OLED Graphics Library User Manual

Sean Kent

July 6, 2020

Contents

1 Overview

The OLED graphics library provides a collection of functions for interacting with the display, including drawing lines and shapes as well as displaying bitmap images and text. This user manual steps through the process of setting up the OLED and PSoC and describes the basic structure of the library.

2 Hardware Setup

2.1 OLED Schematic

The OLED and PSoC communicate with eachother via an I^2C interface. The PSoC, which is configured as the master device, sends commands and data to the OLED, which is a slave device (one of possibly many connected the PSoC). Figure 1 below shows the circuit the schematic for the OLED module. It's I^2C pins, **SCL** and **SDA**, should be connected to their counterparts on PSoC. The **GND** pin should be tied to ground and the $\mathbf{V_{cc}}$ can be powered off of 3-5 V.

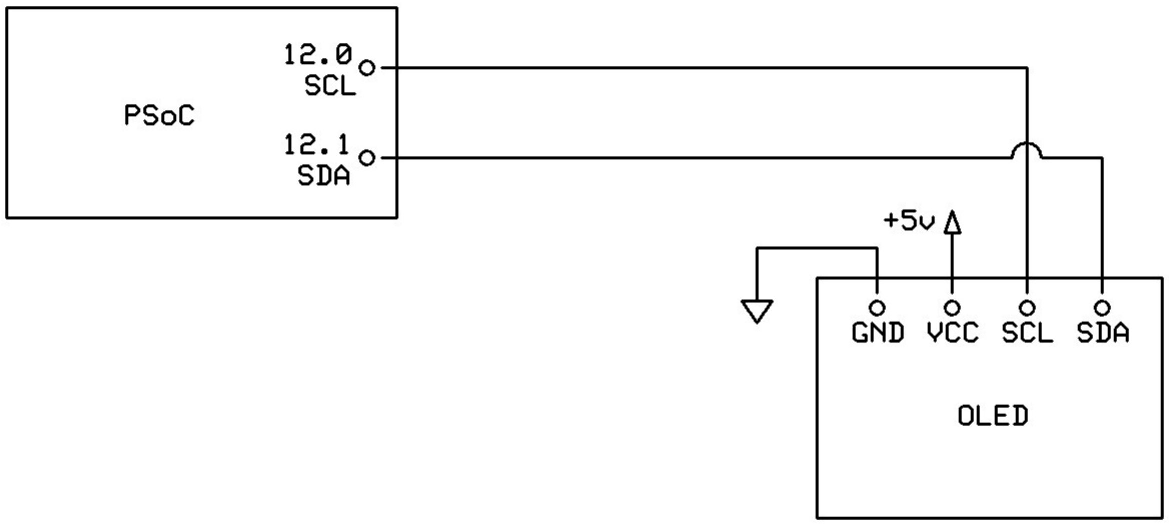


Figure 1: OLED Module Schematic

2.2 PSoC Hardware Setup

Figure 2 shows the hardware setup for the PSoC 5LP Prototyping Kit. The boxed, red pins are used in the example projects. Everything else can be left unconnected or used for other functions.

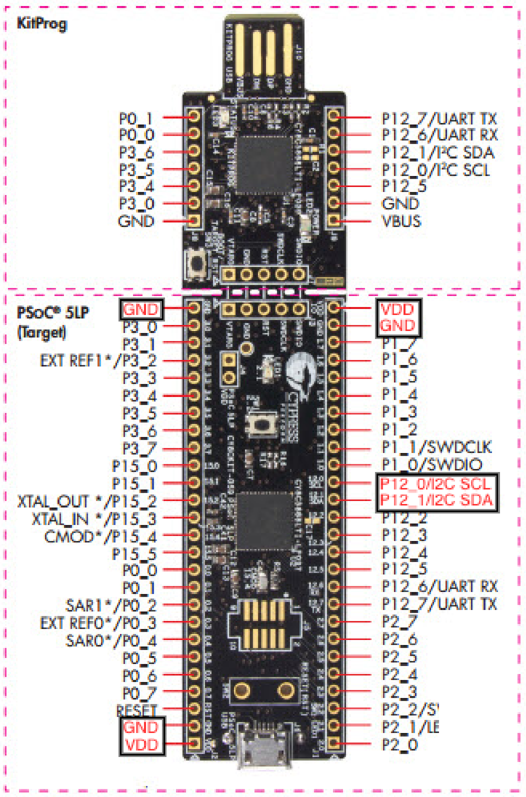


Figure 2: PSoC Hardware Setup

3 Creator Setup

3.1 Cypress Schematic

In order to communicate with the OLED, the PSoC needs to be set up as an I^{2}C master. To do this, click on the TopDesign.cysch file and drag and drop an I2C Master (Fixed Function [v3.50]) component into the schematic. Make sure the component's mode is set to Master. For the fastest performance, set the data rate to its maximum value of 1000 kbps. The I^{2}C Master component and its configuration are shown in the figures below.

