Freescale Semiconductor
Application Note

Document Number: AN3493
Rev. 0, 07/2007

XGATE Library: Driving a TFT LCD Panel

by: Steve McAslan, MCD Applications, East Kilbride
Davor Bogavac, Field Applications, Göteborg

1 Introduction

This driver allows an S12X to directly drive a TFT LCD panel using general-purpose I/O and the XGATE. The driver provides all timing and data signals required by the panel and a simple color encoding scheme that minimizes the size of the internal RAM required to store complex graphics. The implementation described supports a panel up to approximately 30,000 pixels with 16-bit color resolution.

2 TFT LCD Panels

The rapid increase in demand for color graphics in embedded applications has led to the wide availability of cost-effective panels ranging in density from a few thousand pixels to millions of pixels. The panels are manufactured using a variation of the well-known Liquid Crystal Display (LCD) technology involving the use of Thin Film Transistors (TFTs) that are fabricated onto the glass panel of the display. For more information on the underlying LCD technology and how XGATE can directly drive LCD segments please refer to Freescale Application Note AN3219.

Contents

freescale™ semiconductor

2.1 General Features of TFT Panels

The key difference between the standard LCD panel and a TFT panel is that each segment in the panel has a transistor directly connected to it which allows for much larger numbers of segments without a loss of picture quality. This in turn allows the display of full color information because each pixel on the panel is made of three different subpixels colored red, green, and blue respectively, each of which can display a range of intensities.

The panel arranges the pixels in a rectangular matrix that has the format *p* x *l* where *p* is the number of pixels in a line and *l* is the number of lines on the panel. Examples of panel matrices include 320 x 240, 640 x 480, and 320 x 96 (the main example in this application note). The total number of pixels in a panel is the product of the two numbers, for example the 320 x 96 panel has 30,720 pixels.

Each complete set of pixels on a panel is called a frame and there is a minimum rate at which the controller must supply each frame. This is because the subpixels behave like capacitors and the color will fade after a short period of time. If the controller does not provide the next frame fast enough then a flicker becomes visible to a viewer. The rate at which the controller must refresh the frame is typically in the range of 50 to 80 frames per second.

A second difference between the standard LCD panel and a TFT panel is that the external interface is completely changed between the two. The LCD panel arranges the segments in a passive matrix that requires the supply of bias voltages that use multiple voltage steps to enable each segment. The TFT panel uses an active matrix arrangement that requires on-panel electronics and so the external interface more closely resembles that of a standard parallel bus with normal CMOS logic levels.

The characteristics of the parallel bus vary from panel to panel and depend on the panel's feature set but in general the format is as follows:

- A parallel data bus of 3 x *n* bits where *n* is the number of bits used to indicate the intensity of that color. The panels discussed here use 6 bits of information for each color; this requires an 18-bit parallel interface.
- A pixel clock that indicates when data on the parallel bus has changed
- A data synchronization signal that indicates when data is valid on the parallel bus
- A horizontal synchronization signal that indicates the start of a new line of pixels
- A vertical synchronization signal that indicates the start of a new frame of pixels

NOTE

For most panels the interface operates in one direction with the source being the graphic controller chip or in this case the S12X microcontroller.

So that the pixels are visible the panel has a backlight provided by either Cold Cathode Fluorescent Lighting (CCFL) or LEDs. If the display uses a CCFL then a separate voltage supply is also required to drive this bulb. There are standard sub-assemblies available that generate the high voltage required (>1000 V) from +5 V or +12 V supplies. Contact the TFT panel supplier for recommended sub-assemblies. Where the panel uses LED technology the backlight can often use the same CMOS supply as the display's electronics hardware.

3 Hardware Interface to the TFT Panel

The TFT panel interface operates using CMOS voltage levels which are usually 0 V—5 V or 0 V—3.3 V. These ranges are comfortably within the normal operation of most S12X devices and so the general-purpose input/output (GPIO) pins of the MCU can connect directly to the TFT panel with no extra circuitry required.

3.1 Hardware Connections

In this application note the parallel data bus is restricted to 16 bits of unique data. The reason for this is that it is impossible to output 18 bits of data in a single operation from the MCU. The conventional approach to this problem is to connect the least significant red data bit to the second-least significant red bit and then to repeat the same operation on the equivalent blue bits. This provides 5 bits of red and blue intensity and 6 bits of green intensity. This format is known as RGB565. The XGATE writes this 16-bit pixel data out on a 16-bit compatible port on the MCU.

The pixel clock is a high frequency signal that contains only timing information. The panel manufacturer specifies the frequency of the pixel clock, which is loosely related to the size of the frame and the frame refresh rate. In the example discussed here the frame size is 30720 pixels (320 x 96) and the refresh rate is 60 Hz, which translates to an approximate pixel clock of 1.8 MHz. In practice the panel requires various hold-off and blanking times for correct operation and the clock specified is actually 2.5 MHz. The MCU generates this frequency by configuring a PWM signal with a duty one-half of its cycle time and at the required pixel clock frequency. For example for a 40 MHz bus clock and a 2.5 MHz pixel clock configure the PWM with a cycle count of 16 (40/16 = 2.5) and a duty of 8.

The remaining synchronization signals are simple logic levels that the XGATE asserts at the appropriate time in the refresh of the frame. Standard GPIO pins are all that is required and because these signals are asserted at different times during the frame there is no need for them to be on the same GPIO port. The following figures illustrate how these signals appear on a typical interface.

