TFT Screen Library ST7735 (SPI Version)

Contreras R Orlando[1,2*] and Lara H Kevin[1,2*]

[3]Department of Electronic Systems, UAA, Av. Universidad 940, Aguascalientes, 20131, Aguascalientes, México.

*Corresponding author(s). E-mail(s): {al348390,al464376}@edu.uaa.mx;

Abstract

The objective of this TFT_LCD screen driver is implement an easy way to show info on a screen. This is very useful in embedded systems applications, like show images, text, or anything that requires a screen.

Keywords: ARM, C++, Driver, TFT Screen.

1 Introduction

Communication protocols are very useful in a lot of tech applications; all computers, micro controllers, and every circuit with a processor need communication protocols to send or receive I/O data and control external peripherals, some of the most used protocols are SPI. Using this application, the ST7735 TFT screen is a device able to print pixels on a point-matrix, sending via SPI the column and row address to set the internal screen pointer on the pixel that we want to turn on, subsequently we send the color data of this pixel in the color format RGB565.

2 Description

The library for the ST7735-based TFT display provides a simple and efficient interface for handling graphics through the SPI communication protocol.

This library abstracts the complexity of the command sequences and internal operation of the ST7735, offering intuitive functions to initialize the display, print color rectangles,fill the creen with one color, draw pixels, and test the colors on the screen, each one integrated on a 4 simple function to use:

Screen1.FillScreen(color).

Screen1.FillRectangle(posx, posy, width, height, color).

Screen1.DrawPixel(posyx, posy, color).

test(void);

3 Purpose

This library aims to support rapid development of visual interfaces, diagnostic screens, and real-time graphical outputs, offering a set of functions optimized for performance and compatibility with a wide range of microcontrollers. It is designed to serve both beginners and advanced users who require a practical and flexible tool to integrate TFT graphics into their projects.

4 How to use properly

This section explains how to integrate and use the ST7735 TFT driver and the oscilloscope demo on an STM32F446 MCU.

4.1 Driver Initialization

1. Instantiate the driver object (constructor calls low-level `config()` for RCC/GPIO/SPI).
2. Call `INIT_FN()` once after reset to send the ST7735 init sequence.
3. Clear or paint the screen using `FillScreen(color)`.

Example:

```
TFT_ST7735 tft;
tft.INIT_FN();
tft.FillScreen(COLOR_BLACK);
```

4.2 Drawing Primitives

- Pixel: `DrawPixel(x, y, color)`
- Rectangle: `FillRectangle(x, y, w, h, color)` (auto clipping)
- Full screen: `FillScreen(color)`
- Text: `WriteChar(x, y, ch, font, fg, bg)` and `WriteString(x, y, str, font, fg, bg)`

Colors use RGB565; predefined constants (`COLOR_RED`, `COLOR_BLUE`, etc.) live in the header.

4.3 Oscilloscope Demo Flow

Provided in `main_osc.cpp`:

1. Configure PLL and system clock via `conf_osc()`.
2. Enable and start continuous conversion on ADC1 channel at `PA0`.
3. Initialize TFT and draw axes using rectangles.
4. In loop: read `ADC1->DR`, scale value (`>>5`) to horizontal pixel range, increment vertical time index, plot with `DrawPixel()`.
5. When vertical index reaches bottom, clear plot region and restart for a scrolling effect.

4.4 Build Notes

The code uses bare-metal register access (no HAL). Ensure your linker script and startup file match STM32F446. SPI baud is set to fPCLK/16. If the display shows nothing, verify wiring and that HSE/PLL configuration succeeds.

4.5 Testing

1. Power the board and TFT (3.3V only).
2. Flash firmware containing either `main_tst.cpp` (color test) or `main_osc.cpp` (oscilloscope).
3. For oscilloscope: vary the analog input on PA0 (potentiometer or waveform source) and observe horizontal deflection of the cyan trace.

5 Diagram

In the figure 1 the test circuit with the test code printing multiple rectangles, notice that the connections are in the table 5.

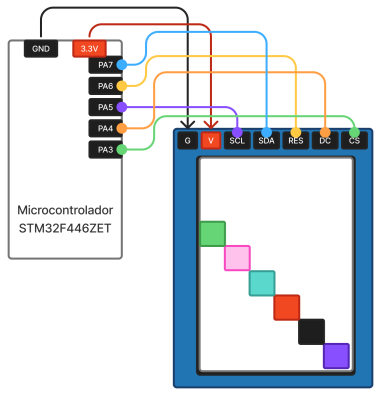

Fig. 1: Schematic

TFT Pin	STM32F446 Pin	Purpose
VCC	3.3V	Power
GND	GND	Ground
SCK	PA5	SPI1 SCK (AF5)
MOSI	PA7	SPI1 MOSI (AF5)
CS	PA3	Chip Select (GPIO)
DC	PA4	Data/Command select (GPIO)
RST	PA6	Hardware reset (GPIO)

Optional: connect a variable analog signal (e.g. potentiometer wiper) to `PA0` for the oscilloscope demo (ADC input).

6 Limitations and Enhancements

6.1 Current Limitations

- **Polling SPI**: All transfers busy-wait; CPU cannot perform other tasks concurrently.
- **No DMA**: Large area fills and waveform plotting are slower than possible; higher CPU usage.
- **Fixed Rotation**: Only one memory access orientation initialized; no runtime rotation switching.
- **Limited Primitives**: Lines, circles, bitmaps, and sprites are not yet implemented.
- **No Text Clipping**: Long strings may draw off-screen without wrapping or truncation.
- **Oscilloscope Scaling**: Simple right-shift scaling (`>>5`) reduces resolution; no trigger or persistence.
- **Blocking Clear**: Screen clear for oscilloscope resets full plot; no partial scrolling buffer.

6.2 Potential Enhancements

- **SPI DMA + Interrupts**: Offload pixel bursts; increase frame throughput.
- **Drawing Toolkit**: Add line (Bresenham), circle (Midpoint), bitmap blitting, and gradient fills.
- **Runtime Rotation**: Implement MADCTL updates with offset correction and cached clipping.
- **Text System**: Clipping region, wrapping, proportional fonts, and transparency support.
- **Waveform Features**: Trigger modes (edge/level), adjustable vertical/horizontal scaling, persistence with ring buffer.
- **Double Buffering**: Compose frames off-screen then push for flicker-free updates.
- **UI Layer**: Basic widgets (labels, value bars, status icons) for sensor dashboards.
- **Color/Theme Profiles**: Map amplitude ranges to color gradients (heatmap visualization).

6.3 Recommended Next Steps

Prioritize DMA integration and trigger logic for oscilloscope, then expand primitive set and text handling to support richer UI applications (data logger, multi-channel monitor).

- Configure SPI in master (one board) and slave (other board) mode.
- Configure UART on both boards for terminal output text (Putty etc).
- Define and use two control commands over SPI: start and finish.
- Implement the count logic (0-9) on the slave, and delay/loop logic on the master.
- Print meaningful messages via UART and maintain a cycle counter on the slave