CrossWorks Graphics Library

Version: 3.2

Contents

CrossWorks Graphics Library

About the CrossWorks Graphics Library

The *CrossWorks Graphics Library* presents a standardized API for delivering high-quality example code for a wide range of microcontrollers and evaluation boards. Additional components that integrate with the Graphics Library are:

- *CrossWorks Device Library*: provides drivers for common digital sensors, such as accelerometers, gyroscopes, magnetometers, and so on.
- *CrossWorks Shield Library*: provides drivers for a range of Arduino-style shields.
- *CrossWorks CoreBASIC Library*: provides a full-features network-enabled BASIC interpreter which makes full use of all the features in these libraries.

Architecture

The *CrossWorks Graphics Library* is one part of the *CrossWorks Target Library*. Many of the low-level functions provided by the target library are built using features of the *CrossWorks Tasking Library* for multi-threaded operation.

Delivery format

The *CrossWorks Graphics Library* is delivered in source form.

Feedback

This facility is a work in progress and may undergo rapid change. If you have comments, observations, suggestions, or problems, please feel free to air them on the **CrossWorks Target and Platform API** discussion forum.

License

The following terms apply to the Rowley Associates Graphics Library.

General terms

The source files in this package are not public domain and are not open source. They represent a substantial investment undertaken by Rowley Associates to assist CrossWorks customers to prototype solutions using well-written, tested drivers.

CrossWorks Commercial License holders

If you hold a paid-for CrossWorks Version 3 or later commercial license, you are free to compile this package and incorporate the resulting object code in your own products without royalties and without additional license fees. Customers holding a CrossWorks Version 1 or 2 commercial license are required to upgrade to CrossWorks Version 3 to use this software.

Non-Commercial License holders

If you hold a paid-for CrossWorks Version 3 or later non-commercial license, you are free to compile this package and incorporate the resulting object code in your own projects, for your own use, without royalties and without additional license fees. You are, however, prohibited from distributing the linked code, incorporating the object code from this library, in binary form. Customers holding a CrossWorks Version 1 or 2 non-commercial license are required to purchase CrossWorks Version 3 to use this software.

Introduction

About the CrossWorks Graphics Library

The *CrossWorks Graphics Library* is a standard API that runs on a collection of popular microprocessors and evaluation boards. It is a way for Rowley Associates to deliver examples, from simple to complex, for those boards.

In particular, the Graphics Library requires the *CrossWorks Tasking Library* for operation. Because the Graphics Library, and facilities built on top of it, use interrupts and background processing, we made the decision to use the CrossWorks Tasking Library as a foundation stone for the Platform Library. We have not abstracted the Graphics Library to use a generic RTOS as this adds more complexity to the design.

Why use the Graphics Library?

Standardizing on the Graphics Library provised a certain amount of portability for you applications. Rather than using vendor-supplied libraries that get you running quickly on their silicon, you can invest some time learning the Graphics Library and use that knowledge across different architectures. You are, however, committing to use CrossWorks, CTL, and the Graphics Library for the long term.

What the Graphics Library isn't

The Graphics Library it is not a general-purpose API supporting every feature offered by common devices, nor does it cater for all devices within a family. The Graphics Library is tested on the microprocessors and evaluation boards that Rowley Associates deliver examples for. Certainly, you can use it with little or no modification on boards that have other processors in the families we support, but you will need to customize the Graphics Library implementation yourself.

What the Graphics Library runs on

The Graphics Library runs on the following microprocessor families:

- LPC1700
- LM3S
- KL05Z
- KL25Z
- STM32F1
- STM32F4

The range of boards and microprocessors that run the Graphics Library continues to expand. Please check the CrossWorks web site for the latest information.

<ctl_gfx.h>

Overview

This is the primary header file for the CrossWorks Graphics Library.

For information on the use of this API, see CrossWorks Graphics Library.

API Summary

Context	
CTL_GFX_CONTEXT_t	Rectangle
CTL_GFX_DRIVER_t	Graphics driver
ctl_gfx_context	Active graphics context
ctl_gfx_driver	Active graphics driver
ctl_gfx_select_driver	Set active graphics driver
Control	
ctl_gfx_flush	Flush drawing pipeline
ctl_gfx_set_backlight	Set backlight level
ctl_gfx_set_contrast	Set contrast
ctl_gfx_set_rotation	Set display rotation
Properties	
CTL_GFX_PROPERTY_t	Graphics controller properties
ctl_gfx_screen_height	Inquire screen height
ctl_gfx_screen_width	Inquire screen width
Drawing	
CTL_GFX_POINT_t	Graphics point
CTL_GFX_RECTANGLE_t	Rectangle
CTL_GFX_RGB	Create API pixel
ctl_gfx_draw_circle	Draw circle
ctl_gfx_draw_line	Draw line
ctl_gfx_draw_lines	Draw polyline
ctl_gfx_draw_pixel	Draw pixel
ctl_gfx_draw_rectangle_wh	Draw rectangle
ctl_gfx_draw_rectangle_xy	Draw rectangle
ctl_gfx_fill_circle	Fill circle

ctl_gfx_fill_polygon	Fill polygon
ctl_gfx_fill_rectangle_wh	Fill rectangle
ctl_gfx_fill_rectangle_xy	Fill rectangle
ctl_gfx_set_pen_color	Set pen color
Text	
CTL_GFX_BITMAP_FONT_t	Raster font
CTL_GFX_GLYPH_t	Graphics point
ctl_gfx_draw_char	Draw character
ctl_gfx_draw_string	Draw string
ctl_gfx_draw_stringn	Draw length-limited string
ctl_gfx_set_text_color	Set text color

CTL_GFX_BITMAP_FONT_t

Synopsis

```
typedef struct {
  unsigned width;
  unsigned height;
  unsigned chars;
  const unsigned char *widths;
  const unsigned short *index;
  const unsigned char *bitmap;
} CTL_GFX_BITMAP_FONT_t;
```

Description

CTL_GFX_BITMAP_FONT_t describes a fixed-width raster font.

`width`

The width of each glyph in the font.

`height`

The height of each glyph in the font.

`chars`

The number of characters in the font.

`width`

A pointer to the width (in pixels) of each glyph in the font.

`index`

UCS encoding of each character in the font.

`bitmap`

A pointer to the raster data of the font.

CTL_GFX_CONTEXT_t

Synopsis

```
typedef struct {
  unsigned long api_pen_color;
  unsigned long device_pen_color;
  unsigned long api_text_color;
  unsigned long device_text_color;
  CTL_GFX_RECTANGLE_t api_clip;
  CTL_GFX_RECTANGLE_t device_clip;
  const CTL_GFX_BITMAP_FONT_t *current_font;
  int rotation;
  void (*convert_point)(CTL_GFX_POINT_t *, int , int);
} CTL_GFX_CONTEXT_t;
```

Description

CTL_GFX_CONTEXT_t contains the context for the active graphics controller.

api_pen_color

The 24-bit RGB device-independent pen color.

device_pen_color

The cached device color corresponding the `api_pen_color`.

api_text_color

The 24-bit RGB device-independent text color.

device_text_color

The cached device color corresponding the `api_text_color`.

api_clip

The clipping rectangle in API coordinates.

device_clip

The clipping rectangle in device coordinates.

current_font

The active font.

rotation

The API-level device rotation. `0` is no rotation, `1` is 90 degrees counter clockwise, 2 is 180 degrees, and 3 is 270 degrees counter clockwise.

convert_point

The function that applies the rotation selected in `rotation` to the API point **p** before being passed to the device driver.

