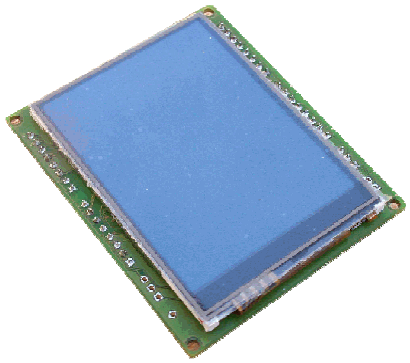
# VT100 Touch Display BV4629_V2



## BV4629_v2

## VT100 LCD 2.8” Touch Display

Product specification August 2010 V0.a

# Contents

| Rev | Change |
|---|---|
| April 2012 | Version 2 hardware |

# 1. Introduction

The BV4629 is a 240 x 320 LED touch screen display.

The display is fully self contained and works with a simple serial or I2C interface to make the burden of interfacing as simple as writing to a UART or I2C bus. Information can be stored and recalled on the built in Flash memory.

The display uses a superset of the VT100 command set and so positioning of text and controlling the display is simply a matter of sending the correct escape sequences. Four bytes for example will clear the screen: 0x1b,0x5b,0x32,0x4a. This looks like "<esc>[2J" as text.

The BV4049_V2 is the SAME hardware as the BV514 but with different firmware installed and so please refer also to that data sheet.

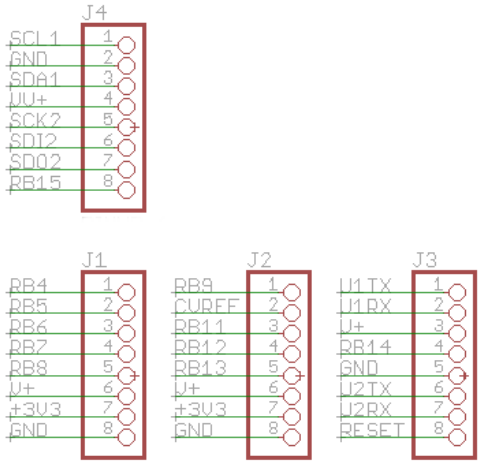
This data sheet will deal specifically with the hardware that utilises the I2C and Serial interfaces. The firmware is completely user interchangeable and so this hardware can be used as a single purpose display (BV4629) or a general purpose controller (BV514)

# 2. Documentation

The main documentation can be found at http://doc.byvac.com under the products menu and will have at least the following:

- BV4629 data sheet (this)
- BV514 data sheet
- BV4629 User Guide

# 3. Serial Interface




The serial interface is via the USB, J3 and J4. To connect the device. The relevant pins for the BV4629 are as follows:

| Pin | Description |
|---|---|
| J4-1 | SCK – I2C clock |
| J4-2 | Ground |
| J4-3 | SDA – I2C data |
| J4-4 | VV+ (see text) |
| J3-3 | V+ power in see text |

**I2C**

| Pin | Description |
|---|---|
| J3-3 | V+ |
| J3-4 | RTS |
| J3-5 | Ground |
| J3-6 | TX |
| J3-7 | RX |
| J3-8 | RESET |
| J2-1 | Int |

**J3 – Serial & Power**

## 3.1. Electrical considerations

All of the pins are 5V tolerant the device will work on either 3V3 or 5V logic.

The display requires at least 3.6V for correct operation. There is an internal voltage regulator and so any voltage up to 18V on pin +V is acceptable.

# 4. Serial Interface

**+V** Is the main power supply for the display which should be capable of providing sufficient current when the back light is illuminated. This goes through the on board 3V3 regulator and so can be any voltage from about 4.5 to 18V

**RX** This is the serial receive and expects signals 0 to 3V3 or 5V The idle state is high. The input is a standard byte frame of 1 start bit 8 data bits and 1 or 2 stop bits.

NOTE the interface will NOT directly accept signals from a standard COM port which outputs +ve and negative voltages.

**TX** This is the output from the display.

**RTS** (output) Hardware handshaking is an essential part of this interface. Without it the display will work but delays must be included after each character otherwise information will be lost (see the ACK section). This line is held low (0V) by the display when it is ready to accept data, it will put this line high when it is busy, the transmitter should monitor this and only send characters when the line is low. This

signal is normally connected to the CTS line of the transmitting device.

**RESET**

Reset: this line when pulsed in either direction will cause the display to reset. It is normally held either low or high and when taken to the other state the display will reset. Leave disconnected if not required.

**Int**

This is an output from the touch screen interface and is normally high. When the screen is touched it goes low and remains low until reset by an appropriate command.

## 5. Handshake and ACK

The ideal interface would implement the hardware handshaking mechanism using RTS/CTS. This is where the device's RTS is connected to the host CTS, when the device is busy, it raises the RTS line and the host temporarily stops transmitting.

Where it is not possible to do this and also where software can be simplified, the ACK mechanism is available. This is switched off by default and so must be enabled using the appropriate command.

