6:}
1 = negative column address
0 = positive column address

\textbf{Bit 7:}
1 = column address is valid
0 = column address is not valid

\textbf{(COLUMN)} An unsigned integer

\textbf{(ROW)} An unsigned integer
DEFINE ENHANCEMENTS (ESC & d)

(0,18) Function Code
SEC SEC: 1 = on, 0 = off.
ENH ENH: the ESC &d code character (@..0)

CURSOR SENSE ABSOLUTE (ESC a)

(0,19) Function Code
((BUFFER)) A pointer to a buffer where two words are returned. The first word is the column number in binary and the second word is the row number in binary.

CURSOR SENSE RELATIVE (ESC ')

(0,20) Function Code
((BUFFER)) A pointer to a buffer where two words are returned. The first word is the column number number in binary and the second word is the row number in binary.
SET CURSOR TYPE

This function sets the alpha cursor type.

(0,21) Function Code

(,TYPE) Alpha cursor type: 0 = underscore
        1 = inverse cell.

READ CURSOR TYPE

This function reads the alpha cursor type.

(0,22) Function Code

((BUFFER)) A pointer to a word location where alpha cursor type data is stored.

READ TERMINAL CONFIGURATION

This function reads the current terminal configurations.

(0,24) Function Code

((BUFFER)) A word pointer to the buffer where the current configuration is returned.

When this function is complete, BUFFER contains:

(,RRRRRTSP) R = reserved bits,
            T = set if touch screen off,
            S = set if softkeys on,
            P = set if remote port 2.

(KEYBOARD LANGUAGE)

(STRING LANGUAGE)

(OP SYS DEVICE) Bits 0-2: addr. 0-7.
                Bits 3-15: dev. 0 = HP-IB, 1 = Accsy.
TOUCH SCREEN FUNCTIONS

The touch features on the HP 150 can be programmed in a variety of ways. The two general types of touch operations are "Field" operations and "Row/Column" operations. These two types can be intermixed.

Several of the touch screen function calls in this section assume keycode mode. Refer to "Keycode Modes" in Section 7 for information on this mode.

FIELD OPERATIONS

There are four types of touch fields you can define. They are:

ASCII Fields:

This mode is very similar to the User-Definable Softkeys (see Section 4). A buffer of ASCII characters is associated with a touch field. A response string of 0 to 80 ASCII characters is obtained by consecutive keyboard input operations. The first input obtains the first ASCII byte, and the second input obtains the second ASCII byte, etc. The response string is generated when the field is touched and should be indistinguishable from the typing of the same string from the keyboard. Auto-repeat is performed.

Keycode Fields:

Keycode fields require that you be in keycode mode (see "Keycodes", Section 7). The two data words of the response string are treated as a keycode and a qualifier and are processed by the regular keyboard routines. The final response to touch depends on the state and mode of keyboard processing. Touch simulates typing on the keyboard. Releasing simulates releasing your finger from the key. Auto-repeat is performed.

Toggle Fields:

The touch field is defined as a toggle switch. Touching the area toggles the field on and off. Whenever the field is touched, sensing information is passed to the application. The sensing information consists of three data bytes. The data is obtained by three consecutive keyboard input operations. The qualifier word of each data byte returned to the application has the touch screen ID. The three data bytes of sensing information are:

\[
\begin{align*}
01H & \quad \text{toggle on field report opcode} \\
d1 & \quad \text{response string first byte} \\
d2 & \quad \text{response string second byte}
\end{align*}
\]
ARGOS Function Call Reference

02H - toggle off field report opcode
d1 - response string first byte
d2 - response string second byte

Normal Fields:

This type of touch field senses touch and/or release. The sensing information consists of three data bytes. The data can be obtained by three consecutive keyboard input operations. The qualifier word of each data byte returned to the application has the touch screen ID. Auto-repeat is performed. The three data bytes of sensing information are:

05H - field touched report opcode
d1 - response string first byte
d2 - response string second byte

06H - field released report opcode
d1 - response string first byte
d2 - response string second byte

Touch fields can overlap. If they do, then the most recent definition for a character cell takes precedent.

ROW COLUMN OPERATIONS

This type of touch operation returns the row and column position when a touch occurs. The row and column position are returned byte-by-byte using the keyboard input function of the operating system. Three data bytes are returned. The qualifier word returned with each byte of data has the touch screen ID. The data bytes for row/column operations are:

03H - row column touch report opcode
row - touched row number in binary
col - touched column number in binary

The data bytes for release report of row/column are:

04H - row column release report opcode
row - touched row number in binary
col - touched column number in binary
DEFINE TOUCH FIELD (ESC - zg)

(0,32) Function code

((STRING)) Pointer to response string. Points to 2 words for keycode field the first word is the qualifier and the second word is the keycode. Points to 2 bytes for toggle or normal field, 0 - 80 bytes for ASCII field.

(LENGTH) Response string length

(ATTRIBUTE,MODE) Touch ATTRIBUTE:

1 = ASCII field
2 = Keycode field
3 = Toggle field
4 = Normal field

Reporting MODE:

1 = Report when touched
2 = Report when released
3 = Report both touch and release

(ON-ENH,OFF-ENH) Enhancements of the field for on and off state for toggle field. Also enhancements of the field when touched and released for normal, ASCII, and keycode fields.

(CURSOR,BEEP) CURSOR:

0 = do not position cursor
1 = position cursor on touch

BEEP:

0 = do not beep
1 = beep on touch

(LR-ROW,LR-COL) Row and column of the lower right corner of the touch field.

(UL-ROW,UL-COL) Row and column of the upper left corner of the touch field.
AGIOS Function Call Reference

DEFINE SOFTKEY FIELD (ESC - z s)

This function defines one of the eight softkey label areas as a touch field. These fields when touched produce the same response as if the corresponding function key is typed. The default is all softkey touch fields are on.

(0,33) Function code

(MODE,KEY) KEY (Softkey number): 1-8

MODE: 1 = on, 0 = off.

DELETE TOUCH FIELD (ESC - z d)

Deletes the touch field with upper left corner at <row> <col>. Nothing happens if there is no touch field there. The row and column are screen relative coordinates.

(0,34) Function code

(UL-ROW,UL-COL) Row and column position of the field to be deleted.

(OFFH,OFFH) deletes all fields.

TOUCH SCREEN RESET (ESC - z j)

Resets all fields to off.

(0,35) Function code
SET TOUCH REPORTING MODES (ESC - z n)

This function determines if, and how, touch is reported to your application by the HP 150 terminal.

(0,36) Function code

(,SCREEN-MODE) Touch Field and Row/Col sensing:

0 - Disable reporting.
1 - Enable sensing for row/column position. Touch fields are inactive.
2 - Enable sensing for touch fields only. Row/column sensing is inactive.
3 - Enable sensing for both row/column and touch fields. Row/column sensing occurs for areas not defined as touch fields.
4 - Toggles touch screen on and off.
10-14 - Same as 0-4, but causes escape sequence reports to be sent. This form is used ONLY by the system parser.

(,TOUCH-MODE) Sense touch or touch-release: (used with Row/Col sensing only)

1 - Report on Touch.
2 - Report on Release
3 - Report on both Touch and Release
Keyboard Intercept functions let you gain more control over the use of the keyboard. Each of the keyboard keys can be individually set to normal processing or one of the special processing modes. It should be noted that the keycodes for [f1] through [f12] are valid only when the application softkey labels are displayed on the screen with AGIOS function call (0,11).

There is no explicit "get keycode and qualifier" function call. The standard operating system console input function returns the normal ASCII code and also keycodes. The qualifiers are also returned if keycode mode is on.

The qualifier word is composed of the following bit values:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-8</td>
<td>Input Device ID:</td>
</tr>
<tr>
<td></td>
<td>0COH = keyboard</td>
</tr>
<tr>
<td></td>
<td>080H = touch screen</td>
</tr>
<tr>
<td></td>
<td>000H = terminal internal</td>
</tr>
<tr>
<td>7</td>
<td>Special key. If set, the data is a non-ASCII keycode.</td>
</tr>
<tr>
<td>6</td>
<td>Reserved.</td>
</tr>
<tr>
<td>5</td>
<td>Left extend char - set when down.</td>
</tr>
<tr>
<td>4</td>
<td>Right extend char - set when down.</td>
</tr>
<tr>
<td>3</td>
<td>Control - set when down.</td>
</tr>
<tr>
<td>2</td>
<td>Left shift - set when down.</td>
</tr>
<tr>
<td>1</td>
<td>Right shift - set when down.</td>
</tr>
<tr>
<td>0</td>
<td>Repeating key when set.</td>
</tr>
</tbody>
</table>

To properly use the key intercept functions, the application program should first put the operating system's console input device into raw mode. This will allow keycodes to be passed through without interpretation by the operating system. Keycode mode should then be turned on. Finally each of the keys on the keyboard can be set to the desired mode of operation. For more information see "Keyboard Interfacing" in Section 7 of this manual.
DEFINE KEY CHARACTERISTICS

This function lets you alter characteristics of any of the special keys. Specifically, you can:

- Process the key normally (Same as on HP 2623 terminal)
- Intercept the key and pass a keycode to the application for processing
- Ignore the key when it is pressed
- Beep when the key is pressed in combination with the above characteristics

(0,40) Function Code

(CCHARACTERISTICS) Key Characteristics

<table>
<thead>
<tr>
<th>Bit</th>
<th>Action</th>
<th>0</th>
<th>beep</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>intercept</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ignore</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3-15</td>
<td>reserved</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If bit 1 and 2 are both set, the key is treated as an intercept key. When both are zero, the key resumes normal functioning.

