1. Introduction

The video driver handles the display of any number of "video pages" in various different display modes, making use of most of the facilities of the NICK chip.

The display is managed in terms of video "pages", with one page corresponding to each EXOS channel which is open to the video driver. Before a channel is opened to the video driver the user must specify various parameters, such as a video mode and page size, by setting EXOS variables. A channel can then be opened to device "VIDEO:". If a filename or unit number is specified then it will be ignored. The video driver will work out how much RAM it needs for this video page and obtain that much RAM from EXOS, including enough for the various variables needed. The only limit on the number of video pages is the amount of available memory. Video pages can only use the internal 64k of video RAM.

Once the channel has been set up in this way, the user can read characters from, or write them to, the video page. The data read or written will have a different meaning for pages of different modes, particularly control characters and escape sequences.

At this stage the video page will not be visible on the screen. A special function call is required to cause a video page to be actually displayed on the screen. It is only at this time that the appropriate line-parameter blocks are set up and the text/graphics will appear. It is possible to display any vertical section of a video page at any vertical position on the screen, covering up anything which was displayed on those scan lines before. If the page width is less than the full screen width then the margins will be adjusted to display the page in the middle of the screen.

Text pages provide displaying of characters from a 128 character font which is initialised to a standard ASCII character set, but any characters may be re-defined by the user. Also text pages provide various control functions including cursor positioning, scrolling of any section of the page and automatic scrolling.

There are various different graphics modes providing a choice of resolution, number of colours and memory usage. All of the graphics modes support drawing of lines and ellipses in a variety of plotting modes and line styles (dotted lines etc.). Characters can be displayed using the same font as text pages. There is a sophisticated flood filling algorithm which will fill any arbitrary shapes subject to stack limitations.

1.1 Co-ordinate Systems

The co-ordinate system used in specifying graphic positions is standardised so that giving the same commands to two pages of different resolutions or colour modes will produce a pattern of the same size on the screen. A graphics page of full screen size will be 972 logical pixels high and 1344 pixels wide. This corresponds to twice the maximum horizontal and four times the vertical resolution available. All beam positions are specified in these co-ordinates, and depending on the colour mode the actual position will have to be an approximation. The origin for this co-ordinate system is the bottom left corner of the graphics page.

Text pages do not use this co-ordinate system, they use a system based on character positions so the top left corner is (1,1) and the top right corner (of a full screen size low resolution text page) is (1,42). Note that the origin for text co-ordinates is the top left of the page.

Attribute graphics mode pages actually keep a beam position in graphics co-ordinates and a separate cursor position in text co-ordinates. The use of these is explained later.

2. Basic Control of Video Pages

As mentioned before, each video page is a separate channel. When a channel is opened to the video driver this implies that another video page is to be created. The video driver looks at EXOS variables which specify the page size, page mode and colour mode. These variables must be set up by the user before opening a video channel. From these variables the video driver determines how much video RAM it needs and obtains that much with an EXOS function call ("Allocate channel buffer").

The video driver maintains the line parameter table in a fixed place in its absolute device RAM area. The line parameter table always consists of 28 line parameter blocks of 9 scan lines each for the display area and various other ones to generate the frame sync and borders. The first line parameter block is reserved for the status line display which is a fixed area of RAM. The other 27 line parameter blocks can display any part of any page, so display is always in vertical units of 9 pixels. All 28 line parameter blocks are initially set up to be blank (ie all border colour). The variable LP_POINTER in the EXOS variable area points to the start of the line parameter table.

2.1 Display Modes

The display mode is specified by an EXOS variable MODE_VID the allowed values of which are:

0 - Hardware text mode (up to 42 chars/line).
1 - High resolution pixel graphics.
2 - Software text mode (up to 84 chars/line).
5 - Low resolution pixel graphics.
15 - Attribute graphics.

Any other values will produce an error (.VMODE) when an attempt is made to open a channel. The three graphics modes correspond to the PIXEL, LPIXEL and ATTRIBUTE modes of the NICK chip (see separate NICK chip specification).