CTL_GFX_DRIVER_t

Synopsis

```
typedef struct {
  long (*get_parameter)(CTL_GFX_DRIVER_s *, int);
  void (*draw_pixel)(CTL_GFX_DRIVER_s *, int , int);
  void (*draw_line)(CTL_GFX_DRIVER_s *, int , int , int , int);
  void (*fill_rectangle)(CTL_GFX_DRIVER_s *, int , int , int , int);
  void (*draw_circle)(CTL_GFX_DRIVER_s *, int , int , int);
  void (*fill_circle)(CTL_GFX_DRIVER_s *, int , int , int);
  void (*draw_glyph)(CTL_GFX_DRIVER_s *, int , int , const CTL_GFX_GLYPH_t *);
  void (*flush)(CTL_GFX_DRIVER_s *);
  unsigned long (*convert_pixel)(unsigned);
  void (*set_backlight)(CTL_GFX_DRIVER_s *, float);
  void (*set_contrast)(CTL_GFX_DRIVER_s *, float);
  int display_depth;
  int display_width;
  int display_height;
  int visible_width;
  int visible_height;
  unsigned default_background;
  unsigned default_foreground;
} CTL_GFX_DRIVER_t;
```

Description

CTL_GFX_DRIVER_t describes a graphics driver for an LCD or similar display.

`extra`

A place to store client-side data, not used by the graphics library.

`get_parameter`

Low-level method to inquire graphics controller capabilities.

`draw_pixel`

Method to draw a pixel at coordinate (*x*, *y*) using the active pen color.

`draw_line`

Method to draw a line from coordinate (*x0*, *y0*) to coordinate (*x1*, *y1*) using the active pen color. The driver implementation of this function does not need to distinguish horizontal and vertical lines, that's detected done by higher-level functions.

`fill_rectangle`

Method to fill a rectangle with top left (*x0*, *y0*) with width *w* and height *h*. The driver implementation can assume *w* and *h* and both non-zero and positive.

`draw_circle`

Method to draw a circle with center coordinate (*x*, *y*) and radius *r* in the active pen color.

`fill_circle`

Method to fill a circle with center coordinate (*x*, *y*) and radius *r* in the active pen color.

`draw_glyph`

Method to draw a raster glyph *glyph* at coordinate (*x*, *y*).

`flush`

Method to flush any outstanding graphics commands; may be zero to indicate that flushing is not supported by the driver.

`convert_pixel`

Method to convert a 24-bit RGB device-independent pixel to a controller-dependent device pixel.

`set_backlight`

Method to set the backlight level, `0` (dimmest) to `1` (brightest).

`set_contrast`

Method to set the contrast level, `0` (lowest) to `1` (highest).

`depth`

Cached color depth of display, in bits.

CTL_GFX_GLYPH_t

Synopsis

```
typedef struct {
  int width;
  int height;
  const unsigned char *raster;
} CTL_GFX_GLYPH_t;
```

Description

CTL_GFX_GLYPH_t describes a glyph in a font. The glyph is **width** pixels wide by **height** pixels high and the raster image is pointed to by **raster**.

CTL_GFX_POINT_t

Synopsis

```
typedef struct {
  int x;
  int y;
} CTL_GFX_POINT_t;
```

Description

CTL_GFX_POINT_t describes an abstract point on the graphics surface at Cartesian coordinate (*x*, *y*).

CTL_GFX_PROPERTY_t

Synopsis

```
typedef enum {
  CTL_GFX_LOGICAL_WIDTH,
  CTL_GFX_LOGICAL_HEIGHT,
  CTL_GFX_VISIBLE_WIDTH,
  CTL_GFX_VISIBLE_HEIGHT,
  CTL_GFX_COLOR_DEPTH,
  CTL_GFX_DEFAULT_BACKGROUND,
  CTL_GFX_DEFAULT_FOREGROUND
} CTL_GFX_PROPERTY_t;
```

Description

CTL_GFX_PROPERTY_t describes graphics controller properties that the client and driver can inquire about.

`CTL_GFX_LOGICAL_WIDTH`

Controller logical width. The controller logical width can be greater than the visible display width for the display panel.

`CTL_GFX_LOGICAL_HEIGHT`

Controller logical height. The controller logical height can be greater than the visible display height for the display panel.

`CTL_GFX_VISIBLE_WIDTH`

Controller visible width. The controller visible width is the number of pixels that the user sees on the display.

`CTL_GFX_VISIBLE_HEIGHT`

Controller visible height. The controller visible height is the number of pixels that the user sees on the display.

`CTL_GFX_COLOR_DEPTH`

Inquire the number of independent colors. For a 16-color palette display, this is 4. For 256 colors in palette, this is 8. For true color (64K+) this is 16 or 24.

`CTL_GFX_DEFAULT_BACKGROUND`

Inquire default background pixel value.

`CTL_GFX_DEFAULT_FOREGROUND`

Inquire default background pixel value.

CTL_GFX_RECTANGLE_t

Synopsis

```
typedef struct {
  CTL_GFX_POINT_t min;
  CTL_GFX_POINT_t max;
} CTL_GFX_RECTANGLE_t;
```

Description

CTL_GFX_RECTANGLE_t describes a rectangle.

`min`

Top left coordinate.

`min`

Bottom right coordinate.

CTL_GFX_RGB

Synopsis

```
#define CTL_GFX_RGB(R, G, B) (((R)<<16) | ((G)<<8) | B)
```

Description

CTL_GFX_RGB creates an RGB pixel from the RGB values (0–255) **R**, **G**, and **B**.

ctl_gfx_context

Synopsis

```
CTL_GFX_CONTEXT_t ctl_gfx_context;
```

Description

ctl_gfx_context contains the active graphics content maintained by the graphics library. It should be considered private.

ctl_gfx_draw_char

Synopsis

```
void ctl_gfx_draw_char(int x,
                       int y,
                       int ch);
```

Description

ctl_gfx_draw_char draws the character **ch** at coordinate (**x**, **y**) using the active font and text color.

ctl_gfx_draw_circle

Synopsis

```
void ctl_gfx_draw_circle(int x0,
                         int y0,
                         int radius);
```

Description

ctl_gfx_draw_circle draws a circle with center coordinate (**x0**, **y0**) and radius **radius** using the current pen color.

ctl_gfx_draw_line

Synopsis

```
void ctl_gfx_draw_line(int x0,
                       int y0,
                       int x1,
                       int y1);
```

Description

ctl_gfx_draw_line draws a one-pixel-wide line from coordinate (*x0*, *y0*) to (*x1*, *y1*) using the pen color.

ctl_gfx_draw_lines

Synopsis

```
void ctl_gfx_draw_lines(const CTL_GFX_POINT_t *points,
                        int n);
```

Description

ctl_gfx_draw_lines draws lines between the points listed in **points**. There are **n** points in the list.

ctl_gfx_draw_pixel

Synopsis

```
void ctl_gfx_draw_pixel(int x,
                        int y);
```

Description

ctl_gfx_draw_pixel draws a single pixel using the pen color at coordinate (*x*, *y*).

ctl_gfx_draw_rectangle_wh

Synopsis

```
void ctl_gfx_draw_rectangle_wh(int x,
                               int y,
                               int w,
                               int h);
```

Description

ctl_gfx_draw_rectangle_wh draws a rectangle at coordinate (**x**, **y**) that is *w* pixels wide and *h* pixels high using the current pen color.

ctl_gfx_draw_rectangle_xy

Synopsis

```
void ctl_gfx_draw_rectangle_xy(int x0,
                               int y0,
                               int x1,
                               int y1);
```

Description

ctl_gfx_draw_rectangle_xy draws a rectangle with opposite vertexes at at coordinates (**x0**, **y0**) and (**x1**, **y1**) using the current pen color.

ctl_gfx_draw_string

Synopsis

```
int ctl_gfx_draw_string(int x,
                        int y,
                        const char *text);
```

Description

ctl_gfx_draw_string draws the string **text** at coordinate (**x**, **y**) using the active font and text color.

