1

LVGL 框架编程与配置说明书

Version:v1.0

Date: 2023.12

FREQCHIP
富芮坤

修订版本

版本	日期	更新内容
V1.0	2023.12.19	首版

目录

1. 概述

LVGL（LittlevGL）是一个轻量级的嵌入式图形用户界面（GUI）库。它设计用于嵌入式系统，提供了丰富的图形界面元素，使开发者能够轻松地创建直观、交互性强的用户界面。系统框图如下图所示。

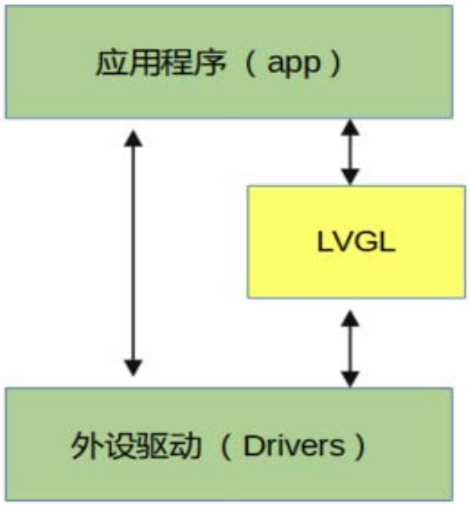


图 1-1 LVGL 系统框图

LVGL 作为一个图形库，被应用程序调用以创建图形用户界面（GUI）。该库包含一个 HAL（硬件抽象层）接口，用于注册显示和输入设备的驱动程序。这些驱动程序不仅负责与特定硬件的交互，还可能包含其他功能，例如驱动显示器到 GPU（如有）以及读取触摸板或按钮/编码器的输入。对于不同类型的显示屏，存在两种典型的设置。一种是带有内置 RAM 的 LCD/TFT 显示屏，另一种是没有内置 RAM 的 LCD/TFT 显示屏。在这两种情况下，都需要一个帧缓冲区来存储屏幕的当前图像，但是对 RAM 的需求大小不同。

1.1. LCD 显示屏有内置 RAM

帧缓冲区：帧缓冲区从 MCU 内部分配 RAM 大小调整，最小 1/10 屏幕大小的 Buffer。驱动方式：显示器的驱动由 MCU 外设接口驱动，LVGL 库通过 HAL 接口调用相应的驱动程序接口，可以按块渲染刷新，通过 LCD 相关的开窗指令更新区域。

1.2. LCD 显示屏没有内置 RAM

帧缓冲区：帧缓冲区至少需要一帧整个屏幕大小的 Buffer。

驱动方式：显示器的驱动由 MCU 外设接口驱动，LVGL 库通过 HAL 接口调用相应的驱动程序接口，不能按块单独刷新，同时需要实时刷新，每次刷新时都要发送一整屏的数据到 LCD。

1.3. LVGL 渲染方式：

LVGL 的绘制不是直接绘制到屏幕上，而是先绘制到内部缓冲区。当绘图（渲染）准备好时，才将该缓冲区的内容刷新到屏幕上。与直接绘制到屏幕相比，这种方法有两个主要优点：

a) 避免绘制界面时的闪烁：如果 LVGL 直接绘制到显示中，例如在绘制背景、按钮和文本时，每个阶段都可能在短时间内可见，导致屏幕闪烁。通过先在内部缓冲区绘制，可以在准备好整个图层后一次性刷新到屏幕，避免了中间阶段的可见性。

b) 提高绘制效率：绘制到内部 RAM 中的缓冲区并最终一次写入，比在每个像素访问时直接读取/写入显示更快。这对于使用带有显存的显示控制器情况很有效。

注意：这个绘制方式与传统的双缓冲不同。传统的双缓冲通常需要两个屏幕大小的帧缓冲区，一个保存当前图像以显示在显示器上，而渲染发生在另一个非活动的帧缓冲区中。使用 LVGL，不必存储两个帧缓冲区（通常需要外部 RAM），而只需存储更小的绘图缓冲区，它可以轻松地加载到内部 RAM 中。这样的设计更为节省资源。

1.4. LVGL 更新屏幕流程

a) 界面发生了一些需要重绘的事件，比如按钮按下、更改图标、发生动画等。

b) LVGL 将改变对象的旧区域和新区域保存到一个称为无效区缓冲区的缓冲区中。在某些情况下，不会将对象添加到缓冲区中，例如隐藏对象、完全脱离其父对象的对象、部分超出父级区域并被裁剪到父级区域以及其他屏幕上的对象。

c) 每隔 LV_DISP_DEF_REFR_PERIOD（这个在 lv_conf.h 中设置）的时间间隔发生以下情况：

1) LVGL 检查无效区域并连接相邻或相交的区域。
2) 获取第一个连接区域，如果它小于绘图缓冲区，则只需将该区域的内容渲染到绘图缓冲区中。如果该区域不适合缓冲区，则在绘图缓冲区中绘制尽可能多的线。
3) 当区域被渲染时，从显示驱动程序调用 flush_cb 回调函数来刷新显示。
4) 如果该区域大于缓冲区，也渲染其余部分。
5) 对所有连接的区域执行相同的操作。

这个过程确保了在需要重绘的情况下，LVGL 会有效地管理和渲染改变的区域，最终刷新到屏幕上，同时通过连接相邻或相交的区域，提高渲染效率。

d) 当需要重绘一个区域时，LVGL 将搜索并找到覆盖该区域的最顶层对象，并从该对象开始绘制，比如说按钮的标签发生了变化，LVGL 只会判断只需要在文本下方绘制按钮，而不必在按钮下方渲染整个屏幕。这种策略确保在渲染时只关注发生变化的区域及其最顶层的对象，提高了渲染的效率。

