UNICA UNIVERSITÀ DEGLI STUDI DI CAGLIARI

This is the Author's *accepted* manuscript version of the following contribution:

A. G. Gonzalez-Rodriguez, E. Ottaviano and P. Rea, "Libraries and Tools for the Design of a GUI on a Touch Screen Controlled by ESP32", *2024 XVI Congreso de Tecnología, Aprendizaje y Enseñanza de la Electrónica (TAEE)*, Malaga, Spain, 2024, pp. 1-7.

© 2024 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

The publisher's version is available at:

http://dx.doi.org/10.1109/TAEE59541.2024.10604979

When citing, please refer to the published version.

This full text was downloaded from UNICA IRIS https://iris.unica.it/

Libraries and tools for the design of a GUI on a touch screen controlled by ESP32

Angel G. Gonzalez-Rodriguez
Dept. of Electronic Engineering and Automation
University of Jaén
Jaén, Spain
0000-0002-7461-9135
agaspar@ujaen.es

Erika Ottaviano
Dept. of Civil and Mechanical Engineering
University of Cassino and Southern Lazio
Cassino, Italy
0000-0002-7903-155X

Pierluigi Rea
Dept. of Mechanical, Chemical and Materials Engineering
University of Cagliari
Cagliari, Italy
0000-0003-0517-394X

The objective of this work is the development of a set of tools to help in the design of a GUI on a touch-screen connected to a microcontroller, specifically an ESP32. The set of tools contains the necessary templates for the location and configuration of the different elements that make up the interface, such as polygons, editable text boxes, labels, check-boxes, button-radio and buttons. For responsive elements, it also allows the definition of the type of interaction with the user: call-back functions, modified variables or entered texts. To facilitate the tedious task of designing an interface, a library has been developed for desktop projects, which can be compiled and debugged on a PC or Laptop and which allows testing 95% of the interface's functionalities, but using a much faster and more comfortable development environment.

***Index Terms*—GUI, ESP32s, Graphical User Interface, touch-screen, cross-deployment, friendly design, C++**

I. Introduction

Thanks to the decreasing costs and improved performance of touchscreens, they are increasingly integrated into embedded systems. This trend has been facilitated by the enhanced speed of newer generations of 32-bit microcontrollers, enabling higher screen refresh rates that accelerate the transition between different windows. Among these microcontrollers, two of the most popular are ESP32 and STM32.

The former is a robust and versatile system-on-chip (SoC) developed by Espressif Systems. It integrates a dual-core Xtensa LX6 CPU with Wi-Fi and Bluetooth capabilities, rendering it well-suited for a wide array of Internet of Things (IoT) applications, spanning from smart home devices to industrial automation systems [1].

As for the STM32 microcontroller, engineered by STMicroelectronics, there exists a diverse range of models tailored to various application needs. Ranging from low-power devices like the STM32L series, which are ideal for battery-operated systems, to high-performance models such as the STM32H series, suitable for demanding applications like motor control and digital signal processing. The STM32 microcontroller series offers comprehensive debugging support, including JTAG (Joint Test Action Group) or SWD (Serial Wire Debug) protocols [2]. This enables efficient debugging of embedded applications during the prototyping phase. These capabilities render the STM32 series a favoured choice among embedded system designers globally.

From a technical standpoint, developing a Graphical User Interface (GUI) on a touchscreen does not pose a particular difficulty once communication with the screen for drawing and capturing touch events has been established. However, this does not negate the need for methodological study of window organization, distribution of user interaction components, their positioning, colors, information filtering, and so forth.

Shneiderman et al. in [3] delineate eight golden rules of interface design and offer practical guidelines with examples to facilitate interface navigation for new users. Meanwhile, Weyers et al. in [4] survey existing formal methods in Human-Computer Interaction for modelling, validation, and simulation, as an extension of GUI.

Although designing a user interface is not particularly challenging from a programming perspective, it is a labor-intensive process until a result meeting the design requirements is achieved. Without a wizard, designers must define the shape, position, and interaction mechanism for each control from scratch. This entails time and effort beyond the task of programming the control function to be executed by the microcontroller.

