IBEX
UNIVERSAL GRAPHIC DISPLAY SOURCE CODE DRIVER PROJECT

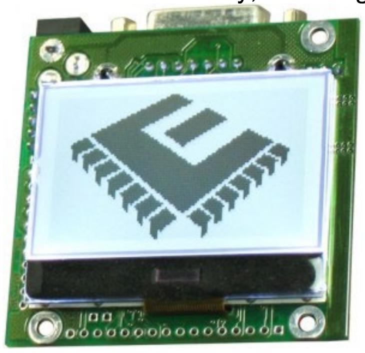
CONTENTS

DRIVER OVERVIEW

OVERVIEW

A typical graphic screen, which may use LCD, E Ink, LED, OLED, Vacuum Fluorescent or some other screen technology, consists of a large array of pixels that are triggered using a X and Y axis matrix of connections.

The actual drive of each pixel is provided by a display controller IC. This may already be built into your screen, may need to be provided by you externally or may be built into the particular microcontroller or processor you are using. This display controller IC will constantly scan the matrix of pixels, turning on or off each pixel based on the data held in the IC's memory. In its basic form the IC will have a data buffer to which you write data to tell the controller IC which pixels you want on and which you want off and for grayscale and colour displays the intensity of each pixel. This driver is designed to be used with any screen which can be controlled in this way, including screens with the simplest controller IC's.



DRIVER FEATURES

Main features:

Designed for use with color, grayscale and monochrome displays.

Use your display in any of the 4 possible orientations.

Display .BMP bitmap files of any size anywhere on the screen.

Display Unicode text strings using user adjustable monospace or proportional width fonts anywhere on a screen.*

Display text strings with left, centre or right alignment. Display a single line of text, or multiple lines contained within a definable region.

Design your screen layouts in code or as HTML files using Dreamweaver web authoring software. This driver parses HTML source files.

Optional touchscreen support – parsed HTML file links may be passed to your application with screen coordinates so it can dynamically process screen touch events

Use the included PC Display Files Converter Application to convert your source font, bitmap, html and css files ready for use in your application.

No reliance on special display Controller IC features – generic driver suited to all types of display controllers where data is written per pixel.

No requirements to use your display in pre-defined ways – use your display exactly as you want with your user interface implemented exactly as you want it, down to the individual pixel!

Fast optimised source code.

No reliance on compiler specific libraries.

Full source code supplied for you to use and modify as required.

(* N.B. To create new fonts and edit fonts you need to purchase BitFontCreator Pro 3.0 from Iseatech Software)

DRIVER INTRODUCTION

Graphic display allow embedded devices to provide rich graphical user interfaces and at ever reducing cost. However successfully using graphic displays comes at the cost of a great deal of complexity. The solution for some engineers is to use an off the shelf GUI module from a vendor which incorporates a processor and proprietary software to provide the graphic display functionality for them, but at a much higher production cost. With colour displays starting at < $10 these days and with typical low cost embedded processors being more than capable enough of driving them paying for GUI modules can be a very false economy. This universal graphic display driver removes much of the complexity of driving graphic displays and allows you to simply display bitmaps of any size and Unicode text strings using user adjustable fonts anywhere on a screen. Once you have this capability using a graphic screen is easy and it’s a simple matter to add any special features you need for your application, such as menus, animated graphics, graphs and more.

Still, designing graphic screen layouts in code, even with a versatile driver, can be a tedious task for more complex projects and a benefit of many GUI modules is that they provide some sort of PC software to facilitate the graphic user interface design. Often this software isn’t particularly friendly or universal, but it’s better than designing from the source code level. This driver takes things one step better and will read HTML files, allowing you to design your screen layouts using Dreamweaver web authoring software if you wish. Not only can this drastically reduce the time and hassle to design user interfaces, it also allows you to design and demonstrate user interface designs before the hardware has even been designed and also allows non programmers to design screen layouts.

The last remaining issue for graphical user interfaces is typically fonts and supporting languages with non ASCII character sets. This driver solves that problem by being designed with full Unicode support and to work with the fonts produced by the great Iseatech Bit Font Creator Pro software. The driver is supplied with several fonts implementing the standard extended ASCII character set which may be used as is. Purchasing Bit Font Creator Pro from Iseatech allows you to edit these included fonts, add new Unicode characters and to also create your own new fonts from any font installed on your Windows PC.

As well as displaying ASCII text strings in standard ‘monospace’ fixed pitch mode (each character takes the same horizontal space regardless of its actual width), this driver also allows you to use font’s created as proportional width. This makes lines of text much more attractive and natural to read and typically allows you to fit additional characters across a screen, as each character only uses as much width as is required to display it. Left, centre or right alignment functions provides the complete toolbox of text display capabilities.

The included Display Files Convert PC application will read all of your source font, bitmap, html and css files and convert them ready for use in your application. It’s output may either be as a standard C header file, allowing all of your source file data to be simply included in your microcontrollers program memory when you compile, or as a binary file for you to store in external flash memory with a generated C header file used by the driver to locate each files start address.

Once you have these capabilities using a graphic screen is easy and it's a simple matter to add any special user interface features you require for your application, such as menus, animated graphics, graphs and more. One of the great advantages of this driver is that it provides you with very simple but very powerful control over everything you display on the screen, allowing you to implement your display, and any user interface functionality, exactly as you want it down to the individual pixel.

This driver can't quite remove all of the complexity for you as you will still need to configure the screen access functions to suit the particular screen / driver IC you are using. This has been made as simple as possible, and some sample screen models are included for you to copy and modify as required. See the technical manual for more details.

The driver code has been designed using ANSI compliant C compilers. Using the driver with other ANSI compliant C compilers and with other processors / microcontrollers should not present significant problems, but you should ensure that you have sufficient programming expertise to carry out any modifications that may be required to the source code. Embedded-code.com source code is written to be very easy to understand by programmers of all levels. The code is very highly commented with no lazy programming techniques. All function, variable and constant names are fully descriptive to help you modify and expand the code with ease.

The driver and associated files are provided under a licence agreement. Please see www.embedded-code.com/licence.php for full details.

The remainder of this manual provides a wealth of technical information about the driver as well as useful guides to get you going. We welcome any feedback on this manual and the driver.

Adding The Driver To Your Project

Notes About Our Source Code Files

How We Organise Our Project Files

There are many different ways to organise your source code and many different opinions on the best method! We have chosen the following as a very good approach that is widely used, well suited to both small and large projects and simple to follow.

Each .c source code file has a matching .h header file. All function and memory definitions are made in the header file. The .c source code file only contains functions. The header file is separated into distinct sections to make it easy to find things you are looking for. The function and data memory definition sections are split up to allow the defining of local (this source code file only) and global (all source code files that include this header file) functions and variables. To use a function or variable from another .c source code file simply include the .h header file.

Originally variable types BYTE, WORD, SIGNED_WORD, DWORD, SIGNED_DWORD were used to allow easy compatibility with other compilers, however these have now mostly been replaced with uint8_t etc type definitions. Our projects include a ‘main.h’ global header file which is included in every .c source code file. This file contained the typedef statements mapping these variable types to the compiler specific types. Our main.h header file also includes project wide global defines.

This is much easier to see in use than to try and explain and a quick look through the included sample project will show you by example.

Please also refer to the resources section of the embedded-code.com web site for additional documentation which may be useful to you.

Modifying Our Project Files

We may issue new versions of our source code files from time to time due to improved functionality, bug fixes, additional device / compiler support, etc. Where possible you should try not to modify our source codes files and instead call the driver functions from other files in your application. Where you need to alter the source code it is a good idea to consider marking areas you have changed with some form of comment marker so that if you need to use an upgraded driver file its as easy as possible to upgrade and still include all of the additions and changes that you have made.

Step By Step Instructions

Move The Main Driver Files To Your Project Directory

The following files are the main driver files which you need to copy to your main project directory and include in your project:

display.c – The main driver functions

display.h

display-model.c – The lower level functions tailored to a specific screen / controller IC

display-model.h

If the driver includes the display-model files for your specific screen they should be copied from that screens sub-directory, or alternatively see later in this manual for details on creating your own version for a new screen model.

If you will be using HTML files with the driver also add these files:

display-html.c

display-html.h

Move The Generic Global Defines File To You Project Directory

The generic global file is located in each driver sample project directory. Select the most suitable sample project based on the compiler used and copy the following file to your main project directory:

main.h – The embedded-code.com generic global file

Check Driver Definitions

Check the definitions in each of the following files to see if any need to be adjusted for the microcontroller / processor you are using, and your hardware connections.:-

display.h

display-model.h

display-html.h

Check the definitions in the following file and adjust if necessary for your compiler-

main.h

Application Requirements

In each .c file of your application that will use the driver functions use:

#include “display.h”

In each .c file of your application that will use the driver HTML functions use:

#include “display-html.h”

Your application needs to call the following function as part of its power-up initialisation:

display_initialise();

The display functions may now be called whenever you wish to update the display.

Calling this function provides a useful quick test of the display:

display_test();

The screen is blanked, a white 1 pixel border is drawn around the screen edge and three RGB colour fade bars are drawn within it.

