

# STM32F767IGT6 LVGL移植

**注意事项：其他开发板均适用**

## 一、开发环境

1.开发板：正点原子STM32F767IGT6阿波罗开发板

2.软件：KEIL

3.源码：LVGL库文件(V8.2版本)、HAL触摸屏实验、HAL基本定时器中断实验、移植好的FreeRTOS工程

4.源码获取途径：

(1)LVGL库文件(V8.2版本)：LVGL官网，网址：<https://github.com/lvgl/lvgl/tree/release/v8.2>

(2)触摸屏实验、基本定时器中断实验：网址：<http://www.openedv.com/docs/>

(3)移植好的FreeRTOS的工程：<https://gitee.com/yuan-zhenbin/lvgl-code-repository>

5.移植要求：

(1)16、32、64位微控制器或处理器

(2)主控频率：大于16MHz

(3)Flash/ROM：大于64kb，建议180kb以上


(4)RAM：大于8kb，建议24kb以上


(5)图形缓冲区：大于水平分辨率，建议大于十分之一屏幕总像素

(6)编译器：C99及更新


## 二、裸机LVGL移植步骤


### 1.裁剪官方LVGL库文件

 demos

 examples

 src

 lv\_conf.h

 lvgl.h

解压LVGL库文件压缩包，并保留以上文件，其中examples文件夹下至保留porting文件夹及里面的文件，将lv\_conf\_template.h重命名为lv\_conf.h。

### 2.修改编译使能选项


打开lv\_conf.h将#if 0改为#if 1

### 3.建立LVGL工程结构

在触摸屏实验的HARDWARE文件夹下，新建LVGL文件夹，在LVGL文件夹下新建GUI、GUI\_APP文件夹，在GUI文件夹下建立lvgl文件夹(注意：该文件夹为小写)。并将基本定时器中断实验中的TIMER文件夹拷贝至触摸屏实验的HARDWARE文件夹下。至此，LVGL移植工程结构建立完成。

### 4.添加文件

(1)复制文件：将example、src、lv\_conf.h和lvgl.h拷贝至HARDWARE/LVGL/GUI/lvgl下，将demos拷贝至HARDWARE/LVGL/GUI\_APP下

(2)新建分组：打开触摸屏实验工程，点击  新建分组，组名分别为：

HARDWARE/lvgl/example/porting、HARDWARE/lvgl/src/core、HARDWARE/lvgl/src/draw、HARDWARE/lvgl/src/extra、HARDWARE/lvgl/src/font、HARDWARE/lvgl/src/gpu、HARDWARE/lvgl/src/hal、HARDWARE/lvgl/src/misc、HARDWARE/lvgl/src/widgets。

(3)添加源文件：

a.添加第一步裁剪后LVGL库文件中examples/porting文件夹下的lv\_port\_disp\_template.c和lv\_port\_indev\_template.c到HARDWARE/lvgl/example/porting下。

b.添加src/core文件夹下所有的.c文件到HARDWARE/lvgl/src/core下。

c.添加src/draw文件夹下除nxp\_nxp、nxp\_vglite、sdl和stm32\_dma2d文件夹之外的所有.c文件添加至HARDWARE/lvgl/src/draw。

d.添加src/extra文件夹下所有的.c文件到HARDWARE/lvgl/src/extra下。

e.添加src/font文件夹下所有的.c文件到HARDWARE/lvgl/src/font下。

f.添加src/draw/stm32\_dma2d和src/draw/sdl文件夹下所有的.c文件到HARDWARE/lvgl/src/gpu下。

g.添加src/hal文件夹下所有的.c文件到HARDWARE/lvgl/src/hal下。

h.添加src/misc文件夹下所有的.c文件到HARDWARE/lvgl/src/misc下。

i.添加src/widgets文件夹下所有的.c文件到HARDWARE/lvgl/src/widgets下。

j.添加HARDWARE/TIMER文件夹下的timer.c文件到HARDWARE下。

(4)添加头文件路径

头文件路径分别

为：..\HARDWARE\LVGL\GUI、..\HARDWARE\LVGL\GUI\lvgl、..\HARDWARE\LVGL\GUI\lvgl\src、..\HARDWARE\LVGL\GUI\lvgl\examples\porting、..\HARDWARE\TIMER、..\HARDWARE\TIMER

(5)开启C99模式

### 5.配置输出设备

(1)把lv\_port\_disp\_template.c和lv\_port\_disp\_template.h中编译使能#if 0改为#if 1。

(2)在lv\_port\_disp\_template.c中包含输出设备头文件，#include "lcd.h"