Furthermore, multiple tests are typically necessary to verify the appearance and proper functioning of each interface

element. Each test requires editing, compilation, uploading, and testing. In case of unexpected operation, debugging tools are necessary to locate errors. The tools commonly used for ESP32 programming, such as Arduino IDE and VS Code + Platform IO extension, are cumbersome and time-consuming for testing and debugging code modifications, potentially extending in time the design process.

A typical application for TFT driven by microcontrollers is as an oscilloscope or spectral analyser, as shown in [5], where the authors employ a development board mounting an ATMega 328 with TFT. Similarly, a multifunction oscilloscope is presented in [6], utilizing an STM32F407 connected to a 3.5-inch TFT with GUI designed using the ucGUI graphical support system.

Xu et al. in [7] connect a 7-inch TFT-LCD and a 32-bit microcontroller MB91F599 to process and display valuable information from a vehicle. Ünsalan et al. in [8] use a development board mounting an STM32F4 and a touchscreen to create a GUI with the help of TouchGFX, software provided by STMicroelectronics for GUI design.

Another typical application of GUIs using microcontrollers is in home automation, as demonstrated in [9], which employs an RX63N microcontroller to control various devices such as fans, lights, and TVs.

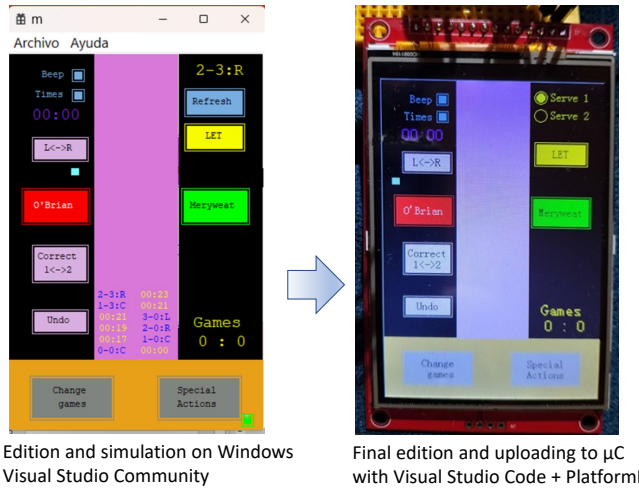
Currently, there are other applications to aid in interface design, such as LVGL [10], offering over 30 built-in Widgets with appealing aesthetics and portable code to platforms like STM32 or ESP32. In the latter case, it allows creating a project compatible with ESP-IDF accessible from Visual Studio Code with the PlatformIO extension (VSC+PIO), albeit on the ESP-IDF framework.

Another option is TouchGFX [11], an advanced graphic software framework optimized for STM32 microcontrollers, providing a gallery of ready-to-use themes, backgrounds, and visuals distributed as a standalone software tool. Finally, μGUI [12] enables the design and simulation of GUIs and is compatible with any commercial RTOS or proprietary operating system. The dialogs can be saved as C files, hindering the use of object-oriented programming (OOP).

The library provided and explained in this work may not be as ambitious in terms of the range of supplied widgets as the aforementioned applications (lacking, for example, sliders, graphics, or meters), or in visual appearance as TouchGFX. However, it can be directly added to the user's C++ project and is directly associated with more agile editing and compilation tools, such as Visual Studio Community IDE (VSCI). This allows for robust compilation for rapid visualization of the interface's appearance and event-based actions.

The objectives and contributions of this work are:

- Create a C++ library with the main components of a user interface, generic for any microcontroller. There are previous works, such as [13], incorporating plot functions (scatter, bar and line) but they are limited in terms of components and functionalities.
- Creation of an environment in VSCI, more comfortable for editing and debugging, that can include this library thus allowing the definition and testing of an interface on a PC or laptop. This environment does not require connection to the final hardware (see left-hand image of Figure 1).
- Definition of a protocol to insert the tested code created on the desired microcontroller platform, which in this case will be an ESP32 (see right-hand image of Figure 1).



Edition and simulation on Windows Visual Studio Community

Final edition and uploading to µC with Visual Studio Code + PlatformIO

Fig. 1. At the left, previsualization of the GUI (one of the windows) as given by the design/simulation tool. At the right, final appearance of this window, as seen in the physical TFT.

The libraries, a complete example, and a template to be modified by the designer, are available at [14].

