Treball de Fi de Grau
Grau en Enginyeria en Tecnologies Industrials

Desenvolupament de llibreries en llenguatge C per poder controlar una pantalla LCD tàctil de 2.2 polzades mitjançant un microcontrolador PIC18F4520.

ANNEX A: Manual d’usuari de les llibreries programades

Autor: Victor Boguña Piferrer
Director/s: Manuel Moreno Eguilaz
Convocatòria: Juliol de 2014
USER MANUAL

Of the libraries:

System
SPIfunctions
LCD_basic_functions
Shapes
Writing_functions
Touch_functions

To control a touchable LCD screen

by Victor Boguña Piferrer
# Table of Contents

1-Introduction: ................................................................................................................. 4

2-How to use this manual: ............................................................................................. 5
   2.1-Text fonts: ............................................................................................................. 5
   2.2-Files: ..................................................................................................................... 5
   2.3-Importing a library: ............................................................................................ 6
   2.4-Manual structure: ............................................................................................... 6
   2.5-Variables type: ................................................................................................... 7

3-High level libraries: ..................................................................................................... 8
   3.1- Shapes library User Manual: ............................................................................. 8
      3.1.1- Shapes Introduction Page: .......................................................................... 8
      3.1.2- Shapes Functions Table: ............................................................................. 9
      3.1.3- Shapes functions description: ................................................................... 10
   3.2- Writing_functions library User Manual: ......................................................... 19
      3.2.1-Writing_functions Introduction Page: ......................................................... 19
      3.2.2-Writing_functions Functions Table: ......................................................... 20
      3.2.3-Writing_functions functions description: ............................................... 21
   3.3- Touch_functions library User Manual: ............................................................ 28
      3.3.1-Touch_functions Introduction Page: ............................................................. 28
      3.3.2-Touch_functions Functions Table: ............................................................... 29
      3.3.3-Touch_functions functions description: .................................................... 30

4-Low level libraries: .................................................................................................. 36
   4.1- system library User Manual: .......................................................................... 36
      4.1.1-system Introduction Page: ........................................................................... 36
      4.1.2-system Functions Table: .............................................................................. 37
      4.1.3-system functions description: .................................................................... 38
   4.2- SPIfunctions library User Manual: ............................................................. 40
      4.2.1-SPIfunctions Introduction Page: ................................................................. 40
      4.2.2-SPIfunctions Functions Table: ................................................................. 41
      4.2.3-SPIfunctions functions description: ......................................................... 42
   4.3- LCD_basic_functions library User Manual: ............................................... 45
4.3.1-LCD_basic_functions Introduction Page: ......................................................... 45
4.3.2-LCD_basic_functions Functions Table: ......................................................... 46
4.3.3-LCD_basic_functions functions description: ................................................. 47

Appendix A: Screen Coordinates ........................................................................ 57
Appendix B: Colors ............................................................................................... 58
Appendix C: Printing on the LCD ...................................................................... 60
Appendix D: ASCII codification ............................................................................ 62
Appendix E: TextFonts ......................................................................................... 65
1-Introduction:

Hello World!

If you, user, have learned some programing language, you will probably recognize this sentence.

The manual you are holding in your hands has only one main purpose: to provide you with a set of intuitive functions to control an LCD screen through a microcontroller using C language.

If you follow the instructions contained in this manual, surely you will be able not only to print the famous sentence in the screen but also develop some awesome applications.

The libraries described in this manual are designed to control a “Waveshare LCD 22” screen through a MICROCHIP PIC18F4520 microcontroller using C language.

Some of the functions contained in the libraries are not designed to be used by the final user so, in order to make this manual easier to understand, we divided the libraries in to sections:

- The first section contains the functions that allow the user to print shapes and text on the screen as well as to obtain the position where the screen is touched. Those are the functions you are going to use as a beginner.
- The other pack of libraries contains some basic functions designed to be used by other functions. If you are an advanced programmer and you want to develop more optimized code, we encourage you to read and understand this set of functions.

The table 1.1 shows how the libraries are distributed:

<table>
<thead>
<tr>
<th>Libraries</th>
<th>Shapes</th>
</tr>
</thead>
<tbody>
<tr>
<td>High Level Libraries</td>
<td>Writing_functions</td>
</tr>
<tr>
<td></td>
<td>Touch_functions</td>
</tr>
<tr>
<td>Low Level Libraries</td>
<td>System</td>
</tr>
<tr>
<td></td>
<td>SPIfunctions</td>
</tr>
<tr>
<td></td>
<td>LCD_basic_functions</td>
</tr>
</tbody>
</table>
2-How to use this manual:

In order to use this manual, and the functions described in it, in an appropriate way, we encourage you to read the following section.

2.1-Text fonts:

While reading this manual, you are going to find different text fonts. Every text font has its own meaning:

- **File.c** or **file.h**: the words that use this font make reference to a file name.
- **Library**: the words that use this font points to a library name.
- **Code**: the text that uses this font is code written in C language.

2.2-Files:

We provide you 6 libraries. Every library is divided in two main files:

- **C file** (with the “.c” extension): this file contains all the functions and global variables of the library.
- **H file** (with the “.h” extension): this file contains all the #include, #define and typedef instructions as well as the function prototypes.

<table>
<thead>
<tr>
<th>Library Name</th>
<th>C file (functions)</th>
<th>H file (header)</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>System.c</td>
<td>System.h</td>
</tr>
<tr>
<td>SPIfunctions</td>
<td>SPIfunctions.c</td>
<td>SPIfunctions.h</td>
</tr>
<tr>
<td>LCD_basic_functions</td>
<td>LCD_basic_functions.c</td>
<td>LCD_basic_functions.h</td>
</tr>
<tr>
<td>Shapes</td>
<td>Shapes.c</td>
<td>Shapes.h</td>
</tr>
<tr>
<td>Writing_functions</td>
<td>Writing_functions.c</td>
<td>Writing_functions.h</td>
</tr>
<tr>
<td>Touch_functions</td>
<td>Touch_functions.c</td>
<td>Touch_functions.h</td>
</tr>
</tbody>
</table>

**Fig. 2.2.1. Libraries files**

We also give you two extra files:

- **ConfigurationBits.h**: this file contains the code used to set the configuration bits of the PIC18F4520 microcontroller.
- **Asciihex8x16.h**: this file contains a text font, used by the Writing_functions library.

To download the files follow this link:

[https://www.dropbox.com/sh/4yqtr7lxnxmlcxs/AAAuqnTB3S6AXcGQENUXAkRpa](https://www.dropbox.com/sh/4yqtr7lxnxmlcxs/AAAuqnTB3S6AXcGQENUXAkRpa)
2.3-Importing a library:

When you want to use a function contained in one of the libraries described in this manual you must follow this instructions to import the library:

First, you will need to place in your working directory the library.c and library.h files.

Then, add in your main.c file the “include” sentence. For example:

```
#include "library.h"
```

Finally, add the library.c file to your source file directory and the library.h file into the header file directory of your project.

Some of the libraries depend on other libraries to work. At the beginning of each library manual you will find a section where is explained what libraries will be needed and how to import them to your project.

2.4-Manual structure:

Every C library we provide you has its own user manual.

Each manual is structured in 3 main parts:

- **Introduction page:** This first page shows the basic information of the library as well as some instructions and requisites needed to use the functions. You must read this first page in order to use properly the functions of the library.

- **Functions table:** this second section is a table with the name of all the functions that belongs to the library and a short description of them as well as a color code. The color code is used to distinguish between the high level functions, which are designed to be used by the user, and the low level function, which are designed to be used by other functions.

<table>
<thead>
<tr>
<th>Color Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>This is a really basic function designed to be used internally by other functions.</td>
</tr>
<tr>
<td>Yellow</td>
<td>This is a medium level function which can be used by the final users but is not recommended.</td>
</tr>
<tr>
<td>Green</td>
<td>This is a high level function designed to be used by the final user.</td>
</tr>
<tr>
<td>Gray</td>
<td>This is a test function which uses some of the high level functions of the library as a didactic example.</td>
</tr>
</tbody>
</table>

---

Fig. 2.4.1. Color code explanation
• Functions description: This section provides an extensive description of each function of the library. Each page of this section can be considered a “function manual” which contains all the information needed to use the function, as well as an explanation about the parameters, the returned values and some examples.

2.5-Variables type:

As it was said before, all the libraries are programmed using C language.

However, you will find some strange variable types. Those variables type are just the common ones (from C language) but renamed, in order to make the programming faster and easier.

