UTFT

Multi-Platform Universal TFT display library

Manual

Introduction:

This library was originally the continuation of my ITDB02_Graph, ITDB02_Graph16 and RGB_GLCD libraries for Arduino and chipKit. As the number of supported display modules and controllers started to increase I felt it was time to make a single, universal library as it will be much easier to maintain in the future.

Basic functionality of this library was originally based on the demo-code provided by ITead studio (for the ITDB02 modules) and NKC Electronics (for the RGB GLCD module/shield).

This library supports a number of 8bit, 16bit and serial graphic displays, and will work with both Arduino, chipKit boards and select TI LaunchPads. For a full list of tested display modules and controllers, see the document **UTFT_Supported_display_modules_&_controllers.pdf**.

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

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

IMPORTANT:

When using 8bit and 16bit display modules there are some requirements you must adhere to. These requirements can be found in the document **UTFT_Requirements.pdf**.
There are no special requirements when using serial displays.

Since most people have only one or possibly two different display modules a lot of memory has been wasted to keep support for many unneeded controller chips.
As of v1.1 you now have the option to easily remove this unneeded code from the library. By disabling the controllers you don't need you can reduce the memory footprint of the library by several Kb. For more information, please refer to **memorysaver.h**.
TFT controllers used only by display modules and shields that have been retired by their vendors are as of v2.82 disabled by default.

If you are using the **"AquaLEDSource All in One Super Screw Shield"** on a chipKit Max32, please read the comment in **hardware/pic32/HW_PIC32_defines.h**

If you are using the **"CTE TFT LCD/SD Shield for Arduino Due"** or the **"ElecHouse TFT LCD Screen Shield for Arduino DUE / Taijiuino"**, please read the comment in **hardware/arm/HW_ARM_defines.h**

8 bit display shields designed for use on Arduino Uno (and similarly sized boards) can now be used on Arduino Megas. Please read the comment in **hardware/avr/HW_AVR_defines.h**

Some of the larger (4.3"+) display modules have not been tested on all supported development boards due to the high current requirement for the LED backlight.

DEFINED LITERALS:

Alignment

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

LEFT:	0
RIGHT:	9999
CENTER:	9998

Orientation

For use with InitLCD()

PORTRAIT:	0
LANDSCAPE:	1

VGA Colors

Predefined colors for use with setColor() and setBackColor()

VGA_BLACK	VGA_SILVER	VGA_GRAY	VGA_WHITE
VGA_MAROON	VGA_RED	VGA_PURPLE	VGA_FUCHSIA
VGA_GREEN	VGA_LIME	VGA_OLIVE	VGA_YELLOW
VGA_NAVY	VGA_BLUE	VGA_TEAL	VGA_AQUA

VGA_TRANSPARENT (only valid for setBackColor())

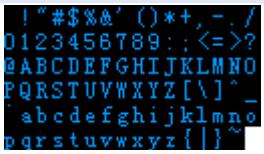
Display model

For use with UTFT()

Please see **UTFT_Supported_display_modules_&_controllers.pdf**

INCLUDED FONTS:

SmallFont



Charactersize: 8x12 pixels
Number of characters: 95

BigFont

Charactersize: 16x16 pixels
Number of characters: 95

SevenSegNumFont

Charactersize: 32x50 pixels
Number of characters: 10

More fonts can be found in the "**Resources**" section of **http://www.RinkyDinkElectronics.com/**. There is also a tool there to make your own fonts if you cannot find any that suit your needs. For those who want to know the specifications of the font arrays there is also an explanation of that there.

FUNCTIONS:

UTFT(Model, RS, WR, CS, RST[, ALE]);

The main class constructor when using 8bit or 16bit display modules.

Parameters:
```
Model:  See the separate document for the supported display modules
RS:     Pin for Register Select
WR:     Pin for Write
CS:     Pin for Chip Select
RST:    Pin for Reset
ALE:    <optional> Only used for latched 16bit shields
        Pin for Latch signal
```

Usage:
```
UTFT myGLCD(ITDB32S,19,18,17,16); // Start an instance of the UTFT class
```

UTFT(Model, SDA, SCL, CS, RST[, RS]);

The main class constructor when using serial display modules.

Parameters:
```
Model:  See the separate document for the supported display modules
SDA:    Pin for Serial Data
SCL:    Pin for Serial Clock
CS:     Pin for Chip Select
RST:    Pin for Reset
RS:     <optional> Only used for 5pin serial modules
        Pin for Register Select
```

Usage:
```
UTFT myGLCD(ITDB18SP,11,10,9,12,8); // Start an instance of the UTFT class
```

InitLCD([orientation]);

Initialize the LCD and set display orientation.