(3)在lv\_port\_disp\_template.c中的disp\_init函数中初始化屏幕设备，设置为横屏

```
static void disp_init(void)
{
    /*You code here*/
    //lcd初始化
    LCD_Init();
    //横屏
    LCD_Display_Dir(1);
}
```

#### (4)设置屏幕尺寸并配置图形数据缓冲区

在lv\_port\_disp\_template.c中找到lv\_port\_disp\_init函数中修改，采用单缓冲，其中MY\_DISP\_HOR\_RES为屏幕水平分辨率像素，为了方便可以在lv\_port\_disp\_template.h中进行宏定义，同时还可以宏定义屏幕垂直分辨率像素MY\_DISP\_VER\_RES，根据屏幕进行修改。MY\_DISP\_HOR\_RES \* 150中的150可以根据自己的开发板进行修改，默认为10，越大显示效果越好，占用内存越高。

```
/* Example for 1) */           //单缓冲
static lv_disp_draw_buf_t draw_buf_dsc_1;
static lv_color_t buf_1[MY_DISP_HOR_RES * 150];           /*A
buffer for 10 rows*/
lv_disp_draw_buf_init(&draw_buf_dsc_1, buf_1, NULL, MY_DISP_HOR_RES * 150);
/*Initialize the display buffer*/
```

注意：注释掉其他的缓冲方式

```
/* Example for 2) */           //双缓冲
static lv_disp_draw_buf_t draw_buf_dsc_2;`
static lv_color_t buf_2_1[MY_DISP_HOR_RES * 10];           /*A
buffer for 10 rows*/`
static lv_color_t buf_2_2[MY_DISP_HOR_RES * 10];           /*An
other buffer for 10 rows*/`
lv_disp_draw_buf_init(&draw_buf_dsc_2, buf_2_1, buf_2_2, MY_DISP_HOR_RES * 10);
/*Initialize the display buffer*/`

/* Example for 3) also set disp_drv.full_refresh = 1 below*/           //全缓冲`
static lv_disp_draw_buf_t draw_buf_dsc_3;`
static lv_color_t buf_3_1[MY_DISP_HOR_RES * MY_DISP_VER_RES];           /*A
screen sized buffer*/`
static lv_color_t buf_3_2[MY_DISP_HOR_RES * MY_DISP_VER_RES];
/*Another screen sized buffer*/`
lv_disp_draw_buf_init(&draw_buf_dsc_3, buf_3_1, buf_3_2, MY_DISP_HOR_RES *
LV_DISP_VER_RES_MAX); /*Initialize the display buffer*/`
```

采用动态获取的方式获取屏幕分辨率

```
//屏幕分辨率
disp_drv.hor_res = lcddev.width;
disp_drv.ver_res = lcddev.height;
```

e.在disp\_full中配置打点函数

```
static void disp_flush(lv_disp_drv_t * disp_drv, const lv_area_t * area,
lv_color_t * color_p)
{
    LCD_Color_Fill(area->x1,area->y1,area->x2,area->y2,(uint16_t*)color_p);
    lv_disp_flush_ready(disp_drv);
}
```

## 6.配置输入设备(触摸)

- (1)把lv\_port\_indev\_template.c和lv\_port\_indev\_template.h中编译使能#if 0改为#if 1。
- (2)在lv\_port\_indev\_template.c中包含输出设备头文件，#include "touch.h"
- (3)裁剪输入设备，只保留touchpad相关的，其他的全部删除。
- (4)在touchpad\_init函数中初始化触摸屏

```
static void touchpad_init(void)
{
    /*Your code comes here*/
    tp_dev.init();           //触摸屏初始化
}
```

- (5)配置触摸检测函数

```
static bool touchpad_is_pressed(void)
{
    tp_dev.scan(0);
    if(tp_dev.sta & TP_PRES_DOWN)
        return true;
    return false;
}
```

- (6)配置坐标获取函数

```
static void touchpad_get_xy(lv_coord_t * x, lv_coord_t * y)
{
    /*Your code comes here*/
    (*x) = tp_dev.x[0];
    (*y) = tp_dev.y[0];
}
```

## 7.提供时基

- (1)添加定时器驱动，由于在移植LVGL的时候已经添加了相关源文件和头文件，这里省略。
- (2)在定时器驱动timer.c中包含：#include "lvgl.h"
- (3)在定时器中断回调函数中调用：lv\_tick\_inc(1);
- (4)初始化定时器，时间为1ms，具体计算可参考正点原子指南。