They are defined in the System.h file and the most common are:

- u8: this variable type is equivalent to an unsigned char.
- u16: this variable is equivalent to an unsigned short.
- xy_t: this is a structure used in the Touching_functions library and contains two variables of 16 bit. For more information read the Touching_functions library introduction.
3-High level libraries:

3.1- Shapes library User Manual:

3.1.1- Shapes Introduction Page:

INTRODUCTION

This library provides some functions that allow the user to print an important set of different geometrical shapes on the LCD screen, from simple pixel dots to complex polygons and circles.

HOW TO USE IT

First, you will need to place in your working directory the following 2 files:

- Shapes.c
- Shapes.h

Then, add in your main.c file the following sentence:

#include "Shapes.h"

Finally, add the Shapes.c file to your source file directory and the Shapes.h file into the header file directory of your project.

PREREQUISITES

Because this library uses some of the functions provided by the system, SPIfunctions and LCD_basic_functions libraries, you must place the system.c, system.h, SPIfunctions.c, SPIfunctions.h, LCD_basic_functions.c and LCD_basic_functions.h files in you working directory as well as in your source and header file directories.

As it is included in the Shapes.h file, you can avoid writing the following sentences:

#include "system.h"
#include "SPIfunctions.h"
#include "LCD_basic_functions.h"

You must follow the same prerequisites explained in the SPIfunctions manual (See page 40)

CALLING THE FUNCTIONS

The first step when you want to use the functions contained in this library is to call the function init_lcd(). This function have to be called at the very beginning of the main function before calling any function described in this manual (For more information about this function see LCD_basic_functions manual on page 45). Once this initial function is called, you can use any of the other functions, but previously we encourage all the users to read and understand the following sections:

- Appendix A: Screen Coordinates
- Appendix B: Colors

These appendixes will be useful to understand which values can be set as parameters of the functions described in this manual.
### 3.1.2- Shapes Functions Table:

<table>
<thead>
<tr>
<th>Color Code</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>draw_pixdot</code></td>
<td>This function colors the selected pixel of the screen with a defined color.</td>
</tr>
<tr>
<td></td>
<td><code>draw_sqdot</code></td>
<td>This function draws a square dot centered on a defined coordinate and with a defined size.</td>
</tr>
<tr>
<td></td>
<td><code>draw_line</code></td>
<td>This function allows the user to draw a line between two selected pixels of the LCD screen.</td>
</tr>
<tr>
<td></td>
<td><code>draw_rectangle</code></td>
<td>This function draws a non-filled rectangle by giving the top-left coordinate and the height and width properties.</td>
</tr>
<tr>
<td></td>
<td><code>draw_filled_rectangle</code></td>
<td>This function draws a filled rectangle by giving the top-left coordinate and the height and width properties.</td>
</tr>
<tr>
<td></td>
<td><code>draw_circumference</code></td>
<td>This function draws a circumference by giving the center and the radius.</td>
</tr>
<tr>
<td></td>
<td><code>draw_circle</code></td>
<td>This function draws a circle by giving the center and the radius.</td>
</tr>
<tr>
<td></td>
<td><code>draw_polygon</code></td>
<td>This function draws an oriented n vertex polygon circumscribed in a circumference.</td>
</tr>
<tr>
<td></td>
<td><code>shapes_test</code></td>
<td>This function calls other functions of this library to serve as a didactic example.</td>
</tr>
</tbody>
</table>

**Fig. 3.1.1.** Shapes library functions table
3.1.3-Shapes functions description:

Name of the function
draw_pixdot

Description
This function colors the selected pixel of the screen with a defined color.

Syntax
void draw_pixdot(u16 x, u16 y, u16 color);

Parameter
The x and y parameters give the coordinates of a pixel on the LCD screen.
Notice that the screen size is 240x320 pixels so x must be contained between 0 and 239 and y have to be contained between 0 and 319.

The parameter color defines the color used to print the pixel.

Note: see the appendix A to know how the screen coordinates are defined.

Note: to see how are the colors represented see the appendix B.

Return Values
None

Precondition
The function init_lcd have to be called at the very beginning of the main function before calling this function.

Side Effects
None

Example
//paint the pixel (100,100) with black color
draw_pixdot(100, 100, 0x0000);
Name of the function
draw_sqdot

Description
This function draws a square dot centered on given coordinates which have a height/width defined by the user.

Syntax
void draw_sqdot(u16 x, u16 y, u8 size, u16 color);

Parameter
The \( x \) and \( y \) parameters give the coordinates of a pixel on the LCD screen.
Notice that the screen size is 240x320 pixels so \( x \) must be contained between 0 and 239 and \( y \) have to be contained between 0 and 319.

The parameter \( size \) provides the height/width of the square dot in pixels.

The parameter \( color \) defines the color used to print the square dot.

Note: see the appendix A to know how the screen coordinates are defined.

Note: to see how are the colors represented see the appendix B.

Return Values
None

Precondition
The function \texttt{init_lcd} have to be called at the very beginning of the \texttt{main} function before calling this function.

Side Effects
None

Example
//draw a square dot centered on (100, 100) which has 5 pixels as height/width
draw_sqdot (100, 100, 5, 0x0000);
Name of the function
draw_line

Description
This function allows the user to draw a line between two selected pixels of the LCD screen.

The function is based on the Bersenham's algorithm. For more information see: http://en.wikipedia.org/wiki/Bresenham's_line_algorithm

Syntax
void draw_line(u16 x_i, u16 y_i, u16 x_f, u16 y_f, u8 thickness, u16 color);

Parameter
The \( x_i / y_i \) and \( x_f / y_f \) parameters give the coordinates of two pixels of the LCD screen. Notice that the screen size is 240x320 pixels so \( x_i \) and \( x_f \) must be contained between 0 and 239 and \( y_i \) and \( y_f \) have to be contained between 0 and 319.

The parameter thickness provides the thickness in pixels of the defined line.

The parameter color defines the color used to draw the line.

Note: see the appendix A to know how the screen coordinates are defined.

Note: to see how are the colors represented see the appendix B.

Return Values
None

Precondition
The function \textit{init_lcd} have to be called at the very beginning of the main function before calling this function.

Side Effects
None

Example
//drawing the diagonals of the LCD screen
draw_line(0, 0, 239, 319, 1, 0x0000);
draw_line(239, 0, 0, 319, 1, 0x0000);
Name of the function
draw_rectangle

Description
This function draws a non-filled rectangle by giving the top-left coordinate and the height and width properties.

Syntax
void draw_rectangle(u16 x, u16 y, u16 height, u16 width, u8 thickness, u16 color);

Parameter
The \( x \) and \( y \) parameters give the coordinates of the rectangle top-left corner. Notice that the screen size is 240x320 pixels so \( x \) must be contained between 0 and 239 and \( y \) have to be contained between 0 and 319.

The parameters \( \text{height} \) and \( \text{width} \) give the dimensions of the rectangle sides in pixels.

The parameter \( \text{thickness} \) provides the thickness in pixels of the lines that conforms the rectangle.

The parameter \( \text{color} \) defines the color used to draw the rectangle.

Note: see the appendix A to know how the screen coordinates are defined.

Note: to see how are the colors represented see the appendix B.

Return Values
None

Precondition
The function \text{init_lcd} have to be called at the very beginning of the main function before calling this function.

Side Effects
None

Example
//drawing a black non-filled rectangle
draw_rectangle(20, 20, 280, 200, 5, 0x0000);
Name of the function
draw_filled_rectangle

Description
This function draws a filled rectangle by giving the top-left coordinate and the height and width properties.

Syntax
void draw_filled_rectangle(u16 x, u16 y, u16 height, u16 width, u16 color);

Parameter
The x and y parameters give the coordinates of the rectangle top-left corner. Notice that the screen size is 240x320 pixels so x must be contained between 0 and 239 and y have to be contained between 0 and 319.

The parameters height and width give the dimensions of the rectangle sides in pixels.

The parameter color defines the color used to print the rectangle.

Note: see the appendix A to know how the screen coordinates are defined.

Note: to see how are the colors represented see the appendix B.

Return Values
None

Precondition
The function init_lcd have to be called at the very beginning of the main function before calling this function.

Side Effects
None

Example
//drawing a black filled rectangle
draw_filled_rectangle(20, 20, 280, 200, 0x0000);
Name of the function
draw_circumference

Description
This function draws a circumference by giving the center and the radius.

Syntax
void draw_circumference(u16 x0, u16 y0, u16 radius, u8 thickness, u16 color);

Parameter
The \textit{x0} and \textit{y0} parameters give the coordinates of the circumference center. Notice that the screen size is 240x320 pixels so \textit{x0} must be contained between 0 and 239 and \textit{y0} have to be contained between 0 and 319.

