Private Eye® Display Controller Programmer's Manual

April 22, 1991

1. Overview

The purpose of this document is to help you to write software for the Private Eye Display Controller.

This document describes Reflection Technology Inc.'s

Part No. 213-008-00

labeled: "Aox VCGA 06205-001"
with the two bug-fix PALs implemented. These PALs are installed on all PEVA boards shipped by Reflection Technology. The PALs contain circuitry that is required in order to support reliable operation of the Private Eye display system. Infcmnation on these bug-fix PALs are available upon request from Reflection Technology.

The software interface to the PEDC is through two mechanisms: the display memory, and the configuration registers. The PEDC displays the contents of the display memory in the Private Eye, in a format governed by the contents of the configuration registers

The PEDC is designed to emulate an IBM CGA display controller as closely as possible. Applications that use IBM text modes and CGA graphics modes (40 and 80 column) will run without modification.

For graphics mode, the CGA graphics modes (640 x 200 and 320 x 200) are supported, but it is recommended that new applications use the 720 x 280 bitmap mode.

The display memory (or *frame buffer) of* the PEDC is, by default, mapped to b800 :0000 (or dOOO :0000 if the PCMODE jumper is set to 2-3)on RTI's PEVA card for the IBM-PC bus. For other machines, see the machine's documentation for the base address of the frame buffer.

Note that the Private Eye Display Controller, unlike the standard IBM CGA display controller, does not cause the display to flicker or flash if the processor updates the display memory during the drawing time.

For register accessing instructions, register numbers, and register bit definitions, see Appendix A in the back of this manual. For electrical and physical data on the PEDC, see the Private Eye Display Controller Specification.

2. Modes

2.1 Text Modes

In Text Mode, the screen has 25 lines of either 40 or 80 characters of text. Each displayed character has its own unique set of attributes, specified in eight bits. There is a flashing cursor that can be positioned on any displayed character position. The shape of the cursor can be changed from software. There are 256 displayable characters; 128 are standard ASCII; the rest are IBM graphics characters. The standard graphics characters contain symbols used to create text windows and boxes.

2.1.1 Display Memory Format

The display memory contains 2000 (for 40 column mode) or 4000 (for 80 column mode) displayed bytes. Each character position on the screen is represented by two bytes (one word) of display memory. The top left screen location is the first one in memory, with the next character on the line coming next. The first character on the second line comes immediately after the last character on the first line.

There is sufficient display memory in Text Mode for 4 (80 column) or 8 (40 column) screens of text. With the Start Address Register you can specify the starting address of the displayed memory area. The value written to the Start Address Registers represents the *word* offset between the display memory base address and the address of the top left corner of the screen. This technique can be used to pre-format up to eight text screens; each can be instantly displayed by simply changing the Start Address Registers.

There is sample code in Appendix B of this manual which shows how to write characters on the screen in Text Mode.

2.1.2 Characters and Attributes

Each character to be displayed on the screen is represented by two bytes of frame buffer memory: the low-order byte contains the ASCII value of the character to be displayed, and the high-order byte contains the attribute byte for the character. The following are good values to use for particular character attributes:

	Appearance
0x00	All black
0x01	Underline
0x07	Normal
0x09	Bold Underline
0x0f	Bold
0x70	Reverse
0x77	Alired
0x78	Reverse Bold
0x81	Flashing Underline
0x87	Flashing
0x89	Bold Flashing Underline
0x8f	Flashing Bold
0x00	Reverse Flashing
0xf 7	Flashing Red
0xf 8	Reverse Flashing Bold

Text Mode Character Attributes

2.1.3 Cursor

The cursor is a reverse-video block, which appears superimposed over the desired character on the display. It is easy to move the cursor from one place to another on the screen, and to change its appearance. The cursor is displayed as a flashing block, on top of the character where the cursor is positioned. The block extends from the left edge to the right edge of the character.

The lower five bits of the Cursor Start Register set the scan line within the character of the top of the block. The lower five bits of the Cursor End Register set the scan line of the bottom of the block.

