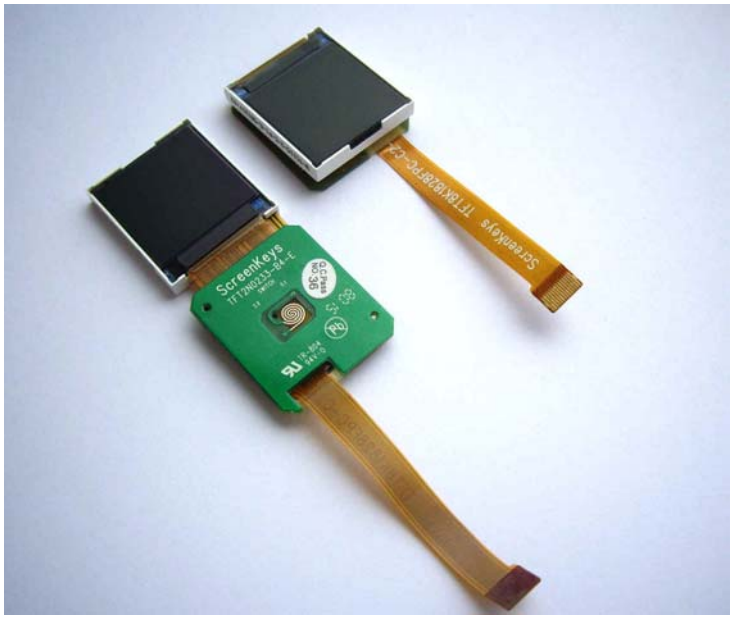
# TFT128D SCREENKEY DISPLAY DATASHEET

**Order Part Number:**

## TFT128D




**SK Interfaces Ltd.**

**Unit 11, Keypoint Business Park,**
**42 Rosemount Park Drive,**
**Ballycoolin Road,**
**Dublin 11, Ireland.**

**Tel: +353-1-8855 075**
**Fax: +353-1-8855 095**


**Please visit our website www.screenkeys.com for additional information.**



Date Issued: May 2010

# Version Control

| *Version* | *Date* | *Notes* |
|---|---|---|
| 1.0 | 28 April, 2009 | 1.0 release of product |
| 1.1 | 4 October, 2009 | Changed graphic image commands (21/22/23) to include wipe direction for painting<br>Added new Cmd 27 for RLE8 images<br>Included information on version number display<br>Added SPI timing information and diagrams<br>Corrected typographic errors |
| 1.2 | 16 May , 2010 | Added additional SPI timing information<br>Added new Cmd 02 to define video sub-window<br>Change to startup mode of backlight (off on startup)<br>Added new Cmd 37 to report image download checksum |

# Contents

# 1. Technical Description

The TFT128D ScreenKey Display is a full-color high resolution TFT graphics display. The display is backlit with a white LED and the TFT liquid crystal display has a resolution of 128*128 pixels where each pixel color can be chosen from a palette of 65,536 different colors.

The ScreenKey Display is an intelligent device with an onboard microcontroller and graphical display controller. The TFT128D fully implements a 4-wire SPI interface with an expanded command set allowing simple ASCII control or full bitmap graphic display, or any combination of the two. The display supports downloadable content and is designed to minimize the bandwidth requirements from a host controller, particularly when used in a multi-ScreenKey environment.

The ScreenKey supports two operating modes:

- *Command mode* offers commands to simplify application integration by using text display, clear display, flash, etc. This mode reduces the maximum frame-rate when transmitting graphic images into the display.
- *High-speed mode* disables all the command processing and implements a simple graphical only interface. This mode delivers approx 29 frames per second with a video sub-window sized at 96*54 pixels and just over 9 frames per second when using the full 128*128 pixels.

A flex-tail cable is used to make connection to the target electronics via a PCB connector.

# 2. Typical Uses and Application

The multi-function TFT128D ScreenKey Display, with its high resolution LCD display and 65,536 color support, is suited for any application requiring a color display and is ideal for many different markets and applications where graphical or text output is required, including:

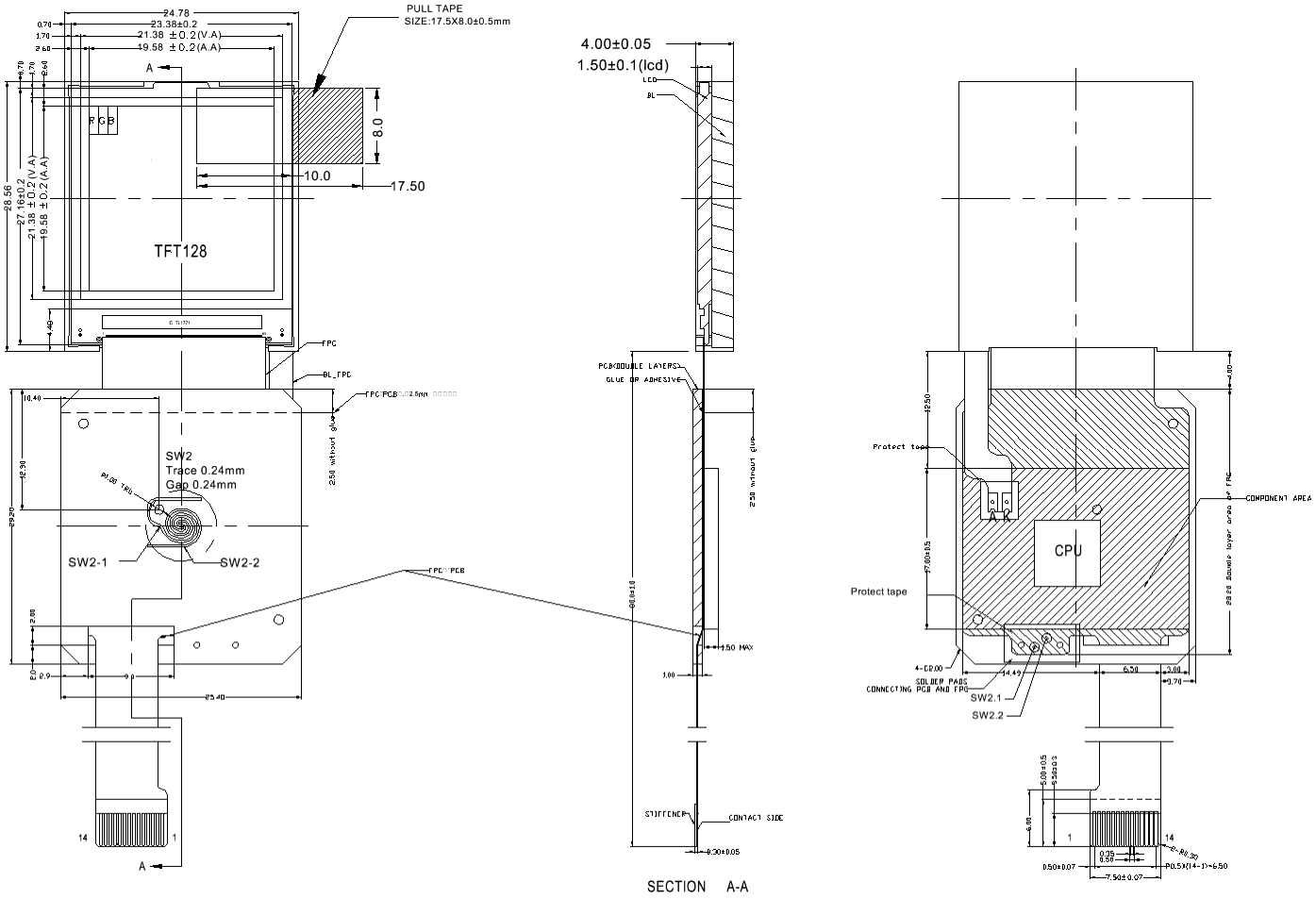
- Media and Broadcasting
- Audio/Visual Studio and Production Equipment
- Industrial controls
- Point-of-Sale, Point-of-Information
- Medical Devices
- Automotive Industry
- Aerospace
- Financial Services / Stock Trading
- Air Traffic Control
- Telecommunications
  etc.

The advantages of ScreenKey Display are that they are simple to integrate into hardware, and software control of the LCD is very simple and straightforward. This allows for the easy integration of the display into products without extensive development efforts.

- Industry standard SPI interface
- 3.3V logic and power supply
- No external display controller required
- Greater signal effectiveness through full color display
- Display text and/or graphics with 128*128 pixel resolution
- Fast data transmission with SPI transmission rates up to 10MHz
- Easy and fast integration with high-level commands
- Displayed image is maintained as long as power is applied (no refresh required)
- Excellent display with high contrast and wide viewing angles
- Landscape or portrait orientation
- Keyswitch lifetime of >3 million operations
- Intuitive user guidance through menu systems

# 3. Dimensions

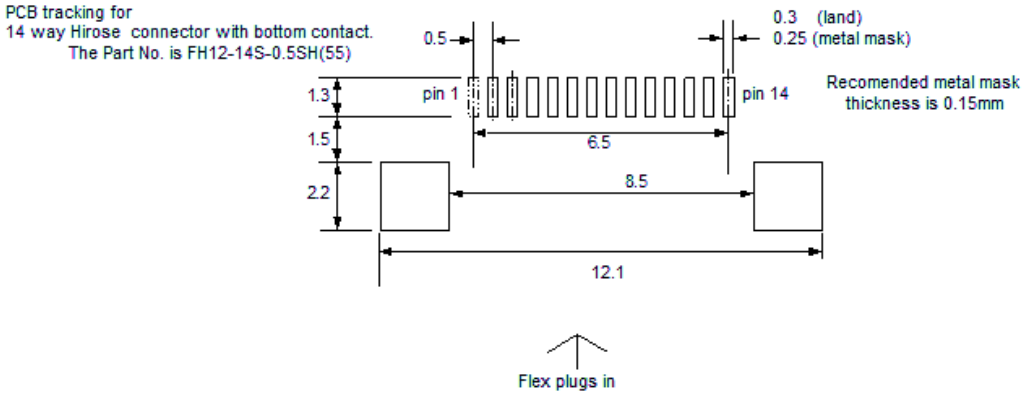
The display may be fitted either flat (TFT and PCB in same plane) or with the TFT folded over onto the PCB – the TFT and PCB do **not** click together.




Drawing: Dimensions of TFT128D ScreenKey Display in mm (all dimensions +/-0.2mm).

## 3.1. FPC Connector PCB Tracking

The diagram below shows the recommended tracking for the FPC connector (if using the recommended FPC connector – see section 3.2 below).




Drawing: PCB tracking layout for recommended FPC connector

## 3.2. Recommended Flex-tail Connector

The flex-tail cable used in the TFT128D is a 14-way 0.5mm pitch single-sided gold contact.

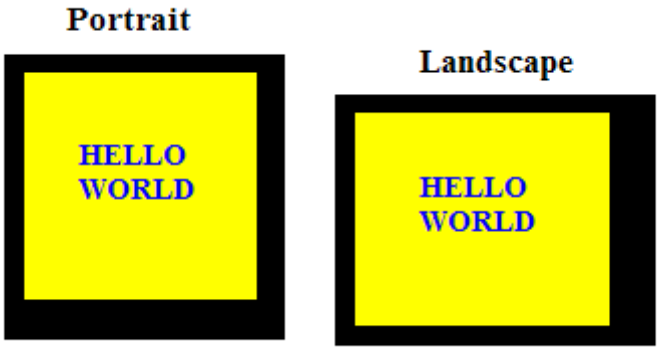
The recommended FPC connector for use with the TFT128D is a ZIF bottom contact connector:

**Hirose (HRS) FH12-14S-0.5SH(55)**

*Note:*
*The TFT128D is **not** supplied with a flex-tail connector as standard.*

## 3.3. Orientation

The TFT128D can be mounted either in portrait (vertically) or horizontally (landscape), depending on the user and application preference.

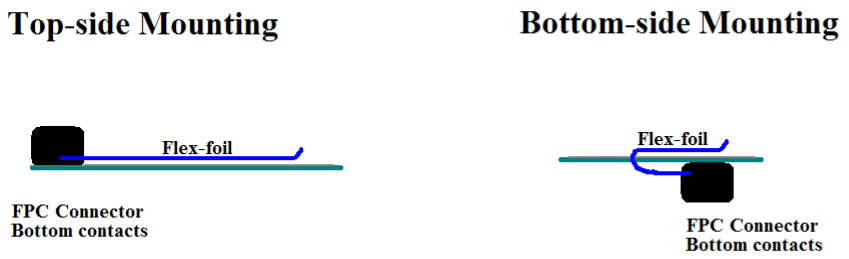



Drawing: TFT128 may be mounted either in portrait or landscape orientation

## 3.4. FPC Connector Mounting

The TFT128D electrically connects to a PCB via a flex-tail connector.

The flex-tail connector may be fitted to the mounting PCB in one of two ways:




Drawing: Flex-tail cable path to connector

1. The flex-tail cable may be fed through a slot in the PCB to a connector mounted on the underside of the PCB. This may be useful when multiple displays are fitted in close proximity.

   *Note that the flex-tail gold fingers will be on the **bottom** of the cable as it enters the connector in this configuration.*

2. An alternative method of mounting is to place the flex-tail connector on the same PCB side as the ScreenKey itself. This may not be suitable if a ScreenKey Display grid layout is required.

   *Note that the flex-tail gold fingers will be on the **bottom** of the cable as it enters the connector in this configuration.*

# 4. ScreenKey Display Specifications

Dimensions:

| ***Description*** | ***Values*** |
|---|---|
| Unfolded excl Flextail (L x W) | 61.8 x 25.4 mm |
| Unfolded incl Flextail) (L x W ) | 105.6 x 25.4 mm |
| Display section only (L x W) | 28.6 x 24.8 mm |
| PCB section only (LxW) | 29.2 x 25.4 mm |
| Flextail between display & PCB | 4mm |
| Felxtail extension beyond PCB | 46.8mm |

Environmental:

| ***Description*** | ***Values*** |
|---|---|
| Operating Temperature | -20° ….. +70° Celsius |
| Storage Temperature | -30° ….. +80° Celsius |
| Humidity | max. 90% relative at 60° Celsius |
| Life Cycle | 8+ years (life cycle may be reduced by exposure to excess humidity, temperature and ultra-violet light) |

Electrical:

| ***Description*** | ***Values*** |
|---|---|
| Operating voltage (Vcc) | 3.0v – 3.3V dc |
| Max supply voltage | 3.6V |
| Max input voltage | Vcc + 0.5V |
| Current Consumption | 27mA (typical) |
| Logic Input Low (max) | 0.2*Vcc – 0.1V |
| Logic Input High (min) | 0.7*Vcc |

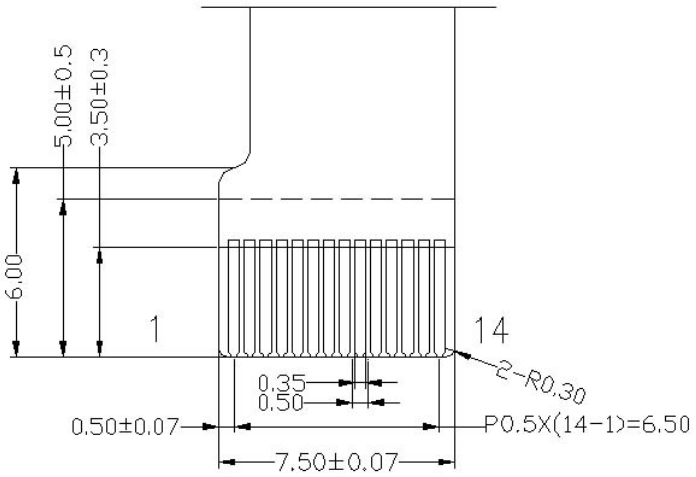
Display:

| ***Description*** | ***Values*** |
|---|---|
| LCD Glass | Thin Film Transistor (TFT) / Transmissive |
| Pixel Field Size ( X x Y) | 19.584 x 19.584 mm |
| Pixel Matrix ( X x Y ) | 128 (RGB) x 128 |
| Viewing Direction | 6 o'clock |
| Viewing Angle | 45 deg (min) |
| Response Times (typical) | 15ms "white-to-black" / 35ms "black-to-white" |
| Contrast Ratio | Min 120 / Typ 170 (white/black) |
| Backlight | White LED (half-life of 50,000 hours) |

# 5. Electrical Interface

Electrical connection is made to the TFT128D via the 14-way flex-tail cable.

| | |
|---|---|
| 1 | Vcc |
| 2 | Vcc – LED |
| 3 | GND – LED |
| 4 | D- (not used) |
| 5 | D+ (not used) |
| 6 | GND |
| 7 | MOSI |
| 8 | SCK |
| 9 | MISO |
| 10 | SS |
| 11 | Not connected |
| 12 | SW-DETECT (not used) |
| 13 | SW-1 (not used) |
| 14 | SW-2 (not used) |




Drawing: Flex-tail cable connections and orientation

The supply voltage (Vcc) is 3.0V to 3.3Vdc. Contacts 3 and 6 must **both** be connected to the supply 0V. Contact 1 must be connected to Vcc. Contact 2 controls the +V supply to the backlight LED (see section 6.2 below for details on how the backlight LED may be controlled).