2.2 Colour Modes

As well as the display mode, each video page is of a particular colour mode. The colour mode is specified by an EXOS variable called COLR_VID. The allowed values for this variable are:

0 - Two colour mode
1 - Four colour mode
2 - Sixteen colour mode
3 - 256 colour mode

Any other values will be reduced modulo 4 and so no errors are produced. For text modes it is only useful to use two colour mode, unless the characters in the font are re-defined for doing some sort of block graphics. Also attribute mode must always be in two colour mode, although sixteen colours will actually be available on the page.

2.3 Page Size

Two EXOS variables, X_SIZ_VID any Y_SIZ_VID, define the size of the page to be created. The vertical size is specified in character rows. It can be any value from 1 to 255 although only 27 rows can be displayed on the screen at one time. The horizontal size is specified in low resolution character widths, and can be any number from 1 to 42. Invalid values will produce an error (.VSIZE) when a channel is opened.

A special function call is provided to return the size of a video page. It returns the number of lines and the number of characters per line. The number of lines will be the same as the Y_SIZ_VID EXOS variable when the channel was opened. The characters per line value returned is the actual number of characters per line so in the case of a software-text mode it will be double the value in X_SIZ_VID when the channel was opened.

The parameters for this call are:

Parameters: A = Channel number (1...255)
B = @@SIZE (=2) (Special function code)

Returns: A = Status
B = Number of characters per row
C = Number of rows
D = Mode of page (0, 1, 2, 5 or 15)

2.4 Display Control

Video pages are not actually displayed on the screen until the user explicitly requests this. This request is done by a special function call to the channel. The parameters for this call are:

Parameters: A = Channel number (1...255)
B = @@DISP (=1) (Special function code)
C = lst row in page to display (1...size)
D = Number of rows to display (1...27)
E = row on screen where first row should appear (1...27).

Returns: A - Status

The three row parameters are all given in character row units since the area of screen specified must be a whole number of line parameter blocks. The displayed page will replace anything which was displayed on that part of the screen before. If the channel is subsequently closed then any part of the screen which was displaying that channel will be made border colour (by bringing the margins in the relevant line parameter blocks right in).

A value of 1 for the position on screen parameter (given in register E) refers to the line on the screen directly below the status line. Thus it is not possible to overlay the status line since zero will not be accepted.

If a value of zero is given for the position in the page parameter (register C) then the portion of the screen defined by the other two parameters will be blanked (ie. made entirely border colour).

If any of the parameters for the function call are invalid for any reason then an error (.VDISP) is returned.

3. Video Modes and RAM Usage

When a channel is opened the video driver obtains sufficient RAM from EXOS to support the page. This includes a certain amount for internal variables (128 bytes), an overhead of two bytes for each line of the page, and enough RAM for the display memory which will vary in size depending on the display mode and page size. Note that for any given display mode and page size, all of the four possible colour modes will use the same amount of RAM since they trade off resolution for number of colours without affecting the memory required.

There is a special function call provided which returns the actual address of the display RAM for that page. In fact two addresses are returned because some modes use two different areas of memory. The exact meaning of the addresses for each mode is described below in the section on the appropriate mode. The addresses which are returned are the addresses as seen by the NICK chip. Thus an address in the range 0000h...3FFFh corresponds to RAM segment 0FCh, 4000h...7FFFh corresponds to segment 0FDh and so on for segments 0FEh and 0FFh.

Parameters:
A = Channel number (1...255)
B = @@ADDR (=3) (Special function code)

Returns:
A = Status
BC = Main display RAM address
DE = Secondary display RAM address

Note that the display RAM for a video channel can be moved by EXOS and so the addresses returned by this call will not always remain the same. The operations which can cause channel RAM areas to move are explained in the EXOS Kernel specification. The most important ones are opening and closing of other video channels and linking in user devices or resident system extensions. If any operations of this type have been performed then this special function call will have to be repeated to obtain the new addresses.

In the sections below, where HEIGHT and WIDTH are referred to in specifying RAM requirements, these values are the actual values from Y_SIZ_VID and X_SIZ_VID respectively. Thus for example a full screen size software text page has a width of 42, even though it actually has 84 characters across.

The RAM requirements given below are the amount of channel RAM which the video driver will ask for. In addition to this each page will require a channel descriptor (of 16 bytes) which is allocated by EXOS.

3.1 Hardware Text Mode (Mode 0)

In hardware text mode, one byte of RAM is allocated for each character position on the page. These bytes contain the character codes for the characters displayed on the page and the NICK chip itself translates these into character shapes using the font, when the display is generated. If the top bit of the character code in the display RAM is set then this character will be displayed using palette colours 2 and 3 rather than 0 and 1.