Font, Bitmap and HTML Files

If you are using some form of filing system in your application and wish to display source files directly from it this section will not apply as you will deal with storing and retrieving the files yourself. This section applies to files added to the project using the Display Files Converter application. Note separate file access defines are used by this driver so that font’s, bitmap files and HTML files may each be stored using Display Files Converter application in internal or external memory or by some other method (i.e. you could store your fonts in program memory and your bitmap and HTML files in external memory or some other filing system).

Create a new directory in your projects main directory called:

display_files\

Into this copy:

The font (.dat) output files you wish to use.

The bitmap (.bmp) image files you wish to display.

The html and css files you wish to use.

Run the Display Files Converter PC application and select this directory. If you will incorporate the files into your microcontrollers program memory select the 'C Header File' option. If you will store the files data in external flash memory and provide functions to access it select the 'Binary File' option.

The 'C Constant Declaration String' should typically be 'const uint8_t'

Press the 'Convert Files' button and the application will generate the output file(s).

For the 'C Header File' option this will be a file called '\output\display_files_c.h' which the driver includes and accesses automatically.

For the 'Binary File' option this will be:

A file called '\output\display_files_bin.h' which the driver includes and accesses automatically to determine each files address.

A file called '\output\display_files_bin.bin' which you need to load into your flash memory device and provide a function for the driver to read bytes (see the display.h header file for details).

Sample Files and Sample Projects

General Notes

Sample display-model.h screen files are included for sample screens. If using a matching screen model and LCD driver IC they can be used directly or if using a different screen type or driver IC they can be used as a basis for your new screen. See later in this manual for details on creating your own version for a new screen model.

Sample projects are included with the driver for specific devices and compilers. The example schematics at the end of this manual detail the circuit each sample project is designed to work with. You may use the sample projects with the circuit shown or if desired use them as a starting block for your own project with a different device or compiler. To use them copy all of the files in the chosen sample project directory into the same directory as the driver files and then open and run using the development environment / compiler the project was designed with.

Sample Screen Driver Files Included

Solomon Systech web site.

Sample display_model files are included for each of the following screens and driver IC's:-

Microchip Multimedia Expansion Board DM320005

RGB 320 x 240 (1/4 VGA) pixel LCD screen.

Driver: Solomon Systech SSD1926 (external)

Colour: 16bit (5Red|6Green|5Blue)

Interface: 16bit Parallel

Available from Microchip Direct

Anders D035QVGAP2

RGB 320 x 240 (1/4 VGA) pixel LCD screen.

Driver: Solomon Systech SSD1926 (external)

Colour: 16bit (5Red|6Green|5Blue)

Interface: 16bit Parallel

Available from Anders Electronics

Truly OEL9M0065-W-E

Mono grayscale 256 x 64 pixel OLED screen

Driver: Solomon Systec SSD1322UR1 (internal)

Colour: 4bit grayscale white

Interface: 8bit Parallel

See later in this manual for details on creating your own version for a new screen model.

Sample Fonts Incluued

disp_font_5w_7h.dat

Proportional width, max 5 pixels wide, 7 pixels high.

Extended ASCII (Unicode characters may be added)

disp_font_5w_11h.dat

Proportional width, max 5 pixels wide, 11 pixels high

Extended ASCII (Unicode characters may be added)

disp_font_15h.dat

Proportional width, 15 pixels high

Extended ASCII (Unicode characters may be added)

disp_font_22h.dat

Proportional width, 22 pixels high

Extended ASCII (Unicode characters may be added)

disp_font_42h.dat

Proportional width, 42 pixels high

Extended ASCII (Unicode characters may be added)

Sample Projects Included

Microchip C32 Compiler

Compiler:

Microchip C32 MPLAB C Compiler for PIC32 family of 32 bit microcontrollers

Device:

PIC32MX795F512L (used in Microchip PIC32 USB Starter Kit 2)

Sample code writes to a 240 x 240 pixel RGB screen (e.g. the Microchip Multimedia Expansion Board DM320005). The basic code shows some of the driver functions:

Microchip XC32 Compiler

A sample project is not provided, but the driver with display-model “driver files for screen IDS CI064-4073-06A” has been used with XC32 V4.

Sample HTML Files Included

OK, these are not going to win any graphical design awards, but they give a good basic example of working files for a 320 x 240 pixel RGB color screen.

global.css

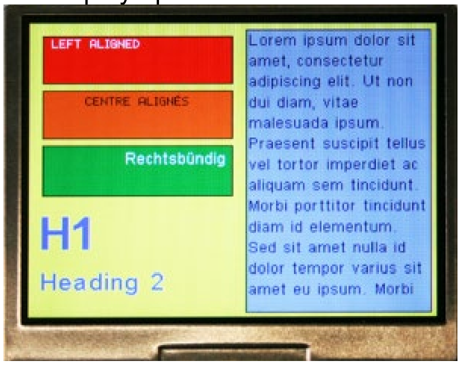
An example of a global style sheet file which may be used by the .htm files

index.htm

A basic example of an image and text.

text1.htm

An example of the various text display options



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut non dui diam, vitae malesuada ipsum. Praesent suscipit tellus vel tortor imperdiet ac aliquam sem tincidunt. Morbi porttitor tincidunt diam id elementum. Sed sit amet nulla id dolor tempor varius sit amet eu ipsum. Morbi

images.htm

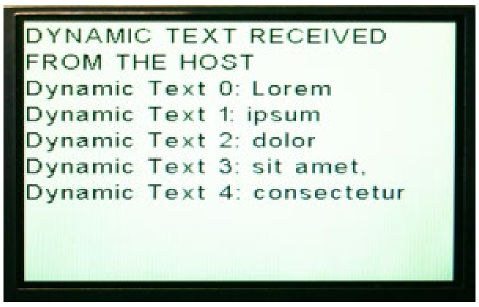
An example of displaying images within and HTML page using border and transparency

text2.htm

An example of a complete screen of text

dyntext.htm

An HTML file which calls the main application for dynamic text to be inserted into the page at run time.



```
DYNAMIC TEXT RECEIVED
FROM THE HOST
Dynamic Text 0: Lorem
Dynamic Text 1: ipsum
Dynamic Text 2: dolor
Dynamic Text 3: sit amet,
Dynamic Text 4: consectetur
```

Also see the sample project for examples of display functions called directly from code.

Using The Driver In Your Project

Updating The Display

Clear The Screen

```
      display_clear_screen(DISPLAY_COLOUR_WHITE);
```

Displaying Lines And Rectangles

```
      //Draw a 1 pixel high line
      display_rectangle(DISPLAY_COLOUR_WHITE, 0, 0, 10, 319, 10);
      //color, thickness(0=fill), x_start_coord, y_start_coord, x_end_coord, 
y_end_coord

      //Draw a filled retangle
      display_rectangle(DISPLAY_COLOUR_WHITE, 0, 20, 40, 120, 80);
      //color, thickness(0=fill), x_start_coord, y_start_coord, x_end_coord, 
y_end_coord

      //Draw a retangle with a 2 pixel wide line
      display_rectangle(DISPLAY_COLOUR_WHITE, 2, 20, 110, 120, 150);
      //color, thickness(0=fill), x_start_coord, y_start_coord, x_end_coord, 
y_end_coord
```

Displaying Bitmaps

The display_bitmap function is used to display .bmp files.

The X and Y coord values specify the top left position of the image, with the top left corner of the screen being coordinate 0, 0

transparency_colour is a colour within your image that should be treated as transparent (not written to the screen). Send as 0xFFFFFFFF to not use this function.

p_bitmap is the pointer to the start of the bitmap file data in the display files converter outputted header file. If your bitmap is stored elsewhere set p_bitmap to zero and make the file ready to read from the first byte by the DISPLAY_BITMAP_READ_NEXT_BYTE define before calling display_bitmap(). The included display_find_file() function allows a bitmap to be found by variable name in the display files converter outputted header file at run time.

Bitmap converted with the Display Files Converter Application who's name is known at design time

```
      display_bitmap(indeximg_bmp, 160, 10, 0x00ffffff);          //p_bitmap, 
x_coord, y_coord, transparency_colour(0xffffffff to not use)
```

Bitmap converted with the Display Files Converter Application who's name is known at run time

```
      filename[0] = 'M';                                          //(Not case 
sensitive)
      filename[1] = 'Y';
      filename[2] = 'F';
      filename[3] = 'I';
      filename[4] = 'L';
      filename[5] = 'E';
      filename[6] = '.';
      filename[7] = 'B';
      filename[8] = 'M';
      filename[9] = 'P';
      filename[10] = 0x00;
      if (display_find_file(filename))
```

```
            display_bitmap(0, 0, 0, 0x00ffffff);      //p_bitmap=null, x_coord,
y_coord, transparency_colour(0xffffffff to not use)
```

.bmp bitmap file stored using your own memory access method (for instance using our SD Card driver)

```
      if (ffs_fopen((const char*)filename, (const char*)"r"))
      //Example function call (not part of this driver)
            display_bitmap(0, 0, 0, 0x00ffffff);
      //p_bitmap=null, x_coord, y_coord, transparency_colour(0xffffffff to not
use)
```

Displaying Text

Use the display_const_string and display_variable_string functions to display null terminated strings. Individual strings may be displayed with left, centre or right alignment and optionally contained within a box with padding and automatic multi line display if necessary.