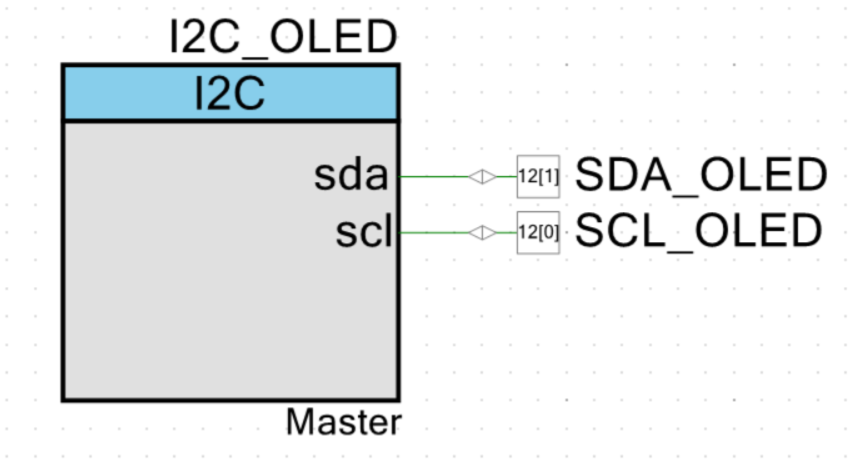
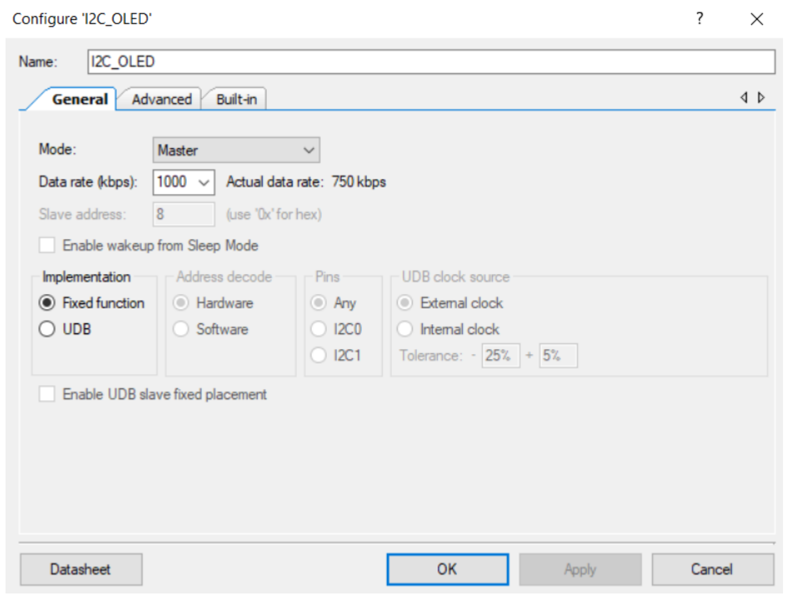


Figure 3: I^{2}C Master Component



Configure 'I2C_OLED'

Name: I2C_OLED

General | Advanced | Built-in

Mode: Master

Data rate (kbps): 1000 Actual data rate: 750 kbps

Slave address: 8 (use '0x' for hex)

Enable wakeup from Sleep Mode [ ]

Implementation: [✓ Fixed function] UDB

Address decode: [✓ Hardware] Software

Pins: [✓ Any] I2C0 I2C1

UDB clock source: [✓ External clock] Internal clock

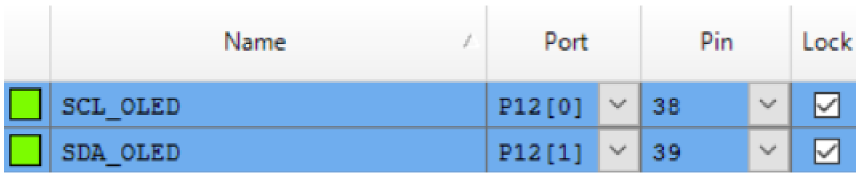
Tolerance: - 25% + 5%

Enable UDB slave fixed placement [ ]

Figure 4: I^{2}C Master Component Configuration

3.2 Design Wide Resources

The I^{2}C component shown above has two output signals: **SCL**, the clock signal, and **SDA**, the data signal. These outputs must be routed to the correct pins of the PSoC. In the Design Wide Resources drop down, click on Pins and select ports **P12[0]** and **P12[1]** for the I^{2}C's **SCL** and **SDA** outputs respectively.