The cursor flashes unless bits 6 and 5 of the Cursor Start Register are 01, in which case the cursor is not displayed. To make the cursor not visible, you can also either move it to a location not displayed on the screen, or set the Cursor Start Register to a value greater than the Cursor End Register.

The Cursor Location Registers are used in Text Mode to control the position of the cursor. The 16-bit number is the offset of the cursor from the beginning of the displayed memory *(not* the beginning of the screen; see the discussion about the Start Address Registers in Section 2.1.1). For example, the cursor will be at the top left corner of the screen when the value of the Cursor Location Register is the same as the value of the Start Address Register.

2.1.4 Initialization

This is the initialization sequence for Text Mode. Flashing and underlining are enabled. *Please note that the initialization routines in this manual work only for the Private Eye Display Controller.* Other CGA display controllers have other registers that need to be initialized. See the documentation for your particular display controller for details.

Sample code for initializing the Private Eye Display Controller is contained in Appendix B of this manual. Note that since most register accesses appear in pairs, the sequences are shown in two columns; read across.

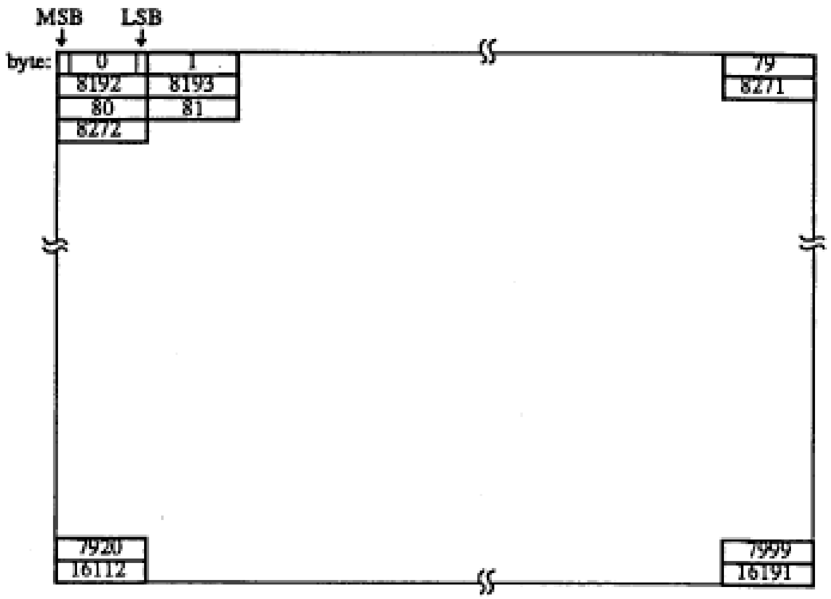
register	value	register	value	
0x04	0x1e	0x05	OxlO	SetUnderline Enable bit
0x04	0x0a	0x05	OxOO	Cursor start is 0
0x04	OxOb	0x05	0x07	Cursor End is 7
0x04	OxOc	0x05	OxQO	Start Address is O
0x04	OxOd	0x05	OxOO	
0x04	OxOe	0x05	OxOO	Cursor Location is O
0x04	OxOf	0x05	0x00	
0x08	OxOO	0x08	0x28	(4Ocolumns)OR
			0x29	(80 columns)

2.2 CGA Graphics Modes

The CGA graphics modes are not recommended for new software, because of their low resolution and their difficult pixel addressing scheme. This mode is displayed on the center 640 x 200 pixels of the Private Eye's 720 x 280 screen.

The 320 x 200 mode nominally supports four colors. A simple pixel substitution algorithm is used to simulate gray scale patterns for this mode.

