# Integrating LVGL with the ILI9341 Driver over SPI

## Overview of Integration Approach

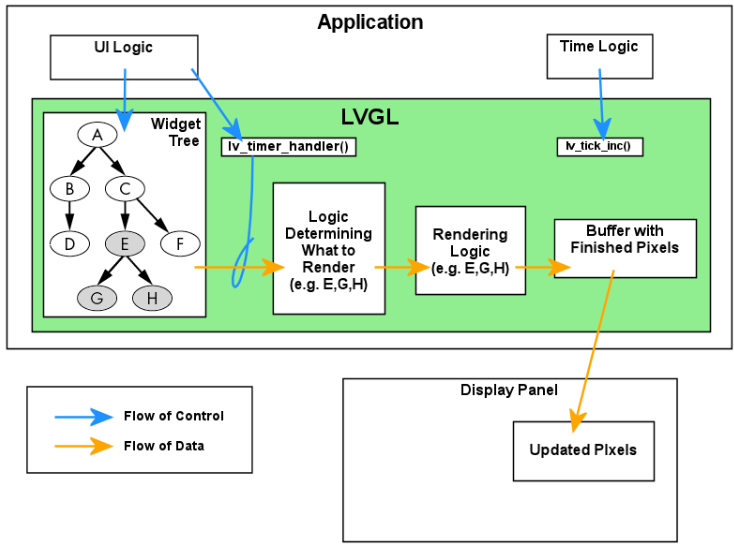
The graphical user interface of our system is rendered using LVGL v9.3, a powerful and lightweight embedded GUI library. To display LVGL's graphical output on our 240x320 TFT screen, we integrated the ILI9341 display driver over the SPI interface on the Raspberry Pi 3 using bare-metal C, without an operating system or abstraction layer.  
  
The integration leverages the driver wrapper provided in the official LVGL source for the ILI9341 controller. In our implementation, we manually configure SPI communication routines in C, then bind them to LVGL’s display driver through two critical functions:  
  
- my\_lcd\_send\_cmd(): Sends commands to the ILI9341 controller  
- my\_lcd\_send\_color(): Sends pixel color data from the LVGL buffer to the display  
  
These functions are passed to the lv\_ili9341\_create() function, which registers them with LVGL's internal display driver abstraction.  


Figure ‑: Overview of LVGL's Data Flow

## Hardware Connections Between ILI9341 and Raspberry Pi 3

The physical connection between the Raspberry Pi 3 and the ILI9341 TFT display is established via the SPI interface. No logic level shifter is needed, as both devices operate at 3.3V levels. The table below summarizes the wiring used:

Table ‑: Hard ware interface between RPI 3 and ILI9341 Controller

|  |  |
| --- | --- |
| ILI9341 Pin | Raspberry Pi 3 GPIO |
| VCC | 3.3V (Pin 1) |
| GND | GND (Pin 6) |
| CS | GPIO 8 (Pin 24) |
| RESET | GPIO 25 (Pin 22) |
| DC | GPIO 24 (Pin 18) |
| MOSI | GPIO 10 (Pin 19) |
| SCK | GPIO 11 (Pin 23) |
| MISO | GPIO 9 (Pin 21) |
| LED | 3.3V (Pin 17) |

## Enabling SPI Interface on Raspberry Pi 3

To communicate with the ILI9341 over SPI, the hardware SPI0 interface must be enabled on the Raspberry Pi 3. This was done using the `raspi-config` utility:

1. Run `*sudo raspi-config`*
2. Navigate to `*Interfacing Options*`
3. Select `*SPI*` and enable it
4. Reboot the Raspberry Pi

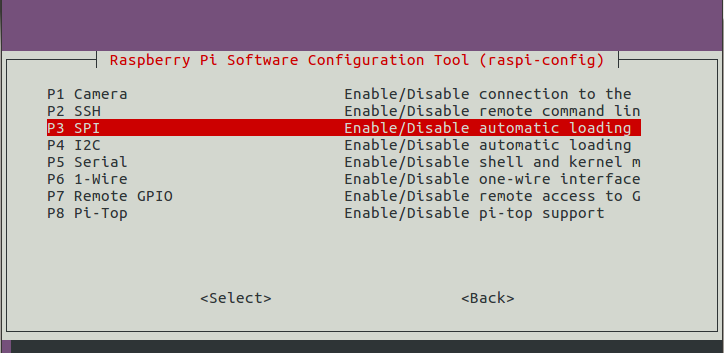


Figure ‑: Screenshot of raspi-config SPI enable screen

Once enabled, the SPI device is accessible at *`/dev/spidev0.0*`. No custom device tree overlays were required.  
  