Initially the ASCII map starts with the top left character of the page and continues across the first line followed by the second line and so on down the page. However once any scrolling operations have been performed the ordering of lines on the page will be different so the first byte of the ASCII map will be the first character of a line but not necessarily the top line. The lines can end up in any arbitrary order and even clearing the page will not re-order them.

The RAM usage and address parameters for a page of this mode are:

DE = BC = Start of ASCII character map

Total RAM = 128 + 2*HEIGHT + WIDTH*HEIGHT

3.2 Software Text Mode (Mode 2)

Software text mode maintains an ASCII copy of the page which corresponds to the memory used in hardware text mode. This is one byte for each character on the page. In addition to this it has a complete bit map of the page. The video driver itself builds up the character shapes in this bit map from the character font. This bit map is used by the NICK chip to generate the display, the ASCII map is only used internally by the video driver software.

The bit map initially corresponds directly with what is seen on the screen (assuming the video page is displayed), with one bit corresponding to each pixel. The first byte therefore corresponds to the first eight pixels on the top line of the page, which is the top line of the first character. The next byte corresponds to the top line of the next character and so on until the end of the first row of characters. The next byte will correspond to the second scan line of the first character. This continues for nine scan lines to complete the first row of characters. Subsequent rows of characters are built up in the same way.

The same comments about scrolling apply to software text pages as to hardware text. Scrolling operations can re-order the lines of a page in any arbitrary order. The ASCII map and the bit map are always re-ordered in the same way.

When the video driver is putting characters from the font onto a software text page it masks out the top and bottom bits of each byte and inserts colour information into these bits in the bit map. The values of these two bits control which palette colours are used to display this byte (all 9 bytes in a character will be the same colour). The meaning of these bits is:

bit-0	bit-7	palette colours used
0	0	0 and 1
0	1	2 and 3
1	0	4 and 5
1	1	6 and 7

The RAM requirement and address parameters for a software text page are:

BC = Address of start of bit map (top scan line of top left character).
DE = Address of start of ASCII map (top left character).

Total RAM = 128 + 2*HEIGHT + 20*HEIGHT*WIDTH

3.3 Pixel Graphics Modes (Modes 1 and 5)

The two pixel graphics modes are high resolution (mode 1) and low resolution (mode 5). The only difference between these modes is the amount of RAM they use, and therefore the resolution. A pixel graphics page has an area of RAM which is a straightforward bit map of the screen. The first byte corresponds to the top left of the page, the next byte to the second byte on the top scan line and so on to the end of the first scan line. This is then repeated for the next scan line and so on until the bottom of the page.

The mapping of these bytes into pixels depends on the colour mode and is described in the separate NICK chip specification.

In low resolution pixel mode the full screen width is 42 bytes and in high resolution pixel mode it is 84 bytes. High resolution pixel mode thus uses twice the amount of RAM to cover the same screen area as low resolution pixel mode.

The address parameters and RAM usage for the two pixel graphics modes are:

BC = Address of top left byte of display RAM
DE = Value is un-defined

Total RAM (low resolution) = 128 + 2*HEIGHT + 9*WIDTH*HEIGHT
Total RAM (high resolution) = 128 + 2*HEIGHT + 18*WIDTH*HEIGHT

3.4 Attribute Graphics Mode (Mode 15)

An attribute graphics page requires two areas of RAM of equal size. The first of these (the pixel data area) is a bit map of the video page which corresponds exactly to the bit map for a two colour low resolution pixel graphics page, with each byte defining eight pixels. The second RAM area is the attribute data. Each byte in the attribute data area defines two palette colours in the range 0...15, the INK attribute and the PAPER attribute. The eight bits in the corresponding pixel data byte define which of the two colurs each of the eight pixels covered by this byte will be.

The format of a byte in the attribute data area is:

b7:b6:b5:b4 - PAPER colour, used if a bit in pixel data byte is clear.
b3:b2:b1:b0 - INK colour, used if bit in pixel data byte is set.