The remaining of the paper is structured as follow: Section II describes the set of components offered by the proposed library; Section III describes the structure of the project; Section IV introduces an example of GUI designed with this procedure; finally, Section V releases the conclusions and discusses some issues observed along the preparation of the work.

II. Components

Part of the GUI library layout includes the generic definition of components or widgets provided for use by the user in their specific design. These are as follows:

Label. Contains a single-line text string, with no possibility of runtime change. It can be seen in all example figures (Figs. 2, 3, 4, and 5).

Textview. Similar to a label, but its content can be changed at runtime. It supports multiple lines (Fig. 4 and 5).

Button. Responsive component that must be associated, through a callback function assigned to a component property, with the action to be performed when the button is pressed. Similarly, it can be seen in all example figures (Figs. 2, 3, 4, and 5). Its state can be defined as Enabled/Disabled and Visible/Invisible.

Textbox. Responsive component similar to the button, but allows for text within it to span multiple lines, adjusting the button height based on the number of lines (Fig. 3).

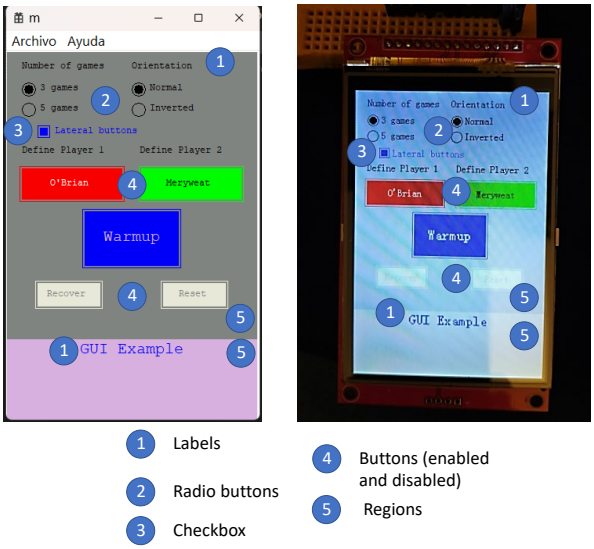


Fig. 2. A window of the user defined GUI with coloured regions, labels, buttons, a checkbox and radio buttons.

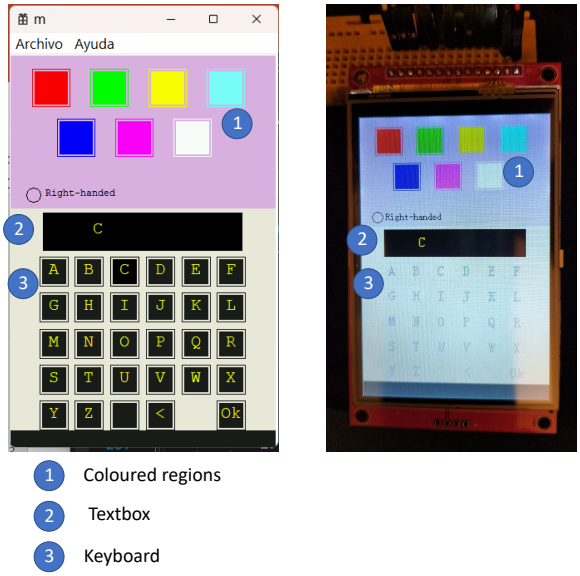


Fig. 3. A window of the user defined GUI with textboxes, coloured regions and the keyboard.

Checkbox. Responsive component with boolean behavior. It can be defined whether the accompanying text appears on the left or right (Fig. 2).

Radio button. Responsive component consisting of several mutually exclusive selectable items (Fig. 2).

Coloured region. Simply a colored rectangle, often used to differentiate parts within the same window. It is defined by the XY coordinates of the definition point, and by the justification applied; for example, if the justification is centered vertically and horizontally, XY corresponds to the center of the region, but if the justification is Upper-Left, then XY corresponds to the upper-left corner. It can be seen in all example figures (Figs. 2, 3, 4, and 5).

Warning message. A popup message that requires an immediate response from the user before continuing with normal behavior (Fig. 4).

