Федеральное государственное бюджетное образовательное учреждение
высшего образования «Воронежский государственный университет»

Отчет по лабораторной работе № 7
По курсу «Введение в интернет вещей»

«шина связи SPI»

Выполнил(а) :
Меркутова Кристина Денисовна
3 группа

Воронеж 2025

Цель работы:

Освоить принципы функционирования и программной реализации шины данных SPI на микроконтроллере ESP32, включая инициализацию шины, обнаружение устройств, обмен данными.

Ход выполнения работы:

1. Ответьте на вопросы в теоретической части максимально подробно;

2. Соберите схему практической части и напишите программу для микроконтроллера. Убедитесь, что код выполняется верно, используйте лабораторный стенд. Скриншоты или фото устройства и листинг программы микроконтроллера обязательны.

1. Теоретическая часть. Ответьте на вопросы:

1. Как расшифровывается аббревиатура SPI? **Ответ:**
   Аббревиатура **SPI** расшифровывается как Serial Peripheral Interface
2. Сколько линий (проводов) обычно используется в базовой конфигурации SPI? **Ответ:** в базовой конфигурации для работы с одним ведомым устройством (Slave) используется 4 линии.
3. Какие функции выполняют сигналы MOSI, MISO, SCLK и SS (или CS)? **Ответ:** MOSI **-** передача выходных данных от ведущего устройства на вход ведомого. MISO - Передача выходных данных от ведомого устройства на вход ведущего. SCLK - синхронизация передачи данных. Он задаёт темп и определяет моменты, когда биты данных являются валидными. SS - выбор конкретного ведомого устройства для обмена данными. Это сигнал управления, а не передачи данных.
4. Чем отличается ведущее (master) устройство от ведомого (slave) в шине SPI? **Ответ:** власть у Master**:** Вся власть сосредоточена в ведущем устройстве. Slave — это подчиненный, который отвечает только когда к нему обращаются.

— Проблема синхронизации решена**:** поскольку такт генерирует один источник (Master), проблемы синхронизации часов между устройствами отсутствуют, что упрощает протокол.

— Slave не может "прервать" Master: если ведомому устройству (например, датчику) срочно нужно сообщить данные, оно не может этого сделать, пока Master его не выберет. Для решения таких задач используют другие протоколы (например, с линией прерывания) или архитектуру.
— Гибкость Master**:** Master может эмулировать работу с разными Slave, имеющими различные требования к режимам SPI (полярность и фаза такта), меняя свои настройки "на лету" перед выбором каждого устройства.

5. Какие преимущества и недостатки у SPI по сравнению с I²C?

Преимущества SPI перед I²C:

— Высокая скорость: значительно быстрее (десятки МГц), так как это простой сдвиговый регистр без формализованного протокола.
— Проще аппаратно и программно: нет сложных протоколов адресации, подтверждений, арбитража. Данные идут потоком.
— Полный дуплекс: может одновременно передавать и принимать, что повышает реальную пропускную способность.
— Гибкая длина данных: работает со словами любой разрядности (не только 8 бит).

Недостатки SPI перед I²C:

— Больше проводов: 4 линии минимум против 2-х у I²C. Для множества устройств проводников становится очень много (отдельный CS для каждого).
— Нет встроенной адресации: нет механизма "вызова по адресу", требуется отдельная линия выбора (CS) для каждого Slave. Это увеличивает загрузку пинов Master.
— Нет подтверждения приёма (ACK/NACK): Master не получает аппаратного подтверждения, что данные дошли. Контроль ошибок — задача программного уровня.
— Нет multi-master режима: Шина фактически поддерживает только одного ведущего без дополнительных ухищрений.
— Нет встроенного управления питанием (sleep-режим), как в некоторых расширениях I²C.

2. Практическая часть:

1. Соберите схему, используя модуль TFT-дисплей 0,96 дюйма (80x160 пикселей). Напишите код для вывода информации, на дисплей, используя esp-idf, используйте библиотеку для работы с датчиком по вашему выбору.

