ReceiveTest Code Explanation

A complete walkthrough of every file and line of code in the project.

---

1. Project Structure

```
ReceiveTest/
├── CMakeLists.txt              # Project config
├── sdkconfig.defaults          # Hardware settings
├── main/
│   ├── CMakeLists.txt          # Main component config
│   ├── idf_component.yml       # Dependencies
│   └── main.c                  # Entry point
└── components/
    ├── display/                # Display hardware driver
    │   ├── CMakeLists.txt
    │   ├── display_init.c
    │   └── include/
    │       ├── display_config.h
    │       └── display_init.h
    └── ui/                     # User interface
        ├── CMakeLists.txt
        ├── ui.c
        └── include/
            └── ui.h
```

---

2. CMakeLists.txt (Project Root)

```
cmake_minimum_required(VERSION 3.16)
include($ENV{IDF_PATH}/tools/cmake/project.cmake)
project(ReceiveTest)
```

Line	Meaning
`cmake_minimum_required(VERSION 3.16)`	Requires CMake version 3.16 or newer to build
`include($ENV{IDF_PATH}/tools/cmake/project.cmake)`	Load ESP-IDF's build system. `$ENV{IDF_PATH}` is where ESP-IDF is installed
`project(ReceiveTest)`	Name this project "ReceiveTest"

---

3. main/idf_component.yml

```
dependencies:
  idf:
    version: '>=5.3.0'
  waveshare/esp_lcd_jd9365_10_1: '*'
  lvgl/lvgl: '~9.2.0'
  espressif/esp_lvgl_port: '^2'
```

This tells ESP-IDF's component manager to download these libraries:

Dependency	What it is
`idf: >=5.3.0`	Require ESP-IDF version 5.3.0 or newer

Dependency	What it is
`waveshare/esp_lcd_jd9365_10_1`	Driver for the specific 10.1" display panel
`lvgl/lvgl: ~9.2.0`	LVGL graphics library (version 9.2.x)
`espressif/esp_lvgl_port: ^2`	Glue code that connects LVGL to ESP-IDF (version 2.x)

---

4. main/CMakeLists.txt

```
idf_component_register(
    SRCS "main.c"
    INCLUDE_DIRS "."
    REQUIRES
        display
        ui
        esp_lvgl_port
        lvgl
)
```

Field	Meaning
`SRCS "main.c"`	Compile this source file
`INCLUDE_DIRS "."`	Look for header files in the current directory
`REQUIRES`	This component depends on these other components

---

5. main/main.c (Entry Point)

Includes

```
#include <stdio.h>
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_log.h"
```

- `stdio.h` / `string.h` - Standard C library (printf, string functions)
- `freertos/FreeRTOS.h` - Real-time operating system (handles multitasking)
- `freertos/task.h` - Task/thread functions like `vTaskDelay`
- `esp_log.h` - ESP-IDF logging (`ESP_LOGI` , `ESP_LOGE` , etc.)

```
#include "lvgl.h"
#include "esp_lvgl_port.h"
```

- `lvgl.h` - The graphics library
- `esp_lvgl_port.h` - ESP-IDF wrapper for LVGL

```
#include "display_init.h"
#include "display_config.h"
#include "ui.h"
```

- Our custom components (display driver and UI)

Tag for Logging

```
static const char *TAG = "ReceiveTest";
```

A tag for log messages. When you see `I (355) ReceiveTest: Starting...`, "ReceiveTest" comes from this.

app_main Function

```
void app_main(void)
{
```

The entry point. ESP-IDF calls this function when the chip boots (like `main()` in regular C).

```
    ESP_LOGI(TAG, "Starting ReceiveTest");
```

Print an **I**nfo log message. Shows up as `I (355) ReceiveTest: Starting ReceiveTest`

```
    esp_lcd_panel_handle_t panel = display_init();
```

Call our display initialization function. Returns a "handle" (pointer) to the display panel hardware.

```
    lvgl_port_init(&(lvgl_port_cfg_t)ESP_LVGL_PORT_INIT_CONFIG());
```

Initialize the LVGL port with default settings. This:

- Creates an LVGL task that runs in the background
- Sets up timers for animations
- Prepares LVGL to receive a display

Display Registration with LVGL

```
    lv_display_t *disp = lvgl_port_add_disp_dsi(
       &(lvgl_port_display_cfg_t){
           .panel_handle = panel,
           .buffer_size = LCD_H_RES * LCD_V_RES * 3,
           .double_buffer = false,
           .hres = LCD_H_RES,
           .vres = LCD_V_RES,
           .color_format = LV_COLOR_FORMAT_RGB888,
           .flags.buff_spiram = true,
           .flags.sw_rotate = true,
       },
       &(lvgl_port_display_dsi_cfg_t){});
```

Register the display with LVGL:

Field	Meaning
`.panel_handle = panel`	The hardware panel we just initialized
`.buffer_size = LCD_H_RES * LCD_V_RES * 3`	Frame buffer size (800×1280×3 = 3MB)
`.double_buffer = false`	Use one buffer, not two (saves memory)
`.hres = LCD_H_RES`	Horizontal resolution (800)
`.vres = LCD_V_RES`	Vertical resolution (1280)
`.color_format = LV_COLOR_FORMAT_RGB888`	24-bit color (8 bits each for R, G, B)
`.flags.buff_spiram = true`	**Put the buffer in PSRAM** (critical!)
`.flags.sw_rotate = true`	Allow software rotation

```
    lv_display_set_rotation(disp, LV_DISPLAY_ROTATION_90);
```

Rotate the display 90 degrees (landscape mode).

```
    ui_init(disp);
```

Initialize our UI (creates the labels on screen).

Main Loop

```
    ESP_LOGI(TAG, "Display ready, starting demo counter");

    int counter = 0;
    while (1) {
        ui_set_number(counter);
        counter++;
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}
```

The main loop:

1. Update the display with current counter value
2. Increment counter
3. Wait 1000 milliseconds (1 second)
4. Repeat forever

`pdMS_TO_TICKS(1000)` converts milliseconds to FreeRTOS "ticks" (the OS's time unit).

---

6. components/display/include/display_config.h

```
#pragma once

#define LCD_H_RES               800
#define LCD_V_RES               1280
#define LCD_BIT_PER_PIXEL       24
#define MIPI_DSI_LANE_NUM       2

#define PIN_NUM_LCD_RST         27
#define PIN_NUM_BK_LIGHT        26
#define LCD_BK_LIGHT_ON_LEVEL   1

#define MIPI_DSI_PHY_PWR_LDO_CHAN       3
#define MIPI_DSI_PHY_PWR_LDO_VOLTAGE_MV 2500
```

Define	Meaning
`LCD_H_RES` / `LCD_V_RES`	Screen dimensions in pixels
`LCD_BIT_PER_PIXEL`	Color depth (24 = RGB888)
`MIPI_DSI_LANE_NUM`	Number of data lanes for MIPI DSI (display interface)
`PIN_NUM_LCD_RST`	GPIO pin 27 controls display reset
`PIN_NUM_BK_LIGHT`	GPIO pin 26 controls backlight
`LCD_BK_LIGHT_ON_LEVEL`	1 = HIGH turns backlight on
`MIPI_DSI_PHY_PWR_LDO_*`	Power settings for the display interface

---

7. components/display/display_init.c

Includes

```
#include "display_init.h"
#include "display_config.h"
#include "driver/gpio.h"
#include "esp_log.h"
#include "esp_lcd_mipi_dsi.h"
#include "esp_lcd_panel_ops.h"
#include "esp_lcd_jd9365_10_1.h"
#include "esp_ldo_regulator.h"
```

Headers for GPIO control, logging, MIPI DSI display interface, and the JD9365 panel driver.

Pixel Format Selection

```
#if LCD_BIT_PER_PIXEL == 24
#define MIPI_DPI_PX_FORMAT LCD_COLOR_PIXEL_FORMAT_RGB888
#else
#define MIPI_DPI_PX_FORMAT LCD_COLOR_PIXEL_FORMAT_RGB565
#endif
```

Choose pixel format based on color depth. 24-bit = RGB888, otherwise RGB565.

display_init Function

```
esp_lcd_panel_handle_t display_init(void)
{
    ESP_LOGI(TAG, "Initializing display hardware");
```

Start of initialization function, returns a handle to the panel.

Backlight Configuration

```
    gpio_config_t bk_cfg = {
        .mode = GPIO_MODE_OUTPUT,
        .pin_bit_mask = 1ULL << PIN_NUM_BK_LIGHT,
    };
    ESP_ERROR_CHECK(gpio_config(&bk_cfg));
    ESP_ERROR_CHECK(gpio_set_level(PIN_NUM_BK_LIGHT, LCD_BK_LIGHT_ON_LEVEL));
```

Configure GPIO 26 as output, then set it HIGH to turn on the backlight.

`ESP_ERROR_CHECK()` crashes with an error message if the function fails.

MIPI DSI Power

```
    esp_ldo_channel_handle_t ldo_mipi = NULL;
    esp_ldo_channel_config_t ldo_cfg = {
        .chan_id = MIPI_DSI_PHY_PWR_LDO_CHAN,
        .voltage_mv = MIPI_DSI_PHY_PWR_LDO_VOLTAGE_MV,
    };
    ESP_ERROR_CHECK(esp_ldo_acquire_channel(&ldo_cfg, &ldo_mipi));
```

Power on the MIPI DSI interface using LDO channel 3 at 2500mV.