In Table I, the properties of each component are presented. The significance of each row is described below, with a particular focus on the meaning of the symbol ✓:

- XY + Justification: ✓if the position of the component is given by coordinates X and Y together with the type of justification: left, centre, right for horizontal; and top, centre (no bottom) for vertical.
- Left/Top: ✓if the position of the component is given by the left and top coordinates of the checkbox or the first radio button.
- Size definition:
  = *Text* if the width and height is given by the dimensions of the text string;
  = *H/W* if they are given explicitly;
  = *Text/W* if the length is given explicitly and the height is calculated from the font size.
- Text position R/L: ✓if the text can be located at the left or the right of the checkbox.

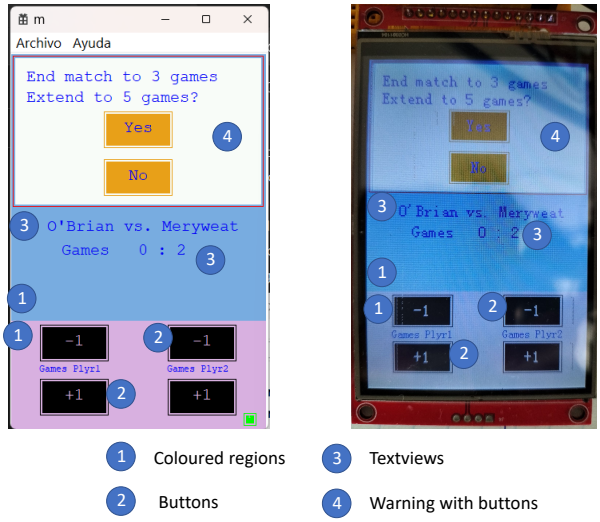


Fig. 4. A window of the user defined GUI with a warning message with buttons, coloured regions, buttons and textviews.

- Font size: ✓if the font size can be selected.
- Modifiable text: ✓if the text can be modifiable in runtime.
- Text single line: ✓if only one line for the text is allowed.
- Text multiline: ✓if multiple lines are allowed.
- Enabled (Yes/No): ✓if it is possible to enable or disable the component responsiveness, thus modifying its colour and impeding the eventually associated action event.
- Visible (Yes/No): ✓if it is possible to enable or disable the component visibility, also impeding the associated action event.
- Text colour: ✓if it is possible to select the text colour.
- Back colour: ✓if it is possible to select the component background colour.

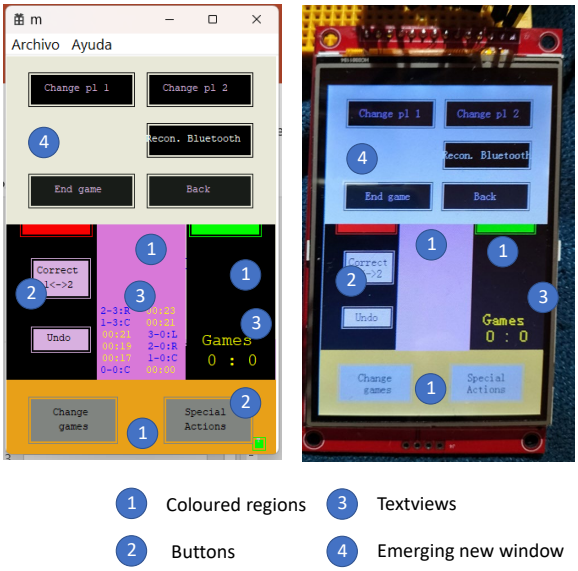


Fig. 5. A window of the user defined GUI with an emerging new window with buttons.

- Action: ✓ if the component has an associated action event when it is pressed.
- Additional property:
  = *previous width and height* if these values are stored for a more efficient deletion;
  = *is checked* to retrieve the status of a checkbox;
  = *selection* to retrieve which radio button is selected.

The warning & popup message is not included in the table. For its definition, two text string must be assigned, the number of buttons that the warning contains, and the texts accompanying each button.

III. Structure of the Projects

As mentioned earlier, designing an interface is laborious and requires numerous tests to adjust positions, sizes, colors, function distribution among different windows, event behavior, etc. Conducting all necessary tests to achieve the desired final result demands a significant amount of compilation time and error debugging. In the case of platforms like STM32, debugging is possible, and compilation and loading times are moderate. However, for platforms like Arduino or ESP32, these times are much longer, and debugging is not feasible for most models. In any case, a connection to the physical microcontroller is required, and from it to the touchscreen, which often introduces additional errors due to the possibility of cable disconnection if not working with an already assembled prototype.