	Name	Port	Pin	Lock
🟩	SCL_OLED	P12[0]	38	✓
🟩	SDA_OLED	P12[1]	39	✓

Figure 5: Pin Selection

3.3 Adding the Graphic Library

Once the OLED and PSoC are wired up and the PSoC is configured for I^{2}C communication, it is time to add the graphics library. To included the library in an existing project, two C source files and two C header files must be added to the project. These files are **oled.c**, **oled.h**, **font.c**, and **font.h** and can be found on the course website. To add the files to your project, copy and paste the four files into your project's .cydsn folder. Then, in PSoC Creator, right click on your project's name in the Workspace Explorer menu and navigate to **Add** → **Existing Item...**. Select the four files you have just copied into the into the .cydsn folder and add them to your project. They should now appear in the Workspace Explorer menu under the **Header Files** and **Source Files** drop down tabs respectively.

4 The oled_t Struct

Central to the function of the graphics library is the `oled_t` struct. It is responsible for storing the current state of the display, library settings (e.g. pen size), and the information and functions required to communicate with the display. For your reference, the `oled_t` definition has been copied below.

If you take a look at the struct definition at the bottom of this section, you will find the first four member definitions included in the display's 7-bit I^{2}C address: `slaveAddr`, followed by three function pointers, `SendStart`, `SendStop`, and `WriteByte` which together allow the library to send data to the display. The three function pointers point at functions provided in the I^{2}C Master component's API, `I2C_MasterSendStart()`, `I2C_MasterSendStop()`, and `I2C_MasterWriteByte()` respectively.

Following the slave address and function pointers are a number of setting variables, including the foreground and background colors, pen size, font, and display modes for bitmap and text functions. To change a setting, it is recommended to use the provided function (e.g. `oled_SetPenSize()`) rather than change the variables directly. For more information on these setting, check out the the Drawing, Bitmap, and Text sections.

The final member of the structure is a 512 ($64 * 8$) byte array which keeps track of the pixel values written to the display's GRAM (Graphic RAM). The reason this array exists is because the display does not support writing to individual pixels. Instead, an 8-bit column of pixels is filled when a byte of graphic data is written to the display, with the LSb of the byte corresponding to the top pixel of the column. To allow the user to change individual pixels, the GRAM array is used to help create the byte of graphic data being sent such that only the pixels that are supposed to be changed are affected by the write. As a user of library you should not have to touch the GRAM array, it is updated automatically by the library functions.

If you plan to use more than one display in your project, each display should have its own instance of an `oled_t` struct. In this way, state and setting information are kept local to each display.

```
//======================================
// oled struct
//======================================
typedef struct {
    uint8 slaveAddr;                                // oled I2C address
    uint8 (*SendStart)(uint8 slaveAddr, uint8 R_nW);  // function pointer to
        I2C_MasterSendStart(). Generates an I2C Start condition and sends the
        slave address with the read/write bit
    uint8 (*SendStop)(void);                        // function pointer to
        I2C_MasterSendStop(). Generates and I2C Stop condition
    uint8 (*WriteByte)(uint8 byte);                 // function pointer to
        I2C_MasterWriteByte(). Send one byte via I2C
    uint8 bkColor;                                  // background color
    uint8 color;                                    // foreground color
    uint8 penSize;                                  // pen size for drawing
        functions
    const font_t * font;                            // text font
    uint8 bmMode;                                   // bitmap mode
    uint8 textMode;                                 // text mode (i.e.
        transparent, fill)
    uint8 GRAM[OLED_WIDTH*(OLED_HEIGHT/8)];         // "internal copy" of oled
        graphic RAM (GRAM)
} oled_t;
```

5 Initialization

Before any of the library functions can be used, an instance of the `oled_t` struct must be created and the initialization function `oled_Init()` must be called.

The initialization function serves two main purposes: to setup the newly created `oled_t` instance (by assigning values to the member variables) and send a series of commands to initialize the display. A pointer to the `oled_t` instance, the slave address of the display, and the three function pointers for sending data over I^{2}C (via an I^{2}C Master component) are passed in as parameters to this function. Note that the function pointers are specific to the I^{2}C Master component connected to the display. For example, if the I^{2}C Master component is named "`I2C_OLED`", the pointers passed in would be `I2C_OLED_MasterSendStart`, `I2C_OLED_MasterSendStop`, and `I2C_OLED_MasterWriteByte`. The remaining members of the struct are set to default values.