The X and Y start coordinate values specify the top left position of where to place the text, with the top left corner of the screen being coordinate 0, 0.

If an end x coordinate is specified:

> The start and end values create a virtual box for the text to be displayed within. If a y end coordinate is also specified this creates the bottom of the virtual box or if not the virtual box is made as high as required to display the text on one or more lines, with the bottom of the screen being the limit if the text is too long to fit on the screen.

If an end x coordinate is not specified:

> The x start coordinate specifies the left, centre or right insertion point for the text depending on the alignment option specified. A virtual box is created based on the text width, using the screen edge as a limit if the text is too long to fit on a single line (i.e. the virtual box automatically becomes as wide as needed to hold the text). If the text is too long to fit on a single line a new virtual box is created for each subsequent line. If a y coordinate is specified this limits the bottom of the text, with the bottom of the screen being the limit if the text is too long to fit on the screen.

When the string will be displayed on multiple lines line breaks are automatically created at space characters.

The horizontal and vertical padding values may be set to a non zero value to cause the text to be contained within a smaller virtual inner box. When using background colors this creates a blank area of background around the text, or where the text reaches the screen edge it ensures the specified number of pixels are not used (effectively moving the screen edge in by the number of pixels specified).

Before calling set display_foreground_colour with the required text colour and set display_background_colour with the required background colour (set to 0xFFFFFFFF for no background colour (leave existing colour).

Constant Text

```
const uint8_t string1[] = {"Hello World!"};

      display_foreground_colour = DISPLAY_COLOUR_VIOLET;
      display_background_colour = DISPLAY_COLOUR_NULL;
                  //(DISPLAY_COLOUR_NULL == no background colour / leave
existing colour)
      display_const_string(disp_font_22h, DISPLAY_TEXT_ALIGN_CENTRE,
            //Font, Options
                                                2, 2,
                                          //Hoz padding, vertical padding
                                                20, 105,
                                          //X start coord, Y start coord
                                                200, 160,
                                          //X end coord, Y end coord (0 if area
containment not required)
                                                string1);
                                          //Text
```

Variable Text

```
      uint8_t my_string[10];

      display_foreground_colour = DISPLAY_COLOUR_VIOLET;
      display_background_colour = DISPLAY_COLOUR_NULL;
                  //(DISPLAY_COLOUR_NULL == no background colour / leave
existing colour)
      my_string[0] = 'H';
      my_string[1] = 'e';
      my_string[2] = 'l';
      my_string[3] = 'l';
      my_string[4] = 'o';
      my_string[5] = 0x00;
      display_variable_string(disp_font_15h, DISPLAY_TEXT_ALIGN_CENTRE,
      //Font, Options
                                                0, 0,
                                    //Horizontal padding, vertical padding
                                                20, 30,
                                          //X start coord, Y start coord
                                                0, 0,
                                    //X end coord, Y end coord
                                                &my_string[0]);
                                    //Text
```

Available options

```
      DISPLAY_TEXT_ALIGN_LEFT
      DISPLAY_TEXT_ALIGN_CENTER or DISPLAY_TEXT_ALIGN_CENTRE
      DISPLAY_TEXT_ALIGN_RIGHT
```

Strings Containing Unicode Characters

The driver supports the standard UTF-8 format for representing Unicode characters as two or three bytes:

2 bytes (11 bit Unicode value)

110xxxxx 10xxxxxx

3 bytes (16 bit Unicode value)

1110xxxx 10xxxxxx 10xxxxxx

It also supports the following standard ASCII notation:

&#N;

where N is either a decimal number representing the Unicode character value, or a hexadecimal number in which case it must be prefixed by ‘x’. For instance these both represent the same Unicode character:

&#8482;

&#x2122;

If you wish to display the actual character sequence “&#” within a string then you must enter one of the characters using this Unicode notation or the character sequence will be interpreted as the start of a Unicode character.

This Ishida Apps provides a very handy Unicode conversion tool.

Displaying HTML Pages

The display_html_setup_read_file function is called first to setup ready to parse the HTML file, or if using a separate file manager you need to load the file ready to read from the first byte using the DISPLAY_HTML_READ_NEXT_BYTE define, and then call the display_html_file function to display the file.

Loading Global Styles At Startup (or at any time)

```
	//LOAD OUR GLOBAL HTML STYLES FILE READY FOR DISPLAY HTML PAGES
	uint8_t dummy_styles_count;
	uint32_t file_size;
	if (display_html_setup_read_file(global_css, 0, &file_size))
	{
		dummy_styles_count = 0;
		display_html_read_styles(&file_size, &dummy_styles_count, 1);
		//1 = this is global styles file
	}
```

HTML file converted with the Display Files Converter Application who's name is known at design time

```
	uint32_t file_size;

	if (display_html_setup_read_file(index_htm, 0, &file_size))
	//Find HTML file ready to display it
		display_html_file(file_size);
```

HTML file converted with the Display Files Converter Application who's name is known at run time

```
	uint8_t display_html_file_name[10];
	uint32_t file_size;

	display_html_file_name[0] = 'T';
	display_html_file_name[1] = 'E';
	display_html_file_name[2] = 'S';
	display_html_file_name[3] = 'T';
	display_html_file_name[4] = '1';
	display_html_file_name[5] = '.';
	display_html_file_name[6] = 'H';
	display_html_file_name[7] = 'T';
	display_html_file_name[8] = 'M';
	display_html_file_name[9] = 0x00;

	if (display_html_setup_read_file(0, display_html_file_name, &file_size))
		display_html_file(file_size);
```

.htm HTML file stored using your own memory access method (for instance using our SD Card driver)

```
	if (ffs_fopen((const char*)filename, (const char*)"r"))
	//Example function call (not part of this driver)
		display_html_file(file_size);
```

HOW TO CREATE GRAPHICAL ELEMENTS

Creating Buttons

For simple buttons you can use the display_rectangle function and then the display_const_string function within it. However for great looking buttons nothing beats creating them as a bitmap image and using the display_bitmap function.

Creating User Interface Menu Bars

Create bitmap images of each of the menu elements. Display these and then use text strings to overwrite on top of each menu item. The display_foreground_colour and display_background_colour variables allow colours to be inverted to indicate the selected item.

Creating Windows With Variable Content

Create a bitmap image of the whole window and then overwrite it with text strings and small bitmap images in the required places to create the variable content inside.

Creating Flashing Or Animated Icons

Create two or more bitmap images of the same size and then display each one over the top of the last using a timer

Creating Progress Bars

Create a bitmap of the empty progress bar and then use a single bitmap of one row of pixels to update the progress bar for each successive step, moving it on one column or row at a time.

Creating Lines

Use the display_rectangle funciton.

Creating Graphs

Create a bitmap for your graph X and Y axis markers and then use a single pixel bitmap to create each of the points of a graph line.

Using Driver With A New Screen Model

LCD Controller IC's

If you are using an LCD display and your lucky you will be able to select a screen with a built in controller IC. In this case the screen datasheet / manufacturer should be able to advise you of the required electrical connections and software initialisation steps. The majority of color LCD displays do not incorporate a built in driver IC and you therefore need to select a suitable IC yourself, design the electrical connections and software initialisation steps. The screen manufacturer may provide a sample reference design for this, or you may be able to find a third party processor evaluation board which uses the display and who's design you can copy, but if not don't despair. LCD datasheets are often pretty daunting and poorly translated, but the basic interface is usually much simpler than you might first assume. What you typically have is:

A parallel bus for loading Red, Green and Blue pixel values

A per pixel clock, which latches in the data present on the parallel bus for each pixel in turn

A horizontal clock, which often marks the start of a new row

A vertical clock, which often marks the start of the first column

There may be other signals but these are typically the basic requirement, with other signals often being optional to use. Even the horizontal or vertical clock is often optional as to refresh a screen you actually only need a clock per pixel and a marker for the start of a new screen refresh. All the LCD controller does is continually output the current brightness level to each pixel in turn from its internal RAM buffer which you / this driver writes with the required pixel values. The screen may have pixels before and after each row that are dummy pixels (need to be output but don't actually do anything) and the LCD controller is simply setup at initialisation with these parameters. So to get a new screen working whilst on first glance it might appear a nightmare, it's actually often very straightforward:

You select a LCD controller with enough RAM to store the output data for your screen. E.g.

For a 320 x 240 RGB colour display this is 320 x 240 = 76800 bytes, x 2 to allow 16 bit colour control per pixel = 153600 bytes. (Note that the number of color bits per pixel does not need to relate exactly to the number of color data bits per color x 3 passed to your screen – 16 bit colour is typical, with 24 bit possible but not essential if your screen has 8 bit control per pixel color).

For a 320 x 240 grayscale display this is 320 x 240 = 76800 bytes, x 1 to allow 8 bit control per pixel = 76800 bytes.

For a 120 x 64 monochrome display this is 120 x 64 = 7680 bytes, / 8 for 1 bit control per pixel = 960 bytes.

Ensure the LCD controller output signals are compatible with your display. Most good LCD controllers have lots of output options to allow them to work with many screen interface variants. See if the controller IC has application notes as well as a datasheet, as application notes often provide much better details of all the options.