(KEYCODE) Keycode from table on previous page. A value of OFEH will set all special keys to the specified characteristics.

GET KEY CHARACTERISTICS

This function returns the characteristics of a key to the caller.

(0,41) Function Code

((BUFFER)) Pointer to a buffer where the key's characteristics will be returned.

(KEYCODE) Keycode
AGIOS Function Call Reference

PUT KEY

This function lets you specify direct the terminal to process the keycode normally. Use this function when you wish to 'process' a keypress which was read in when intercept mode was active. For normal ASCII keys, you would simply 'echo' the character in place of using this function.

(0,42) Function Code

(QUALIFIER) Qualifier

Bit: Interpretation:
15-8 = Input Device ID (0COH = keyboard)
7 = Special key. Must be set.
6 = Reserved
5 = Left extend char, set when down
4 = Right extend char, set when down
3 = Control, set when down
2 = Left shift, set when down
1 = Right shift, set when down
0 = Not used

(KEYCODE) Keycode

KEYCODE ON/OFF

This function turns the keycode mode of the console device on and off. If keycode mode is off, each key hit on the keyboard returns one byte of data when the console input device is read. If keycode mode is on, each key press returns four bytes of data. The first two bytes form a word of qualifiers and the next two bytes form a word of key data. See 'Keycode Modes' in Section 7 for more detail.

(0,43) Function Code

MODE Keycode mode:
1 = on
0 = off
KEYCODE STATUS

This function returns the on/off status of a keycode.

(0,44) Function Code

((BUFFER)) A pointer to a byte location where the keycode on/off status is returned.

READ KEYPAD STATUS

This function returns the status of whether the extended keypad is in numeric or graphics mode.

(0,44) Function Code

((BUFFER)) A pointer to a byte location where the keypad status is returned.

When this function is complete, BUFFER contains 0 if numeric mode is set, and 1 if graphics mode is set.
DISPLAY CONTROL (ESC * d)

CLEAR GRAPHICS MEMORY (ESC * d a)

This function clears graphics display memory to 0. The complete displayable graphics area of 512 x 390 dots are cleared.

(4, 1) Function Code

SET GRAPHICS MEMORY (ESC * d b)

This function sets graphics display memory to 1. The complete displayable graphics area of 512 x 390 dots are set.

(4, 2) Function Code

TURN ON GRAPHICS DISPLAY (ESC * d c)

This function turns on the graphics display. The data in graphics memory is not affected.

(4, 3) Function Code
TURN OFF GRAPHICS DISPLAY (ESC * d d)

This function turns off the graphics display. The data in graphics memory is not affected.

(4, 4)          Function Code

TURN ON ALPHANUMERIC DISPLAY (ESC * d e)

This function turns on the alphanumerics display and alphanumerics cursor. The data in alphanumerics memory is not affected.

(4, 5)          Function Code

TURN OFF ALPHANUMERIC DISPLAY (ESC * d f)

This function turns off the alphanumerics display. The data in alphanumerics memory is not affected.

(4, 6)          Function Code

TURN ON GRAPHICS CURSOR (ESC * d k)

This function turns on the graphics cursor. The data in graphics memory is not affected.

(4, 7)          Function Code
TURN OFF GRAPHICS CURSOR (ESC * d l)

This function turns off the graphics cursor. The data in graphics memory is not affected.

(4, 8) Function Code

TURN ON RUBBER BAND LINE (ESC * d m)

This function turns on the rubber band line and graphics cursor.

(4, 9) Function Code

TURN OFF RUBBER BAND LINE (ESC * d n)

This function turns off the rubber band line.

(4,10) Function Code

MOVE GRAPHICS CURSOR ABSOLUTE (ESC * d <x,y> o)

This function moves the graphics cursor to the specified location. The move occurs even if the cursor is turned off.

(4,11) Function Code

(X-COORD) The X coordinate of the new cursor position expressed as an absolute number in the range of plus and minus 16383.
The Y coordinate of the new cursor position expressed as an absolute number in the range of plus and minus 16383.

MOVE GRAPHICS CURSOR INCREMENTAL (ESC * d <x,y> p)

This function moves the graphics cursor to the specified location. The move occurs even if the cursor is turned off.

Function Code

The X coordinates of the new cursor position expressed as a number that is relative to the current cursor position. Its range extends from -32768 to +32767.

The Y coordinate of the new cursor position expressed as a number that is relative to the current cursor position. Its range extends from -32768 to +32767.

TURN ON ALPHANUMERIC CURSOR (ESC * d q)

This function turns on the alphanumeric cursor. The data in alphanumeric memory is not affected.

Function Code
AGIOS Function Call Reference

TURN OFF ALPHANUMERIC CURSOR (ESC * d r)

This function turns off the alphanumeric cursor. The data in alphanumeric memory is not affected.

(4,14) Function Code

TURN ON GRAPHICS TEXT MODE (ESC * d s)

This function turns on graphics text mode. Characters that normally go to the alphanumeric display will be drawn on the graphics display.

(4,15) Function Code

TURN OFF GRAPHICS TEXT MODE (ESC * d t)

This function turns off graphics text mode.

(4,16) Function Code
VECTOR DRAWING MODE (ESC * m)

SELECT DRAWING MODE (ESC * m <mode> a)

This function selects the vector drawing mode.

\[(4,17)\] Function Code
\[(MODE)\] Drawing Mode:

0 = Graphics memory not changed
1 = Clear mode
2 = Set mode
3 = Complement mode
4 = Jam mode

SELECT LINE TYPE (ESC * m <type> b)

This function selects the vector line type.

\[(4,18)\] Function Code
\[(TYPE)\] Line Type:

1 = 
2 = User defined line pattern
3 = Current area fill pattern
4 = ____________
5 = _____ _____ _____
6 = ________
7 = ________________
8 = ___.__.__.__.__.__.__.
9 = ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___
10= ______________________________________
11= Point plot
AGIOS Function Call Reference

DEFINE LINE PATTERN AND SCALE (ESC * m <pattern><scale> c)

This function defines a user line pattern and scale.

(4,19) Function Code
(,PATTERN) The line pattern expressed as an eight bit binary number.
(SCALE) The line scale expressed as a binary number from 1 to 16.

DEFINE AREA FILL PATTERN (ESC * m <pattern> d)

This function defines a user area fill pattern of 8 by 8 screen dots.

(4,20) Function Code
(,DATA ROW1) You specify all eight rows of the 8 by 8 fill pattern. Each
(,DATA ROW2) DATA-ROW is an eight-bit byte which defines a particular row
(,DATA ROW3) of the pattern.
(,DATA ROW4)
(,DATA ROW5)
(,DATA ROW6)
(,DATA ROW7)
(,DATA ROW8)
FILL RECTANGULAR AREA, ABSOLUTE (ESC * m <x1,y1,x2,y2> e)

This function fills a rectangular area with the selected line or area fill pattern. The rectangular region is defined by specifying the lower left and upper right coordinates.

(4,21) Function Code

(LWR LEFT X-COORD) Each coordinate is in the range of -16384 to +16383.
(LWR LEFT Y-COORD)
(UPR RIGHT X-COORD)
(UPR RIGHT Y-COORD)

FILL RECTANGULAR AREA, RELOCATABLE (ESC * m <x1,y1,x2,y2> f)

This function fills a rectangular area with the selected line or area fill pattern. The rectangular region is defined by specifying the lower left and upper right coordinates.

(4,22) Function Code

(LWR LEFT X-COORD) Each value is in the range from -32768 to +32767.
(LWR LEFT Y-COORD)
(UPR RIGHT X-COORD)
(UPR RIGHT Y-COORD)

SELECT POLYGONAL FILL PATTERN (ESC * m <pattern> g)

This function selects a pattern for polygonal and rectangular area fill.
Function Call Reference

(4,23) Function Code

(PATTERN) Area Fill Pattern:

1 = Solid fill pattern
2 = User-defined fill pattern
3-10 = Pre-defined fill pattern

SELECT BOUNDARY PEN (ESC * m <pen> h)

This function selects the pen to be used to draw the boundary of a filled polygon. The actual value is not significant in a black and white system. This function turns on boundary drawing with a solid line pattern. The drawing of each edge of the boundary can be individually controlled.