Because the device takes a finite time to carry out a command, say clearing the screen, it is useful to have an indication of when that has happened and more accurately when the device is ready to accept another command. This is what the ACK mechanism is for. When enabled and when the device has completed a command and is ready to accept another command it will send an ACK character. The actual character (value from 1 to 255) can be specified by the user.

Not all commands will send an ACK where this is so it is indicated on the command table. Sending plane text for example does not send an ACK character. To get round this without resorting to a hardware handshake a command is provided that will just return the ACK character.

If this is used for every 60 or so characters sent, if the buffer is filling up then sending this character and waiting for the ACK will effectively empty the buffer so more characters can be sent.

## 6. Control Commands

There are TWO distinct methods of controlling the device, one is by serial commands and the other is by I2C. Only one can be active at any one time, see the section on I2C for how to activate this method of command.

The device when used serially is controlled by issuing control commands that always begin with and escape byte (0x1b). The escape is followed by various sequences to make up individual commands.

The command is accepted as soon as the last byte of the command is entered, so for example when clearing the screen with esc[2J the screen will clear as soon as 'J' is entered.

## 7. Serial

When connecting remove the jumper below J1 marked USB, this will disable the FTDI USB chip.

### 7.1.1.Connections

| | |
|---|---|
| J3-6 | to host RX |
| J3-7 | to host TX |
| J3-3 | Power in up to 18V |
| Ground to any connector that has it, e.g. J4-2 | |
| The following are optional | |
| J3-4 | to host CTS |
| J3-8 | to host GPIO for reset |

The serial interface uses 1 start bit, 8 data bits and 1 or 2 stop bits, no parity. By default the Baud rate is determined by the first character received which MUST be a CR (carriage return) byte value 13 or 0x0d. The rate is selected from the following rates:

2400, 4800, 9600, 14400, 19200, 38400, 56700, 115200 and 2000000

No other rates are acceptable and so the host must be set to one of these. It is possible to fix one of these rates (see the command table). Once the Baud rate has been established any Baud rate can be set for this device, see the appropriate command in the table.

There is a sign-on screen that will indicate when communication has been established, when this happens anything placed on the serial bus will appear on the screen.

This also applies to the **USB** which is simply another way 'into' the serial interface.

## 8. I2C

**(Default address 0x66, 8 bit) (0x33, 7 bit)**

### 8.1.1.Connections

| | |
|---|---|
| J4-1 | to SCL |
| J4-2 | to GND |
| J4-3 | to SCL |
| V+ can be any V+ pin, e.g. J3-3 | |
| The I2C Jumper below J4 must have 3v3 and VV+ connected together. | |

The device can operate from serial **OR** I2C. This is determined by the SDA line which, when disconnected, is held low by a high value

resistor. The mode is determined at reset or switch on.

With no I2C bus connected the SDA line is low and thus the device detects this and starts up in serial mode. With an I2C bus connected the pull up resistors on the master or bus hold this line high and the device goes into I2C mode. The mode remains until reset.

*The I2C Select jumper at the bottom of J3 (3 pads) should be left open unless this device is going to provide the pull up resistors, if so see the BV514 data sheet.*

The SCL and SDA lines are 5V tolerant and so will work from either 3V3 or 5V.

To make interfacing easier the VT100 commands have an equivalent I2C command. All I2C commands start with 27 (0x1b). Any I2C byte that is not 27 will be sent directly to the display.

When connecting remove the jumper below J1 marked USB, this will disable the FTDI USB chip.

#### 8.1.2.Write

1. Send start condition
2. Send device address with write
3. Send 27
4. Send command/s
5. Send stop condition

#### 8.1.3.Read

1. Send start with write
2. Send device address with write
3. Send 27
4. Send command/s
5. Send restart condition with read
6. Read bytes
7. Send stop condition

ALL commands on this device begin with a write, even commands that read something a command is sent first.

### 8.2. Writing Example

To clear the screen the following bytes are sent:

0x1b, 0x2d, 0x1 or 27,45,1 in decimal

```
void LCD_cls
{
        I2Cstart();
        I2Csend(0x66);
        I2Csend(27);
        I2Csend(45);
        I2Csend(1);
        I2Cstop();
}
```

The input to the display is buffered and so it can accept the characters as fast as they can be sent, however the display does take a finite time to carry out the command.

### 8.3. Reading Example

Reading is more involved as it requires a write first then a restart but this is quite common for most I2C devices. The example is to get the device ID which is:

27, 35

```
int LCD_deviceID()
{
int rv;
        I2Cstart();
        I2Csend(0x66);
        I2Csend(27);
        I2Csend(35);
        delay(1);
        I2Crestart()
        I2Csend(0x67);
        rv=I2Cget() << 8; // high
        rv=+I2Cget() & 0xff; // low
        I2Cstop();
        return rv;
}
```

The code first sends out the command for reading the device ID, this will fill the I2C output buffer with two bytes that can be subsequently read with the two read commands.