Return Value

The value returned is the number of pixels taken by the string **text**.

ctl_gfx_draw_stringn

Synopsis

```
int ctl_gfx_draw_stringn(int x,
                         int y,
                         const char *text,
                         size_t len);
```

Description

ctl_gfx_draw_stringn draws the string **text** at coordinate (**x**, **y**) using the active font and text color. At most **len** characters are written. Any zero character within the first *len* characters will terminate drawing early.

Return Value

The value returned is the number of pixels taken by the string **text**.

ctl_gfx_driver

Synopsis

```
CTL_GFX_DRIVER_t *ctl_gfx_driver;
```

Description

ctl_gfx_driver is a pointer to the active graphics driver. If zero, no graphics driver is active.

ctl_gfx_fill_circle

Synopsis

```
void ctl_gfx_fill_circle(int x0,
                         int y0,
                         int radius);
```

Description

ctl_gfx_fill_circle fills a circle with center coordinate (**x0**, **y0**) and radius **radius** using the current pen color.

ctl_gfx_fill_polygon

Synopsis

```
void ctl_gfx_fill_polygon(CTL_GFX_POINT_t *vertices,
                          int n,
                          int xoffset,
                          int yoffset);
```

Description

ctl_gfx_fill_polygon fills a polygon bounded by the vertexes listed in **vertices**. There are **n** points in the list.

ctl_gfx_fill_rectangle_wh

Synopsis

```
void ctl_gfx_fill_rectangle_wh(int x,
                               int y,
                               int w,
                               int h);
```

Description

ctl_gfx_fill_rectangle_wh fills a rectangle at coordinate (**x**, **y**) that is *w* pixels wide and *h* pixels high using the current pen color.

ctl_gfx_fill_rectangle_xy

Synopsis

```
void ctl_gfx_fill_rectangle_xy(int x0,
                               int y0,
                               int x1,
                               int y1);
```

Description

ctl_gfx_fill_rectangle_xy fills a rectangle with opposite vertexes at at coordinates (**x0**, **y0**) and (**x1**, **y1**) using the current pen color.

ctl_gfx_flush

Synopsis

```
void ctl_gfx_flush(void);
```

Description

ctl_gfx_flush flushes any outstanding graphics operations to the display. Many displays update automatically as they are drawn to, but some display drivers cache drawing requests and require a flush to ensure that what is shown on the display corresponds to the drawing commands made.

ctl_gfx_screen_height

Synopsis

```
int ctl_gfx_screen_height(void);
```

Description

ctl_gfx_screen_height returns the visible screen height, in pixels.

ctl_gfx_screen_width

Synopsis

```
int ctl_gfx_screen_width(void);
```

Description

ctl_gfx_screen_width returns the visible screen width, in pixels.

ctl_gfx_select_driver

Synopsis

```
void ctl_gfx_select_driver(CTL_GFX_DRIVER_t *self);
```

Description

ctl_gfx_select_driver selects the driver **self** and makes it the active driver for subsequent graphics operations.

ctl_gfx_set_backlight

Description

ctl_gfx_set_backlight sets the backlight level to **level**, `0` (dimmest) to `1` (brightest).

ctl_gfx_set_contrast

Description

ctl_gfx_set_contrast sets the contrast to **level**, `0` (lowest) to `1` (highest).

ctl_gfx_set_pen_color

Synopsis

```
void ctl_gfx_set_pen_color(unsigned long color);
```

Description

ctl_gfx_set_pen_color sets the pen color to the API color *color*.

ctl_gfx_set_rotation

Description

ctl_gfx_set_rotation rotates the display counter clockwise by **rotation** × 90 degrees.

ctl_gfx_set_text_color

Synopsis

```
void ctl_gfx_set_text_color(unsigned long color);
```

Description

ctl_gfx_set_text_color sets the text color to the API color *color*.

<ctl_gfx_controller.h>

Overview

This header file provides utility functions for bus-interfaced graphics controllers. Typically, LCD modules come with an LCD panel and controller IC, and optionally a touch screen.

API Summary

Context	
CTL_GFX_CONTROLLER_t	Instance data
Control	
ctl_gfx_controller_setup_begin	Start graphics controller setup
ctl_gfx_controller_setup_end	Complete graphics controller setup
Setup	
ctl_gfx_controller_configure_12b_depth	Configure 12-bit color depth
ctl_gfx_controller_configure_16b_depth	Configure 16-bit color depth
ctl_gfx_controller_configure_16b_width	Configure 16-bit co-ordinate width
ctl_gfx_controller_configure_18b_depth	Configure 18-bit color depth
ctl_gfx_controller_configure_8b_depth	Configure 8-bit color depth
ctl_gfx_controller_configure_8b_spi_protocol	Configure 9-bit SPI protocol
ctl_gfx_controller_configure_8b_width	Configure 8-bit co-ordinate width
ctl_gfx_controller_configure_9b_spi_protocol	Configure 9-bit SPI protocol
ctl_gfx_controller_configure_depth	Configure color depth
ctl_gfx_controller_configure_epson_command_set	Configure driver for an Epson-style command set
ctl_gfx_controller_configure_philips_command_set	Configure driver for an Philips-style command set
Operation	
ctl_gfx_controller_hitachi_spi_write_pixels	Write pixels using Hitachi SPI protocol
ctl_gfx_controller_hitachi_spi_write_register	Write register using Hitachi SPI protocol
ctl_gfx_controller_move_cursor_at_0x4e_0x4f	Move cursor of an Samsung-like controller
ctl_gfx_controller_set_window_at_0x44_0x45_0x46	Set window of a Samsung-like controller
ctl_gfx_controller_set_window_at_0x50_0x51_0x52_	Set window of an Hitachi-like controller
ctl_gfx_write_command	Write command to controller
ctl_gfx_write_command_8b	Write command and 8-bit parameter to controller
Utility	
ctl_gfx_controller_write_16b_sequence	Send 16-bit command and parameter sequence

ctl_gfx_controller_write_8b_sequence	Send 8-bit command and parameter sequence
Controller	
ctl_gfx_controller_move_cursor_at_0x20_0x21	Move cursor of an Hitachi-like controller

CTL_GFX_CONTROLLER_t

Synopsis

```
typedef struct {
  CTL_GFX_DRIVER_t core;
  CTL_SPI_DEVICE_t *dev;
  void (*set_reset)(CTL_GFX_CONTROLLER_s *, int);
  void (*set_window)(CTL_GFX_CONTROLLER_s *, int , int , int , int);
  void (*write_register)(CTL_GFX_CONTROLLER_s *, unsigned , unsigned);
  void (*write_pixels)(CTL_GFX_CONTROLLER_s *, unsigned , int);
  void (*move_cursor)(CTL_GFX_CONTROLLER_s *, int , int);
  void (*write_command)(CTL_GFX_CONTROLLER_s *, int);
  void (*write_data_8b)(CTL_GFX_CONTROLLER_s *, int);
  void (*write_data_16b)(CTL_GFX_CONTROLLER_s *, int);
  void (*write_data_24b)(CTL_GFX_CONTROLLER_s *, int);
  void (*set_dc)(CTL_GFX_CONTROLLER_s *, int);
  unsigned char __window_trashed;
  unsigned char controller_address;
  unsigned char controller_caset;
  unsigned char controller_paset;
  unsigned char controller_ramwr;
} CTL_GFX_CONTROLLER_t;
```

Description

CTL_GFX_CONTROLLER_t contains the instance data for a graphics driver that is implemented by a simple windowing, bus-interfaced (or otherwise) graphics controller. Typical graphics controllers provide a register-data interface that is abstracted by the `write_register` and `write_pixels` methods, and an operating window that is abstracted by the `set_window` method.

`core`

The core graphics driver.

`dev`

The SPI device used to control SPI-attached controllers. If the device is attached using I2C or a parallel bus, this member must be set to zero.

`set_reset`

A method to control the reset signal to the controller. Many drivers simply ignore this method and rely on the system's power-on reset or higher-level code to reset the controller.

`set_window`

Method to set the window that **write_pixels** works on.