(4,24) Function Code

(PEN) Boundary Pen Number

NO POLYGON BOUNDARY (ESC * m h)

This function turns off drawing of boundary around a polygon.

(4,25) Function Code
SET RELOCATABLE ORIGIN (ESC * m <x,y> j)

This function sets the relocatable origin to the specified absolute location.

(4,26) Function Code

(X-COORD) The X coordinate is the new relocatable origin expressed as an absolute number in the range of -16384 to +16383.

(Y-COORD) The Y coordinate is the new relocatable origin expressed as an absolute number in the range of -16384 to +16383.

SET RELOCATABLE ORIGIN TO PEN POSITION (ESC * m k)

This function sets the relocatable origin to the current pen position.

(4,27) Function Code

SET RELOCATABLE ORIGIN TO CURSOR POSITION (ESC * m l)

This function sets the relocatable origin to the current cursor position.

(4,28) Function Code
The HP 150 offers a comprehensive graphics character set in read-only memory (ROM). This standard character set is used by all graphics text operations. However, you do have the ability to create custom characters of your own design and to use these singly or to replace the entire built-in character set.

**SET GRAPHICS TEXT SIZE (ESC * m <size> m)**

This function sets the graphics text size. The vector lists that define the current character set are scaled using this text size.

(4,29) Function Code

(X-SCALE) The X coordinate scale factor for text characters. The format is a 16 bit number with the radix point between bits 7 and 8:

- Bits 1-8 = integer
- Bits 9-16 = fraction

(Y-SCALE) The Y coordinate scale factor for text characters. The format is a 16 bit number with the radix point between bits 7 and 8:

- Bits 1-8 = integer
- Bits 9-16 = fraction

**SET GRAPHICS TEXT ORIENTATION (ESC * m <orientation> n)**

This function selects the graphics text orientation. This also changes the direction of line feed, carriage return, and backspace. The desired orientation is specified by a number defined as:
Graphics Text Orientation:

1 = Normal
2 = Rotate 90 degrees counterclockwise
3 = Rotate 180 degrees counterclockwise
4 = Rotate 270 degrees counterclockwise

TURN ON TEXT SLANT (ESC * m o)

This function turns on the 26.57 degree slant of graphics text characters.

TURN OFF TEXT SLANT (ESC * m p)

This function turns off 26.57 degrees slant of graphics text characters.
AGIOS Function Call Reference

SET GRAPHICS TEXT ORIGIN (ESC * m <0-9> q)

This function sets the graphics text origin to one of twelve positions of text justification. The positions are shown in this figure:

```
3------6------9
|     |     |
|     |     |
|     |     |
|2  5 8|
Base line -> 0 10 11
|     |     |
|1------4------7|
```

(4,33) Function Code

(ORIGIN) Graphics Text Origin:

A number from 0 to 11.

GRAPHICS TEXT LABEL (ESC * l <text>)

This function outputs a string of graphics characters.

(4,34) Function Code

((TEXT)) Segment and offset address of a string of characters. The string must be terminated by CR, LF, CR LF, or LF CR.
DEFINE USER CHARACTER SET

This function lets you re-define the entire graphics character set. All subsequent graphics text operations will use this character set. This includes text size, orientation, slant, and justification.

(4,35) Function Code

((TABLE)) Segment and offset address of the table that points to the vector lists of characters.

SELECT DEFAULT CHARACTER SET

This function sets the character set to the default set maintained by the system. The cell size is 7 x 10.

(4,36) Function Code

OUTPUT SINGLE TEXT CHARACTER

This function outputs a single graphics character defined by a vector list. All current graphics text operations such as size and orientation apply.

(4,37) Function Code

((CHARACTER)) Segment and offset address of the vector list of a single character.
This function sets the graphics parameters to their default values.

The defaults affected by this call are:
- Pen down
- Line type 1
- User-defined line pattern solid
- User-defined area fill pattern solid
- Boundary pen off
- Drawing mode set
- Relocatable origin 0,0
- Text size 1
- Text origin 1
- Text slant off
- Text orientation 1
- Graphics text off
- Graphics display on
- Alphanumeric display on
- Graphics cursor off
- Alphanumeric cursor on
- Rubber band line off
- Graphics cursor position 0,0
SET PICTURE DEFINITION DEFAULTS (ESC * m 1 r)

This function sets the picture definition parameters to their default values.

(4,72) Function Code

(RESET LEVEL) The level of graphics reset. On the HP 150 the value '1' is the only supported level.

The picture defaults are:

- Pen down
- Line type 1
- User-defined line pattern solid
- User-defined area fill pattern solid
- Boundary pen off
- Drawing mode set
- Text size 1
- Text origin 1
- Text slant off
- Text orientation 1
- Graphics text off

GRAPHICS HARD RESET (ESC * w r)

Sets the graphics parameters to their power on state.

(4,73) Function Code
AGIOS Function Call Reference

GRAPHICS PLOTTING (ESC * p )

LIFT PEN (ESC * p a)

This function lifts the pen.

(4,39) Function Code

VECTOR MOVE (ESC * p a <x,y>)

This function lifts the pen and moves the pen to the new coordinate position. The pen is lowered at the end of the operation.

There are three ways to specify the new coordinate position:

(4,40) Function Code
(X-COORD) (Y-COORD) The X and Y numbers give an absolute coordinate position in the range from -16384 to +16383.

(4,41) Function Code
(X-COORD) (Y-COORD) The X and Y numbers give an incremental coordinate position in the range from -32768 to +32767.

(4,42) Function Code
(X-coord) (Y-coord) The X and Y numbers give a relocatable coordinate position in the range from -32768 to +32767.
LOWER PEN (ESC * p b)

This function lowers the pen.

(4,43) Function Code

VECTOR DRAW (ESC * p b <x,y>)

This function lowers the pen and draws a vector to the new coordinate position. The pen is lowered at the end of the operation.

There are three ways to specify the new vector coordinates:

(4,44) Function Code
(X-COORD) (Y-COORD) The X and Y numbers give the absolute coordinates of the vector position. They are in the range from -16384 to +16383.

(4,45) Function Code
(X-COORD) (Y-COORD) The X and Y numbers give the incremental coordinates of the vector position. They are in the range from -32768 to +32767.

(4,46) Function Code
(X-COORD) (Y-COORD) The X and Y numbers give the relocatable coordinates of the vector position. They are in the range from -32768 to +32767.
AGIOS Function Call Reference

PLOT TO CURSOR POSITION (ESC * p c)

This function moves the pen from its current position to the current cursor position if the pen is up. A draw is performed from the current pen position to the current cursor position if the pen is down.

(4,47) Function Code

POINT PLOT (ESC * p d)

This function draws a dot at the current pen position and then lifts the pen.

(4,48) Function Code

SET RELOCATABLE ORIGIN TO PEN POSITION (ESC * p e)

This function sets the relocatable origin to the current pen position.

(4,49) Function Code
START POLYGONAL AREA FILL (ESC * p s)

This function starts polygonal area fill. The boundary pen is lowered with this function.

(4,50) Function Code

TERMINATE POLYGONAL AREA FILL (ESC * p t)

This function terminates the polygon definition and fills the polygon.

(4,51) Function Code

POLYGON MOVE

This function closes the polygon defined up to this point and moves the pen to the new coordinate position to start a new polygon.

There are three ways to specify the new coordinate position:

(4,52) Function Code

(X-COORD) (Y-COORD)

The X and Y numbers give the absolute coordinates of the new position. They are in the range from -16384 to +16383.

(4,53) Function Code

(X-COORD) (Y-COORD)

The X and Y numbers give the incremental coordinates of the new position. They are in the range from -32767 to +32767.

(4,54) Function Code
POLYGON DRAW

This function defines the edge of a polygon from the current pen position to the new coordinate position.

There are three ways that you can specify the new coordinate position:

- **Function Code**
  - **(4,55)**
  - **(X-COORD)**
  - **(Y-COORD)**
  - The X and Y numbers give the absolute coordinates of the new position. They are in the range from -16384 to +16383.

- **Function Code**
  - **(4,56)**
  - **(X-COORD)**
  - **(Y-COORD)**
  - The X and Y numbers give the incremental coordinates of the new position. They are in the range from -32768 to +32767.

- **Function Code**
  - **(4,57)**
  - **(X-COORD)**
  - **(Y-COORD)**
  - The X and Y numbers give the relocatable coordinates of the new position. They are in the range from -32768 to +32767.

LIFT BOUNDARY PEN (ESC * p u)

This function lifts the polygon boundary pen. Undrawn edges of the polygon are not drawn. This remains in effect until the boundary pen is lowered.

- **Function Code**
  - **(4,58)**
LOWER BOUNDARY PEN (ESC * p v)

This function lowers the polygon boundary pen. If a boundary pen has been specified, undrawn edges of the polygon are drawn with a solid line pattern. This remains in effect until the boundary pen is lifted.