A typical initialization sequence may look as follows. Make sure to start the I^{2}C component before calling `oled_Init()`.

```
//======================================
// include
//======================================
#include "oled.h"

//======================================
// main
//======================================
int main(void)
{
    CyGlobalIntEnable;                // enable global interrupts
    I2C_OLED_Start();                 // initialize I2C_OLED
    CyDelay(100);                     // delay while OLED powers on
    oled_t oled;                      // create oled_t instance
    oled_Init(&oled, 0x3c, I2C_OLED_MasterSendStart, I2C_OLED_MasterSendStop,
        I2C_OLED_MasterWriteByte);   // initialize display

    for(;;){
        // put application code here ...
    }
}
```

6 Drawing Features

One of the central features of this graphics library is its drawing API, which contains functions for drawing lines and other basic shapes such as rectangles and circles. A full list of the drawing functions can be found in the API section.

The general operation of these functions should be relatively self-explanatory. For instance, the function

```
oled_DrawLine(&oled, 32, 16, 96, 48);
```

draws a line between the points (32, 16) and (96, 48). The first parameter of the function, `&oled`, is a pointer to an `oled_t` instance and specifies which display the function is for. What is less obvious from the function name and parameters is how to choose the color of the line and its thickness. These parameters are actually set in the display's `oled_t` struct and apply to all drawing functions.

6.1 Choosing a Color

The color with which lines and shapes are drawn is determined by the value of the foreground color variable, `oled.color`, in the `oled_t` struct. The display itself has two possible pixel values, black and white, which can be selected by setting `oled.color = 0` or `oled.color = 1` respectively. There is an exception to this rule when using functions with the word "`Clear`" in their name. These functions, such as `oled_Clear()` and `oled_ClearRect()`, use the background color, `oled.bkColor`, when drawing pixels on the display.

When changing either the foreground color or background color, it is recommended you use the provided functions, `oled_SetColor()` and `oled_SetBkColor()`, rather than reassigning the member variables directly.

6.2 Line Thickness

The thickness of a line is controlled by the `oled.penSize` variable. The pen size can take on any integer value in the range 0 to 127, with a pen size of 0 drawing the thinnest lines (1 pixel thick). However, it is not recommended to use excessively large pen sizes, as it will likely result in the noticeably slower operation of some drawing functions. When using functions with "`Fill`" in their name, pen size is not used.

6.3 Drawing Example

The example below shows how to draw a circle with a 20 pixel radius in the center of the display. In this example, the background color is black and the foreground color is white. The thickness of the circle is 3 pixels thick, which corresponds to a pen size of 1.

```
//========================================
// include
//========================================
#include "oled.h"

//========================================
// main
//========================================
int main(void)
{
    CyGlobalIntEnable;                      // enable global interrupts
    I2C_OLED_Start();                       // initialize I2C_OLED
    CyDelay(100);                           // delay while OLED powers on
    oled_t oled;                            // create oled_t instance
    oled_Init(&oled, 0x3c, I2C_OLED_MasterSendStart, I2C_OLED_MasterSendStop,
        I2C_OLED_MasterWriteByte);          // initialize display

    oled_Clear(&oled);                      // clear the display
    oled_SetColor(&oled, 1);                // set the foreground color
    oled_SetBkColor(&oled, 0);              // set the background color
    oled_SetPenSize(&oled, 1);              // set the pen size
    oled_DrawCircle(&oled, 64, 32, 20); // draw a circle of radius 20 centered at
        (64, 32)

    for(;;){}
}
```

7 Displaying Bitmap

Another main feature of the library is displaying bitmap images. A bitmap image is a array of pixel values which form an image when displayed on a screen. In the specific case of our OLED display, each value can take on either a 0 or a 1, which correspond to a black or white pixel respectively.

7.1 .xbm Files

The graphics library expects bitmaps to be in a .xbm file format. Unlike most image files, .xbm files are stored as C source files. They can therefore be added and included as separate files in your project or you can copy and paste their contents into an existing file in your project.