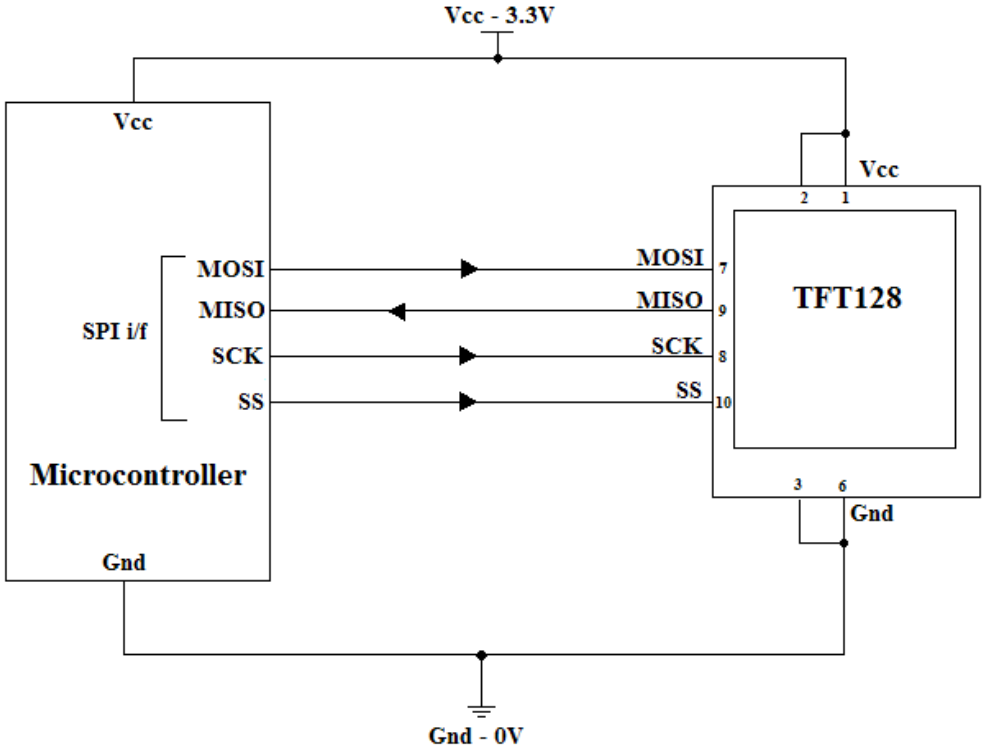
MOSI/SCK/MISO and SS are the 4-wire SPI interface.

SW-1, SW-2 and SW_DETECT are switch contacts which are not used in this product. These should be left unconnected.

D+/D- are a USB interface that can be used to update the internal firmware in the ScreenKey. A USB user interface may be made available in the future using these pins. These pins should **not** be connected.

## 5.1. Single-Display Wiring Diagram

The following circuit provides an example of how to connect a single TFT128D ScreenKey Display to a microcontroller with external switch monitoring:




## 5.2. Backlight Control Options

The TFT128D incorporates an internal white LED backlight. Power to this backlight is supplied via the flex-tail contacts 2 (+Vcc) and 3 (Gnd).

Typically, contact 2 should be directly connected to the main supply Vcc on contact position 1. The backlight Gnd contact (position 3) **must** always be connected to the same Gnd plane as the main supply, e.g. contact position 6.

The TFT128D allows users to adjust the backlight brightness via a SPI command when used in high-level command mode.

However, when used in high-speed mode the backlight cannot be controlled via software and is set to be always on. In this mode, it is possible for users to apply external hardware control of the backlight via contact 2 (Vcc – LED). Applying 3.3V sets the maximum brightness level. The brightness can be dimmed by reducing applied voltage downwards.

# 6. Bitmap Graphics

The TFT128D display supports bitmapped graphics where each pixel can display any color from a 65,536 color palette.

To define a single pixel, a two byte word is used where red and blue are specified by 5 bits each and green is specified using 6 bits:

| R | R | R | R | R | G | G | G | G | G | G | B | B | B | B | B |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Red | | | | | Green | | | | | | Blue | | | | |
| CH (MSB byte) | | | | | | | | CL (LSB byte) | | | | | | | |

For example,

| | |
|---|---|
| F800 hex | Red |
| 07E0 hex | Green |
| 001F hex | Blue |

The display resolution is 128*128 pixels, i.e. 16,384 pixels. Using 2 bytes per pixel to define each color requires 32,768 bytes to display one full bitmap on a TFT128D ScreenKey Display. Clearly this requires significant bandwidth and ways to reduce this requirement are used wherever possible.

For ease of use and reduced processing overheads, the TFT128D also permits graphic updates using a form of the Windows BMP 256-color graphic format. This uses a 256 color palette where each pixel is represented by a single byte reference or index into this palette. This allows a full bitmap image to be defined using 16,384 bytes with a color palette table of 512 bytes (256 colors * 2 bytes). To further reduce bandwidth and storage requirements, the TFT128 also supports 8-bit RLE compression of Windows BMP-type image data.

It is possible to download graphics to the TFT128D where full 16-bit color information is provided per pixel. Similarly, 256-color palletized (including RLE compressed) images can also be downloaded into the TFT128D onboard memory.

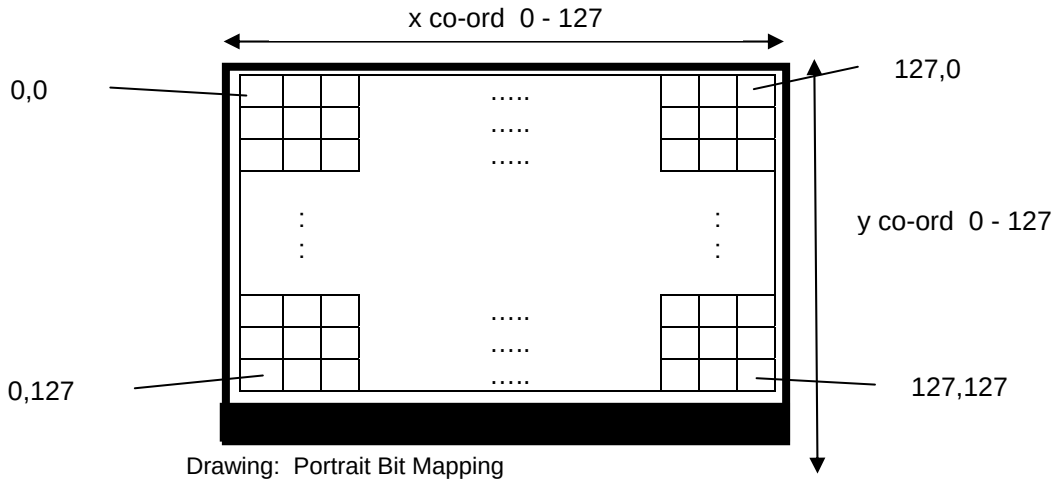
All graphic commands and images are designed or positioned using an x,y co-ordinate system (see below). The ‘x’ co-ordinate is always the horizontal direction from the users perspective, referenced at 0 from the top left pixel, whether the key is in portrait or landscape mode. The ‘y’ co-ordinate is the vertical direction also referenced to 0 at the top left pixel position.

To better support Windows BMP images, you can specify the “wipe” direction for images. Windows BMP files usually store the bitmap data from the bottom left pixel through to the top left pixel (moving left to right and bottom to top). The TFT128D supports this BMP-style data presentation by simply stating the wipe direction in the image header.

## 6.1. Portrait Bit-mapping

The visual top left corner of the display is referenced as position x=0, y=0 (0,0), where x is the horizontal direction and y is the vertical direction.

The display has 128 pixels horizontally and 128 pixels vertically.

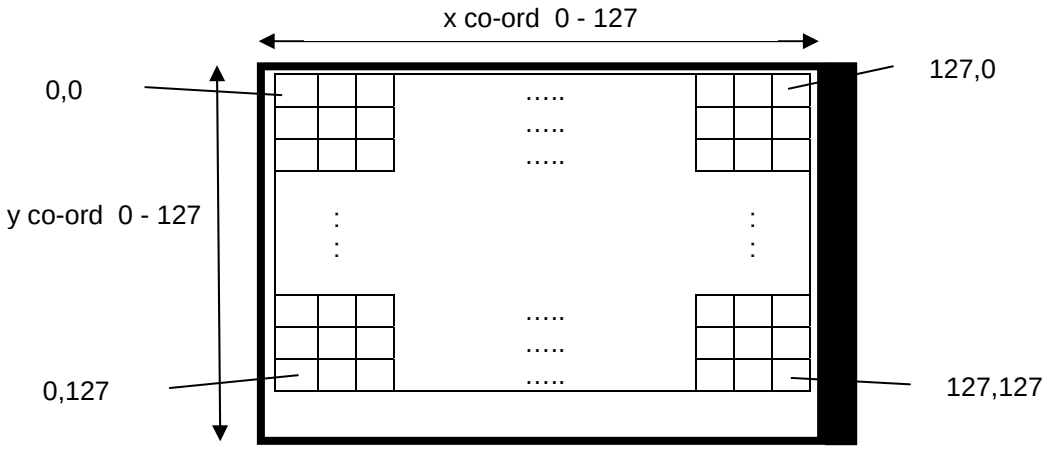



Drawing: Portrait Bit Mapping

## 6.2. Landscape Bit-mapping

In landscape mode, the visual top left corner of the display is referenced as position x=0, y=0 (0,0), where x is the horizontal direction and y is the vertical direction.

The display has 128 pixels horizontally and 128 pixels vertically.




Drawing: Landscape Bit Mapping

## *6.3. RLE Compression*

The TFT128 supports RLE8 image data compression. This is a simple compression scheme which is often used with 256-color Windows BMP images.

RLE compression recognises that many pixel color values are repeated sequentially in a display row. Instead of storing the color value (palette reference) separately for each pixel, an “encoded run” is stored which uses just two bytes. The first byte stores the count of the number of pixels in the run, and the second byte contains the color palette reference (pixel color). Runs of up to 128 identical pixel values may be encoded as only two bytes of data (The 128 limit is determined by the maximum line size of the TFT128).

In addition to encoded runs, there are unencoded runs, delta markers, end-of-scan-line markers, and an end-of-RLE-data marker.

The 8-bit RLE algorithm (RLE8) stores repeating pixel values as encoded runs. The first byte of an encoded run will be in the range of 1 to 255. The second byte is the value of the 8-bit pixels in the run. For example, an encoded run of 05 18 would decode into five pixels each with the value 18, or 18 18 18 18 18.

When a scan line does not contain enough pixel runs to achieve a significant amount of compression, contiguous pixel values may be stored as literal or unencoded runs. An unencoded run may contain from 3 to 128 pixel values. The first byte of an unencoded run is always zero. This makes it possible to tell the difference between the start of an encoded and the start of an enencoded run. The second byte value is the number of unencoded pixel values that follow. If the number of pixels is odd, then a 00 padding value must also follow.

Here are some examples of encoded and unencoded data streams:

| **Encoded Bytes** | **Decoded Bytes** |
|---|---|
| 05 10 | 10 10 10 10 10 |
| 00 05 23 65 34 56 45 00 | 23 65 34 56 45 |
| 0A 0A | 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A |
| 00 04 46 57 68 79 | 46 57 68 79 |

Three marker values may also be found in the RLE data. Each of these markers begins with a zero-byte value. The second byte value indicates the type of marker. These markers specify positional information relating to the decoded bitmap data and do not generate any data themselves. The first marker is the end-of-scan-line marker and is identified by two byte values 00 and 00. This marker is an indication that the end of data for the current scan line has been reached. Encoded data occurring after this marker is decoded starting at the beginning of the next scan line. If an end-of-scan-line marker is not present in the encoded data, then the pixels will automatically wrap from the end of one scan line to the start of the next. The next marker is the end of RLE data marker. It is identified by the two byte values 00 and 01. This marker occurs only as the last two bytes of the RLE data. This marker is an indication that the reader should stop decoding data.

The final marker byte is called a Delta vector. This marker is identified by the two byte value 00 and 02. It is followed by a two byte offset position (x and y) which indicates where the next pixels should be written/displayed. This marker is **not** supported by the TFT128 and is rarely found in Windows BMP RLE-compressed images.