The address parameters and RAM requirements for an attribute graphics page are:

BC = Address of start of pixel data area
DE = Address of start of attribute data area

Total RAM = 128 + 2*HEIGHT + 18*WIDTH*HEIGHT

4. Character Output

The screen driver supports both the single character write and the block write EXOS function calls. A block write is exactly equivalent to writing all the characters individually, except that it is rather faster as it avoids the overhead of going through EXOS for every character.

4.1 Printing Characters

All character codes above 31 will be treated as printing characters and will be put at the appropriate place on the video page. All modes have some sort of "cursor" which moves when a character is printed but the details vary between different modes.

The bit maps for characters are stored in a fixed character font which is initialised to an ASCII character set. Each character is eight bits wide and nine bytes deep, including the space between characters and between lines. The user can re-define any of these characters by an escape sequence as specified below.

Character codes in the range 32 to 127 will be displayed as the correct character number from the font. Characters in the range 128 to 255 will be displayed as characters 0 to 127 from the font. Thus writing character 160 (128+32) to a video page will have exactly the same effect as writing character 32. However writing character 159 (128+31) will display character number 31 from the font on the page but writing character 31 will do nothing because it will be interpreted as a control code (this particular control code is ignored).

4.1.1 Text Mode Character Printing

Text pages (modes 0 and 2) maintain a single text cursor which is in text co-ordinates. The printing character is displayed at this position and the cursor moved to the next character slot. At the end of a line the cursor automatically moves to the start of the next line, with automatic scrolling if it is at the bottom of the screen (this automatic scrolling can be disabled).

4.1.2 Pixel Graphics Mode Character Printing

Pixel graphics pages maintain a beam pointer in graphics co-ordinates. The printing character is displayedat this beam position and the beam moved to the next character position, moving to the start of the next line if at the end of a line. Characters can be displayed at any pixel position, not just on character boundaries. There is no scrolling. If the beam is too near the bottom of the page to fit the whole character on then it will not display anything.

Characters are displayed in the current ink colour regardless of whether the beam is on or off. The character is displayed by reading bits out of the font and if the bit is set then a pixel is plotted in the current ink colour using the current line mode. If the bit is clear then the corresponding pixel will be left unchanged. A character will therefore overlay rather than replace anything which is already on the page.

The characters will be the correct shape in any colour mode but the aspect ratio will vary with different modes. To improve legibility the character height is doubled for sixteen and 256 colour mode.

4.1.3 Attribute Graphics Mode Character Printing

An attribute graphics page maintains a text cursor in text co-ordinates and a separate graphics beam position in graphics co-ordinates. Characters are printed at the text cursor position and so will always be in exact character positions. At the end of a line the text cursor will go on to the start of the next line and at the end of the page it will go back to the top left of the page - there is no scrolling.

How the character is displayed depends on the current value of the attribute flags byte for this page. This is a byte which can be set by an escape sequence and is described in more detail later on. It basically controls what sections of the attribute and pixel data will be affected by writing characters or plotting graphics.

4.2 Control Codes and Escape Sequences.

Character in the range 0 to 31 are control characters and are not printed. Some of these are interpreted by video pages, depending on the mode. Any which are not understood are simply ignored. A special control code is ESCAPE (ASCII 1Bh) which is used to start an escape sequence for carrying out various functions.

Below is a list of the control codes and escape sequences interpreted by the various modes. The more complex of these are explained in the next section.

4.2.1 Codes Interpreted by Any Video Page

^Z (1Ah) - Clear entire page and home cursor/beam.

^J (0Ah) - Line-feed. Move cursor down to next line (scrolls if at bottom of screen in text mode and scroll is enabled.)

^M (0Dh) - Carriage return. Returns cursor to start of current line

^^ (1Eh) - Cursor/beam home. (ASCII RS)

escK - Define character (see below)

escC - Set all palette colours \ see below for
escc - Set one palette colour / parameters.

escI<n> - Set ink colour to <n> \ See below for details
escP<n> - Set paper colour to <n> / in different modes

esc=<y><x> - Set cursor position (see below)