2.2.1 Display Memory Format



CGA Graphics Mode Display Format

2.2.2 Initialization

```
      0x04  0x1e  0x05  0x00  ClearExtendedModeRegister
      0x04  0x0c  0x05  OxOO  clearStartAddressRegister
      0x04  0x0d  0x05  0x00
      0x08  0x00  0x08  0x0a  (320x200)OR
                        0x1a  (640x 200)
```

2.3 Extended Graphics Mode

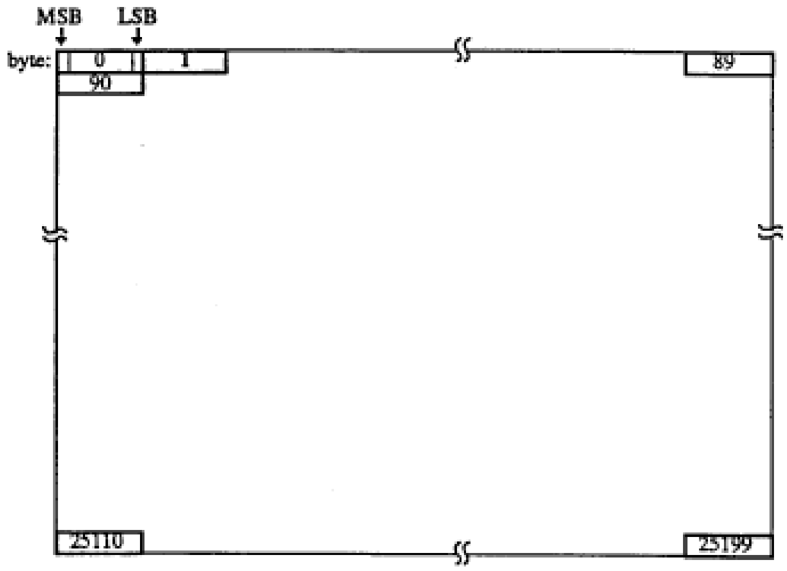
Extended Graphics Mode shows a 720h x 280v, horizontally oriented, non-interlaced bitmap. It displays the first 25200 bytes of the frame buffer memory on the full area of the Private Eye display. The Start Address Registers must be preset to a certain value, and cannot be used in the same manner as in Text Mode. Double buffering is not supported.

Each bit in the display memory corresponds to one pixel on the Private Eye display. A ‘1’ bit is displayed as a red (illuminated) pixel, and a ‘0’ bit is displayed as a black (dark) pixel. Example code for setting pixels on the display in Extended Graphics Mode is in the Appendix of this manual.

MS-DOS users: if you use this mode, you must reset the Extended Graphics Mode bit in the Extended Mode Register when your program exits; otherwise the MS-DOS BIOS will not be able to configure the display for Text Mode.
The PEDC starts to update the Private Eye’s entire display memory when the processor writes to the display memory.

2.3.1 Display Memory Format

The display memory format for Extended Graphics Mode is that of a traditional raster display; see the figure below.



Graphics Mode Screen Format

2.3.2 Initialization

0x04	0x1e	0x05	OxOO	Clear Extended Mode Register
0x04	0x0c	0x05	0x21	SetStartAddressRegister
0x04	0x0d	0x05	0x98	
0x08	0x00	0x08	0x1e	Set Mode Control Register
0x04	0x1e	0x05	0x01	Set Extended Mode Register

3. Options

When configuring the Private Eye display, is is often necessary to change one bit in a write-only register. To do this, one must maintain a copy of the register as a variable in your software, and write it to the display controller when it is changed. The examples below all use this mechanism.

3.1 Right / Left Eye

Private Eye systems should have a method for inverting the screen image, so that left-eyed people can use them. To invert the image in the Private Eye display, set the Left Eye Enable bit (0x80) in the RTSI Command Register shadow variable, then write the variable to the RTSI Command Register.

```
            register value
      0x04 0x1f    0x05  newRTSJ CommandRegister value
```

3.2 Dim Mode

Private Eye systems should have a method for dimming the screen image, so that the Private Eye can be used in dark environments without destroying the user's night vision. To dim the image in the Private Eye display, set the Dim Enable bit 0x10 in the RTSI Command Register.

```
      register value  register      value
      0x04 0x1f       0x05    new RTSI Command Register value
```

3.3 Power-Down Mode

To put the Private Eye in its lowest power mode, set the Standby Mode Enable bit (0x40) in the RTSI Command Register. In this mode, the Private Eye's mirror stops moving, soit will take a couple of seconds for the mirror to start up before the image is visible again after resetting the Standby Mode Enable bit.

```
register value register value
0x04 0x1f      0x05  new RTSJ Command Register value
```