Design your PCB and include test pads so you can probe the important signals on your prototype (if your using small SMD IC's, connectors etc).

Once your hardware is built you adjust one of our sample screen driver files for your particular hardware and use a scope to verify that the LCD controller is outputting the signals you need. Once they are right (often only nearly right) you'll see results on the screen itself and it's then just a case of tweaking setup values to make the display driver perfect.

That's it, your now ready to use the full power of the Universal Graphic Display Driver. It wasn't as hard as you thought after all was it!

Our driver comes with sample screen driver files based around the use of the Solomon Systech SSD1926 LCD controller IC which is incredibly versatile and suitable for a wide variety of LCD displays.

LCD Display Selection Tips

Color LCD display models are usually designed specifically for a particular customers high volume product. This might seem surprising but this is very often the case and this creates an issue in that if the high volume customer stops purchasing the screen can suddenly become discontinued without any warning. The best way to select a color LCD screen for a new long term product is to speak directly to a LCD screen distributor and ask them to suggest models they believe are likely to have several years production life. This may mean paying more for a screen than the cheapest model that will do the job, but bear in mind that the very cheap screens may be be cheap because the manufacturer wants to clear their stock. Distributors often don't know what a manufacturers intentions are, but they should have good experience with particular manufacturers, which screens tend to be purchased in sufficient volume by customers who are likely to still want them in a few years time and if there are any suitable screens which the manufacturer is providing a production life guarantee on.

Some screens, particularly OLED screens, are supplied with a type of flexible tail which needs to be soldered to the users PCB using a special high volume soldering tool. A low volume solution to this is often to use a FFC connector and get hold of some stiffener sheet to fix to the back of the screen flexible tail, to make it suitable for the FFC connector.

Beware of screen mechanical drawings – just because a data sheet shows a screen in a particular orientation doesn't mean that this is the correct orientation to use the screen! If the datasheet doesn't specify the correct viewing angle the best way to check is to get the screen working and see which orientation provides the best viewing angle for the intended application. If you can't do this ask the manufacturer to confirm the correct orientation or find a photo of the screen in use. The bitmap converter application allows you to rotate your bitmap images and fonts into all 4 orientations, allowing you to use a screen any way round.

Configuring The Solomon Systech SSD1926 For A New Screen Model

This driver is not based on requiring the SSD1926 but we selected it for our sample projects because it is a very good low cost versatile LCD screen controller which can be configured for a variety of LCD screens. It has 256kB of display RAM, supports many display types and 1, 2, 4, 8, 16 or 32 bits per pixel (bpp) color depth. It is also a very popular LCD driver which is sold in high volumes and is therefore low cost. Even though this driver doesn't make use of all of it's powerful feature set it's still often lower cost than simpler display controllers because of it's high sales volume. Note that this driver does not make use of the SSD1926 JPEG decoder (although you could add this functionality if you wanted to).

Step 1 – Copy a pair of display-model.c and .h driver files using the SSD1926 to use for your new screen.

Step 2 – Make sure the interface to the SSD1926 good by running the driver without your screen connected as is and verify that the LCD_SHIFT pin outputs a clock signal after the display_model_initialise() function is called.

Step 3 – Set the 'Panel Type Register' register to suit your display (see the 'SSD1926 Application Note' for details of the available options).

Step 4 – Set the 'Pixel Output Clock' to suit your screens recommended 'DOTCLK frequency' or whatever similar name your screen specifies it as. This is the fast per pixel clock signal and will typically be several MHz. Verify the clock frequency is correct by using a scope on the LCD_SHIFT output pin.

Using A Different Display Controller

Using the driver with a new screen model requires adjusting each of the functions in the display-model.c file to suit your screen and its driver (these files contain just the functions that are screen specific and don't contain any of the general display processing driver functions). Unfortunately due to the massive amount of screens and screen controller IC's available there is no generic way of doing this, but following this guide should make the process straightforward.

Copy A Pair Of Sample Files

First copy the sample display-model.c and display-model.h pair of files for the included screen types that is most similar to your screen to use as a reference. These can then be modified as necessary for your screen and its controller IC.

Screen Specific Defines

Go through each of the defines in the display-model.h file and update as required for your screen.

DISPLAY_WIDTH_PIXELS

The width of the screen in pixels (X coordinate)

DISPLAY_HEIGHT_PIXELS

The height of the screen in pixels (Y coordinate)

IO Defines

Alter the IO defines to provide all of the input and output pins required by your display controller / screen.

Screen Specific Functions

```
void display_model_initialise(void)
```

Adjust this to provide the initialisation sequence required by your display controller IC / screen. This can sometimes be found in the screens data sheet or by contacting the screen manufacturer to request the initialisation sequence required. Display controller IC's, either designed in by you or built into the screen, are often quite generic and can therefore be confusing to know exactly the configuration that is required for the particular screen model they have been used in. For screens with a built in controller IC It is reasonable to expect the screen manufacturer to provide the required initialisation sequence if they have failed to include it in a screen datasheet.

```
void display_delay_ms (uint16_t delay_ms)
```

May be needed if your initialisation sequence needs to incorporate waits before moving on.

```
void display_write_block (uint16_t x_start_coord, uint16_t y_start_coord,
uint16_t x_end_coord, uint16_t y_end_coord, uint32_t colour)
```

This function simply writes individual pixels. If your LCD controller has built in block drawing capabilities you could update this function to use them for improved speed, or just use the basic function as is.

```
void display_write_pixel (uint16_t x_coord, uint16_t y_coord, uint32_t color)
```

This is a key function which writes a single pixel at the specified address. Typically this will involve writing to a ram location in the display controller IC. This function needs to convert the supplied x_coord and y_coord values so that the values 0,0 are the top left of the screen.

For monochrome screens, or for greyscale screens that use < 8 bits per pixel this function will need to read the current byte value which contains the addressed pixel and then write the modified byte value back. For monochrome screens it can often be useful to use a local ram buffer in your microcontroller as a copy of the pixels output, so that you can read and update it then write the byte out to the screen / display contorller IC and therefore avoid needing to read from it (e.g. for fastest speed or for screens that only allow writing).

This driver always uses a 24bit color value. However your screen may not and it is this function that you convert the null|red|green|blue colour value to suit your screen.

For a monochrome screen this is typically setting the pixel on for any > 0 value.

For a greyscale screen this is typically setting the pixel value by selecting the highest value from the RGB bytes.

For a color screen this is converting the three 8 bit values to suite the color depth of you screen.

```
void display_set_register (uint16_t address, uint8_t value)
```

This function writes values to a specific display controller IC register. It is designed to work specifically for the display controller IC in use.

Setting Contrast

If your screen includes digital contrast control then create a function to set this or include it in the display_model_initialise() function.

```
display_output_buffer()
```

Does your display-model.c file implement the function display_output_buffer() (see if its mentioned in display-model.h). If so, remember you need to call it to update the display with new changes.

Getting A New Screen To Work

Some screens, such as monochrome and greyscale LCD screens, have specific voltage requirements. If you are lucky your screen will use a controller IC that includes generation of these voltages. You will still need to configure the driver to provide the voltages required by the screen, but you will not need to provide the voltages externally. If not then you will need to provide the required voltage(s). For LCD screens there are many IC's available designed to provide bias voltages, often with a digital interface to provide the contrast adjustment (variation of the bias voltage). Otherwise a simple potentiometer may be used. Be aware that sometimes the bias voltage needs to be negative, so you may have to use a DCDC converter if your application has no negative voltage rail.

A controller IC provides the actual drive of the display pixels and a standard data bus connection to your processor / microcontroller. Some controller IC's provide advanced features such as more than 1 'layer' of display output. This driver uses a single layer so any other layers can be ignored (or this driver can be adapted to output to different layers if you require layering functionality)

For screens with a built in display controller:

Screen data sheets don't always provide very good information to help you get a new screen working. Some datasheets simply provide the mechanical details of the screen, the electrical characteristics and specify which display controller IC is used in the screen. Here's some tips:-

Get the datasheet for the display controller IC being used. Also check to see if there is a technical manual as some IC datasheets provide little helpful information on how to get a screen working because this is provided in a separate technical manual or application note.

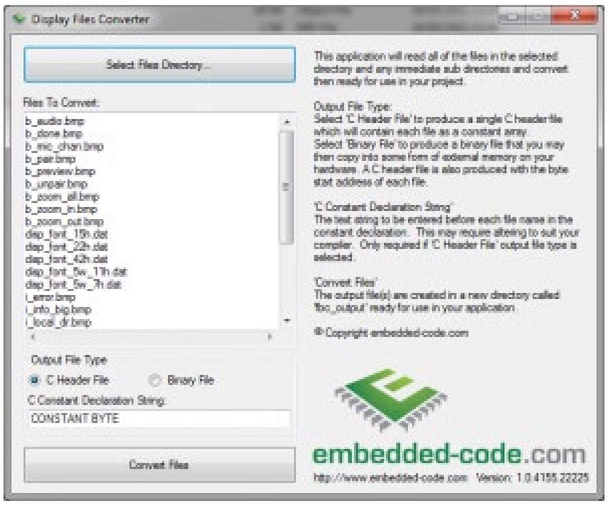
Your screen may use 2 off the same display driver IC. This makes the screen access functions slightly more complicated as each IC has to be individually addressed but it is simply a case of sending the initialisation sequence to both IC's one after the other and selecting which IC to write to when outputting pixels.