The parameter \textit{radius} sets the radius of the circumference.

The parameter \textit{thickness} provides the thickness in pixels of the circumference.

The parameter \textit{color} defines the color used to draw the circumference.

\textbf{Note:} see the appendix A to know how the screen coordinates are defined.

\textbf{Note:} to see how are the colors represented see the appendix B.

Return Values
None

Precondition
The function \textit{init_lcd} have to be called at the very beginning of the \texttt{main} function before calling this function.

Side Effects
None

Example
//drawing a black circumference
draw_circumference (50, 50, 25, 3, 0x0000);
Name of the function
draw_circle

Description
This function draws a circle by giving the center and the radius.

Syntax
void draw_circle(u16 x0, u16 y0, u16 radius, u16 color);

Parameter
The $x_0$ and $y_0$ parameters give the coordinates of the circle center. Notice that the screen size is 240x320 pixels so $x_0$ must be contained between 0 and 239 and $y_0$ have to be contained between 0 and 319.

The parameter $radius$ sets the radius of the circle.

The parameter $color$ defines the color used to print the circle.

Note: see the appendix A to know how the screen coordinates are defined.

Note: to see how are the colors represented see the appendix B.

Return Values
None

Precondition
The function init_lcd have to be called at the very beginning of the main function before calling this function.

Side Effects
None

Example
//drawing a black circumference
draw_circle (50, 50, 25, 0x0000);
Name of the function
draw_polygon

Description
This function draws an oriented n vertex polygon circumscribed in a circumference.

Syntax
void draw_polygon(u16 x0, u16 y0, u16 radius, u8 n_sides, float angle, u8 thickness, u16 color);

Parameter
The x0 and y0 parameters give the coordinates of the circumference center. Notice that the screen size is 240x320 pixels so x0 must be contained between 0 and 239 and y0 have to be contained between 0 and 319.

The parameter radius sets the radius of the circumference where the polygon will be circumscribed.

The parameter n_sides sets the number of vertex the polygon will have.

The parameter angle orientates the polygon. This parameter is defined in degrees so it has to be contained between 0 and 360.

The parameter color defines the color used to draw the polygon.

Note: see the appendix A to know how the screen coordinates are defined.

Note: to see how are the colors represented see the appendix B.

Return Values
None

Precondition
The function init_lcd have to be called at the very beginning of the main function before calling this function.

Side Effects
None

Example
//drawing a black triangle oriented to the right.
draw_polygon(100, 100, 50, 3, 0, 2, 0x0000);
Name of the function
shapes_test

Description
This function calls the functions draw_filled_rectangle, draw_rectangle, draw_circle, draw_circumference, draw_line and draw_polygon to serve as a didactic example.

Syntax
void shapes_test(void);

Parameter
None

Return Values
None

Precondition
The function init_lcd have to be called at the very beginning of the main function before calling this function.

Side Effects
None

Example
//executing the example function of the library Shapes
shapes_test();
3.2- Writing_functions library User Manual:

3.2.1-Writing_functions Introduction Page:

INTRODUCTION

This library provides a set of functions that allows the user to select a font, set its properties and then print a char or string in the LCD screen.

HOW TO USE IT

First, you will need to place in your working directory the following 2 files:

- Writing_functions.c
- Writing_functions.h

Then, add in your main.c file the following sentence:

```c
#include "Writing_functions.h"
```

Finally, add the Writing_functions.c file to your source file directory and the Writing_functions.h file into the header file directory of your project.

PREREQUISITES

Because this library uses some of the functions provided by the system, SPIfunctions and LCD_basic_functions libraries, you must place the system.c, system.h, SPIfunctions.c, SPIfunctions.h, LCD_basic_functions.c and LCD_basic_functions.h files in your working directory as well as in your source and header file directories.

As it is included in the Writing_functions.h file, you can avoid writing the following sentences:

```c
#include "system.h"
#include "SPIfunctions.h"
```

You must follow the same prerequisites explained in the SPIfunctions manual (See page 40)

CALLING THE FUNCTIONS

The first step when you want to use the functions contained in this library is to call the function init_lcd(). This function have to be called at the very beginning of the main function before calling any function described in this manual (For more information about this function see LCD_basic_functions manual on page 45). Once this initial function is called, you can use any of the other functions, but previously we encourage all the users to read and understand the following sections:

- Appendix A: Screen Coordinates
- Appendix B: Colors

These appendixes will be useful to understand which values can be set as parameters of the functions described in this manual.

Furthermore, before using the functions contained in this library you must ensure to have defined the font you are going to use.

We provide you a font called ascii in the file asciihex8x16.h. Include this file in your working directory as well as in the header files directory.

For more information about the fonts and how to import them see the appendix E.
## 3.2.2- Writing_functions Functions Table:

<table>
<thead>
<tr>
<th>Color Code</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>set_text_font</td>
<td>This function allows the user to select a text font.</td>
</tr>
<tr>
<td></td>
<td>set_text_size</td>
<td>This function allows the user to choose the size of the text font.</td>
</tr>
<tr>
<td></td>
<td>set_text_color</td>
<td>This function allows the user to select the text and background colors and also if he wants the background to be transparent.</td>
</tr>
<tr>
<td></td>
<td>DisplayChar</td>
<td>This function draws a character with the properties defined in the previous functions.</td>
</tr>
<tr>
<td></td>
<td>DisplayString_grid</td>
<td>This function allows the user to place a set of characters into fixed positions of the screen.</td>
</tr>
<tr>
<td></td>
<td>DisplayString_xy</td>
<td>This function allows the user to print a set of characters on a selected coordinates of the screen.</td>
</tr>
<tr>
<td></td>
<td>WritingTest</td>
<td>This function calls other functions of this library to serve as a didactic example.</td>
</tr>
</tbody>
</table>

---

*Fig. 3.2.1. Writing_functions library functions table*
3.2.3- Writing_functions functions description:

**Name of the function**
set_text_font

**Description**
This function allows the user to select a text font.

**Syntax**
void set_text_font(const rom unsigned char *font, u8 width, u8 height);

**Parameter**
The `font` parameter must be an array which contains the font you want to use.
The `width` parameter has to be set as the width (in pixels) of the characters contained in the font.
The `height` parameter has to be set as the height (in pixels) of the characters contained in the font.

**Note:** for more information about the fonts and how to import them see the appendix E.

**Return Values**
None

**Precondition**
The function `init_lcd` have to be called at the very beginning of the `main` function before calling this function.

**Side Effects**
None

**Example**
//selecting the font called ascii
set_text_font(ascii, 8, 16);
**Name of the function**

`set_text_size`

**Description**

This function allows the user to choose the size of the text font.

**Syntax**

```c
void set_text_size(u8 size);
```

**Parameter**

The parameter `size` defines the size of the characters which will be printed in the screen. The final size of the characters is obtained by multiplying the value of the `size` parameter with the height and width (properties that are defined through the function `set_text_font`). Because of that, the `size` parameter must be a natural number.

**Return Values**

None

**Precondition**

The function `init_lcd` have to be called at the very beginning of the `main` function before calling this function.

**Side Effects**

None

**Example**

```c
//setting the size of the characters as 3 times the size of the font.
set_text_size(3);
```
Name of the function
set_text_color

Description
This function allows the user to select the text and background colors and also if he wants the background to be transparent.

Syntax
void set_text_color(u16 text_color, u16 back_color, u8 transparent_background);

Parameter
The text_color parameter represents the color of the characters you want to print in the screen.

The back_color parameter represents the color characters background.

The transparent_background parameter only can take two possible values:
  1: The characters background will not be colored.
  0: The characters background will be printed with the color selected with the back_color variable.

Note: to see how are the colors represented see the appendix B.

Return Values
None

Precondition
The function init_lcd have to be called at the very beginning of the main function before calling this function.

Side Effects
None

Example
//setting the font color properties. Text color: black. Background color: white. No transparent background.
set_text_color(0x0000, 0xffff, 0);
Name of the function
DisplayChar

Description
This function draws a character with the properties defined through the functions set_text_font, set_text_size and set_text_color.

Syntax
void DisplayChar(u8 casc, u16 position_x, u16 position_y);

Parameter
The casc parameter is the character you want to print in the screen.
You can set this parameter using the marks ‘…’ and “…” or using the ASCII codification.

The position_x and position_y parameters give the coordinates of the top-left corner of the letter you want to print.
Notice that the screen size is 240x320 pixels so position_x must be contained between 0 and 239 and position_y have to be contained between 0 and 319, but taking into account the dimensions of the text font.