`move_cursor`

Method to set the cursor to within the window (if required by the controller).

`write_register`

Method to write **data** to register **reg**.

`write_pixels`

Method to write **pixel** to graphics RAM **n** times.

`write_command`

A method to write a command to the graphics controller. Graphics controllers typically separate commands from data by a single bit that is presented on an address line or digital control signal, within the SPI protocol or within the I2C protocol.

`write_data_8b`

Write 8 bits of data to the graphics controller.

`write_data_16b`

Write 16 bits of data to the graphics controller in the byte order appropriate for the controller.

`write_data_24b`

Write 16 bits of data to the graphics controller in the byte order appropriate for the controller.

`set_dc`

Set the state of the D/C or RS signal for SPI-connected graphics controllers. Graphics controllers that are interfaced in 8-bit SPI mode require a separate D/C or RS signal to indicate whether the frames transferred over the SPI bus are to be interpreted as data or commands. For I2C, 9-bit SPI, and parallel-interfaced controllers, this member must be zero.

`controller_address`

Internal address for I2C-connected and SPI-connected graphics controllers (if required).

`__window_trashed`

Internal flag indicating whether the window is invalid and requires initializing before writing to GRAM.

`controller_caset`

An internal member set by the device driver that corresponds to the "column address" command of the graphics controller.

`controller_paset`

An internal member set by the device driver that corresponds to the "page address" command of the graphics controller.

`controller_ramwr`

An internal member set by the device driver that corresponds to the "RAM write" command of the graphics controller.

ctl_gfx_controller_configure_12b_depth

Synopsis

```
void ctl_gfx_controller_configure_12b_depth(CTL_GFX_CONTROLLER_t *self);
```

Description

ctl_gfx_controller_configure_12b_depth configures the driver **self** methods `draw_pixel`, `fill_rectangle`, and `convert_pixel` for 12-bit color depth and sets the `display_depth` member to 12.

ctl_gfx_controller_configure_16b_depth

Synopsis

```
void ctl_gfx_controller_configure_16b_depth(CTL_GFX_CONTROLLER_t *self);
```

Description

ctl_gfx_controller_configure_16b_depth configures the driver **self** methods `draw_pixel`, `fill_rectangle`, and `convert_pixel` for 16-bit color depth and sets the `display_depth` member to 16.

ctl_gfx_controller_configure_16b_width

Synopsis

```
void ctl_gfx_controller_configure_16b_width(CTL_GFX_CONTROLLER_t *self);
```

Description

ctl_gfx_controller_configure_16b_width configures the driver **self** to use 16-bit coordinates in SPI-based commands for Epson and Philips controllers.

ctl_gfx_controller_configure_18b_depth

Synopsis

```
void ctl_gfx_controller_configure_18b_depth(CTL_GFX_CONTROLLER_t *self);
```

Description

ctl_gfx_controller_configure_18b_depth configures the driver **self** methods `draw_pixel`, `fill_rectangle`, and `convert_pixel` for 18-bit color depth and sets the `display_depth` member to 18.

ctl_gfx_controller_configure_8b_depth

Synopsis

```
void ctl_gfx_controller_configure_8b_depth(CTL_GFX_CONTROLLER_t *self);
```

Description

ctl_gfx_controller_configure_8b_depth configures the driver **self** methods `draw_pixel`, `fill_rectangle`, and `convert_pixel` for 8-bit color depth and sets the `display_depth` member to 8.

ctl_gfx_controller_configure_8b_spi_protocol

Synopsis

```
void ctl_gfx_controller_configure_8b_spi_protocol(CTL_GFX_CONTROLLER_t *self,
                                                  CTL_SPI_DEVICE_t *dev,
                                                  void (*set_dc)
(CTL_GFX_CONTROLLER_t *, int));
```

Description

ctl_gfx_controller_configure_8b_spi_protocol configures the driver **self** to use 4-wire SPI 8-bit protocol on the device **dev** with D/C provided as a separate digital output controlled by **set_dc**.

ctl_gfx_controller_configure_8b_width

Synopsis

```
void ctl_gfx_controller_configure_8b_width(CTL_GFX_CONTROLLER_t *self);
```

Description

ctl_gfx_controller_configure_8b_width configures the driver **self** to use 8-bit coordinates in SPI-based commands for Epson and Philips controllers.

ctl_gfx_controller_configure_9b_spi_protocol

Synopsis

```
void ctl_gfx_controller_configure_9b_spi_protocol(CTL_GFX_CONTROLLER_t *self,
                                                  CTL_SPI_DEVICE_t *dev);
```

Description

ctl_gfx_controller_configure_9b_spi_protocol configures the driver **self** to use 3-wire SPI 9-bit protocol on the device **dev** with D/C provided in a prefix bit.

ctl_gfx_controller_configure_depth

Synopsis

```
void ctl_gfx_controller_configure_depth(CTL_GFX_CONTROLLER_t *self,
                                        int depth);
```

Description

ctl_gfx_controller_configure_depth configures the driver **self** methods `draw_pixel`, `fill_rectangle`, and `convert_pixel` for the color depth **depth**, which must be 8, 12, 16, or 16, by calling the appropriate setup routine.

See Also

ctl_gfx_controller_configure_8b_depth, **ctl_gfx_controller_configure_12b_depth**, **ctl_gfx_controller_configure_16b_depth**, **ctl_gfx_controller_configure_18b_depth**

ctl_gfx_controller_configure_epson_command_set

Synopsis

```
void ctl_gfx_controller_configure_epson_command_set(CTL_GFX_CONTROLLER_t *self);
```

Description

ctl_gfx_controller_configure_epson_command_set configures the driver **self** for an Epson-style command set by setting the correct commands in `controller_caset`, `controller_paset`, and `controller_ramwr`.

ctl_gfx_controller_configure_philips_command_set

Synopsis

```
void ctl_gfx_controller_configure_philips_command_set(CTL_GFX_CONTROLLER_t *self);
```

Description

ctl_gfx_controller_configure_philips_command_set configures the driver **self** for an Epson-style command set by setting the correct commands in `controller_caset`, `controller_paset`, and `controller_ramwr`.

ctl_gfx_controller_hitachi_spi_write_pixels

Synopsis

```
void ctl_gfx_controller_hitachi_spi_write_pixels(CTL_GFX_CONTROLLER_t *self,
                                                 unsigned pixel,
                                                 int n);
```

Description

ctl_gfx_controller_hitachi_spi_write_pixels issues a command to the graphics controller **self** using Hitachi-like SPI protocol to *n* repeated pixels with value **pixel**. The value **pixel** is a converted, controller-specific pixel value.

ctl_gfx_controller_hitachi_spi_write_register

Synopsis

```
void ctl_gfx_controller_hitachi_spi_write_register(CTL_GFX_CONTROLLER_t *self,
                                                    unsigned reg,
                                                    unsigned data);
```

Description

ctl_gfx_controller_hitachi_spi_write_register issues a command to the graphics controller **self** using Hitachi-like SPI protocol to write register **reg** with **data**.

ctl_gfx_controller_move_cursor_at_0x20_0x21

Synopsis

```
void ctl_gfx_controller_move_cursor_at_0x20_0x21(CTL_GFX_CONTROLLER_t *self,
                                                 int x,
                                                 int y);
```

Description

ctl_gfx_controller_move_cursor_at_0x20_0x21 issues the "move cursor" command to an Hitachi-like graphics controller with horizontal and vertical registers at `0x20` and `0x21`.

ctl_gfx_controller_move_cursor_at_0x4e_0x4f

Synopsis

```
void ctl_gfx_controller_move_cursor_at_0x4e_0x4f(CTL_GFX_CONTROLLER_t *self,
                                                 int x,
                                                 int y);
```

Description

ctl_gfx_controller_move_cursor_at_0x4e_0x4f issues the "move cursor" command to an Samsung-like graphics controller with horizontal and vertical registers at `0x4e` and `0x4f`.