(4,59) Function Code
GRAPHICS STATUS (ESC * s)

READ DEVICE ID (ESC * s 1)

This function returns the device id of the HP 150.

(4,60) Function Code

((BUFFER)) Segment and offset address of the buffer to be used for returned device data.

When this function is complete, BUFFER contains an ASCII string that identifies the device.

READ PEN POSITION (ESC * s 2)

This function returns the current position and the state of the pen.

(4,61) Function Code

((BUFFER)) Segment and offset address of the buffer to be used for returned pen status data.

When this function is complete, BUFFER contains:

(X-COORD) The binary X and Y coordinates of the current pen position
(Y-COORD)
(STATE) 0 = pen lifted, 1 = pen lowered
READ CURSOR POSITION (ESC * s 3)

This function returns the current position of the cursor.

(4,62) Function Code

((BUFFER)) Segment and offset address of the buffer to be used for the returned cursor data.

When this function is complete, BUFFER contains:

(X-COORD) The X and Y coordinates of current cursor position.
(Y-COORD)

READ CURSOR POSITION, WAIT FOR KEY (ESC * s 4)

This function returns the current position of the cursor, but lets the user move it on the display first. The user can type any ASCII key or the SELECT key on the keyboard to move the cursor. As soon as one of these characters is typed, the cursor coordinates are returned to the program.

(4,63) Function Code

((BUFFER)) Segment and offset address of the buffer to be used for returned cursor position.

When this function is complete, BUFFER contains:

(X-COORD) The X and Y coordinates of current cursor position.
(Y-COORD)

(CODE) The character code of the key that was typed.
AGIOS Function Call Reference

READ DISPLAY SIZE (ESC * s 5)

This function returns the number of displayable units and also the number of units in millimeters.

(4,64) Function Code

((BUFFER)) Segment and offset address of the buffer to be used for the returned displayable size and unit data.

When this function is complete, BUFFER contains:

(X-LWR-LEFT) The X and Y coordinates of the maximum display size.
(Y-LWR-LEFT)
(X-UPR-RIGHT)
(Y-UPR-RIGHT)

(X-MM) The X and Y dimensions in dots / millimeters.
(Y-MM)

READ GRAPHICS SETTINGS (ESC * s 6)

This function returns information about the current graphics settings in effect.

(4,65) Function Code

((BUFFER)) Segment and offset address of the buffer to be used for returned graphics settings.

When this function is complete, BUFFER contains the settings in consecutive words:

(CLEAR DISPLAY)
(NUMBER OF PENS)
(RESERVED)
(RESERVED)
(AREA SHADING)
(RESERVED)
(RESERVED)
(DYNAMIC MODIFICATION)
(GRAPHICS CHARACTER SIZE)
(GRAPHICS CHARACTER ANGLES)

8-48
READ GRAPHICS TEXT STATUS (ESC * s 7)

This function returns the current attributes of graphics text.

(4,66) Function Code

((BUFFER)) Segment and offset address of the buffer to be used for the returned graphics attributes.

When this function is complete, BUFFER contains:

(X SIZE) The character cell size.
(Y SIZE)

(ORIGIN) The text origin.

(ANGLE) The text orientation.

(SLANT) The character slant.

READ ZOOM STATUS (ESC * s 8)

This function returns the terminal's zoom setting.

(4,67) Function Code

((BUFFER)) Segment and offset address of the buffer to be used for the returned zoom setting.
When this function is complete, BUFFER contains:

(ZOOM SIZE)  1 - 16 (the HP 150 always returns 1).
(ZOOM ON/OFF)  0 = off, 1 = on (the HP 150 always returns 0).

READ RELOCATABLE ORIGIN (ESC * s 9)

This function returns the current relocatable origin.

(4,68) Function Code

((BUFFER)) Segment and offset address of the buffer to be used for
the returned origin.

When this function is complete, BUFFER contains:

(X-COORD) The X and Y coordinates of the current relocatable origin.
(Y-COORD)

READ RESET STATUS (ESC * s 10)

This function returns information on whether the terminal has executed a
full reset since the last time reset status was checked.

(4,69) Function Code

((BUFFER)) Segment and offset address of the buffer to be used for
the returned reset status.

When this function is complete, BUFFER contains:

(RESET STATUS)
(RESERVED)
(RESERVED)
(RESERVED)
(RESERVED)
(RESERVED)
(RESERVED)

8-50
READ AREA SHADING (ESC * s 11)

This function returns information on the area shading capability of the terminal.

(4,70) Function Code

((BUFFER)) Segment and offset address of the buffer to be used for the returned shading data.

When this function is complete, BUFFER contains:

(CAPABILITIES) The area shading capabilities.

(WIDTH) The area shading pattern size.

READ DYNAMICS (ESC * s 12)

This function returns information on the terminal's dynamic graphics capabilities.

(4,71) Function Code

((BUFFER)) Segment and offset address of the buffer to be used for the returned dynamic graphics data.

When this function is complete, BUFFER contains:

(SELECTIVE-ERASE-CAPABILITIES)

(COMPLEMENT-CAPABILITIES)
AGIOS Function Call Reference

READ EXTENDED SCREEN DIMENSIONS

This function provides information about the alpha and graphics screen size plus the relationship between the two.

(0,74) Function Code

((BUFFER)) Segment and offset address of the buffer to be used for the returned screen size data.

When this function is complete, BUFFER contains:

(X-PIXELS) 512 graphics display size in pixels.
(Y-PIXELS) 390

(ROWS) 27 alpha display size.
(COLUMNS) 80

(X-MM) 160 graphics display size in mm.
(Y-MM) 120

(ROW-MM) 150 alpha display size in mm.
(COL-MM) 116

(DELTA-X) 10 graphics origin minus alpha origin in mm.
(DELTA-Y) 4
Schematic diagrams for the HP 150 are provided in this appendix.
# Table of Contents

Touchscreen PCA .................................................................................. A-1
Keyboard PCA ...................................................................................... A-3
Processor PCA ..................................................................................... A-5
Processor Front Plane Interface .......................................................... A-7
Processor I/O Bus .................................................................................. A-9
Processor Datacomm Port ................................................................. A-11
Video Alpha RAM Subsystem ......................................................... A-13
Video Alpha Display Subsystem ....................................................... A-15
Video Graphics Display Subsystem ................................................ A-17
Thermal Printer Interface (Part of Front Plane PCA) ......................... A-19
Mezzanine Memory PCA ................................................................ A-21
Mezzanine Datacomm PCA .............................................................. A-23
Sweep PCA ......................................................................................... A-25
RAM (Memory Extender) PCA .......................................................... A-27
Language PCA ................................................................................. A-29
Figure A-1. Touchscreen PCA
This page is blank
Figure A-2. Keyboard PCA
This page is blank
Figure A-3. Processor PCA
Figure A-4. Processor Front Plane PCA
This page is blank
Figure A-5. Processor I/O Bus
This page is blank
Figure A-6. Keyboard PCA Port
This page is blank
Figure A-7. Video Alpha RAM Subsystem
This page is blank
Figure A-8. Video Alpha Display Subsystem
This page is blank
Figure A-9. Video Graphics Display Subsystem
This page is blank
Figure A-10. Thermal Printer Interface (Part of Front Plane PCA)
This page is blank
Figure A-11. Mezzanine Memory PCA
This page is blank
Figure A-12. Mezzanine Datacomm PCA
This page is blank
Figure A-13. Sweep PCA
This page is blank
Figure A-14. RAM (Memory Extender) PCA
This page is blank
Additional information may be found in the following documents:

<table>
<thead>
<tr>
<th>Title</th>
<th>HP Product No.</th>
<th>HP Part No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>HP 150 Service Manual</td>
<td>45611A</td>
<td>45611-90002</td>
</tr>
<tr>
<td>HP 150 Owner's Guide</td>
<td>45621A</td>
<td>45621-90001</td>
</tr>
<tr>
<td>HP 150 Terminal User's Guide</td>
<td>45623A</td>
<td>45623-90002</td>
</tr>
<tr>
<td>MS-DOS User's Guide</td>
<td>45624A</td>
<td>45624-90001</td>
</tr>
</tbody>
</table>
INDEX