Схема TFT + Touchscreen + Motion sensor:

Фото когда движения датчик не работает(тк нету движения)

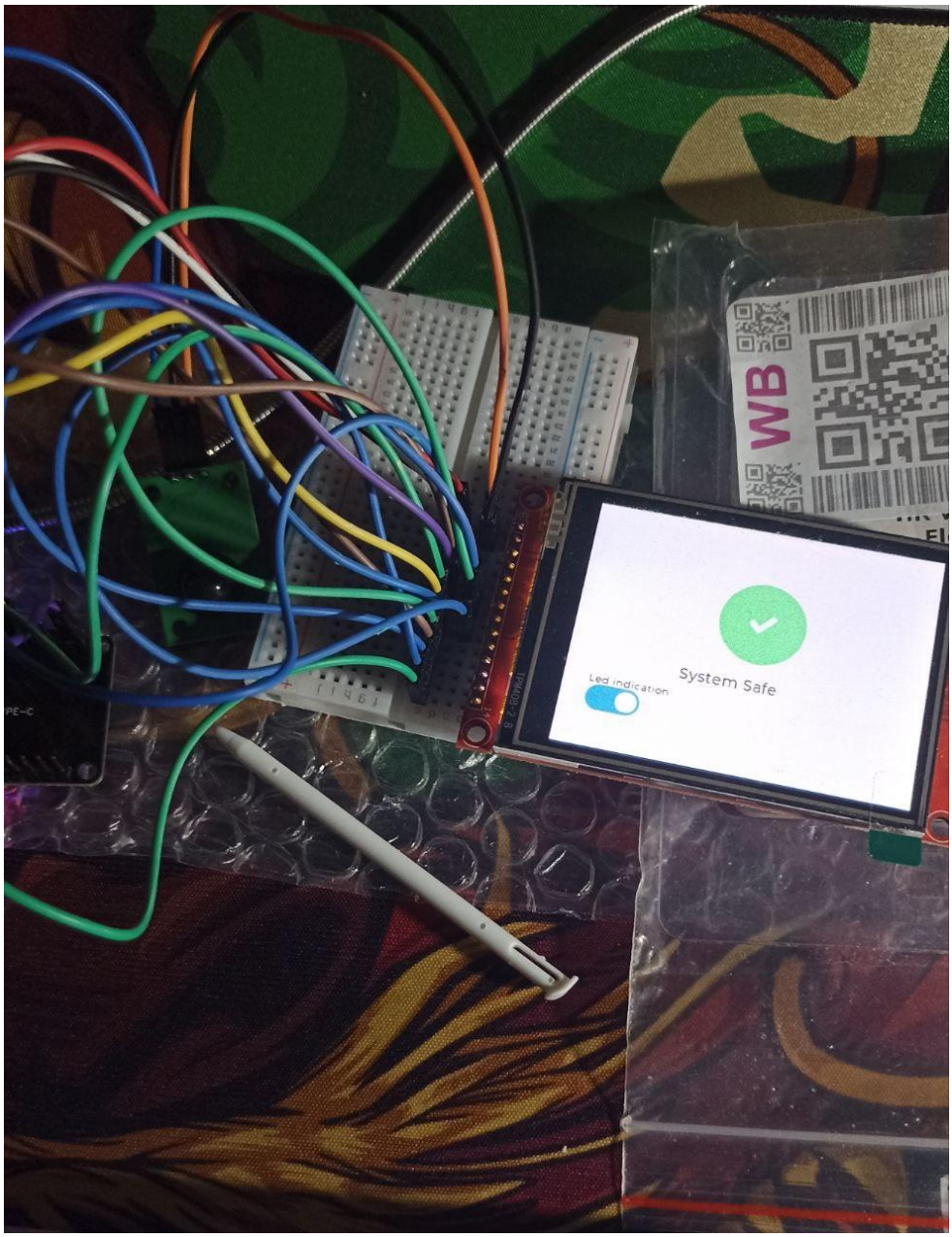


Фото когда датчик движения сработал