ctl_gfx_controller_set_window_at_0x44_0x45_0x46

Synopsis

```
void ctl_gfx_controller_set_window_at_0x44_0x45_0x46(CTL_GFX_CONTROLLER_t *self,
                                                     int x,
                                                     int y,
                                                     int w,
                                                     int h);
```

Description

ctl_gfx_controller_set_window_at_0x44_0x45_0x46 issues the "set window" command to a Samsung-like graphics controller with a single 16-bit vertical start/end register at `0x44` and horizontal start/end registers at `0x46` and `0x46`.

ctl_gfx_controller_set_window_at_0x50_0x51_0x52_0x53

Synopsis

```
void ctl_gfx_controller_set_window_at_0x50_0x51_0x52_0x53(CTL_GFX_CONTROLLER_t *self,
                                                          int x,
                                                          int y,
                                                          int w,
                                                          int h);
```

Description

ctl_gfx_controller_set_window_at_0x50_0x51_0x52_0x53 issues the "set window" command to an Hitachi-like graphics controller with horizontal start/end registers at `0x50` and `0x51` and vertical start/end registers at `0x52` and `0x53`.

ctl_gfx_controller_setup_begin

Synopsis

```
void ctl_gfx_controller_setup_begin(CTL_GFX_CONTROLLER_t *self);
```

Description

ctl_gfx_controller_setup_begin starts set up of the graphics controller **self** by zeroing all structure members and calling `ctl_gfx_setup_begin`.

ctl_gfx_controller_setup_end

Synopsis

```
void ctl_gfx_controller_setup_end(CTL_GFX_CONTROLLER_t *self);
```

Description

ctl_gfx_controller_setup_end completes set up of the graphics controller **self** and calls `ctf_gfx_setup_end` to ensure that the device driver has set members appropriately.

ctl_gfx_controller_write_16b_sequence

Synopsis

```
void ctl_gfx_controller_write_16b_sequence(CTL_GFX_CONTROLLER_t *self,
                                           const unsigned short *seq,
                                           size_t n,
                                           unsigned delay);
```

Description

ctl_gfx_controller_write_16b_sequence sends 16-bit command and parameters pointed to by **seq** using the **write** function. The size of the sequence is **n** bytes. The parameter *delay* indicates a distinguished value that, if found in the command sequence, indicates a delay, in milliseconds, taken from the parameter.

ctl_gfx_controller_write_8b_sequence

Synopsis

```
void ctl_gfx_controller_write_8b_sequence(CTL_GFX_CONTROLLER_t *self,
                                          const unsigned char *seq,
                                          size_t n,
                                          unsigned delay);
```

Description

ctl_gfx_controller_write_8b_sequence sends 8-bit command and parameters pointed to by **seq** using the **write** function. The size of the sequence is **n** bytes. The parameter *delay* indicates a distinguished value that, if found in the command sequence, indicates a delay, in milliseconds, taken from the parameter.

ctl_gfx_write_command

Synopsis

```
void ctl_gfx_write_command(CTL_GFX_CONTROLLER_t *self,
                           unsigned command);
```

Description

Write the command **command** to the controller **self** using the `write_command` method. This simply wraps the `write_command` method so that client source code looks clean.

ctl_gfx_write_command_8b

Synopsis

```
void ctl_gfx_write_command_8b(CTL_GFX_CONTROLLER_t *self,
                              unsigned command,
                              unsigned parameter);
```

Description

Write the command **command** to the controller **self** using the `write_command` method and write the data item **parameter** using the `write_data_8b` method. This simply wraps the two methods so that client source code looks clean.

<ctl_gfx_modules.h>

Overview

This header file provides utility functions for LCD modules that we have direct experience with and can test. We will expand this as we write code for new modules delivered on evaluation boards.

API Summary

AM320240LDTNQW	
am320240ldtnqw_power_up	Power up panel
am320240ldtnqw_setup_begin_spi	Start driver setup
am320240ldtnqw_setup_end	Complete driver setup
AM240320D4TOQW	
am240320d4toqw_power_up	Power up panel
am240320d4toqw_setup_begin_bus	Start driver setup
am240320d4toqw_setup_end	Complete driver setup
HW240320F-2D-0B-L1-T4	
hw240320f_2d_0b_l1_t4_power_up	Power up panel
hw240320f_2d_0b_l1_t4_setup_begin_bus	Start driver setup
hw240320f_2d_0b_l1_t4_setup_end	Complete driver setup
FGD280E3715V1	
fgd280e3715v1_power_up	Power up panel
fgd280e3715v1_setup_begin_bus	Start driver setup
fgd280e3715v1_setup_end	Complete driver setup
FS-K320QVB-V1	
fs_k320qvb_v1_power_up	Power up panel
fs_k320qvb_v1_setup_begin_bus	Start driver setup
fs_k320qvb_v1_setup_end	Complete driver setup
FS-K350QVG-V2	
fs_k350qvg_v2_power_up	Power up panel
fs_k350qvg_v2_setup_begin_bus	Start driver setup
fs_k350qvg_v2_setup_end	Complete driver setup
LPH88	
lph88_power_up	Power up panel
lph88_setup_begin_spi	Start driver setup

lph88_setup_end	Complete driver setup
LY120-096096	
ly120_096096_power_up	Power up panel
ly120_096096_setup_begin_i2c	Start driver setup
ly120_096096_setup_end	Complete driver setup
LY120-096016	
ly120_096016_setup_begin_i2c	Start driver setup
ly120_096016_setup_end	Complete driver setup
LY190-128064	
ly190_128064_power_up	Power up panel
ly190_128064_setup_begin_i2c	Start driver setup
ly190_128064_setup_end	Complete driver setup
NHD-C12832A1Z	
nhd_c12832a1z_power_up	Power up panel
nhd_c12832a1z_setup_begin_spi	Start driver setup
nhd_c12832a1z_setup_end	Complete driver setup
Olimex GE8	
olimex_ge8_n6110_power_up	Power up panel
olimex_ge8_n6110_setup_begin	Start driver setup
olimex_ge8_n6110_setup_end	Complete driver setup
Olimex GE12	
olimex_ge12_n6110_power_up	Power up panel
olimex_ge12_n6110_setup_begin	Start driver setup
olimex_ge12_n6110_setup_end	Complete driver setup

am240320d4toqw_power_up

Synopsis

```
void am240320d4toqw_power_up(CTL_GFX_CONTROLLER_t *self);
```

Description

am240320d4toqw_power_up powers up the panel.

am240320d4toqw_setup_begin_bus

Synopsis

```
void am240320d4toqw_setup_begin_bus(CTL_GFX_CONTROLLER_t *self);
```

Description

am240320d4toqw_setup_begin_bus completes initialization of the panel after it has been been powered up.

Note

As seen on a STM3240G-EVAL board with an MB785 LCD daughterboard.

am240320d4toqw_setup_end

Synopsis

```
void am240320d4toqw_setup_end(CTL_GFX_CONTROLLER_t *self);
```

Description

am240320d4toqw_setup_end completes initialization of the panel after it has been been powered up.

am320240ldtnqw_power_up

Synopsis

```
void am320240ldtnqw_power_up(CTL_GFX_CONTROLLER_t *self);
```

Description

am320240ldtnqw_power_up powers up the panel.

am320240ldtnqw_setup_begin_spi

Synopsis

```
void am320240ldtnqw_setup_begin_spi(CTL_GFX_CONTROLLER_t *self,
                                    CTL_SPI_DEVICE_t *dev,
                                    int controller_address);
```

Description

am320240ldtnqw_setup_begin_spi start the set up of the AM320240LDTNQW panel **self** using the SPI device **dev** with prefix address **controller_address**.