Therefore, the use of a more agile and faster cross-development tool is highly convenient to reduce time and effort in the design and fine-tuning of a GUI. In this work, a tool is provided in the form of a library along with a design template and examples with two projects: one to be compiled and executed from VSCI, and another to be compiled and downloaded from VSC+PIO.

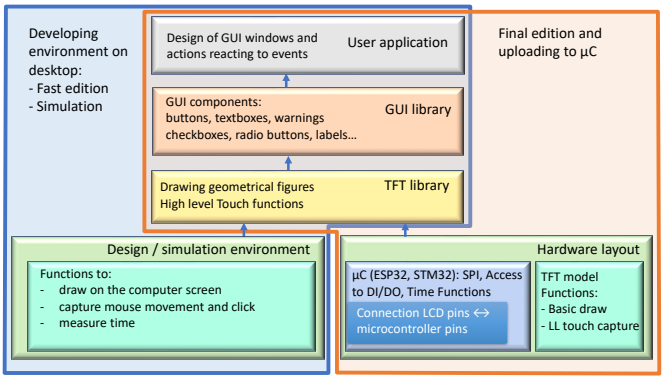


Fig. 6. Structure of the two projects organized in layouts: at the left, outlined in blue, hierarchy of layouts for the project in VSCI for the design and simulation; at the right, outlined in brown, hierarchy for the project in VSC+PIO for the final adjustments and uploading to the microcontroller.

As depicted in Figure 6, both projects share the geometric shapes library (highlighted in yellow) and high-level functions for capturing touch positions (or mouse selection). In the figure, this is labelled as the TFT library. The geometric shapes library uses basic drawing and filling functions to draw and fill (if possible) lines, circles, triangles and rectangles. The high-level functions for capturing touch position transform the screen coordinates as given by the hardware (TFT pulsation or mouse) into XY coordinates taking into account offsets or rotations.

The GUI library (highlighted in salmon) is built on top of this library with the generic definition of the components described in Section II.

This library is utilized by the upper layer (in gray), which contains the specific user application along with the definition of each GUI window. The figures shown in this work are related to a squash scoreboard, which is the specific user application employed as an example. This will be the only layer that the user needs to modify once the target platform and TFT screen are defined. Specifically, this library has been tested on an ESP32 microcontroller and is being prepared for STM32 (models STM32F103 on a *Blue pill* and STM32F407 on a *Black pill*), using a TFT with a touch screen, model 3.5inch SPI Module ILI9488 SKU:MSP3520 [15].

At a lower level, the TFT library relies on different functions, which differ between the design (VSCI) and uploading (VSC+PIO) environments. Regarding the design environment on VSCI, several functions have been created or adapted to draw pixels and fill rectangles on the PC screen, manage mouse click capture, and measure times. Regarding the compilation and loading environment using VSC+PIO, the following must be defined:

Microcontroller

- Functions to write and read digital outputs and inputs respectively.
- Functions to manage the SPI.
- Functions to measure milliseconds and wait for delays.

TABLE I
SUMMARY OF COMPONENT PROPERTIES

	Label	TextView	Button	TextBox	CheckBox	RadioButton	Region
XY + Justification	✓	✓	✓	✓			✓
Left/Top					✓	✓	
Size definition	Text	Text	H/W	Text/W	Text	Text	H/W
Text position					Right/Left		
Font size	✓	✓	✓	✓	✓	✓	
Modifiable text		✓	✓	✓	✓	✓	
Text single line	✓		✓	✓	✓		
Text multiline		✓		✓		✓	
Enabled (yes/no)			✓	✓	✓	✓	
Visible (yes/no)	✓	✓	✓	✓	✓	✓	
Text colour	✓	✓	✓	✓	✓	✓	
Back colour			✓	✓			✓
Action			✓	✓	✓	✓	
Additional property	previous width and height				is checked	selection	

Additionally, the designer must specify which pins of the microcontroller are connected to the pins of the TFT.

TFT model

- Functions for the configuration and initialization of the TFT according to the instructions given by the manufacturer.
- Basic functions to draw pixels and fill rectangles.
- Basic functions to capture touch positions of the TFT screen.