In this situation clock stretching cannot be used to handshake the I2C and so a delay must be allowed to enable the display to fill the output buffer with the values before attempting to read them out. The recommended time for this delay is given in the command table.

Note that the address notation used is 8 bit, this means that the W/R bit is part of the address and so when the bit is set to 0, this is the LSB of the byte, it means write and thus the address will be an even number. When the LSB is set to read (high), this will give an odd address and so to read the device the address is 0x67 as shown above.

The code shown above illustrates the principal and is not actual code as most 'C' implementations have their own method of manipulating the I2C bus.

### 8.4. Buffering

Because the output from the slave is buffered any unread bytes will be still in the buffer waiting to be read out so it is important to read the correct number of bytes for each command.

### 8.5. Address

The I2C address is set to **0x66** by default. This is the 8 bit address which includes the read/write bit (see www.i2c.byvac.com ) for information about 7 and 8 bit I2C addresses.

This can be changed to any address in the range 2 to 254 by the command given in the command table.

## 9. Text

The display will, by default, display any text sent to it via the serial or I2C interface. This means that any bytes with a value other than 27 will be displayed on the screen. When the text reaches the bottom of the screen scrolling will take place and continue to do so until the screen clear command is received.

*Scrolling will only work when the display is in portrait orientation.*

Cursor movements, clearing lines and scrolling etc. are all based on the **current** font which may give some unexpected results. If a double height font is in use then all of the movement commands will use that height.

Once scrolling begins, the cursor movement commands will be unpredictable and so these commands should not be used if it is intended to scroll the display.

## 10. Scrolling & Landscape

The display will work in either portrait (default) or landscape mode. Scrolling is only available in portrait mode. When in landscape and the text reached the bottom of the screen, the next line will appear at the top of the screen.

Once scrolling has started the x and y co-ordinates change position and so it is not practical to mix graphics and text positioning with scrolling as again unexpected results will occur.

## 11. Flash Memory

The display is fitted with a 16Mb flash memory that is organized as 500 pages of 4kB (bytes) each. Each page is called a block and can be written to or erased using the system commands.

When writing to a block, if the information written is larger than a single block, the next block will be used up until the maximum number of blocks. It is up to the user to make sure that the block/s are erased before use.

*Blocks must be erased by the user before use, using the erase command.*

As an example: if a picture 100x100 pixels is being stored then this will take up 100x100x3 bytes, as each pixel requires 3 bytes. The total storage is therefore 30,000 bytes. Writing this picture to say block 10 will occupy block 10 through 18. This is because dividing 30,000 by 4096 gives just over 7 blocks. ( 30,000/4096 = 7.3).

The system will take care of writing to the blocks without user intervention and also when displaying the picture the system will use all 8 blocks. The user just has to be aware of the space used.

Block 0 is reserved for system use and this is where the defaults for the display are kept. It can be erased by the user (using esc{0,1E or the I2C equivalent) but can only be written by the user using the system save command. Semi-permanent defaults are kept in here such as the Baud rate and touch screen calibration etc.

If this block is erased then the display will revert to the system defaults and if a touch screen is present then calibration will be requested at start up.

Several commands are associated with the flash memory to allow full manipulation.

**Memory Map**

It may be useful to know which blocks are in use and which are free, the memory map command will show this as a graphical representation of coloured rectangles starting at block 0 and ending at block 499. Each row is 10 blocks:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 10 | 11 | 12 | 13 | | | | | | |
| 20 | .. | .. | | | | | | | |
| etc…… | | | | | | | | | |
| 480 | | | | | | | | | |
| 490 | | | | | | | | | 499 |

**Colour coding**

- Green – unoccupied
- Red – in use
- Blue – start block of a bitmap
- Yellow – start block of a macro
- Magenta – start of a Font

The design goal behind the use of the Flash is to allow the host system to store often used information and recall it very quickly, graphical buttons for example.

## 12. Pixel Format

Each pixel requires 3 bytes of information for it to be displayed and is always specified as Red, Green and Blue in that order. The intensity of the pixel is given by the value of the byte that can range from 0 (off) to 63 (fully on).

Combining these three values will produce a range of 262,144 colours. As an example to produce a bright green pixel the following is specified:

0,63,0

This will just illuminate the green pixel, similarly white would be

63,63,63.

A full picture is made up in the same way, so when calculating the number of bytes that a picture required the formula is height x width x 3.

The above information should be used when specifying the foreground and background colours.

## 13. Picture Format

There are two methods of displaying a picture, one by sending data from the com port to the TX line and the other is to store it in flash and recall it later. The second option takes longer to write to flash but is displayed much quicker one recalled.

The format required is:

<height><width><r,g,b,r,…..

Height and width are specified as 16 bit values, the first byte is the high byte of the value and the second is the low byte. The picture data is specified as bytes. Here is a snippet of a picture file, values in hex:

01 03 00 24 00 00 3F 00 00 3F 00 00 3F…

The height is 01 03 which is 259 decimal and the width is 24 which is 36 in decimal. Three pixels are show with maximum green intensity as red is 0, blue is 0 but green is 0x3f (63).