Note

As seen on a Keil MCBSTM32C board.

am320240ldtnqw_setup_end

Synopsis

```
void am320240ldtnqw_setup_end(CTL_GFX_CONTROLLER_t *self);
```

Description

am320240ldtnqw_setup_end completes initialization of the panel after it has been been powered up.

fgd280e3715v1_power_up

Synopsis

```
void fgd280e3715v1_power_up(CTL_GFX_CONTROLLER_t *self);
```

Description

fgd280e3715v1_power_up powers up the panel.

fgd280e3715v1_setup_begin_bus

Synopsis

```
void fgd280e3715v1_setup_begin_bus(CTL_GFX_CONTROLLER_t *self);
```

Note

As seen on a Seeed Studio TFT Touch Shield.

fgd280e3715v1_setup_end

Synopsis

```
void fgd280e3715v1_setup_end(CTL_GFX_CONTROLLER_t *self);
```

Description

fgd280e3715v1_setup_end completes initialization of the panel after it has been been powered up.

fs_k320qvb_v1_power_up

Synopsis

```
void fs_k320qvb_v1_power_up(CTL_GFX_CONTROLLER_t *self);
```

Description

fs_k320qvb_v1_power_up powers up the panel.

fs_k320qvb_v1_setup_begin_bus

Synopsis

```
void fs_k320qvb_v1_setup_begin_bus(CTL_GFX_CONTROLLER_t *self);
```

Note

As seen on an Olimex STM32-LCD.

fs_k320qvb_v1_setup_end

Synopsis

```
void fs_k320qvb_v1_setup_end(CTL_GFX_CONTROLLER_t *self);
```

Description

fs_k320qvb_v1_setup_end completes initialization of the panel after it has been been powered up.

fs_k350qvg_v2_power_up

Synopsis

```
void fs_k350qvg_v2_power_up(CTL_GFX_CONTROLLER_t *self);
```

Description

fs_k350qvg_v2_power_up powers up the panel.

fs_k350qvg_v2_setup_begin_bus

Synopsis

```
void fs_k350qvg_v2_setup_begin_bus(CTL_GFX_CONTROLLER_t *self);
```

Note

As seen on the element-14 STM32-BB for the STM32F4DISCOVERY.

fs_k350qvg_v2_setup_end

Synopsis

```
void fs_k350qvg_v2_setup_end(CTL_GFX_CONTROLLER_t *self);
```

Description

fs_k350qvg_v2_setup_end completes initialization of the panel after it has been been powered up.

hw240320f_2d_0b_l1_t4_power_up

Synopsis

```
void hw240320f_2d_0b_l1_t4_power_up(CTL_GFX_CONTROLLER_t *self);
```

Description

hw240320f_2d_0b_l1_t4_power_up powers up the panel.

hw240320f_2d_0b_l1_t4_setup_begin_bus

Synopsis

```
void hw240320f_2d_0b_l1_t4_setup_begin_bus(CTL_GFX_CONTROLLER_t *self);
```

Note

As seen on an Adafruit TFT Touch Shield and a NuElectronics TFT Touch Shield.

hw240320f_2d_0b_l1_t4_setup_end

Synopsis

```
void hw240320f_2d_0b_l1_t4_setup_end(CTL_GFX_CONTROLLER_t *self);
```

Description

hw240320f_2d_0b_l1_t4_setup_end completes initialization of the panel after it has been been powered up.

lph88_power_up

Synopsis

```
void lph88_power_up(CTL_GFX_CONTROLLER_t *self);
```

Description

lph88_power_up powers up the panel.

lph88_setup_begin_spi

Synopsis

```
void lph88_setup_begin_spi(CTL_GFX_CONTROLLER_t *self,
                           CTL_SPI_DEVICE_t *dev,
                           int controller_address);
```

Description

lph88_setup_begin_spi start the set up of the LPH88 controller **self** using the SPI device **dev** with prefix address **controller_address**.

Note

The LPH88*xxx* panel is typically seen in S65 displays. It's a 132×176×256K color display.

lph88_setup_end

Synopsis

```
void lph88_setup_end(CTL_GFX_CONTROLLER_t *self);
```

Description

lph88_setup_end completes initialization of the panel after it has been been powered up.

ly120_096016_setup_begin_i2c

Synopsis

```
void ly120_096016_setup_begin_i2c(SSD1327_DRIVER_t *self,
                                  CTL_I2C_BUS_t *i2c,
                                  int addr);
```

Note

As seen on a Seeed Studio 96×96 OLED twig module.

ly120_096016_setup_end

Synopsis

```
void ly120_096016_setup_end(SSD130x_DRIVER_t *self);
```

Description

ly120_096016_setup_end completes initialization of the panel after it has been been powered up.

ly120_096096_power_up

Synopsis

```
void ly120_096096_power_up(SSD1327_DRIVER_t *self);
```

Description

ly120_096096_power_up powers up the panel.

ly120_096096_setup_begin_i2c

Synopsis

```
void ly120_096096_setup_begin_i2c(SSD1327_DRIVER_t *self,
                                  CTL_I2C_BUS_t *i2c,
                                  int addr);
```

Note

As seen on a Seeed Studio 96×96 OLED twig module.

ly120_096096_setup_end

Synopsis

```
void ly120_096096_setup_end(SSD1327_DRIVER_t *self);
```

Description

ly120_096096_setup_end completes initialization of the panel after it has been been powered up.

ly190_128064_power_up

Synopsis

```
void ly190_128064_power_up(SSD130x_DRIVER_t *self);
```

Description

ly190_128064_power_up powers up the panel.

ly190_128064_setup_begin_i2c

Synopsis

```
void ly190_128064_setup_begin_i2c(SSD130x_DRIVER_t *self,
                                  CTL_I2C_BUS_t *i2c,
                                  int addr,
                                  void *frame);
```

Note

As seen on a Seeed Studio 128×64 OLED brick module.

ly190_128064_setup_end

Synopsis

```
void ly190_128064_setup_end(SSD130x_DRIVER_t *self);
```

Description

ly190_128064_setup_end completes initialization of the panel after it has been been powered up.

nhd_c12832a1z_power_up

Synopsis

```
void nhd_c12832a1z_power_up(ST7565_DRIVER_t *self);
```

Description

nhd_c12832a1z_power_up powers up the panel.

nhd_c12832a1z_setup_begin_spi

Synopsis

```
void nhd_c12832a1z_setup_begin_spi(ST7565_DRIVER_t *self,
                                   CTL_SPI_DEVICE_t *dev,
                                   void (*set_dc)(CTL_GFX_CONTROLLER_t *, int));
```

Note

As seen on an mbed Application Board.

nhd_c12832a1z_setup_end

Synopsis

```
void nhd_c12832a1z_setup_end(ST7565_DRIVER_t *self);
```

Description

nhd_c12832a1z_setup_end completes initialization of the panel after it has been been powered up.

olimex_ge12_n6110_power_up

Synopsis

```
void olimex_ge12_n6110_power_up(CTL_GFX_CONTROLLER_t *self);
```

Description

olimex_ge12_n6110_power_up powers up the panel.

olimex_ge12_n6110_setup_begin

Synopsis

```
void olimex_ge12_n6110_setup_begin(CTL_GFX_CONTROLLER_t *self,
                                   CTL_SPI_DEVICE_t *dev,
                                   int depth);
```

olimex_ge12_n6110_setup_end

Synopsis

```
void olimex_ge12_n6110_setup_end(CTL_GFX_CONTROLLER_t *self);
```

Description

olimex_ge12_n6110_setup_end completes initialization of the panel after it has been been powered up.

olimex_ge8_n6110_power_up

Synopsis

```
void olimex_ge8_n6110_power_up(CTL_GFX_CONTROLLER_t *self);
```

Description

olimex_ge8_n6110_power_up powers up the panel.

olimex_ge8_n6110_setup_begin

Synopsis

```
void olimex_ge8_n6110_setup_begin(CTL_GFX_CONTROLLER_t *self,
                                  CTL_SPI_DEVICE_t *dev,
                                  int depth);
```

Description

olimex_ge8_n6110_setup_begin sets up the CrossWorks graphics driver **d** to work correctly with an Olimex N6110 ‘GE8’ display with bit depth **depth**. This driver supports a depths of 8 and 12 bits—that is either 256 or 4,096 colors.