2. LVGL 初始配置

在使用 LVGL 创建 GUI 界面时，需要按照以下顺序进行初始化。

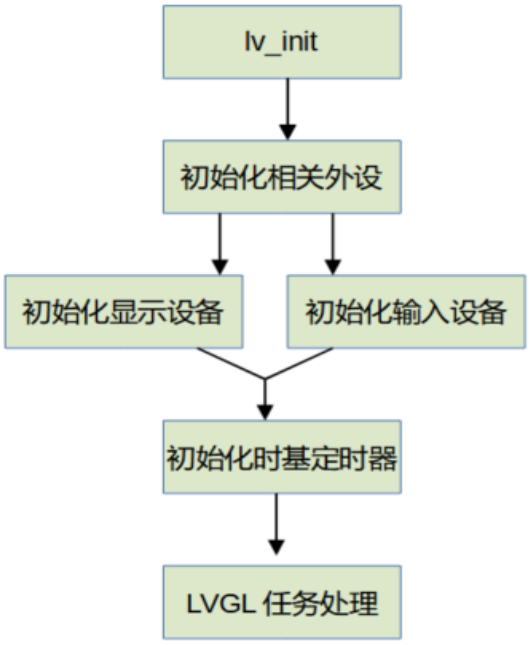


图 2-1 初始化流程图

1) 调用 lv_init() 初始化 LVGL 库

LVGL 库的初始化是整个过程的第一步，通过调用 lv_init()函数来初始化 LVGL 库，为后续的操作做好准备。

2) 初始化驱动程序

针对你的硬件设置，需要初始化显示和输入设备的驱动程序。这包括配置显示器的驱动方式、初始化触摸屏或按钮的输入驱动等。

3) 在 LVGL 中注册显示和输入设备驱动程序

将已初始化的显示和输入设备驱动程序注册到 LVGL 库中，这样 LVGL 就能够正确地与硬件进行交互。这通常通过调用 LVGL 提供的接口函数来完成。

4) 在中断中每隔 x 毫秒调用 lv_tick_inc(x)

LVGL 需要了解经过的时间，以便进行一些时间相关的操作，比如动画、计时器等。因此，在中断服务中，需要以固定的时间间隔调用 lv_tick_inc(x)，告知 LVGL 已经过去了多少时间。

5) 每隔 x 毫秒定期调用 lv_task_handler()

LVGL 库使用任务来管理各种图形元素的更新和绘制。因此需要定期调用 lv_task_handler()函数，以处理与 LVGL 相关的任务，确保图形界面的正常运行一般来说 lv_task_handler()应该在主循环中被定期调用。

3. 显示驱动器配置

为了设置显示,需要初始化 lv_disp_buf_t 和 lv_disp_drv_t 变量。这两个结构体分别用于保存显示缓冲区的信息和显示驱动程序的相关信息。

3.1. lv_disp_buf_t 结构体

lv_disp_buf_t 结构体保存了显示缓冲区的信息。显示缓冲区是用于存储屏幕当前图像的内存区域。这个结构体的初始化通常包含了以下信息：

buf1 和 buf2：指向两个显示缓冲区的指针，这两个缓冲区在双缓冲模式下交替使用，可以提高渲染帧率。size：每个缓冲区的大小，以字节为单位。换句话说，lv_disp_buf_t 结构体定义了 LVGL 库如何管理和使用显示缓冲区。

3.2. lv_disp_drv_t 结构体

lv_disp_drv_t 结构体用于定义 HAL 要注册的显示驱动程序的相关信息。

这个结构体的初始化通常包含了以下信息：

draw_buf:指向 lv_disp_buf_t 结构体的指针，表示使用的显示缓冲区信息。

flush_cb：刷新回调函数，用于将显示缓冲区的内容刷新到实际的屏幕上。

hor_res 和 ver_res：屏幕的水平和垂直分辨率,程序中用 LV_HOR_RES_MAX，LV_VER_RES_MAX 来配置，这个要跟屏幕大小配置一样。

rotated 交换 hor_res 和 ver_res 。两种情况下 LVGL 的绘制方向相同 (从上到下的线条)，因 此还需要重新配置驱动程序以更改显示器的填充方向。其他与图形相关的设置，例如颜色深度、像素格式等。

sw_rotate：设置软件旋转功能，非常耗算力，一般不推荐使用，最好是使用 lcd 自带的旋转功能。这个结构体还可以包含回调函数，用于处理与图形相关的事件，例如屏幕旋转、屏幕休眠等。

配置代码如下图所示。

```c
static uint16_t display_framebuffer[LV_HOR_RES_MAX * LV_VER_RES_MAX];
static void lvgl_init(void)
{
    lv_init();

    lv_disp_draw_buf_init(&disp_buf, (void *)display_framebuffer, NULL, LV_HOR_RES_MAX * LV_VER_RES_MAX);
    static lv_disp_drv_t disp_drv;          /*Descriptor of a display driver*/
    lv_disp_drv_init(&disp_drv);            /*Basic initialization*/
    disp_drv.flush_cb = my_disp_flush;      /*Set your driver function*/
    disp_drv.draw_buf = &disp_buf;          /*Assign the buffer to the display*/
    disp_drv.hor_res = LV_HOR_RES_MAX;
    disp_drv.ver_res = LV_VER_RES_MAX;
    disp_drv.physical_hor_res = -1;
    disp_drv.physical_ver_res = -1;
    disp_drv.offset_x = 0;
    disp_drv.offset_y = 0;
    disp_drv.full_refresh = 1;
    lv_disp_drv_register(&disp_drv);        /*Finally register the driver*/
```

图 3-1 配置代码示例

将缓冲区的内容通过 LCD 显示相关 HAL 接口，发送到液晶显示，代码如下所示。

```c
static void my_disp_flush(lv_disp_drv_t * disp, const lv_area_t * area, lv_color_t * color_p)
{
#if 1
    if(!lv_fetch_disp_buffer_func(area))
    {
        lv_disp_flush_ready(disp);
        return;
    }
    display_set_window(HOR_OFFSET+area->x1,
                       HOR_OFFSET+area->x2,
                       VER_OFFSET+area->y1,
                       VER_OFFSET+area->y2);
    display_update_dma((area->x2+1-area->x1)*(area->y2+1-area->y1), 16, (void *)color_p);

}
```