The resource zip file contains a program that will convert bmp files to the above format, the extension given when converted is bmc. In addition the BV-Tools program will also convert and display BMP images.

## 14. User Fonts

User fonts can be created using the CBFG program and then converted to the correct format using BV-Tools. There is more information on how to do this in the User guide. NOTE that there is a limit of 12k to the size of the user font file, this is because it is stored in RAM.

## 15. Touch Screen

For displays fitted with a touch screen an initial calibration is required. This will be requested on first switch on unless it is stored in the system flash at block 0.

If calibration is requested, this will happen if the user erases block 0, then touch the points as requested. When calibration is complete the values will be saved to block 0 of flash automatically.

If re-calibrating the display using the calibrate command then the user will need to specifically save the calibration using the esc[?27D command.

Co-ordinates can read form the touch screen and acted on by the host system. Various commands allow information such as where and when the touch screen was pressed. There is also an interrupt flag and hardware output to indicate that the screen has been touched.

**Scan**

The accuracy can be varied depending on the application, the trade of is time it takes to register the touch. The touch screen commands will scan the screen 'x' number of times as soon as a touch is detected. The average is taken and this is returned back to the user. The more scans the longer it takes but the higher accuracy.

The default is set at 10 but this can altered if required.

NOTE that 10 scans will take less than 1ms.

**Touch Commands**

Several commands are provided that make life easier for the host and where a command returns a value the format is described in the command table.

The screen is examined every 5ms and if touched will set the Int pin low and set a touch flag to 1. It will also set the Ring Indicator (RI) on the USB interface so that this can be picked up by a PC program if required. BV_Tools uses this to display the touched co-ordinates.

The interrupt pin can be examined by hardware and the flag can be examined by software. The last touch command will return the x and y co-ordinates of where the screen was last touched. It also returns the flag and sets 'Int' back to high. If it is set to 1 then the co-ordinates are 'fresh'. The last touch command will clear the flag back to 0.

An alternative method of reading the touch screen is to use a command that waits for the touch to be made.

## 16. Sign On and Macro

A custom sign on screen can be created using any mixture of commands. It is recorded using the esc{<bl>M and can be tested using the esc{<bl>r command. This can be set to run at start up using the esc[<bl>m command.

*There is also an example of using macro in the user guide.*

To give an example, suppose that there is an image in block 1 we wish to display with some text. The macro would have the following commands: (# is a comment)

esc{2M # start macro

esc[2J # clear screen

My Image > # text that will show

esc{1r # show image in block 1

esc esc # 2 esc finishes macro

The macro can now be tested using esc{2r If it does not work then block 2 must be erased before trying again. If it does work, to make it appear at start up the following is required:

esc[2m # set the macro start block

esc[?27D # write defaults

All of the above can of course be placed into a script file and sent using the text send facility on most terminals.

**NOTES:**

1. The maximum size of a macro is 100 bytes. This is not a particular restriction as a macro can call text and other images from Flash.
2. When saving a macro to run at start up, the baud rate also becomes fixed to whatever it is currently set at.
3. Turning off the macro will not automatically turn off the fixed Baud rate.

## 17. Factory Reset

The configuration information is stored on block 0 so erase this block to get back to the default settings.

## 18. Uploading new Firmware

This device has the capability of upgrading its own firmware should bugs be found or new features added. The BV-Tools program is used for doing this, see the User Guide.

In addition BASIC can be loaded onto this device to make it a general purpose microcontroller.

## 19. Issues

The following are known issues. These may be as a result of 'software features' or simply that it is not possible to overcome the issue due to the physical characteristics of the display or controlling devices.

1) back space does not fully erase the previous character. This is because currently the widths of the displayed characters are not stored.
2) The Maximum user font size is limited by the available RAM.

## 20. VT100

Once connected the display simply accepts text and duplicates it to the screen.

To control the display, for example clearing the screen or placing letters in a certain location escape codes are used. An escape code is the escape key (byte decimal 27 or hex 0x1b) followed by various symbols, numbers and letters.

Escape codes have been used for many years and were a feature of the early terminals. The most popular set of codes is called VT100 used by old DEC terminals. It was so popular that it became a standard way to control terminal screens before the days of the PC.

This display wherever possible uses the most common of these codes.

### 20.1. Implemented Escape Codes & I2C commands

In the table columns indicate:

**S** is the VT100 standard, Y is standard, P is partially standard, N is non-standard

**Code** is the escape sequence to invoke the command

**DEF** column shows the default values from the factory, and F in this column will indicate that this value will be stored when writing the system to block 0

**ACK** column indicates that this command is capable of sending an ACK character when it has finished.

**I2C:** IMPORTANT the numbers referred to in this column are actual values rather than their ASCII text equivalents and unless otherwise stated are 1 byte.