Figure 1 shows a typical hardware connection between a TFT panel and the S12X. The exact requirements of the interface vary slightly between manufacturers. The example shown is for an Optrex T-51686GD049H-FW-AA panel and it is this panel that the included software drives. It may be necessary to modify the hardware connections and software timings to match the display of your choice.

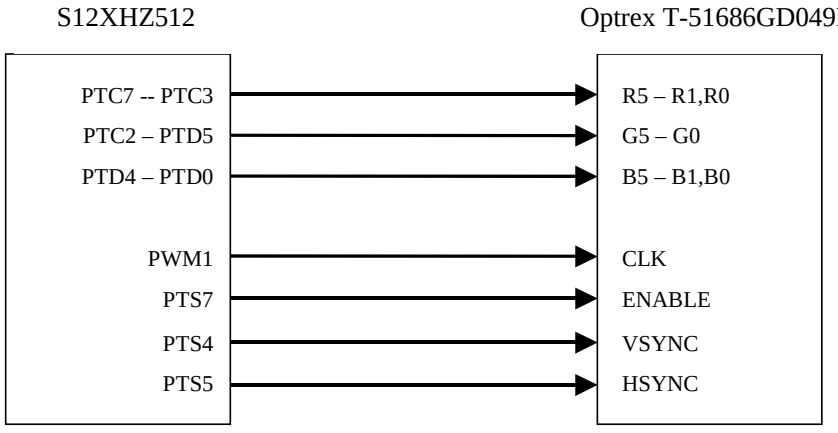

Figure 1. Hardware Connections for Optrex Display

3.2 Hardware Timing

As in the case of the hardware connection, the specific TFT panel determines the timing description of the interface.

Figure 2 shows the timing for each pixel. The PWM is free running and each time a falling edge occurs the XGATE changes the value on PORTC and PORTD to that of the next pixel. The XGATE detects the presence of this edge by reading the PWM port bit.

Figure 3 shows the timing for a line of this display.

Figure 4 shows the timing for a full frame of data.

Note the horizontal and vertical blanking times where no pixel data are written to the panel; this timing opens the opportunity for XGATE to perform another function or to allow synchronization between the XGATE and CPU. Because the data are written to the display in bursts of one line at a time this forms the basis of the timing of the XGATE software.