图 3-2 显示相关配置代码

4. 输入设备配置

通过 lv_indev_drv_t 结构体变量来配置输入设备相关信息

输入设备类型:

- LV_INDEV_TYPE_POINTER 触摸板或鼠标
- LV_INDEV_TYPE_KEYPAD 键盘
- LV_INDEV_TYPE_ENCODER 编码器
- LV_INDEV_TYPE_BUTTON 外部按钮

输入设备读取回调函数:

read_cb（indev_drv.read_cb）是一个函数指针，用于定期读取调用以上报输入设备的当前状态。这个函数有两个主要功能：

1. 上报输入设备的当前状态

   当调用 read_cb 时，它应该返回当前输入设备的状态，例如鼠标的位置、按钮的状态等。LVGL 库通过调用这个函数来获取最新的输入设备状态，以便更新和处理图形界面。

2. 缓冲数据并返回状态

   read_cb 函数还可以负责缓冲输入设备的数据，并在两种情况下返回不同的状态：

   a) 如果缓冲区不为空，函数返回 true，表示有更多的数据要读取。

   b) 如果缓冲区为空，函数返回 false，表示当前没有更多的数据要读取。

   通过这种机制，LVGL 库可以根据 read_cb 的返回值来确定是否需要继续调用该函数以获取输入设备的状态。如果返回 true，则表示缓冲区中还有未处理的数据，需要继续调用以获取最新的状态。如果返回 false，则表示当前没有更多的数据可用，LVGL 将等待下一次调用 read_cb 以获取最新的输入设备状态。

   输入设备初始化代码（触摸+KEYPAD）如下图所示。

```
    static lv_indev_drv_t indev_drv;         /*Descriptor of a input device driver*/
    lv_indev_drv_init(&indev_drv);           /*Basic initialization*/
    indev_drv.type = LV_INDEV_TYPE_POINTER; /*Touch pad is a pointer-like device*/
    indev_drv.read_cb = my_touchpad_read;    /*Set your driver function*/
    lv_indev_drv_register(&indev_drv);       /*Finally register the driver*/
    static lv_indev_drv_t indev_drv1;
    lv_indev_drv_init(&indev_drv1);
    indev_drv1.type = LV_INDEV_TYPE_KEYPAD;
    indev_drv1.read_cb = keypad_read;
    indev_keypad = lv_indev_drv_register(&indev_drv1);
```

图 4-1 设备初始化示例代码

回调函数里面实现读取触摸坐标值，按键编码器输入 KEY 值代码如下图所示。

```
static void my_touchpad_read(struct _lv_indev_drv_t * indev, lv_indev_data_t * data)
{
    int16_t x, y;bool pressed;
    pressed = touchpad_read(&x, &y);
    if (pressed) {
        if ((x < HOR_OFFSET)
                || (x >= HOR_OFFSET + LV_HOR_RES_MAX)
                || (y < VER_OFFSET)
                || (y >= (VER_OFFSET + LV_VER_RES_MAX))) {
            data->state = LV_INDEV_STATE_REL;
        }
        else {
            data->state = LV_INDEV_STATE_PR;
            data->point.x = x - HOR_OFFSET;
            data->point.y = y - VER_OFFSET;
        }
        setup_gui_running();
    }
    else
    {
        data->state = LV_INDEV_STATE_REL;
    }
}

static void keypad_read(lv_indev_drv_t * indev_drv, lv_indev_data_t * data)
{
    static uint32_t last_key = 0;
    uint32_t act_key = keypad_get_key();
    if(act_key != 0) {···
    } else {
        data->state = LV_INDEV_STATE_REL;
    }
    data->key = last_key;
}
```

图 4-2 按键编码器输入 KEY 值示例代码

5. LVGL 系统滴答时基

滴答时基（Tick Timebase）是一个用于跟踪时间流逝的机制，用于支持与时间相关的功能，例如动画、计时器、事件超时等。

为了确保 LVGL 库能够准确地跟踪时间，需要定期调用 lv_tick_inc(tick_period)函数，并以毫秒为单位指定调用的周期。例如通过使用 lv_tick_inc(1)，可以表示每毫秒调用一次。需要将 lv_tick_inc 函数放置在一个比 lv_task_handler()更高优先级的例程中，例如放在中断服务中。这样即使 lv_task_handler()执行花费较长时间，LVGL 库也能够在更高的优先级例程中及时更新时间，使用 FreeRTOS 时，可以在 vApplicationTickHook 中调用 lv_tick_inc，如下图所示。

```
void vApplicationTickHook(void)
{

    lv_tick_inc(1);
#if ENABLE_RTOS_MONITOR == 1
    CPU_RunTime++;
#endif
}
```