Grayed out cells are not implemented on this device. Unless otherwise stated all numbers **are in decimal**.

| S | Code | I2C | DEF | ACK | Name | Description |
|---|---|---|---|---|---|---|
| | **Cursor Movement and Text Placement** | | | | | |
| Y | LF | 10 | | | Line feed (10) | Moves the cursor down one line, if the cursor is on the last line then the command is ignored. |
| Y | CR | 13 | | | Carriage return (13) | Moves the cursor to the beginning of the current line |
| Y | BS | 8 | | | Back space (8) | Moves the cursor back one space and deletes the character immediately to the left before moving the cursor. When the cursor reached the left margin it stops. |
| Y | esc[<row>;<col>H<br>esc[H | 27 14 row col<br>27 15 | | Y | Move cursor to line and column | Moves cursor to specified line and character position. The first position is 1 and the last depends on the display and settings. esc[0;0H will move the cursor to the home position. Values out of range will be ignored.<br>This command is generally used for text and will place the cursor on a text boundary rather than a pixel boundary. For more precise pixel level positioning see the graphics and pixel drawing commands.<br>NOTE: Once scrolling has started, this command will have unexpected results as the home position changes. |
| Y | esc[<num>A<br>esc[A | 27 16 num<br>27 17 | | Y | Move cursor up. | Moves the cursor up one or more lines, specified on its own 'esc[A' it will move the cursor up one line, specified with a number it will move the cursor up that many lines. When the cursor reaches the first line subsequent calls are ignored.<br>esc[A can be used instead of esc[1A |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | NOTE: Once scrolling has started, this command will have unexpected results as the home position changes. |
| Y | esc[<num>B<br>esc[B | 27 18 num<br>27 19 | | Y | Move cursor down. | As esc[<num>A but moves cursor down, this is the equivalent of line feed.<br>esc[B can be used instead of esc[1B |
| Y | esc[<num>C<br>esc[C | 27 20 num<br>27 21 | | Y | Move cursor right. | As esc[<num>A but moves the cursor to the right.<br>esc[C can be used instead of esc[1C |
| Y | esc[<num>D<br>esc[D | 27 22 num<br>27 23 | | Y | Move cursor left. | As esc[<num>A but moves cursor to left.<br>esc[D can be used instead of esc[1D |
| Y | esc[H | 27 15 | | Y | Cursor home | Moves cursor to home position does not clear the display<br>NOTE: Once scrolling has started, this command will have unexpected results as the home position changes. |
| | **System Settings** | | | | | |
| N | esc[?10a | 27 24 1 | On F | | Add Line Feed | Adds Line feed when CR is received as some terminals only send CR |
| N | esc[?10b | 27 24 0 | | | Remove Line Feed | Turns off the additional line feed set by esc[?10a |
| N | esc[?10L | 27 25 1 | | | Landscape | Display is orientated in landscape. The landscape orientation will only apply to newly entered text.<br>** Note that scrolling is not available in this mode, the text will wrap back to the top of the screen |
| N | esc[?10P | 27 25 0 | On F | | Portrait | Display is in portrait orientation |
| N | esc[?11B | 27 26 1 | On | Y | Back light on | |
| N | esc[?11b | 27 26 0 | | Y | Back light off | |
| Y | esc[?25I | 27 28 0 | | | Hide cursor | turns the cursor off |
| Y | esc[?25h | 27 28 1 | On F | | Show cursor | turns the cursor on |
| N | esc[<n>f | 27 29 n | | | Cursor | Sets cursor flash rate in 5ms intervals. For example to flash the cursor every 100ms use the command: |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | Flash Rate | esc[20f<br>Minimum value is 1 and maximum is 255. |
| N | esc[?29 <baud>a | | F | | Sets a new Baud rate | Sets a new Baud rate that will take immediate effect. The number should be entered in decimal and there must be a **space between** the 29 and the value. Example:<br>esc[?29 38400a<br>With this command it is possible to set any Baud rate from 1 to 2,000,000 and the effect is immediate as soon as the 'a' is received. The rate will be set only until the device is reset. To make it permanent follow the command with [?27D to write the defaults to flash.<br>To return to an automatic Baud rate set the Baud rate to 0 and follow this with a write to flash, thus:<br>esc[?29 0a esc[?27D |
| N | esc[<num>E | 27 32 n | Off | | ACK | Sets and activates the ACK character as an alternative to hardware handshake. Some commands will send a character to the receiving device, this can be used to determine when the command has fished. The host can wait for this character before sending the next command. To set up the ACK to send 'X for example the following is used:<br>esc[88E<br>88 Is the ASCII code for 'X'. To turn off the ACK system use esc[0E (esc[zeroE). |
| N | esc[E | 27 33<br>(1ms) | | Y | ACK | This will return the ACK character regardless of the ACK setting. This command can be useful for commands that don't normally return ACK and there is no hardware handshake. |
| N | esc[?30a | 27 34 | | | Displays current status | |
| N | esc[?31d | 27 35<br>(1ms) | | Y | Device ID | Returns device ID as a number |
| | | | | | | **I2C**: This is returned as a 16 bit value, in two bytes, high byte first. |
| N | esc[?31y | 27 36 1 | On<br>F | | Com Flag (0n) | When communication is established with the device, i.e. at switch on or when the auto Baud mechanism has selected the correct Baud rate, ASCII code 42 '*' is sent to indicate this has happened. This commands switches this on, it is on by default |
| N | esc[?31n | 27 36 0 | | | Com Flag (off) | Switches the flag off see esc[?31y |