These functions have been adapted from similar ones obtained from [15].

The layered organization allows for their reuse across different microcontrollers and TFT screens.

A. Structure of the GUI program

In this work, the structure serving as the foundation for GUI design has been programmed. This structure comprises the previously mentioned libraries, *TFT library* and *GUI library*. Utilizing these, the GUI designer can code their own interface, adding actions to be performed upon, for example, pressing the buttons.

To facilitate the creation of a new GUI, template files have been added that will be modified by the designer according to their requirements. In the project, these files are named `Template_window1.cpp`, `Template_window2.cpp`, along with the header file `includes_template.h`. The following will display the most significant details of these files. A detailed explanation of the code is beyond the scope of this work. The complete listing can be found in [14].

1) Content of the User Source Files: The designer must modify the content of the `Template_windowX.cpp` files and, optionally, also their names.

Shown below is an example code snippet that allows the designer to include the necessary buttons and configure their appearance and associated action. As can be seen, there are indications of the code parts that can be modified by the designer.

```
/* - COMPONENT DEFINITIONS FOR THIS GUI ----*/
static button_t THESE_BUTTONS[] =
// ---------- User code begin  --------
{
  {action_1_window1, (u8*)"Bt1_W1", {Center, 70,
      100, 100, 40, YELLOW}, BLACK, 16, ENABLED,
      VISIBLE},
  {action_2_window1, (u8*)"Bt1_W1", {Center, 70,
      150, 100, 40, ORANGE}, BLACK, 14, ENABLED,
      VISIBLE},
  {action_3_window1, (u8*)"Bt1_W1", {Center, 220,
      135, 140, 60, PURPLE_RED}, BLACK, FONT24,
      ENABLED, VISIBLE}
  // -------- User code end    --------
};

typedef enum
{
  // ------- User code begin  ---------
  en_wd1_bt_1,
  en_wd2_bt_1,
  // --------- User code end    --------
} THESE_ENUM_BUTTONS;
```

Logically, the actions associated with the buttons, in this case, `action_X_window1`, must be previously defined or declared. For example, if pressing the button `Bt1_W1` leads to `Window2`, then the event would be coded as follows:

```
void action_1_window1(void)
{
  Launch_window2();
}
```

where, in this instance, `Launch_window2` is the main function within the file `Template_window2.cpp` that manages the appearance and behavior of the interface for this window. The instructions in this function are common to any window, and the user only needs to modify its name, as shown in the following snippet.

```
// -------- User code begin  ----
```

```
// Modify function name to your convenience
void Launch_window2()
{
  // ------ User code end  --------
  UPDATE_ACTIVE_WINDOW;

  Reset_numbers_GUI();

  // Initialize buttons

  gui.buttons = THESE_BUTTONS;
  gui.num_buttons = sizeof(THESE_BUTTONS) / sizeof(
      button_t);

  ... Analogous code_for the other components ...

  Detect_void_arrays(gui);

  Operations_pre_initialize();
  Initialize_active_window();
  Operations_post_initialize();
}
```

At the end of this code snippet, three functions are called. The second one is responsible of drawing the components, and must not be modified. The first and third functions determine the operations to be performed before and after drawing the components and, if necessary, can be modified by the user.

2) Content of the file main.cpp: When editing the project in VSC+PIO for compilation and uploading to the microcontroller, it is necessary to modify the name of the function that initiates the first GUI window. Below is the `main.cpp` file:

```
#include "includes_TFT.h"
#include "includes_GUI.h"
// ---------------- User code begin  --------------
#include "includes_Template.h"
// 1st window of the GUI
#define LAUNCH_FIRST_WINDOW_OF_MY_GUI()
    Launch_window1()
// ----------------- User code end  ---------------

void SPI_Init(void) {
  SPI.begin();
  SPI.setBitOrder(MSBFIRST);
  SPI.setDataMode(SPI_MODE0);
  SPI.setFrequency(80000000);
}
void setup() {
  SPI_Init();
  LCD_Init();
  TP_Init(); // Calibrate the touch screen

  LAUNCH_FIRST_WINDOW_OF_MY_GUI();
}

void loop() {
  TP_Scan(true);
  if (Get_TP_status() & TP_PRES_DOWN) {
    point16 pt16;
    Get_TP(&pt16);
    if (pt16.x < LCD_WIDTH && pt16.y < LCD_HEIGHT)
        {
      Manage_touch(pt16);      }
  } else  {
    Sleep(10);
  }
}
```

