尚硅谷嵌入式技术之 LVGL

（作者：尚硅谷研究院）

版本：V1.0.0

第 1 章 LVGL 概述

1.1 什么是 LVGL

官网: https://lvgl.io/

LVGL（Light and Versatile Graphics Library：轻量和通用图形库），是一款免费且开源的图形库，提供创建嵌入式 GUI 所需的一切，包括易于使用的图形元素、美丽的视觉效果和低内存占用。

1.2 LVGL 的特点

（1）**丰富且强大的模块化图形组件**：按钮（buttons）、图表（charts）、列表（lists）、滑动条（sliders）、图片（images）等。

（2）**高级的图形引擎**：动画、抗锯齿、透明度、平滑滚动、图层混合等效果。

（3）**支持多种输入设备**：**触摸屏**、键盘、编码器、按键等。

（4）支持多显示设备。

（5）不依赖特定的硬件平台，可以在任何显示屏上运行。

（6）配置**可裁剪**（最低资源占用：64 kB Flash，16 kB RAM）。

（7）基于 UTF-8 的多语种支持，例如中文、日文、韩文、阿拉伯文等。

（8）可以通过类 CSS 的方式来设计、布局图形界面（例如：Flexbox、Grid）。

（9）支持操作系统、外置内存、以及硬件加速（LVGL 已内建支持 STM32 DMA2D、NXP PXP 和 VGLite）。

（10）即便仅有单缓冲区（frame buffer）的情况下，也可保证渲染如丝般顺滑。

（11）全部由 C 编写完成，并支持 C++调用。

（12）支持 Micropython 编程，参见：LVGL API in Micropython。

（13）支持模拟器仿真，可以无硬件依托进行开发。

（14）丰富详实的例程。

（15）在 MIT 许可下免费和开源。

1.3 LVGL 对硬件的要求

基本上，每个能够驱动显示器的现代控制器都适合运行 LVGL。 最低要求是：

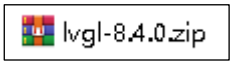
名字	最低	推荐
架构	16、32 或 64 位微控制器或处理器（arm 或 x86）	
时钟	>16MHz	>48MHz
Flash(ROM)	>64KB	>180KB
SRAM	>16KB	>48KB
显示缓冲区	>1 * 1 * 水平分辨率	> 10 * 10 * 水平分辨率
帧缓冲区	MCU 或外部显示控制器中的一个帧缓冲区	
编译器	C99 或更新的编译器（keil5 务必开启 c99 支持）	
需要的知识	C 或 c++	