Parameters:
```
Orientation: <optional>
             PORTRAIT
             LANDSCAPE (default)
```

Usage:
```
myGLCD.initLCD(); // Initialize the display
```

Notes: This will reset color to white with black background. Selected font will be reset to *none*.

getDisplayXSize();

Get the width of the screen in the current orientation.

Parameters: None

Returns: Width of the screen in the current orientation in pixels

Usage:
```
Xsize = myGLCD.getDisplayXSize(); // Get the width
```

getDisplayYSize();

Get the height of the screen in the current orientation.

Parameters: None

Returns: Height of the screen in the current orientation in pixels

Usage:
```
Ysize = myGLCD.getDisplayYSize(); // Get the height
```

lcdOff();

Turn off the LCD. No commands will be executed until a lcdOn(); is sent.

Parameters:	None
Usage:	myGLCD.lcdOff(); // Turn off the lcd
Notes:	This function is currently only supported on PCF8833 and CPLD-based displays. CPLD-based displays will only turn off the backlight. It will accept further commands/writes.

lcdOn();

Turn on the LCD after issuing a lcdOff()-command.

Parameters:	None
Usage:	myGLCD.lcdOn(); // Turn on the lcd
Notes:	This function is currently only supported on PCF8833 and CPLD-based displays. CPLD-based displays will only turn on the backlight.

setContrast(c);

Set the contrast of the display.

Parameters:	c: Contrast-level (0-64)
Usage:	myGLCD.setContrast(64); // Set contrast to full (default)
Notes:	This function is currently only supported on PCF8833-based displays

setBrightness(br);

Set the brightness of the display backlight.

Parameters:	br: Brightness-level (0-16)
Usage:	myGLCD.setBrightness(16); // Set brightness to maximum (default)
Notes:	This function is currently only supported on CPLD-based displays

setDisplayPage(pg);

Set which memory page to display.

Parameters:	pg: Page (0-7) (0 is default)
Usage:	myGLCD.setDisplayPage(4); // Display page 4
Notes:	This function is currently only supported on CPLD-based displays

setWritePage(pg);

Set which memory page to use for subsequent display writes.

Parameters:	pg: Page (0-7) (0 is default)
Usage:	myGLCD.setWritePage(2); // Use page 2 for subsequent writes
Notes:	This function is currently only supported on CPLD-based displays

clrScr();

Clear the screen. The background-color will be set to black.

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

fillScr(r, g, b);

Fill the screen with a specified color.

Parameters: r: Red component of an RGB value (0-255)
g: Green component of an RGB value (0-255)
b: Blue component of an RGB value (0-255)
Usage: `myGLCD.fillScr(255,127,0); // Fill the screen with orange`

fillScr(color);

Fill the screen with a specified pre-calculated RGB565 color.

Parameters: color: RGB565 color value
Usage: `myGLCD.fillScr(VGA_RED); // Fill the screen with red`

setColor(r, g, b);

Set the color to use for all draw*, fill* and print commands.

Parameters: r: Red component of an RGB value (0-255)
g: Green component of an RGB value (0-255)
b: Blue component of an RGB value (0-255)
Usage: `myGLCD.setColor(0,255,255); // Set the color to cyan`

setColor(color);

Set the specified pre-calculated RGB565 color to use for all draw*, fill* and print commands.

Parameters: color: RGB565 color value
Usage: `myGLCD.setColor(VGA_AQUA); // Set the color to aqua`

getColor();

Get the currently selected color.

Parameters: None
Returns: Currently selected color as a RGB565 value (word)
Usage: `Color = myGLCD.getColor(); // Get the current color`

setBackColor(r, g, b);

Set the background color to use for all print commands.

Parameters: r: Red component of an RGB value (0-255)
g: Green component of an RGB value (0-255)
b: Blue component of an RGB value (0-255)
Usage: `myGLCD.setBackColor(255,255,255); // Set the background color to white`

setBackColor(color);

Set the specified pre-calculated RGB565 background color to use for all print commands.

Parameters: color: RGB565 color value
Usage: `myGLCD.setBackColor(VGA_LIME); // Set the background color to lime`

getBackColor();

Get the currently selected background color.

Parameters: None
Returns: Currently selected background color as a RGB565 value (word)
Usage: `BackColor = myGLCD.getBackColor(); // Get the current background color`

drawPixel(x, y);

Draw a single pixel.

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

Usage:
```
myGLCD.drawPixel(119,159); // Draw a single pixel
```

drawLine(x1, y1, x2, y2);

Draw a line between two points.

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:
```
myGLCD.drawLine(0,0,239,319); // Draw a diagonal line
```

drawRect(x1, y1, x2, y2);

Draw a rectangle between two points.