At the top of a .xbm file, there are two `#define` statements that define the width and height of the bitmap. Following these preprocessor statements is the image data in the form of a 1-D array of unsigned chars. Each pixel value is represented by a single bit, thus each char contains 8 pixel values. Although the array is 1-D, it represents a 2-D grid of pixels. The top left pixel of this grid corresponds to the LSb of the first char in the array and extends right LSb to MSb until the width of the grid has been reached. If the width is not a multiple of 8, the extra bits in the last char are ignored so that the next row starts on the LSb of the next char.

A 8x8 checkerboard bitmap and its corresponding .xbm file are shown below.

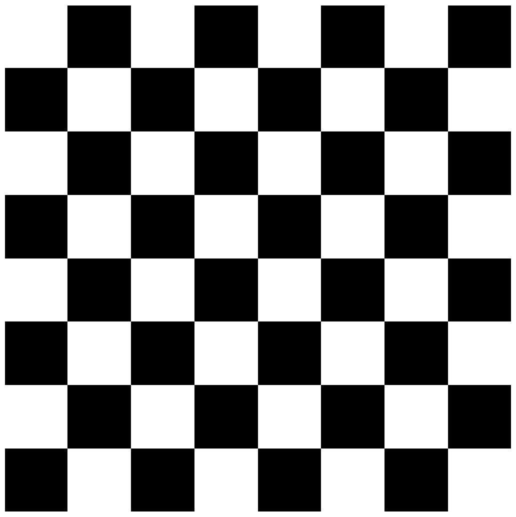


Figure 6: Checkerboard Bitmap

```
//=======================================
// checkerboard .xbm file (saved as checkerboard.h)
//=======================================
#define checkerboard_width 8
#define checkerboard_height 8
static unsigned char checkerboard_bits[] = {
  0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa };
```

If you are using an application such as GIMP to create a bitmap image, you may notice that when you export the image as a .xbm file the black pixel get saved as 1's and the white pixels get saved as 0's. For instance, the image of the checkerboad above would be saved as

```
static unsigned char checkerboard_bits[] = {
  0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55 };
```

instead of as

```
static unsigned char checkerboard_bits[] = {
  0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa };
```

7.2 Bitmap Modes

When displaying bitmaps, there are several modes to choose from which determine how the bitmap is displayed. The mode can be changed using the function `oled_SetBmMode()`. The available modes are:

1. `OLED_BM_NORMAL`: The bitmap is displayed as specified in the image data array.
2. `OLED_BM_INV`: The bitmap displayed with pixels values inverted.
3. `OLED_BM_WHITE_TRAN`: White pixels in the bitmap are displayed in the current foreground color. Black pixels in the bitmap are left transparent (the current pixel values at these locations are not overwritten when the bitmap is displayed).
4. `OLED_BM_BLACK_TRAN`: Black pixels in the bitmap are displayed in the current foreground color. White pixels in the bitmap are left transparent (the current pixel values at these locations are not overwritten when the bitmap is displayed).
5. `OLED_BM_WHITE_FILL`: White pixels in the bitmap are displayed in the foreground color. Black pixels are displayed in the background color.
6. `OLED_BM_BLACK_FILL`: Black pixels in the bitmap are displayed in the foreground color. White pixels are displayed in the background color.

7.3 Bitmap Example

The following example code displays the checkerboard bitmap with its top left corner located at (0, 0). In this example, the image is included as a C header file called **checkerboard.h**.

```
//=======================================
// include
//=======================================
#include "oled.h"
#include "checkerboard.h"

//=======================================
// main
//========================================
int main(void)
{
    CyGlobalIntEnable;                        // enable global interrupts
    I2C_OLED_Start();                         // initialize I2C_OLED
    CyDelay(100);                             // delay while OLED powers on
    oled_t oled;                              // create oled_t instance
    oled_Init(&oled, 0x3c, I2C_OLED_MasterSendStart, I2C_OLED_MasterSendStop,
        I2C_OLED_MasterWriteByte);            // initialize display

    oled_Clear(&oled);                        // clear the display
    oled_SetBmMode(&oled, OLED_BM_NORMAL);    // set bitmap mode
    oled_DispBitmap(&oled, 0, 0, checkerboard_bits, checkerboard_width,
        checkerboard_height);                 // display bitmap

    for(;;){}
}
```

8 Displaying Text

The graphics library also includes basic features for displaying text. In many ways, it is simply an extension of of the bitmap display features, as text characters are themselves bitmaps.

The library includes functions for displaying individual characters as well as strings of text. The font used for the characters is determined by the `oled.font` setting in the `oled_t` struct and can be changed using the function `oled_SetFont()`.

