New PERQ Tablet and Cursor Interface

Written by: Brad A. Myers

Abstract:

There is a new interface to the PERQ's tablet and cursor that provides more functionality than the old interface. The old interface has therefore been removed and clients should begin using the current interface with this release(OS B.1). Note that any old programs compiled under the old system will still operate, but they will have to be modified to compile under the new system. This document describes the new interface.

Overview:

The old interface to the PERQ's tablet and screen cursor were adequate for the first applications but were found to be too limited for more powerful systems. A new interface has therefore been designed that provides more of the basic capabilities of the hardware to the client. This document describes the new interface which REPLACES the old one. All programs compiled under the previous system will still operate, but to compile any program under the new release, the changes described here will have to be made.

All the procedures to interface to the tablet and the cursor are provided by the module IO.PAS. Below is a listing of the part of this module that deals with the tablet and cursor. Below that is an explanation of the parts that changed and the usage of the new procedures.

```
{tablet/cursor procedures }
  Type  XXXOLDCurMode = (XXXAbsCursor, XXXRelCursor, XXXUserCursor,
                         XXXOffCursor);  {*****THIS TYPE NO LONGER
                         VALID.  DO NOT USE*****}
        CursFunction = (CTWhite, CTCursorOnly, CTBlackHole,
                CTInvBlackHole, CTNormal, CTInvert, CTCursCompl,
                CTInvCursCompl);
        TabletMode = (relTablet, scrAbsTablet, tabAbsTablet);
        CursMode = (OffCursor, TrackCursor, IndepCursor);
        CursorPattern = array[0..63,0..3] of integer;
        CurPatPtr = ^CursorPattern;

  Var   TabRelX, TabRelY: integer;  { tablet relative coordinates }
        TabAbsX, TabAbsY: integer;  { tablet absolute coordinates }
        TabFinger: boolean;         { finger on tablet }
        TabSwitch: boolean;         { switch pushed down }
        DefaultCursor: CurPatPtr;   { default cursor pattern }

Procedure XXXIOSetCursorMode( M: XXXOldCurMode ); {***THIS PROCEDURE
    SHOULD NOT BE USED.  USE IOSetTabletMode or IOCursorMode ***}

Procedure IOLoadCursor(Pat: CurPatPtr; pX, pY: integer);
                                        { load user cursor pattern }
Procedure XXXIOMoveCursor(x,y: Integer); {***THIS PROCEDURE SHOULD NOT
    BE USED.  USE IOSetCursorPos or IOSetTabPos ***}

Procedure IOReadTablet(var tabX, tabY: integer); {read tablet coords}

Procedure IOSetFunction(f: CursFunction);

Procedure IOSetModeTablet(m: TabletMode); { set the mode to tell what
                                  kind of tablet is currently in use }
Procedure IOCursorMode (m: CursMode); { if track is true, then Tablet
                                  coordinates are copied every 1/60th
                                  second into the cursor position. if
                                  indep, then coordinates are changed
                                  only by user.  If off, then no
                                  cursor displayed }
Procedure IOSetCursorPos(x,y: Integer); {if trackCursor is false, then
                                       sets cursor x and y pos.  If
                                       tracking, then sets both
                                       tablet and cursor.  }
Procedure IOSetTabPos (x,y: Integer); { if trackCursor is false, then
                                  sets tablet x and y pos.  If tracking,
                                  then sets both tablet and cursor }
Procedure IOReadCursPicture(pat: CurPatPtr; var px, py: integer);
                                  { copies current cursor picture
                                  into pat and sets px and py with
                                  the offsets for the current cursor}
{Procedures to access the Time variable maintained by the tablet}
  Procedure IOGetTime(var t: Double);
  Procedure IOSetTime(t: Double);

{NOTE: Any names starting with "XXX" are not to be used.}
```

The new type "CursFunction" describes the various ways the cursor can be displayed. Cursor functions can be used to turn off the screen display and to change the interpretation of ones and zeros from black/white to white/black. They can thus be used to flash the screen. The client changes the function in use by calling IOSetFunction. The functions are:

CTWhite: Ones in the cursor appear black on the screen and zeros appear white. The rest of the screen is displayed as white irrespective of what is in the screen memory. This is useful when the screen memory is being used for code or data.

CTCursorOnly: This is the negative of CTWhite. Ones in the cursor appear white on the screen and zeros appear black. The rest of the screen is displayed as black irrespective of what is in the screen memory. This is useful when the screen memory is being used for code or data.