Tips on configuring the driver for your screen

If your screen has built in contrast control a good way to check that your initialisation sequence is working is to use a routine to cycle through the contrast values. If all is well you should see the contrast change on the screen regardless of what the screen might be displaying. This confirms that you are at least communicating correctly with the screen. If there is no contrast command maybe your screen driver has special commands to say turn all pixels on, again providing a means to confirm that you are communicating correctly.

DISPLAY FILES CONVERTER APPLICATION

OVERVIEW



Display Files Converter

Select Files Directory...

Files To Convert:

b_audio.bmp
b_done.bmp
b_mic_chan.bmp
b_pair.bmp
b_preview.bmp
b_unpair.bmp
b_zoom_all.bmp
b_zoom_in.bmp
b_zoom_out.bmp
disp_font_15h.dat
disp_font_22h.dat
disp_font_42h.dat
disp_font_5w_11h.dat
disp_font_5w_7h.dat
i_error.bmp
i_info_big.bmp
i_local_dir.bmp

Output File Type

(●) C Header File ( ) Binary File

C Constant Declaration String:

CONSTANT BYTE

Convert Files

This application will read all of the files in the selected directory and any immediate sub directories and convert them ready for use in your project.

Output File Type:
Select 'C Header File' to produce a single C header file which will contain each file as a constant array.
Select 'Binary File' to produce a binary file that you may then copy into some form of external memory on your hardware. A C header file is also produced with the byte start address of each file.

'C Constant Declaration String'
The text string to be entered before each file name in the constant declaration. This may require altering to suit your compiler. Only required if 'C Header File' output file type is selected.

'Convert Files'
The output file(s) are created in a new directory called 'fbc_output' ready for use in your application.

© Copyright embedded-code.com

embedded-code.com

http://www.embedded-code.com Version: 1.0.4155.22225

If you are using some form of filing system in your application and wish to display source files directly from it this section may not apply as you will deal with storing and retrieving the files yourself. However even if bitmaps and HTML pages are stored in a filing system it is often useful for speed to store fonts in program memory and therefore in this case the Display Files Converter application is still used for this.

Separate file access defines are used by this driver so that font's, bitmap files and HTML files may each be stored using Display Files Converter application in internal or external memory or by some other method (i.e. your own filing system).

The Display Files Converter application does not convert the data format of fonts, bitmaps or HTML files – the driver reads and displays these files as originally created. Instead the Display Files Converter application converts the files so that they are included byte by byte in program memory by your compiler or so they are packed into a single binary data file for you to store in an external flash memory IC. Therefore all it is doing is packing all of the files together and creating a header file which allows this driver to find each individual file within the overall packed data.

USING THE DISPLAY FILES CONVERTER

These instructions assume you will store all of your files in program memory, which is a good place to start for a new project.

Create a new directory in your projects main directory called:

display_files\

Into this copy:

The font (.dat) output files you wish to use.

The bitmap (.bmp) image files you wish to display.

The html and css files you wish to use.

Run the Display Files Converter PC application and select this directory. As You will incorporate the files into your microcontrollers program memory select the 'C Header File' option.

The 'C Constant Declaration String' should typically be 'const uint8_t'.

Forcing program memory usage inXC32:

The following can be used: const unsigned char __attribute__((space(prog)))

However one issue comes up when building, XC32 will error due to the following lines in display_files_c.h

```
const unsigned char __attribute__((space(prog))) *display_filenames[];
```

Move the pointer * so its like this to resolve:

```
const unsigned char* __attribute__((space(prog))) display_filenames[];
```

Press the 'Convert Files' button and the application will generate the output header file, called '\output\display_files_c.h'. As supplied the driver includes and accesses this file automatically.

If you If you will store the files data in external flash memory and provide functions to access it select the 'Binary File' option.

Press the 'Convert Files' button and the application will generate two output files – .

A file called '\output\display_files_bin.h' which the driver includes and accesses automatically to determine each files address.

A file called '\output\display_files_bin.bin' which you need to load into your flash memory device and provide a function for the driver to read bytes (see the display.h header file for details).

IMAGES

IMAGE FILES

Supported Files

The display driver works with .bmp files, which are uncompressed so fast to display and allow the full colour pallet to be used.

The driver supports uncompressed “Windows” formatted .BMP files of the following types:

1 bit Monochrome bitmap

4 bit 16 colour bitmap

8 bit 256 colour bitmap (usually the best setting to use for non photo images due to the smaller file size)

16 bit (high colour) bitmap (many display drivers only provide 16 bits of colour resolution making using files with a larger bit value pointless).

24 bit (true colour) bitmap

32 bit (true colour) bitmap (not recommended as the 4th byte is unused resulting in no additional actual bitmap data – use 24 bit instead to reduce file size).

The bitmap size must not be larger than the screen resolution.

Note that files must be saved as uncompressed. The sample files we’re created using Adobe Photoshop and Illustrator.

Transparency

BMP files don’t support transparency but the driver allows for transparency by specifying a colour to be treated as transparent in the images alt tag within HTML files or by a parameter when calling the display_bitmap() function. When pixels of this color are to be displayed they are instead skipped leaving the existing pixel untouched.

Images In HTML Files

The display driver works with .bmp files, which are uncompressed so fast to display and allow the full colour pallet to be used. However some HTML editors, such as Dreamweaver, won’t display .bmp files when designing a page. To get round this you can either preview the page in a browser on the PC (e.g. using F12 in Dreamweaver) or alternatively create a copy of each image file using the .png format. PNG images will display in Dreamweaver and the driver will automatically convert any filename ending in .png to be .bmp when displaying to avoid any changes needing to be made to the HTML when the file is used by the driver. For instance both of these image tags will cause the image myimage.bmp to be displayed:

```
<img src=”myimage.bmp”/>

<img src=”myimage.png”/>
```

(The Display Files Converter application will automatically skip and .png, .ai and .psd files when converting).

FONTS

BIT FONT CREATOR PRO

This driver is designed to support Unicode fonts created using the Bit Font Creator Pro application from Iseatech. For many applications the font files included with the driver will provide the all of the extended ASCII characters requried, but if you wish to edit these, add new Unicode characters or create new fonts you will need to purchase Bit Font Creator Pro (it is not included as part of this driver). Please see their web site for details and note that is is the Unicode 'Pro' version 3 or above that is requried.

EDIT EXISTING FONT

Open the font .foc file with Bit Font Creator Pro (V3 and above)

Edit the file as required. To add or remove characters use the 'Edit Table' button. Ensure the font remains set to use 'Unicode' encoding.

Export Font

Press 'Step 4 Configure Data Format'

Bit Order = Big Endian

Scan Based = Row

Scan Preferred = Row

Data Packed = Yes

Data Length = 8 bits

Press 'Step 5 Export Bitmap Data' button

Data format = Binary File (*.dat)

Save the file using the name you will refer to the font as in your code.

Copy the outputted .dat font file to your projects display files folder and use the Display Files Converter Application to create a new output file which will include the new font.

*For BitFontCreator Pro V3.2 and above select "Old BitFontCreator Format .bin" and rename the outputted *.bin file to be *.dat.*

CREATE A FONT FROM SCRATCH

Font Tips

Good sites for free fonts:

http://www.dafont.com

When trying out fonts in BitFontCreator pro, try with several font size settings, as some fonts will suddenly perfectly work at the right setting.

You can copy in a Photoshop BMP file and paste into a character in BitFontCreator Pro.

Create The Basic Font

Open Bit FontCreatorPro (V3 or above)

File > New Font > Import An Existing Font

> Select the font and style. The ‘size’ you select will determine the pixels height / width size of the font. Important note if creating fonts that will support specific character sets (e.g. Japanese, Chinese etc):
>
> > In Windows if you select a font to display characters it does not support, Windows will use another font to display those characters. That is to say, if you use Arial to display Japanese characters, Windows will actually use another font to display them because Arial doesn’t include Japanese characters. Different systems will choose different fonts, based what the fonts you have installed in Windows. BitFontCreator needs to use source fonts that include all the characters you will use as otherwise it can produce bad characters when Windows substitutes fonts in the background (e.g. it may add the characters but with the width clipped). To avoid this use a font which includes all of the characters. For instance ‘Lucida Sans Unicode’ provides many character sets. You can use the Windows “Character Map” in Windows Programs > Accessories > System Tools to see what characters a particular font supports.
>
> Keep re-importing the font until you get the character size you want, by looking at the creation results of specific max and min characters (e.g. ‘i’, ‘l’, ‘W’, ‘w’, accented characters etc).

Select The Characters To Include

Press ‘Step 2 Edit Characters Table’ button

> Ensure the encoding is set to Unicode
>
> The font you have selected is likely to provide characters in many character sets. Select the character set you want to use.
>
> A great quick method to only include the characters your application will use is to save all of those characters in a .txt file. It doesn’t matter if the characters repeat in the file and .txt files may be saved in Unicode format. Use the ‘Load Table’ button to open the file and BitFontCreator will automatically enable each character in the file and disable all other characters.

Save the file

Edit The Characters

Press the ‘Remove Blank Columns’ button to remove whitespace to the left and right of each character.