Note: for more information about the ASCII codification see the appendix D.

Note: see the appendix A to know how the screen coordinates are defined.

Return Values
None

Precondition
The function init_lcd have to be called at the very beginning of the main function before calling this function.

Side Effects
None

Example
// Printing the "A" letter in position (100, 100):
// using the ASCII codification:
DisplayChar(65, 100, 100);
// using the marks ‘…’:
DisplayChar('A', 100, 100);
Name of the function
DisplayString_grid

Description
This function allows the user to place a set of characters into fixed positions of the screen. The screen is divided like an imaginary grid where every section is defined by the height and width of the font.

Syntax
void DisplayString_grid(const rom u8 *s, u16 x, u16 y);

Parameter
The s parameter is the string you want to print in the screen. Is mandatory to define this string as a “const rom unsigned char” type.

The x and y parameters give the coordinates of the first letter you want to print. When using this function, the screen can be considered a matrix where the number rows and columns have to be calculated using the following equations:

\[
\text{Rows} = \frac{320}{\text{height}\times\text{size}}
\]
\[
\text{Columns} = \frac{240}{\text{width}\times\text{size}}
\]

The height, width and size are parameters used in the functions set_text_font and set_text_size.
So, the x coordinate must be contained between 0 and “Columns” and y coordinate must be between 0 and “Rows”.

Return Values
None

Precondition
The function init_lcd have to be called at the very beginning of the main function before calling this function.

Side Effects
None

Example
//Printing “Hello World” in the to-left corner of the screen
DisplayString_grid((const rom usigned char)"Hello World!", 0, 0);
Name of the function
DisplayString_xy

Description
This function allows the user to print a set of characters on a selected coordinates of the screen.

Syntax
DisplayString_xy(const rom u8 *s, u16 x, u16 y);

Parameter
The s parameter is the string you want to print in the screen. Is mandatory to define this string as a “const rom unsigned char” type.

The x and y parameters give the coordinates of the top-left corner of the first letter you want to print.
Notice that the screen size is 240x320 pixels so x must be contained between 0 and 239 and y have to be contained between 0 and 319, but taking into account the dimensions of the text font.

Note: see the appendix A to know how the screen coordinates are defined.

Return Values
None

Precondition
The function init_lcd have to be called at the very beginning of the main function before calling this function.

Side Effects
None

Example
//Printing “Hello World” using an (x,y) coordinate
DisplayString_xy((const rom unsigned char)“Hello World!”, 20, 100);
**Name of the function**
WritingTest

**Description**
This function uses the functions contained in this library to show different messages on the screen with different properties in order to serve as a didactic example.

**Syntax**

```c
void WritingTest(void);
```

**Parameter**
None

**Return Values**
None

**Precondition**
The function `init_lcd` have to be called at the very beginning of the main function before calling this function.

**Side Effects**
None

**Example**
```c
//executing the example function of the library Writing_functions
WritingTest();
```
3.3- Touch_functions library User Manual:

3.3.1-Touch_functions Introduction Page:

INTRODUCTION

This library provides the user with a set of functions to control and read the position where the LCD screen is touched.

HOW TO USE IT

First, you will need to place in your working directory the following 2 files:

- Touch_functions.c
- Touch_functions.h

Then, add in your main.c file the following sentence:

#include "Touch_functions.h"

Finally, add the Touch_functions.c file to your source file directory and the Touch_functions.h file into the header file directory of your project.

PREREQUISITES

Because this library uses some of the functions provided by the system, SPIfunctions and LCD_basic_functions libraries, you must place the system.c, system.h, SPIfunctions.c, SPIfunctions.h, LCD_basic_functions.c and LCD_basic_functions.h files in your working directory as well as in your source and header file directories.

As it is included in the Touch_functions.h file, you can avoid writing the following sentences:

#include "system.h"
#include "SPIfunctions.h"
#include "LCD_basic_functions.h"

You must follow the same prerequisites explained in the SPIfunctions manual (See page 40)

CALLING THE FUNCTIONS

The first step when you want to use the functions contained in this library is to call the function init_lcd(). This function have to be called at the very beginning of the main function before calling any function described in this manual (For more information about this function see LCD_basic_functions manual on page 45).

Once this initial function is called, you can use any of the other functions.

Some of the functions explained in this library use the xy_t structure. This structure (defined in the system.h file) saves two 16 bit coordinates.

If you want to access to the information you must use the following instructions:

//defining a structure
xy_t temp_xy;
//saving information in x and y positions.
temp_xy.x=0;
temp_xy.y=0;
### 3.3.2- Touch_functions Functions Table:

<table>
<thead>
<tr>
<th>Color Code</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>get_touch_position</td>
<td>This function returns the unscaled coordinates of the touch position.</td>
</tr>
<tr>
<td></td>
<td>get_touch_rescaled_position</td>
<td>This function returns the scaled coordinates of the touch position.</td>
</tr>
<tr>
<td></td>
<td>set_accuracy</td>
<td>This function allows the user to modify the global variable &quot;accuracy&quot; used in the function &quot;get_xy&quot;</td>
</tr>
<tr>
<td></td>
<td>get_xy</td>
<td>This function returns the average of several scaled coordinates of the touch position.</td>
</tr>
<tr>
<td></td>
<td>get_direction</td>
<td>This function returns the direction of the touch position trajectory.</td>
</tr>
<tr>
<td></td>
<td>touch_test</td>
<td>This function calls other functions of this library to serve as a didactic example.</td>
</tr>
</tbody>
</table>

**Fig. 3.3.1.** Touch_functions library functions table
3.3.3- Touch_functions functions description:

Name of the function
get_touch_position

Description
This function returns the unscaled coordinates of the touch position.

Syntax
xy_t get_touch_position(void);

Parameter
None

Return Values
The returned value is a xy_t structure which contains the x and y unscaled coordinates of the touch position.
To get the x and y values use the instructions xy_t_structure.x and xy_t_structure.y, where xy_t_structure is the name you give to the variable which contains the returned value.

If the screen is not touched the function returns the value 0xFFFF for both x and y coordinates.

Precondition
The function init_lcd have to be called at the very beginning of the main function before calling this function.

The instruction en_touch() have to be called before using this function and the instruction dis_touch() must be written after calling this function.

Side Effects
None

Example
//Defining a variable which will contain the returned value
xy_t aux_xy;
//getting the unscaled touch position
while(condition){
    aux_xy get_touch_position();
}
//Note: is recommended to use this function inside a loop.
Name of the function
get_touch_rescaled_position

Description
This function returns the rescaled coordinates of the touch position.

Syntax
xy_t get_touch_rescaled_position(void);

Parameter
None

Return Values
The returned value is a xy_t structure which contains the x and y rescaled coordinates of the touch position.
To get the x and y values use the instructions xy_t_structure.x and xy_t_structure.y, where xy_t_structure is the name you give to the variable which contains the returned value.

If the screen is not touched the function returns the value 0xFFFF for both x and y coordinates.

Note: see the appendix A to know how the screen coordinates are defined.

Precondition
The function init_lcd have to be called at the very beginning of the main function before calling this function.

The instruction en_touch() have to be called before using this function and the instruction dis_touch() must be written after calling this function.

Side Effects
None

Example
//Defining a variable which will contain the returned value
xy_t aux_xy;
//getting the rescaled touch position
while(condition){
    aux_xy get_touch_rescaled_position();
}
//Note: is recommended to use this function inside a loop.
Name of the function
set_accuracy

Description
This function allows the user to modify the global variable accuracy used in the function get_xy.

Syntax
void set_accuracy(u8 var);

Parameter
The parameter var is an unsigned char so the values which can take are contained between 0 and 255.

Return Values
None

Precondition
The function init_lcd have to be called at the very beginning of the main function before calling this function.

Side Effects
None

Example
//Setting the value of the global variable “accuracy”
set_accuracy(10);
Name of the function
get_xy

Description
This function returns the average of several scaled coordinates of the touch position.

As a user, you can set the number of samples you want to average by modifying the global variable `accuracy` through the function `set_accuracy`.

The number of samples you need depends on the use of the function you want to make:
If you want to obtain a high amount of samples in a short period of time (for example, to develop a cursor) you will need to set the accuracy variable with a low value, but the accuracy of the samples you will get will be also poor.
If you want to obtain a very accurate coordinates (for example, for touching buttons on the screen) you will need to set the accuracy variable with a high value, but the time between you get samples will be also larger.