## 6.4. Factory Default 256-Color Palette

When using 256-color palletized images, the actual 16-bit color value written to the pixel is taken from a look-up table or palette. The table b elow shows the on-board factory default palette. This palette matches the Windows 8-bit palette used in Windows 256-color BMP files. For example, Adobe PhotoShop allows images to be stored as “indexed color”, 8-bit colors with or without RLE compression. One of the palettes offered for use is defined as the “Windows” palette (as below).

| *Color Ref* | *16-bit Color Ref 5-6-5 (5 bit red/6 bit green/5 bit blue)* | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **0 - 7** | 0x0000 | 0x7800 | 0x03e0 | 0x7be0 | 0x000f | 0x780f | 0x03ef | 0x7bef |
| **8-15** | 0xbed7 | 0xa63d | 0x29f4 | 0x29ff | 0x2ae0 | 0x2aea | 0x2af4 | 0x2aff |
| **16-23** | 0x2be0 | 0x2bea | 0x2bf4 | 0x2bff | 0x2ce0 | 0x2cea | 0x2cf4 | 0x2cff |
| **24-31** | 0x2de0 | 0x2dea | 0x2df4 | 0x2dff | 0x2ee0 | 0x2eea | 0x2ef4 | 0x2eff |
| **32-39** | 0x2fe0 | 0x2fea | 0x2ff4 | 0x2fff | 0x5000 | 0x500a | 0x5014 | 0x501f |
| **40 – 47** | 0x50e0 | 0x50ea | 0x50f4 | 0x50ff | 0x51e0 | 0x51ea | 0x51f4 | 0x51ff |
| **48 – 55** | 0x52e0 | 0x52ea | 0x52f4 | 0x52ff | 0x53e0 | 0x53ea | 0x53f4 | 0x53ff |
| **56 – 63** | 0x54e0 | 0x54ea | 0x54f4 | 0x54ff | 0x55e0 | 0x55ea | 0x55f4 | 0x55ff |
| **64 – 71** | 0x56e0 | 0x56ea | 0x56f4 | 0x56ff | 0x57e0 | 0x57ea | 0x57f4 | 0x57ff |
| **72 – 79** | 0x7800 | 0x780a | 0x7814 | 0x781f | 0x78e0 | 0x78ea | 0x78f4 | 0x78ff |
| **80 – 87** | 0x79e0 | 0x79ea | 0x79f4 | 0x79ff | 0x7ae0 | 0x7aea | 0x7af4 | 0x7aff |
| **88 – 95** | 0x7be0 | 0x7bea | 0x7bf4 | 0x7bff | 0x7ce0 | 0x7cea | 0x7cf4 | 0x7cff |
| **96 – 103** | 0x7de0 | 0x7dea | 0x7df4 | 0x7dff | 0x7ee0 | 0x7eea | 0x7ef4 | 0x7eff |
| **104 – 111** | 0x7fe0 | 0x7fea | 0x7ff4 | 0x7fff | 0xa000 | 0xa00a | 0xa014 | 0xa01f |
| **112 – 119** | 0xa0e0 | 0xa0ea | 0xa0f4 | 0xa0ff | 0xa1e0 | 0xa1ea | 0xa1f4 | 0xa1ff |
| **120 – 127** | 0xa2e0 | 0xa2ea | 0xa2f4 | 0xa2ff | 0xa3e0 | 0xa3ea | 0xa3f4 | 0xa3ff |
| **128 – 135** | 0xa4e0 | 0xa4ea | 0xa4f4 | 0xa4ff | 0xa5e0 | 0xa5ea | 0xa5f4 | 0xa5ff |
| **136 – 143** | 0xa6e0 | 0xa6ea | 0xa6f4 | 0xa6ff | 0xa7e0 | 0xa7ea | 0xa7f4 | 0xa7ff |
| **144 – 151** | 0xc800 | 0xc80a | 0xc814 | 0xc81f | 0xc8e0 | 0xc8ea | 0xc8f4 | 0xc8ff |
| **152 – 159** | 0xc9e0 | 0xc9ea | 0xc9f4 | 0xc9ff | 0xcae0 | 0xcaea | 0xcaf4 | 0xcaff |
| **160 – 167** | 0xcbe0 | 0xcbea | 0xcbf4 | 0xcbff | 0xcce0 | 0xccea | 0xccf4 | 0xccff |
| **168 – 175** | 0xcde0 | 0xcdea | 0xcdf4 | 0xcdff | 0xcee0 | 0xceea | 0xcef4 | 0xceff |
| **176 – 183** | 0xcfe0 | 0xcfea | 0xcff4 | 0xcfff | 0xf80a | 0xf814 | 0xf8e0 | 0xf8ea |
| **184 – 191** | 0xf8f4 | 0xf8ff | 0xf9e0 | 0xf9ea | 0xf9f4 | 0xf9ff | 0xfae0 | 0xfaea |
| **192 – 199** | 0xfaf4 | 0xfaff | 0xfbe0 | 0xfbea | 0xfbf4 | 0xfbff | 0xfce0 | 0xfcea |
| **200 – 207** | 0xfcf4 | 0xfcff | 0xfde0 | 0xfdea | 0xfdf4 | 0xfdff | 0xfee0 | 0xfeea |
| **208 – 215** | 0xfef4 | 0xfeff | 0xffea | 0xfff4 | 0xc65f | 0xfe5f | 0x37ff | 0x67ff |
| **216 – 223** | 0x97ff | 0xc7ff | 0x03e0 | 0x03ea | 0x03f4 | 0x03ff | 0x04e0 | 0x04ea |
| **224 – 231** | 0x04f4 | 0x04ff | 0x05e0 | 0x05ea | 0x05f4 | 0x05ff | 0x06e0 | 0x06ea |
| **232 – 239** | 0x06f4 | 0x06ff | 0x07ea | 0x07f4 | 0x2800 | 0x280a | 0x2814 | 0x281f |
| **240 – 247** | 0x28e0 | 0x28ea | 0x28f4 | 0x28ff | 0x29e0 | 0x29ea | 0xffdd | 0x9cf3 |
| **248 – 255** | 0x7bef | 0xf800 | 0x07e0 | 0xffe0 | 0x001f | 0xf81f | 0x07ff | 0xffff |

# 7. Text Display

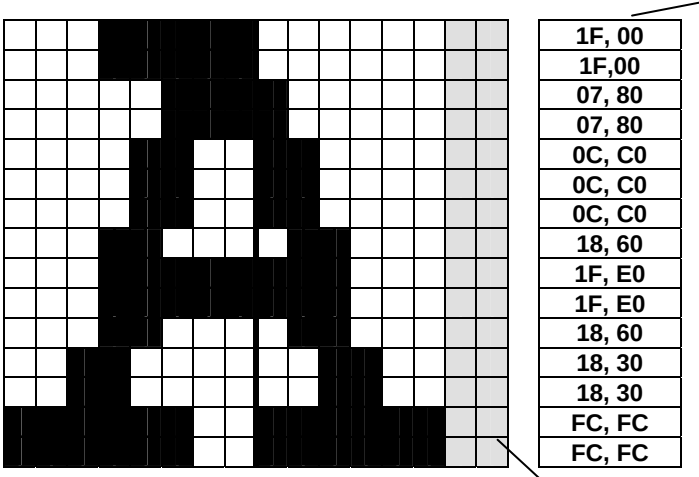
To enable fast integration and to lower bandwidth requirements, the TFT128D ScreenKey supports ASCII text display from simple commands (in *high-level command mode* only). For example, an application may send the ASCII string “Hello World” and the TFT128D will handle all font and character manipulations to generate the bit-mapped graphic image to represent this string.

The TFT128D incorporates a factory default 256 character fixed size raster font. The font is a standard “Courier” style pattern and uses a 14 column x 15 row pixel grid for each character.

The TFT128D maintains a *cursor position* which is initially set to 0,0. This relates to the top left pixel of the character to be displayed. After each font character is displayed, a set number of pixels are skipped for the inter-character spacing (inter-character spacing is set to zero pixels for the factory default font).

As characters are drawn horizontally across the screen, the TFT128D continually monitors the remaining horizontal space. If there is insufficient pixels to draw a new character (14 pixels required for the default factory font) then the cursor position is mapped to the left most position on the next character line below. A pre-set number of pixels is skipped between lines and this is called the inter-line spacing. The factory default font’s inter-line spacing is set to three pixels.

For example, the factory default “A” character is defined as follows:

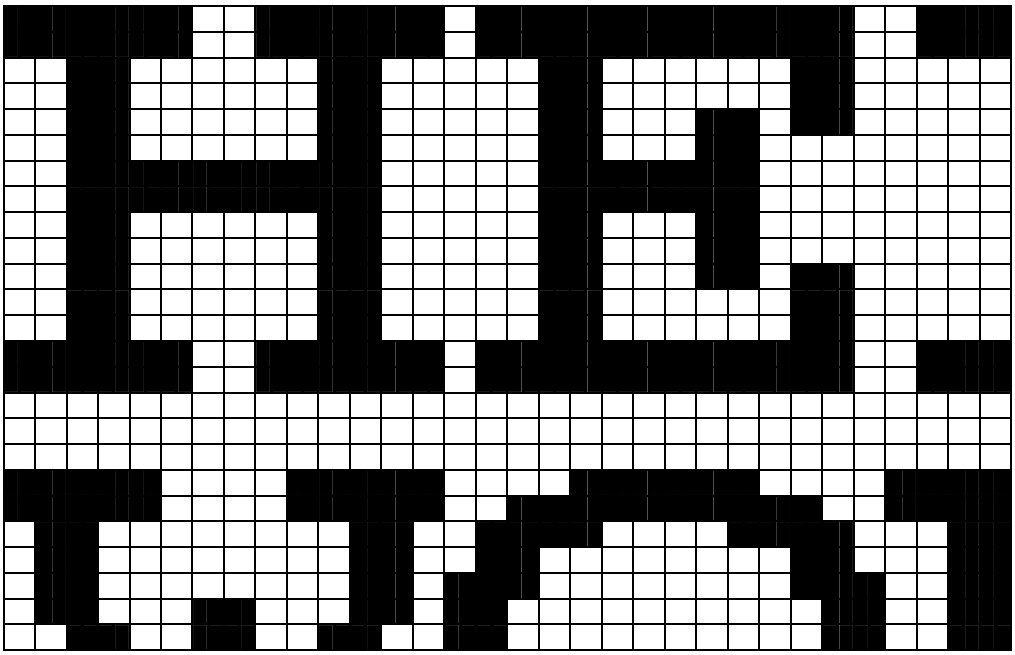



Factory default font:

| | |
|---|---|
| 2 | bytes per row |
| 14 | pixels per row |
| 30 | bytes per character |
| 0 | pixels inter-character spacing |
| 3 | pixels inter-line spacing |
| 256 | number of characters in font |
| 0 | first character offset |

The drawing below describes how the text strings are placed on the TFT display (using factory default font).

The string “HELLO” is requested on the first character line of the display and the text string “WORLD” on the second character line of the display.



The 3 pixel inter-line spacing is shown in the gap between the bottom row of pixels in the first line of characters and the top row of pixels in the second line of characters.

The factory default font has a zero inter-character spacing. This is because the majority of the character designs leave the first (leftmost) column of pixels clear, i.e. a default one pixel inter-character spacing is built into the 14x15 grid. There are some characters which do not leave the first column clear, i.e. the extra wide characters such as W, H, K, M etc. It is possible that these characters will ‘touch’ the character directly to their left. This is accepted in order to fit more characters per line.

## *7.1. Factory Default Character Set*

The character set implemented by the factory default font is based primarily on the ISO 8859-15 (Latin alphabet no. 9) character set:

| Low nibble \ High nibble | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | | | | 0 | @ | P | ` | p | ─ | ═ | | ° | À | Đ | à | ð |
| **1** | | | ! | 1 | A | Q | a | q | │ | ║ | ¡ | ± | Á | Ñ | á | ñ |
| **2** | | | " | 2 | B | R | b | r | ┌ | ╔ | ¢ | ² | Â | Ò | â | ò |
| **3** | | | # | 3 | C | S | c | s | ┐ | ╗ | £ | ³ | Ã | Ó | ã | ó |
| **4** | | | $ | 4 | D | T | d | t | └ | ╚ | € | Ž | Ä | Ô | ä | ô |
| **5** | | | % | 5 | E | U | e | u | ┘ | ╝ | ¥ | µ | Å | Õ | å | õ |
| **6** | | | & | 6 | F | V | f | v | ├ | ╠ | Š | ¶ | Æ | Ö | æ | ö |
| **7** | | | ' | 7 | G | W | g | w | ┤ | ╣ | § | · | Ç | × | ç | ÷ |
| **8** | | | ( | 8 | H | X | h | x | ┬ | ╦ | š | ž | È | Ø | è | ø |
| **9** | | | ) | 9 | I | Y | i | y | ┴ | ╩ | © | ¹ | É | Ù | é | ù |
| **A** | | | * | : | J | Z | j | z | ┼ | ╬ | ª | º | Ê | Ú | ê | ú |
| **B** | | | + | ; | K | [ | k | { | ▲ | ☺ | « | » | Ë | Û | ë | û |
| **C** | | | , | < | L | \ | l | \| | ► | ☼ | ¬ | Œ | Ì | Ü | ì | ü |
| **D** | | | - | = | M | ] | m | } | ▼ | ♀ | | œ | Í | Ý | í | ý |
| **E** | | | . | > | N | ^ | n | ~ | ◄ | ♂ | ® | Ÿ | Î | Þ | î | þ |
| **F** | | | / | ? | O | _ | o | | ✖ | ✓ | ¯ | ¿ | Ï | ß | ï | ÿ |

Additional characters have been inserted into the "*C1 Control Code*" space, i.e. positions 80 hex through to 9F hex. These additional characters are useful to easily construct simple images using text commands.

There are no printable characters in positions 0 through 1F hex. User defined downloadable fonts may use this area to embed specific character designs as desired.

## 7.2. Downloadable Fonts

The TFT128D ScreenKey Display supports user downloadable fonts (see section below on High-Level Command Set).

When a new font is downloaded into the TFT128D memory the factory default font is deleted and cannot be recovered. The factory default font may be re-established by downloading it again via the interface command set.

When designing downloadable fonts the following information must be provided:

*Number of characters in font*

The default factory font has 256 characters defined. To save space, sometimes character sets of 128 characters are used by leaving out the upper 128 characters. To further reserve space, the font may be designed as a reduced number of characters, e.g. 96 or even less. The only requirement is that all characters must be contiguous in the character set.

*Offset to first character*

When a reduced size font is used (i.e. less than 256 characters) then the offset to the first font is required. This is useful if the first number of characters is not required, e.g. in printer fonts the first printable character is usually "space" which begins at position 32. To save space, the first 32 characters may be discarded in which case this offset is required to identify the starting position of the font.

*Inter-line spacing*

Displayed text becomes difficult to read if there is insufficient spacing between characters and lines. This value defines the gap in pixels to use between the bottom of one line of text and the top row of the next line of text.

*Inter-character spacing*

This value defines the automatic gap to place between characters horizontally. As demonstrated with the factory default font, this gap may be created in the design of the character layouts and so may be set to zero.

*Bytes per character*

Each font is simply a contiguous collection of data bytes. This value defines the number of bytes that are assigned to one character in the font. The total size of the font must be the total of "*bytes per character" X "number of characters in font*".

*Bytes per row*

This value defines the number of bytes that is used to define the pixels for a single row for a character in this font. The total number of rows for each character is *"bytes per character"* divided by the *"bytes per row"*.

*Pixels per row*

Not every bit in the data bytes for a row are used, e.g. the factory default font uses 14 bits. This value defines the number of used bits and begins with the MSB bit of the first data byte.

*Note:*

*The TFT128D does not implement a specific baseline for handling characters which extend below the baseline, e.g. "g", "j", etc. If required, this must be handled in the inherent font design.*

## 7.3. Reserved Font Memory

Downloadable fonts are very flexible and can vary in size and content based on the parameters described above. The only limitation is the maximum physical memory size of the font.

The TFT128D reserves a memory block of 8kbytes for font usage. Downloadable fonts cannot exceed 8,196 bytes in size (contiguous data bytes).

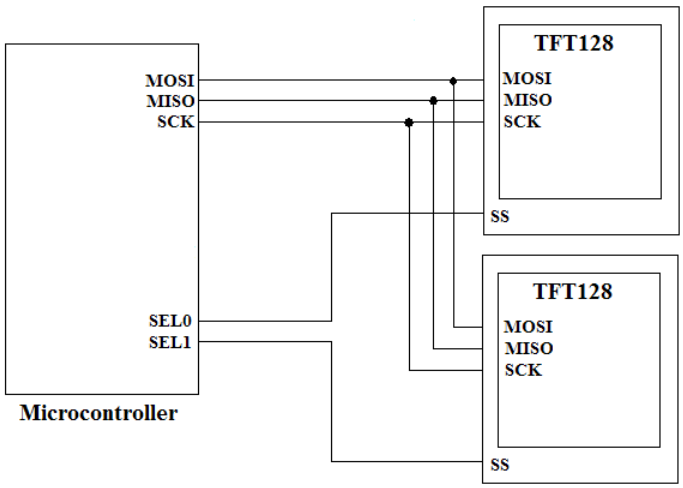
# 8. Serial Peripheral Interface

The TFT128D operates as a slave device on a Serial Peripheral Interface bus (SPI). It supports full-duplex, synchronous, serial communication with a master device at clock frequencies up to 10 MHz.

Four SPI interface signals are used:

- MOSI Master Out Slave In (data IN to ScreenKey)
- MISO Master In Slave Out (data OUT from ScreenKey)
- SCK Clock (controlled by Master)
- SS Slave Select (ScreenKey select line)

Multiple ScreenKey Display devices can be connected to a SPI bus as follows:




In this configuration, the MOSI, MISO and SCK signals are common to all devices. A unique SS line is required from the master to individually select one ScreenKey at a time.