Select each character in turn and check it / make any edits required.

Create any special symbol characters you want using the drawing tools.

Now go through all the characters and see how many unused pixels there are top and bottom. Notes:

> Typically a row or 2 at the bottom can be removed as the few characters that use it can be moved up.
>
> The grey line shows the fonts base line, to help you see under line parts of characters (note this is a guide only and isn’t used when the driver displays fonts).

Press ‘Step 3 Change Font Height’ button

> This step may be necessary as when discarding characters we don’t want (Unicode, European etc) there may now be unused rows that we can get rid of. (Take a look at a capital letter with a top accent and a lowercase ‘y’ to quickly see if there is likely to be an unused row). Save before trying anything risky!

This command literally clips or adds. Once you use clip those pixels are gone – if you add later on the added rows will be blank (i.e. if you over clip you can't undo by using add, so get it right first time, or do successive clips.

Export Font

Press 'Step 4 Configure Data Format'

Bit Order = Big Endian

Scan Based = Row

Scan Preferred = Row

Data Packed = Yes

Data Length = 8 bits

Press 'Step 5 Export Bitmap Data' button

Data format = Binary File (*.dat)

Save the file using the name you will refer to the font as in your code,

Copy the outputted .dat font file to your projects display files folder and use the Display Files Converter Application to create a new output file which will include the new font.

*For BitFontCreator Pro V3.2 and above select "Old BitFontCreator Format .bin" and rename the outputted *.bin file to be *.dat.*

HTML FILES

HTML BASICS

Main Features

- Designed to display HTML files created using Adobe Dreamweaver. Display of HTML files created with other applications should not present significant problems (however due to the massive number of applications available we only provide support for files created using Dreamweaver)
- Supports styles, including global styles stored in a separate css file.
- HTML file display may include inline dynamic text, generated by your application at run time.
- Provides image transparency using bmp files and a special tag.

HTML files should use standard UTF-8 encoding, where the ASCII characters are preserved unchanged against ASCII.

The driver parses HTML files in order. Therefore if your design calls for overlaying elements, for instance text on top of images, simply place the text after the image in the HTML file.

Note that the HTML parser is not designed to parse any HTML file or provide high levels of error checking. The HTML display capability is provided to allow display setups to be quickly and easily created, but you need to ensure you follow the guidelines here for trouble free operation.

Please see the included sample HTML files for examples of working uses of the various HTML features.

Designing HTML Pages

The driver does not blank the screen when displaying a new HTML page. This is a deliberate feature which can often be useful as it allows you to show elements over the top of an existing screen if desired, for instance as a pop up box. To blank the screen use a style with a DIV tag surrounding the contents of your page, set to the size of your screen and with the desired background colour. For example:

```
.background {

background-color: #FFFFFF;

height: 240px;

width: 320px;

}
```

The driver ignores generic styles such as body. This can be useful to allow the design view to be setup with a colour that will allow the actual screen area to be shown on top of it. For instance the following style definition will set the page to grey:

```
body {

background-color: #999999;

margin: 0px;

padding: 0px;
```

}

Then surrounding your page content with the background style shown above, for instance, you will see the screen area in your design application such as Dreamweaver.

Dreamweaver Specific Usage Notes

To apply a new style and positioning to text and images

Select the text / image

Menu > Insert > Layout Objects > DIV Tag

Press New CSS Style

Selector type = Class

Enter the style name (max 10 characters)

Press OK

Then create any of the permitted CSS properties you want.

Press OK

Leave the class name to match the entered style name and leave the ID blank

This will wrap the selection in a new DIV tag using the class name you used.

The properties of this class can now be edited if needed using the CSS styles box. The same style can be added to other text / images by using Menu > Insert > Layout Objects > DIV Tag and then selecting the existing class name.

HTML Fonts

Fonts are selected using the font-size attribute and are specified in pixels (px). The html_font_sizes and associated defines in display-html.h specify the fonts to be used, with the driver included fonts files providing these pixel sizes:

7, 11, 15, 22, 42

Specifying a different font-size value will cause the nearest font available that is >= to the pixel size specified to be selected, starting from the smallest.

Unicode Characters Within HTML

The driver supports the standard UTF-8 format for representing Unicode characters as two or three bytes:

2 bytes (11 bit Unicode value)

110xxxxx 10xxxxxx

3 bytes (16 bit Unicode value)

1110xxxx 10xxxxxx 10xxxxxx

It also supports the following standard ASCII notation:

&#N;

where N is either a decimal number representing the Unicode character value, or a hexadecimal number in which case it must be prefixed by ‘x’. For instance these both represent the same Unicode character:

```
&#8482;

&#x2122;
```

If you wish to display the actual character sequence “&#” within an HTML page then you must display one of the characters using this Unicode notation or the character sequence will be interpreted as the start of a Unicode character.

This Ishida Apps provides a very handy Unicode conversion tool.

HTML IMAGES

The display driver works with .bmp files, which are uncompressed so fast to display and allow the full colour pallet to be used. However some HTML editors, such as Dreamweaver, won’t display .bmp files when designing a page. To get round this you can either preview the page in a browser on the PC (e.g. using F12 in Dreamweaver) or alternatively create a copy of each image file using the .png format. PNG images will display in Dreamweaver and the driver will automatically convert any filename ending in .png to be .bmp when displaying to avoid any changes needing to be made to the HTML when the file is used by the driver. For instance both of these image tags will cause the image myimage.bmp to be displayed:

```
<img src=”myimage.bmp”/>

<img src=”myimage.png”/>
```

(The Display Files Converter application will automatically skip and .png, .ai and .psd files when converting).

BMP files don’t support transparency but the driver allows for transparency by specifying a colour to be treated as transparent in the images alt tag. For instance the following image will be shown with any pixels that are the color 0xFEFEFE not displayed:

```
<img src=”myimage.bmp” alt=”#FEFEFE”/>
```

Width and height attributes are ignored by the driver (image scaling is not provided).

HTML STYLES

The display_html_read_styles() function may be used to load a global style sheet file if desired. Any styles it contains are added to the drivers internal style stack.

When a new HTML page is loaded the driver will also load any styles it contains in its header section temporarily for that page.

The maximum number of styles the driver will store is set using the MAX_STYLES_TO_STORE define.

The driver ignores standard styles such as body, p etc. All styling of your HTML pages must be carried out using DIV tags containing style parameters, for example:

```
<div style=”position:absolute; top:60px; left:20px; height:30px; width:80px; background-
color:#CCFF00;”> </DIV>
```

or DIV tags referencing a class that is defined as a style, for example:

```
<div class=”astylename”> </div>
```

This approach is really flexible and with DIV tags being the standard method of laying out HTML pages in applications such as Dreamweaver this makes creating and editing pages using DIV tags very simple. Note that styles should only be defined once (i.e. all parameters need to be contained within a single style definition).

The following style attributes are accepted by the driver (others may be used but will be ignored) top ‘px’

Pixel values only accepted. Position is assumed to be absolute.

left ‘px’

Pixel values only accepted. Position is assumed to be absolute.

height ‘px’

Pixel values only accepted. Position is assumed to be absolute.

width ‘px’

Pixel values only accepted. Position is assumed to be absolute.

background-color

‘#xxxxxx’ values only accepted where xxxxxx is the hex value.

color

‘#xxxxxx’ values only accepted where xxxxxx is the hex value.

border

‘px’ and ‘#xxxxxx’ values only accepted.

border-#-width

‘px’ pixel values only accepted.
# is top, left, right or bottom.

border-#-color

‘#xxxxxx’ values only accepted.
# is top, left, right or bottom.

text-align

left, center or right.

font-size

‘px’ pixel values only accepted. This alone determines the font that will be used from the available fonts (other font attributes are ignored).

padding or padding-#

‘px’ pixel values only accepted.
# is top, left, right or bottom. Note that only vertical and horizontal padding attributes are available from the driver. Therefore when using different values the left/ right and top/bottom values are assumed to be the same.

Common tags and styles not supported that may catch you out:

margin

The driver is based on absolute positioning and therefore doesn’t process margins.

span

The driver does not allow formatting of text within lines of text. Instead of selecting text and applying a style, surround the text with a DIV tag and alter it’s style.

The included sample HTML pages demonstrate usage of the tags.

DYNAMIC TEXT

The driver allows for dynamic text to be automatically added inside HTML content. This can be displayed text or HTML code. The dynamic text referred to is simply added to the HTML immediately following the comment and prior to the rest of the HTML document being read. The format for dynamic text is:

<!–#00–>

where “00”’ is a value from 00 to 99 (always formatted as 2 digits), referencing dynamic text indexes from 0 to 99. The dynamic text may be inserted by your application using the HTML_GET_DYNAMIC_TEXT_FUNCTION define (see display-html.h for details).

TOUCHSCREEN LINKS (ANCHORS)

Your HTML files may contain standard anchor links which can optionally be passed to your application using the HTML_PASS_ANCHOR_LINK_FUNCTION define (see display-html.h for details). HTML file names may be used, allowing you to test operation as the user interface is designed in your browser, or the href=”” may contain any text value for your application to process. It is up to your application to provide touchscreen sensing if you are using one, with this driver functionality providing the screen coordinates of each anchor link area of the screen together with the href”” string for your application to store as the HTML file is displayed and then act upon as it senses touch events. In it’s simplest form you application could simply store the list of passed href filename strings and associated coordinates and when a touch is sensed within the coordinates area trigger the driver to load the corresponding HTML file.