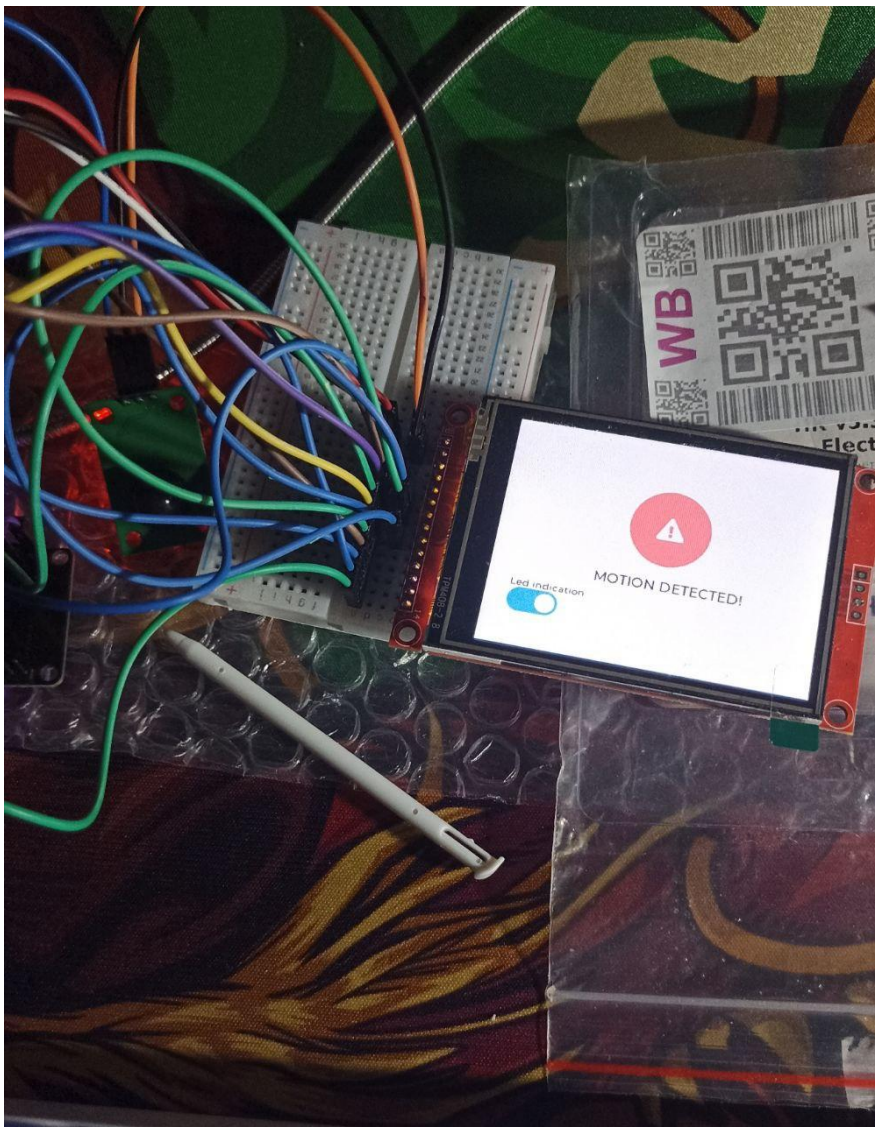
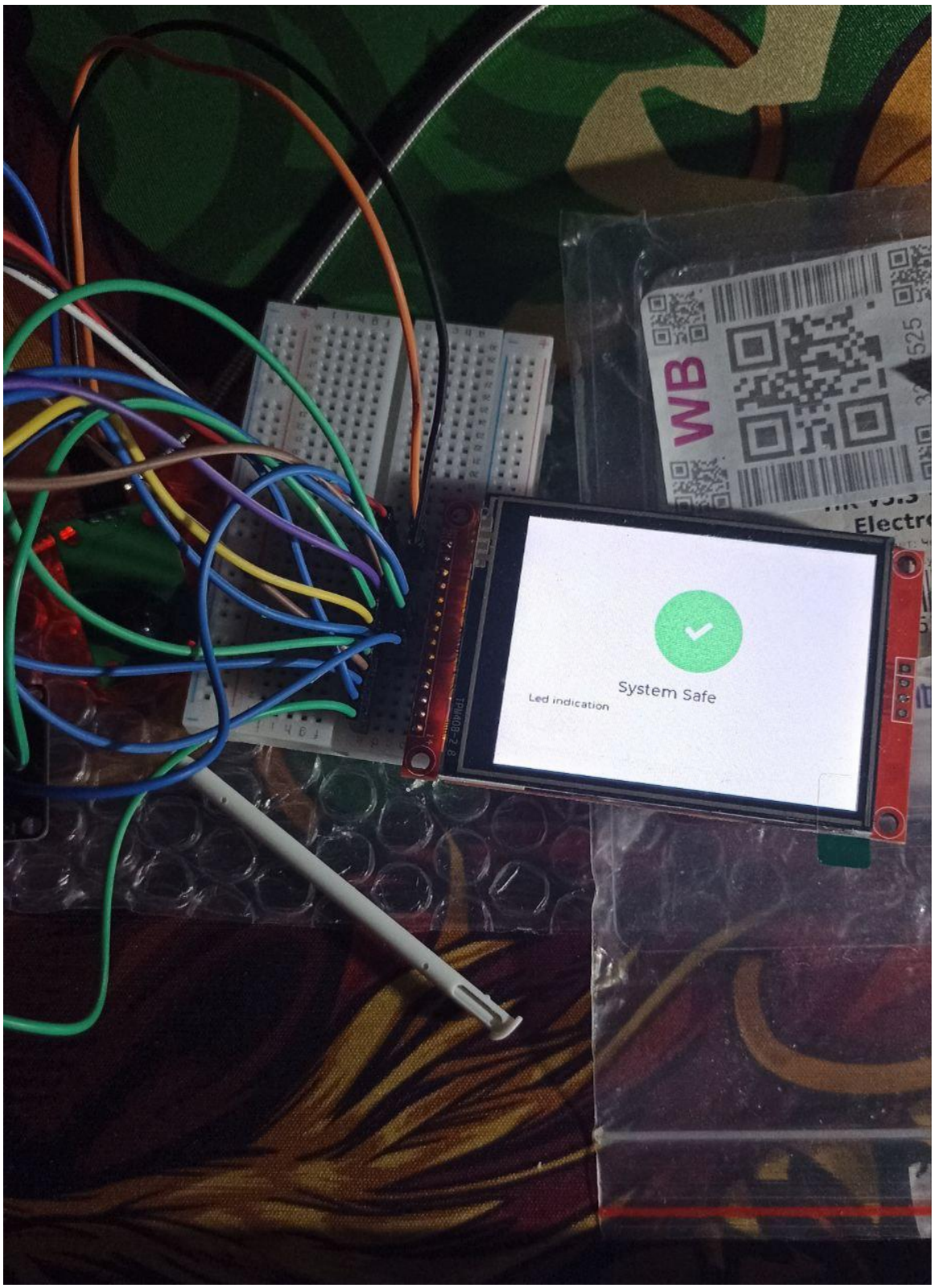