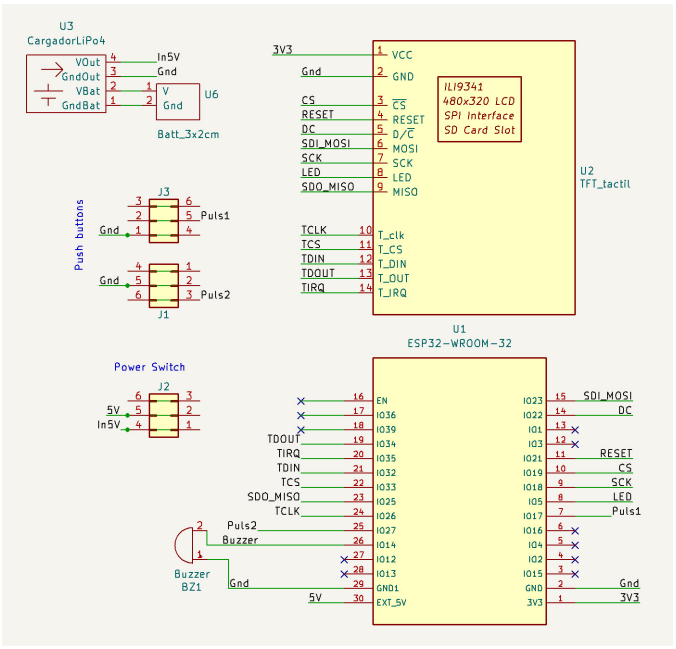


Fig. 7. Schematic of the connections between an ESP32 microcontroller and a 3.5-inch touchscreen 480x320 with a ILI9341/9488 controller

3) Content of the Main File of the VSCI Project: The name of this file will depend on the name given to the project. Its content is more extensive and will not be displayed here. In any case, its content should not be modified, except to assign the first function to be called to launch the first interface, with a code snippet similar to that seen for `main.cpp`.

```
// ---------------- User code begin
    --------------
#include "includes_Template.h"
// 1st window of the GUI
#define LAUNCH_FIRST_WINDOW_OF_MY_GUI()
    Launch_window1()
// ----------------- User code end
    ---------------
```

IV. Test in a Real Application

This combined project (within the design/simulation and compilation/uploading environments) has been tested to create an interface managing a scoreboard for squash matches. The application allows defining the match characteristics, player details, time counting, and recording results (points and sets), among other functionalities. The various windows have been displayed in Figures 2, 3, 4, and 5. The application's code can be found in [14].

Figure 7 shows the schematic (edited with KiCAD) for the connections to be made between an ESP32 microcontroller and a 3.5-inch touchscreen 480x320 with a ILI9488 controller.

The schematic also includes a battery charger powered from a mini USB port, a housing for a 3.7V LiPo battery, and a power switch. For the specific scoreboard application, two push buttons and a buzzer are also included. Source files

together with the schematic and PCB layout are available at [14].

V. Conclusion

In this work, the structure of two projects that significantly expedite the design process of a GUI using a microcontroller and a touchscreen has been presented. These projects share a library of controls or widgets for designing the interface. One of the projects, within the VSCI environment, facilitates the interface design and its behavior simulation. Subsequently, the design and functionalities programmed by the user are transferred to another project, in the VSC+PIO environment, which allows for the final compilation and uploading to the physical microcontroller.

The project is demonstrated for an ESP32 microcontroller and a touchscreen with an ILI9488 controller. However, the layered structure of the libraries enables the use of other hardware components. In fact, this project was initially developed for STM32, which indicates that the proposed template can be used with almost any microcontroller (fast enough), although it is required to modify the functions and `defines` described in the block *Hardware layout* in Figure 6.

The outcome of this work is provided as two projects available in the repository [14], with an example of a squash scoreboard, and an almost empty template ready to be customized by the user.