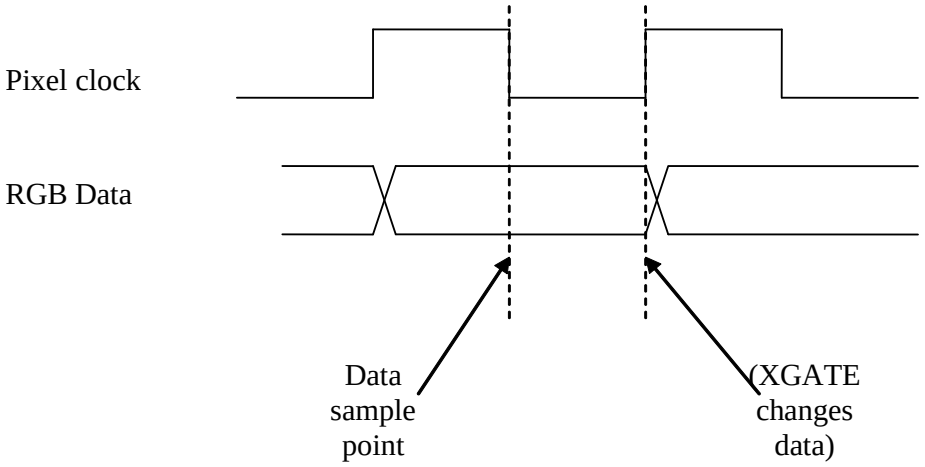

Figure 2. Panel Bit Timing

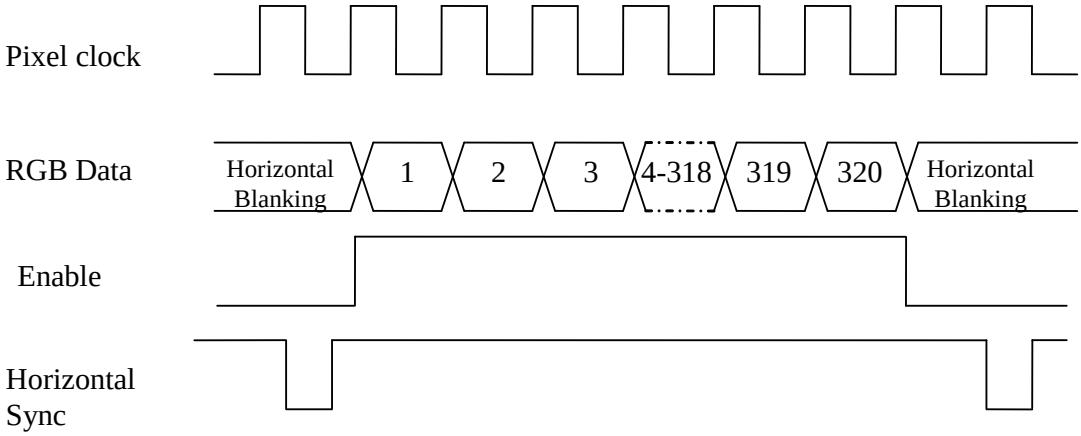

Figure 3. Panel Line Timing

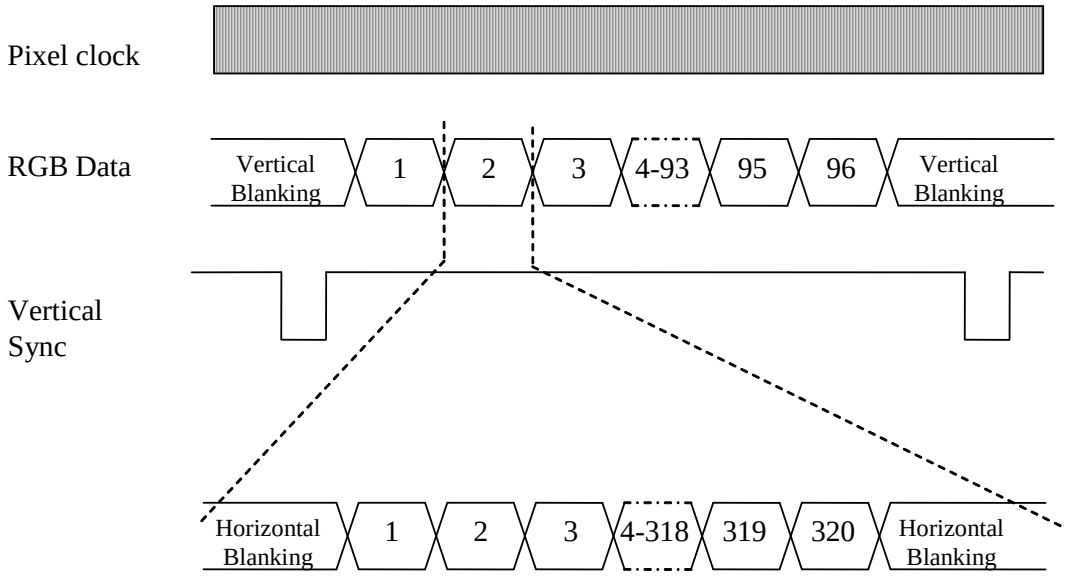

Figure 4. Panel Frame Timing

4 Software Architecture

The functionality of the TFT driver breaks down into two main tasks:

1. Real-time updating of the frames on the panel
2. Management of the content of the frames

In this implementation each of the two processor cores (CPU and XGATE) run one of the tasks and communicate using a buffer in RAM.

Figure 5 shows the architecture of the software and the buffer. In this application the structure of the buffer defines the implementation detail of the software and so this note describes it first and then returns to explain the details of the software tasks.

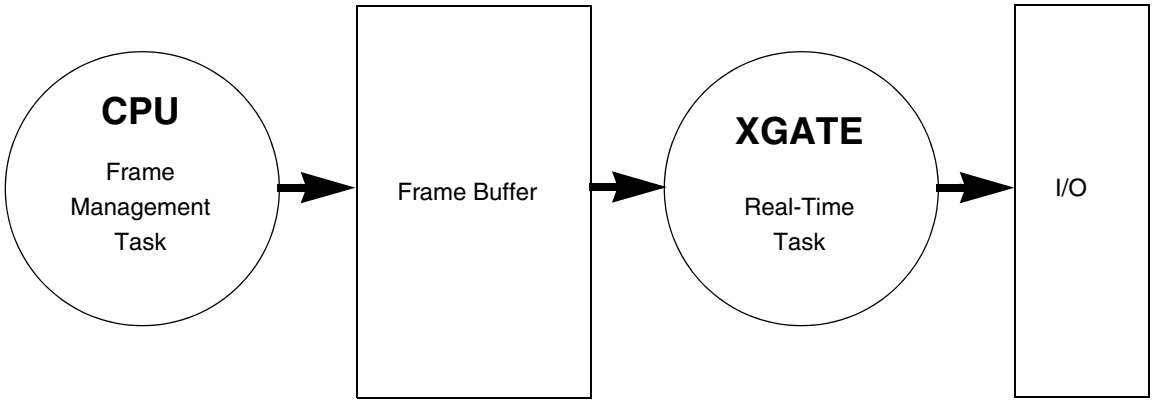

Figure 5. Software Architecture

5 Frame Buffer

The frame buffer contains all the information required to display a single frame on the panel. The simplest form of frame buffer is one where each entry contains the value of one pixel on the panel. The real-time task simply reads the value of the pixel from the buffer and writes it to the panel. In practice it is not efficient to implement the frame buffer in this way because even small panels require more RAM than exists on a typical microcontroller. In our example the 320 x 96 panel would require 60 Kbytes of RAM for each frame.

A more efficient approach is to encode the pixel information such that each one takes no more RAM than it requires. This not only saves significant amounts of RAM but also reflects the nature of most graphics because an image will rarely use the full 65536 colors available.

The solution presented here uses the well-known approach of Color Look-Up Tables (CLUT) to encode the frame buffer.

5.1 Look-Up Tables

In this approach each entry in the buffer is given a value that is not the color of the pixel but instead is an index to a table. The table stores the actual color (known as the direct color) of the pixel at the index. See Table 1 for an example of a look up table that contains four different pixel colors.