Фото если нажать на switch , то даже при срабатывании датчика ui не поменяется



Код:

```c
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/gpio.h"
#include "driver/spi_master.h"
#include "esp_timer.h"
#include "esp_log.h"
#include "esp_lcd_panel_io.h"
#include "esp_lcd_panel_ops.h"
#include "esp_lcd_panel_vendor.h"
#include "esp_lcd_ili9341.h"
#include "esp_lcd_touch.h"
#include "esp_lcd_touch_xpt2046.h"
#include "lvgl.h"
#include "ui.h"
#include "screens.h"

static const char *TAG = "System_Fixed";

// --- ПИНЫ ---
#define PIN_MOTION_SENSOR   27

#define LCD_HOST            SPI3_HOST
#define LCD_H_RES_PHYS      240
#define LCD_V_RES_PHYS      320

#define PIN_NUM_MISO        19
#define PIN_NUM_MOSI        23
#define PIN_NUM_SCLK        18
#define PIN_NUM_CS          5
#define PIN_NUM_DC          2
#define PIN_NUM_RST         4
#define PIN_NUM_BCKL        21

#define TOUCH_HOST          SPI3_HOST
#define PIN_NUM_TOUCH_CS    33
#define PIN_NUM_TOUCH_IRQ   25

#define LCD_PIXEL_CLOCK_HZ  (20 * 1000 * 1000)
#define LVGL_BUF_SIZE       (320 * 40 * sizeof(uint16_t))

// --- КАЛИБРОВКА ---
#define TOUCH_RAW_X_MIN 28
#define TOUCH_RAW_X_MAX 293
#define TOUCH_RAW_Y_MIN 14
#define TOUCH_RAW_Y_MAX 222

static lv_display_t *lv_display = NULL;
static esp_lcd_touch_handle_t tp_handle = NULL;

long map(long x, long in_min, long in_max, long out_min, long out_max) {
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}

static bool notify_lvgl_flush_ready(esp_lcd_panel_io_handle_t panel_io, esp_lcd_panel_io_event_data_t *edata, void *user_ctx) {
    if (lv_display) lv_display_flush_ready(lv_display);
    return false;
}

static void example_disp_flush(lv_display_t *disp, const lv_area_t *area, uint8_t *px_map) {
    esp_lcd_panel_handle_t panel_handle = (esp_lcd_panel_handle_t) lv_display_get_user_data(disp);
    uint32_t len = (area->x2 - area->x1 + 1) * (area->y2 - area->y1 + 1);
    uint16_t *buf16 = (uint16_t *)px_map;
    for (uint32_t i = 0; i < len; i++) {
        buf16[i] = (buf16[i] << 8) | (buf16[i] >> 8);
    }
    esp_lcd_panel_draw_bitmap(panel_handle, area->x1, area->y1, area->x2 + 1, area->y2 + 1, px_map);
}

// --- ЧТЕНИЕ ТАЧСКРИНА ---
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wdeprecated-declarations"

static void lvgl_touch_read(lv_indev_t * indev, lv_indev_data_t * data) {
    uint16_t touch_x[1];
    uint16_t touch_y[1];
    uint8_t touch_cnt = 0;

    esp_lcd_touch_read_data(tp_handle);
    bool touchpad_pressed = esp_lcd_touch_get_coordinates(tp_handle, touch_x, touch_y, NULL, &touch_cnt, 1);

    if (touchpad_pressed && touch_cnt > 0) {
        data->state = LV_INDEV_STATE_PRESSED;

        int raw_x = touch_x[0];
        int raw_y = touch_y[0];

        int cal_x = map(raw_y, TOUCH_RAW_Y_MIN, TOUCH_RAW_Y_MAX, 320, 0);
        int cal_y = map(raw_x, TOUCH_RAW_X_MIN, TOUCH_RAW_X_MAX, 240, 0);

        if (cal_x < 0) cal_x = 0;
        if (cal_x > 319) cal_x = 319;
        if (cal_y < 0) cal_y = 0;
        if (cal_y > 239) cal_y = 239;

        data->point.x = cal_x;
        data->point.y = cal_y;
    } else {
```

```
        if (cal_y < 0) cal_y = 0;
        if (cal_y > 239) cal_y = 239;

        data->point.x = cal_x;
        data->point.y = cal_y;
    } else {
        data->state = LV_INDEV_STATE_RELEASED;