A

AGIOS Buffer, Character Set Code.................................8-4
AGIOS Buffer, Enhancement Code....................................8-4
AGIOS Call Syntax.........................................................8-1
AGIOS Calls.................................................................7-5
AGIOS Control Functions.................................................8-11
AGIOS Function Set.........................................................2
AGIOS Function..............................................................7-5
AGIOS Touch Screen Functions.........................................8-15
AGIOS Video Intrinsics.....................................................8-4
AGIOS, Null Data Buffer..................................................8-4
AGIOS..........................................................5-49, 7-5
AUX Device.................................................................5-16
Accessories Subsystem...................................................3-64
Accessory Card Extractor................................................3-80
Accessory Card Extractor Pin..........................................3-80
Accessory Card Hardware and Electrical.........................3-73
Accessory Card Slot ID....................................................7-65
Accessory Cards............................................................7-65
Accessory Connector Signal Descriptions .......................3-71
Accessory Front Plane Connectors................................3-5, 3-70
Accessory Hardware Design Guidelines.........................3-64
Accessory Signal Loading Restrictions..........................3-65
Accessory Slot Bus Cycle Timing......................................3-69
Accessory Wait State Insertion.......................................3-69
Address Generation......................................................3-14
Alpha Character Cell Format.........................................3-28
Alpha Screen Format...................................................3-26
Alpha Video Enhancements............................................3-30
Alpha/Graphic Input/Output System..............................5-49
Alphanumeric Cursor....................................................8-28
Alphanumeric Display..................................................1-2
Alphanumeric Display Memory........................................7-8
Alphanumericics Display ON/OFF, AGIOS.........................8-25
Analog Boards.............................................................2-6
Application Program Area..............................................5-11
Applications Programs..................................................5-2
Application Softkeys, AGIOS..........................................8-9
Area Fill, AGIOS..........................................................8-30
Attributes, Display......................................................7-10
Availdev Command, Config.Sys......................................5-55

B

BDOS.................................................................5-3, 5-7
BIOS.................................................................5-3, 5-7
BIOS Devices..........................................................5-51
Basic Disc Operating System........................................5-3
Index

Basic Input/Output System .............................................. 5-3
Batch AGIOS Call, Example ........................................... 8-2
Baud Rate Generator .................................................. 3-51
Binary Data Comm Mode .............................................. 7-49
Block Device ................................................................ 5-22, 5-28, 5-30
Block Device Drivers ................................................... 5-35
Block Devices .................................................................. 5-20
Board Blank - No I/O Panel ........................................... 3-80
Board Blank with I/O Panel (Accessory Card Details) ........ 3-80
Board Front Plane Connector ........................................ 3-4
Boot Sector ..................................................................... 5-58
Boot Sector, Disc .......................................................... 5-61
Booting .......................................................................... 5-61
Booting, System ........................................................... 5-55
Break Command, Config.Sys ......................................... 5-55
Break, Data Communications ......................................... 7-49
Buffers Command, Config.Sys ....................................... 5-55
Build BPP Device Function ........................................... 5-33
Built-In Commands ....................................................... 5-2
Bus Cycles ....................................................................... 3-14

C

CLOCK Device .................................................................... 5-16
CMOS Decoding and Access ........................................... 3-62
CMOS Power ................................................................. 3-62
CMOS Power Circuit ..................................................... 3-13
CMOS Power, and Test Strap Logic................................. 3-13
CMOS RAM ...................................................................... 3-61
COM Devices .................................................................... 7-34
COM1 Device ................................................................... 5-16
COM2 Device ................................................................... 5-16
COMMAND.COM .............................................................. 5-15
CON Device ....................................................................... 5-16
CRT .................................................................................. 3-23
CRT Controller Registers ............................................... 4-3
Character Cell Example ............................................... 3-30
Character Device .......................................................... 5-22
Character Devices ......................................................... 5-20
Character Set Code, AGIOS .......................................... 8-4
Character Sets, Graphics ............................................... 8-37
Clear Area, AGIOS ........................................................ 8-6
Clear Graphics Memory, AGIOS ..................................... 8-24
Clock Device .................................................................... 5-52
Clock Generation .......................................................... 3-35
Clock Generator ........................................................... 3-14
Clusters, Disc ................................................................. 5-62
Command Processor ....................................................... 5-2
Command.Com .............................................................. 5-2
Communication Port ....................................................... 1-3
Communications .......................................................... 2-3
Communications Interface Circuitry ............................... 3-53
Communications/Peripherals ......................................... 1-3
Con Device ....................................................................... 5-51
Index

Config.Sys .........................................................5-2, 5-8, 5-10, 5-24
Config.Sys File .....................................................5-55
Configuring Data Comm ..............................................7-39
Console Device ....................................................5-51
Console RAW/COOKED Mode ..........................................7-23
Control C Check .....................................................5-55
Control Functions, AGIOS ..........................................8-11
Current Block .......................................................5-9
Current Record ......................................................5-10
Cursor ON/OFF, AGIOS ..............................................8-28
Cursor Positioning ..................................................7-6
Cursor Positioning, AGIOS ..........................................8-12
Cursor Sense Absolute, AGIOS ......................................8-13
Cursor Sense Relative, AGIOS ......................................8-13
Cursor Sensing, AGIOS ..............................................8-13
Cursor Type, Reading ...............................................8-14
Cursor Type, Setting ................................................8-14
Cursor, Alphanumeric ...............................................8-27
Cursor, Graphics ....................................................8-26

D

DC Loading Transceiver Circuit ....................................3-75
DEVCONF.EXE .........................................................5-17
DOSCALL Function ..................................................7-2
Data Comm Configuration ..........................................7-39
Data Comm Control Functions ......................................7-48
Data Comm I/O .......................................................7-34
Data Comm I/O, Fast ...............................................7-48
Data Comm Logical/Physical Mapping .........................7-34
Data Communications Devices ....................................5-52
Data Communications, Programming .............................7-34
Data Transactions ...................................................3-14
Datacomm ...............................................................3-21
Datacomm Baud Rate Generation ..................................4-13
Datacomm Clock Source Select ..................................4-12
Datacomm Controller (7201/8274) .............................4-14
Datacomm Port 1 Control Lines ..................................4-12
Datacomm Port 2 Control Lines ..................................4-12
Datacomm Subsystem ...............................................3-50
Date of Last Write ..................................................5-9
Decoding ..............................................................3-63
Define Area Fill Pattern, AGIOS .................................8-30
Define Area, AGIOS ................................................8-5
Define Enhancements, AGIOS ......................................8-13
Define Key Characteristics, AGIOS .........................8-21
Define Line Pattern and Scale, AGIOS .....................8-30
Define Softkey Field, AGIOS ......................................8-18
Define Touch Field, AGIOS ........................................8-17
Define User Character Set, AGIOS ............................8-37
Delete Touch Field, AGIOS ........................................8-18
Device Command, Config.Sys ....................................5-55
Device Configuration Utility .....................................5-17
Device Driver .........................................................5-18
# Index

Device Driver Creation ........................................... 5-26  
Device Driver Example ........................................... 5-39  
Device Driver Functions ......................................... 5-31  
Device Driver Header ............................................. 5-40  
Device Driver Installation ....................................... 5-55  
Device Driver Parameters ......................................... 5-31  
Device Driver Structure ......................................... 5-21  
Device Functions ................................................. 5-31  
Device Header ...................................................... 5-21  
Device List .......................................................... 5-24, 5-27  
Device Mapping ....................................................... 5-16  
Devices ................................................................. 5-16  
Digital Logic Boards ............................................... 2-6  
Directory ............................................................... 5-8  
Directory, Disc ........................................................ 5-63  
Disc Boot Sector ..................................................... 5-61  
Disc Buffer Cache .................................................... 5-7  
Disc Clusters .......................................................... 5-62  
Disc Data Area ........................................................ 5-58  
Disc Device ............................................................ 5-53  
Disc Directory ........................................................ 5-58, 5-63  
Disc Directory Structure .......................................... 5-57  
Disc Format ............................................................. 5-57  
Disc Header Record .................................................. 5-59  
Disc Sector Buffers ................................................. 5-55  
Disc Sizes ............................................................... 5-57  
Disc Storage Capacity ............................................. 5-57  
Disk Transfer Address .............................................. 5-14  
Dispatch Table, Device Command ................................ 5-41  
Display ................................................................. 1-2  
Display Attributes .................................................... 7-10  
Display Character Codes .......................................... 7-10  
Display Control Functions, AGIOS .............................. 8-24  
Display Enhancements ............................................. 1-2, 7-10  
Display Interfacing .................................................. 7-8  
Display Memory Organization ..................................... 7-8  
Display ON/OFF, AGIOS ............................................ 8-24  
Display Row Pointers .............................................. 7-9  
Display Softkey Label, AGIOS .................................... 8-10  
Display Writes, Fast ................................................ 7-13  
Displaying Alpha Characters ...................................... 3-27  
Drawings ............................................................... 3-80  
Driver Number ...................................................... 5-9  
Dynamic RAM .......................................................... 3-62  
Dynamic RAM Refresh .............................................. 3-63