8.1 The font_t Struct

Information about a font, including its all of its bitmap characters, is contained in a `font_t` struct, defined below.

```
//========================================
// font struct
//========================================
typedef struct {
    const uint8 width;              // width of a character (in pixels)
    const uint8 height;             // height of a character (in pixels)
    const uint8 characters[95][32]; // 2-D array of ASCII characters
} font_t;
```

The struct is comprised of three members. The first two, `width` and `height`, define the width and height of a single character bitmap. Each character bitmap in a given font must have the same width and height. The third member, `characters`, is a 2-D array of character bitmaps. The array is designed to contain the block of 95 ASCII characters starting with the Space character and ending with the tilde character.

The second dimension in the `character` array definition (32 in the example above) is based on the number of bytes it takes to store each character's bitmap. The default font uses a 12x16 bitmap for the characters, which requires 32 bytes to store. If you create a font which needs more bytes, the second dimension must be increased to match the maximum number of bytes needed.

8.2 Font Modes

Similar to bitmaps, there are several modes to choose from which displaying text. The mode can be changed using the function `oled_SetTextMode()`. The available modes are:

1. `OLED_TEXT_TRAN`: The characters (letters, numbers, etc.) are displayed in the foreground color. The surrounding pixels are transparent.

2. `OLED_TEXT_FILL`: The characters (letters, numbers, etc.) are displayed in the foreground color. The surrounding pixels are filled with the background color.

8.3 Font Example

The example below displays the string "Hello" with the top left corner of the first character at (0, 0).

```
//======================================
// include
//======================================
#include "oled.h"

//======================================
// main
//======================================
int main(void)
{
   CyGlobalIntEnable;                           // enable global interrupts
   I2C_OLED_Start();                            // initialize I2C_OLED
   CyDelay(100);                                // delay while OLED powers
      on
   oled_t oled;                                 // create oled_t instance
   oled_Init(&oled, 0x3c, I2C_OLED_MasterSendStart, I2C_OLED_MasterSendStop,
      I2C_OLED_MasterWriteByte);                // initialize display

   oled_Clear(&oled);                           // clear the display
   oled_SetTextMode(&oled, OLED_TEXT_TRAN);     // set text mode
   oled_DispString(&oled, 0, 0, "Hello");       // display "Hello"

   for(;;){}
}
```

9 Graphics Library API

9.1 Initialization API

9.1.1 oled_Init()

Description

This function initializes the `oled_t` struct and sends the necessary initialization commands to the display.

Prototype

```
void oled_Init(oled_t * oled,
               uint8 slaveAddr,
               uint8 (*SendStart)(uint8, uint8),
               uint8 (*SendStop)(void),
               uint8 (*WriteByte)(uint8));
```

Parameters

`oled`	Pointer to an `oled_t` struct
`slaveAddr`	7-bit I2C address
`SendStart`	Function pointer to `I2C_MasterSendStart()`
`SendStop`	Function pointer to `I2C_MasterSendStop()`
`WriteByte`	Function pointer to `I2C_MasterWriteByte ()`

9.2 Color API

9.2.1 oled_SetColor()

Description

This function sets the foreground color.

Prototype

```
void oled_SetColor(oled_t * oled,
                   uint8 color);
```

Parameters

`oled`	Pointer to an `oled_t` struct
`color`	Foreground color (0 or 1)

9.2.2 oled_GetColor()

Description

This function returns the current foreground color.

Prototype

```
uint8 oled_GetColor(oled_t * oled);
```

Parameters

`oled`	Pointer to an `oled_t` struct

9.2.3 oled_SetBkColor()

Description
This function sets the background color.

Prototype

```
void oled_SetBkColor(oled_t * oled,
                     uint8 bkColor);
```

Parameters

`oled`	Pointer to an `oled_t` struct
`bkColor`	Background color (0 or 1)

9.2.4 oled_GetBkColor()

Description
This function returns the current background color.

Prototype

```
uint8 oled_GetBkColor(oled_t * oled);
```

Parameters

`oled`	Pointer to an `oled_t` struct

9.2.5 oled_SetPenSize()

Description
This function sets the pen size for drawing operations. A pen size of 0 draws the thinnest lines (1 pixel thick).

Prototype

```
void oled_SetPenSize(oled_t * oled,
                     uint8 penSize);
```

Parameters

`oled`	Pointer to an `oled_t` struct
`penSize`	Pen size

9.2.6 oled_GetPenSize()

Description

This function returns the current pen size.

Prototype

```
uint8 oled_GetPenSize(oled_t * oled);
```

Parameters

oled	Pointer to an oled_t struct

9.2.7 oled_Clear()

Description

This functions clears the screen (fills it with the background color).