Syntax
```c
xy_t get_xy(void);
```

Parameter
None

Return Values
The returned value is a `xy_t` structure which contains the x and y coordinates of the touch position.
To get the x and y values use the instructions `xy_t_structure.x` and `xy_t_structure.y`, where `xy_t_structure` is the name you give to the variable which contains the returned value.

If the screen is not touched the function returns the value `0xFFFF` for both x and y coordinates.

Note: see the appendix A to know how the screen coordinates are defined.

Precondition
The function `init_lcd` have to be called at the very beginning of the main function before calling this function.

Side Effects
None

Example
```c
//Defining a variable which will contain the returned value
xy_t aux_xy;
//setting the variable accuracy
set_accuracy(10);
//getting the average of 10 rescaled touch positions
while(condition){
    aux_xy get_xy();
}
//Note: is recommended to use this function inside a loop.
```
Name of the function
get_direction

Description
This function returns the direction of the touch position trajectory.

This function uses the function get_xy described in this manual. As a consequence, this function also depends on the variable accuracy settled with the function set_accuracy. Read the considerations described in the previous function before using this function and assign a value to the accuracy variable that fits with your application’s purpose.

Syntax
u8 get_direction(void);

Parameter
None

Return Values
The returned value is an unsigned char which can take only 5 possible values:
- 0: the screen is not touched or is touched but with no displacement.
- 1: UP
- 2: LEFT
- 3: DOWN
- 4: RIGHT

The direction of the movement is defined in concordance with how the screen coordinates are defined.

Note: see the appendix A to know how the screen coordinates are defined.

Precondition
The function init_lcd have to be called at the very beginning of the main function before calling this function.

Side Effects
None

Example
//Defining an unsigned char which will contain the returned value
u8 dir;
//getting direction of the touch position trajectory
while(condition){
    dir=get_direction();
}
//Note: is recommended to use this function inside a loop.
Name of the function
touch_test

Description
This function uses the function get_xy contained in this library to print with black color the pixel of the screen which is being touched by the user.

Syntax
void touch_test(void);

Parameter
None

Return Values
None

Precondition
The function init_lcd have to be called at the very beginning of the main function before calling this function.

Side Effects
None

Example
//executing the example function of the library Touch_functions
touch_test();
4-Low level libraries:

4.1- system library User Manual:

4.1.1-system Introduction Page:

INTRODUCTION

This library contains the definition of all data types used in this set of libraries as well as the definition of the xt_t structure and two delay functions,

HOW TO USE IT

First, you will need to place in your working directory the following 2 files:

- system.c
- system.h

Then, add in your main.c file the following sentence:

#include "sistem.h"

Finally, add the system.c file to your source file directory and the system.h file into the header file directory of your project.

PREREQUISITES

You must follow the same prerequisites explained in the SPIfunctions manual (See page 40)

CALLING THE FUNCTIONS

The only consideration when using the functions contained in this library is to have configured the frequency of the microcontroller as HSPLL by using the sentence:

#pragma config OSC = HSPLL

or by adding the ConfigurationBits.h to the project.
4.1.2-system Functions Table:

<table>
<thead>
<tr>
<th>Color Code</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>delay_ms</td>
<td>This function executes a delay during the number of milliseconds indicated by the user.</td>
</tr>
<tr>
<td></td>
<td>delay_us</td>
<td>This function executes a delay during the number of microseconds indicated by the user.</td>
</tr>
</tbody>
</table>

Fig. 4.2.1, system library functions table
4.1.3-system functions description:

Name of the function
delay_ms

Description
This function executes a delay during the number of milliseconds indicated by the user.

Syntax
void delay_ms(u16 ms);

Parameter
The parameter ms represents the amount of milliseconds it takes the delay.

Return Values
None

Precondition
None

Side Effects
None

Example
//Executing a delay during half a second.
delay_ms(500);
Name of the function
delay_us

Description
This function executes a delay during the number of microseconds indicated by the user.

Syntax
void delay_us(u16 us);

Parameter
The parameter us represents the amount of microseconds it takes the delay.

Return Values
None

Precondition
None

Side Effects
None

Example
//Executing a delay during half a millisecond.
delay_us(500);
INTRODUCTION

This library allows the communication, through the SPI bus, between a PIC18F4520 and both the screen display controller (BD663474) and the touch screen controller (XPT2046) from a WaveShare LCD 22.

HOW TO USE IT

First, you will need to place in your working directory the following 2 files:

- SPIfunctions.c
- SPIfunctions.h

Then, add in your main.c file the following sentence:

#include "SPIfunctions.h"

Finally, add the SPIfunctions.c file to your source file directory and the SPIfunctions.h file into the header file directory of your project.

PREREQUISITES

In order to have an optimized transfer speed you may set the oscillator frequency of the PIC18F4520 as high as you can.

You can add the following sentence to your code to improve the speed:

#pragma config OSC = HSPLL
//Normal frequency: OSC = HS

Or you can add to your project the ConfigurationBits.h file we provide you.

Note: be careful because the frequency of the Micro will be multiplied by 4 so some functions like delays can be affected.

CALLING THE FUNCTIONS

The communication between the microcontroller and the LCD screen is via SPI so, if you want to use the post_data() function you have to enable the CS (Chip Select) of the BD663474 and if you want to use the get_touch_data() function you have to enable the CS of the XPT2046.

You can enable and disable the CS using the following sentences.

For the screen display controller (BD663474):

- en_lcd()
- dis_lcd()

For the touch screen controller (XPT2046):

- en_touch()
- dis_touch()
### 4.2.2- SPIfunctions Functions Table:

<table>
<thead>
<tr>
<th>Color Code</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>init_spi</td>
<td>This function allows the user to start the communication between the PIC18F4520 and the WaveShare LCD 22.</td>
</tr>
<tr>
<td></td>
<td>post_data</td>
<td>This function sends a 16 bit word to the screen display controller.</td>
</tr>
<tr>
<td></td>
<td>get_touch_data</td>
<td>This function allows to receive the x or y position from the screen touch controller.</td>
</tr>
</tbody>
</table>

Fig. 4.2.1. SPIfunctions library functions table
4.2.3-SPI functions functions description:

Name of the function

init_spi

Description

This function starts the communication between the PIC18F4520 and the WaveShare LCD 22.

Syntax

void init_spi(void);

Parameter

None

Return Values

None

Precondition

None

Side Effects

None

Example

//Initializing the SPI communication
init_spi();
Name of the function
post_data

Description
This function sends a 16 bit word to the screen display controller through the SPI channel (using 2 byte transfers).

Syntax
void post_data(u16 data);

Parameter
The parameter data can be a 16 bit length register index, instruction or data.

If you want to send a register index you must write the instruction \texttt{en_lcd_index()} before calling the function.
If you want to send an instruction or some data you must write the instruction \texttt{en_lcd_data()} before calling the function.

Both \texttt{en_lcd_index()} and \texttt{en_lcd_data()} instructions are defined at SPIfunctions.h file.

Return Values
None

Precondition
The function \texttt{init_spi} have to be called (only once in the program) before using this function.

The instruction \texttt{en_lcd()} must be called before using this function and the instruction \texttt{dis_lcd()} have to be written after calling this function.

Side Effects
None

Example
//Enables the Chip Select of the screen display controller.
en_lcd();
//Sends a register index
en_lcd_index();
post_data(index);
//Sends a 16 bit data.
en_lcd_data();
post_data(data);
//Disables the Chip Select of the screen display controller.
dis_lcd();
Name of the function
get_touch_data

Description
This function allows to receive the x or y position from the screen touch controller.

Syntax
u16 get_touch_data(u16 cmd)

Parameter
The cmd parameter is the address of the register to be read.
There are only two possibilities (which are defined at SPIfunctions.h file):

- TOUCH_CMD_X The register of the x position
- TOUCH_CMD_Y The register of the y position

Return Values
The returned value is a 16 bit number that represents the x or y position where the screen is touched.

Precondition
The function init_spi have to be called (only once in the program) before using this function.
The instruction en_touch() have to be called before using this function and the instruction dis_touch() must be written after calling this function.

Side Effects
None

Example
//Declares 2 variables of 16 bits.
u16 x;
u16 y;
//Enables the Chip Select of the touch screen controller.
en_touch();
//receives the x position
x=get_touch_data(TOUCH_CMD_X);
// receives the y position
y=get_touch_data(TOUCH_CMD_Y);
//Disables the Chip Select of the touch screen controller.
dis_touch();
4.3- LCD_basic_functions library User Manual:

4.3.1-LCD_basic_functions Introduction Page:

INTRODUCTION

This library provides some basic functions that can be used to control some of the possibilities that the screen display controller (BD663474) offers.