    }
}
#pragma GCC diagnostic pop

static void lvgl_tick_cb(void *arg) {
    (void) arg;
    lv_tick_inc(5);
}

// --- ОБНОВЛЕНИЕ UI ---
void update_ui_status(bool alarm_active) {
    if (!objects.status_indicator) return;

    if (alarm_active) {
        // === ТРЕВОГА (КРАСНЫЙ) ===
        lv_obj_set_style_bg_color(objects.status_indicator, lv_color_hex(0xFF5555), LV_PART_MAIN);

        // Английский текст, чтобы убрать квадратики
        if(objects.status_text) lv_label_set_text(objects.status_text, "MOTION DETECTED!");
        if(objects.status_icon) lv_label_set_text(objects.status_icon, "\uf071");
    } else {
        // === ПОКОЙ (ЗЕЛЕНЫЙ) ===
        lv_obj_set_style_bg_color(objects.status_indicator, lv_color_hex(0x4CD964), LV_PART_MAIN);

        // Английский текст
        if(objects.status_text) lv_label_set_text(objects.status_text, "System Safe");
        if(objects.status_icon) lv_label_set_text(objects.status_icon, "\uf00c");
    }
}

// --- ГЛАВНАЯ ЛОГИКА ---
void lvgl_task(void *pvParameter) {
    bool last_final_state = false;

    while (1) {
        // 1. Свитч
        bool is_system_armed = true;
        if (objects.led_swith) {
            is_system_armed = lv_obj_has_state(objects.led_swith, LV_STATE_CHECKED);
        }

        // 2. Датчик
        int raw_sensor = gpio_get_level(PIN_MOTION_SENSOR);

        // === ИСПРАВЛЕНИЕ ЛОГИКИ ===
        // Я заменил (raw_sensor == 1) на (raw_sensor == 0).
        // Теперь, если при 1 была ошибка, при 0 должно работать верно.
        bool motion_detected = (raw_sensor == 0);

        // 3. Решение
        bool trigger = false;
        if (is_system_armed && motion_detected) {
            trigger = true;
        }

        // 4. Обновление
        if (trigger != last_final_state) {
            update_ui_status(trigger);
            last_final_state = trigger;
        }

        lv_timer_handler();
        vTaskDelay(pdMS_TO_TICKS(10));
    }
}

void app_main(void) {
    // Датчик
    gpio_reset_pin(PIN_MOTION_SENSOR);
    gpio_set_direction(PIN_MOTION_SENSOR, GPIO_MODE_INPUT);
    gpio_set_pull_mode(PIN_MOTION_SENSOR, GPIO_PULLDOWN_ONLY);

    // SPI
    spi_bus_config_t buscfg = {
        .sclk_io_num = PIN_NUM_SCLK,
        .mosi_io_num = PIN_NUM_MOSI,
        .miso_io_num = PIN_NUM_MISO,
        .quadwp_io_num = -1,
        .quadhd_io_num = -1,
        .max_transfer_sz = LVGL_BUF_SIZE + 100,
    };
    ESP_ERROR_CHECK(spi_bus_initialize(LCD_HOST, &buscfg, SPI_DMA_CH_AUTO));

    // LCD
    esp_lcd_panel_io_handle_t io_handle = NULL;
    esp_lcd_panel_io_spi_config_t io_config = {
        .dc_gpio_num = PIN_NUM_DC,
        .cs_gpio_num = PIN_NUM_CS,
        .pclk_hz = LCD_PIXEL_CLOCK_HZ,
        .lcd_cmd_bits = 8,
        .lcd_param_bits = 8,
        .spi_mode = 0,
        .trans_queue_depth = 10,
        .on_color_trans_done = notify_lvgl_flush_ready,
    };
```

```
        lv_timer_handler();
        vTaskDelay(pdMS_TO_TICKS(10));
    }
}

void app_main(void) {
    // Датчик
    gpio_reset_pin(PIN_MOTION_SENSOR);
    gpio_set_direction(PIN_MOTION_SENSOR, GPIO_MODE_INPUT);
    gpio_set_pull_mode(PIN_MOTION_SENSOR, GPIO_PULLDOWN_ONLY);

    // SPI
    spi_bus_config_t buscfg = {
        .sclk_io_num = PIN_NUM_SCLK,
        .mosi_io_num = PIN_NUM_MOSI,