For a non-trivial application, as in this case, the use of a more agile compilation tool, with simulation capabilities, has resulted in a significant time-saving compared to what would have been achieved if all coding had been done directly on the microcontroller connected to the TFT. This ability to debug, execute, and display in a desktop environment is the main quality of the proposed tool compared to other existing solutions. The main disadvantage is its limited set of controls or widgets and the lack of a parallel application to position components on the design canvas with the mouse.

Another disadvantage of a GUI application is the time required to refresh the screen or change the state of a button or area. This time is necessary, regardless of which tool was used to design the GUI, although the time taken will be greater the larger the number of pixels involved, with a lower limit in the case of refreshing the entire screen. In the case of redrawing only a button or area, the proposed utility is designed to isolate the rectangle to be redrawn in order to minimize the redraw time. During this time, it will not be possible to perform any kind of control unless it is carried out using timed interrupts (as usual in continuous control) or external interrupts.

For future works, the inclusion of additional widgets, for example, for displaying graphs, is considered, as well as completing the adaptation to STM32, particularly regarding the SPI protocol.

Acknowledgment

This work has been funded by the University of Jaén, as part of the Teaching Innovation and Improvement Plan PIMED-UJA 2019-2023 (PIMED-UJA 2022 call).

References

[1] V. Pravalika, C. R. Prasad *et al.*, "Internet of things based home monitoring and device control using esp32," *International Journal of Recent Technology and Engineering*, vol. 8, no. 1S4, pp. 58–62, 2019.

[2] F. Wang, W. Z. Lou, M. R. Guo, and Y. F. Lu, "Intelligent logistics monitoring microsystem based on stm32," *Key Engineering Materials*, vol. 645, pp. 896–899, 2015.

[3] B. Shneiderman, C. Plaisant, M. Cohen, S. Jacobs, N. Elmqvist, and N. Diakopoulos, *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, 6th ed. Pearson, 2016.

[4] B. Weyers, J. Bowen, A. Dix, and P. Palanque, *The Handbook of Formal Methods in Human-Computer Interaction*. Springer Cham, 2017. [Online]. Available: https://doi.org/10.1007/978-3-319-51838-1

[5] F. Dragan, R. Holonec, and R. Copindean, "Data acquisition system with tft graphic interface," in *Acta Electrotehnica 94*, M. S. Publisher, Ed., 2014.

[6] X. Zhou, Y. Tian, J. Shi, N. Zhao, and Y. Yang, "Design of low-power portable multi-function digital oscilloscope," in *2021 IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, vol. 4, June 2021, pp. 2101–2105. [Online]. Available: 10.1109/IMCEC51613.2021.9482318

[7] Y. Xu, S. Yin, J. Yu, and H. R. Karimi, "Design of a tft-lcd based digital automobile instrument," *Mathematical Problems in Engineering*, vol. 2014, p. 549790, 2014. [Online]. Available: https://doi.org/10.1155/2014/549790

[8] C. Ünsalan, H. D. Gürhan, and M. E. Yücel, *LCD, Touch Screen, and Graphical User Interface Formation*. Cham: Springer International Publishing, 2022, pp. 415–454. [Online]. Available: https://doi.org/10.1007/978-3-030-88439-0_11

[9] N. P. Gireesh and Deepa, "Designing hmi using rx series rx63n microcontroller," in *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, May 2017, pp. 1049–1053. [Online]. Available: 10.1109/RTEICT.2017.8256759

[10] LVGL (light and versatile graphics library). Accessed March 2024. [Online]. Available: https://docs.lvgl.io/master/intro/index.html

[11] TouchGFX documentation, 2024. Accessed March 2024. [Online]. Available: https://support.touchgfx.com

[12] μGUI. Accessed March 2024. [Online]. Available: https://embeddedlightning.com/ugui/

[13] S. Dreborg, "Embedded gui library development." [Online], Accessed March 2024, Uppsala Universitet, 2022, accessed March 2024. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:1651032/FULLTEXT01.pdf

[14] A. Gonzalez, E. Ottaviano, and P. Rea, "GUI ESP32," Mendeley Data, V1, 2024. [Online]. Available: 10.17632/k6ckh38bj5.1

[15] (2021) 3.5inch SPI module ILI9488 SKU:MSP3520. Accessed March 2024. [Online]. Available: http://www.lcdwiki.com/3.5inch_SPI_Module_ILI9488_SKU:MSP3520