It is recommended to read the appendix C: “Printing on the LCD” to understand how to use some important functions described in this manual.

HOW TO USE IT

First, you will need to place in your working directory the following 2 files:

- LCD_basic_functions.c
- LCD_basic_functions.h

Then, add in your main.c file the following sentence:

#include "LCD_basic_functions.h"

Finally, add the LCD_basic_functions.c file to your source file directory and the LCD_basic_functions.h file into the header file directory of your project.

PREREQUISITES

Because this library uses some of the functions provided by the system and SPIfunctions libraries, you must place the system.c, system.h, SPIfunctions.c and SPIfunctions.h files in you working directory as well as in your source and header file directories.

As it is included in the LCD_basic_functions.h file, you can avoid writing the following sentences:

#include "system.h"
#include "SPIfunctions.h"
#include "LCD_basic_functions.h"

You must follow the same prerequisites explained in the SPIfunctions manual (See page 40)

CALLING THE FUNCTIONS

The first step when you want to use the functions contained in this library is to call the function init_lcd().

This function sets all the configuration bits of the screen display controller (BD663474) and initializes the screen as well as the SPI bus. It already calls the init_spi() function described in the SPI functions manual so you don’t have to call the mentioned function.

The init_lcd() function, is mandatory to be called at the very beginning of your main function because all the functions described in this set of manuals needs this sentence to work properly.

Once this initial function is called, you can use any of the other functions.
## 4.3.2-LCD_basic_functions Functions Table:

<table>
<thead>
<tr>
<th>Color Code</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>lcd_rst</td>
<td>This function executes a reset by hardware of the LCD display.</td>
</tr>
<tr>
<td>Red</td>
<td>post_cmd</td>
<td>This function places a 16 bit word into a defined register of the screen display controller.</td>
</tr>
<tr>
<td>Green</td>
<td>init_lcd</td>
<td>This function sets all the configuration bits of the screen display controller (BD663474) and initializes it.</td>
</tr>
<tr>
<td>Red</td>
<td>SetRAMAddress</td>
<td>This function allows the user to set the GRAM address (and the Address Counter too).</td>
</tr>
<tr>
<td>Red</td>
<td>ResetRAMAddress</td>
<td>This function sets as 0 the GRAM address (and the Address Counter too).</td>
</tr>
<tr>
<td>Red</td>
<td>SetWindowAddress</td>
<td>This function allows the user to select an area from the screen to write (the Address Counter will only point this area following the same order as the full screen mode).</td>
</tr>
<tr>
<td>Red</td>
<td>ResetWindowAddress</td>
<td>This function sets the entire screen as a drawing area.</td>
</tr>
<tr>
<td>Green</td>
<td>ClearLCDscreen</td>
<td>This function prints the entire screen with one color.</td>
</tr>
<tr>
<td>Green</td>
<td>ClearLCDarea</td>
<td>This function prints a selected area of the screen with one color.</td>
</tr>
<tr>
<td>Gray</td>
<td>LCD_test</td>
<td>This function calls other functions of this library to serve as a didactic example.</td>
</tr>
</tbody>
</table>

*Fig. 4.3.1. LCD_basic_functions library functions table*
4.3.3-LCD_basic_functions functions description:

Name of the function
lcd_rst

Description
This function executes a Reset by Hardware of the LCD display.

Syntax
void lcd_rst(void);

Parameter
None

Return Values
None

Precondition
None

Side Effects
None

Example
//Reset the LCD screen
lcd_rst();
Name of the function
post_cmd

Description
This function sends a 16 bit word to the screen display controller through the SPI channel (using 2 byte transfers).

Syntax
void post_cmd(u16 index, u16 cmd);

Parameter
The index parameter defines an address register from the screen display controller.

The cmd parameter contains the data that want to be placed in the selected register (set by the index parameter)

For more information about the registers and which values can contain, read the BD663474 datasheet.

Return Values
None

Precondition
The function init_spi have to be called (only once in the program) before using this function.

Side Effects
None

Example
// Enables the Chip Select of the screen display controller.
en_lcd();
// Sends a 16 bit data word and the index of the register where the data is wanted to be placed.
post_cmd(register, data);
// Disables the Chip Select of the screen display controller.
dis_lcd();
Name of the function
init_lcd

Description
This function sets all the configuration bits of the screen display controller (BD663474) and initializes it.

Syntax
void init_lcd(void);

Parameter
None

Return Values
None

Precondition
None

Side Effects
This function executes a reset by software of the LCD screen (by calling the function lcd_rst.

Also calls the function init_spi which enables the SPI communication.

Example
//Initialization of the LCD screen display controller
init_lcd();
Name of the function
SetRAMAddress

Description
This function allows the user to set the GRAM address (and the Address Counter too).

Note: The GRAM address points to a GRAM (Graphic RAM of the display screen controller) position that represents the pixel in the screen. For more information about the process of printing images on the LCD screen see the appendix C of this manual.

Syntax
void SetRAMAddress(u8 x, u16 y);

Parameter
The \( x \) and \( y \) parameters represent a pixel on the LCD screen. Notice that the screen size is 240x320 pixels so \( x \) must be contained between 0 and 239 and \( y \) have to be contained between 0 and 319.

Note: see the appendix A to know how the screen coordinates are defined.

Return Values
None

Precondition
The function init_lcd have to be called at the very beginning of the main function before calling this function.

Side Effects
None

Example
//Setting the GRAM address as the center of the screen
SetRAMAddress(120, 160);
Name of the function
ResetRAMAddress

Description
This function sets as 0 the GRAM address (and the Address Counter too).

Note: The GRAM address points to a GRAM (Graphic RAM of the display screen controller) position that represents the pixel in the screen. For more information about the process of printing images on the LCD screen see the appendix C of this manual.

Syntax
void ResetRAMAddress(void);

Parameter
None

Return Values
None

Precondition
The function init_lcd have to be called at the very beginning of the main function before calling this function.

Side Effects
None

Example
//Setting the GRAM address as x=0, y=0.
ResetRAMAddress();
Name of the function
SetWindowAddress

Description
This function allows the user to select a square area from the screen to draw (the Address Counter will only point this area following the same order as the full screen mode).

Note: For more information about the process of printing images on the LCD screen see the appendix C of this manual.

Syntax
void SetWindowAddress(u8 initial_x, u16 initial_y, u8 final_x, u16 final_y);

Parameter
The initial_x / initial_y and final_x / final_y parameters represent the top-left and bottom-right pixels of a square area on the LCD screen.
Notice that the screen size is 240x320 pixels so initial_x and final_x must be contained between 0 and 239 and initial_y and final_y have to be contained between 0 and 319.

Note: see the appendix A to know how the screen coordinates are defined.

Return Values
None

Precondition
The function init_lcd have to be called at the very beginning of the main function before calling this function.

Side Effects
None

Example
//Setting the Window address as the top-half screen
SetWindowAddress(0, 0, 239, 160);
Name of the function
ResetWindowAddress

Description
This function sets the entire screen as a drawing area.

Note: For more information about the process of printing images on the LCD screen see the appendix C of this manual.

Syntax
void ResetWindowAddress(void);

Parameter
None

Return Values
None

Precondition
The function init_lcd have to be called at the very beginning of the main function before calling this function.

Side Effects
None

Example
//Resetting the Window address.
ResetWindowAddress();
Name of the function
ClearLCDscreen

Description
This function prints the entire screen with one color.

Syntax
void ClearLCDscreen(u16 color);

Parameter
The parameter color defines one of the 63,536 colors that the LCD screen can display.

Note: to see how are the colors represented see the appendix B.

Return Values
None

Precondition
The function init_lcd have to be called at the very beginning of the main function before calling this function.

Side Effects
None

Example
// Printing all the screen with white color
ClearLCDscreen(0xffff);
Name of the function
ClearLCDarea

Description
This function prints a selected square area of the screen with one color.

Syntax
void ClearLCDarea(u8 initial_x, u16 initial_y, u8 final_x, u16 final_y, u16 color);

Parameter
The initial_x / initial_y and final_x / final_y parameters represent the top-left and bottom-right pixels of a square area on the LCD screen. Notice that the screen size is 240x320 pixels so initial_x and final_x must be contained between 0 and 239 and final_x and final_y have to be contained between 0 and 319.

The parameter color defines one of the 63,536 colors that the LCD screen can display.

Note: see the appendix A to know how the screen coordinates are defined.

Note: to see how are the colors represented see the appendix B.

Return Values
None

Precondition
The function init_lcd have to be called at the very beginning of the main function before calling this function.