Table 1. Example of Look-Up Table

Index	Pixel Color
1	0x0000 (Black)
2	0xFFFF (White)
3	0xF800 (Red)
4	0x001F (Blue)

Because the index indirectly describes a direct-color by using an index to a table it is described as an indexed color. In some contexts the look-up table is also known as a palette.

Consider a graphic image of 100 x 100 pixels that contains only four colors (for example a company logo). In un-encoded format this image would require 20,000 bytes of information in the frame buffer. By using the CLUT we no longer have to store each pixel as a 16-bit value, instead we store it as a 2-bit value (which allows four entries in the table). This reduces the size of the graphic in the frame buffer to 2500 bytes. Of course the real-time software must obtain the pixel's actual color by reading the value stored in the CLUT before writing it to the display. In addition the CLUT itself requires a small amount of memory.

The driver further enhances the usage of RAM by providing different CLUTs for different parts of the panel display. These CLUTs can be different sizes and can contain different pixel colors. The driver stores the encoding of the frame buffer in a software structure known as a *scene*.

5.2 Scenes

The scene exists independently from the frame buffer and determines the visual structure of the image on the panel. For an example see Figure 6.

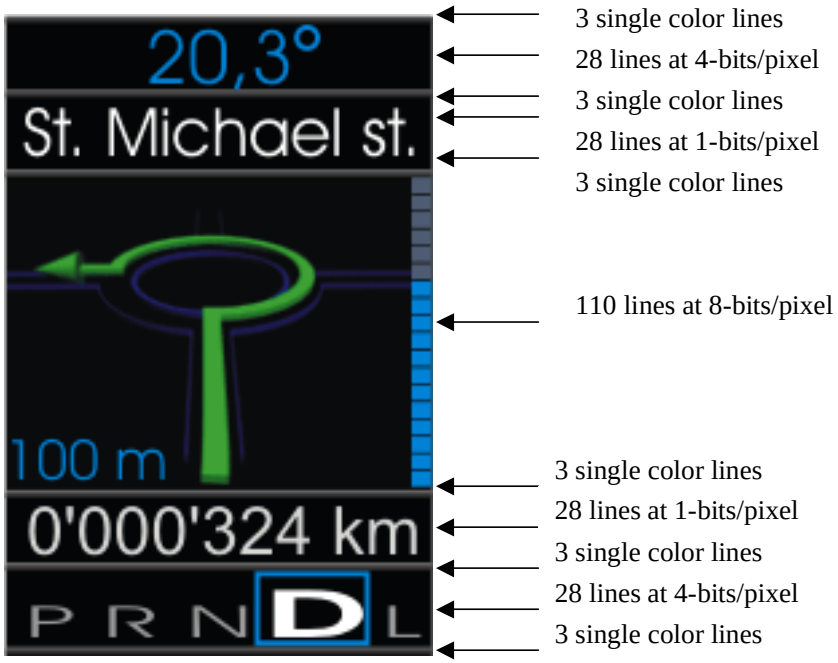

Figure 6. Example of 1/8 VGA Panel (pictured panel is QVGA)

Here we can see that the scene describes an image in 11 different regions giving a total of 240 horizontal lines of pixels. Each scene can contain any number of regions (up to the number of lines on the display). The scene also determines the coding of each region and in Figure 6 we see regions that contain:

- A single color only (for horizontal lines)
- 1 bit per pixel (1 bpp) lines for simple text
- 4 bits per pixel (4 bpp) for simple graphics
- 8 bits per pixel (8 bpp) for more complex graphics

It is possible to encode the pixel data using 2 bits per pixel but that is not shown in this example.

In the example there are 240 lines each of 160 pixels. The total memory required is the sum of:

- 18 lines of single color = 0 bytes (the line color is encoded in the CLUT index)
- 56 lines of 1 bpp = 56 x (160/8) = 348 bytes
- 56 lines of 4 bpp = 56 x (160/2) = 4480 bytes
- 110 lines of 8 bpp = 110 x 160 = 17600 bytes
- Total frame buffer size = 22428 bytes

NOTE

It is the frame buffer that contains the actual graphical information; the scene is simply a structure that describes the encoding of the graphics. To change to different text or graphics within the format shown it is only necessary to change the contents of the frame buffer.

If the panel has to display a different graphical layout then it is possible to create alternative scenes that the user can switch between in software.

In a typical application the user creates the graphical content, encodes it and then includes it in the application along with the driver software. The definition of the scene is part of the graphic design process.

6 Software Implementation

This section describes how the real-time software writes the encoded graphical information to the panel and how the CPU can change the graphics on the panel. It also describes the implementation of the frame buffer.

6.1 Real-Time Software

The purpose of this software is to manage the hardware interface to the TFT panel and to decode the frame information held in RAM. This software runs entirely on the XGATE and provides functionality similar to a hardware graphics controller.

Figure 7 describes a simplified flow of the XGATE driver software.