图 5-1 vApplicationTickHook 函数示例

6. LVGL 任务处理

6.1. LVGL 任务机制

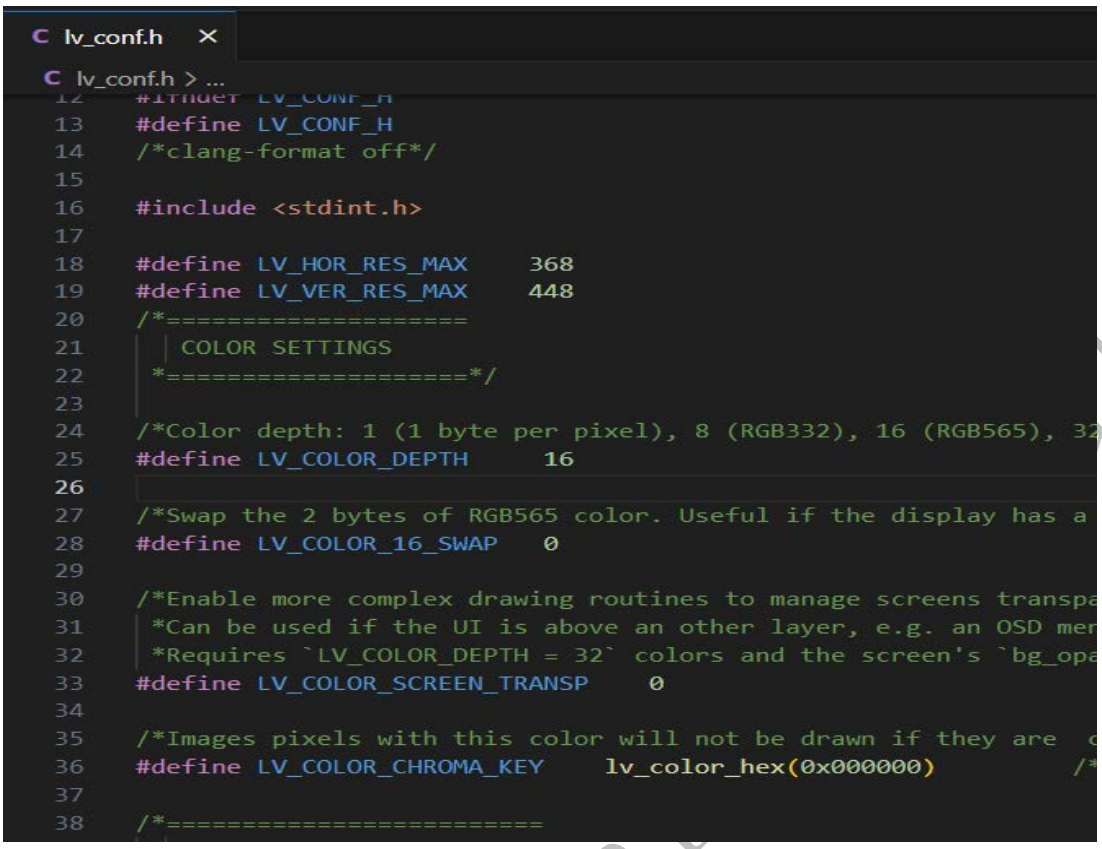
LVGL 库通过任务机制来管理和更新图形元素，包括控件的刷新、动画的更新、事件的处理等，通过函数 lv_task_handler 来处理的。一般来说 lv_task_handler() 应该在主循环中被定期调用。通常建议将其放置在一个周期性的定时器中，以确保图形界面能够得到及时的处理。例如，在每个主循环迭代中都调用 lv_task_handler()，以确保及时处理所有与图形界面相关的任务和事件。在我们的 SDK 工程系统中专门创建了一个任务来处理，需要注意的是 LVGL 默认情况下不是线程安全的，也就是在 RTOS 系统中使用时不能多个任务里面去调用 LVGL 相关的 API,应该通过发送消息事件到当前 GUI 任务中去处理显示。

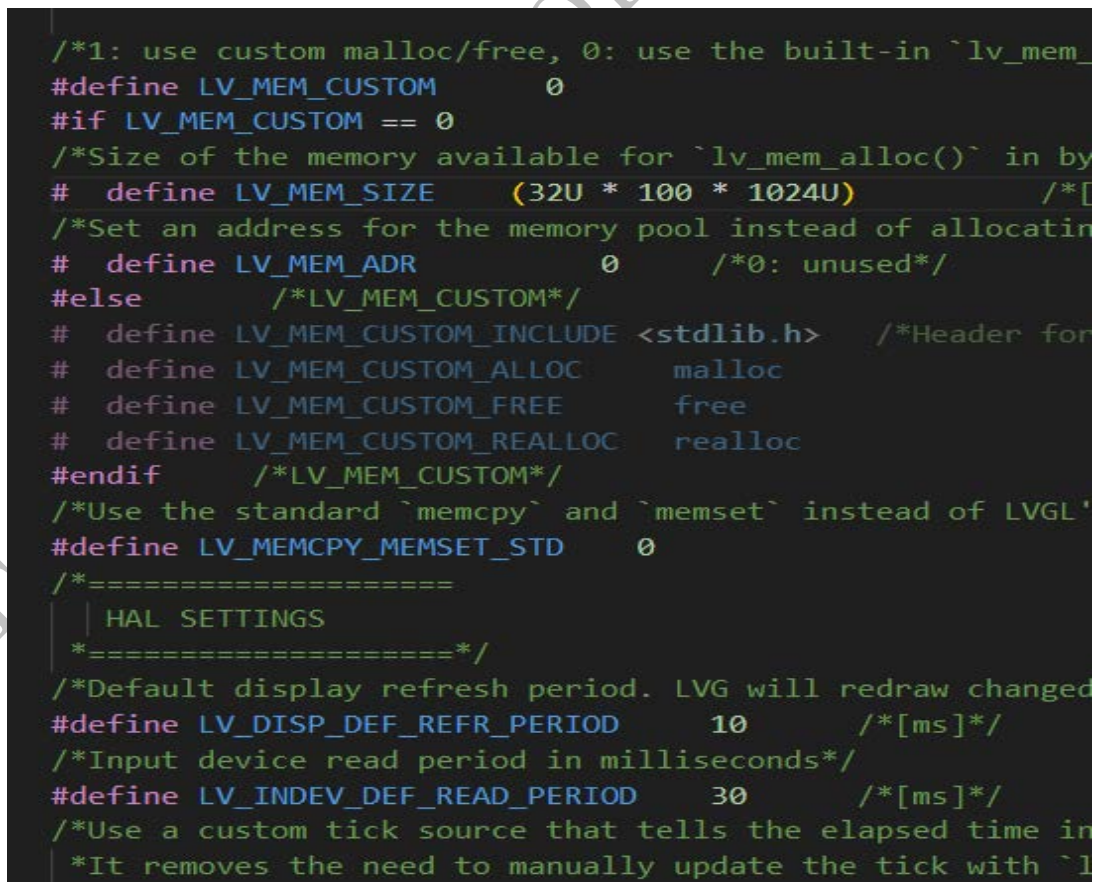
```
//gui task
static void gui_task(void *arg)
{
    gui_task_msg_t queue_event;
    printf("gui_task \r\n");
    /* lfs init */
    lfs_custom_init();
    dev_rtc_time_init();
    lvgl_init();
    // Create  queue for gui task
    gui_queue_handle = xQueueCreate(GUI_TASK_QUEUE_LENGTH, sizeof(gui_task_msg_t));
    //battery detect
    adc_vbat_start_detect();
    setup_gui_running();
    while(1) {
        vTaskDelay(1);
        lv_timer_handler();
        if (xQueueReceive(gui_queue_handle, &queue_event, 0) == pdPASS)
        {
            gui_task_queue_callback(&queue_event);
        }       owen, 3个月前 • 1. initial version
    }
}
```