Side Effects
None

Example
//Printing the top-left quarter of the screen with balck color
ClearLCDarea(0, 0, 120, 160, 0x0000);
Name of the function
LCD_test

Description
This function uses other functions contained in this library to divide the LCD screen in eight different colored rows.

Syntax
void LCD_test(void);

Parameter
None

Return Values
None

Precondition
The function init_lcd have to be called at the very beginning of the main function before calling this function.

Side Effects
None

Example
//executing the example function of the library Touch_functions
LCD_test();
Appendix A: Screen Coordinates

A lot of functions of this manual use x and y coordinates as a parameters to place different shapes on the screen or as a returned values as the functions of the Touch_functions library.

Before using these functions, you need to know how the coordinate axes are defined.

The used LCD screen, which has 240 pixels of width and 320 of height, has the coordinate’s origin at the top-left corner. The x coordinate is increased from 0 to 239 while crossing the screen from left to right and the y coordinate is increased from 0 on the top to 319 in the bottom.

In the following picture you can see how the coordinate axes are defined and the coordinates of the four edges of the LCD screen.

The following example shows how a square is printed in the screen using the function ClearLCDarea and the explained coordinates system.

```c
//Printing a black square on the screen.
ClearLCDarea(100, 100, 200, 200, 0x0000);

//Notice that the top-left corner is placed at coordinates x=100, y=100 and the bottom-right corner is placed at x=200 and y=200.
```

Fig. A.1. Screen coordinates system

Fig. A.2. Example that shows a square printed using the ClearLCDarea function.
Appendix B: Colors

There are 65,536 available colors in the WaveShare LCD 22 screen but, how can you represent these colors?

The variables which contain the color information of the pixels are stored in 18 bit registers using RGB codification. As you can imagine, the saturation of each component of the RGB codification (Red, Green and Blue) is defined using 6 of the 18 bits. Each RGB component and can take values from 0 to 63.

There is a big problem because, when you send the color information, this information has to be sent through a 16 bit register. This fact implies that the RGB information has to be codified used 16 bits, not 18.

To solve this problem, the 18 data have to miss 2 bits of information. Those bits are the less significant bits of red and blue colors. As a consequence the saturation of the red and blue components can only take 32 possible values instead of the 64 values that the green component can assume.

Now, the color information can be send to the screen using a 16 bit variable and when the information arrives to the color register, the screen automatically transforms the 16 bit data to a 18 bit data by replicating the most significant bits of both red and blue colors and placing those replicated bits to the less significant bits positions.

The following diagram shows how this process happens.

![Color codifying diagram](image)

All the functions of this manual which need a color variable as a parameter, expect the 16 bit codification. It is very convenient to represent this 16 bit word using the hexadecimal notation.

The following table shows some hexadecimal codification for some basic colors:

<table>
<thead>
<tr>
<th>Color</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
<td>0x0000</td>
</tr>
<tr>
<td>Red</td>
<td>0xf800</td>
</tr>
<tr>
<td>Yellow</td>
<td>0xfffe0</td>
</tr>
<tr>
<td>White</td>
<td>0xffff</td>
</tr>
<tr>
<td>Green</td>
<td>0x07e0</td>
</tr>
<tr>
<td>Cyan</td>
<td>0x07ff</td>
</tr>
<tr>
<td>Blue</td>
<td>0x001f</td>
</tr>
<tr>
<td>Magenta</td>
<td>0xf81f</td>
</tr>
</tbody>
</table>
Imagine you want to print a color. You must follow the steps recently explained.

The first step is to select the saturation of the RGB components (between 0 and 63).

We are going to choose:

- Red: 21
- Green: 21
- Blue: 21

Then you have to fill the table with the selected saturations using the binary codification.

![Table showing color codification](image)

**Fig. B.2. Example of color codification**

The parameter you must give to the function is the 16 bit word using hexadecimal codification:

- 0x52aa

The “Final 18 bit” word allows you to see how the initial word has changed.

As you can see, the final saturation values are:

- Red: 20
- Green: 21
- Blue: 20

If you try to codify the RGB colors (21, 21, 21), (20, 21, 20), (21, 21, 20), (20, 21, 21) you will always obtain the same “Final 18 bit”: (20, 21, 20). This is a consequence of the “compression” of the 18 bit word to a 16 bit data.

Be careful when you choose the colors!
Appendix C: Printing on the LCD

There are some functions contained in the `LCD_basic_functions` library which cannot be understood unless the process of printing images on the screen is explained.

Those functions are: `SetRAMAddress`, `ResetRAMAddress`, `SetWindowAddress`, `ResetWindowAddress`.

You may think that, when you want to print an image on the screen, you have to select the coordinates of every single pixel on the screen and then set the color.

That is no true at all. There is a fast and optimized way to set the pixels’ color.

The screen display controller has a register called “address counter” which points the pixel of the screen that will be printed with a color. Every time a pixel’s color is set, this counter is automatically updated and points the next pixel.

This counter is automatically updated from left to right in the screen x axis and up to down when reaches the top x coordinate.

This counter can go through all the screen coordinates or only point the coordinates of a selected square area of the screen.

So, when you want to print any kind of image, firstly you must set the screen area and then set the first value of the address counter.

- The functions `SetRAMAddress` and `ResetRAMAddress` allow the user to select the area of the screen that will be pointed by the Address Counter and printed.
- The functions `SetRAMAddress` and `ResetRAMAddress` allow the user to select a pixel of the square area that is being printed. Is usually used to set the first position of the Address Counter (the top-left corner of the square area that is being printed)

The following image shows how the Address counter is updated:

![Address Counter update](image-url)
To explain this process we are going to show the C code that makes the function ClearLCDarea works.

```c
void ClearLCDarea(u8 initial_x, u16 initial_y, u8 final_x, u16 final_y, u16 color){
    //variables used in the loops
    u8 n;
    u16 m;
    //enabling the chip select of the screen display controller
    en_lcd();
    //setting the area that will be pointed by the Address Counter.
    SetWindowAddress(initial_x, initial_y, final_x, final_y);
    //Setting the first value of the Address Counter.
    SetRAMAddress(initial_x, initial_y);
    //send the index of the writing data register
    en_lcd_index();
    post_data(0x202);
    //enabling the data transference
    en_lcd_data();
    //the variable “color” is send through the SPI channel as many times as pixel has the defined area.
    for(m=0;m<(final_y-initial_y+1);m++)
    {
        for(n=0;n<(final_x-initial_x+1);n++)
        {
            post_data(color);
        }
    }
    //disabling the chip select of the screen display controller
    dis_lcd();
}
```
Appendix D: ASCII codification

The ASCII codification (American Standard Code for Information Interchange) is one of the most classic character codifications.

The ASCII codification associates a number, represented by 8 bits, with a character. There are 32 control characters (from 0 to 31) and 95 printable characters (from 32 to 126).

The following table shows the 95 printable characters of the ASCII notation and the associate number (represented in decimal, hexadecimal and binary notation).