3.4 Underline Mode

In some situations, it is advantageous to use the underline feature in text mode. With software that has user-definable character attributes, Underline Mode can be used to good advantage. This mode is normally turned off. The following register sequence turns it on:

```
register value     register value
OxO 4 0x1e  0x05  0x10  Set Underline Enable bit
```

3.5 Determining Presence of Private Eye

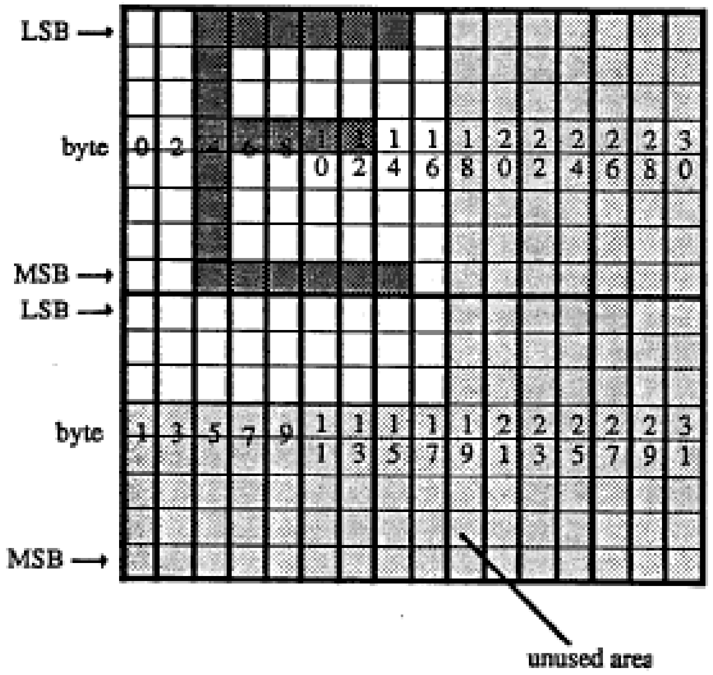
In some circumstances, it may be valuable to detect whether the Private Eye display is actually connected. For instance, you may put the computer into a low-power mode if the Private Eye is not connected, since the user cannot be looking at the display if it isn't plugged in. To detect the presence of the Private Eye, look for transitions in the RTSI Ready signal in bit 4 of the CRT Status Register.

3.6 Downloadable Character Sets

It is possible to replace the font EPROM on the PEVA card with an SRAM chip, enabling the use of downloadable fonts.

The character generator contains two fonts: the normal font, and the bold font. Each font contains 256 characters. Each character is constructed from a 9 x 11 cell contained in 32 bytes, made up of a 16 x 16 block of bits. The organization of the character in the block is shown in the figure below. The cell's bits are set to 1 for pixels that are to be illuminated, and 0 for dark pixels. The bits in the unused area are ignored, although it is recommended that they be set to 0.

For the standard Private Eye font, the bottom row and left column of the cell are blank for characters that do



not touch each other. Characters are 8 rows tall, with 2 rows used for descenders.

The normal font occupies the first **8K** bytes of the character generator, and the bold font occupies the second 8K. Each character in the font is stored in 32 consecutive bytes, and the characters are arranged in ascending order.

Character Generator Cell Format

In order to use the downloadable character set option, you must replace the 27128 EPROM on the PEVA card with a 32K x 8 static RAM, l00ns access time (Toshiba TC55257APL-l0 or equivalent). If you have an older board with two sockets, not that the EPROM character generator goes in the upper of the two sockets on the board; newer boards mily have one socket.

To download a new font, set the Character RAM Access Enable bit in the Extended Mode register:

```
register value    register  value
0x04     0x1e     0x05      0x08 Set Character RAM Access Enable bit
```

At this point, the display controller's memory space is mapped into the character generator RAM. Write the font to the display memory. When you're done, reset the Character RAM Access enable bit:

```
register value    register  value
0x04     0x1e     0x05      0x00 Reset Character RAM Access Enable bit
```

Appendix

A. Registers

Note that if the PEDC is configured for shadow mode, it is not possible to read back the Contents of the PEDC's registers. This means that the procedure for determining the presence of a Private Eye will not work if the PEDC is configured for Shadow Mode.