| | | | | | | |
|---|---|---|---|---|---|---|
| N | esc[?32d | 27 37 | | | Debug | Shows the input as hex rather than text, this is useful for seeing what is actually being sent to the display when trying to figure out command sequences. The only way out of this mode is to reset the display. |
| N | esc[<num>a | 27 38 adr | | Y | Set I2C Address | This will set the I2C address and must be followed by EEPROM write. **The new address will not take effect until reset.** The default address is 0x64 (100). As an example to set the address to 0x42 (66) :<br><esc>[66a<esc>[?27D<br>Both of the commands must be specified to change the address and then restart or <esc>c<br>IMPORTANT An even number MUST be specified to give a valid 8 bit address, the device will not work if an odd address is specified, the device does not check the address you enter. |
| | **Miscellaneous** | | | | | |
| N | esc[<pitch>;<duration>b | 27 39 | | Y | Beep | Sounds beeper with a pitch and duration, the higher the pitch value the lower the sound. Any values are acceptable but useful values would be around the 900 mark for the pitch and around 1 or 2 for the duration. Don't expect too much from this as the tone is generated between interrupts and so the sound may not be completely predictable. |
| | | | | | | **I2C:** Currently this will just emit a beep |
| N | esc[nb | 27 30 n | | | Disable beep | Disables / Enables warning beep.<br>esc[0b Disables the warring beep<br>esc[1b Enables the warning beep (Default enabled) |
| P | escc | 27 40 | | | Reset device | Carries out a software reset similar to switching the device on. It does not reset back to the default settings, to do this erase block 0, see command esc{<bl><n>E |
| | **Font and display settings** | | | | | |
| P | esc(0 | | | Y | User Font | After issuing this command send a binary font file (.BF extension). The font will then be tset to the user font just downloaded. |
| P | esc(n | 27 41 n | On F | Y | Font size 1 | Sets built in font as working font 1 to 9. |
| N | esc[<num>;<num>;<num>f | 27 42 r g b | | Y | Sets foreground colour | The colour is specified as three numbers red, green and blue each colour can have a value of 0-63, numbers outside of this value will be taken to the nearest value. So for example if 99 was specified then 63 would be entered.<br>To set the foreground colour to yellow for example would be:<br>esc[63;63;0f |

