OLED_I2C

Arduino and chipKit library for I^{2}C 128x64 pixel SSD1306 OLEDs

Manual

Introduction:

This library has been made to make it easy to use 128x64 pixel OLED displays based on the SSD1306 controller chip with an Arduino or a chipKit.

This library will default to I^{2}C Fast Mode (400 KHz) when using the hardware I^{2}C interface.

The library has not been tested in combination with the Wire library and I have no idea if they can share pins. Do not send me any questions about this. If you experience problems with pin-sharing you can move the displays SDA and SCL pins to any available pins on your development board. This library will in this case fall back to a software-based, TWI-/I^{2}C-like protocol which *will* require exclusive access to the pins used.

If you are using a chipKit Uno32 or uC32 and you want to use the hardware I^{2}C interface you must remember to set the JP6 and JP8 jumpers to the I^{2}C position (closest to the analog pins).

You can always find the latest version of the library at **http://www.RinkyDinkElectronics.com/**

For version information, please refer to **version.txt**.

Defined Literals:

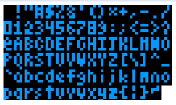
Alignment

For use with print(), printNumI() and printNumF()

LEFT:	0
RIGHT:	9999
CENTER:	9998

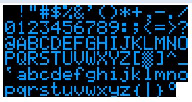
Included Fonts:

TinyFont



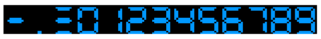
Charactersize:	4x6 pixels
Number of characters:	95

SmallFont



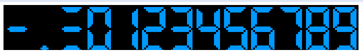
Charactersize:	6x8 pixels
Number of characters:	95

MediumNumbers



Charactersize:	12x16 pixels
Number of characters:	13

BigNumbers



Charactersize:	14x24 pixels
Number of characters:	13

Functions:

OLED(Data, Clock, [Reset]);	
The main class constructor.	

Parameters:	`Data:   Pin for Data transfer`
	`Clock:  Pin for Clock signal`
	`Reset:  Pin for Reset` **`<optional>`**

Usage:	`OLED myOLED(SDA, SCL); // Start an instance of the OLED class with the hardware TWI/I`^{2}`C and no reset`

begin();	
Initialize the display.	
Parameters:	`None`
Usage:	`myOLED.begin(); // Initialize the display`
Notes:	`This will reset (if a reset pin is used) and clear the display.`

setBrightness(value);	
Set the brightness of the display.	
Parameters:	`value:  Specify a value to use for brightness (0-255)`
Usage:	`myOLED.setBrightness(207); // Sets the brightness to the default value of 207`
Notes:	`Note that the span of the adjustable brightness is quite small and might not be noticeable`

update();

Copy the screen buffer to the screen.
This is the only command, except invert() and setBrightness(), that will make anything happen on the physical screen. All other commands only modify the screen buffer.

Parameters:	`None`
Usage:	`myOLED.update(); // Copy the screen buffer to the screen`
Notes:	`Remember to call update() after you have updated the screen buffer.`

clrScr();

Clear the screen buffer.

Parameters:	`None`
Usage:	`myOLED.clrScr(); // Clear the screen buffer`

fillScr();

Fill the screen buffer.

Parameters:	`None`
Usage:	`myOLED.fillScr(); // Fill the screen buffer`

invert(mode);

Set inversion of the display on or off.

Parameters:
```
mode: true  - Invert the display
      false – Normal display
```

Usage:
```
myOLED.invert(true); // Set display inversion on
```

setPixel(x, y);

Turn on the specified pixel in the screen buffer.

Parameters:
```
x:  x-coordinate of the pixel
y:  y-coordinate of the pixel
```

Usage:
```
myOLED.setPixel(0, 0); // Turn on the upper left pixel (in the screen buffer)
```

clrPixel(x, y);

Turn off the specified pixel in the screen buffer.