Description

Olimex evaluation boards and the MOD_NOKIA6110 displays are shipped with a label, ‘GE8’ or ‘GE12,’ that identifies the particular graphics controller the display uses.

Resources

The GE8 display is based on an Epson S1D15G00 controller which runs at up to 6MHz.

Web page

http://www.olimex.com/dev/mod_nokia6610.html

olimex_ge8_n6110_setup_end

Synopsis

```
void olimex_ge8_n6110_setup_end(CTL_GFX_CONTROLLER_t *self);
```

Description

olimex_ge8_n6110_setup_end completes initialization of the panel after it has been been powered up.

<ctl_gfx_private.h>

Overview

This is the primary header file for implementing device drivers for the CrossWorks Graphics Library.

API Summary

Setup	
ctl_gfx_setup_begin	Begin driver initialization
ctl_gfx_setup_end	Finalize driver initialization
ctl_gfx_write_8b_commands	Send 8-bit command sequence
Pixels	
ctl_gfx_bgr_444	Convert 24-bit color to 12-bit BGR444
ctl_gfx_bgr_565	Convert 24-bit color to 16-bit BGR565
ctl_gfx_bgr_888	Convert 24-bit color to 24-bit BGR888
ctl_gfx_mono	Convert monochrome to device-independent pixel
ctl_gfx_rgb_332	Convert 24-bit color to 8-bit RGB332
ctl_gfx_rgb_565	Convert 24-bit color to 16-bit RGB565
ctl_gfx_rgb_888	Convert 24-bit color to 24-bit RGB888
Fonts	
ctl_gfx_find_glyph	Find glyph in raster font

ctl_gfx_bgr_444

Synopsis

```
unsigned long ctl_gfx_bgr_444(unsigned long u);
```

Description

ctl_gfx_bgr_444 converts a 24-bit RGB device-independent pixel to BGR444 format. In this format, the bits are arranged BBBB | GGGG | RRRR.

ctl_gfx_bgr_565

Synopsis

```
unsigned long ctl_gfx_bgr_565(unsigned long u);
```

Description

ctl_gfx_bgr_565 converts a 24-bit RGB device-independent pixel to BGR565 format. In this format, the bits are arranged BBBBB | GGGGGG | RRRRR.

ctl_gfx_bgr_888

Synopsis

```
unsigned long ctl_gfx_bgr_888(unsigned long u);
```

Description

ctl_gfx_bgr_888 converts a 24-bit RGB device-independent pixel to BGR888 format. In this format, the bits are arranged BBBBBBBB | GGGGGGGG | RRRRRRRR.

ctl_gfx_find_glyph

Synopsis

```
int ctl_gfx_find_glyph(const CTL_GFX_BITMAP_FONT_t *font,
                       int ch,
                       CTL_GFX_GLYPH_t *glyph);
```

Description

ctl_gfx_find_glyph finds the gly with UCS code **ch** in the raster font **font** and assigns glyph information to object pointer to by **glyph**. **glyph** can be zero, indicating that the additional information is not required.

Return Value

ctl_gfx_find_glyph returns zero if the glyph cannot be found and non-zero if the glyph is found.

ctl_gfx_mono

Synopsis

```
unsigned long ctl_gfx_mono(unsigned long u);
```

Description

ctl_gfx_mono converts a single-bit monochrome pixel (zero is black, non-zero is white) to a 24-bit RGB device-independent pixel.

ctl_gfx_rgb_332

Synopsis

```
unsigned long ctl_gfx_rgb_332(unsigned long u);
```

Description

ctl_gfx_rgb_332 converts a 24-bit RGB device-independent pixel to RGB332 format. In this format, the bits are arranged RRR | GGG | BB.

ctl_gfx_rgb_565

Synopsis

```
unsigned long ctl_gfx_rgb_565(unsigned long u);
```

Description

ctl_gfx_rgb_565 converts a 24-bit RGB device-independent pixel to RGB565 format. In this format, the bits are arranged RRRRR | GGGGGG | BBBBB.

ctl_gfx_rgb_888

Synopsis

```
unsigned long ctl_gfx_rgb_888(unsigned long u);
```

Description

ctl_gfx_rgb_888 converts a 24-bit RGB device-independent pixel to RGB888 format. In this format, the bits are arranged RRRRRRRR | GGGGGGGG | BBBBBBBB and the correspondence is one-to-one.

ctl_gfx_setup_begin

Synopsis

```
void ctl_gfx_setup_begin(CTL_GFX_DRIVER_t *self);
```

Description

ctl_gfx_setup_begin initializes the driver **self** and selects it as the active graphics driver.

ctl_gfx_setup_end

Synopsis

```
void ctl_gfx_setup_end(CTL_GFX_DRIVER_t *self);
```

Description

ctl_gfx_setup_end finalizes the initialization of **self** and selects it as the active graphics driver.

ctl_gfx_write_8b_commands

Synopsis

```
void ctl_gfx_write_8b_commands(const unsigned char *seq,
                               size_t n,
                               unsigned delay,
                               void (*write)(unsigned));
```

Description

ctl_gfx_write_8b_commands sends 8-bit commands pointed to by **seq** using the **write** function. The size of the sequence is **n** bytes. The parameter *delay* indicates the distinguished value that, if found in the command sequence, indicates a delay, in milliseconds, taken from the following byte.

<ili9325.h>

Overview

Device driver for an Ilitek ILI9325.

The AdaFruit and NuElectronics shields have one of these, along with a number of others.

API Summary

Setup	
ili9325_setup_begin_bus	Start driver initialization (bus)
ili9325_setup_end	Complete driver initialization

ili9325_setup_begin_bus

Synopsis

```
void ili9325_setup_begin_bus(CTL_GFX_CONTROLLER_t *self);
```

Description

ili9325_setup_begin_bus starts initialization of the ILI9325. After calling this function, you must fill in both the `write_register` and `write_pixel` methods, power-on the display, and initialize it.

ili9325_setup_end

Synopsis

```
void ili9325_setup_end(CTL_GFX_CONTROLLER_t *self);
```

Description

ili9325_setup_end finalizes initialization of the ILI9325. After calling this function, the graphics controller is initialized, selected, and ready for use.

<hd66773.h>

Overview

Device driver for an Hitachi (Renesas) HD66773 controller.

This controller can be driven using a 6800 bus, 8080 bus, or by SPI.

This is 132x176x64K colors.

The device can be 0x70 or 0x74 depending on the ID pin.

API Summary

Setup	
hd66773_setup_begin_bus	Start driver initialization (bus)
hd66773_setup_begin_spi	Start driver initialization (SPI)
hd66773_setup_end	Complete driver initialization

hd66773_setup_begin_bus

Synopsis

```
void hd66773_setup_begin_bus(CTL_GFX_CONTROLLER_t *self);
```

Description

hd66773_setup_begin_bus starts initialization of the HD66773.

hd66773_setup_begin_spi

Synopsis

```
void hd66773_setup_begin_spi(CTL_GFX_CONTROLLER_t *self,
                             CTL_SPI_DEVICE_t *dev,
                             int controller_address);
```

Description

hd66773_setup_begin_spi starts initialization of the HD66773.

hd66773_setup_end

Synopsis

```
void hd66773_setup_end(CTL_GFX_CONTROLLER_t *self);
```

Description

hd66773_setup_end finalizes initialization of the HD66773. After calling this function, the graphics controller is initialized, selected, and ready for use.

<ks0108.h>

Overview

Device driver for a Samsung KS0108 display.