All data is transmitted as 8-bit words with the most significant bit (MSB) transmitted first and the least significant bit (LSB) transmitted last.

The master device controls the clock. The TFT128D ScreenKey Display supports clock frequencies up to a maximum of 10MHz.

The TFT128D is a complex device supporting multiple high-level commands. Each command has a different execution time and/or different data throughputs. To support these differences, the SPI Master must listen to the return data path via MISO, to know when the slave is busy and when a command has been fully executed (unless in *high-speed mode*). Additionally, the ScreenKey has a maximum data throughput that requires the SPI Master to implement an inter-byte delay.

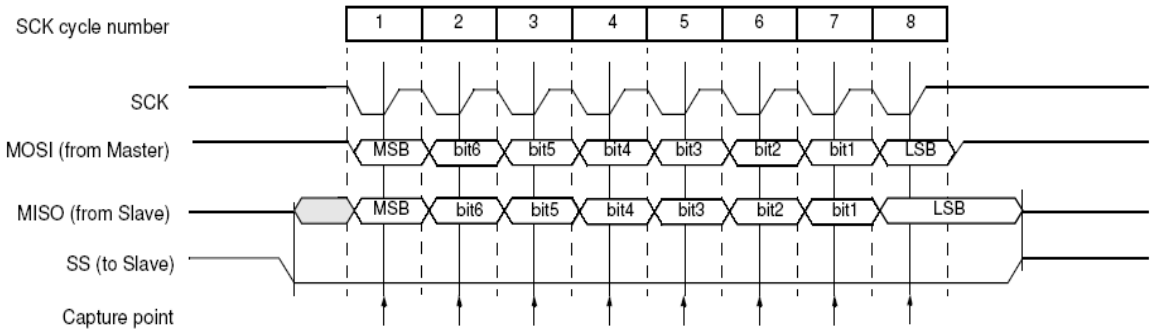
A minimum inter-byte delay is recommended to achieve maximum throughput for all high-level commands. If using 256-color graphic commands, a frame refresh rate of approximately four frames per second is achieved with these values.

A high-speed mode is offered in which the there is no data returned on MISO and the inter-byte delay is reduced significantly. This mode enables approximately ten frames per second for full color (16-bit) display. High-speed mode only provides a direct paint feature and does not allow access to the high-level commands such as text drawing, image downloads, etc. **In this mode, the MISO line may be ignored and can be left disconnected if preferred.**

## 8.1. SPI Transmission

The bus is controlled by the master device. The ScreenKey should first be selected (SS=0) by the master before transmission commences. Only one ScreenKey should be selected at a time, otherwise bus conflicts may occur on the MISO line causing data corruption.

Data transmission is described in the drawing below:




Drawing: SPI bus transmission format

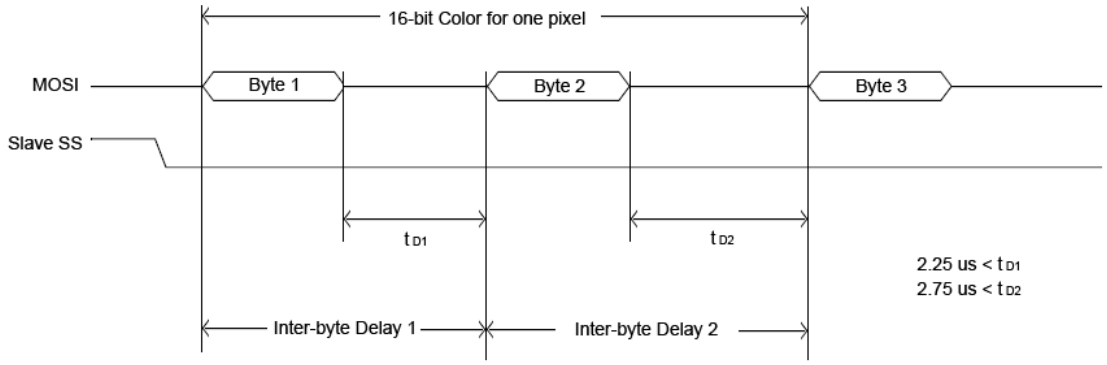
The TFT128D ScreenKey implements “mode 3” of the SPI transmission format, i.e. clock polarity (CPOL) high and clock phase (CPHA) high. The idle state of the clock line (SCK) is high and the data is sampled by the ScreenKey when the clock returns to the idle state.

To transmit a byte from the master to a ScreenKey, first the master ensures the SCK line is in its idle state (high). Next the master selects the ScreenKey by driving SS low. The clock transitions from high to low, and the master drives the MSB of the data byte on the MOSI line. At the same time the ScreenKey drives the MSB of its return data byte on the MISO line. The master then transitions the clock line back to idle (low to high) and the data is sampled by the ScreenKey and master on the MOSI and MISO lines respectively.

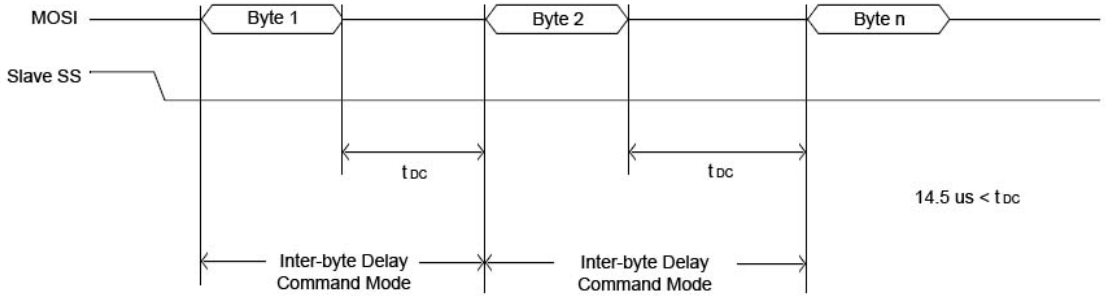
This is repeated for the remaining bits through to the LSB. Once all eight bits have been sent/received, the clock line remains at idle and the master implements an inter-byte delay before sending the next byte. A full command packet or frame should be sent before the master deselects the ScreenKey by driving SS high.

## *8.2. Multiple Byte Transmission*

When sending a package of information, i.e. multiple bytes, the master may leave the SS line low and continue to clock successive data words while observing the inter-byte delay (see diagrams below). The inter-byte delay differs between command mode and high-speed mode.



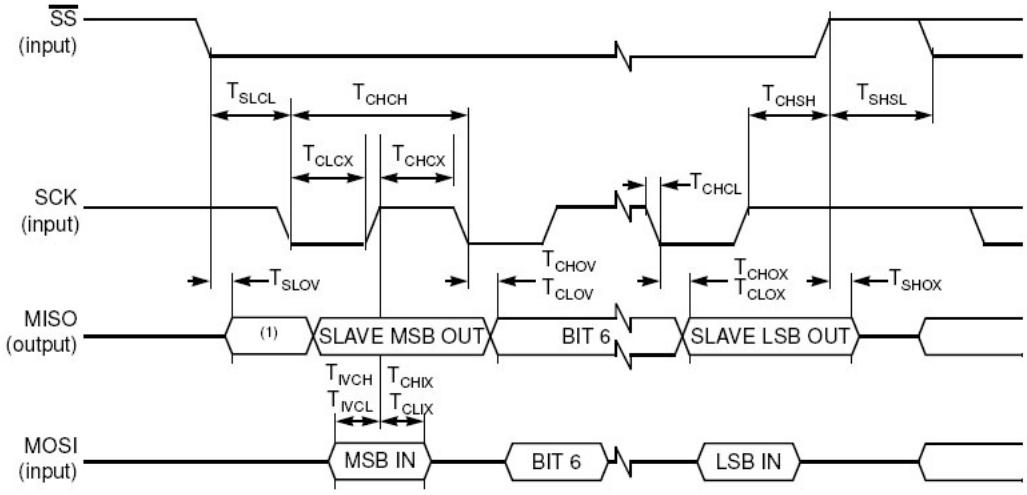

Drawing: Multiple byte transmission over SPI bus in high-speed mode




Drawing: Multiple byte transmission over SPI bus in command mode

## 8.3. General SPI Timing Specifications

The diagram below indicates the general SPI timing specifications for the TFT128 ScreenKey.




Note: 1. Not Defined but generally the LSB of the character which has just been received.

| Symbol | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| $T_{CHCH}$ | Clock Period | 100 | | ns |
| $T_{CHCX}$ | Clock High Time | 80 | | ns |
| $T_{CLCX}$ | Clock Low Time | 80 | | ns |
| $T_{SLCL}$ | $\overline{SS}$ Low to Clock edge | 100 | | ns |
| $T_{IVCH}$ | Input Data Valid to Clock Edge | 50 | | ns |
| $T_{CHIX}$ | Input Data Hold after Clock Edge | 50 | | ns |
| $T_{CLOV}$ | Output Data Valid after Clock Edge | | 50 | ns |
| $T_{CLOX}$ | Output Data Hold Time after Clock Edge | 0 | | ns |
| $T_{CHSH}$ | $\overline{SS}$ High after Clock Edge | 0 | | ns |
| $T_{SLOV}$ | $\overline{SS}$ Low to Output Data Valid | | 220 | ns |
| $T_{SHOX}$ | Output Data Hold after $\overline{SS}$ High | | 200 | ns |
| $T_{SHSL}$ | $\overline{SS}$ High to $\overline{SS}$ Low | 220 | | ns |
| $T_{ILIH}$ | Input Rise Time | | 2 | µs |
| $T_{IHIL}$ | Input Fall Time | | 2 | µs |
| $T_{OLOH}$ | Output Rise time | | 100 | ns |
| $T_{OHOL}$ | Output Fall Time | | 100 | ns |

# 9. High-Level Command Mode

In *Command Mode* the TFT128D ScreenKey supports an extensive command interface that is designed to minimize the bandwidth requirements necessary to drive multiple ScreenKeys within the same panel.

For example, many systems will use text displays extensively. High-Level Command Mode offers a text display command in which a simple ASCII string is sent to the display and the font manipulation and rendering is handled within the TFT128D itself. Similarly, simple commands are provided to quickly blank the display to a certain color or to display small graphics in sub-windows, etc.

To support this feature, the TFT128D implements a 'soft' handshaking protocol over SPI. This is necessary as different commands can take longer to execute, other commands operate on the data as it is received while yet other commands buffer the incoming data before executing the instruction.

## *9.1. Command Structure*

The master controls the TFT128D ScreenKey Display by issuing a command packet. The ScreenKey reports its status for each command byte transmitted. Where required, the ScreenKey can return data information if requested by the master.

Command packets sent by the master conform to the following packet structure:

| *Header* | | | | *Data* | *Trailer* | | |
|---|---|---|---|---|---|---|---|
| CMD ID | XOR | LEN (MSB) | LEN (LSB) | … *data* … | 0x55 | 0xAA | 0x00 |
| byte 1 | byte 2 | byte 3 | byte 4 | byte 5 … byte n | byte n+1 | byte n+2 | byte n+3 |

| | |
|---|---|
| CMD ID | unique command identifier (00 – FF) |
| XOR | exclusive-or value of the CMD ID (00-FF) |
| LEN | length of data following (0000 - FFFF) |
| data | 8-bit data bytes as necessary |
| trailer | 3 bytes must always be sent after each *data* section (0x55 0xAA 0x00) |

## 9.2. ScreenKey Status Byte

While data bytes are clocked into the ScreenKey Display on the MOSI line, the return information clocked out of the ScreenKey on the MISO is the ***ScreenKey Status Byte***.

The ScreenKey Status Byte (SSB) is an 8-bit byte transmitted MSB first:

| | |
|---|---|
| bit 7 | SW (not used by TFT128D)<br>*0 = switch not pressed*<br>*1 = switch pressed* |
| bit 6 | Reserved – *returns 0* |
| bit 5 | Reserved – *returns 0* |
| bit 4 | Reserved – *returns 0* |
| bit 3 | ONLINE<br>*1 = TFT128 present and online*<br>*0 = invalid reply from ScreenKey* |
| bit 2 | CMDOK<br>*0 = normal state during data transfer*<br>*1 = Command accepted and processed by ScreenKey* |
| bit 1 | NACK<br>*0 = Data accepted*<br>*1 = Command/data rejected by ScreenKey* |
| Bit 0 | BUSY<br>*0 = Byte accepted OK*<br>*1 = ScreenKey busy, resend current byte* |

*Note:*

*The ScreenKey Status Byte is* ***not*** *returned when operating in* high-speed mode.

**SW (SWITCH DETECT) – not used by TFT128D**

The SW bit is used by the TFT128 ScreenKey **switch product**. It is not relevant to the TFT128D. The TFT128 ScreenKey switch product continually monitors its internal switch mechanism and reports a switch depression by setting the SW bit. This bit remains set until the master issues a *Switch Acknowledge* command. The SW bit is then cleared until the next key press is detected.

**BUSY (SCREENKEY BUSY)**

During receipt of a command packet, the ScreenKey may require the master to delay the next byte transmission. The ScreenKey sets the BUSY bit to indicate this to the master. The master should continue to resend its current byte until the WAIT bit is cleared by the ScreenKey. The inter-byte delay period must be adhered to between byte resends.

Commands with longer execution times may set the BUSY bit during receipt of the *trailer* bytes. The master may proceed to service another key and return later to resend the byte to the key that requested a WAIT.

**CMDOK (COMMAND OK)**

After receiving a full command packet including the three trailing bytes, the ScreenKey will set the CMDOK bit to indicate that the command has been fully received and executed. If CMDOK bit is **not** set in the SBB returned on the third trailing byte then the command has not been received correctly and the master should resend.

If CMDOK is not set when expected, the master should resync with the ScreenKey using the resync sequence described below.

**NACK (COMMAND NOT ACKNOWLEDGED)**

The NACK bit may be set at any time during command reception. This indicates that the ScreenKey found an error or unacceptable value during command reception and has rejected the command. The ScreenKey automatically returns to waiting for the start of the next command packet. The master must decide to resend the command if required.

If NACK is set at any time, the master should resync with the ScreenKey using the resync sequence described below.

**ONLINE (SCREENKEY PRESENT AND ONLINE)**

The ONLINE bit should always be set. It may be used to detect the presence of a TFT ScreenKey on the SPI bus and to confirm that the ScreenKey is operating normally and is in high-level command mode. If this bit position returns 0 (when not in high-speed mode) then the device is not a ScreenKey or it is not operating correctly.

*Refer to communications flowchart below that describes how these bits are used to control command transmission.*

*Note that during font, graphic and color downloads the SPI interface will automatically issue 0xFF relies to the master while it stores the downloaded information into its flash memory (approx every 128 bytes). The SPI master software should recognize this as a BUSY reply from the ScreenKey and continue to reissue the last byte as per the normal busy response.*

## *9.3. Error Recovery & Resync*

In the event of a command transmission failure, the master should resync with the ScreenKey. The master can send zero-value bytes (0x00) before starting a new command to resync with the ScreenKey. The ScreenKey returns it's SSB and the master can confirm that the BUSY, CMDOK and NACK bits are all returned to 'normal', i.e. all off (0).

While the ScreenKey is not in a command reception sequence, it ignores preamble zeroes and returns its SSB to each. Receipt of a non-zero value is taken by the ScreenKey as the CMD ID of the next command.

It is important that the inter-byte delay is observed between zero-value bytes as otherwise the returned value from the key is undefined.

## 9.4. Data Information *from* ScreenKey

Where commands request the ScreenKey to return information, the master must complete the full command packet including the three trailer bytes.