Parameters:
```
x:  x-coordinate of the pixel
y:  y-coordinate of the pixel
```

Usage:
```
myOLED.clrPixel(0, 0); // Turn off the upper left pixel (in the screen buffer)
```

invPixel(x, y);

Invert the state of the specified pixel in the screen buffer.

Parameters:
```
x:  x-coordinate of the pixel
y:  y-coordinate of the pixel
```

Usage:
```
myOLED.invPixel(0, 0); // Invert the upper left pixel (in the screen buffer)
```

print(st, x, y);

Print a string at the specified coordinates in the screen buffer.
You can use the literals LEFT, CENTER and RIGHT as the x-coordinate to align the string on the screen.

Parameters:
```
st:  the string to print
x:   x-coordinate of the upper, left corner of the first character
y:   y-coordinate of the upper, left corner of the first character
```

Usage:
```
myOLED.print("Hello World",CENTER,0); // Print "Hello World" centered at the top of the screen (in
the screen buffer)
```

Notes: The string can be either a char array or a String object

printNumI(num, x, y[, length[, filler]]);

Print an integer number at the specified coordinates in the screen buffer.
You can use the literals LEFT, CENTER and RIGHT as the x-coordinate to align the string on the screen.

Parameters:
```
num: the value to print (-2,147,483,648 to 2,147,483,647) INTEGERS ONLY
x:   x-coordinate of the upper, left corner of the first digit/sign
y:   y-coordinate of the upper, left corner of the first digit/sign
length: <optional>
        minimum number of digits/characters (including sign) to display
filler: <optional>
        filler character to use to get the minimum length. The character will be inserted in front
        of the number, but after the sign. Default is ' ' (space).
```

Usage:
```
myOLED.print(num,CENTER,0); // Print the value of "num" centered at the top of the screen (in the
screen buffer)
```

printNumF(num, dec, x, y[, divider[, length[, filler]]]);

Print a floating-point number at the specified coordinates in the screen buffer.
You can use the literals LEFT, CENTER and RIGHT as the x-coordinate to align the string on the screen.
WARNING: Floating point numbers are not exact, and may yield strange results when compared. Use at your own discretion.

Parameters:
```
num: the value to print (See note)
dec: digits in the fractional part (1-5) 0 is not supported. Use printNumI() instead.
x:   x-coordinate of the upper, left corner of the first digit/sign
y:   y-coordinate of the upper, left corner of the first digit/sign
divider:  <Optional>
          Single character to use as decimal point. Default is '.'
length:   <optional>
          minimum number of digits/characters (including sign) to display
filler:   <optional>
          filler character to use to get the minimum length. The character will be inserted in front
          of the number, but after the sign. Default is ' ' (space).
```

Usage:
```
myOLED.print(num, 3, CENTER,0); // Print the value of "num" with 3 fractional digits top centered
(in the screen buffer)
```

Notes:
```
Supported range depends on the number of fractional digits used.
Approx range is +/- 2*(10^(9-dec))
```

invertText(mode);

Select if text printed with print(), printNumI() and printNumF() should be inverted.

Parameters:
```
mode: true  - Invert the text
      false – Normal text
```

Usage:
```
myOLED.invertText(true); // Turn on inverted printing
```

Notes: SetFont() will turn off inverted printing

setFont(fontname);

Select font to use with print(), printNumI() and printNumF().

Parameters:
```
fontname: Name of the array containing the font you wish to use
```

Usage:
```
myOLED.setFont(SmallFont); // Select the font called SmallFont
```

Notes: You must declare the font-array as an external or include it in your sketch.

drawLine(x1, y1, x2, y2);

Draw a line between two points in the screen buffer.

Parameters:
```
x1: x-coordinate of the start-point
y1: y-coordinate of the start-point
x2: x-coordinate of the end-point
y2: y-coordinate of the end-point
```

Usage:
```
myOLED.drawLine(0,0,127,63); // Draw a line from the upper left to the lower right corner
```

clrLine(x1, y1, x2, y2);

Clear a line between two points in the screen buffer.