| | | | | | | |
|---|---|---|---|---|---|---|
| N | esc[<num>;<num>;<num>b | 27 43 r g b | | Y | Sets background colour | This will set the background colour. The colour is used from when it is set and so to clear the screen to a new background colour, use this first and then command esc[2J |
| | **Screen clearing** | | | | | |
| Y | esc[2J | 27 45 1 | | Y | Clear screen | Clears display and homes cursor also stops scrolling and so cursor movement commands will work as expected after this command |
| N | esc[3J | 27 45 2 | | Y | Clear screen and reset all defaults | Clears display, homes cursor and resets to the defaults values. Note if the values have been set by the user then this will use those values. |
| Y | esc[K | 27 45 3 | | Y | Clear line from cursor right | All characters are cleared form the cursor to the end of the line, the cursor remains where it is and the character under the curser is also erased. |
| Y | esc[1K | 27 45 4 | | Y | Clear line from cursor left | The characters are cleared form the display starting at the beginning of the line to the current cursor position. |
| Y | esc[2K | 27 45 5 | | Y | Clear entire line | The current line is cleared and the cursor is places at the beginning of the line. |
| | **Touch Screen Commands** | | | | | |
| N | esc[c | 27 50 1 | F | | Calibrate touch screen | If the device is fitted with a touch screen the calibration will be requested at start up if this is the first time use or after a factory reset. The calibration settings are be stored in Flash when the command completes if requested from start up or after erasing block 0. If re-calibrating using this command then **save** the results using esc[?27D |
| N | esc[t | 27 50 2 | | | Test calibration | This will simply draw pixels under the stylus to check if the calibration is somewhere near. It will stay is this mode until any characters are sent to the serial interface. |
| N | esc[e | | | Y | | As esc[l but returns immediately with a result, if the screen is not being pressed it returns with "*CR" |
| | | | | | | **I2C:** This command is not available in I2C as it suspends operation and an attempt to read the I2C buffer when there is nothing in it may result in an error. Use esc[l instead. |
| N | esc[m | | | Y | Move to Touch | Moves cursor to touched position. The command will **wait** until screen is touched and if ACK set return the ACK when the screen has been touched. |
| | | | | | | **I2C:** This command is not available in I2C as it suspends operation and an attempt to read the I2C buffer when |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | there is nothing in it may result in an error. Use esc[l instead. |
| N | esc[<num>s | 27 51 num | F | | Set Touch Scan | By default the touch screen is scanned up to 10 times and an average taken to determine the best possible accuracy. If the user stops touching before 20 is up then the number of scans so far is used. This command alters the maximum number of times the scanning will take place and can range from 1 to 255. |
| N | esc{x,y,x1,y1h | | | Y | Area Hit | Specify an area of the screen and this will return 1 if the screen was touched within that area and 0 if it was touched outside that area. This command is useful for detecting buttons and saves the host some work. The command will not return until the screen has been touched.<br>x, y specify the top left corner of a rectangle and x1, y1 specify the bottom right corner. x1 must be larger then x and y1 must be larger than y for the command to work. |
| N | esc{x,y,x1,y1H | | | Y | Specific Area Hit | Specify an area of the screen and this will return 1 when the screen has been touched within the specified area, it will not return otherwise. Programmatically it is better to use the previous command as this has the potential of locking up the device.<br>x, y specify the top left corner of a rectangle and x1, y1 specify the bottom right corner. x1 must be larger then x and y1 must be larger than y for the command to work. |
| | | | | | | **I2C:** The above two commands are not available in I2C as they suspend operation and an attempt to read the I2C buffer when there is nothing in it may result in an error. Use esc[l instead. |
| N | esc[l [1] | 27 53<br>(1.2ms) | | Y | Last touch | Returns the co-ordinates from the last touch position. The format is:<br>x,y,f0[ACK][x13]<br>If ACK is enabled the terminating character will be ACK otherwise it will by 0x13.<br>Where x is the x co-ordinate, y is the y co-ordinate and f is a flag that is set to 1 if the screen has been touched since the last read. This command will clear this flag back to 0. This command can also be used with the touch interrupt line, see the section on the touch screen in the text body. |
| | | | | | | **I2C:** When used with the I2C interface five bytes are returned, a 16 bit value for each of the co-ordinates and an 8 bit value for the flag:<br><X high byte><X low byte><Y high byte><Y low byte><Flag> |
| | **Graphic Commands and Pixel Drawing** | | | | | |
| | | | | | Curly brackets | When used for drawing and bitmap commands co-ordinates are specified with a non-numeric character between, for example a space or comma. The x co-ordinate is the width or horizontal which can have a value of 0-240 and the y co-ordinate is the height that can have a value 0-319. Example of valid co-ordinate specifications:<br>esc{10 10 100 100L or esc{10;10;100;100L<br>Drawing is carried out using the current foreground and background colours. Some commands have alternative |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | formats where shown. When the co-ordinates are not given the current co-ordinates are used. |
| | | | | | | **I2C:** The co-ordinates are 16 bit values, high first and so a command that requires 4 values, 8 bytes will be sent. As an example, drawing a line from 10,20 to 270,90 would require the following:<br><br>x0 = 10 which is 2 bytes 0x0 and 0x0a<br>y0 = 20 which is 2 bytes 0x0 and 0x14<br>x1 = 270 which is 2 bytes 0x01 and 0x0e<br>y1 = 90 which is 2 bytes 0x0 and 0x5a<br><br>The complete byte transfer would be (in hex) 1b 70 9 a 9 14 1 e 0 5a |
| N | esc{x0 y0 x1 y1L<br><br>esc{x1 y1L | 27 60 x0 y0 x1 y1<br><br>27 61 x1 y1 | | Y | Draws a line | Draws a line in any direction from the co-ordinates x0,y0 to x1,y1. There must be a non-numeric character between the co-ordinates, this is acceptable esc{10,10,40,40L or esc{10;10;40;40L |