4.2.2 Codes Interpreted by Text Pages Only

^Y (19h)	-	Clear to end of line. Does not move cursor.
^H (08h)	-	Cursor left. (ASCII BS)
^I (09h)	-	Cursor right. (ASCII TAB)
^K (0Bh)	-	Cursor up. (ASCII VT)
^V (16h)	-	Cursor down. (ASCII SYN)
esc?	-	Read cursor position. Also supported in attribute mode. (see below)
esc.<n>	-	Set cursor character to character code <n>.
escM<n>	-	Set cursor to palette colour <n>
escO	-	Set cursor display on.
esco	-	Set cursor display off.
escS	-	Set automatic scroll on
escs	-	Set automatic scroll off
escU<m><n>	-	Scroll up lines (m-20h) to (n-20h) m <= n
escD<m><n>	-	Scroll down lines (m-20h) to (n-20h) m <= n

4.2.3 Codes Interpreted by Graphics Pages Only

escA<xx><yy>	-	Position beam at co-ordinates (xx,yy) where xx & yy are each 16-bit hex numbers specified low byte first.
escR<xx><yy>	-	Relative beam movement by amount (xx,yy).
esc@	-	Read beam position. (see below)
escS	-	Set beam on.
escs	-	Set beam off.
esc.<n>	-	Set beam to line style <n> - see below.
escM<n>	-	Set beam to line mode <n> - see below.
esca<n>	-	Set attribute flags byte to <n>. Only allowed in attribute mode (see below).
escF	-	Graphics fill - see below.
escE	-	Plot ellipse - see below.

5. Details of Escape Sequences

5.1 Position Cursor

The escape sequence to position the cursor works in all modes. In text mode it simply moves the cursor. In attribute graphics mode it moves the text cursor but leaves the graphics beam pointer alone. In pixel graphics modes it moves the graphics beam pointer to the appropriate text co-ordinates, so it will be on a character boundary.

The format of the escape sequence is:

esc=<y><x>

This sets the cursor to row (y-20h) and column (x-20h). If either <x> or <y> is 20h (thus setting row or column zero) then that co-ordinate will remain un-changed. This allows just the row or column to be set.

5.2 Define Character

This escape sequence allows the user to re-define one of the 256 characters. Although it is sent to a specific channel, it actually affects the global character font and will thus affect other channels. Characters already displayed on a page will only be affected for hardware text pages. In other modes only subsequently written characters will be affected. The syntax of the escape sequence is:

escK<n><r1><r2><r3><r4><r5><r6><r7><r8><r9>

where: <n> is the character number (0...255)
<r1>...<r9> are the bytes for the nine rows of the character. <r1> is the top row.

Note: In high resolution text mode, only the middle six bits of the character bytes will actually be displayed as the other two are masked out and used to control the colour selection.

5.3 Palette Colours

Each video page has a palette of eight colours associated with it which is initialised to a useful set of colours when the channel is opened. There is an escape sequence with which the user can change all these colours. the format of this is:

escC<c><c><c><c><c><c><c><c>

Each <c> is a byte specifying one of the palette colours and there must always be eight of them.

There is another escape sequence which allows just one palette colour to be changed. The format of this is:

escc<n><c>

Where <n> is the palette colour number 0...7 and <c> is the new value for this palette colour.

When new palette colours are selected any line parameter blocks which correspond to this video page will be updated so the colours on the screen will change.

5.4 Ink and Paper Colours

The user may specify a palette colour for both the ink and paper colour with separate escape sequences. For all video modes the ink colour defaults to one and the paper colour to zero.

For pixel graphics pages the allowed ink and paper colours depend on the colour mode, so it is 0 or 1 in two colour mode, 0...3 in four colour mode, 0...15 in sixteen colour mode and 0...255 in 256 colour mode. Pixels are always plotted in the current ink colour. The paper colour of the display is only changed when the page is cleared.

For attribute graphics mode the ink and paper colours can be in the range 0...15 and they control what colours are put into the attribute bytes. See also the section on the attribute flags byte below which determines whether the attribute and pixel data is actually updated.

In four, sixteen or 256 colour text modes the ink and paper colours have no useful effect because the palette colours for each pixel are determined directly from bits in the character font. In fact they do have some interaction with the displayed colours and it is best to leave them set to their default values. These modes are only useful if the characters have been redefined to provide some sort of block graphics.

In two colour hardware text mode the ink and paper colours are always (0,1) or (2,3). If the ink or paper is changed then the other one is changed to correspond.

In two colour software text mode four colour pairs are available, (1,0), (2,3), (4,5) and (6,7). Characters are always printed in the current colour pair.