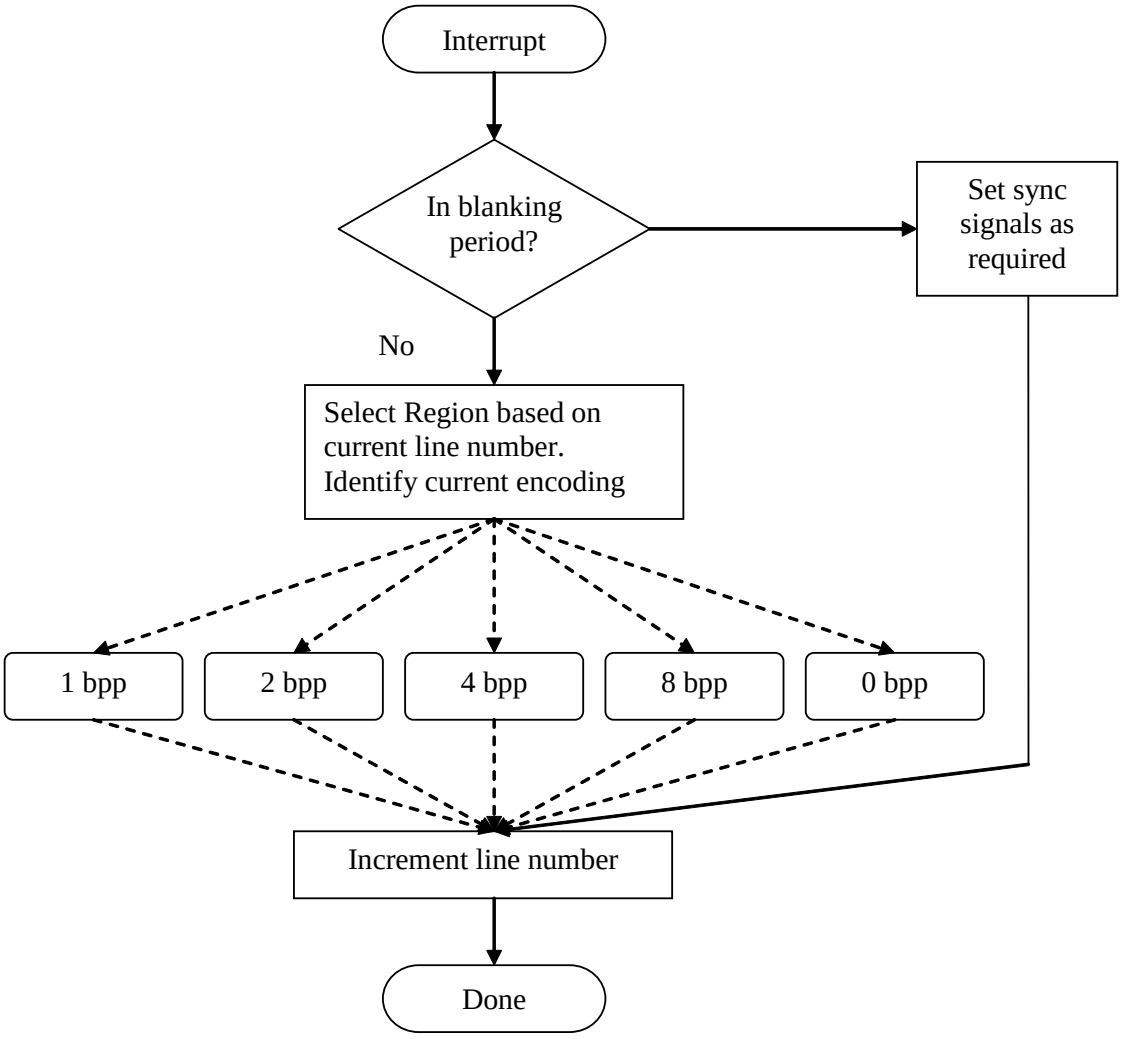

Figure 7. Flowchart of XGATE Software

An interrupt launches this XGATE thread. This interrupt occurs once for every line on the display which is a rate of 6.3 kHz for the Optrex panel. The rate will vary depending on the panel that you choose. The example software uses the PIT module as the source of this interrupt.

Each time the thread executes it updates the line number and after it has written data to every line and completed the blanking period it resets the line number back to zero and repeats the process. The length of the blanking period depends on the display. In the current example, the software assumes a blanking time of 10 lines so the total number of times the thread executes to write a full frame is 96 + 10.

The behavior of the XGATE driver depends on the current line number. For the first 10 lines the thread will set the vertical and horizontal synchronization signals to allow the panel to correctly synchronise.

After the line number exceeds 10 the thread will begin to write data to the panel. For each line, the thread writes exactly 320 RGB565 words to the panel (for the example panel). Before the thread can write the word to the panel it must correctly decode the data in the frame buffer. The decoding process also depends on the line number because this determines the region.

The thread then selects one of five possible decoding algorithms: 1 bpp, 2 bpp, 4 bpp, 8 bpp, or 0 bpp (single color line). Each decoding algorithm reads a byte from the frame buffer, extracts the indexed color for one word, looks up the direct color from the CLUT and writes this direct color to the panel synchronous with the pixel clock.