**E**

ESD protection ....................................................... 3-38  
Electrical Design ................................................... 3-73  
Electrical Interface ............................................... 3-37  
Enhance Area, AGIOS .............................................. 8-6  
Enhancement Code, AGIOS .......................................... 8-4  
Enhancement Definition, AGIOS .................................. 8-13
<table>
<thead>
<tr>
<th>Index</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enhancements, Display</td>
<td>7-10</td>
</tr>
<tr>
<td>Environmental Conditions</td>
<td>2-2</td>
</tr>
<tr>
<td>Escape Sequence, AG10S</td>
<td>8-11</td>
</tr>
<tr>
<td>Escape Sequences</td>
<td>7-1</td>
</tr>
<tr>
<td>Extension, FCB</td>
<td>5-9</td>
</tr>
<tr>
<td>Extractor</td>
<td>3-78</td>
</tr>
<tr>
<td>Extractor Retainer Pin</td>
<td>3-78</td>
</tr>
</tbody>
</table>

### F

<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAT</td>
<td>5-62</td>
</tr>
<tr>
<td>FCB</td>
<td>5-8</td>
</tr>
<tr>
<td>Field Operations, AG10S</td>
<td>8-16</td>
</tr>
<tr>
<td>File Allocation Table</td>
<td>5-55, 5-58, 5-62</td>
</tr>
<tr>
<td>File Control Blocks</td>
<td>5-8</td>
</tr>
<tr>
<td>File Manager</td>
<td>5-2</td>
</tr>
<tr>
<td>File Size</td>
<td>5-9</td>
</tr>
<tr>
<td>Filename</td>
<td>5-9, 5-64</td>
</tr>
<tr>
<td>Filename Extension</td>
<td>5-64</td>
</tr>
<tr>
<td>Files Command, Config.Sys</td>
<td>5-55</td>
</tr>
<tr>
<td>Files, Number of Open</td>
<td>5-55</td>
</tr>
<tr>
<td>Fill Rectangular Area, Absolute, AG10S</td>
<td>8-31</td>
</tr>
<tr>
<td>Fill Rectangular Area, Relocatable, AG10S</td>
<td>8-31</td>
</tr>
<tr>
<td>Firmware</td>
<td>2</td>
</tr>
<tr>
<td>Firmware Calls</td>
<td>6-1</td>
</tr>
<tr>
<td>Firmware ES Register</td>
<td>7-67</td>
</tr>
<tr>
<td>Firmware Entry Points</td>
<td>6-1</td>
</tr>
<tr>
<td>Firmware Jump Vectors</td>
<td>6-1</td>
</tr>
<tr>
<td>Firmware RAM</td>
<td>6-1</td>
</tr>
<tr>
<td>Firmware Variables</td>
<td>5-7</td>
</tr>
<tr>
<td>Firmware Yielding</td>
<td>7-27</td>
</tr>
<tr>
<td>Flush Device Function</td>
<td>5-38</td>
</tr>
<tr>
<td>Flushing Console Buffer</td>
<td>7-1</td>
</tr>
<tr>
<td>Flushing Keyboard Buffer</td>
<td>7-26</td>
</tr>
<tr>
<td>Front Plane</td>
<td>3-1</td>
</tr>
<tr>
<td>Front Plane Connectors and Signals</td>
<td>3-2</td>
</tr>
<tr>
<td>Front Plane PCA</td>
<td>2-7</td>
</tr>
<tr>
<td>Front Plane PCA Connector Layout</td>
<td>3-2</td>
</tr>
<tr>
<td>Front Plane Signal Description</td>
<td>3-7</td>
</tr>
<tr>
<td>Frontplane Connectors</td>
<td>3-77</td>
</tr>
<tr>
<td>Further Reference Documents</td>
<td>B-1</td>
</tr>
</tbody>
</table>

### G

<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>GO Generator</td>
<td>3-16</td>
</tr>
<tr>
<td>General Description</td>
<td>2-1</td>
</tr>
<tr>
<td>General Description, Datacomm Subsystem</td>
<td>3-50</td>
</tr>
<tr>
<td>General Schematic Discussion</td>
<td>3-73</td>
</tr>
<tr>
<td>Get Key Characteristics, AG10S</td>
<td>8-21</td>
</tr>
<tr>
<td>Graphics Area Fill</td>
<td>8-30</td>
</tr>
<tr>
<td>Graphics Bit Map</td>
<td>7-18</td>
</tr>
<tr>
<td>Graphics Cursor Moves, AG10S</td>
<td>8-26</td>
</tr>
<tr>
<td>Graphics Cursor ON/OFF, AG10S</td>
<td>8-25</td>
</tr>
<tr>
<td>Graphics Display</td>
<td>1-2, 3-31</td>
</tr>
</tbody>
</table>
Index

Graphics Display Interfacing .......................................................... 7-18
Graphics Display ON/OFF, AGIOS .................................................. 8-24
Graphics Hard Reset, AGIOS ........................................................... 8-39
Graphics Line Patterns .................................................................... 8-30
Graphics Line Type, AGIOS .............................................................. 8-29
Graphics Memory Clear, AGIOS ....................................................... 8-24
Graphics Memory Set, AGIOS ........................................................... 8-24
Graphics Plotting Functions, AGIOS ................................................ 8-40
Graphics RAM ................................................................................. 7-18
Graphics Status, AGIOS ................................................................. 8-46
Graphics Tablet Interface ............................................................... 7-58
Graphics Text Label, AGIOS ........................................................... 8-36
Graphics Text Mode ......................................................................... 8-28
Graphics Text, AGIOS ...................................................................... 8-34

H

HP 150 Input/Output Map ................................................................. 4-9
HP 150 Printed Circuit Assemblies .................................................... 2-5
HP 150 System Overview .................................................................. 1-1
HP-IB Controller and Interface ....................................................... 3-21
HP-IB Port ......................................................................................... 1-3
HP-IB Controller (9914) ................................................................. 4-14
HP-IB Driver ..................................................................................... 7-52
HP-IB Interfacing .............................................................................. 7-52
HP-IB Printer, Device ........................................................................ 5-53
HP-IB Programming ......................................................................... 7-52
HP-IB Templates ............................................................................... 7-54
HPBDEV Device ................................................................................. 5-16, 5-52, 5-54
Hardware Overview .......................................................................... 2-1
Header Record ................................................................................... 5-58
Header Record, Disc ......................................................................... 5-59
Height Restriction ............................................................................ 3-76
Height Restriction (Accessory Card Clearances) ................................ 3-80
Helpful Design Hints ........................................................................ 3-73, 3-76
Hold State .......................................................................................... 3-14

I

I/O Bus Device Block Diagram ......................................................... 3-20
I/O Control For Devices ..................................................................... 5-49
I/O Decoding ...................................................................................... 3-20
I/O Devices ........................................................................................ 3-19
I/O Mapped Devices .......................................................................... 5-49
I/O Panel ............................................................................................. 3-80
I/O Panel Design ................................................................................. 3-76
I/O Panel Pain Specifications ............................................................ 3-78
ID BYTE ............................................................................................ 3-73
INT Device ........................................................................................ 5-16
IO.SYS ............................................................................................... 5-3
Init Device Function .......................................................................... 5-31
Initialization ....................................................................................... 3-45
Initialization, System ......................................................................... 5-55
Installable Device ............................................................................... 5-24
Index

Installable DEvice Command Routines ........................................ 5-46
Installable Device Driver ....................................................... 5-39
Installable Device Drivers ..................................................... 5-10
Installable Devices ................................................................... 5-18
Integral Printer Connector ....................................................... 3-7
Integral Printer Device ............................................................. 5-53
Integral Printer Interface ......................................................... 4-11
Interface Description ................................................................ 3-41
Interrupt Controller ................................................................... 3-18
Interrupt Controller (8259A) .................................................... 4-13
Interrupt Routine, Device ......................................................... 5-23, 5-44
Interrupt Type Code ................................................................... 5-5
Interrupt Vectors ....................................................................... 5-5
Interrupts, Hardware .................................................................. 5-7
Interrupts, MS-DOS .................................................................... 5-7
Introduction ................................................................................. 1-1

K

Key Characteristics ..................................................................... 7-24
Key Put, AGIOS ......................................................................... 8-23
Keyboard ..................................................................................... 1-3, 3-37
Keyboard Buffer .......................................................................... 7-26
Keyboard Characteristics, AGIOS ............................................. 8-21
Keyboard Driver ......................................................................... 7-25
Keyboard Intercept, AGIOS ....................................................... 8-20
Keyboard Interfacing .................................................................. 7-22
Keyboard Key Codes ................................................................... 7-29
Keyboard Operation .................................................................... 3-37
Keyboard PCA Port ..................................................................... A-14
Keyboard PCA ............................................................................ 2-7, A-3
Keyboard Status .......................................................................... 7-26
Keyboard and Touchscreen Controller ....................................... 3-20
Keyboard and Touchscreen Data Input (I/O Port 0018H) .......... 3-49
Keyboard and Touchscreen Subsystem ..................................... 3-37
Keyboard/Touchscreen Controller (8041A) ...................................... 4-11
Keycode Mode ............................................................................ 7-22, 8-20, 8-22
Keycode ON/OFF, AGIOS ....................................................... 8-22
Keycode Status, AGIOS ............................................................. 8-23
Keycodes .................................................................................... 7-22, 7-29