Prototype

```
void oled_Clear(oled_t * oled);
```

Parameters

oled	Pointer to an oled_t struct

9.2.8 oled_ClearRect()

Description

This functions clears a rectangle defined by its upper left corner (x0, y0) and lower right corner (x1, y1).

Prototype

```
void oled_ClearRect(oled_t * oled,
                    int x0,
                    int y0,
                    int x1,
                    int y1);
```

Parameters

oled	Pointer to an oled_t struct
x0	Upper left x-coordinate
y0	Upper left y-coordinate
x1	Lower right x-coordinate
y1	Lower right y-coordinate

9.2.9 oled_DrawPixel()

Description

This function fills the pixel located at (x, y) with the forground color.

Prototype

```
void oled_DrawPixel(oled_t * oled,
                    int x,
                    int y);
```

Parameters

`oled`	Pointer to an `oled_t` struct
`x`	x-coordinate of pixel
`y`	y-coordinate of pixel

9.2.10 oled_DrawLine()

Description

This functions draws a line between the points (x0, y0) and (x1, y1).

Prototype

```
void oled_DrawLine(oled_t * oled,
                   int x0,
                   int y0,
                   int x1,
                   int y1);
```

Parameters

`oled`	Pointer to an `oled_t` struct
`x0`	x-coordinate of the first point
`y0`	y-coordinate of the first point
`x1`	x-coordinate of the second point
`y1`	y-coordinate of the second point

9.2.11 oled_DrawRect()

Description

This functions draws a rectangle defined by its upper left corner (x0, y0) and lower right corner (x1, y1).

Prototype

```
void oled_DrawRect(oled_t * oled,
                   int x0,
                   int y0,
                   int x1,
                   int y1);
```

Parameters

oled	Pointer to an oled_t struct
x0	Upper left x-coordinate
y0	Upper left y-coordinate
x1	Lower right x-coordinate
y1	Lower right y-coordinate

9.2.12 oled_DrawRoundedRect()

Description
This functions draws a rectangle defined by its upper left corner (x0, y0) and lower right corner (x1, y1). The corners of the rectangle are rounded with radius r.

Prototype

```
void oled_FillRect(oled_t * oled,
                   int x0,
                   int y0,
                   int x1,
                   int y1,
                   int r);
```

Parameters

oled	Pointer to an oled_t struct
x0	Upper left x-coordinate
y0	Upper left y-coordinate
x1	Lower right x-coordinate
y1	Lower right y-coordinate
r	Radius of rounded corners (must be less than or equal to half the smallest side length)

9.2.13 oled_FillRect()

Description
This functions fills a rectangle defined by its upper left corner (x0, y0) and lower right corner (x1, y1).

Prototype

```
void oled_FillRect(oled_t * oled,
                   int x0,
                   int y0,
                   int x1,
                   int y1);
```

Parameters

`oled`	Pointer to an `oled_t` struct
`x0`	Upper left x-coordinate
`y0`	Upper left y-coordinate
`x1`	Lower right x-coordinate
`y1`	Lower right y-coordinate

9.2.14 oled_FillRoundedRect()

Description
This functions fills a rectangle defined by its upper left corner (x0, y0) and lower right corner (x1, y1). The corners of the rectangle are rounded with radius r.

Prototype

```
void oled_FillRect(oled_t * oled,
                   int x0,
                   int y0,
                   int x1,
                   int y1,
                   int r);
```

Parameters

`oled`	Pointer to an `oled_t` struct
`x0`	Upper left x-coordinate
`y0`	Upper left y-coordinate
`x1`	Lower right x-coordinate
`y1`	Lower right y-coordinate
`r`	Radius of rounded corners (must be less than or equal to half the smallest side length)

9.2.15 oled_DrawCircle()

Description
This functions draws a circle of radius r centered at (xc, yc).

Prototype

```
void oled_DrawCircle(oled_t * oled,
                     int xc,
                     int yc,
                     int r);
```

Parameters

`oled`	Pointer to an `oled_t` struct
`xc`	x-coordinate of the center of the circle
`yc`	y-coordinate of the center of the circle
`r`	Radius of circle

9.2.16 oled_DrawArc()

Description

This function draws an arc of radius r centered at (xc, yc). a0 and a1 specify the start and end angles for the arc.