The extraction of the indexed color varies according to the encoding specified by the region:

- For 0 bpp no extraction takes place, instead the thread writes the direct-color chosen by the region 320 times to the panel
- For 1 bpp each bit in the frame buffer byte represents a pixel on the panel. The thread extracts the direct-color from the 1 bpp CLUT and writes this to the panel. After it has processed all bits in the byte it increments its pixel counter by 8 and checks to see if it has read 40 bytes from memory (= 320/8)
- For 2 bpp each pair of bits in the frame buffer byte represents a pixel on the panel. The thread extracts the direct-color from the 2 bpp CLUT and writes this to the panel. After it has processed all four pairs in the byte it increments its pixel counter by 4 and checks to see if it has read 80 bytes from memory (= 320/4)
- For 4 bpp each nibble in the frame buffer byte represents a pixel on the panel. The thread extracts the direct-color from the 4 bpp CLUT and writes this to the panel. After it has processed both nibbles in the byte it increments its pixel counter by 2 and checks to see if it has read 160 bytes from memory (= 320/2)
- For 8 bpp each frame buffer byte represents a pixel on the panel. The thread extracts the direct-color from the 8 bpp CLUT and writes this to the panel. After it has processed the byte it increments its pixel counter by 1 and checks to see if it has read 320 bytes from memory.

In the example software, the thread is called XGATE_SWISR0 and is found in the file TFTdriver_Optrex_T-51686gd049h.cxgate.

To customize this driver to suit your panel see Section 7, “Configuration.”

6.2 Frame Management Software

The purpose of this software is to change the contents of the frame buffer such that the panel provides new information and animated sequences. If this software does not change the frame buffer then the panel will display the same static image.

In the included example the management software runs on the CPU and most of the functions are specific to the actual graphics in use. For example, there are custom functions to clear and animate sections of the screen. Depending on your requirements it may be more efficient to create or buy a graphics library to manipulate the frame buffer in the way that you specifically require. To help you understand the basic function of the management software we shall consider a simple function that displays a graphic on the panel. Consider the software listing in Figure 8.

```
void InitNavi(){
  tU16 i,x,y;

  /* Load the LUT for the navigation colors */
  for (i=0;i<0xFF;i++){
    LUT2[i] = (tU16)((NaviPanel[i*2] <<8) + NaviPanel[(i*2) + 1]);
  }
  y=0;
  for (i=0;i<70;i++){ /* load the navigation image */
    for (x=0;x<320;x++) {
      Vram[BPP8_OFFS + y + x] = NaviPanel[(320*i) + x + 512];
    }
    y+=HSIZE;
  }
}
```

Figure 8. Loading a Graphic into the Frame Buffer

The listing shows that there are two steps to loading an image into the buffer:

1. The software initializes the appropriate CLUT with the correct direct colors
2. The software copies the image into the frame buffer at the correct location

Let’s look at each step in turn. It is possible that all graphics shown on the panel can use the same palette of colors; however, it is much more desirable to allow each to have its own. The software therefore copies the 256 direct-colors (for an 8 bpp image) into the appropriate CLUT. The CLUT for the image is usually stored in Flash. It is possible to avoid this copy operation by accessing the image CLUT directly but this costs extra RAM or slows the XGATE software operation.

The second step is to copy the image into the frame buffer. In this case the copy simply begins at the first pixel and continues until the entire image is in the frame buffer. This software example takes account of the number of lines on the display by using the HSIZE parameter. It is possible to design more complex copy operations that can start and stop at any pixel on the panel. These operations are commonly known as “bit blitting” (for bit **bl**ock **t**ransfer) and place an image of any size at any location on the panel. You can find implementation details of this type of copy in graphics textbooks or on the World Wide Web.

See Section 10, “Creating the Content” for descriptions of other more complex software functions included in the example.

6.3 Software Resource Sharing

The example software has the real time and frame management software running asynchronously to each other. In other words it is possible for the CPU to update the contents of the frame buffer at exactly the same time as the XGATE is reading it. In the software example the CPU changes graphics faster than the human eye can detect (about 40 Hz) and so this asynchronous approach yields acceptable performance. It may be necessary in more complex cases for the CPU and XGATE to update the frame buffer completely independently.

The blanking times of the TFT interface allow an opportunity to synchronize the cores because the XGATE has no requirement to copy data from the frame buffer to the panel at this time. To implement this change the XGATE can send an interrupt to the CPU when the blanking period is about to start. The CPU then updates the frame buffer in the interrupt service routine or schedules a task to do this. Of course the CPU must complete its update to the frame buffer within this blanking period because the XGATE cannot delay making the next update.

6.4 Frame Buffer

You define a scene at design time by creating an array of type Scene_t and initializing it. Each entry in the array defines the size, coding, and color offset of a region in the scene. The array implicitly defines the region order such that the first element in the array defines the configuration of the first line and the last element configures the last line on the display.

Figure 9 shows the type declaration, definition, and initialisation of a simple scene. In the example code the scene definition is in the S12Xgraphics.c file. When configuring the scene or scenes you should ensure that the frame buffer size can accommodate the largest scene.

```
typedef struct {
  tU16  line    :16;
  tU08  type    :8;
  tU08  c_offs  :8;
}Scene_t;

volatile Scene_t Scene1[16];

Scene1[0].line = 10;
Scene1[0].type = 1;    //1bpp
Scene1[0].c_offs = 0;

Scene1[1].line = 11;
Scene1[1].type = 0;    //SCL
Scene1[1].c_offs = 9;

Scene1[2].line = 12;
Scene1[2].type = 0;    //SCL
Scene1[2].c_offs = 0xD;

Scene1[3].line = 82;
Scene1[3].type = 8;    //8bpp
Scene1[3].c_offs = 0;
```

Figure 9. Scene Definition, Declaration, and Initialization

For each entry in the Scene array the parameters set in the Scene_t structure determine the configuration of the region:

- Line defines the last line that belongs to the region. The line count starts at 0.
- Type defines the coding for that line (and associated CLUT) and takes the values of 0, 1, 2, 4, or 8 where 0 indicates a single color line, 1 indicates 1 bpp, 2 indicates 2bpp and so on.
- Coffs is used when you want to increase the number of colors available for a particular coding method – see Section 6.5, “Color Offset Parameter” for further details

In Figure 9 the software initializes Scene1[0].line to 10 and Scene1[0].type to 1. This causes the first region on the display to consist of line numbers 0 to 10 and the coding to be 1 bpp.