To verify the SPI module is active, use the following command:

*ls /dev/spi\**

You should see:  
*/dev/spidev0.0*

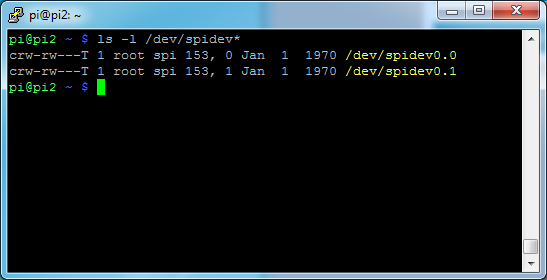
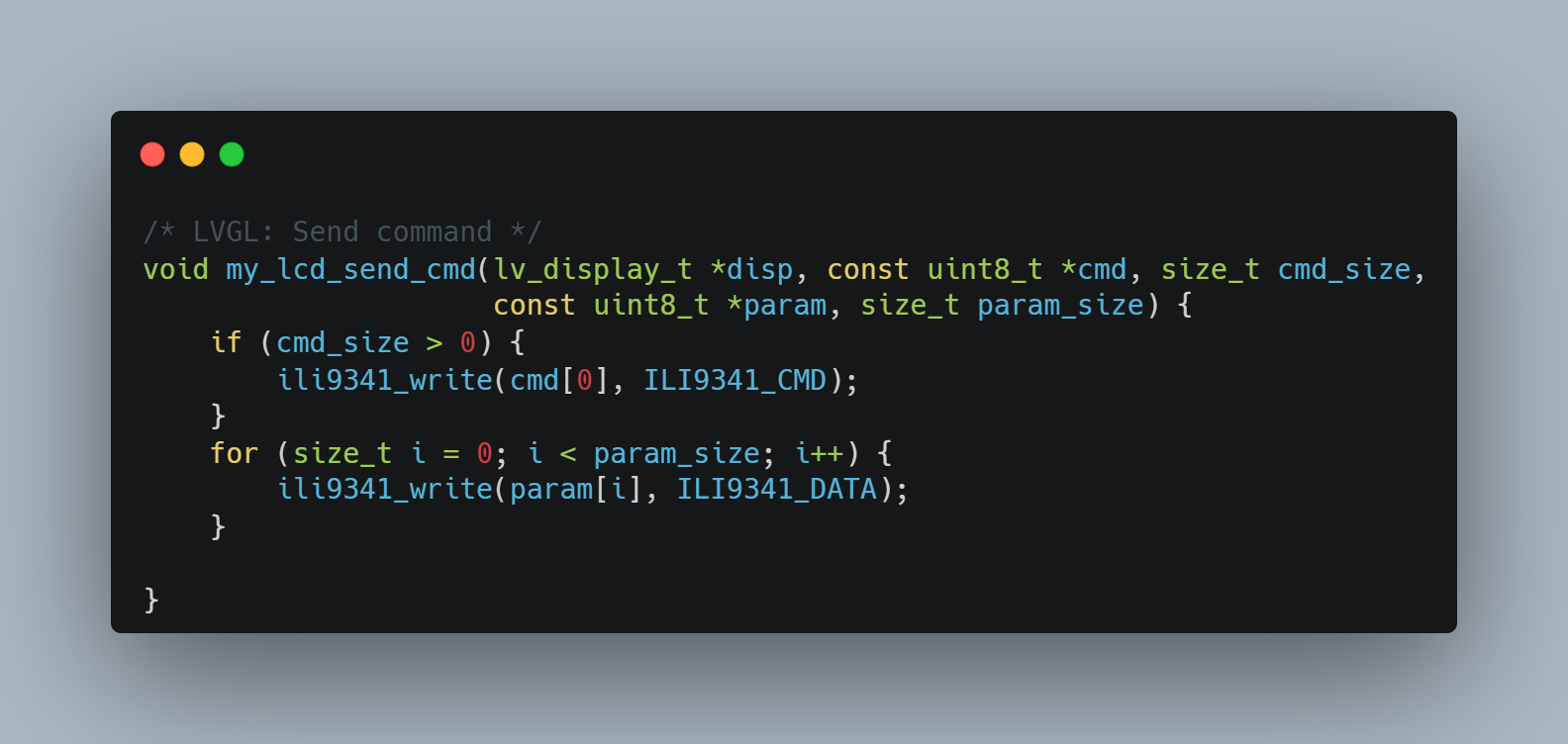