DSI Bus Creation

```
    esp_lcd_dsi_bus_handle_t dsi_bus = NULL;
    esp_lcd_dsi_bus_config_t bus_cfg = JD9365_PANEL_BUS_DSI_2CH_CONFIG();
    ESP_ERROR_CHECK(esp_lcd_new_dsi_bus(&bus_cfg, &dsi_bus));
```

Create the MIPI DSI bus (the high-speed connection to the display). `JD9365_PANEL_BUS_DSI_2CH_CONFIG()` is a macro from the Waveshare driver with preset values.

Panel I/O Creation

```
    esp_lcd_panel_io_handle_t panel_io = NULL;
    esp_lcd_dbi_io_config_t io_cfg = JD9365_PANEL_IO_DBI_CONFIG();
    ESP_ERROR_CHECK(esp_lcd_new_panel_io_dbi(dsi_bus, &io_cfg, &panel_io));
```

Create an I/O interface for sending commands to the panel (separate from pixel data).

Panel Configuration

```
    esp_lcd_dpi_panel_config_t dpi_cfg =
        JD9365_800_1280_PANEL_60HZ_DPI_CONFIG(MIPI_DPI_PX_FORMAT);
```

Configure the panel for 800×1280 at 60Hz with our chosen pixel format.

```
    jd9365_vendor_config_t vendor_cfg = {
        .flags = {
            .use_mipi_interface = 1,
        },
        .mipi_config = {
            .dsi_bus = dsi_bus,
            .dpi_config = &dpi_cfg,
            .lane_num = MIPI_DSI_LANE_NUM,
        },
    };
```

Vendor-specific configuration for the JD9365 panel.

Panel Creation

```
    esp_lcd_panel_handle_t panel = NULL;
    esp_lcd_panel_dev_config_t panel_cfg = {
        .reset_gpio_num = PIN_NUM_LCD_RST,
        .rgb_ele_order = LCD_RGB_ELEMENT_ORDER_RGB,
        .bits_per_pixel = LCD_BIT_PER_PIXEL,
        .vendor_config = &vendor_cfg,
    };

    ESP_ERROR_CHECK(
        esp_lcd_new_panel_jd9365(panel_io, &panel_cfg, &panel)
    );
```

Create the panel object with all our settings.

Panel Initialization

```
    ESP_ERROR_CHECK(esp_lcd_panel_reset(panel));
    ESP_ERROR_CHECK(esp_lcd_panel_init(panel));
    ESP_ERROR_CHECK(esp_lcd_panel_disp_on_off(panel, true));
```

1. Reset the panel (pulse the reset pin)
2. Send initialization commands to the panel
3. Turn the display on

```
    ESP_LOGI(TAG, "Display hardware ready");
    return panel;
}
```

Log success and return the panel handle.

---

8. components/ui/ui.c

Includes and Global State

```
#include "ui.h"
#include "esp_lvgl_port.h"
#include "lvgl.h"
#include <stdio.h>

static lv_obj_t *number_label = NULL;
```

`number_label` is a **static global pointer**. It stores a reference to the label widget so `ui_set_number()` can update it later.

ui_init Function

```
void ui_init(lv_display_t *disp)
{
    lvgl_port_lock(0);
```

Lock LVGL. LVGL runs in a background task. Before modifying UI objects, you must lock to prevent race conditions. `0` means wait forever until lock is acquired.

```
    lv_obj_t *scr = lv_display_get_screen_active(disp);
```

Get the active screen (the root container for all widgets).

Background Styling

```
    lv_obj_set_style_bg_color(scr, lv_color_hex(0x003366), LV_PART_MAIN);
    lv_obj_set_style_bg_opa(scr, LV_OPA_COVER, LV_PART_MAIN);
```

Set background to dark blue (`#003366`) with full opacity.

Title Label

```
    lv_obj_t *title = lv_label_create(scr);
    lv_label_set_text(title, "The data being sent is:");
    lv_obj_set_style_text_color(title, lv_color_white(), LV_PART_MAIN);
    lv_obj_set_style_text_font(title, &lv_font_montserrat_32, LV_PART_MAIN);
    lv_obj_align(title, LV_ALIGN_CENTER, 0, -50);
```

1. Create a label widget as a child of the screen
2. Set its text
3. Set text color to white
4. Set font to Montserrat 32px
5. Position it centered, 50 pixels above center

Dynamic Number Label

```
    number_label = lv_label_create(scr);
    lv_label_set_text(number_label, "---");
    lv_obj_set_style_text_color(number_label, lv_color_hex(0x00FF00), LV_PART_MAIN);
    lv_obj_set_style_text_font(number_label, &lv_font_montserrat_48, LV_PART_MAIN);
    lv_obj_align(number_label, LV_ALIGN_CENTER, 0, 50);
```

Same thing, but:

- Store pointer in `number_label` so we can update it later
- Green text (`#00FF00`)
- Larger font (48px)
- 50 pixels below center

```
    lvgl_port_unlock();
}
```

Release the lock so LVGL's background task can render.

ui_set_number Function

```
void ui_set_number(int number)
{
    if (number_label == NULL) {
        return;
    }

    lvgl_port_lock(0);
    lv_label_set_text_fmt(number_label, "%d", number);
    lvgl_port_unlock();
}
```

1. Safety check: if UI wasn't initialized, do nothing
2. Lock LVGL
3. Update label text using printf-style formatting (`%d` = integer)
4. Unlock LVGL

---

9. sdkconfig.defaults (Hardware Settings)

```
# Target
CONFIG_IDF_TARGET="esp32p4"

# PSRAM / SPIRAM (required for display frame buffer ~3MB)
CONFIG_SPIRAM=y
CONFIG_SPIRAM_MODE_HEX=y
CONFIG_SPIRAM_SPEED_200M=y
CONFIG_SPIRAM_BOOT_INIT=y
CONFIG_SPIRAM_USE_MALLOC=y
CONFIG_SPIRAM_MEMTEST=y
CONFIG_SPIRAM_MALLOC_ALWAYSINTERNAL=16384
CONFIG_SPIRAM_MALLOC_RESERVE_INTERNAL=32768

# PSRAM power (LDO channel 2 at 1.8V for ESP32-P4)
CONFIG_ESP_LDO_CHAN_PSRAM_DOMAIN=2
CONFIG_ESP_LDO_VOLTAGE_PSRAM_1800_MV=y

# Ethernet
CONFIG_ETH_ENABLED=y
CONFIG_ETH_USE_ESP32_EMAC=y

# LVGL fonts
CONFIG_LV_FONT_MONTSERRAT_32=y
CONFIG_LV_FONT_MONTSERRAT_48=y

# Stack size
CONFIG_ESP_MAIN_TASK_STACK_SIZE=8192
```

Setting	Purpose
`CONFIG_SPIRAM=y`	Enable external PSRAM support
`CONFIG_SPIRAM_MODE_HEX=y`	Use 8 data lines instead of 4 (faster)
`CONFIG_SPIRAM_SPEED_200M=y`	Run the SPI bus at 200MHz
`CONFIG_SPIRAM_BOOT_INIT=y`	Initialize PSRAM at boot
`CONFIG_SPIRAM_USE_MALLOC=y`	Allow malloc() to use PSRAM for large allocations
`CONFIG_ESP_LDO_CHAN_PSRAM_DOMAIN=2`	Use LDO channel 2 to power PSRAM

Setting	Purpose
`CONFIG_ESP_LDO_VOLTAGE_PSRAM_1800_MV=y`	Supply 1.8 volts to PSRAM
`CONFIG_ETH_ENABLED=y`	Enable Ethernet support
`CONFIG_LV_FONT_MONTSERRAT_*`	Include these font sizes in the build

---

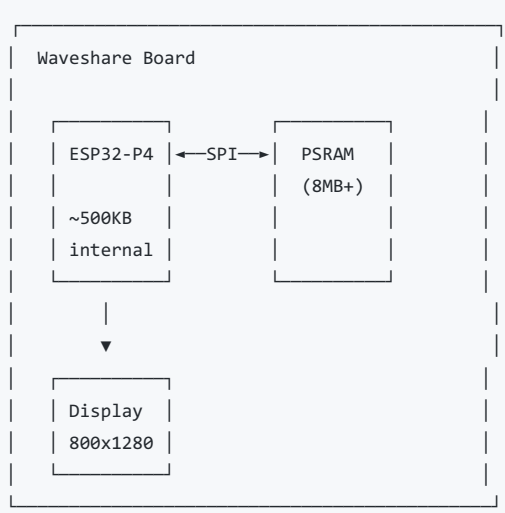
10. The Complete Flow

```
Boot
  │
  ▼
app_main()
  │
  ├─► display_init() ─► Hardware ready, backlight on
  │
  ├─► lvgl_port_init() ─► LVGL background task starts
  │
  ├─► lvgl_port_add_disp_dsi() ─► LVGL knows about the display
  │
  ├─► ui_init() ─► Labels created on screen
  │
  └─► while(1) loop
        │
        ├─► ui_set_number(counter) ─► Update green number
        ├─► counter++
        └─► wait 1 second
```

---

11. Memory Architecture

The display needs ~3MB for its frame buffer (800×1280×3 bytes), but the ESP32-P4 only has ~500KB internal RAM.



The PSRAM (external RAM) holds the large frame buffer. When `malloc()` is called for the 3MB buffer, `CONFIG_SPIRAM_USE_MALLOC=y` automatically places it in PSRAM.

Document generated for MSU Solar Racing Team