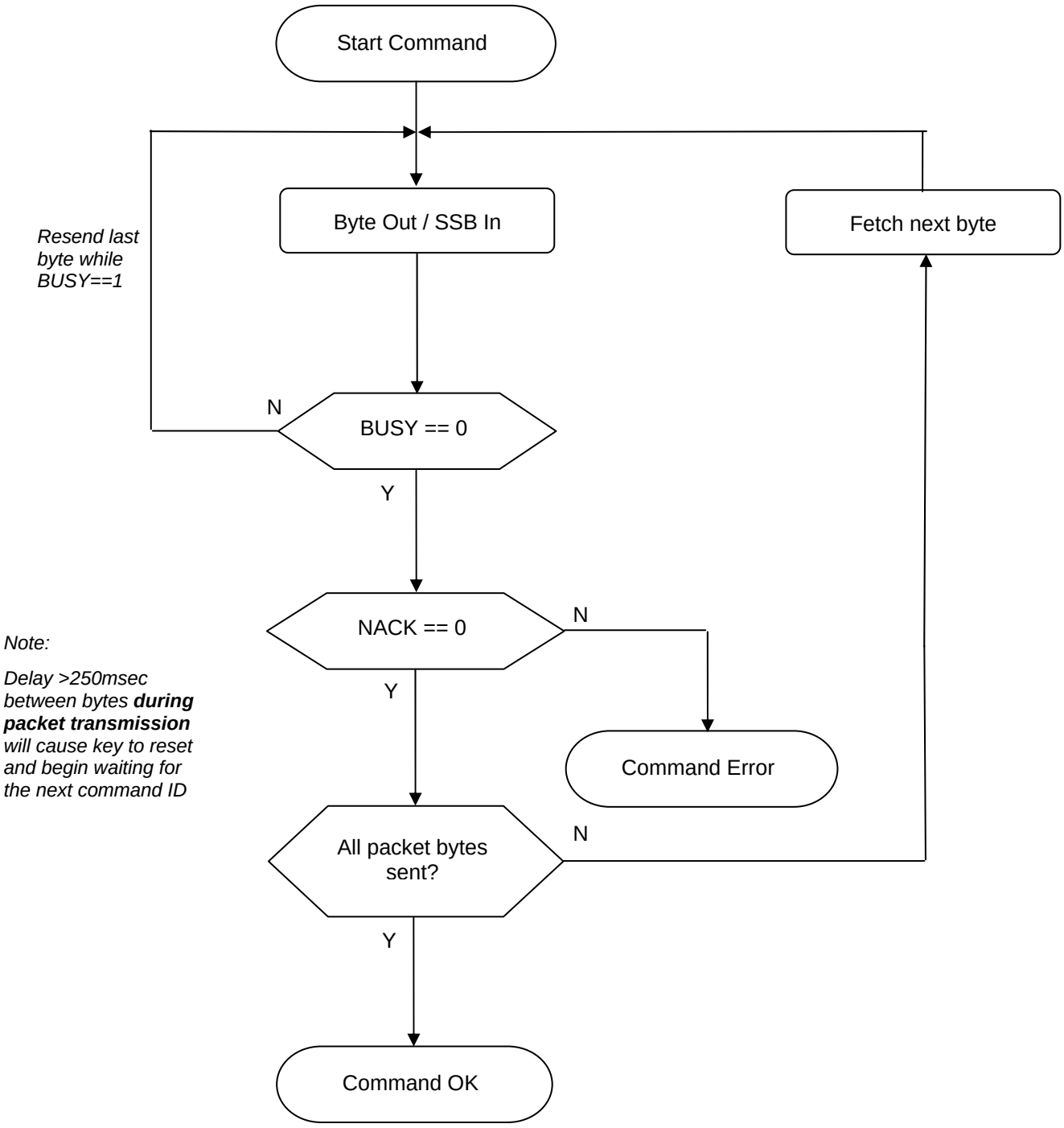
Before returning to normal processing, the master must then issue zero bytes (00hex) to the ScreenKey to trigger the return of the requested information from the ScreenKey. The master must issue the correct number of zero bytes to match the requested number of return information for that command.

## 9.5. Inactivity Timeout

During command packet transmission it is possible for the master to “disappear” and fail to send the full command packet or for the slave ScreenKey to believe in error that the master has not sent the full command.

This situation is detected by the inactivity timeout. Once a new command packet is initiated, the ScreenKey begins a 250 millisecond timer after each received byte. If the next byte is not received in this period, then the key discards any information received in this partial packet and it returns to waiting for a new CMD byte.

## *9.6. Communications Flowchart*



## 9.7. Example Communication Scenarios

**COMMAND SENT AND EXECUTED SUCCESSFULLY**

| ***Timeslot*** | | ***1*** | ***2*** | ***…*** | ***n-2*** | ***n-1*** | ***n*** |
|---|---|---|---|---|---|---|---|
| | | CMD | XOR | *data* | 0x55 | 0xAA | 0x00 |
| ***BUSY*** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ***NACK*** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ***CMDOK*** | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

The master sends the CMD byte and checks BUSY in SSB return. The ScreenKey keeps the NACK bit clear while it continues to accept data. The master sends the data bytes while always checking the BUSY bit (BUSY is set by the key to indicate that the master should wait before sending a new byte). All commands must send the three trailer bytes (0x55, 0xaa, 0x00) after the command data bytes. If all is OK with the ScreenKey, then it will set the CMDOK bit as the final 0x00 is sent from the master.

**COMMAND SENT AND EXECUTED SUCCESSFULLY BUT WAIT REQUESTED DURING TRANSMISSION**

| ***Timeslot*** | | ***1*** | ***2*** | ***3*** | ***4*** | ***5*** | ***n-1*** | ***n*** | ***n+1*** |
|---|---|---|---|---|---|---|---|---|---|
| | | CMD | XOR | data | *DataX* | DataX resent | 0x55 | 0xAA | 0x00 |
| ***BUSY*** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| ***NACK*** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ***CMDOK*** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

The transaction begins as normal but during data transmission the key sets BUSY bit. The master continues resending the current byte (DataX) until BUSY is cleared by the key. The application programmer should decide how many time loops to wait before deciding that the transmission has been compromised. After BUSY is cleared the transmission then continues as normal.

**BAD DATA RECEIVED BY KEY**

| ***Timeslot*** | | ***1*** | ***2*** | ***…*** | ***n-2*** | ***n-1*** | ***n*** |
|---|---|---|---|---|---|---|---|
| | | CMD | XOR | *Data* | 0x55 | 0xAA | 0x00 |
| ***BUSY*** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ***NACK*** | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| ***CMDOK*** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

The master sends a command packet as in the first example. At the third trailer byte, the ScreenKey reports NACK to the master meaning that the command was unsuccessful. The master must decide to resend the packet or service another key.

#### COMMAND NOT FULLY RECEIVED BY KEY

| ***Timeslot*** | | ***1*** | ***2*** | ***…*** | ***n-2*** | ***n-1*** | ***n*** |
|---|---|---|---|---|---|---|---|
| | | CMD | XOR | *Data* | 0x55 | 0xAA | 0x00 |
| ***BUSY*** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ***NACK*** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ***CMDOK*** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

If the master sends the full packet including traler bytes but the ScreenKey responds without either a NACK or CMDOK, then this indicates that the ScreenKey has missed bytes during the command transmission and it still believes it has more bytes to receive.

The master can handle this scenario by allowing the inter-byte timeout (50msec) to activate and so return the ScreenKey to waiting for the next command. The master should send the zero-preamble before starting the next command to confirm that all is OK.

#### MASTER FAILS TO SEND FULL COMMAND PACKET

If the master fails to complete the sending of a full command packet then the inter-byte timeout will trigger on the ScreenKey. This will discard any partial data received and set the key into waiting for a new CMD byte.

## *9.8. Screen Refresh Rate*

Using the 256-color command (see section below), each pixel is defined by a single byte value. A full screen refresh requires 128*128 bytes, i.e. 16,384 bytes.

Using 15.5 microsecond inter-byte delay, the minimum transmission period for 16,384 bytes is 0.254 seconds.

The screen refresh rate is therefore ~4 frames per second in this mode when sending non-compressed 256-color image data.

Of course, image data does not have to fit onto the full screen area and sub-windows may be defined and populated, thus reducing the amount of data to be transmitted and the resulting screen refresh rate.

In high-speed mode (see below), it is possible to achieve much faster screen refresh rates as the inter-byte delay is greatly reduced. Using a 96*54 pixel video sub-window, over 29 frames per second can be achieved while full 128*128 screens accomplish just over 9 frames per second (sending 32,768 bytes 16-bit color but with 2.25/2.75 microsecond inter-byte delays).

It should be noted that the internal refresh rate is approx 69ms, i.e. when a blank screen command is received, it takes 69ms to repaint the screen to the requested color.

The dominant factor in screen refresh is the transmission of data. In command mode, data transmission may be reduced by using RLE compressed image data (see command 27 below).

## 9.9. High-Level Command Set

## Command 01h – Key Mode Reset

**Format:**

| | | |
|---|---|---|
| CMD ID | 01h | |
| XOR | FEh | |
| LEN | 00h/02h | |
| Data | MM | *Mode* |
| | RR | *Orientation* |

**Command Data:**

| | | |
|---|---|---|
| MM = | 00 | Set operating mode to *command mode* (default) |
| | 01 | Set operating mode to *high-speed* |
| RR = | x0 | Reset to *Landscape* |
| | x1 | Reset to *Portrait* (default) |
| | 0x | Set graphic wipe direction from top-left downwards (default) |
| | 1x | Set graphic wipe direction from bottom-left upwards |

**Return Data:**

None.

**Description:**

This command sets the ScreenKey mode:

For *command* mode it restores the display to its power on state, resets all communication states and settings to their startup values and clears the display to its default background color.

For *high-speed* mode, it puts the ScreenKey into high-speed mode but does not reset any variables or other settings. When put into high-speed mode, the active window area is defined according to the settings passed in Cmd 02 (Video Sub-window). If this has not been set then the active window area is set to full screen (128 * 128 pixels).

If high-speed mode is activated with this command then the display treats all further master transmissions as 16-bit color instructions (see section 10 below).

Once high-speed mode is activated, the display remains in this mode until power-off or until the *Exit High-Speed* signal is detected (see section 10).

It is also necessary to define the orientation with this command (see command 10h below).

Two graphic wipe directions are supported. The default is *"top-left downwards"* which means that graphic image bytes populate the display screen from the top left pixel first, filling left across the pixel row and then continuing to populate the next row down, etc.

The *"bottom-left upwards"* wipe direction is provided for Windows BMP files in which image data is stored from the bottom left pixel position, across the row and then continuing with the next row above.

The graphic wipe direction only applies to the display graphic commands (Cmd 21/22/23/27) and to *High-Speed mode*.

If this command is sent to put the display into *command mode*, then the display is blanked to white and any downloaded images are erased. If setting operation to *high-speed mode*, this does not change the screen display nor affect downloaded images.

***Default: The default mode is COMMAND MODE and PORTRAIT (top-left downwards)***

## *Command 02h – Video Sub-window*

**Format:**

| | | |
|---|---|---|
| CMD ID | 02h | |
| XOR | FDh | |
| LEN | 00h/04h | |
| Data | X1 | *Top left 'x' co-ord* |
| | Y1 | *Top left 'y' co-ord* |
| | X2 | *Bottom right 'x' co-ord* |
| | Y2 | *Bottom right 'y' co-ord* |

**Command Data:**

| | |
|---|---|
| X1,Y1 | 'x,y' co-ordinates of top left position (0-127,0-127) |
| X2,Y2 | 'x,y' co-ordinates of bottom right position (0-127,0-127) |

**Return Data:**

None.

**Description:**

This command defines a video sub-window that will be populated when in *high-speed* mode.

If this command is not issued, then data issued in high-speed mode populates the full 128*128 screen. With this command, it is possible to define a smaller sub-window which can be placed anywhere within the full screen area and which can have any ratio required.

This allows the user to define an active video sub-window which has the same ratio as a video stream. For example, 16:9 video could be displayed in a sub-window using 96 pixels wide by 54 pixels high. This has the added benefit of reducing the amount of data to be sent to the TFT128 per frame and so enables the frames per second rate to be increased.

Using a video sub-window of 96*54, almost 29 frames per second could be achieved.

As only the active video sub-window is populated in high-speed mode, the area outside of this window could be used to display a static frame or a video source identifier. This area should be painted prior to entering high-speed mode.

***Default: Video sub-window defaults to the full screen area (Horizontal 0 – 127, Vertical 0 – 127)***

## *Command 09h – Switch Acknowledge – not used by TFT128D Display*

**Format:**

| | |
|---|---|
| CMD ID | 09h |
| XOR | F6h |
| LEN | 00h/00h |

**Command Data:**

None.

**Return Data:**

None.

**Description:**

This command instructs the ScreenKey that the master has acknowledged the reported switch press. The ScreenKey resets its SW bit in the SSB until a new switch press is detected, where the ScreenKey then sets the SW bit again.

## *Command 10h – Set Orientation*

**Format:**

| | | |
|---|---|---|
| CMD ID | 10h | |
| XOR | EFh | |
| LEN | 00h/01h | |
| Data | MM | *Orientation* |

**Command Data:**

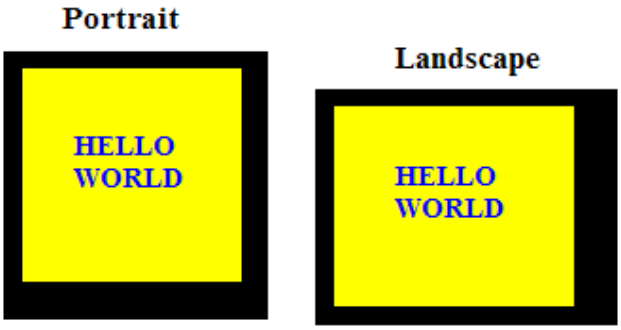
| | | |
|---|---|---|
| MM = | x0 | Reset to *Landscape* |
| | x1 | Reset to *Portrait* (default) |
| | 0x | Set graphic wipe direction from top-left downwards (default) |
| | 1x | Set graphic wipe direction from bottom-left upwards |

**Return Data:**

None.

**Description:**

The TFT can be physically mounted in a portrait or landscape orientation.




Drawing: Portrait and landscape physical orientations

Normal text and drawing commands are referenced with respect to the top left-hand corner of the key display. Physically, this differs depending on how the key is physically mounted. Once this command is issued, all future text or display commands reference themselves to the relevant top left corner.

Any currently display images or text will continue to be displayed in the same positions after this command has been issued. If used dynamically, this command would usually be followed by the *Clear Display* command.

Two graphic wipe directions are supported. The default is *"top-left downwards"* which means that graphic image bytes populate the display screen from the top left pixel first, filling left across the pixel row and then continuing to populate the next row down, etc.

The *"bottom-left upwards"* wipe direction is provided for Windows BMP files in which image data is stored from the bottom left pixel position, across the row and then continuing with the next row above.

The graphic wipe direction only applies to the display graphic commands (Cmd 21/22/23/27).

Issuing this command will reset any flash settings (see Cmd 26h).

***Default: The default mode is PORTRAIT (top-left downwards)***

## *Command 11h – Set Color*

**Format:**

| | | |
|---|---|---|
| CMD ID | 11h | |
| XOR | EEh | |
| LEN | 00h/03h | |
| Data | RR | *Color reference* |
| | CH | *Color value (MSB)* |
| | CL | *Color value (LSB)* |

**Command Data:**

| | |
|---|---|
| RR | Color reference (position in color table 0-15) |
| CH/CL | 16 bit color value |

**Return Data:**

None.