图 6-1 gui_task 函数示例

6.2. lv_config.h 文件配置

下图中这些参数是我们平时使用中会涉及到的一些配置。





a) LV_HOR_RES_MAX，LV_VER_RES_MAX 配置屏幕分辨率大小。

b) LV_COLOR_DEPTH 颜色深度

c) LV_COLOR_16_SWAP 设置高低字节交换，有些硬件采用大端字节序，而有些采用小端字节序因此，LV_COLOR_16_SWAP 的定义可能需要根据实际硬件平台进行适当的设置，以确保正确的颜色显示。

d) LV_MEM_CUSTOM 用户可以自行实现 LVGL 使用的内存分配和释放函数，以适应特定的应用场景，设置为 0 的话需要设置一个内存地址给内存池来分配，而不是将其作为普通数组分配，可以在外部 SRAM/PSRAM 分配。设置为 1 使用从用户实现的 malloc/free 函数来分配。

e) LV_DISP_DEF_REFR_PERIOD 表示默认的屏幕刷新周期，即多久刷新一次屏幕内容。

在 LVGL 中，屏幕的刷新是通过 lv_refr_now 和 lv_task_handler 来触发的。lv_refr_now 函数用于立即刷新屏幕，而 lv_task_handler 函数用于定期刷新。LV_DISP_DEF_REFR_PERIOD 宏定义了定期刷新的默认周期的毫秒数。

f) LV_INDEV_DEF_READ_PERIOD 是用于定义默认输入设备读取周期的宏。表示默认情况下 LVGL 库读取输入设备的周期，即多久读取一次输入设备的状态。

7. LVGL 页面框架介绍

7.1. 主界面框图

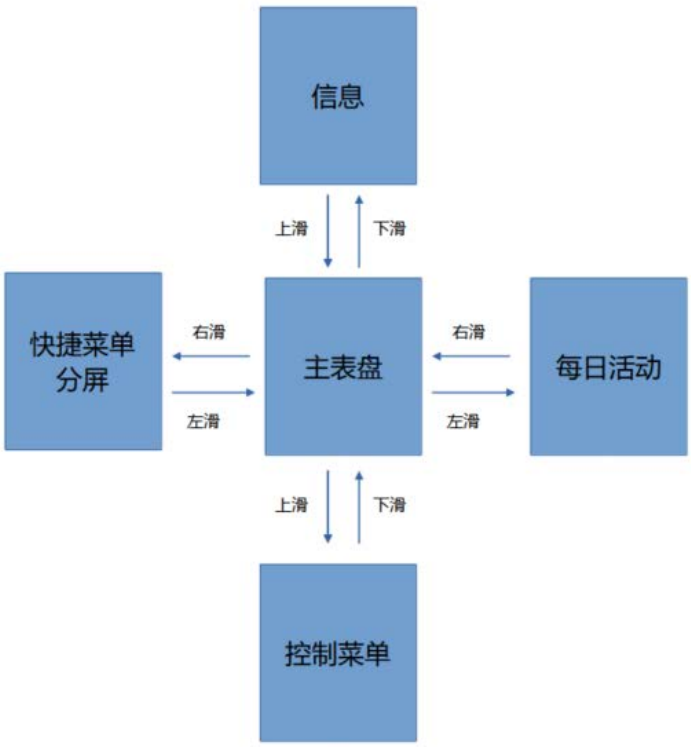


图 7-1 主界面系统框图

目前 trunkey 工程框架已经实现了主界面交互逻辑，子界面 ui 节点栈管理，用户只需要将需要的功能 func 页面函数放到对应页面表中即可。

7.2. LVGL 应用入口函数

lv_prj_main 函数为 lvgl 应用逻辑主入口函数，内部实现了外部表盘的加载，主界面数量的初始化配置，父对象 prj_parent_cont 的创建，父对象事件的添加，工作组的创建绑定，将父对象添加到组中，再创建一个 prj_prev_cont 对象，做为当前活动界面的父对象。然后将 ui 节点栈进行初始化，最后对表盘界面的初始化，进入到主表盘界面。

```
void lv_prj_main(void)
{
    // ui_clock_group[fr_ui_watchface_total()- 1] = (ui_func)*entry_main;
    // //配置自动表盘调用函数
    extern_ui_api_config();
    fr_load_app_from_extern_flash(0xA00000);    //0x940000 is address wf in extern flash
    ///初始化自定义卡片
    fr_lv_custom_page_init();
    prj_parent_cont = lv_obj_create(lv_scr_act());
    lv_obj_set_style_pad_all(prj_parent_cont, 0, 0);
    lv_obj_set_style_border_width(prj_parent_cont, 0, 0);
    lv_obj_set_size(prj_parent_cont, LV_PCT(100), LV_PCT(100));
    lv_obj_add_flag(prj_parent_cont, LV_OBJ_FLAG_SCROLL_ONE);
    lv_obj_set_scroll_snap_x(prj_parent_cont, LV_SCROLL_SNAP_CENTER);
    lv_obj_set_scroll_snap_y(prj_parent_cont, LV_SCROLL_SNAP_CENTER);
    lv_obj_set_scrollbar_mode(prj_parent_cont, LV_SCROLLBAR_MODE_OFF);
    lv_obj_clear_flag(prj_parent_cont, LV_OBJ_FLAG_SCROLL_ELASTIC);
    lv_obj_add_event_cb(prj_parent_cont, main_event_handler, LV_EVENT_ALL, NULL);
    prj_group = lv_group_create();
    lv_indev_set_group(indev_keypad, prj_group);
    lv_group_add_obj(prj_group, prj_parent_cont);
    group_tail_obj = prj_parent_cont;
    prj_prev_cont = lv_obj_create(prj_parent_cont);
    //frame_log_enable = true;
    //printf("frame_log_enable:%d\r\n",frame_log_enable);
    fr_system_set_language(1);
    fr_system_set_step_target(5000);
    page_hor_id = 0;
    if (dsp_get_ui_scene_page_id() > 3)···

    // ota 页面优先···
    if(dsp_get_ui_scene_mode() == UI_APP_OTA)···
    else if(dsp_get_ui_scene_mode() == 6)    //charge···
    else
    {
        if(lv_ui_node_depth() == 0)···
```