Example:

```
<a href="testlink.htm">
  <div ...
  ... </div>
</a>
```

INFORMATION

FREQUENTLY ASKED QUESTIONS

Where are the bitmaps,HTML files and fonts stored?

The included Display Files Converter PC Application allows you to convert all of your bitmap graphic, HTML files and fonts into a C compliant header file, so that the data is stored in your devices program memory, or as a binary file so that you can store them in external memory.

Is The Driver Difficult To Use?

Setting a new screen and driver IC up for the first time can be difficult. Due to the huge number of screens and driver IC's available it would be impossible for us to provide individual drivers for every screen and driver combination. Therefore we have made this driver as generic as possible and you will need to alter the display-model.c and display-model.h files to work with your particular screen. See earlier in this manual for details of how to do this. Once this is done you then only need to select the orientation you want to use the screen in and the driver will then be working correctly for your screen. That's the hard bit out of the way – with your screen setup correctly using the driver is a breeze and the flexibility of being able to put what you want wherever you want on the screen makes designing user interfaces really enjoyable.

Do I Need A Fast Microcontroller?

This depends on what you are displaying. For instance, an 8 bit microcontroller running with a 10MHz instruction clock is typically fine for a monochrome and many grayscale screens in lower resolutions (e.g. 64 x 128 pixels). For a colour 320 x 240 pixel screen an 8 bit microcontroller isn't going to cut it and a 16 or 32bit microcontroller is going to be needed operating at a reasonably fast speed (e.g. a PIC32 @ 80MIPS is fine for a 320×240 RGB screen with the compiler optimisation set to fastest). The main limit factor is clearing an entire screen and writing new content over an entire screen. Each write to each screen pixel takes time and if your microcontroller is too slow the screen refresh becomes very noticeably slow and the user has to wait while the screen updates.

Can I Use Special Features Of My Display Controller IC, Do Special Things?

This driver is designed to be universal and works on the very simple principal of writing a single layer of pixel data to be displayed to the display controller IC's ram buffer. This data is then directly displayed on the screen. Depending on your screen this ram buffer may have 1 bit representing 1 screen pixel (monochrome screens), 1 byte representing 1 screen pixel (typical grayscale screens) or 2 bytes representing a RGB pixel (typical color screens). Other memory uses are also possible, but these are typical uses. All this display driver does is write to these memory locations whenever you call a display function. There are no background processes or function calls so when you are not calling a function this driver is not doing anything. This means that if you want to you can manipulate the memory that the display controller writes to, to achieve things like this (if your display controller IC has suitable resources):

> Output a new display screen to a section of the display controller IC's ram which is not currently displaying in the display_write_pixel() function. Then slide the new screen buffer in by telling the display controller to use the new ram area column by column or row by row.
>
> Use layers by controlling which display controller IC layer is written to in the display_write_pixel() function.
>
> Output to multiple screens by selecting which display controller IC / screen is written to before displaying an HTML page, bitmap, text, etc.

Will the driver work with an RTOS?

Yes. The driver has no background processes so is fully compatible with an RTOS environment. The driver is implemented as a single thread so you just need to make sure it is always called from a single thread (it is not designed to be thread safe).

Do I Need To Use The HTML Functionality
No. All of the HTML functions are contained in dedicated display-html.c and display-html.h files and if you don't need to display HTML files these can simply be omitted from your project.

SPECIFICATIONS

Maximum screen size
5000 x 5000 pixels

Using The Driver With a RTOS or Kernel
The driver has no background processes and is fully compatible with an RTOS environment. The driver is implemented as a single thread so you just need to make sure it is always called from a single thread (it is not designed to be thread safe).

CODE AND DATA MEMORY REQUIREMENTS

Code Size Example
PIC32 Sample Project with 320 x 240 RGB screen, including HTML parsing functions:

> Approximately 39628 program memory bytes (9907 x 32 bit instructions) excluding sample font, bitmap and html data. Compiled using the Microchip C32 MPLAB C Compiler for PIC32 family of 32 bit microcontrollers with all optimisations turned off.

PIC32 Sample Project with 320 x 240 RGB screen, with HTML parsing functions removed:

> Approximately 23620 program memory bytes (5905 x 32 bit instructions) excluding sample font, bitmap and html data. Compiled using the Microchip C32 MPLAB C Compiler for PIC32 family of 32 bit microcontrollers with all optimisations turned off.

Variables Memory Space
The basic driver does not requrie a great deal fixed RAM or temporary variable storage space from the stack, as bitmap data is read and displayed one byte at a time. HTML parsing (if used) requries variable storage space to buffer styles, parsed text etc and various MAX_LENGTH defines allow these requriements to be customised if desired.

How The Driver Works

Note – this section of the manual is for information should you wish to gain an understanding of how each of the driver components works.

Overview

Generic Driver and Screen Specific Files

The ‘display.c’ and ‘display.h’ files contain all of the driver generic functions and defines.

The ‘display-model.c’ and display-model.h’ files contain functions and defines that are particular to a specific screen and controller IC.

The ‘display-html.c’ and display-html.h’ files contain functions and defines that are used to display HTML files.

Initialise The Screen

void display_initialise (void)

This function calls a screen specific function in the ‘display-model.c’ file. The function contains the screen initialisation sequence.

If configuring the driver for a new screen then copy sample ‘display-model.c’ and ‘display-model.h’ files for a screen that is most similar and then modify this function to suit the screen in use.

Test Display

void display_test (void)

This function is a handy way of checking a new screen is setup correctly. It draws a 1 pixel wide border round the edge of the screen and the following color fade bars:

Red
Green
Blue
Turquoise
Violet
Yellow
White

This is a universal function – modification is not normally required.

Clear Screen

```
void display_clear_screen (uint32_t colour)
```

Clears the entire screen using the specified color

This is a universal function – modification is not normally required.

Display Rectangle

```
void display_rectangle (uint32_t color, uint8_t thickness, uint16_t
x_start_coord, uint16_t y_start_coord, uint16_t x_end_coord, uint16_t
y_end_coord)
```

Displays a rectangle contained within the coordinates specified.

thickness

Set to 0 to create a rectangle fill. Set >0 to create a rectangle line of the thickness specified.

This is a universal function – modification is not normally required.

Display Bitmap

```
uint16_t display_bitmap (const uint8_t *p_bitmap, uint16_t x_coord, uint16_t
y_coord, uint32_t transparency_colour)
```

The display driver works with .bmp files, which are uncompressed so fast to display and allow the full colour pallet to be used. The driver supports uncompressed "Windows" formatted .BMP files of the following types:

1 bit Monochrome bitmap
4 bit 16 colour bitmap
8 bit 256 colour bitmap (usually the best setting to use for non photo images due to the smaller file size)
16 bit (high colour) bitmap (many display drivers only provide 16 bits of colour resolution making using files with a larger bit value pointless).
24 bit (true colour) bitmap
32 bit (true colour) bitmap (not recommended as the 4th byte is unused resulting in no additional actual bitmap data – use 24 bit instead to reduce file size).

The bitmap size must not be larger than the screen resolution. Note that files must be saves as uncompressed. The Embedded Code sample files we're created using Adobe Photoshop and Illustrator.

p_bitmap

Pointer to the bitmap file you want to display (or 0x00 if you have setup some file source ready for DISPLAY_BITMAP_READ_NEXT_BYTE to be called)

x_coord

Left starting coord

y_coord

Top starting coord

transparency_colour

Optional colour that should be treated as transparent (not written to the screen). Send as 0xFFFFFFFF to not use.

The function returns the height of the bitmap displayed

After calling:

display_auto_x_coordinate = the pixel after the right of the image

display_auto_y_coordinate = the pixel after the bottom pixel of the image

This is a universal function – modification is not normally required.

Display String

Version for displaying constant strings:-

```
const uint8_t *display_const_string (const uint8_t *p_font, uint16_t options,
uint8_t horizontal_padding, uint8_t vertical_padding, uint16_t x_start_coord,
uint16_t y_start_coord, uint16_t x_end_coord, uint16_t y_end_coord, const
uint8_t *p_string)
```

Version for displaying variable strings:-

```
const uint8_t *display_variable_string (const uint8_t *p_font, uint16_t options,
uint8_t horizontal_padding, uint8_t vertical_padding, uint16_t x_start_coord,
uint16_t y_start_coord, uint16_t x_end_coord, uint16_t y_end_coord, uint8_t
*p_string)
```

Use these functions to display null terminated strings. Individual strings may be displayed with left, centre or right alignment and optionally contained within a box with padding and automatic multi line display if necessary.

The X and Y start coordinate values specify the top left position of where to place the text, with the top left corner of the screen being coordinate 0, 0.

If an end x coordinate is specified:

> The start and end values create a virtual box for the text to be displayed within. If a y end coordinate is also specified this creates the bottom of the virtual box or if not the virtual box is made as high as required to display the text on one or more lines, with the bottom of the screen being the limit if the text is too long to fit on the screen.