L

L-Bracket .................................................................................... 3-78, 3-80
LED Decoding ............................................................................. 3-62
LED Register Reset ..................................................................... 3-62
LEDs .......................................................................................... 3-62
LPT1 Device ................................................................................ 5-16
LPT2 Device ................................................................................ 5-16
LPT3 Device ................................................................................ 5-16
LST Device .................................................................................. 5-16
Language PCA ............................................................................ A-29
Lift Boundary Pen, AGIOS ............................................................. 8-44
Lift Pen, AGIOS .......................................................................... 8-40
Index

List of Vendors ............................................................. 3-77
Logic Diagrams ............................................................. A-1
Logical Devices ............................................................. 5-16
Lower Boundary Pen, AGIOS ................................................. 8-45
Lower Pen, AGIOS ............................................................. 8-41

M

MPSC ................................................................................... 4-14
MS-DOS ............................................................................... 5-1
MS-DOS Calls ................................................................. 7-2
MSDOS.SYS. ...................................................................... 5-3
Manuf Test Repeat ............................................................. 4-12
Maxalloc .............................................................................. 5-11
Mechanical Description ......................................................... 3-38
Mechanical Design ............................................................. 3-36
Mechanical Specifications ..................................................... 3-64
Media Check Device Function .............................................. 5-32
Media Descriptor Byte ......................................................... 5-30, 5-34
Memory Map ........................................................................... 4-1
Memory Map, Firmware ....................................................... 6-1
Memory and I/O Mappings ................................................... 3-11
Mezzanine Datacomm PCA .................................................... A-23
Mezzanine Memory Connector Signal ...................................... 3-57
Mezzanine Memory PCA Block Diagram .................................. 3-56
Mezzanine Memory PCA ......................................................... 2-7, 3-55, A-21
Mezzanine Memory Subsystem ............................................... 3-55
Microprocessor System Architecture ....................................... 3-13
Minalloc ............................................................................... 5-11
Modem Disconnect .............................................................. 7-49
Monitor Data Comm Mode ..................................................... 7-49
Move Graphics Cursor Absolute, AGIOS ................................ 8-26
Move Graphics Cursor Incremental, AGIOS ............................. 8-27
Multi-Protocol Controller ..................................................... 3-52

N

NSLOTSEL ............................................................................. 3-73
Name Field, Device ............................................................ 5-23
No Polygon Boundary, AGIOS ............................................. 8-32
Non Destructive Read No Wait Device Function ..................... 5-36

O

Operating System Disc .......................................................... 5-61
Operating System Memory Usage ........................................... 5-4
Operating System Structure .................................................. 5-1
Operating System, MS-DOS 2.0 ............................................ 1-1
Output Single Text Character, AGIOS ................................... 8-37
Overview ............................................................................. 3-10

P

I-8
<table>
<thead>
<tr>
<th>Index</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAMCODE.EXE</td>
<td>5-10</td>
</tr>
<tr>
<td>PCA Configuration</td>
<td>3-63</td>
</tr>
<tr>
<td>PCA Overview</td>
<td>3-55</td>
</tr>
<tr>
<td>PLT Device</td>
<td>5-16</td>
</tr>
<tr>
<td>PRN Device</td>
<td>5-16</td>
</tr>
<tr>
<td>Pair Address to Row, Column No. Conversion</td>
<td>3-42</td>
</tr>
<tr>
<td>Personal Applications Manager (P.A.M)</td>
<td>1-1, 5-2</td>
</tr>
<tr>
<td>Physical Devices</td>
<td>5-16</td>
</tr>
<tr>
<td>Physical Disc Format</td>
<td>5-57</td>
</tr>
<tr>
<td>Physical Specifications</td>
<td>2-2</td>
</tr>
<tr>
<td>Plot to Cursor Position, AGIOS</td>
<td>8-42</td>
</tr>
<tr>
<td>Plotter Device</td>
<td>5-53</td>
</tr>
<tr>
<td>Point Plot, AGIOS</td>
<td>8-43</td>
</tr>
<tr>
<td>Polygon Draw, AGIOS</td>
<td>8-44</td>
</tr>
<tr>
<td>Polygon Move, AGIOS</td>
<td>8-43</td>
</tr>
<tr>
<td>Position Cursor, AGIOS</td>
<td>8-12</td>
</tr>
<tr>
<td>Power Requirements</td>
<td>2-3, 3-64</td>
</tr>
<tr>
<td>Power Supply Connector</td>
<td>3-7</td>
</tr>
<tr>
<td>Power Supply PCA</td>
<td>2-6</td>
</tr>
<tr>
<td>Processor Board Block Diagram</td>
<td>3-11, 3-13</td>
</tr>
<tr>
<td>Processor Board and Memory Board Block Diagram</td>
<td>3-10</td>
</tr>
<tr>
<td>Processor Front Plane PCA</td>
<td>A-7</td>
</tr>
<tr>
<td>Processor I/O Bus</td>
<td>A-9</td>
</tr>
<tr>
<td>Processor PCA</td>
<td>A-5</td>
</tr>
<tr>
<td>Processor Subsystem</td>
<td>3-10</td>
</tr>
<tr>
<td>Processor System Overview</td>
<td>3-10</td>
</tr>
<tr>
<td>Product Regulations</td>
<td>2-3</td>
</tr>
<tr>
<td>Product Specifications</td>
<td>2-1</td>
</tr>
<tr>
<td>Program Receiving Control</td>
<td>5-13</td>
</tr>
<tr>
<td>Program Segment Prefix (PSP) Control Block</td>
<td>5-11</td>
</tr>
<tr>
<td>Program Segment</td>
<td>5-11</td>
</tr>
<tr>
<td>Program Termination</td>
<td>5-13</td>
</tr>
<tr>
<td>Programming Application Softkeys</td>
<td>8-9</td>
</tr>
<tr>
<td>Programming the HP 150</td>
<td>2</td>
</tr>
<tr>
<td>Put Key, AGIOS</td>
<td>8-22</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM (Memory Extender) PCA</td>
<td>A-27</td>
</tr>
<tr>
<td>RAM Expansion</td>
<td>4-2</td>
</tr>
<tr>
<td>ROM</td>
<td>3-59</td>
</tr>
<tr>
<td>ROM Decoding</td>
<td>3-59</td>
</tr>
<tr>
<td>ROM Timing</td>
<td>3-60</td>
</tr>
<tr>
<td>RS-232</td>
<td>1-3</td>
</tr>
<tr>
<td>RS-232 Communication Port</td>
<td>1-3</td>
</tr>
<tr>
<td>RS-232/RS-422 Communication Port</td>
<td>1-3</td>
</tr>
<tr>
<td>RS-232/RS-422</td>
<td>1-3</td>
</tr>
<tr>
<td>RS232C/422 Datacomm Module Connector</td>
<td>3-50</td>
</tr>
<tr>
<td>RS232C/422 Datacomm PCA</td>
<td>2-7</td>
</tr>
<tr>
<td>Raster Scan</td>
<td>3-23</td>
</tr>
<tr>
<td>Read Area, AGIOS</td>
<td>8-6</td>
</tr>
<tr>
<td>Read Area Shading, AGIOS</td>
<td>8-51</td>
</tr>
<tr>
<td>Read Cursor Position, AGIOS</td>
<td>8-47</td>
</tr>
<tr>
<td>Read Cursor Position, Wait For Key, AGIOS</td>
<td>8-47</td>
</tr>
</tbody>
</table>
Index

Read Cursor Type, AGIOS ............................................. 8-14
Read Device Function ................................................. 5-35
Read Device ID, AGIOS .............................................. 8-46
Read Display Size, AGIOS .............................................. 8-48
Read Dynamics, AGIOS .............................................. 8-51
Read Extended Screen Dimensions, AGIOS ......................... 8-52
Read Graphics Setting, AGIOS ....................................... 8-48
Read Graphics Text Status, AGIOS ................................ 8-49
Read Keypad Status, AGIOS ........................................... 8-23
Read Pen Position, AGIOS ............................................. 8-46
Read Relocatable Origin, AGIOS .................................... 8-50
Read Reset Status, AGIOS ............................................. 8-50
Read Softkey Label, AGIOS ............................................. 8-9
Read Terminal Configuration, AGIOS ................................ 8-14
Read Zoom Status, AGIOS ............................................. 8-49
Real Time Clock ..................................................... 3-21
Real Time Clock (MM58167A) ......................................... 4-10
Record Size ............................................................ 5-9
Relative Record ..................................................... 5-10
Request Header ....................................................... 5-25
Request Header, Device ............................................ 5-42
Reset Logic ............................................................ 3-13
Resolution Versus Number of Pairs ................................ 3-42
Retainer Thumbscrew ................................................ 3-80
Ring Retainer .......................................................... 3-77
Row Column Operations, AGIOS ...................................... 8-16
Row Pointer Table ..................................................... 7-9
Rubber Band Line, AGIOS .............................................. 8-26