Parameters:
```
x1: x-coordinate of the start-point
y1: y-coordinate of the start-point
x2: x-coordinate of the end-point
y2: y-coordinate of the end-point
```

Usage:
```
myOLED.clrLine(0,0,127,63); // Clear a line from the upper left to the lower right corner
```

drawRect(x1, y1, x2, y2);

Draw a rectangle between two points in the screen buffer.

Parameters:
```
x1: x-coordinate of the start-corner
y1: y-coordinate of the start-corner
x2: x-coordinate of the end-corner
y2: y-coordinate of the end-corner
```

Usage:
```
myOLED.drawRect(64,32,127,63); // Draw a rectangle in the lower right corner of the screen
```

clrRect(x1, y1, x2, y2);

Clear a rectangle between two points in the screen buffer.

Parameters:
```
x1: x-coordinate of the start-corner
y1: y-coordinate of the start-corner
x2: x-coordinate of the end-corner
y2: y-coordinate of the end-corner
```

Usage:
```
myOLED.clrRect(64,32,127,63); // Clear a rectangle in the lower right corner of the screen
```

drawRoundRect(x1, y1, x2, y2);

Draw a rectangle with slightly rounded corners between two points in the screen buffer.
The minimum size is 5 pixels in both directions. If a smaller size is requested the rectangle will not be drawn.

Parameters:
```
x1: x-coordinate of the start-corner
y1: y-coordinate of the start-corner
x2: x-coordinate of the end-corner
y2: y-coordinate of the end-corner
```

Usage:
```
myOLED.drawRoundRect(0,0,63,31); // Draw a rounded rectangle in the upper left corner of the screen
```

clrRoundRect(x1, y1, x2, y2);

Clear a rectangle with slightly rounded corners between two points in the screen buffer.
The minimum size is 5 pixels in both directions. If a smaller size is requested the rectangle will not be drawn/cleared.

Parameters:
```
x1: x-coordinate of the start-corner
y1: y-coordinate of the start-corner
x2: x-coordinate of the end-corner
y2: y-coordinate of the end-corner
```

Usage:
```
myOLED.clrRoundRect(0,0,63,31); // Clear a rounded rectangle in the upper left corner of the screen
```

drawCircle(x, y, radius);

Draw a circle with a specified radius in the screen buffer.

Parameters:
```
x:      x-coordinate of the center of the circle
y:      y-coordinate of the center of the circle
radius: radius of the circle in pixels
```

Usage:
```
myOLED.drawCircle(63,31,20); // Draw a circle in the middle of the screen with a radius of 20 pixels
```

clrCircle(x, y, radius);

Clear a circle with a specified radius in the screen buffer.

Parameters:
```
x:      x-coordinate of the center of the circle
y:      y-coordinate of the center of the circle
radius: radius of the circle in pixels
```

Usage:
```
myOLED.clrCircle(63,31,20); // Clear a circle in the middle of the screen with a radius of 20 pixels
```

drawBitmap (x, y, data, sx, sy);	
Draw a bitmap in the screen buffer.	
Parameters:	x: x-coordinate of the upper, left corner of the bitmap y: y-coordinate of the upper, left corner of the bitmap data: array containing the bitmap-data sx: width of the bitmap in pixels sy: height of the bitmap in pixels
Usage:	myOLED.drawBitmap(0, 0, bitmap, 32, 32); // Draw a 32x32 pixel bitmap in the upper left corner
Notes:	You can use the online-tool "*ImageConverter Mono*" to convert pictures into compatible arrays. The online-tool can be found on the Rinky-Dink Electronics website. Requires that you *#include <avr/pgmspace.h>* when using an Arduino other than Arduino Due. While the bitmap data *MUST* be a multiple of 8 pixels high you do not need to display all the rows. Example: If the bitmap is 24 pixels high and you specify sy=20 only the upper 20 rows will be displayed.