        .miso_io_num = PIN_NUM_MISO,
        .quadwp_io_num = -1,
        .quadhd_io_num = -1,
        .max_transfer_sz = LVGL_BUF_SIZE + 100,
    };
    ESP_ERROR_CHECK(spi_bus_initialize(LCD_HOST, &buscfg, SPI_DMA_CH_AUTO));

    // LCD
    esp_lcd_panel_io_handle_t io_handle = NULL;
    esp_lcd_panel_io_spi_config_t io_config = {
        .dc_gpio_num = PIN_NUM_DC,
        .cs_gpio_num = PIN_NUM_CS,
        .pclk_hz = LCD_PIXEL_CLOCK_HZ,
        .lcd_cmd_bits = 8,
        .lcd_param_bits = 8,
        .spi_mode = 0,
        .trans_queue_depth = 10,
        .on_color_trans_done = notify_lvgl_flush_ready,
    };
    ESP_ERROR_CHECK(esp_lcd_new_panel_io_spi((esp_lcd_spi_bus_handle_t)LCD_HOST, &io_config, &io_handle));

    esp_lcd_panel_handle_t panel_handle = NULL;
    esp_lcd_panel_dev_config_t panel_config = {
        .reset_gpio_num = PIN_NUM_RST,
        .rgb_endian = LCD_RGB_ENDIAN_BGR,
        .bits_per_pixel = 16,
    };
    ESP_ERROR_CHECK(esp_lcd_new_panel_ili9341(io_handle, &panel_config, &panel_handle));
    ESP_ERROR_CHECK(esp_lcd_panel_reset(panel_handle));
    ESP_ERROR_CHECK(esp_lcd_panel_init(panel_handle));
    ESP_ERROR_CHECK(esp_lcd_panel_invert_color(panel_handle, false));
    ESP_ERROR_CHECK(esp_lcd_panel_swap_xy(panel_handle, true));
    ESP_ERROR_CHECK(esp_lcd_panel_mirror(panel_handle, true, true));
    ESP_ERROR_CHECK(esp_lcd_panel_disp_on_off(panel_handle, true));

    gpio_set_direction(PIN_NUM_BCKL, GPIO_MODE_OUTPUT);
    gpio_set_level(PIN_NUM_BCKL, 1);

    // TOUCH
    esp_lcd_panel_io_handle_t tp_io_handle = NULL;
    esp_lcd_panel_io_spi_config_t tp_io_config = {
        .cs_gpio_num = PIN_NUM_TOUCH_CS,
        .pclk_hz = 2 * 1000 * 1000,
        .lcd_cmd_bits = 8,
        .lcd_param_bits = 8,
        .spi_mode = 0,
        .trans_queue_depth = 10,
    };
    ESP_ERROR_CHECK(esp_lcd_new_panel_io_spi((esp_lcd_spi_bus_handle_t)TOUCH_HOST, &tp_io_config, &tp_io_handle));

    esp_lcd_touch_config_t tp_cfg = {
        .x_max = LCD_V_RES_PHYS,
        .y_max = LCD_H_RES_PHYS,
        .rst_gpio_num = -1,
        .int_gpio_num = PIN_NUM_TOUCH_IRQ,
        .levels = { .reset = 0, .interrupt = 0 },
        .flags = { .swap_xy = 0, .mirror_x = 0, .mirror_y = 0 },
    };
    ESP_ERROR_CHECK(esp_lcd_touch_new_spi_xpt2046(tp_io_handle, &tp_cfg, &tp_handle));

    // LVGL
    lv_init();
    lv_display_t *disp = lv_display_create(LCD_V_RES_PHYS, LCD_H_RES_PHYS);
    lv_display = disp;
    void *buf1 = heap_caps_malloc(LVGL_BUF_SIZE, MALLOC_CAP_DMA);
    lv_display_set_buffers(disp, buf1, NULL, LVGL_BUF_SIZE, LV_DISPLAY_RENDER_MODE_PARTIAL);
    lv_display_set_flush_cb(disp, example_disp_flush);
    lv_display_set_user_data(disp, panel_handle);

    lv_indev_t * indev = lv_indev_create();
    lv_indev_set_type(indev, LV_INDEV_TYPE_POINTER);
    lv_indev_set_read_cb(indev, lvgl_touch_read);

    const esp_timer_create_args_t periodic_timer_args = {
        .callback = &lvgl_tick_cb, .name = "lvgl_tick"
    };
    esp_timer_handle_t periodic_timer;
    ESP_ERROR_CHECK(esp_timer_create(&periodic_timer_args, &periodic_timer));
    ESP_ERROR_CHECK(esp_timer_start_periodic(periodic_timer, 5000));

    // UI
    ui_init();
    xTaskCreatePinnedToCore(lvgl_task, "LVGL", 4096 * 4, NULL, 5, NULL, 1);
}
```