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:
```
myGLCD.drawRect(119,159,239,319); // Draw a rectangle
```

drawRoundRect(x1, y1, x2, y2);

Draw a rectangle with slightly rounded corners between two points. 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:
```
myGLCD.drawRoundRect(0,0,119,159); // Draw a rounded rectangle
```

fillRect(x1, y1, x2, y2);

Draw a filled rectangle between two points.

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:
```
myGLCD.fillRect(119,0,239,159); // Draw a filled rectangle
```

fillRoundRect(x1, y1, x2, y2);

Draw a filled rectangle with slightly rounded corners between two points. 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:
```
myGLCD.fillRoundRect(0,159,119,319); // Draw a filled, rounded rectangle
```

drawCircle(x, y, radius);

Draw a circle with a specified radius.

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:
```
myGLCD.drawCircle(119,159,20); // Draw a circle with a radius of 20 pixels
```

fillCircle(x, y, radius);

Draw a filled circle with a specified radius.

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:
```
myGLCD.fillCircle(119,159,10); // Draw a filled circle with a radius of 10 pixels
```

print(st, x, y[, deg]);

Print a string at the specified coordinates.
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
deg: <optional>
     Degrees to rotate text (0-359). Text will be rotated around the upper left corner.
```
Usage:
```
myGLCD.print("Hello, World!",CENTER,0); // Print "Hello, World!"
```
Notes:
```
CENTER and RIGHT will not calculate the coordinates correctly when rotating text.
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.
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:
```
myGLCD.printNumI(num,CENTER,0); // Print the value of "num"
```

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

Print a floating-point number at the specified coordinates.
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:
```
myGLCD.printNumF(num, 3, CENTER,0); // Print the value of "num" with 3 fractional digits
```
Notes:
```
Supported range depends on the number of fractional digits used.
Approx range is +/- 2*(10^(9-dec))
```

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:
```
myGLCD.setFont(BigFont); // Select the font called BigFont
```
Notes:
```
You must declare the font-array as an external or include it in your sketch.
```

getFont();

Get the currently selected font.

Parameters:
```
None
```
Returns:
```
Currently selected font
```
Usage:
```
CurrentFont = myGLCD.getFont(); // Get the current font
```

getFontXsize();

Get the width of the currently selected font.

Parameters:
```
None
```
Returns:
```
Width of the currently selected font in pixels
```
Usage:
```
Xsize = myGLCD.getFontXsize (); // Get font width
```

getFontYsize();

Get the height of the currently selected font.

Parameters:
```
None
```
Returns:
```
Height of the currently selected font in pixels
```
Usage:
```
Ysize = myGLCD.getFontYsize (); // Get font height
```

drawBitmap (x, y, sx, sy, data[, scale]);

Draw a bitmap on the screen.

Parameters:
```
x:     x-coordinate of the upper, left corner of the bitmap
y:     y-coordinate of the upper, left corner of the bitmap
sx:    width of the bitmap in pixels
sy:    height of the bitmap in pixels
data:  array containing the bitmap-data
scale: <optional>
       Scaling factor. Each pixel in the bitmap will be drawn as <scale>x<scale> pixels on screen.
```

Usage:
```
myGLCD.drawBitmap(0, 0, 32, 32, bitmap); // Draw a 32x32 pixel bitmap
```

Notes:
```
You can use the online-tool "ImageConverter 565" or "ImageConverter565.exe" in the Tools-folder to
convert pictures into compatible arrays. The online-tool can be found on my website.
Requires that you #include <avr/pgmspace.h> when using an Arduino other than Arduino Due.
```

drawBitmap (x, y, sx, sy, data, deg, rox, roy);

Draw a bitmap on the screen with rotation.

Parameters:
```
x:    x-coordinate of the upper, left corner of the bitmap
y:    y-coordinate of the upper, left corner of the bitmap
sx:   width of the bitmap in pixels
sy:   height of the bitmap in pixels
data: array containing the bitmap-data
deg:  Degrees to rotate bitmap (0-359)
rox:  x-coordinate of the pixel to use as rotational center relative to bitmaps upper left corner
roy:  y-coordinate of the pixel to use as rotational center relative to bitmaps upper left corner
```

Usage:
```
myGLCD.drawBitmap(50, 50, 32, 32, bitmap, 45, 16, 16); // Draw a bitmap rotated 45 degrees around
its center
```

Notes:
```
You can use the online-tool "ImageConverter 565" or "ImageConverter565.exe" in the Tools-folder to
convert pictures into compatible arrays. The online-tool can be found on my website.
Requires that you #include <avr/pgmspace.h> when using an Arduino other than Arduino Due.
```