A.1 External Registers

The PEDC's registers are located starting at 3d0 (or 310 if the PCMODE jumper on the PEVA card is set to 2-3) of the 1./0 space in MS-DOS machines, accessible using the inportb and outportboperators provided with most development environments. To generate the address for a particular register, add the starting location to the Register Number. For non-MS-DOS implementations, see the accompanying documentation to determine how to access the PEDC's registers.

Register Name	Address	
Mode Control	0x0 8	Write-only
CRT Status	0x0A	Read-only
CRTC Address	0x04	Write-only
CRTC Data	0x05	Read/Write

A.1.1 Mode Control Register

The Write-Only Mode Control Register is used to control the PEDC's CGA-compatible modes. In order to put the PEDC into one of these modes, the HorizontalMode Enable bit in the Extended Mode Register must be off. Also note that with these modes, other registers must be set to appropriate values before the display will show the desired information.

Value	Mode
0x00	Blank Display
0x08	40 x 25 text mode, no flashing
0x09	80 x 25 text mode, no flashing
0x0a	320 x 200 CGA graphics
0x1a	640 x 200 CGA graphics
0x28	40 x *25* text mode, flashing enabled
0x2 9	80 x 25 text mode, flashing enabled

A.1.2 CRT Status Register

The Read-Only CRT Status Register contains bits indicating the status of the horizontal and vertical sync signals. In the PEDC, these bits are simulated for compatibility with the standard CGA display controller. There is also a bit that returns the active-low state of the RTSI Ready signal. The Ready signal indicates the position of the mirror in the display, and when it is allowed to transmit data to the display. This is useful for synchronizing screen updates, and for determining whether or not a Private Eye display is plugged in. See the RTSI Specification for more details on the RTSI Ready signal.

bit	Settings
7	X
6	X
5	x
4	/RTSI Ready
3	Simulated Vertical Sync
2	X
1	x
0	Simulated Horizontal Sync

A.2 CRTC Registers

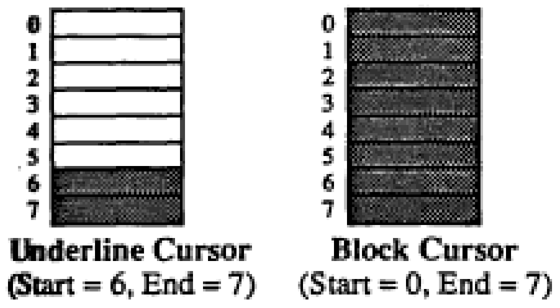
For compatibility with the CGA display controller, most of the PEDC's internal registers are accessed through two external registers: the CRTC Address Register, and the CRTC Data Register. To write a CRTC register, the register number is written to the CRTC Address Register, and the desired register contents are written to the CRTC Data Register. To read a CRTC register, write the register number to the CRTC Address Register, then read the contents of the internal register from the CRTC Data Register.

There are two extra registers in the PEDC (in italics below) that are not present in standard CGA display controllers. Most of the unique features of the Private Eye are accessible through these registers.

Cursor Start	0x0a	Write-only
Cursor End	0x0b	Write-only
Start Address High	0x0c	Write-only
Start Address Low	0x0d	Write-only
Cursor Location High	0x0e	Read/Write
Cursor Location Low	0x0f	Read/Write
Light Pen High	0x10	Read-only
Light Pen Low	0x11	Read-only
Extended Mode Register	0x1e	Read/Write
RTSI Command Register	0x1f	Read/Write

A.2.1 Cursor Start and End Registers

These registers are used to control the appearance of the cursor in text modes. In graphics modes, the contents of these registers is ignored. The cursor is a flashing area on top of the desired character, extending from the left edge to the right edge of the character. Bits 4-0 of the Cursor Start Register sets the scan line within the character of the top of the block. Bits 4-0 of the Cursor End Register sets the scan line of the bottom of the block. In addition, bits *5* and 6 of the Cursor Start Register control the appearance of the cursor: this cursor is not displayed if bit *5* is 1 and bit 6 is 0. Two common examples of cursors:



A.2.2 Start Address Registers

The Write-Only Start Address Registers are used in text mode to set the start of the active frame buffer. The contents of these registers, as a 16-bit value, is multiplied by two and added to the base address to give the actual starting address of the text mode display memory. In CGA Graphics modes, both these registers must be set to zero. In Extended Graphics Mode, these registers must be set to predefined values (see Section 2.3).