**Description:**

Every pixel can be set to one of 65,536 colors, with each color definable as a 16-bit value:

| R | R | R | R | R | G | G | G | G | G | G | B | B | B | B | B |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Red | | | | | Green | | | | | | Blue | | | | |
| CH (MSB byte) | | | | | | | | CL (LSB byte) | | | | | | | |

The TFT128D maintains a Color Reference Table of 16 different colors:

| *Color Ref* | *Default Color* | *RGB Value* |
|---|---|---|
| 0 | Black | 00 00 |
| 1 | Dark Red | D8 00 |
| 2 | Dark Green | 07 60 |
| 3 | Dark Yellow | FF 80 |
| 4 | Dark Blue | 00 1B |
| 5 | Dark Magenta | D8 1B |
| 6 | Dark Cyan | 0D FB |
| 7 | Light Grey | F7 9E |

| *Color Ref* | *Default Color* | *RGB Value* |
|---|---|---|
| 8 | Dark Grey | D6 9A |
| 9 | Red | F8 00 |
| 10 | Green | 07 E0 |
| 11 | Yellow | FF E0 |
| 12 | Blue | 00 1F |
| 13 | Magenta | F8 1F |
| 14 | Cyan | 07 FF |
| 15 | White | FF FF |

This command is used to redefine the colors in the above table. Text commands reference the colors in this table for their foreground and background colors.

Simple 16-color graphics can also be drawn with reference to this color palette.

## *Command 12h – Set Cursor Position*

**Format:**

| | | |
|---|---|---|
| CMD ID | 12h | |
| XOR | EDh | |
| LEN | 00h/02h | |
| Data | XX | *'x' co-ordinate* |
| | YY | *'y' co-ordinate* |
| CKSUM | CC | |

**Command Data:**

| | |
|---|---|
| XX | 'x' co-ordinate for new cursor position (0-127) |
| YY | 'y' co-ordinate for new cursor position (0-127) |

**Return Data:**

None.

**Description:**

Text commands display text on screen beginning at the current cursor position. The cursor position is updated after each character has been displayed.

This command allows the user to set the cursor position to a new co-ordinate so that finite text positioning is possible.

***Default: The default cursor position is 0,0***

## *Command 13h – Clear Display*

**Format:**

| | | |
|---|---|---|
| CMD ID | 13h | |
| XOR | ECh | |
| LEN | 00h/01h | |
| Data | RR | *Color reference* |

**Command Data:**

| | |
|---|---|
| RR | Index into Color Reference Table |

**Return Data:**

None.

**Description:**

This command blanks the full display to the specified color from the Color Reference Table. Issuing this command will reset any flash settings (see Cmd 26h).

***Default: The default power-on screen color is WHITE***

## *Command 14h – Replace Color*

**Format:**

| | | |
|---|---|---|
| CMD ID | 14h | |
| XOR | EBh | |
| LEN | 00h/09h | |
| Data | TT | *Type of color reference* |
| | RH/RL | *Color to replace* |
| | NH/NL | *New color (16-bit value)* |
| | X1 | *Top left 'x' co-ord* |
| | Y1 | *Top left 'y' co-ord* |
| | X2 | *Bottom right 'x' co-ord* |
| | Y2 | *Bottom right 'y' co-ord* |

**Command Data:**

| | |
|---|---|
| TT | Type of color reference:<br>0 = index into Color Palette Table (256 colors)<br>1 = actual 16-bit color RGB value<br>2 = index into Color Reference Table (16 colors) |
| RH/RL | Color to be replaced (RH = high byte always 00h if type = 0 or 2) |
| NH/NL | New color to use (16-bit color value) |
| X1,Y1 | 'x,y' co-ordinates of top left position (0-127,0-127) |
| X2,Y2 | 'x,y' co-ordinates of bottom right position (0-127,0-127) |

**Return Data:**

None.

**Description:**

This command can be used to replace a specified color with a new color within a specified rectangular area of the display.

The rectangular area is defined by the x1,y1 and x2,y2 co-ordinates (top left and bottom right respectively).

Two bytes are used to define a color and the value in these bytes can be interpreted differently, depending on the value of TT:

**TT = 0 => Color values are indexes into the 256-color Color Palette Table**

*The color high byte (RH) will always be 00h*

*The color low byte (RL) will contain a value between 00h and FFh which is an index into the Color Palette Table*

**TT = 1 => Color values are actual 16-bit RGB color**

*The high/low byte combination define the full RGB value*

**TT = 2 => Color values are indexes into the 16-color Color Reference Table**

*The color high byte (RH) will always be 00h*

*The color low byte (RL) will contain a value between 00h and 0Fh which is an index into the Color Reference Table*

The color specified by Nh/NL is **always** the actual 16-bit RGB color value.

*Note: The active window area is automatically returned to 128*128 after completion of this command.*

## *Command 15h – Set Backlight*

**Format:**

| | | |
|---|---|---|
| CMD ID | 15h | |
| XOR | EAh | |
| LEN | 00h/02h | |
| Data | BB | *Backlight brightness setting* |
| | TT | *Backlight timeout* |

**Command Data:**

| | |
|---|---|
| BB | Backlight brightness setting, 00h = off, FFh = full |
| TT | Backlight timeout setting in minutes (0 = always on) |

**Return Data:**

None.

**Description:**

The TFT128D includes an integrated white LED backlight. This can be turned on or off and the brightness adjusted with this command. This is useful to put a panel into “sleep” mode, e.g. for energy conservation.

The ScreenKey automatically enters sleep mode after a preset timeout period and the backlight is turned off. The default timeout period is 20 minutes but this can be changed from always on (value = 0) to any period up to 255 minutes. Sleep mode is turned off whenever the ScreenKey is pressed or if a valid SPI command is received.

The brightness can be adjusted between off (00h) and full brightness (FFh).

Brightness settings below approx 40 effectively turn the backlight off. The actual useful range for the backlight setting is 40 – 255 and use 0 for off.

The ScreenKey starts up with the backlight OFF but with the ‘on’ setting at maximum (255). If the key is pressed or a valid SPI command is received, then the backlight is automatically turned on at this setting.

***Default Values: Backlight OFF; set to full brightness (255); timeout = 20 minutes***

## *Command 20h – Display Text*

**Format:**

| | | |
|---|---|---|
| CMD ID | 20h | |
| XOR | DFh | |
| LEN | 00h/xxh | |
| Data | RR | *Fore and back colors* |
| | BB | *Bit operation* |
| | …*text*… | *ASCII text* |

**Command Data:**

| | |
|---|---|
| RR | High nibble is Color Reference Table of foreground color for text<br>Low nibble is Color Reference Table of background color for text |
| BB | Bit operation to perform |
| *text* | ASCII text to display |

**Return Data:**

None.

**Description:**

The TFT128D has an integrated fixed size font that allows ASCII text to be displayed quickly and easily. The ASCII string passed in this command is painted from left to right, starting at the current cursor position. Characters are drawn to the end of the line and continued from the left most position on the next line. This is continued until the last character position on the display is reached. All further text is discarded.

The starting cursor position is defined as 0,0 or the top left pixel position.

The foreground and background color of the text is defined in the first passed data byte (RR), where the high nibble refers to the text background color and the low nibble refers to the text foreground color. Each nibble is an index into the Color Reference Table.

| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
|---|---|---|---|---|---|---|---|
| Background color index | | | | Foreground color index | | | |

The bit operation (BB) defines how to paint the font character onto the display:

| *Value* | *Description* |
|---|---|
| 00h | Paint text with referenced colors from RR byte, including painting “off” pixels with specified background color |
| 01h | Paint font “on” pixels with specified foreground color but do not perform any action for font “off” pixels |
| 02h | Paint “on” pixels from font as 1’s complement of current pixel display color. Do not perform any action for font “off” pixels. |

Note that this command automatically stops any flash settings setup with Cmd 26h.

## *Command 21h – Display 256-Color Graphic*

**Format:**

| | | |
|---|---|---|
| CMD ID | 21h | |
| XOR | DEh | |
| LEN | xxh/xxh | |
| Data | XX | *'x' co-ord for graphic display* |
| | YY | *'y' co-ord for graphic display* |
| | WW | *Width of graphic* |
| | HH | *Height of graphic* |
| | *…graphic…* | *Bitmap data* |

**Command Data:**

| | |
|---|---|
| XX,YY | 'x,y' co-ordinates of top left position for graphic placement (0-127) |
| WW | Width of graphic in pixels (1-128) |
| HH | Height of graphic in pixels (1-128) |
| *graphic* | Graphic bitmap data (WW*HH bytes) |

**Return Data:**

None.

**Description:**

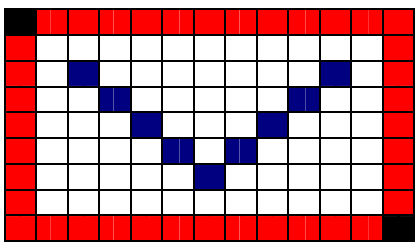
The TFT128D can light a pixel to any of 65,536 colors, as explained in section 9, where each pixel color is defined by a two-byte value. However, to display a full screen image using 2 bytes per pixel would require 32Kbytes of information. To reduce the amount of data required to define graphics, the TFT128D can work with 256 color graphics. In 256-color mode, each pixel is defined by a single byte which is a reference into a *Color Palette Table (CPT)*. The CPT holds the relevant two-byte full color codes for each unique color in the 256-color graphic. This is a similar structure as used in 256-color Windows BMP files.

The Color Palette Table is downloaded into the TFT128D using a separate command. Multiple graphics may then be designed and manipulated using the same palette.

Graphic data is transmitted row by row starting with the top left pixel.

The following abstract graphic (13 pixels wide by 9 pixels high) is defined by the data bytes to the right. These are shown in the order in which they are transmitted, i.e. left to right and top to bottom.

The data bytes shown assume a Color Palette Table based on the default Color Reference Table (see command 11), i.e. black is in position 0 in the CPT, red is in position 9, white in position 15 (0F hex) and blue in position 4.



```
00 09 09 09 09 09 09 09 09 09 09 09 09
09 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 09
09 0F 04 0F 0F 0F 0F 0F 0F 0F 04 0F 09
09 0F 0F 04 0F 0F 0F 0F 0F 04 0F 0F 09
09 0F 0F 0F 04 0F 0F 0F 04 0F 0F 0F 09
09 0F 0F 0F 0F 04 0F 04 0F 0F 0F 0F 09
09 0F 0F 0F 0F 0F 04 0F 0F 0F 0F 0F 09
09 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 09
09 09 09 09 09 09 09 09 09 09 09 09 00
```

Total no of data bytes = 117

Note that this command automatically stops any flash settings setup with Cmd 26h.

## *Command 22h – Display Full-Color Graphic*

**Format:**

| | | |
|---|---|---|
| CMD ID | 22h | |
| XOR | DDh | |
| LEN | xxh/xxh | |
| Data | XX | *'x' co-ord for graphic display* |
| | YY | *'y' co-ord for graphic display* |
| | WW | *Width of graphic* |
| | HH | *Height of graphic* |
| | *…graphic…* | *Bitmap data* |

**Command Data:**

| | |
|---|---|
| XX,YY | 'x,y' co-ordinates of top left position for graphic placement (0-127) |
| WW | Width of graphic in pixels (1-128) |
| HH | Height of graphic in pixels (1-128) |
| *graphic* | Graphic bitmap data |

**Return Data:**

None.

**Description:**

The command allows graphics to be drawn on the TFT128D in full-color, i.e. all 65k colors can be used for each pixel.

Each pixel color is defined by two bytes:

| R | R | R | R | R | G | G | G | G | G | G | B | B | B | B | B |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Red | | | | | Green | | | | | | Blue | | | | |
| CH (MSB byte) | | | | | | | | CL (LSB byte) | | | | | | | |

The graphic bitmap data is transmitted in the same order as command 21, i.e. left to right and top to bottom, starting with the top left pixel.

Note that this command automatically stops any flash settings setup with Cmd 26h.

## *Command 23h – Display 16-Color Graphic*

**Format:**

| | | |
|---|---|---|
| CMD ID | 23h | |
| XOR | DCh | |
| LEN | xxh/xxh | |
| Data | XX | *'x' co-ord for graphic display* |
| | YY | *'y' co-ord for graphic display* |
| | WW | *Width of graphic* |
| | HH | *Height of graphic* |
| | *…graphic…* | *Bitmap data* |

**Command Data:**

| | |
|---|---|
| XX,YY | 'x,y' co-ordinates of top left position for graphic placement (0-127) |
| WW | Width of graphic in pixels (1-128) |
| HH | Height of graphic in pixels (1-128) |
| *graphic* | Graphic bitmap data |

**Return Data:**

None.

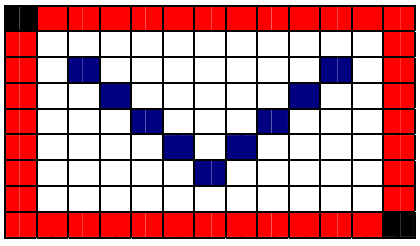
**Description:**

Graphical data may be reduced significantly by reducing the number of colors in an image. This is useful to reduce the bandwidth requirements of managing multiple ScreenKeys from the same controller.

The TFT128D includes a 16 color *Color Reference Table* (CRT - see command 11) which can be used as a 16-color palette. 16-color graphics can use this palette where each pixel is defined as a reference into the CRT.

Graphic data is transmitted row by row starting with the top left or bottom left pixel (depending on specified wipe direction – see Cmd 01 and Cmd 10).

As each pixel reference only requires 4 bits, one byte in the graphic data may be used to define two pixels, e.g.



```
09 99 99 99 99 99 9_
_9 FF FF FF FF FF F9
9F 4F FF FF FF 4F 9_
_9 FF 4F FF FF 4F F9
9F FF 4F FF 4F FF 9_
_9 FF FF 4F 4F FF F9
9F FF FF 4F FF FF 9_
_9 FF FF FF FF FF F9
99 99 99 99 99 99 0x
```

```
Total no of
data bytes = 59

Last byte
includes unused
lower nibble
```

Note that this command automatically stops any flash settings setup with Cmd 26h.

## *Command 24h – Draw Rectangle*

**Format:**

| | | |
|---|---|---|
| CMD ID | 24h | |
| XOR | DBh | |
| LEN | 00h/06h | |
| Data | X1 | *Top left 'x' co-ord* |
| | Y1 | *Top left 'y' co-ord* |
| | X2 | *Bottom right 'x' co-ord* |
| | Y2 | *Bottom right 'y' co-ord* |
| | LL | *Line color* |
| | WW | *Line width* |

**Command Data:**

| | |
|---|---|
| X1,Y1 | 'x,y' co-ordinates of top left position (0-127,0-127) |
| X2,Y2 | 'x,y' co-ordinates of bottom right position (0-127,0-127) |
| LL | Index into Color Reference Table (0-15) |
| WW | Line width (1-127) |

**Return Data:**

None.

**Description:**

This command will draw a rectangle using the specified co-ordinates. The line width can be specified from 1 pixel upwards. The specified co-ordinates are always the outer limits of the rectangle, i.e. line widths greater than 1 pixel extend towards the interior of the rectangle.

The line color (LL) passed with this command is an index into the 16-color Color Reference Table (see command 20).

This command may also be used to draw lines.

Note that this command automatically stops any flash settings setup with Cmd 26h.

## *Command 25h – Draw Circle*

**Format:**

| | | |
|---|---|---|
| CMD ID | 25h | |
| XOR | DAh | |
| LEN | 00h/05h | |
| Data | X1 | *Centre 'x' co-ord* |
| | Y1 | *Centre 'y' co-ord* |
| | RR | *Radius of circle in pixels* |
| | LL | *Line color* |
| | WW | *Line width* |

**Command Data:**

| | |
|---|---|
| X1,Y1 | 'x,y' co-ordinates of centre of circle (0-127,0-127) |
| RR | Radius of circle in pixels (1-127) |
| LL | Index into Color Reference Table (0-15) |
| WW | Line width (1-127) |

**Return Data:**

None.