S

SHOLDA Timing ....................................................... 3-69
SYSINIT Routine ..................................................... 5-55
Screen Memory Organization ....................................... 7-8
Screen Writes, Fast .................................................. 7-13
Sector Allocation, Disc ........................................... 5-58
Sectors, Disc .......................................................... 5-57
Select Boundary Pen, AGIOS ........................................ 8-32
Select Default Character Set, AGIOS ................................ 8-37
Select Drawing Mode, AGIOS ....................................... 8-29
Select Line Type, AGIOS .............................................. 8-29
Select Polygonal Fill Pattern, AGIOS ............................... 8-31
Set Cursor Type, AGIOS .............................................. 8-14
Set Graphics Default, AGIOS ....................................... 8-38
Set Graphics Memory, AGIOS ....................................... 8-24
Set Graphics Text Origin, AGIOS .................................... 8-36
Set Graphics Test Orientation, AGIOS .............................. 8-34
Set Graphics Text Size, AGIOS ..................................... 8-34
Set Picture Definition Defaults, AGIOS ........................... 8-39
Set Relocatable Origin to Cursor Position, AGIOS ............... 8-33
Set Relocatable Origin to Pen Position, AGIOS ................. 8-33, 8-42
Set Relocatable Origin, AGIOS ..................................... 8-33
Set Touch Reporting Modes, AGIOS ................................ 8-19
Shell Command, Config.Sys ........................................ 5-56
Shift Area, AGIOS .......................................................... 8-7
Signal Timing Diagrams ................................................... 3-67
Slot Select, Accessory .................................................... 7-65
Slot Selection Generation ............................................... 3-60
Soft Key Direct Display Writes ....................................... 7-15
Softkey Label Display, AGIOS ......................................... 8-10
Softkey Label Read, AGIOS ............................................. 8-9
Softkey Label Update, AGIOS .......................................... 8-9
Softkey Touch Field Definition, AGIOS .............................. 8-18
Softkeys, Programming .................................................. 8-9
Specifications .............................................................. 3-40
Start Polygonal Area Fill, AGIOS ..................................... 8-43
Static Request Header ................................................... 5-27
Status Device Function .................................................. 5-37
Status Generation ....................................................... 3-14
Status Line, Writing To .................................................. 7-15
Status Register (I/O Port 0019H) ...................................... 3-45
Status Word ................................................................. 5-28
Strategy Routine, Device ................................................. 5-23, 5-44
Sub-Directories ............................................................ 5-64
Subsystem Power Requirements ......................................... 2-4
Sweep Board Connector ................................................... 3-6
Sweep Board ................................................................. 3-23
Sweep PCA .................................................................... 2-6, A-25
Switchar Command, Config.Sys ........................................ 5-55
System Architecture ...................................................... 5-1
System Booting ............................................................. 5-55
System Function Calls .................................................... 5-3, 7-2
System Initialization ..................................................... 5-61
System Timing and Control Logic ...................................... 3-14

T

TPM .......................................................... 3-1
Technology and Display Format ....................................... 3-23
Templates, HPIB .......................................................... 7-54
Terminal Configuration, Reading ...................................... 8-14
Terminal Emulator ........................................................ 7-37
Terminate Polygonal Area Fill, AGIOS ............................... 8-43
Test Strap Logic .......................................................... 3-13
Text Mode, Graphics .................................................... 8-28
Thermal (Integral) Printer Interface ................................. 3-1
Thermal Limits ............................................................. 3-65
Thermal Printer Interface ................................................. A-19
Thumbscrew ................................................................. 3-77
Thumbscrews ............................................................... 3-76
Time Device ................................................................. 5-52
Time of Last Write ........................................................ 5-10
Touch Field Definition, AGIOS ....................................... 8-17
Touch Field Deletion, AGIOS .......................................... 8-18
Touch Reporting Modes, AGIOS ........................................ 8-19
Touch Screen ASCII Fields ............................................. 8-15
Touch Screen Functions, AGIOS ....................................... 8-15
Touch Screen keycode Fields .......................................... 8-15
<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Touch Screen Normal Fields</td>
</tr>
<tr>
<td>Touch Screen Reset, AGIOS</td>
</tr>
<tr>
<td>Touch Screen Row Column Operations</td>
</tr>
<tr>
<td>Touch Screen Toggle Fields</td>
</tr>
<tr>
<td>Touchscreen</td>
</tr>
<tr>
<td>Touchscreen Block Diagram</td>
</tr>
<tr>
<td>Touchscreen Connector</td>
</tr>
<tr>
<td>Touchscreen PCA</td>
</tr>
<tr>
<td>Touchscreen PCA</td>
</tr>
<tr>
<td>Tracks, Disc</td>
</tr>
<tr>
<td>Tranceiver Schematic</td>
</tr>
<tr>
<td>Tranceiver Schematic Discussion</td>
</tr>
<tr>
<td>Transparent Data Comm Mode</td>
</tr>
<tr>
<td>Turn Off Alphanumeric Cursor, AGIOS</td>
</tr>
<tr>
<td>Turn Off Alphanumeric Display, AGIOS</td>
</tr>
<tr>
<td>Turn Off Graphics Text Mode, AGIOS</td>
</tr>
<tr>
<td>Turn Off Rubber Band Line, AGIOS</td>
</tr>
<tr>
<td>Turn Off Text Slant, AGIOS</td>
</tr>
<tr>
<td>Turn On Alphanumeric Cursor, AGIOS</td>
</tr>
<tr>
<td>Turn On Alphanumerics Display, AGIOS</td>
</tr>
<tr>
<td>Turn On Graphics Cursor, AGIOS</td>
</tr>
<tr>
<td>Turn On Graphics Text Mode, AGIOS</td>
</tr>
<tr>
<td>Turn On Rubber Band Line, AGIOS</td>
</tr>
<tr>
<td>Turn On Text Slant, AGIOS</td>
</tr>
<tr>
<td>Turn off Graphics Display, AGIOS</td>
</tr>
<tr>
<td>Turn on Graphics Display, AGIOS</td>
</tr>
<tr>
<td>Unit Code</td>
</tr>
<tr>
<td>Update Softkey Label, AGIOS</td>
</tr>
<tr>
<td>Vector Draw, AGIOS</td>
</tr>
<tr>
<td>Vector Drawing Mode, AGIOS</td>
</tr>
<tr>
<td>Vector Move, AGIOS</td>
</tr>
<tr>
<td>Video Alpha Display Subsystem</td>
</tr>
<tr>
<td>Video Alpha RAM Subsystem</td>
</tr>
<tr>
<td>Video Area Clear, AGIOS</td>
</tr>
<tr>
<td>Video Area Enhance, AGIOS</td>
</tr>
<tr>
<td>Video Area Read, AGIOS</td>
</tr>
<tr>
<td>Video Area Shift, AGIOS</td>
</tr>
<tr>
<td>Video Area Write, AGIOS</td>
</tr>
<tr>
<td>Video Attribute Latch</td>
</tr>
<tr>
<td>Video Board</td>
</tr>
<tr>
<td>Video Board Overview</td>
</tr>
<tr>
<td>Video Define, AGIOS</td>
</tr>
<tr>
<td>Video Frame Format</td>
</tr>
<tr>
<td>Video Graphics Display Subsystem</td>
</tr>
<tr>
<td>Video Intrinsics, AGIOS</td>
</tr>
<tr>
<td>Video Line Write, AGIOS</td>
</tr>
<tr>
<td>Video Memory Organization</td>
</tr>
<tr>
<td>Video PCA</td>
</tr>
</tbody>
</table>
Index

Video Row Pointers ................................................................. 7-9
Video Subsystem ........................................................................... 3-23
Video Writes, Fast ........................................................................ 7-13

W

Wait State Disable ................................................................. 3-59
Wait State Generation .............................................................. 3-15
Write Area, AGIOS .................................................................... 8-5
Write Device Function ............................................................... 5-35
Write Line, AGIOS ..................................................................... 8-8

Y

Yielding To Firmware .............................................................. 7-27
We welcome your evaluation of this manual. Your comments and suggestions help us to improve our publications. Please use additional pages if necessary.

Is this manual technically accurate? Yes [ ] No [ ] (If no, explain under Comments, below.)

Are the concepts and wording easy to understand? Yes [ ] No [ ] (If no, explain under Comments, below.)

Is the format of this manual convenient in size, arrangement and readability? Yes [ ] No [ ] (If no, explain under Comments, below.)

Comments:

Date:__________________________

FROM:

Name__________________________Title__________________________

Company____________________________________________________

Address_____________________________________________________

City__________________________State______Zip____________________
Hewlett-Packard Company  
Personal Office Computer Division  
P.O. Box 486  
974 E. Arques Avenue  
Sunnyvale, CA 94086  
Attn: Technical Support Manager