## 8.移植main函数

(1)包含头文件: #include "timer.h"、#include "lvgl.h"、#include "lv\_port\_indev\_template.h"、lv\_port\_disp\_template.h

(2)初始化定时器、LVGL、输入输出设备: TIM3\_Init(10-1,10800-1)、lv\_init()、lv\_port\_disp\_init()、lv\_port\_indev\_init(), 注意: 定时器初始化参数根据自己开发板定时器时钟频率设置, 只要时间周期为1ms即可。

(3)在while(1)中调用lv\_timer\_handle();

(4)编写测试代码: lv\_obj\_t\* switch\_obj = lv\_switch\_create(lv\_scr\_act());lv\_obj\_set\_size(switch\_obj,120,60);lv\_obj\_align(switch\_obj,LV\_ALIGN\_CENTER,0,0);

(5)编译测试: 这里可以看到延迟很大, 修改lv\_conf.h中#define LV\_DISP\_DEF\_REFR\_PERIOD 4 和 #define LV\_INDEV\_DEF\_READ\_PERIOD 4 即可, 一般设置为5ms。

(6)官方demo测试

压力测试:

a.添加头文

件: ..\HARDWARE\LVGL\GUI\_APP\demos\stress、..\HARDWARE\LVGL\GUI\_APP\demos

b.在lv\_conf.h中找到#define LV\_USE\_DEMO\_STRESS, 将宏定义0改为1

c.新建项目组: 添加分组HARDWARE/LVGL/GUI\_APP, 添加demos/stress文件夹下下lv\_demo\_stress.c文件

d.包含头文件: 在main中包含: #include "lv\_demo\_stress.h"

e.初始化demo: lv\_demo\_stress();

音乐播放器移植:

a.添加头文

件: ..\HARDWARE\LVGL\GUI\_APP\demos\music、..\HARDWARE\LVGL\GUI\_APP\demos

b.在lv\_conf.h中找到#define LV\_USE\_DEMO\_MUSIC, 将宏定义0改为1

c.新建项目组: 添加分组HARDWARE/LVGL/GUI\_APP, 添加demos/music文件夹下下所有.c文件

d.包含头文件: 在main中包含: #include "lv\_demo\_stress.h"

e.初始化demo: lv\_demo\_music();

f.如果编译错误, 在lv\_conf.h中找到对应字体, 将宏定义0改为1

## 三、带FreeRTOS的LVGL移植

注意: 此实验是基于裸机LVGL移植

### 1.复制文件

(1)将准备的已经移植好FreeRTOS的工程中FreeRTOS拷贝至HARDWARE文件夹下

(2)将FreeRTOSConfig.h文件拷贝至User文件夹下

(3)将准备的已经移植好FreeRTOS的工程中SYSTEM拷贝至SYSTEM文件夹下, 进行替换

## 2.新建分组

HARDWARE/FreeRTOS\_CORE、HARDWARE/FreeRTOS\_PORT

## 3.添加文件

(1)HARDWARE/FreeRTOS\_CORE: croutine.c、event\_groups.c、list.c、queue.c、stream\_buffer.c、tasks.c、timers.c

(2)HARDWARE/FreeRTOS\_PORT: heap\_a.c、port.c

port.c根据自己开发板内核进行选择

## 4.添加头文件

..\HARDWARE\FreeRTOS\include、..\HARDWARE\FreeRTOS\portable\RVDS\ARM\_CM7\r0p1

## 5.屏蔽SysTick、SVC、PendSV中断

STM32F1----->stm32f1xx\_it.c

STM32F4----->stm32f4xx\_it.c

STM32F7----->stm32f7xx\_it.c

STM32H7----->stm32h7xx\_it.c

```
void SVC_Handler(void)
{
}

void PendSV_Handler(void)
{
}

void SysTick_Handler(void)
{
    HAL_IncTick();
}
```

## 6.修改宏定义

将\_\_NVIC\_PRIO\_BITS, 由4U改为4

STM32F1----->stm32f103\_xe.h

STM32F4----->stm32f407xx.h或者stm32f429xx.h

STM32F7----->stm32f750xx.h或者stm32f767xx.h

STM32H7----->stm32h750xx.c或者stm32h743xx.h

```
#define __NVIC_PRIO_BITS 4
```

## 7.修改时基

在lv\_conf.h中找到LV\_TICK\_CUSTOM，将它的宏由0改为1，并对应修改一下内容

```
/* 使用自定义tick源，以毫秒为单位告诉运行时间。它不需要手动更新 `lv_tick_inc()` */
#define LV_TICK_CUSTOM 1
#if LV_TICK_CUSTOM
    #define LV_TICK_CUSTOM_INCLUDE "FreeRTOS.h"
    /* 系统时间函数头 */
    #define LV_TICK_CUSTOM_SYS_TIME_EXPR (xTaskGetTickCount())
    /* 计算系统当前时间的表达式(以毫秒为单位) */
#endif /*LV_TICK_CUSTOM*/
```