CTBlackHole: This is currently implemented incorrectly and is therefore essentially worthless. Ones in the screen memory are displayed white, zeros black (inverse). Zeros in the cursor are black and ones let whatever is underneath show through.

CTInvBlackHole: This is the inverse of CTBlackHole. It is currently implemented incorrectly and is therefore essentially worthless. Ones in the screen memory are displayed black, zeros white (normal). Zeros in the cursor are white and ones let whatever is underneath show through.

CTNormal: This is the default and the only mode previously available. Ones in the screen memory are black and zeros white (normal). Ones in the cursor are black and zeros allow whatever is behind to show through. Thus an OR is done between the cursor and the screen.

CTInvert: This is the inverse of CTNormal. Ones in the screen memory are white and zeros black (inverted). Ones in the cursor are white and zeros allow whatever is behind to show through.

CTCursCompl: This is the a useful function that insures that the cursor is always visible. Ones in the screen memory are black and zeros white (normal). Ones in the cursor show as the opposite of the screen memory underneath. Zeros in the cursor allow the screen to show through. Thus an XOR is performed between the cursor and the screen.

CTInvCursCompl: This is the inverse of CTCursCompl. Ones in the screen memory are white and zeros black (inverted). Ones in the cursor show as the opposite of

the screen memory underneath. Zeros in the cursor allow the screen to show through.

The various modes of the cursor in the previous systems have been separated in this release into separate modes for the tablet and the cursor. The tablet modes describe the types of tablets that can be attached to the PERQ. With the 3RCC Touch Tablet, RelTablet allows the tablet to be used in a relative mode where a touch is interpreted as the current position and movements are offset from there (similar to the way a track-ball or "mouse" works). ScrAbsTablet takes the upper left corner as (0,0) and all touches are determined from there. TabAbsTablet uses the lower left corner as (0,0). Note that the name refers to the PERQ screen which has (0,0) as the upper left corner. The default is RelTablet and this should be sufficient for all applications using either the BitPad or 3RCC Touch Tablet. The client can set the tablet mode using the IOSetModeTablet procedure.

Unlike the tablet modes, clients may want to change Cursor Modes frequently. The default mode, OffCursor, means the cursor is not displayed. This is the default. TrackCursor means the tablet positions are copied into the cursor positions sixty times a second. IndepCursor means that the cursor is visible, but the tablet positions are not copied into the cursor. This mode is useful when the client wants to assign the cursor positions himself. The cursor mode is set with the function IOCursorMode.

New procedures have been provided to manipulate the cursor and tablet positions. In this release, they can now be manipulated independantly using the two functions IOSetTabPos and IOSetCursorPos. If the cursor is in TrackCursor mode, setting either of these will set the other. Otherwise, the settings will be independant. Note that if the client intends to set the cursor while off and then turn the cursor to track and have it in the new position, the tablet position should be set since as soon as the cursor is put in TrackCursor mode, the cursor will jump to the position of the tablet and the old cursor position will be erased.

IOLoadCursor, which sets the picture to be used for the cursor, has not been changed. NOTE THAT THE CurPatPtr MUST POINT TO A ARRAY WHICH IS QUAD-WORD ALLIGNED (e.g. one that was allocated using "NEW(0, 4, myCursorPicture)". The other two arguments to IOLoadCursor tell IO where in the cursor picture, the position should be thought to be. Thus, if the cursor was a circle of radius 21, the client might specify that the position is in the center of the circle by passing 10,10 to IOLoadCursor. The numbers are the offsets from the upper left corner. The default cursor is an arrow pointing towards the upper left corner with offsets of 0,0. Note that only the leftmost 56 bits of width are used in the 4 words of the cursor. The rightmost 8 bits are thrown away.

A new function has been provided that will return the current

cursor picture. This is useful to clients that wish to change the cursor and then restore it to its previous form. IOReadCursorPicture also returns the old offsets. Note again that the array the pointer refers to must be quad-word alligned. The rightmost 8 bits of the array are not modified by this procedure.

Variables exported by IO tell whether the tablet is pressed (TabSwitch) and whether a finger is on the tablet (TabFinger). Note that the latter is only meaningful for the 3RCC Touch Tablet; it is always TRUE if a BitPad is in use. DefaultCursor points to an array with the default cursor picture in it.

Two procedures not related to the cursor are also now exported from IO.PAS. IOGetTime returns a double word that is incremented 60 times a second, and IOSetTime allows the client to set this value. In future systems, this will be used as a time-of-day and date clock and a procedure will be provided for translating the number into a time and date.