| N | esc{ x0 y0 x1 y1R<br><br>esc{x1 y1R | 27 62 x0 y0 x1 y1<br><br>27 63 x1 y1 | | Y | Draws a rectangle | The co-ordinates are the top left and bottom right of the rectangle |
| N | esc{ x0 y0 x1 y1F<br><br>esc{x1 y1F | 27 64 x0 y0 x1 y1<br><br>27 65 x1 y1 | | Y | Draws a filled rectangle | As the R command but fills with foreground colour. |
| N | esc{x0 y0 radC<br><br>esc{radC | 27 66 x0 y0 r<br><br>27 67 r | | Y | Draws a circle | A circle is drawn with the given radius (rad) at the specified x and y co-ordinates. |
| | | | | | | **I2C:** The radius is a single byte. |
| N | esc{x0 y0P<br><br>esc{P | 27 68 x y<br><br>27 69 | | | Draws a pixel | A single pixel is drawn at the given co-ordinates |
| N | esc{x0 y0p<br><br>esc{p | 27 70 x y<br><br>27 71 | | | Moves to pixel | This is similar to the 'P' command but no pixel is drawn, this command will move the cursor to an exact pixel and is useful for aligning text for example. |
| N | esc{x0 y0b<br><br>esc{b | 27 72 x y <bytes><br><br>27 73 <bytes> | | Y | Bitmap | Send a bitmap picture as binary bytes. |
| N | esc{g | 27 74 | | | Returns current co- | Returns the x (horizontal) and y(vertical) co-ordinate pixel values. The format is: |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | (0.5ms) | | | ordinates | x,y<CR> |
| | | | | | | **I2C:** When the screen is being presses and there is a result, it is returned as two 16 bit values:<br>xHigh<br>xLow<br>yHigh<br>yLow |
| | **Flash memory & Macro Commands** | | | | | |
| N | esc[f | 27 80 | | Y | Flash Memory Map | Draws a graphical representation of the blocks in use. Fifty rows or 10 blocks are drawn on the screen, the red blocks are in use and the green blocks are free. It is possible (but highly unlikely) though that a block in use could be shown as green. This is because only the first byte of the block is checked. |
| N | esc[?27D | 27 81 | | | Write defaults to Flash | The default settings are stored in block 0 of the Flash user memory and can change the way the display behaves at start up. All of the variables with 'F' in the 'Def' column will be written to the EEPROM and that will be the new default at switch on. See note [2].<br>To set display back to its defaults then erase block 0 on the user flash, thus esc{0,1E |
| N | esc{<bl>q | | | Y | Query block | Queries block and returns the following information about that block:<br>'0' Block is blank Green<br>'1' Block is not blank Red<br>'b' Block is first block of a bitmap Blue<br>'m' Block is first block of a macro Yellow<br>'f' Block is first block of a font Fucha |
| N | esc[q | 27 87 | | Y | Flash size | displays flash size, number of blocks |
| N | esc{<bl>d | | | | Dump block | Dumps a block to the serial port, this is mainly for debugging to check the contents of the memory, erased memory has all 0xff. |
| | | | | | | **I2C:** This command is not available in I2C mode |
| N | esc{<bl>B | 27 82 bl <bytes> | | Y | Write bmc | Writes a 24 bit bmc file to Flash starting at the given block (1-499), the device will continue to accept and write to the flash until input stops. There is a short time out that will detect any pause and terminate the command. This does not begin until after the first character is received. A pause of approximately 100ms will cause the command to terminate. The ACK mechanism can be used to detect when this has happened.<br>The block/s must be erased first. The block size is 4096 bytes and multiple consecutive blocks will be used. |
| N | esc{<bl>M | 27 83 bl <bytes> | | | Write macro | Writes a macro starting at the given block (1-499), the device will continue to accept and write to the flash until two consecutive escape characters are received. |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | See text for more details about macro's<br>The block/s must be erased first. The maximum size of a macro is 100 bytes. |
| | | | | | | **I2C:** Start bl a 16 bit value, high byte first |
| N | esc{<bl>F | | | | Font | Writes a Font file (with a bf extension) to the Flash for later use by the display. The font file is created from a BFF file using BV-COM tools. See the user Guide. |
| N | esc[<bl>m | 27 84 start | F | | Sets macro start | It is possible to have a block run at start up before any commands are accepted. This is useful for sign on screens or indeed the whole application screen can be set up for a host to read. Any readable block can be run at start up and of course this block can call other blocks.<br>This command will set which block (1-499) is to be run at start up. To turn it off us a value of block 0, e.g. esc[0m.<br>This command must be followed by a write defaults to flash (esc[?27D) otherwise it will be lost on the next restart. |
| | | | | | | **I2C:** Start is a 16 bit value, high byte first |
| N | esc{<bl><n>E | 27 85 bl n | | A | Erase block/s | Erases blocks of flash memory starting at the given block <bl> for the number of blocks <n>. To erase one block, say block 6 the command would be esc{6,1E (or esc{6 1E). A block is 4096 bytes. The lowest block is 0 and the highest block is 499, any numbers outside of this and the command will be ignored.<br>NOTE: Erasing block 0 will reset the device back to its factory defaults. |
| | | | | | | **I2C:** bl and n are 16 bit values, high byte first |
| N | esc{<bl>r | 27 86 bl | | | Read Flash to screen | Reads the contents of flash starting at the given block and presents it to the screen. The presentation will depend on what type of data s in the starting block which in turn is determined by which command was used to write it. If a block is specified that is not a starting block an error will occur and a beep will sound. |
| | | | | | | **I2C:**bl is a 16 bit value, high byte first |

**Notes**

[1] The letter following number in the escape code is lower case L

[2] At start up (reset) block 0 of the Flash user memory is read and various parameters are stored to RAM, the parameters that are read from the Flash are indicated in the third column by being marked with an 'F'. During operation a parameter may be changed, for example font size 2 could be set. This will take immediate effect but if the device is reset the parameter will revert back to the default. To 'fix' the new value command esc[?27D will write all of the parameters in RAM to Flash thus 'fixing' their values. This command writes ALL of the parameters marked as F.