## 8.新建lvgl\_demo.c、lvgl\_demo.h

```
//lvgl_demo.c
#include "lvgl_demo.h"
#include "../HARDWARE/LED/led.h"
#include "FreeRTOS.h"
#include "task.h"

#include "lvgl.h"
#include "lv_port_disp_template.h"
#include "lv_port_indev_template.h"
#include "lv_demo_stress.h"
#include "lv_demo_music.h"
/*FreeRTOS配置*/

/* START_TASK 任务 配置

* 包括：任务句柄 任务优先级 堆栈大小 创建任务
*/
#define START_TASK_PRIO 1 /* 任务优先级 */
#define START_STK_SIZE 128 /* 任务堆栈大小 */
TaskHandle_t StartTask_Handler; /* 任务句柄 */
void start_task(void *pvParameters); /* 任务函数 */

/* LV_DEMO_TASK 任务 配置

* 包括：任务句柄 任务优先级 堆栈大小 创建任务
*/
#define LV_DEMO_TASK_PRIO 3 /* 任务优先级 */
#define LV_DEMO_STK_SIZE 1024 /* 任务堆栈大小 */
TaskHandle_t LV_DEMOTask_Handler; /* 任务句柄 */
void lv_demo_task(void *pvParameters); /* 任务函数 */

/* LED_TASK 任务 配置

* 包括：任务句柄 任务优先级 堆栈大小 创建任务
*/
#define LED_TASK_PRIO 4 /* 任务优先级 */
#define LED_STK_SIZE 128 /* 任务堆栈大小 */
TaskHandle_t LEDTask_Handler; /* 任务句柄 */
void led_task(void *pvParameters); /* 任务函数 */
```

```

/*****
*****

void lvgl_demo(void)
{
    lv_init();                                /* lvgl系统初始化 */
    lv_port_disp_init();                      /* lvgl显示接口初始化,放在
lv_init()的后面 */
    lv_port_indev_init();                    /* lvgl输入接口初始化,放在
lv_init()的后面 */
    xTaskCreate((TaskFunction_t )start_task,    /* 任务函数 */
                (const char* )"start_task",    /* 任务名称 */
                (uint16_t )START_STK_SIZE,      /* 任务堆栈大小 */
                (void* )NULL,                  /* 传递给任务函数的参数 */
                (UBaseType_t )START_TASK_PRIO,  /* 任务优先级 */
                (TaskHandle_t* )&StartTask_Handler); /* 任务句柄 */

    vTaskStartScheduler();                    /* 开启任务调度 */
}

void start_task(void *pvParameters)
{
    taskENTER_CRITICAL();                    /* 进入临界区 */

    /* 创建LVGL任务 */
    xTaskCreate((TaskFunction_t )lv_demo_task,
                (const char* )"lv_demo_task",
                (uint16_t )LV_DEMO_STK_SIZE,
                (void* )NULL,
                (UBaseType_t )LV_DEMO_TASK_PRIO,
                (TaskHandle_t* )&LV_DEMO_Task_Handler);

    /* LED测试任务 */
    xTaskCreate((TaskFunction_t )led_task,
                (const char* )"led_task",
                (uint16_t )LED_STK_SIZE,
                (void* )NULL,
                (UBaseType_t )LED_TASK_PRIO,
                (TaskHandle_t* )&LED_Task_Handler);

    taskEXIT_CRITICAL();                    /* 退出临界区 */
    vTaskDelete(StartTask_Handler); /* 删除开始任务 */
}

void lv_demo_task(void *pvParameters)
{
    lv_demo_music();                        /* 测试的demo */

    while(1)
    {
        lv_timer_handler(); /* LVGL计时器 */
        vTaskDelay(5);
    }
}

void led_task(void *pvParameters)

```



```
{  
    while(1)  
    {  
        LED0_TOGGLE();  
        vTaskDelay(1000);  
    }  
}  
  
//lvgl_demo.h  
  
#ifndef __LVGL_DEMO_H  
#define __LVGL_DEMO_H  
  
void lvgl_demo(void); /* lvgl例程 */  
  
#endif
```

在main.c中包含头文件lvgl\_demo.h，并调用lvgl\_demo();编译运行即可。