Prototype

```
void oled_DrawArc(oled_t * oled,
                  int xc,
                  int yc,
                  int r,
                  int a0,
                  int a1);
```

Parameters

oled	Pointer to an oled_t struct
xc	x-coordinate of the center of the arc (center of circle)
yc	y-coordinate of the center of the arc (center of circle)
r	Radius of arc
a0	Start angle (angle = $a0*\pi/4$)
a1	End angle (angle = $a1*\pi/4$)

9.2.17 oled_DrawPoint()

Description

This function fills in a circle centered at (xc, yc). The radius of the circle is the current pen size.

Prototype

```
void oled_DrawCircle(oled_t * oled,
                     int xc,
                     int yc);
```

Parameters

oled	Pointer to an oled_t struct
xc	x-coordinate of point
yc	y-coordinate of point

9.2.18 oled_FillCircle()

Description

This function fills in a circle with radius r centered at (xc, yc).

Prototype

```
void oled_FillCircle(oled_t * oled,
                     int xc,
                     int yc,
                     int r);
```

Parameters

oled	Pointer to an oled_t struct
xc	x-coordinate of the center of the circle
yc	y-coordinate of the center of the circle
r	Radius of circle

9.2.19 oled_FillPie()

Description

This function fills a circular sector of radius r centered at (xc, yc). a0 and a1 specify the start and end angles for the sector.

Prototype

```
void oled_FillPie(oled_t * oled,
                  int xc,
                  int yc,
                  int r,
                  int a0,
                  int a1);
```

Parameters

oled	Pointer to an oled_t struct
xc	x-coordinate of the center of the circle
yc	y-coordinate of the center of the circle
r	Radius of circular sector
a0	Start angle (angle = $a0*\pi/4$)
a1	End angle (angle = $a1*\pi/4$)

9.3 Bitmap API

9.3.1 oled_SetBmMode()

Description

This function sets the bitmap display mode.

Prototype

```
void oled_SetBmMode(oled_t * oled,
                    uint8 bmMode);
```

Parameters

oled	Pointer to an oled_t struct
bmMode	Bitmap display mode

9.3.2 oled_GetBmMode()

Description

This function returns the current bitmap display mode.

Prototype

```
uint8 oled_GetBmMode(oled_t * oled);
```

Parameters

`oled`	Pointer to an `oled_t` struct

9.3.3 oled_DispBitmap()

Description

This function displays the given bitmap with its top left corner at (x0, y0).

Prototype

```
void oled_DispBitmap(oled_t * oled,
                     int x0,
                     int y0,
                     const uint8 * bitmap,
                     uint16 width,
                     uint16 height);
```

Parameters

`oled`	Pointer to an `oled_t` struct
`x0`	x-coordinate for upper left corner of bitmap
`y0`	y-coordinate for upper left corner of bitmap
`bitmap`	Pointer to an array containing pixel values (.xbm format)
`width`	Width of the bitmap (in pixels)
`height`	Height of the bitmap (in pixels)

9.4 Text API

9.4.1 oled_SetFont()

Description

This function sets the font for text displaying operations.

Prototype

```
void oled_SetFont(oled_t * oled,
                  font_t * font);
```

Parameters

`oled`	Pointer to an `oled_t` struct
`font`	Pointer to a `font_t` struct

9.4.2 oled_SetTextMode()

Description
This function sets the text display mode.

Prototype

```
void oled_SetTextMode(oled_t * oled,
                      uint8 textMode);
```

Parameters

`oled`	Pointer to an `oled_t` struct
`textMode`	Text display mode

9.4.3 oled_GetTextMode()

Description
This function returns the current text display mode.

Prototype

```
uint8 oled_GetTextMode(oled_t * oled);
```

Parameters

`oled`	Pointer to an `oled_t` struct

9.4.4 oled_DispChar()

Description
This function displays the character c with its top left corner at (x0, y0). The character is displayed in the font specified by oled.font.

Prototype

```
void oled_DispChar(oled_t * oled,
                   int x0,
                   int y0,
                   const char c);
```

Parameters

`oled`	Pointer to an `oled_t` struct
`x0`	x-coordinate for upper left corner of bitmap
`y0`	y-coordinate for upper left corner of bitmap
`c`	Character to be displayed

9.4.5 oled_DispString()

Description

This function displays the string s with the top left corner of the first character at (x0, y0). The characters are displayed in the font specified by oled.font.

Prototype

```
void oled_DispString(oled_t * oled,
                     int x0,
                     int y0,
                     const char * s);
```

Parameters

`oled`	Pointer to an `oled_t` struct
`x0`	x-coordinate for upper left corner of bitmap
`y0`	y-coordinate for upper left corner of bitmap
`s`	Pointer to the string