A.2.3 Cursor Location Registers

The Read/Write Cursor Location Registers are used in text mode to control the position of the cursor. They are ignored in graphics modes. The 16-bit number is the offset of the cursor from the beginning of the display memory *(not* the beginning of the screen). For example, if the Start Address Registers ate set to a non-zero value, the cursor will be at the top left corner of the screen when the value of the Cursor Location Register is the same as the value of the Start Address Register. The contents of these registers are ignored in graphics modes.

A.2.4 Light Pen Registers

The Read-Only Light Pen Registers are supported only for compatibility with standard CGA software. Since the CGA Mode Control Register is Write-Only, some MS-DOS software uses these registers to determine the current mode. This is done by waiting for the beginning of the vertical blank interval (by watching the Vertical Sync bit in the CRT Status Register) and examining the contents of the Light Pen Registers at that instant. The contents will be different, depending on the mode.

A.2.5 Extended Mode Register

The contents of the Extended Mode Register controls several aspects of the PEDC's operation that are not part of the CGA standard. All the bits in this register are active-high.

bit	Settings
7	Reserved (must be 0)
6	Continuous RTSI Update Mode Enable
5	Reserved (must be 0)
4	Underline Mode Enable
3	Character RAM (Downloadable Font) Access Enable
2	Reserved (must be 0)
1	Reserved (must be 0)
0	Horizontal Mode Enable

A.2.6 RTSI Command Register

The RTSI command register's contents are transmitted to the Private Eye when the register is written to by the host.
In Horizontal Graphics Mode, the contents of the RTSI Command Register are not reliably transmitted to the Private Eye display. Write the register in a loop, so that it eventually gets transmitted. You must determine empirically how many times to write the register so that it always gets through.

bit	Settings
7	Left Eye Mode Enable
6	Standby Mode Enable
5	Display Blank Enable
4	Low Intensity Enable
3	Reserved (must be 0)
2	Reserved (must be 0)
1	Reserved (must be 0)
0	Reserved (must be 0)

B. Sample Code
This code is written in C Everyone except those using MS-DOS ‘tiny’ or ‘small’ memory models can ignore the ‘far’ keyword.

B.1 Initialize Extended Graphics Mode

```
typedef struct regval

unsigned reg, val;
regval;

regval peinit[] = {{0x04, 0x1e), {0x05, OxOO), {0x04, OxOc), {0x05, 0x21}, {0x04, OxOd},
(0x05, 0x98}, {0x08, 0x00}, {0x08, Oxiel, (0x04, Oxie), {0x05, 0x01});

unsigned PERegs – Ox3dO;

/* initialize PEDC */
for (i = 0; i < 10; i++)
outportb(PERegs + peinit(i) .reg, peinit[i] .val);
```

B.2 Write characters in Text Mode

```
int x, y;
int cols = 80; /* maybe 40...
unsigned char far *PEFBAddr = (unsigned char far *) Oxb8000000; unsigned
char far *ptr;


X=y

/* draw the character (ch) and the attribute (attr) at (x, y) */
     ptr  =    PEFBAddr  +    (((cols   *    y)   +    x)   *    2);
*ptr++ = ch;
*ptr = attr;
```

B.3 Set (or Reset) Pixels in Extended Graphics Mode

```
int x, y;   0xb8000000;
unsigned char far *PEFBAddr = (unsigned char far *) unsigned char far *ptr;


X=y

ptr = PEFBAddr + ((90 * y) + (x I 8));
*(ptr)      j= 0x80 » (x % 8); /* turn on the pixel */
```

or

```
*(ptr) &= -(0x80 » (x % 8)); /* turn off the pixel */
```