5.5 Graphics Line Style

For a graphics page the line-style may be set with an appropriate escape sequence. The line style affects line drawing and ellipse drawing but not character plotting or filling. The values for the line style byte are:

1	-	Solid line (default)
2..14	-	Various types of broken and dotted lines.

5.6 Graphics Line Mode

An escape sequence specifies the line mode byte which has the following meanings:

0	-	PUT plotting (default)
1	-	OR plotting
2	-	AND plotting
3	-	XOR plotting

For pixel graphics pages when plotting a pixel the current ink colour is combined with the old colour of the pixel according to the operation selected by the line mode and then stored.

5.7 Attribute Flags Byte

The attribute flags byte is only supported for attribute graphics pages. It consists of eight separate flags, four for plotting graphics (points, lines, ellipses and filling) and four for plotting characters. The basic operation which is affected is that of plotting a pixel.

When plotting a pixel in attribute mode there are three items which can be affected. There is the bit in the pixel data byte which corresponds to this pixel, and there are the two colours (ink and paper) in the attribute byte which corresponds to this pixel data byte. The attribute flags byte contains a flag for each of these three items which controls whether or not it is affected when a pixel is plotted. If the ink attribute is to be affected then it will be set to the current ink colour. If the paper attribute is to be affected then it will be set to the current paper colour.

The pixel data is rather more complex. Assuming that the pixel data is to be affected then there is another bit in the attribute flags byte which controls what is done with the pixel data, in conjunction with the current line mode. For plotting characters, if this bit is set then the character will be complemented before being plotted. For plotting graphics, if the corresponding bit is set then a zero will be put into the pixel bit instead of the usual one. When plotting graphics (but not characters) the bit is not in fact put directly into the pixel byte but is combined with the current value of the bit according to the current line mode. This means for example that exclusive or plotting will still work.

The top four bits of the attribute flags byte control character plotting and the bottom four control graphics plotting. For all bits the 'normal' state is zero which results in all attributes and pixel data being affected, and plotting to be in the normal sense. The assignement of bits is:

bit-7 Affect paper attributes in character plotting
bit-6 Affect ink attributes in character plotting
bit-5 Affect pixel dtaa in character plotting
bit-4 set to complement character before plotting

bit-3 Affect paper attributes in graphics plotting
bit-2 Affect ink attributes in graphics plotting
bit-1 Affect pixel data in graphics plotting
bit-0 set to do graphics plotting in 0's instead of 1's

Note that when filling bit-1 of the attribute byte is assumed to be clear regardless of its actual state. This is necessary because if the pixel data were not affected then the fill algorithm would never terminate.

5.8 Graphics Fill - Paint

The graphics fill command is a simple escape sequence which does a fill from the current beam position. It fills in the current ink colour up to any boundary which is not the same colour as the current beam position. It handles concave shapes and tests for reaching the edge of the video page. It may fail to fill the entire shape if it runs out of stack but this should only happen with extremely complex shapes since it does garbage collection on the stack when it gets full.

5.9 Graphics Ellipse Drawing

The ellipse drawing routine takes two 16-bit parameters (low byte first) specifying the x and y radii. To draw a circle these should be the same value. The centre of the ellipse will be at the current beam position. The format of the escape sequence is:

escE<xx><yy>

6. Character Input

6.1 Simple character input

When a character is read from the video driver the result depends on the mode. In text modes the ASCII code of the character at the cursor position will be returned, without moving the cursor. In graphics mode the palette colour of the pixel at the current beam position will be returned. This will be 0 or 1 in two colour mode, 0...3 in four colour mode, 1...15 in sixteen colour or attribute modes and 0...255 in 256 colour mode.

6.2 Reading Cursor Position

The escape sequence: esc? is supported in text and attribute graphics modes. It triggers the video channel to return the current cursor co-ordinates as the next two characters read from this channel. The co-ordinates will be returned in the same way as they are specified for cursor positioning, ie. with a 20h offset added on.

6.3 Reading beam position

This is supported by graphics pages only (pixel and attribute modes). The escape sequence: esc@ will trigger the channel to return the current graphics beam position as the next four bytes read from the video page. The co-ordinates are returned in the same format as they would be specified for an absolute beam position.