**Description:**

This command will draw a circle with a radius of RR pixels from the centre point defined by the co-ordinates x1,y1. The line width can be specified from 1 pixel upwards. Line widths greater than 1 pixel extend towards the interior of the circle.

The line color (LL) passed with this command is an index into the 16-color Color Reference Table (see command 20).

Note that this command automatically stops any flash settings setup with Cmd 26h.

## *Command 26h – Set Flash*

**Format:**

| | | |
|---|---|---|
| CMD ID | 26h | |
| XOR | D9h | |
| LEN | 00h/xxh | |
| Data | X1 | *Top left 'x' co-ord* |
| | Y1 | *Top left 'y' co-ord* |
| | X2 | *Bottom right 'x' co-ord* |
| | Y2 | *Bottom right 'y' co-ord* |
| | RR | *Flash colors* |
| | X3 | *Top left 'x' co-ord for text* |
| | Y3 | *Top left 'y' co-ord for text* |
| | TT | *Text color* |
| | *…text…* | *Text string* |

**Command Data:**

| | |
|---|---|
| X1,Y1 | 'x,y' co-ordinates of top left rectangle (0-127,0-127) |
| X2,Y2 | 'x,y' co-ordinates of bottom right rectangle (0-127,0-127) |
| RR | High nibble is Color Reference Table of first flash color (0-15)<br>Low nibble is Color Reference Table of alternate flash color (0-15) |
| X3,Y3 | 'x,y' co-ordinates for text placement (0-127,0-127) |
| TT | High nibble is Color Reference Table of first text color (0-15)<br>Low nibble is Color Reference Table of alternate text color (0-15) |
| *text* | Text to display in rectangle |

**Return Data:**

None.

**Description:**

Flashing between two colors can be a useful effect for attracting a users attention, e.g. if an alarm condition occurs, or if a default option should be specifically highlighted, etc.

This command encapsulates the parameters required to flash a sub-section of the display between two specified colors and it can specify a text string to display within this region.

The x/y co-ordinates are provided that define the rectangular area to flash. The flash colors are set in RR, where the high nibble and low nibble are color indexes into the Color Reference Table. The display color of the defined rectangular area is continually flashed between these two colors every second. For best visual performance, the rectangular area should be kept relatively small (the full area of the display should not be used).

Usually, it will be required to also display text in this flashing area. The text location and fore/flash color is defined.

Flashing will be disabled as soon as another display command is received. The last displayed color and text will remain on screen and will be painted over by the most recent command. It is recommended to use the Clear Display command when flashing is no longer required.

Note that the flash period is automatically fixed to display each alternate image for 1 second. This period cannot be modified.

## *Command 27h – Display 256-Color RLE8 Graphic*

**Format:**

| | | |
|---|---|---|
| CMD ID | 27h | |
| XOR | D8h | |
| LEN | xxh/xxh | |
| Data | XX | *'x' co-ord for graphic display* |
| | YY | *'y' co-ord for graphic display* |
| | WW | *Width of graphic* |
| | HH | *Height of graphic* |
| | *…graphic…* | *Bitmap data* |

**Command Data:**

| | |
|---|---|
| XX,YY | 'x,y' co-ordinates of top left position for graphic placement (0-127) |
| WW | Width of graphic in pixels (1-128) |
| HH | Height of graphic in pixels (1-128) |
| *graphic* | Graphic bitmap data (LEN – 4 bytes) |

**Return Data:**

None.

**Description:**

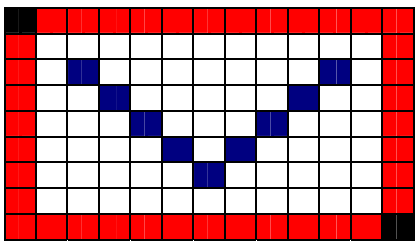
The TFT128D can light a pixel to any of 65,536 colors, as explained in section 8, where each pixel color is defined by a two-byte value. However, to display a full screen image using 2 bytes per pixel would require 32Kbytes of information. To reduce the amount of data required to define graphics, the TFT128D can work with 256 color graphics. In 256-color mode, each pixel is defined by a single byte which is a reference into a *Color Palette Table (CPT)*. The CPT holds the relevant two-byte full color codes for each unique color in the 256-color graphic. The number of bytes used to define a 256-color image is further reduced using RLE8 image data compression which is often used with Windows BMP files. Section 7.3 describes how RLE8 compression is structured.

The Color Palette Table is downloaded into the TFT128D using a separate command. Multiple graphics may then be designed and manipulated using the same palette.

Graphic data is transmitted row by row starting with the top left or bottom left pixel (depending on specified wipe direction – see Cmd 01 and Cmd 10).

The following abstract graphic (13 pixels wide by 9 pixels high) is defined by the data bytes to the right. These are shown in the order in which they are transmitted, i.e. left to right and top to bottom.

The data bytes shown assume a Color Palette Table based on the default Color Reference Table (see command 11), i.e. black is in position 0 in the CPT, red is in position 9, white in position 15 (0F hex) and blue in position 4.



```
                  01 00 0B 09 00 00
               01 09 0A 0F 01 09 00 00
01 04 09 0F 04 0F 06 0F 00 03 04 0F 09 00 00 00
00 04 09 0F 0F 04 05 0F 00 04 04 0F 0F 09 00 00
01 09 03 0F 01 04 03 0F 01 04 03 0F 01 09 00 00
01 09 04 0F 00 03 04 0F 04 00 03 0F 01 09 00 00
      01 09 05 0F 01 04 05 0F 01 09 00 00
            01 09 0A 0F 01 09 00 00
               0C 09 01 00 00 01
```

Total no of data bytes = 104

Note that this command automatically stops any flash settings setup with Cmd 26h.

## *Command 30h – Download Font*

**Format:**

| | | |
|---|---|---|
| CMD ID | 30h | |
| XOR | CFh | |
| LEN | xxh/xxh | |
| Data | NN | *Number of characters in font* |
| | OO | *Offset to first character* |
| | LL | *Inter-line spacing* |
| | SS | *Inter-character spacing* |
| | BB | *Bytes per character* |
| | RR | *Bytes per row* |
| | PP | *Pixels per row* |
| | *…chars…* | *Bitmap data* |

**Command Data:**

| | |
|---|---|
| NN | Number of characters defined in font (1-255) |
| OO | Offset to first character in font (0-255) |
| LL | Number of pixels to skip between text lines |
| SS | Number of pixels to skip between adjacent characters |
| BB | Number of bytes used to define a single character bitmap |
| RR | Number of bytes that define a single row of a character bitmap |
| PP | Number of pixels used in a single row of a character bitmap |
| *chars* | Character bitmap data |

**Return Data:**

None.

**Description:**

Please refer to section 8 above which explains how text commands, fonts and character sets are handled in the TFT128D.

The default factory font has 256 characters defined and is based on the “Courier” style layout with the ISO 8859-15 (Latin alphabet no. 9) character set (see section 8.1).

New font styles and character sets may be defined and downloaded into the TFT128D using this command.

The maximum size of a downloadable font is 8,196 bytes, including the font parameters, i.e. number of characters, inter-line spacing, etc.

Only one font can be present at the same time. The factory default font is replaced when a new font is downloaded using this command. Downloaded fonts remain in place after power down. The only way to re-establish the factory default font is to download this font again using this command.

*Note: During font download the TFT128D SPI interface will automatically issue 0xFF relies to the master while it stores the downloaded information into its flash memory. The SPI master software must recognize this as a BUSY reply from the ScreenKey and continue to reissue the last byte as per the SPI interface description in section 10.2.*

*Note: The TFT128D flash memory has a maximum of 100k write cycles.*

## *Command 31h – Download Color Palette Table*

**Format:**

| | | |
|---|---|---|
| CMD ID | 31h | |
| XOR | CEh | |
| LEN | 02h/00h | |
| Data | *…colors…* | *Color Palette Table* |

**Command Data:**

| | |
|---|---|
| *colors* | 256 entry Color Palette Table |

**Return Data:**

None.

**Description:**

To reduce the amount of data required to define a graphic image (and reduce the amount of bandwidth to transmit graphic images), the TFT128D can use a separate pallet of colors where the graphic data are references into this table.

The TFT ScreenKey supports 256 color images using a 256-entry Color Palette Table (CPT). Each entry in the CPT is two bytes wide, defining a unique color from the 65,536 colors supported by the display:

| R | R | R | R | R | G | G | G | G | G | G | B | B | B | B | B |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Red | | | | | Green | | | | | | Blue | | | | |
| CH (MSB byte) | | | | | | | | CL (LSB byte) | | | | | | | |

Only one CPT is in use at any one time. All 256-color graphic images downloaded use the current CPT for displaying. Changing the CPT after an image has been displayed has no effect on the displayed image. Any subsequent image displays will use this new CPT.

*Note: During color palette downloads the TFT128D SPI interface will automatically issue 0xFF relies to the master while it stores the downloaded information into its flash memory. The SPI master software must recognize this as a BUSY reply from the ScreenKey and continue to reissue the last byte as per the SPI interface description in section 10.2.*

*Note: The TFT128D flash memory has a maximum of 100k write cycles.*

## *Command 32h – Download Graphic*

**Format:**

| | | |
|---|---|---|
| CMD ID | 32h | |
| XOR | CDh | |
| LEN | xxh/xxh | |
| Data | WW | *Width of graphic* |
| | HH | *Height of graphic* |
| | NN | *Graphic ID* |
| | TT | *Type of graphic* |
| | *…graphic…* | *Bitmap data* |

**Command Data:**

| | |
|---|---|
| WW | Width of graphic in pixels (1-128) |
| HH | Height of graphic in pixels (1-128) |
| NN | Graphic ID (1-127) |
| TT | Graphic type – 00 = 256-color, 01 = full-color, 02 = 16-color, 03-256-Color RLE |
| *graphic* | Graphic bitmap data |

**Return Data:**

None.

**Description:**

The TFT128D has limited storage available for downloaded images (max 10 images). Using downloaded images can improve response times in multi-key systems by removing the need to constantly download new images to changing circumstances. Downloaded images are easily recalled using a simple command.

When downloading images, the height and width of the image must be provided as well as the type. Different image types have different storage requirements and pixel mappings:

| | | |
|---|---|---|
| 256-color image | One byte per pixel | Each byte is index into Color Palette Table |
| Full-color image | Two bytes per pixel | Two bytes are actual RGB color value |
| 16-color image | One nibble per pixel | Each nibble is index into Color Reference Table |
| 256-Color RLE | One byte per pixel | Each byte isindex into Color Palette Table but uses RLE compression |

Refer to the display Image commands for full descriptions of the actual structure of image data for each type.

Each download must have a unique numeric graphic ID as this number is used in Command 33 to recall the downloaded image. It is important to ensure that there is sufficient memory space to store the graphic image before downloading. The available remaining space is obtained using Command 34 (see below).

*Note: This command does **not** immediately display the downloaded image. Command 33 must be used to display the image that is downloaded with this command.*

*Note: During graphic downloads the TFT128D SPI interface will automatically issue 0xFF relies to the master while it stores the downloaded information into its flash memory. The SPI master software must recognize this as a BUSY reply from the ScreenKey and continue to reissue the last byte as per the SPI interface description in section 10.2.*

*Note: The TFT128D flash memory has a maximum of 100k write cycles.*

## *Command 33h – Recall Graphic*

**Format:**

| | | |
|---|---|---|
| CMD ID | 33h | |
| XOR | CCh | |
| LEN | 00h/03h | |
| Data | XX | *'x' co-ord for graphic display* |
| | YY | *'y' co-ord for graphic display* |
| | NN | *Graphic ID* |

**Command Data:**

| | |
|---|---|
| XX,YY | 'x,y' co-ordinates of top left position for graphic placement |
| NN | Unique graphic ID |

**Return Data:**

None.

**Description:**

This command is used to recall images that have been downloaded previously using Command 32. The *Graphic ID* (specified in Command 32) is used to uniquely identify the image to be recalled.

Note that this command automatically stops any flash settings setup with Cmd 26h.

## *Command 34h – Report Free Graphic Memory*

**Format:**

| | |
|---|---|
| CMD ID | 34h |
| XOR | CBh |
| LEN | 00h/00h |

**Command Data:**

None.

**Return Data:**

| | | |
|---|---|---|
| Data | MLh | *LSB of available free memory* |
| | MHh | *MSB of available free memory* |

**Description:**

This command is used to check that the ScreenKey has sufficient free space to accept a new graphic download. The ScreenKey returns an integer value (2 bytes) with the amount of free space remaining in the key.

To receive the return information from the TFT128D, the user must issue zero-value bytes (0x00) for each expected return byte. These zero-value bytes must be issued **after** the command trailer is sent (0x55/0xAA/0x00). It is important that the inter-byte delay is observed when issuing the zero-value bytes as otherwise the returned data from the ScreenKey can be undefined or unreliable.

*Note: The TFT128 has approx 9,300 bytes available for general purpose user storage.*

## *Command 35h – Report Memory Contents*

**Format:**

| | | |
|---|---|---|
| CMD ID | 35h | |
| XOR | CAh | |
| LEN | 00h/04h | |
| Data | X1 | *Top left 'x' co-ord* |
| | Y1 | *Top left 'y' co-ord* |
| | X2 | *Bottom right 'x' co-ord* |
| | Y2 | *Bottom right 'y' co-ord* |

**Command Data:**

| | |
|---|---|
| X1,Y1 | 'x,y' co-ordinates of top left position (0-127,0-127) |
| X2,Y2 | 'x,y' co-ordinates of bottom right position (0-127,0-127) |

**Return Data:**

| | | |
|---|---|---|
| Data | CHh | *MSB of pixel color* |
| | CLh | *LSB of pixel color* |

**Description:**

This command requests the actual graphic data in the specified region (i.e. x1,y1 to x2,y2). The ScreenKey reports this information pixel by pixel, starting with the top left pixel and moving left to right and top to bottom until all pixel information has been reported. The return information is two bytes per pixel which reports the exact pixel color as follows:

| R | R | R | R | R | G | G | G | G | G | G | B | B | B | B | B |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Red | | | | | Green | | | | | | Blue | | | | |
| CH (MSB byte) | | | | | | | | CL (LSB byte) | | | | | | | |

This command may be used to extract some specific graphic from the ScreenKey display. This could then be manipulated and then rewritten to the same location using command 22.

To receive the return information from the TFT128D display, the user must issue zero-value bytes (0x00) for each expected return byte. These zero-value bytes must be issued **after** the command trailer is sent (0x55/0xAA/0x00). It is important that the inter-byte delay is observed when issuing the zero-value bytes as otherwise the returned data from the ScreenKey can be undefined or unreliable.

*Note: The maximum allowed requested number of return bytes is 512. Multiple commands may be issued if more than 512 bytes are required.*

## *Command 36h – Report Product Version*

**Format:**

| | |
|---|---|
| CMD ID | 36h |
| XOR | C9h |
| LEN | 00h/00h |

**Command Data:**

None

**Return Data:**

| | | |
|---|---|---|
| Data | PP | *Product Type* |
| | VV | *Version Number* |

**Description:**

This command requests the product type and version number. The ScreenKey reports this information as two bytes:

| | | |
|---|---|---|
| PP | Product Type | *always 01h for TFT128D ScreenKey Display* |
| VV | Version Number | *current version 20h* |

To receive the return information from the TFT128D display, the user must issue zero-value bytes (0x00) for each expected return byte. These zero-value bytes must be issued **after** the command trailer is sent (0x55/0xAA/0x00). It is important that the inter-byte delay is observed when issuing the zero-value bytes as otherwise the returned data from the ScreenKey can be undefined or unreliable.