图 7-2 lv_prj_main 函数示例

7.3. 页面表介绍

7.3.1. 风格菜单表

存放满天星列表，普通列表，轨道列表功能函数实现，用户如需增加其他列表，将实现的 func 函数添加到表中即可，函数接口格式参考《LVGL 应用编程文档》。

```
LV_ATTRIBUTE_LARGE_CONST const ui_func style_func_tab[] =
{
    fr_lv_app_starsky_func,
    fr_lv_arc_list_func,
    fr_lv_app_circle_func,
    fr_lv_track_roller_func,
    fr_lv_list_menu_func,
};
```

7.3.2. 主页面功能表

主要存放主界面的功能菜单函数，如心率，血氧，天气，血压，运动等等，这个支持用户自定义页面数量，最大支持表中添加的数量。

```
static const ui_func custom_ui_arry[CUSTOM_UI_ARRY_NUMBER]=
{
    fr_lv_add_cards_page_func,
    fr_lv_app_exercise_func,
    fr_lv_sleep_page_func,
    fr_lv_weather_report_func,
    fr_lv_pressure_measure_func,
    fr_lv_heart_rate_list_func,
    fr_lv_blood_oxygen_measure_func,
    fr_lv_blood_pressure_measure_func,
    fr_lv_app_exercise_default_list_func,
    fr_lv_pressure_emotion_func,
    fr_lv_respiratory_rate_func,
    fr_lv_body_temperature_display_func,
};
```

默认支持的数量在 fr_lv_custom_page_init 里面定义，如下图所示。

```
void fr_lv_custom_page_init(void)
{
    page_list.data = page_sqlist_space;
    page_list.length = 0;
    //---------------------------------
    uint8_t elem_len = splist_get_elem_number();
    if (elem_len == 0) // list is empty;
    {
        //clock0 and fr_lv_add_cards_page_func
        fr_lv_custom_page_add(0);
        fr_lv_custom_page_add(1);
        fr_lv_custom_page_set_watch_face(0);
        //---------------------------------
        fr_lv_custom_page_add(1);
        fr_lv_custom_page_add(2);
        fr_lv_custom_page_add(3);
        fr_lv_custom_page_add(4);
        fr_system_set_custom_page_mode();
    }
    else
    {
        uint8_t *custom_mode = fr_system_get_custom_page_mode();
        for (uint8_t i = 0; i < elem_len; i++)
        {
            fr_lv_custom_page_add(custom_mode[i]);
        }
    }
}
```

7.3.3. 时钟表盘页面

主要存放表盘时钟界面，根据用户需求自行增减，编码器切换表盘时也是使用当前这个表中定义的表盘。

```
static LV_ATTRIBUTE_LARGE_CONST const ui_func ui_clock_group[] =
{
    fr_lv_main_clock2_func,
    fr_lv_main_clock1_func,
    fr_lv_main_clock0_func,
    fr_lv_main_clock3_func,
    fr_lv_main_clock4_func,
    fr_lv_main_clock5_func,
    NULL,   //*entry_main
};
```

7.3.4. 节点页面函数表

主要存放当前工程中所有 ui 功能页面，ui 节点栈管理页面时会从当前表中获取页面，用户新增页面只需要将功能函数添加到表中即可。

```
// 节点页面的函数表，所有子页面函数都可以加入函数表以便调用
static LV_ATTRIBUTE_LARGE_CONST const ui_func ui_func_tab[] =
{

    fr_lv_app_exercise_default_list_func,
    fr_lv_exercise_record_func_ex,
    // fr_lv_target_setting_func,
    fr_lv_heart_rate_list_func,
    fr_lv_heart_rate_warning_func,
    fr_lv_heart_rate_quietness_func,
    fr_lv_heart_rate_curve_func,
    fr_lv_app_exercise_func,
    fr_lv_sleep_page_func,
    fr_lv_blood_oxygen_show_func,
    // fr_lv_app_star_sky_func,
    fr_lv_app_starsky_func,
    fr_lv_track_roller_func,
    fr_lv_setting_list_func,
    fr_lv_dorp_down_func,
    fr_lv_flashlight_func,
    fr_lv_find_phone_display_func,
    fr_lv_raise_wake_hint_func,
    fr_lv_theme_select_func,
    fr_lv_screen_off_dial_func,
    fr_lv_dial_preview_func,
    fr_lv_effective_time_home_func,
    fr_lv_time_date_home_func,
    fr_lv_dial_theme_setting_func,
    fr_lv_bright_home_func,
    fr_lv_soundvibration_home_func,
    fr_lv_password_home_func,
    fr_lv_not_disturb_func,
    fr_lv_app_downkey_func,
    fr_lv_home_short_menu_func,
    fr_lv_exercise_detection_func
```