Figure ‑: SPI verification command and output

## 2. Display Initialization

Before LVGL can render frames, the ILI9341 must be initialized. While the low-level initialization sequence is handled internally by lv\_ili9341\_create(), it requires that SPI communication functions are properly implemented and provided by the user



Code Snipped ‑: Command sending logic

**Behavior:**

1. If a command is present, it is sent using ili9341\_write(cmd[0], ILI9341\_CMD), which handles the SPI write and properly sets the Data/Command pin (DC) to command mode.
2. Each parameter byte is then sent in sequence using ili9341\_write(..., ILI9341\_DATA), switching the DC line to data mode automatically.
3. The function abstracts both low-level SPI and control signal management.

This design ensures that command sequences like setting column/page addresses or memory access mode can be cleanly sent from the LVGL rendering pipeline.

## 3. Transferring Pixel Data to ILI9341

LVGL periodically invokes a flush\_cb() callback to send a portion of its display buffer to the screen. The ILI9341 expects pixel data in RGB565 format, which is already compatible with LVGL’s default color depth.

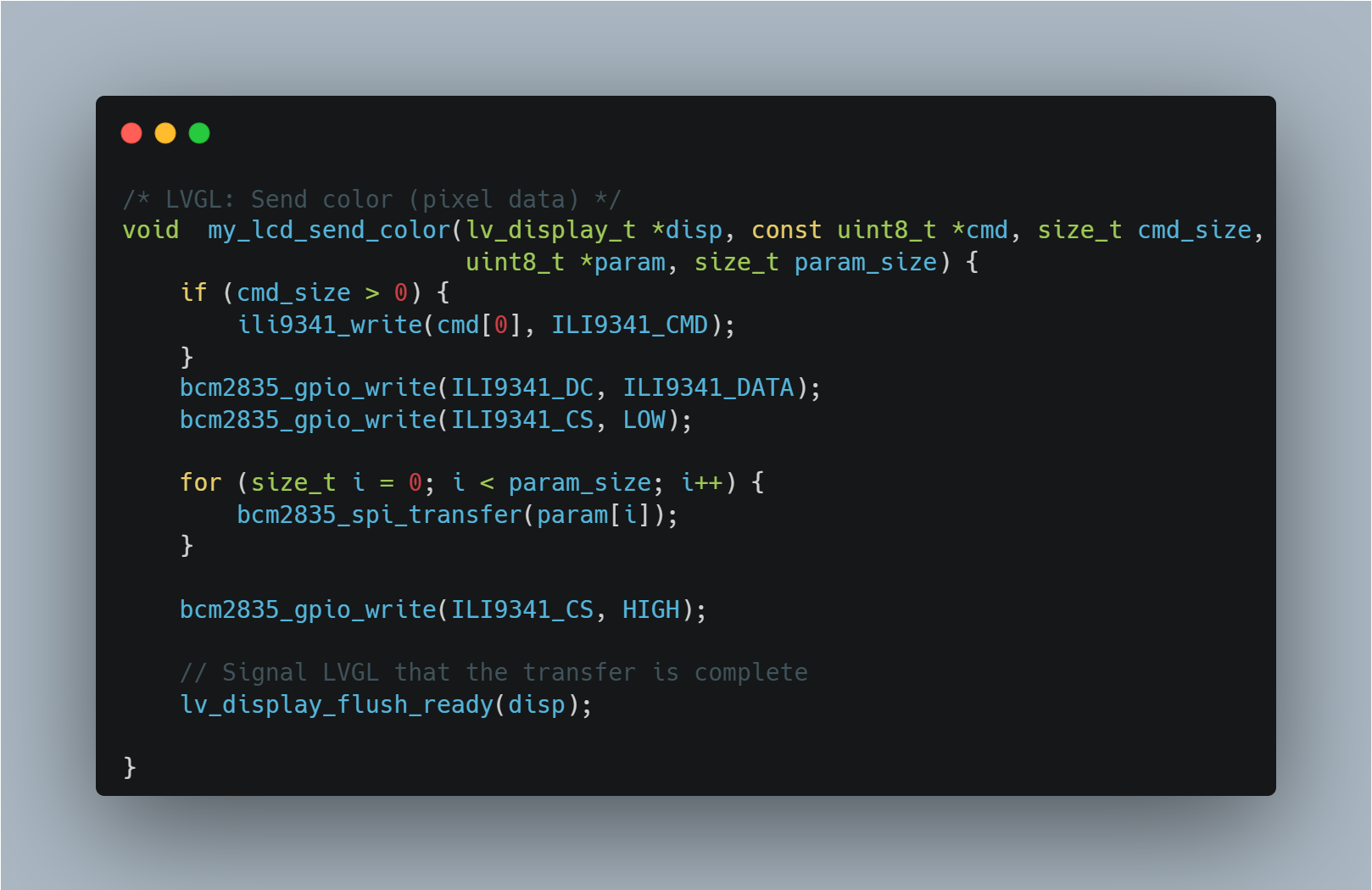


Figure ‑: Pixel data transfer

**Key Operations:**

1. If a command is present (usually 0x2C for **memory write**), it is sent using ili9341\_write(..., ILI9341\_CMD) to set the display in pixel-write mode.
2. Control of the DC and CS lines is manually asserted to **data mode** and **start transmission** respectively.
3. A loop sends all pixel bytes using bcm2835\_spi\_transfer(), ensuring fast bulk transmission.