## *Command 37h – Report Checksum*

**Format:**

| | | |
|---|---|---|
| CMD ID | 37h | |
| XOR | C8h | |
| LEN | 00h/01h | |
| Data | MM | Item to checksum |

**Command Data:**

| | |
|---|---|
| MM | Item to checksum - 00 = CPT, 01 = Font, 1X = Downloaded Image 'X' ID |

**Return Data:**

| | | |
|---|---|---|
| Data | CLh | *LSB of checksum* |
| | CHh | *MSB of checksum* |

**Description:**

This command requests the computed checksum of the referenced item. The ScreenKey reports the 16-bit checksum as two bytes (lsb first):

| | |
|---|---|
| CLh | Checksum LSB |
| CHh | Checksum MSB |

To receive the return information from the TFT128 ScreenKey, the user must issue zero-value bytes (0x00) for each expected return byte. These zero-value bytes must be issued **after** the command trailer is sent (0x55/0xAA/0x00). It is important that the inter-byte delay is observed when issuing the zero-value bytes and that the busy bit is monitored and adhered to as otherwise the returned data from the ScreenKey can be undefined or unreliable.

The checksum calculation is a 16-bit sum of all the bytes related to the referenced item. For example, the original Color Palette Table consists of 512 bytes and has a checksum-16 value of 0x5761 (LSB = 61h, MSB = 57h). The original onboard font has a checksu-16 value of 0x4042.

## *9.10. Programming Example*

This example describes how to display the text “Hello World” with blue foreground color and yellow background on a TFT128D in portrait mode.

Issue reset to key on power up, set orientation to portrait

**0x01 0xFE 0x00 0x02 0x00 0x01 0x55 0xAA 0x00**

*Always issue key reset after power on*

Clear display to “yellow” background color

**0x13 0xEC 0x00 0x01 0x0B 0x55 0xAA 0x00**

*Clear Display uses Color Reference Table (yellow = 11 dec = 0B hex)*

Set cursor position to 26,37

**0x12 0xED 0x00 0x02 0x1A 0x25 0x55 0xAA 0x00**

*Reset positions cursor position to 0,0 but need text to be centered on screen*
*Need to set new cursor position to ensure text is located as desired*

Display text “Hello”

**0x20 0xDF 0x00 0x07 0xCB 0x01 0x48 0x65 0x6C 0x6C 0x6F 0x55 0xAA 0x00**

*Best performance is achieved if only the “on” pixels are painted on screen*
*Use bit operation 01 to achieve this*

Set new cursor position to 26,73

**0x12 0xED 0x00 0x02 0x1A 0x49 0x55 0xAA 0x00**

*Normal cursor position updates to next character or beginning of new line*
*Sometimes cursor position needs to be explicitly set for desired formatting*

Display text “World”

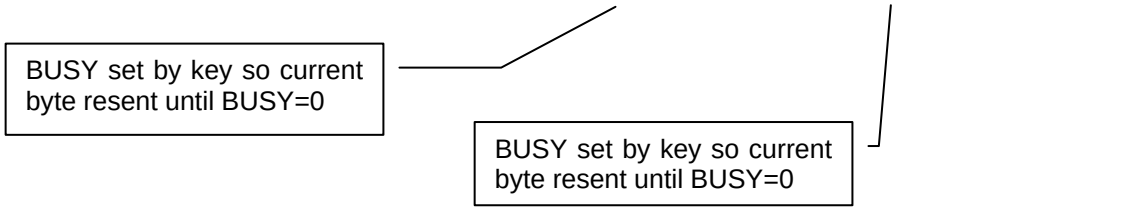
**0x20 0xDF 0x00 0x07 0xCB 0x01 0x57 0x6F 0x72 0x6C 0x64 0x55 0xAA 0x00**

*Use bit operation 01 to oonly write “on” pixels for best performance*

This example demonstrates how WAIT is handled:

Set new cursor position to 26,73

**0x12 0xED 0x00 0x02 0x1A 0x49** *0x49* *0x49* **0x55** *0x55* *0x55* **0xAA 0x00**



# 10. High-Speed Mode

In *High-Speed Mode* the TFT128D ScreenKey Display is focused solely on sequentially populating the active screen area with pixel data. In this mode, the inter-byte delay is significantly reduced and the display does not implement the busy/cmdok handshake policy required with command mode.

Command mode does not offer any of the high-level commands described in the section above. High-speed mode provides direct access to the graphic area of the ScreenKey display and all rendering is the responsibility of the application or system integrator.

In High-Speed Mode, there is no data or status information returned from the ScreenKey. The data that is placed on MOSI line by the master appears directly on the MISO line in the next byte transfer sequence. If only using the key in this mode, the MISO line may be left unconnected.

The achievable frame refresh rate is subject to the size of the defined video sub-window (see command 02 in section 10 above). If using the full screen area of 128*128 pixels, then the refresh rate in high-speed mode will be just over 9 frames per second. However, using a reduced video sub-window greatly reduces the number of bytes to be transmitted and so increases the frame refresh rate. For example, with a 16:9 video aspect ratio, a video sub-window of 96*54 pixels could be defined in the centre of the TFT128. A static frame could be painted around this sub-window which defines or names the video stream. Using a 96*54 video sub-window almost 29 frames per second could be achieved.

## *10.1. Activating High-Speed Mode*

By default, on power-on, the TFT128D starts in command mode. To activate high-speed mode, it is necessary to issue the following command sequence:

| *Header* | | | | *Data* | | *Trailer* | | |
|---|---|---|---|---|---|---|---|---|
| CMD ID | XOR | LEN (MSB) | LEN (LSB) | mode | orientation | byte1 | byte2 | byte3 |
| 0x01 | 0xFE | 0x00 | 0x02 | 0x01 | 0xRR | 0x55 | 0xAA | 0x00 |

The “mode” byte defines the required operating mode of the display:

0x00 Command Mode
0x01 High-Speed Mode

The “orientation” byte defines the orientation of the display (see section 3.3 above) and the “wipe” direction:

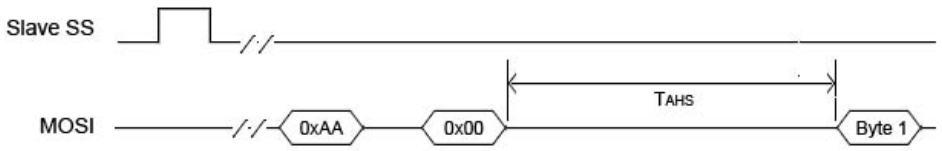
0x_0 Reset to *Landscape*
0x_1 Reset to *Portrait* (default)
0x0_ Set graphic wipe direction from top-left downwards (default)
0x1_ Set graphic wipe direction from bottom-left upwards

The wipe direction defines the position of the top left pixel position (top-down) or bottom left pixel position (bottom-up) which is the starting point from where the graphic display will be populated. The actual pixel position on screen is defined by the active video sub-window.

During transmission of this command, the ScreenKey will return the ScreenKey Status Byte until the final trailer byte (byte3) is sent. Thereafter the data received from the master will be reflected back on the MISO line.

The command mode inter-byte delay of 14.5 microseconds should be observed while transmitting this sequence. Data transmission after this reset command has been issued should use the 2.25/2.75 microsecond inter-byte delay for high-speed mode (see section 9 for specific information).

When initiating high-speed mode, the following timings should be observed:




| Symbol | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| $T_{AHS}$ | Delay after entering HS before 1st byte | 26.0 | | us |

## *10.2. Sending Graphical Data*

As already stated, high-speed mode only supports direct display of graphical data on the ScreenKey's display. The amount of data to be transmitted depends on the defined video sub-window (Cmd02). A full screen video sub-window image requires:

**128 pixels wide** by **128 pixels high** by **16-bit pixel color** (2 bytes) = **32,768 bytes**

A 16:9 aspect ratio video sub-window using 96*54 pixels requires:

**96 pixels wide** by **54 pixels high** by **16-bit pixel color** (2 bytes) = **10,368 bytes**

Similar calculations can be made for different video sub-window sizes.

Each pixel is defined by two bytes which are sent sequentially – high byte first:

| R | R | R | R | R | G | G | G | G | G | G | B | B | B | B | B |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Red | | | | | Green | | | | | | Blue | | | | |
| MSB | | | | | | | | LSB | | | | | | | |

The order in which pixels are drawn depends on the wipe direction. Each line is drawn left to right in all cases, with the wipe direction determining if lines are drawn from top to bottom or bottom to top.

When the last pixel in the defined video sub-window has been painted, the internal cursor position automatically sets itself to the first pixel in the video sub-window. Using this process, it is possible to continually transmit video frames sequentially without having to deselect the ScreenKey (i.e. SS).

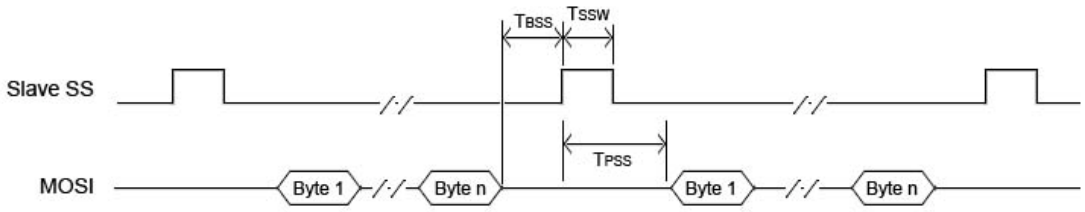
However, the SS line provides the opportunity for the master to explicitly instruct the TFT128D to begin a new video frame. This is done by raising the SS line for a pre-set period and then re-selecting the key (see section on Frame Synchronization below). The start of a new frame is detected when the SS (Slave Select) line is activated (pulled low). This resets the cursor position to either the top left or bottom left pixel position of the active video sub-window (referenced to the defined orientation and graphic wipe direction).

## 10.3. Frame Synchronization

As there is no return feedback from the display, it is critical that the ScreenKey and master have a method for synchronization of frames. This is achieved by the SS (Slave Select) line.

The ScreenKey identifies when the SS line is raised (deselecting the current key) and recognises this as an end of frame. The cursor position is automatically returned to the first pixel position in the active video sub-window in preparation for the next frame transmission.

To use this synchronization, the master must ensure that the SS line is raised and lowered between each frame transmission:




| Symbol | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| $T_{SSW}$ | Width of SS for Frame Sync | 2.0 | 5.0 | us |
| $T_{BSS}$ | Delay after last byte before SS Frame sync | 2.75 | | us |
| $T_{PSS}$ | Delay after SS Frame sync before 1st byte | 11.75 | | us |

Drawing: High-speed frame transmission with frame sync

The frame synchronization timings shown in the table above must be observed.

## 10.4. Screen Refresh Rate

Each pixel is defined by a two-byte value (16-bit color). A full screen refresh requires 128*128*2 bytes, i.e. 32,768 bytes. A reduced size video sub-window requires less data, for example 96*54 pixel window @ 2 bytes per pixel requires 10,368 bytes.

The following formula can be used to determine the frames refresh time:

**{ Num horiz pixels * Num vert pixels * [ (16*10$^6$/SPIClk) + 2.25us + 2.75us ] } + 2.0us + 9.75us**

The “frames per second” value is 1 sec divided by this frame refresh time.

For example, using 16:9 aspect ration video in a sub-window of 96*54 pixels with a 10MHz SPI clock:

**Frame refresh = { 96 * 54 * [ 1.6us + 2.25us + 2.75us ] } + 2.0us + 9.75us**

**= { 5,184 * 6.6us } + 11.75us**

**= 34,226.15us = 34.226 milliseconds per frame**

The screen refresh rate at 96*54 is therefore ~29 frames per second.

Using the full screen area of 128 * 128 pixels, the frame refresh time is 108.146ms, which equates to ~9 frames per second.

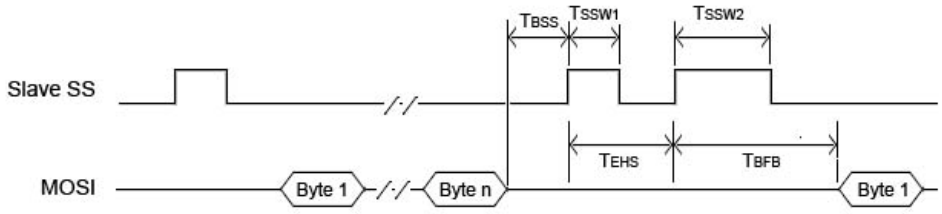
## *10.4. Exit High-Speed Mode*

It is possible to exit high-speed mode and return to command mode.

To do this, the master should raise the SS line (deselect the ScreenKey), then lower the SS line (reselect key) but **do not send any data**. Then again raise the SS line to deselect the display. The ScreenKey is then restored to command mode and will respond to normal commands including returning the SSB.

Always issue 00h bytes to the key before sending a new command to ensure the SSB is being returned correctly.

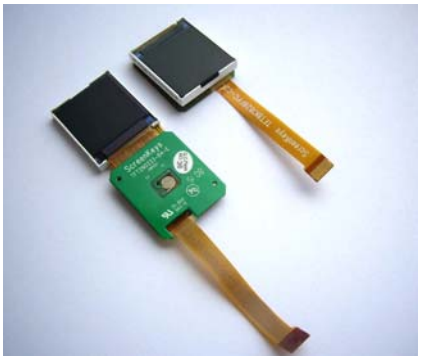
The following timings should be observed when exiting high-speed mode:




| Symbol | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| $T_{SSW1}$ | Width of first SS | 2.0 | 5.0 | us |
| $T_{BSS}$ | Delay after last byte before 1st SS High | 2.75 | | us |
| $T_{EHS}$ | Delay after 1st SS High before next SS High | 11.75 | | us |
| $T_{SSW2}$ | Width of second SS | 2.0 | | us |
| $T_{BFB}$ | Delay before 1st byte of next packet | 13.5 | | us |

Drawing: Timings for exiting high-speed mode

# 11. Order Information

| Order No. | Description |
|---|---|
| TFT128D | TFT128D ScreenKey Display with full-color 128*128 display resolution  |

# 12. Contact Information

For further information on TFT ScreenKeys, RGB and LC Series ScreenKeys and other information, including technical documentation, datasheets, user manuals, software downloads, development and prototyping tools, please visit our website at: www.ScreenKeys.com or email us at info@screenkeys.com.

**SK Interfaces Ltd**
**Unit 11, Keypoint Business Park,**
**42 Rosemount Park Drive,**
**Ballycoolin Road, Dublin 11, Ireland.**
**Tel: +353-1-88 55 075**
**Fax: +353-1-88 55 095**

# Important Notice

## *Warranty Disclaimer*

SK INTERFACES LIMITED GRANTS NO WARRANTY WITH RESPECT TO THIS DATA SHEET, NEITHER EXPLICIT NOR IMPLIED, AND IT IS NOT LIABLE FOR DIRECT OR INDIRECT DAMAGES. SOME STATES DO NOT GRANT THE EXCLUSION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES AND, THEREFORE, THIS STATEMENT MAY NOT BE VALID IN SUCH CASES.

## *Copyright Notice*



This data sheet has been produced with all due care. However, since errors cannot be excluded, SK Interfaces Limited does not grant any warranty or accept any legal responsibility or liability in any form for erroneous statements herein.

## *General Notice*

This data sheet is intended for technically qualified personnel trained in the field of electronics.

The knowledge and technically correct implementation of the content of this data sheet are required for problem free installation and safe operation of the described product. Only qualified personnel has the required know how to implement the specifications given in this data sheet.

For clarity, not all details regarding the product or its implementation, installation, operation, or maintenance have been included. Should you require additional information, please contact SK Interfaces Limited or visit our website at www.ScreenKeys.com.