图 7-3 功能函数列表示例

7.3.5. 主页面叠加页面处理

通过如下宏配置支持的滑动方向，在 fr_ui_func_get_float_up，fr_ui_func_get_float_down，fr_ui_func_get_float_left 中添加实现对应功能页面的函数，框架跑起来后会自动管理界面的交互。

```
// 在此配置框架支持的左，上，下叠加页面的滑动
#define FR_MAIN_ROLL_UP 0x01
#define FR_MAIN_ROLL_DOWN 0x02
#define FR_MAIN_ROLL_LEFT 0x04

uint8_t fr_main_roll_get_dir(void)
{
    return FR_MAIN_ROLL_UP | FR_MAIN_ROLL_DOWN | FR_MAIN_ROLL_LEFT;
    //return FR_MAIN_ROLL_UP | FR_MAIN_ROLL_DOWN;
}
ui_func fr_ui_func_get_float_up(void)
{
    return fr_lv_message_hint_page_func;
}

ui_func fr_ui_func_get_float_down(void)
{
    return fr_lv_dorp_down_func;
}

ui_func fr_ui_func_get_float_left(void)
{
    return fr_lv_home_short_menu_func;
}
```

图 7-4 滑动宏及函数示例

当我们要实现叠加透明效果可以在以下这些函数自定义透明度，这样在滑动过程中就可以实现透明度变化效果。

```c
void fr_frame_top_scroll_begin(lv_obj_t * obj)
{

}

void fr_frame_top_scrolling(lv_obj_t * obj)
{
    lv_coord_t temp = lv_obj_get_scroll_top(obj);
    lv_opa_t zoom = lv_map(temp,0,lv_disp_get_ver_res(lv_disp_get_default()),LV_OPA_90,LV_OPA_0);
    lv_alpha_fade_set(lv_obj_get_child(obj,0),zoom);
}

void fr_frame_top_scroll_end(lv_obj_t * obj)
{
    lv_coord_t temp = lv_obj_get_scroll_top(obj);
    lv_opa_t zoom = lv_map(temp,0,lv_disp_get_ver_res(lv_disp_get_default()),LV_OPA_100,LV_OPA_0);
    lv_alpha_fade_set(lv_obj_get_child(obj,0),zoom);
}
void fr_frame_bottom_scroll_begin(lv_obj_t * obj)
{
    lv_obj_t * child_1;
    if(lv_obj_get_child_cnt(obj) > 0)
    {
        child_1 = lv_obj_get_child(obj,1);
        if(lv_obj_is_valid(child_1))
            lv_obj_set_style_bg_opa(child_1,LV_OPA_30,0);
    }
}

void fr_frame_bottom_scrolling(lv_obj_t * obj)
{

}
```

图 7-5 透明度函数示例

7.3.6. 页面跳转功能函数

页面的跳转主要通过以下这些 api 实现，框架中通过入栈出栈的方式来管理界面跳转逻辑，用户无需关心具体实现，只要在应用层去使用 api 进行链接页面即可，同时以下 api 还支持交互特效，用户可以根据需求选择使用。

lv_ui_node_add，最普通的方式无特效，直接跳转到指定页面，lv_ui_node_alpha_fade_func,lv_ui_node_alpha_blend_add,lv_ui_node_zoom_blend_func 这三个 api 带特效支持混合缩放渐变效果。

具体使用方式参考例程中的调用方法即可：

```
void lv_ui_node_stack_init(void);
void lv_ui_node_func(lv_obj_t *parent, lv_ui_node_t *node);
void lv_ui_node_add(lv_obj_t *node_parent,lv_obj_t * obj_scroll,ui_func cur,ui_func next,bool scroll_dir);

void lv_ui_node_alpha_fade_func(lv_obj_t *parent, lv_ui_node_t *node);
void lv_ui_node_alpha_fade_add(lv_obj_t *node_parent,lv_obj_t * obj_scroll,ui_func cur,ui_func next,bool scroll_dir);

void lv_ui_node_alpha_blend_func(lv_obj_t *parent, lv_ui_node_t *node);
void lv_ui_node_alpha_blend_add(lv_obj_t *node_parent,lv_obj_t * obj_scroll,ui_func cur,ui_func next,bool scroll_dir);

void lv_ui_node_zoom_blend_func(lv_obj_t *parent, lv_ui_node_t *node);
void lv_ui_node_zoom_blend_add(lv_obj_t *node_parent,lv_obj_t * obj_scroll,ui_func cur,ui_func next,bool scroll_dir);

bool lv_ui_node_set_cur_p(lv_point_t * top);
bool lv_ui_node_get_cur(lv_ui_node_t * e);
bool lv_ui_node_set_cur(lv_ui_node_t * e);
bool lv_ui_node_pop_ex(lv_ui_node_t * e);
void lv_ui_node_hor_prev(lv_obj_t *parent);
```

图 7-6 页面跳转功能函数示例

这些 api 函数实现的跳转进入后，默认退出方式都是屏幕从右往左滑动退出到上一级界面，当我们需要其他退出方式时可以调用 lv_ui_node_return_to_prev_node 函数实现。

7.3.7. LVGL 自定义事件处理

a) 首先在 GUI 任务里面通过消息队列接收其他任务发送过来的数据，然后在对应的 cb 函数里面将事件通过 lvgl 的 lv_event_send 函数发送到父对象。

```
void gui_task_queue_callback(gui_task_msg_t * event)
{
    switch(event->msg_type)
    {
        case BUTTON_KEY_EVT:
            {…
            break;

        case ENCODE_KEY_EVT:
            {…
            break;

        case BUTTON_KEY1_EVT:
            {…
            break;

        case BUTTON_KEY2_EVT:
            {                …
            break;

        case MESSAGE_IN_EVT:
            {
                if(lv_obj_is_valid(prj_parent_cont))
                {
                    lv_event_send(prj_parent_cont, LV_EVENT_ANCS_MSG_IN, NULL);
                }
            }
            break;

        case PHONE_CALL_IN_EVT:
            {
                uint8_t call_mode = user_get_call_mode();
                if(lv_obj_is_valid(prj_parent_cont))
                {
                    if(call_mode!=5)
                    {
                        lv_event_send(prj_parent_cont, LV_EVENT_CALL_IN, NULL);
                    }else{
```