There are simply too many products to mention that contain one of these.

This driver assumes a parallel-bus-connected array of up to three KS0108 controllers acting as a unified display. It's possible to use this driver with a single I2C bus expander to run an emulated 8080-style bus.

API Summary

Context	
KS0108_DRIVER_t	Instance data
KS0108	
ks0108_setup_begin	Start driver setup
AM320240LDTNQW	
ks0108_setup_end	Complete driver setup

KS0108_DRIVER_t

Synopsis

```
typedef struct {
  CTL_GFX_DRIVER_t core;
  void (*set_controls)(KS0108_DRIVER_s *, unsigned);
  int (*controller_selects)(KS0108_DRIVER_s *, unsigned);
  unsigned short __controls;
  unsigned char __shadow[];
  volatile unsigned __dirty;
  int __controllers;
} KS0108_DRIVER_t;
```

Description

KS0108_DRIVER_t contains the instance data for the KS0108 graphics driver.

core

The core graphics driver.

set_controls

Method to set the control state of the emulated bus.

controller_selects

Method to get the chip select states for logical display #*index*. Displays are numbered left to right with increasing index, with display

__controls

Private member containing the current control signal state.

__shadow

Private member containing the current bitmapped display state.

__dirty

Private member indicating when the shadow contents differ from the display contents (and therefore require flushing to the display). One bit per controller.

__controllers

Private member indicating the number of KS0108 display controllers in the display module. Up to three KS0108 controllers can be ganged together to provide a 192×64 display.

ks0108_setup_begin

Synopsis

```
void ks0108_setup_begin(KS0108_DRIVER_t *self,
                        int controller_count);
```

Description

ks0108_setup_begin start the set up the driver **self** as an array of **controller_count** KS0108 controllers that make up the display module. The number of controllers specified by **controller_count** must be be in the range 1 to 3.

After initialization, the client is responsible for setting up the methods `set_controls` and `controller_selects` before calling `ks0108_setup_end`.

ks0108_setup_end

Synopsis

```
void ks0108_setup_end(KS0108_DRIVER_t *self);
```

Description

ks0108_setup_end completes initialization of the panel after the methods `set_controls` and `controller_selects` have been set.

<pcd8544.h>

Overview

Device driver for a Philips PCD8544-based 48x84x1 display.

This is mainly the Nokia 3310 and 5510 displays you commonly find around the net.

API Summary

Types	
PCD8544_DRIVER_t	PCD8544 device driver class
Functions	
pcd8544_power_up	Power up display
pcd8544_set_bias	Write the PCD8544 bias setting
pcd8544_set_polarity	Write the PCD8544 polarity
pcd8544_setup_begin_spi	Start initialization of PCD8544 display
pcd8544_setup_end	Finish initialization of PCD8544 display

PCD8544_DRIVER_t

Synopsis

```
typedef struct {
  CTL_GFX_DRIVER_t core;
  CTL_SPI_DEVICE_t *dev;
  void (*set_dc_state)(int);
  void (*set_reset_state)(int);
  int needs_flush;
  unsigned char frame_buffer[];
} PCD8544_DRIVER_t;
```

Description

PCD8544_DRIVER_t is the class for driving the PCD8544 over an SPI bus.

Structure

`core`

The abstract graphics driver base class.

`dev`

The SPI device associated with this controller.

`set_dc_state`

Method to set the state of the D/C# signal for 8-bit SPI mode.

`set_reset_state`

Method to set the state of the RESET signal.

`frame_buffer`

The frame buffer maintained internally by the class. Because pixels are not individually addressable using the PCD8544 command set, this maintains the state of the display.

`needs_flush`

Indicates whether the internal frame buffer and LCD display differ, which is an indication that it is worthwhile flushing the display.

pcd8544_power_up

Synopsis

```
void pcd8544_power_up(PCD8544_DRIVER_t *self,
                      int start_line);
```

Description

pcd8544_power_up powers up the display. The parameter **start_line** is required for some compatible controllers to define the internal LCD start line; clients can specify this as zero for a standard controller and LCD.

pcd8544_set_bias

Synopsis

```
void pcd8544_set_bias(PCD8544_DRIVER_t *self,
                      int bias);
```

Description

pcd8544_set_bias sets the Bias bits (BS[0:2]) of the display **self** to **bias**. Please refer to the PCD8544 datasheet for an exact description of the bias bit encoding.

pcd8544_set_polarity

Synopsis

```
void pcd8544_set_polarity(PCD8544_DRIVER_t *self,
                          int invert);
```

Description

pcd8544_set_polarity sets the polarity of the display **self** to normal or inverted depending upon **invert**. If **invert** is non-zero, the display is inverted at the display controller level.

You can use this to flash the display.

pcd8544_setup_begin_spi

Synopsis

```
void pcd8544_setup_begin_spi(PCD8544_DRIVER_t *self,
                             CTL_SPI_DEVICE_t *dev);
```

Description

pcd8544_setup_begin_spi starts the initialization of the PCD8544 display **self** using SPI device **dev**. The client is responsible for initializing the **set_dc_state** and **set_reset_state** methods and attaching **dev** to the appropriate SPI bus before calling **pcd8544_setup_end**.

The SPI device is initialized to communicate at 1 MHz. You can raise the bit rate when this function returns (up to 4 MHz for the standard controller).

By default, the PCD8544 is initialized to positive polarity for a monochrome display where the background is initialized to white and writing a one bit will turn a pixel to black.

pcd8544_setup_end

Synopsis

```
void pcd8544_setup_end(PCD8544_DRIVER_t *self);
```

Description

pcd8544_setup_end completes the initialization of the PCD8544 **self**.

<pcf8833.h>

Overview

Device driver for a Philips PCF8833.

This is typically used in Nokia 6110 displays, There are lots of 6110 clone displays knocking about the net. If you get an LCD from Olimex with a "GE12" sticker on it, then your display uses the PCF8833 controller and if it has a "GE8" sticker on it then it uses the S1D15G00 controller.

API Summary

Setup	
pcf8833_setup_begin_bus	Start driver initialization (bus)
pcf8833_setup_begin_spi	Start driver initialization (SPI)
pcf8833_setup_end	Complete driver initialization

pcf8833_setup_begin_bus

Synopsis

```
void pcf8833_setup_begin_bus(CTL_GFX_CONTROLLER_t *self,
                             int depth);
```

Description

pcf8833_setup_begin_bus starts initialization of the PCF8833.

pcf8833_setup_begin_spi

Synopsis

```
void pcf8833_setup_begin_spi(CTL_GFX_CONTROLLER_t *self,
                             CTL_SPI_DEVICE_t *dev,
                             int depth);
```

Description

pcf8833_setup_begin_spi starts initialization of the PCF8833. The device **dev** is configured for 9-bit SPI mode at 6 MHz.

pcf8833_setup_end

Synopsis

```
void pcf8833_setup_end(CTL_GFX_CONTROLLER_t *self);
```

Description

pcf8833_setup_end finalizes initialization of the PCF8833. After calling this function, the graphics controller is initialized, selected, and ready for use.

<s6d1121.h>

Overview

Device driver for a Samsung S6D1121.

The ITDB02-2.4E has one of these.

API Summary

Setup	
s6d1121_setup_begin	Start driver initialization
s6d1121_setup_end	Complete driver initialization

s6d1121_setup_begin

Synopsis

```
void s6d1121_setup_begin(CTL_GFX_CONTROLLER_t *self);
```

Description

s6d1121_setup_begin starts initialization of the S6D1121. After calling this function, you must fill in both the `write_register` and `write_pixel` methods, power-on the display, and initialize it.

s6d1121_setup_end

Synopsis

```
void s6d1121_setup_end(CTL_GFX_CONTROLLER_t *self);
```

Description

s6d1121_setup_end finalizes initialization of the S6D1121. After calling this function, the graphics controller is initialized, selected, and ready for use.