If an end x coordinate is not specified:

> The x start coordinate specifies the left, centre or right insertion point for the text depending on the alignment option specified. A virtual box is created based on the text width, using the screen edge as a limit if the text is too long to fit on a single line (i.e. the virtual box automatically becomes as wide as needed to hold the text). If the text is too long to fit on a single line a new virtual box is created for each subsequent line. If a y coordinate is specified this limits the bottom of the text, with the bottom of the screen being the limit if the text is too long to fit on the screen.

When the string will be displayed on multiple lines line breaks are automatically created at space characters.

The horizontal and vertical padding values may be set to a non zero value to cause the text to be contained within a smaller virtual inner box. When using background colors this creates a blank area of background around the text, or where the text reaches the screen edge it ensures the specified number of pixels are not used (effectively moving the screen edge in by the number of pixels specified).

Before calling:

> Set display_foreground_colour with the requried text colour
>
> Set display_background_colour with the requried background colour. Set to 0xffffffff for no background colour (leave existing colour).

*p_font

> Pointer to the font to be used (or 0x00 if you have setup some file source ready for DISPLAY_FONT_READ_NEXT_BYTE to be called)

options

> Text display options (see display.h)

horizontal_padding

Pixels of background requried left and right of text

vertical_padding

> Pixels of background requried above and below text

x_start_coord, y_start_coord

Start coordinate

x_end_coord, y_end_coord

End coordinate (either may be 0 if area containment not required or screen edge to be used as limit)

*p_ascii_string

Pointer to the string to display

Returns

0x00 if entire string was displayed, or the pointer to the start of the next word if the entire string wouldn't fit

After calling:

display_auto_x_coordinate = the pixel after the last pixel of the last character

display_auto_y_coordinate = the pixel below the last line displayed

These are universal functions – modification is not normally required.

Load Font

```
void display_load_font (const uint8_t *p_font)
```

Used by the display_const_string() and display_variable_string() functions to setup with the font to be used to display the string.

This is a universal function – modification is not normally required.

Get Font Character

```
void display_get_font_character (uint16_t character)
```

Used by the display_const_string() and display_variable_string() functions to setup for each character to be displayed, finding its location and width.

This is a universal function – modification is not normally required.

Display Character

```
void display_character (void)
```

Used by the display_const_string() and display_variable_string() functions to display each text character.

This is a universal function – modification is not normally required.

Find File

```
uint8_t display_find_file (uint8_t *search_for_filename)
```

Use this function to search for a bitmap file that has been converted using the Display Files Converter application. This function is a method to search for files whose name is not know at design time (i.e. from a variable string filename, for instance in a HTML file). Files may have been stored using the C Header File or Binary File option of the Display Files Converter application and this function will find them by using the Display Files Converter outputted header file.

search_for_filename

The available files will be searched to see if any match this filename.

Returns

1 if the file was found and DISPLAY_FILE_SET_READ_ADDRESS was used to setup its address (i.e. p_bitmap can be set to 0 when calling display_bitmap()),

0 if not

This is a universal function – modification is not normally required.

Display Delay

```
void display_delay_ms (uint16_t delay_ms)
```

The accuracy of this function is not important as long as calling with a value of 1 means the delay will be at least 1mS – it is only used for the initialise function.

Write Block

```
void display_write_block (uint16_t x_start_coord, uint16_t y_start_coord,
uint16_t x_end_coord, uint16_t y_end_coord, uint32_t colour)
```

Writes a rectangular block of pixels

This function simply writes individual pixels. If your LCD controller has built in block drawing capabilities you could update this function to use them for improved speed.

This is a universal function – modification is only required if you wish to make use of faster special features of your display driver.

If configuring the driver for a new screen then copy sample 'display-model.c' and display-model.h' files for a screen that is most similar and then modify this function to suit the screen / display controller in use.

Write Pixel

```
void display_write_pixel (uint16_t x_coord, uint16_t y_coord, uint32_t colour)
```

Writes a single pixel at the specified address, converting the provided 24bit colour value to the displays / driver IC colour data format.

If configuring the driver for a new screen then copy sample 'display-model.c' and display-model.h' files for a screen that is most similar and then modify this function to suit the screen / display controller in use.

Set Register

```
void display_set_register (uint16_t address, uint8_t value)
```

This function writes values to a specific display controller IC register. It is designed specifically for the display controller IC in use.

If configuring the driver for a new screen then copy sample 'display-model.c' and display-model.h' files for a screen that is most similar and then modify this function to suit the screen / display controller in use.

THE HTML DRIVER FUNCTIONS AND DEFINES

Note – this section of the manual is for information should you wish to gain an understanding of how each of the driver components works.

Generic Driver Files

The 'display-html.c' and display-html.h' files contain functions and defines that are used to display HTML files.

Setup To Read A HTML File

```
uint8_t display_html_setup_read_file (const uint8_t *p_file_pointer, uint8_t
*search_for_filename, uint32_t *file_size)
```

Use this function to setup ready to read for files that have been converted using the Display Files Converter application. Files may have been stored using the C

Header File or Binary File option.

p_file_pointer

Load with the pointer to the file or set to 0 for function to search for file name using search_for_filename

search_for_filename

If p_file_pointer == 0 then the available files will be searched to see if any match this filename. Set to 0 if not required.

Returns

1 if file found and ready to read, 0 if not

This is a universal function – modification is not normally required.

Display HTML File

```
void display_html_file (uint32_t file_size)
```

Call this function after calling display_html_setup_read_file() or after your application has setup accessing your html file ready for the DISPLAY_HTML_READ_NEXT_BYTE define to be used

This function parses the entire HTML file and displays it, displaying all text and bitmaps as it find them.

This is a universal function – modification is not normally required.

DIV styles buffer push & pop

```
void display_html_push_stack_for_new_tag (DISPLAY_HTML_TAGS_STACK *tags_stack,
uint8_t *current_depth, uint8_t *text_buffer_start, uint8_t *p_text_buffer)

void display_html_pop_stack_for_closing_tag (DISPLAY_HTML_TAGS_STACK
*tags_stack, uint8_t *current_depth, uint8_t *text_buffer_start, uint8_t
*p_text_buffer)
```

These are sub functions used by the display_html_file() function to push and pop styles as the html file is parsed.

These are universal functions – modification is not normally required.

Display any pending elements

```
void display_any_pending_elements (DISPLAY_HTML_TAGS_STACK *tags_stack, uint8_t
current_depth, uint8_t *text_buffer_start, uint8_t *p_text_buffer)
```

This is a sub function used by the display_html_file() function to display any pending text or bitmap before it starts a new section of the HTML page.

This is a universal function – modification is not normally required.

Process attribute value

```
void process_attribute_value (uint8_t doing_html_tag, DISPLAY_HTML_TAGS_STACK
*tags_stack, uint8_t style_index, uint8_t *style_attribute_name, uint8_t
*style_attribute_value)
```

This is a sub function used by the display_html_file() function to process an attribute value as it is found.

This is a universal function – modification is not normally required.

Look for matching style

```
void display_html_look_for_matching_style (DISPLAY_HTML_TAGS_STACK *tags_stack,
uint8_t total_styles_count, uint8_t *style_name)
```

This is a sub function used by the display_html_file() function to look to see if a style has been stored with a matching name

This is a universal function – modification is not normally required.

Read styles

```
void display_html_read_styles (uint32_t *file_size, uint8_t
*this_doc_styles_count, uint8_t doing_global_styles)
```

This function is used to read a global styles .css file and called by the display_html_file() function to read styles in the header of an HTML page being displayed.

This is a universal function – modification is not normally required.

Find first occurance of string within a string

```
uint8_t *find_string_in_string (uint8_t *examine_string, const uint8_t
*looking_for_string)
```

This function Looks for the first occurrence of a constant string within a variable string

This is a universal function – modification is not normally required.

TROUBLESHOOTING

GENERAL TROUBLESHOOTING NOTES

Getting a new screen to work can be frustrating as until you get the control signals and initialisation sequence right you won't see anything on the screen. Here are a few tips to help:-

Double check IO pin definitions in the driver header file.

Verify with a scope that all of the control and data pins to the screen are working correctly.

Add additional null execution steps to the DISPLAY_WRITE_DATA and DISPLAY_READ_DATA defines in the display.h file in case more time is required for signals to stabilise on your PCB. This is more likely to be an issue for 2 layer PCB's (no ground plane) with long tack lengths or screens with long ribbon / FFC cable connections.

Check that no other device on the data bus is outputting while the driver is trying to communicate with the screen.

If your screen controller IC includes a status register try reading the status from the screen and confirm that you get the expected response.

If you're using output latches for some of the screen control pins, instead of pins connected directly to your processor, check your output latch function restores the previous output on the data bus when it exits, to avoid destroying the data the driver function is writing to the screen.

Check that your microcontroller is not resetting due to a watchdog timer timeout.

Check that you have enough stack space allocated. Maybe a stack overrun is occuring?

Re-read the datasheet to find out what you're doing wrong!

Designed by:

IBEX UK Limited
32A Station Road West
Oxted
Surrey
RH8 9EU
England
Tel: +44 (0)1883 716 726
E-mail: info@ibexuk.com
Web: www.ibexuk.com