<table>
<thead>
<tr>
<th>DEC</th>
<th>HEX</th>
<th>BIN</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>20</td>
<td>00100000</td>
<td>!</td>
<td>Exclamiation mark</td>
</tr>
<tr>
<td>33</td>
<td>21</td>
<td>00100001</td>
<td>&quot;</td>
<td>Double quotes (or speech marks)</td>
</tr>
<tr>
<td>34</td>
<td>22</td>
<td>00100010</td>
<td>#</td>
<td>Number</td>
</tr>
<tr>
<td>35</td>
<td>23</td>
<td>00100011</td>
<td>$</td>
<td>Dollar</td>
</tr>
<tr>
<td>36</td>
<td>24</td>
<td>00100100</td>
<td>%</td>
<td>Procenttecken</td>
</tr>
<tr>
<td>37</td>
<td>25</td>
<td>00100101</td>
<td>&amp;</td>
<td>Ampersand</td>
</tr>
<tr>
<td>38</td>
<td>26</td>
<td>00100110</td>
<td>'</td>
<td>Single quote</td>
</tr>
<tr>
<td>39</td>
<td>27</td>
<td>00100111</td>
<td>(</td>
<td>Open parenthesis (or open bracket)</td>
</tr>
<tr>
<td>40</td>
<td>28</td>
<td>00101000</td>
<td>)</td>
<td>Close parenthesis (or close bracket)</td>
</tr>
<tr>
<td>41</td>
<td>29</td>
<td>00101001</td>
<td>*</td>
<td>Asterisk</td>
</tr>
<tr>
<td>42</td>
<td>2A</td>
<td>00101010</td>
<td>+</td>
<td>Plus</td>
</tr>
<tr>
<td>43</td>
<td>2B</td>
<td>00101011</td>
<td>,</td>
<td>Comma</td>
</tr>
<tr>
<td>44</td>
<td>2C</td>
<td>00101100</td>
<td>.</td>
<td>Period, dot or full stop</td>
</tr>
<tr>
<td>45</td>
<td>2D</td>
<td>00101101</td>
<td>/</td>
<td>Slash or divide</td>
</tr>
<tr>
<td>46</td>
<td>2E</td>
<td>00101110</td>
<td>0</td>
<td>Zero</td>
</tr>
<tr>
<td>47</td>
<td>2F</td>
<td>00101111</td>
<td>1</td>
<td>One</td>
</tr>
<tr>
<td>48</td>
<td>30</td>
<td>00110000</td>
<td>2</td>
<td>Two</td>
</tr>
<tr>
<td>49</td>
<td>31</td>
<td>00110001</td>
<td>3</td>
<td>Three</td>
</tr>
<tr>
<td>50</td>
<td>32</td>
<td>00110010</td>
<td>4</td>
<td>Four</td>
</tr>
<tr>
<td>51</td>
<td>33</td>
<td>00110011</td>
<td>5</td>
<td>Five</td>
</tr>
<tr>
<td>52</td>
<td>34</td>
<td>00110100</td>
<td>6</td>
<td>Six</td>
</tr>
<tr>
<td>53</td>
<td>35</td>
<td>00110101</td>
<td>7</td>
<td>Seven</td>
</tr>
<tr>
<td>54</td>
<td>36</td>
<td>00110110</td>
<td>8</td>
<td>Eight</td>
</tr>
<tr>
<td>55</td>
<td>37</td>
<td>00110111</td>
<td>9</td>
<td>Nine</td>
</tr>
<tr>
<td>56</td>
<td>38</td>
<td>00111000</td>
<td>;</td>
<td>Colon</td>
</tr>
<tr>
<td>57</td>
<td>39</td>
<td>00111001</td>
<td>;</td>
<td>Semicolon</td>
</tr>
<tr>
<td>58</td>
<td>3A</td>
<td>00111010</td>
<td>&lt;</td>
<td>Less than (or open angled bracket)</td>
</tr>
<tr>
<td>59</td>
<td>3B</td>
<td>00111011</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>3C</td>
<td>00111100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decimal</td>
<td>ASCII</td>
<td>Binary</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------------</td>
<td>----------------------</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>3D</td>
<td>00111101</td>
<td>=</td>
<td></td>
</tr>
</tbody>
</table>
| 62      | 3E    | 00111110   | >  
Greater than (or close angled bracket) |
| 63      | 3F    | 00111111   | ?  
Question mark |
| 64      | 40    | 01000000   | @  
At symbol |
| 65      | 41    | 01000001   | A  
Uppercase A |
| 66      | 42    | 01000010   | B  
Uppercase B |
| 67      | 43    | 01000011   | C  
Uppercase C |
| 68      | 44    | 01000100   | D  
Uppercase D |
| 69      | 45    | 01000101   | E  
Uppercase E |
| 70      | 46    | 01000110   | F  
Uppercase F |
| 71      | 47    | 01000111   | G  
Uppercase G |
| 72      | 48    | 01001000   | H  
Uppercase H |
| 73      | 49    | 01001001   | I  
Uppercase I |
| 74      | 4A    | 01001010   | J  
Uppercase J |
| 75      | 4B    | 01001011   | K  
Uppercase K |
| 76      | 4C    | 01001100   | L  
Uppercase L |
| 77      | 4D    | 01001101   | M  
Uppercase M |
| 78      | 4E    | 01001110   | N  
Uppercase N |
| 79      | 4F    | 01001111   | O  
Uppercase O |
| 80      | 50    | 01010000   | P  
Uppercase P |
| 81      | 51    | 01010001   | Q  
Uppercase Q |
| 82      | 52    | 01010010   | R  
Uppercase R |
| 83      | 53    | 01010011   | S  
Uppercase S |
| 84      | 54    | 01010100   | T  
Uppercase T |
| 85      | 55    | 01010101   | U  
Uppercase U |
| 86      | 56    | 01010110   | V  
Uppercase V |
| 87      | 57    | 01010111   | W  
Uppercase W |
| 88      | 58    | 01011000   | X  
Uppercase X |
| 89      | 59    | 01011001   | Y  
Uppercase Y |
| 90      | 5A    | 01011010   | Z  
Uppercase Z |
| 91      | 5B    | 01011011   | [  
Opening bracket |
| 92      | 5C    | 01011100   | \  
Backslash |
| 93      | 5D    | 01011101   | ]  
Closing bracket |
| 94      | 5E    | 01011110   | ^  
Caret – circumflex |
| 95      | 5F    | 01011111   | _  
Underscore |
| 96      | 60    | 01100000   | `  
Grave accent |
| 97      | 61    | 01100001   | a  
Lowercase a |
| 98      | 62    | 01100010   | b  
Lowercase b |
| 99      | 63    | 01100011   | c  
Lowercase c |
| 100     | 64    | 01100100   | d  
Lowercase d |
<table>
<thead>
<tr>
<th>ASCII Code</th>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01100101</td>
<td>e</td>
<td>Lowercase e</td>
</tr>
<tr>
<td>01100110</td>
<td>f</td>
<td>Lowercase f</td>
</tr>
<tr>
<td>01100111</td>
<td>g</td>
<td>Lowercase g</td>
</tr>
<tr>
<td>01101000</td>
<td>h</td>
<td>Lowercase h</td>
</tr>
<tr>
<td>01101001</td>
<td>i</td>
<td>Lowercase i</td>
</tr>
<tr>
<td>01101010</td>
<td>j</td>
<td>Lowercase j</td>
</tr>
<tr>
<td>01101011</td>
<td>k</td>
<td>Lowercase k</td>
</tr>
<tr>
<td>01101100</td>
<td>l</td>
<td>Lowercase l</td>
</tr>
<tr>
<td>01101101</td>
<td>m</td>
<td>Lowercase m</td>
</tr>
<tr>
<td>01101110</td>
<td>n</td>
<td>Lowercase n</td>
</tr>
<tr>
<td>01101111</td>
<td>o</td>
<td>Lowercase o</td>
</tr>
<tr>
<td>01110000</td>
<td>p</td>
<td>Lowercase p</td>
</tr>
<tr>
<td>01110001</td>
<td>q</td>
<td>Lowercase q</td>
</tr>
<tr>
<td>01110010</td>
<td>r</td>
<td>Lowercase r</td>
</tr>
<tr>
<td>01110011</td>
<td>s</td>
<td>Lowercase s</td>
</tr>
<tr>
<td>01110100</td>
<td>t</td>
<td>Lowercase t</td>
</tr>
<tr>
<td>01110101</td>
<td>u</td>
<td>Lowercase u</td>
</tr>
<tr>
<td>01110110</td>
<td>v</td>
<td>Lowercase v</td>
</tr>
<tr>
<td>01110111</td>
<td>w</td>
<td>Lowercase w</td>
</tr>
<tr>
<td>01111000</td>
<td>x</td>
<td>Lowercase x</td>
</tr>
<tr>
<td>01111001</td>
<td>y</td>
<td>Lowercase y</td>
</tr>
<tr>
<td>01111010</td>
<td>z</td>
<td>Lowercase z</td>
</tr>
<tr>
<td>01111011</td>
<td>{</td>
<td>Opening brace</td>
</tr>
<tr>
<td>01111100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01111101</td>
<td>}</td>
<td>Closing brace</td>
</tr>
<tr>
<td>01111110</td>
<td>~</td>
<td>Equivalency sign – tilde</td>
</tr>
</tbody>
</table>

**Fig. D.1. ASCII codification table**
Appendix E: TextFonts

The text fonts are represented with a one-dimensional array which contains a set of 8 bit variables.

In this project, this array is contained in the `Asclihex8x16.h` file.

Each character is codified using a set of 8 bit variables where each bit represents a pixel. If a bit is set a 1, the associate pixel belongs to the character and if the bit is set as 0, the associate pixel belongs to the background.

If the character’s width is less than 8 bits, the rows of pixels that compose the characters are represented using 8 bits variables and if the character’s width is larger than 8 pixels, the 8 bits variables just represent part of a row.

Because of that, each character occupies as much places inside the text font array as the result of multiplying the height of the character (in pixels) by the number of 8 bit variables used to codify a single row.

Furthermore, the set of 8 bit variables used to codify a character has to be placed inside the text font array following the order established by the ASCII codification.