The software then sets Scene1[1].line to 11 and Scene1[1].type to 0 which causes the second region to consist of a single line (line 11) in a single color defined by the coffs parameter. This process continues for each region until every line in the scene has a coding definition.

6.5 Color Offset Parameter

You can use the coffs parameter if you want to adjust the colors used in a particular region. This member acts as an offset into the CLUT and so adjusts the direct color that an indexed color actually selects.

The driver uses a single CLUT for each coding scheme to maximize efficient use of memory. In the case of the 1 bpp scheme the index color is a single bit with the values 0 or 1 and therefore the usual size of the CLUT would be two entries. To enhance the functionality of the driver it is possible to create a larger table (containing more direct colors) than the coding method can represent. In the case of 1bpp, we could have a 16 entry CLUT which would allow the choice of different colors for different 1bpp regions in the scene. In the software the value of Scene1[n].coffs is added to the value of the indexed color to select the required entry in the CLUT.

The benefit of this approach is that it is possible to share a single CLUT among different regions. It also allows the possibility of the software changing the direct color of certain graphics by simply adjusting the coffs parameter rather than rewriting the direct color in the CLUT.

7 Configuration

In general, configuration of the driver will mostly apply to the TFTdriver.h file. It may also be necessary to modify the TFTdriver.cxgate file to adjust for particular timings on your chosen panel. The configuration of some macros depends on the hardware connection between the MCU and the panel on your design.

See Table 2 for a description of the macros to modify in TFTdriver.h.

The XGATE library files contain descriptions of the various variables used in the software.

Table 2. Configuration Macros

Macro name	Value
VSIZE	Number of pixels in vertical axis of panel
HSIZE	Number of pixels in horizontal axis of panel
pCLKDIVIDER	Divider that sets the frequency of the pixel clock (= E clock/pCLKDIVIDER)
SPL	Definition of port pin connected to the panel ENABLE/SPL pin
VSYNC	Definition of port pin connected to the panel VSYNC pin
HSYNC	Definition of port pin connected to the panel HSYNC pin
HSYNCDELAY	Counter to determine the length of the active horizontal synchronization pulse
DATAPORT	Definition of port connected to the panel data bus
FRONTPORCH	The number of lines in the vertical blanking period

8 Performance

The performance and peripheral usage of the driver to some extent depends on the TFT panel in use. The values in Table 3 apply to the XGATE driver used with the Optrex T-51686GD049H-FW-AA panel.

Table 3. TFT Driver Performance

Parameter	Value
Code size (bytes)	1282
Data size (bytes)	CLUT: 1280; frame buffer: 24200; Scene: 96; Others: 8
Maximum execution time	135 μs
Maximum latency	Consult panel manufacturer timing specification.
XGATE load	76.5%
Peripherals used	PIT channel, PWM channel, 19 General purpose outputs

The data size parameter depends on the graphical images used (and how they are encoded). The value provided is that for the graphics used in the example code.

9 Software Example

Along with the XGATE library driver there is a software example that demonstrates various simple and complex graphic content for the Optrex T-51686GD049H-FW-AA panel. A description of some of the graphics techniques follows in Section 10, “Creating the Content.”

10 Creating the Content

As previously described the CPU is responsible for loading the appropriate graphics into the frame buffer and the XGATE then writes the frame buffer contents to the panel. Creating and manipulating these graphics is beyond the scope of this application note but it is helpful to discuss how to prepare the graphics in a suitable format and include them in the project.

This section also proposes some techniques to create simple animations on the panel.

10.1 Including Graphics in the Project

Simple graphics used with this driver must be in a RGB565 format because this is the format that the XGATE delivers to the panel. However, you may find it easier to create the graphics using a full-quality graphics program and then convert the final design into the RGB565 format. There are various programs and plug-ins that can perform this task. You may need to adjust your design parameters to account for the fact that the resulting gamut on your chosen display may not match the gamut of the original design.

The RGB565 format must have a maximum color depth of 8 bits that use a color palette and be uncompressed. In practice the easiest file format to use is a Windows bitmap (BMP).

After you have the uncompressed RGB565 file you can extract the palette (to use as the CLUT) and the pixel information and store these as C arrays. Open-source programs are available that perform this type of task, for example Hexplorer.

10.2 Simple Animation

You can achieve an animated effect by loading a sequence of pre-rendered bitmap images into the frame buffer. One of the authors created a rotating S12X image (see Figure 10) to evaluate possible design techniques and the performance of the S12X. The rotating image was created using 20 images that are 140 x 70 pixels in size and coded in 8 bpp. Various programs will perform this type of operation including Autodesk 3DS MAX and several freeware and low-cost solutions, for example:

www.pysoft.com,

www.povray.com,

www.openfx.org.