7. Miscellaneous

7.1 Status Line

As mentioned earlier there is a special status line display which is outide the normal page/channel structure. The first line parameter block is reserved for this status line and will never be used to display any video page. The two byte variable LP_POINTER which is at a fixed address in the EXOS variable area contains the address of the start of this line parameter block. This is used by the cassette driver to modify the palette colours of the status line to provide the cassette level meter display. It could also be used by other programs to manipulate the status line display.

An EXOS variable (ST_FLAG) is provided which will cause the status line to be displayed if it is zero and removed from the display if it is non-zero (that region of the screen will be border colour). This is implemented by the video driver examining ST_FLAG every interrupt and setting the margins in the reserved line parameter block appropriately.

There is a two byte pointer (ST_POINTER) defined at a fixed address in EXOS variable area which contains the address of the 40 byte area of RAM which is the status line. This pointer will point to Z-80 page-2 and the RAM will be at this address in segment 0FFh. The user or external devices can access the status line by finding its address from ST_POINTER. Built in devices can access it directly by using the public sysmbol ST_LINE which is the start of the status line RAM (in Z-80 page-2 segment 0FFh).

7.2 Border and Fixed Bias Registers

The EXOS variable BORD_VID defines the current border colour and is written out to the Nick chip border register on every video interrupt. The border colour can thus be changed simply by changing the EXOS variable.

The Nick chip also has a fixed bias register which is written to by EXOS on every video interrupt. However since its bits are used for various different purposes, the value written out is combined from three different EXOS variables.

The EXOS variable BIAS_VID will have its top 5 bits written out to the bottom 5 bits of the fixed bias register. This controls palette colours 8 to 15 in sixteen colour mode as described in the separate Nick chip specification.

The bottom two bits of the EXOS variable SPRITE will be written to the top two bits of the fixed bias register. This controls the priority selection of external colour inputs from the expansion bus and may thus be used with a sprite generator. For details of exactly what these bits do, see the separate Nick chip specification.

SPRITE EXOS Variable does not exist in version 2.0

Bit 5 of the fixed bias register is written out from the EXOS variable MUTE_SND and is used to enable or disable the internal speaker. The EXOS variable should be zero to enable it and 0FFh to disable it.

7.3 Resetting the Character Font

As mentioned before the video driver maintains a single 128 character font which is used for all text pages and also for displaying characters on graphics pages. This is initially set up at cold start time, but is not re-initialised at subsequent device initialisations. Thus if any characters in the font are re-defined then these re-definitions will survive a warm reset or a transfer of control to a new applications program.

The font can be reset to its initial state by making a special function call to any video channel. This will set all 128 characters back to their initial shapes. Note that this will affect all characters on a hardware text page instantly but will only affect subsequently printed characters on a software text or graphics page. The parameters for this call are:

Parameters: A = Channel number (1...255)
B = @@FONT (=4) (Special function code)

Returns: A = Status

8. Quick Reference Summary

8.1 EXOS calls.

OPEN/CREATE CHANNEL - Treated identically. Supports multiple channels. Device name "VIDEO:". Filename and unit number ignored. EXOS variables MODE_VID, COLR_VID, X_SIZ_VID, Y_SIZ_VID must be set before open.

CLOSE/DESTROY CHANNEL - Treated identically.

READ CHARACTER/BLOCK - Returns cursor character or pixel colour. Can be used to read cursor/beam position.

WRITE CHARACTER/BLOCK - Displays printing characters. Many control codes and escape sequences interpreted to provide special features.

READ STATUS - Always returns C=0.

SET STATUS - Not supported.

SPECIAL FUNCTION -

@@DISP = 1	Display page on screen
@@SIZE = 2	Return mode & size of page
@@ADDR = 3	Return video RAM address
@@FONT = 4	Reset character font

8.2 EXOS Variables

MODE_VID	-	Video mode	\
COLR_VID	-	Colour mode	\ Must be set up before
X_SIZ_VID	-	Characters/line (1...42)	/ a channel is opened
Y_SIZ_VID	-	Lines in page	/
ST_FLAG	-	Zero to display status line.	
BORD_VID	-	Overall screen border colour	
BIAS_VID	-	Colour bias for palette colours 8 to 15.	
SPRITE	-	External colour priority.	

SPRITE EXOS variable does not exist in version 2.0

++++++++++ END OF DOCUMENT ++++++++++