4. Finally, CS is deserted, and lv\_display\_flush\_ready(disp) is called to notify LVGL that the drawing operation has completed.

**Design Note:**  
While ili9341\_write() could have been reused here, the manual handling of CS and DC was intentionally chosen to optimize throughput for continuous pixel data, avoiding the overhead of per-byte function calls. This improves SPI transfer performance, especially when rendering large screen areas.

## 4. Registering the Display Driver

After defining the communication routines, we register them using the helper provided by LVGL:



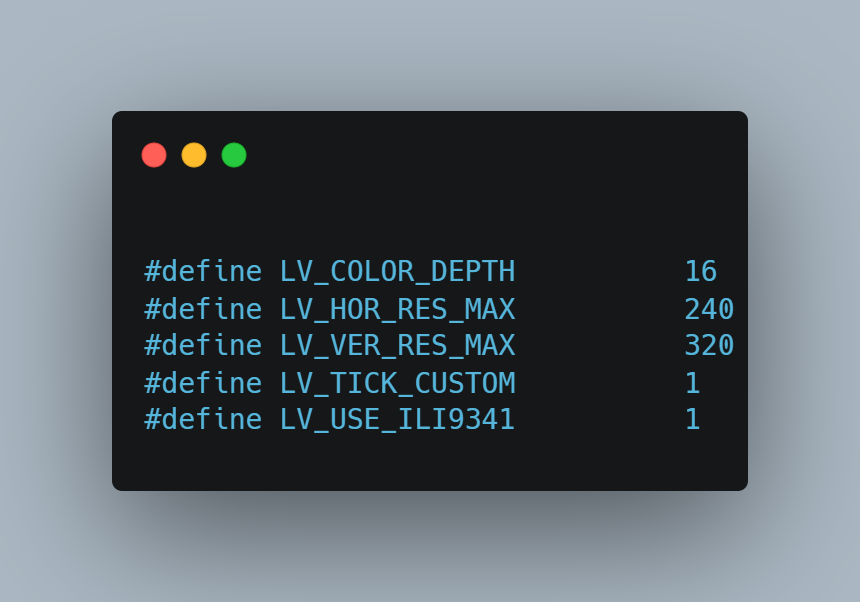
Code Snipped ‑: Display registration

**Behavior:**

* Creates an LVGL display object using lv\_lcd\_generic\_mipi\_create(), linking it to the ILI9341 through your custom send functions.
* Sets the screen rotation to 0°.
* Calculates the buffer size based on screen resolution and color format.
* Allocates two frame buffers for double buffering.
* Verifies buffer allocation success.
* Registers the buffers with LVGL using partial rendering mode.

## 5. Configuration in lv\_conf.h

The following parameters were customized in lv\_conf.h to match our hardware:



Code Snipped ‑: Customized LVGL configuration

* LV\_COLOR\_DEPTH is set to 16 to use RGB565.
* LV\_HOR\_RES\_MAX and LV\_VER\_RES\_MAX are configured per screen resolution.
* LV\_TICK\_CUSTOM is enabled to allow hardware timer tick integration.
* LV\_USE\_ILI9341 is enabled ILI9341 controller to work with LVGL

## References

[1] LVGL Official Documentation – Display Porting Guide  
https://docs.lvgl.io/latest/porting/display.html  
[2] LVGL ILI9341 Display Driver (source and usage):  
https://github.com/lvgl/lvgl