图 7-7 gui 任务消息队列回调函数示例

b) 父对象的回调函数里面处理相关事件和显示，如下图，接收到来电事件，和消息事件，这些都属于用户自定义事件，在回调里面创建新的 obj 对象来处理页面显示弹框信息，当点击删除时也不影响 prj_prev_cont 原来的显示页面内容。其他用户事件处理可以参考类似这种方式来实现。

```c
static void main_event_handler(lv_event_t *e)
{
    lv_event_code_t event = lv_event_get_code(e);
    lv_obj_t *parent = lv_event_get_target(e);
    if(!lv_obj_is_valid(parent))
        return;
    if(event == LV_EVENT_CALL_IN)
    {
        //来电
        if(!lv_obj_is_valid(prj_call_cont) && (prj_call_cont==NULL))
        {
            prj_call_cont = lv_obj_create(parent);
            fr_lv_app_dialing_function(prj_call_cont,NULL);
        }

    }
    else if(event == LV_EVENT_FLEXIBLE_IN)
    {
        printf("LV_EVENT_FLEXIBLE_IN\r\n");
        if(prj_flexible_cont != NULL)
        {
            lv_obj_del(prj_flexible_cont);
            prj_flexible_cont = NULL;
        }
        prj_flexible_cont = lv_obj_create(parent);
        fr_flexible_msg_set_type(FLEXIBLE_PHONE_CALL);
        fr_lv_flexible_msg_func(prj_flexible_cont,NULL);
    }
    else if(event == LV_EVENT_CALL_OK)
    {
        printf("LV_EVENT_CALL_OK\r\n");
        if(lv_obj_is_valid(prj_flexible_cont)&&(prj_flexible_cont!=NULL))
            return;
        //接通
        if(prj_call_cont != NULL)
```

图 7-8 父对象回调函数示例

c) 编码器事件的处理，当我们系统中有按键或者编码器时，需要将按键值和编码器值通过 KEY 事件来传递，这里需要注意的是我们需要编码器来控制或者改变的页面对象内容，可以将其添加到 prj_group 组中，当有 KEY 值上报时，绑定到组中的对象就能接收到 KEY 事件，回调中就可以处理当前事件。如下图实例，通过编码器的值来滚动菜单列表。

```c
void fr_lv_list_menu_func(lv_obj_t *parent, lv_point_t *top) //列表菜单
{
    if(!lv_obj_is_valid(parent))
        return;
    UI_PARENT_INIT(parent);
    lv_obj_set_scrollbar_mode(parent, LV_SCROLLBAR_MODE_OFF);
    lv_obj_t *sub_cont1 = lv_obj_create(parent);
    lv_obj_set_size(sub_cont1,LV_PCT(90),LV_PCT(100));
    lv_obj_set_style_bg_color(sub_cont1,lv_color_black(),0);
    lv_obj_set_style_pad_all(sub_cont1,0,0);
    lv_obj_set_style_pad_bottom(sub_cont1,20,0);
    lv_obj_set_style_pad_top(sub_cont1,20,0);
    lv_obj_set_style_border_width(sub_cont1,0,0);
    lv_obj_set_scrollbar_mode(sub_cont1, LV_SCROLLBAR_MODE_OFF);
    //lv_obj_clear_flag(sub_cont1, LV_OBJ_FLAG_SCROLLABLE);
    //lv_obj_align(sub_cont1,LV_ALIGN_TOP_LEFT,10,0);
    lv_obj_set_pos(sub_cont1,30,10);
    lv_obj_set_flex_flow(sub_cont1, LV_FLEX_FLOW_COLUMN);
    lv_obj_set_style_clip_corner(sub_cont1, true, 0);
    lv_obj_t *old_obj;
    for(uint8_t i = 0; i < LIST_MENU_NUM; i++)…
    if(top != NULL)
    {
        lv_obj_update_layout(sub_cont1);
        lv_obj_scroll_by(sub_cont1, -top->x, -top->y, LV_ANIM_OFF);
    }
    else
    {
        lv_obj_scroll_to_view(lv_obj_get_child(sub_cont1, 0), LV_ANIM_OFF);
    }
    lv_obj_add_event_cb(sub_cont1,fr_lv_encoder_obj_event_cb,LV_EVENT_ALL,NULL);
    lv_group_add_obj(prj_group, sub_cont1);
    group_tail_obj = sub_cont1;

}
```

```c
void fr_lv_encoder_obj_event_cb(lv_event_t *e)
{
    lv_obj_t *target = e->target;
    lv_event_code_t code = e->code;
    uint8_t child_cn = lv_obj_get_child_cnt(target);
    if (!lv_obj_is_valid(target))
        return;

    if(code == LV_EVENT_KEY)
    {
        uint32_t *key = lv_event_get_param(e);
        uint32_t input_key = *key;
        // printf("key:%d\r\n",input_key);
        lv_coord_t obj_y;
        #if 0…
        #else
        if(setting_timer_flag == false)
        {
            if(input_key == 130)
            {
                lv_obj_scroll_by(target, 0, ENCODE_OFFSET, LV_ANIM_ON);
                user_click_flag = true;
            }
            else if(input_key == 129)
            {
                lv_obj_scroll_by(target, 0, -ENCODE_OFFSET, LV_ANIM_ON);
                user_click_flag = true;
            }
        }
        #endif

    }
    else if (code == LV_EVENT_DELETE)
```


联系方式

欢迎大家针对富芮坤产品和文档提出建议。

反馈：doc@freqchip.com.

网站：www.freqchip.com

销售：sales@freqchip.com

电话：+86-21-5027-0080