Figure 10. S12X Image that Rotates

After the animation was complete, the author used the design software to extract the 20 individual images in BMP format. The resulting total size of images is 189,560 bytes which would consume a significant portion of most S12X's internal Flash memory. An investigation on the content of these files allowed the author to develop two techniques to reduce the total size to 52.908 bytes. Further reduction may be possible depending on the nature of the graphic sequence.

The first technique recognizes that when the animation tool creates the individual images each has its own optimized palette because the files contain different colors. Therefore, one approach to reduce the total size of the animation is to normalize the color palette. This involves creating a new common palette and adjusting the indexed colors in the individual image files to use this new palette. For the purposes of the investigation the author created a custom program to analyze and convert the images although it is likely that commercially available packages will be able to perform this task as well.

Another benefit of having a common CLUT is that it simplifies loading the next image for the animation asynchronously to the display. Loading a new CLUT for every new image could result in XGATE fetching the wrong direct color because the existing image data uses a different CLUT. A possible workaround for this problem could be to perform a synchronous load in the panel blanking time but this will significantly reduce flexibility and the image size you can use.

In summary, we can say that a common CLUT avoids pixel distortion in the temporal domain.

The second technique reduces the size of the graphics by compressing the background color information. On the rotating graphic the background color is black and always has index 0 in the CLUT. We can reduce the size of the graphic image by compressing it such that every time an indexed color has the value “0” the following byte contains the number of pixels that are the background color. All other values are copied as is.

To re-create the animation on the panel the CPU software decompresses the data for every image and writes it into the frame buffer, image by image.

For more details on how this is done refer to the example software file S12XGraphics.c. Functions InitAni() and RunAni() contain the decompression code and sequence animation.

10.3 Using CLUTs to Create Animation

The example software contains a parking demonstration graphic that simulates a proximity warning to the front and rear of the car — see Figure 11 for a still image from the animation.

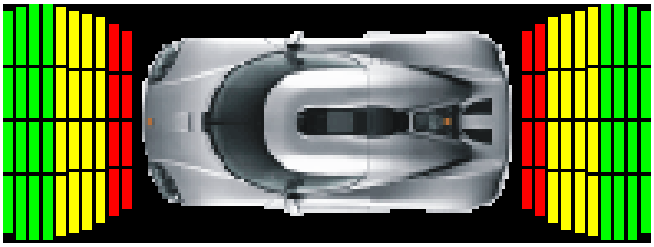

Figure 11. Parking Image

When creating images on the panel the standard approach is to set the indexed color to the value that points to the CLUT entry that creates a pixel in a certain color. In the parking demonstration the sensor element animation is done the other way around.

The sensor animation consists of 40 segments / side. They are drawn such that all pixel data values in a segment are the same, and all segments have a unique value. When the corresponding CLUT entry changes color all the pixels in a segment will get that color. This is a very performance effective way of turning groups of pixels on and off or changing color.

Because changing the CLUT changes the color of every pixel using that color it is important that the car image does not use any of those CLUT entries. There are 80 parking segments and so that leaves only 176 entries in the CLUT that are usable by the car.

The car in this image uses a CLUT that contains only 170 colors. You can create this kind of image using standard graphics packages such as Corel draw and then save them with restricted palettes or use an alternative tool such as IrfanView to convert an image with a larger palette.

In this example the parking segments use the CLUT positions 170—255 and the animation software turns them on segment by segment by loading the preferred direct color into the corresponding CLUT position.

11 Glossary

Term	Meaning
Pixel	The basic graphical element on a TFT LCD panel. Can display a range of colors depending on the value of the red, green and blue values written to it. Normally arranged in a rectangular array.
Segment	Part of a pixel that displays a single color (red, green or blue)
Frame	The collection of all pixels on a panel
Direct color	The full 18-bit value actually written to a pixel to create a color
Indexed color	An index into a table containing direct-colors. Usually smaller in size than the direct color
Scene	An array that contains the color indexing scheme used by the frame buffer
Frame buffer	An array containing the values of all pixels in a frame. In this application the pixels use indexed colors.
RGB565	A color description scheme that describes the color gamut using 5 bits for red, 6 bits for green and 5 bits for blue.
Gamut	The set of colors that a panel can display. In most cases a panel cannot display the full gamut of colors visible to the human eye.
CLUT	Color Look-up table. The table that contains the direct-color values that correspond to the indexed-colors stored in the frame buffer
Panel	A TFT LCD containing an array of colored pixels.
Palette	The set of colors used in a graphic design. Similar to a CLUT.

12 Acknowledgements

The authors would like to acknowledge the contribution of their colleagues Philippe Simon and Jean-Francois Thery in the early development of this application note.

THIS PAGE IS INTENTIONALLY BLANK

THIS PAGE IS INTENTIONALLY BLANK

THIS PAGE IS INTENTIONALLY BLANK

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3493
Rev. 0
07/2007

RoHS-compliant and/or Pb-free versions of Freescale products have the functionality and electrical characteristics as their non-RoHS-compliant and/or non-Pb-free counterparts. For further information, see http://www.freescale.com or contact your Freescale sales representative.

For information on Freescale’s Environmental Products program, go to http://www.freescale.com/epp.

freescale™
semiconductor