第 2 章 LVGL 移植

2.1 下载 LVGL 源码

我们选择 8.4.0

下载地址: https://github.com/lvgl/lvgl/releases



2.2 构建项目

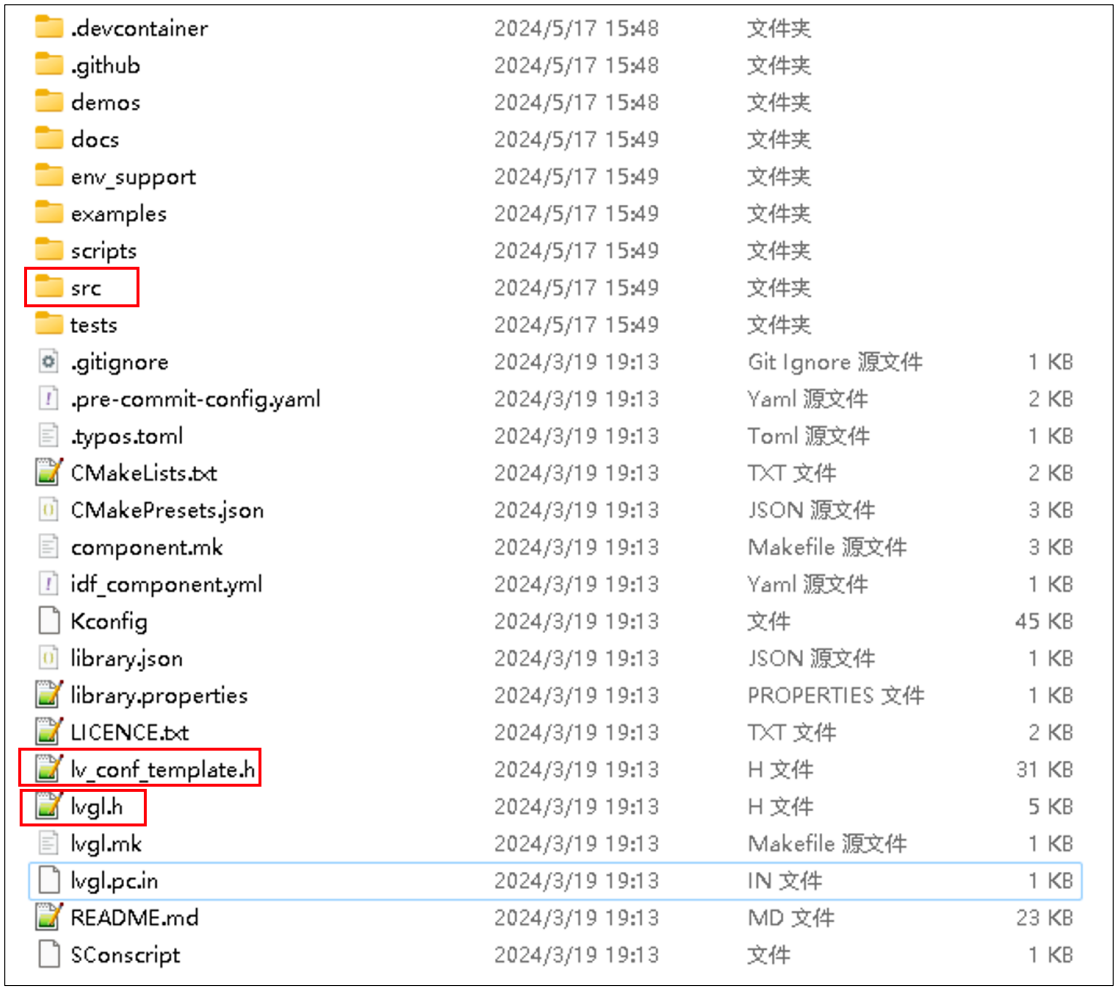
2.2.1 准备一个工程

准备一个已经配置好你的屏幕的工程。比如我们的 LCD 是使用 FSMC 来驱动的，所以需要配置好 FSMC，让 LCD 可以正常工作!

如果需要触摸输入的还需要配置好触摸屏驱动!

2.2.2 copy 源码

在工程的中间层层创建一个目录：lvgl，然后 copy 以下文件到 lvgl 中（注意不要更改 lvgl 源码的目录结构）。



.devcontainer	2024/5/17 15:48	文件夹	
.github	2024/5/17 15:48	文件夹	
demos	2024/5/17 15:48	文件夹	
docs	2024/5/17 15:49	文件夹	
env_support	2024/5/17 15:49	文件夹	
examples	2024/5/17 15:49	文件夹	
scripts	2024/5/17 15:49	文件夹	
src	2024/5/17 15:49	文件夹	
tests	2024/5/17 15:49	文件夹	
.gitignore	2024/3/19 19:13	Git Ignore 源文件	1 KB
.pre-commit-config.yaml	2024/3/19 19:13	Yaml 源文件	2 KB
.typos.toml	2024/3/19 19:13	Toml 源文件	1 KB
CMakeLists.txt	2024/3/19 19:13	TXT 文件	2 KB
CMakePresets.json	2024/3/19 19:13	JSON 源文件	3 KB
component.mk	2024/3/19 19:13	Makefile 源文件	3 KB
idf_component.yml	2024/3/19 19:13	Yaml 源文件	1 KB
Kconfig	2024/3/19 19:13	文件	45 KB
library.json	2024/3/19 19:13	JSON 源文件	1 KB
library.properties	2024/3/19 19:13	PROPERTIES 文件	1 KB
LICENCE.txt	2024/3/19 19:13	TXT 文件	2 KB
lv_conf_template.h	2024/3/19 19:13	H 文件	31 KB
lvgl.h	2024/3/19 19:13	H 文件	5 KB
lvgl.mk	2024/3/19 19:13	Makefile 源文件	1 KB
lvgl.pc.in	2024/3/19 19:13	IN 文件	1 KB
README.md	2024/3/19 19:13	MD 文件	23 KB
SConscript	2024/3/19 19:13	文件	1 KB

还有 examples 中的 porting 文件夹，也 copy 到 lvgl 目录下。

此电脑 › Desktop › lvgl › 02_资料 › 01_lvgl源码 › lvgl-8.4.0 › examples

名称	修改日期	类型	大小
anim	2024/5/19 16:34	文件夹	
arduino	2024/5/19 16:34	文件夹	
assets	2024/5/19 16:34	文件夹	
event	2024/5/19 16:34	文件夹	
get_started	2024/5/19 16:34	文件夹	
layouts	2024/5/19 16:34	文件夹	
libs	2024/5/19 16:34	文件夹	
others	2024/5/19 16:34	文件夹	
porting	2024/5/19 16:34	文件夹	
scroll	2024/5/19 16:34	文件夹	
styles	2024/5/19 16:34	文件夹	
widgets	2024/5/19 16:34	文件夹	
header.py	2024/3/19 12:51	Python 源文件	1 KB
lv_examples.h	2024/3/19 12:51	H 文件	1 KB
lv_examples.mk	2024/3/19 12:51	Makefile 源文件	1 KB
test_ex.sh	2024/3/19 12:51	SH 文件	1 KB

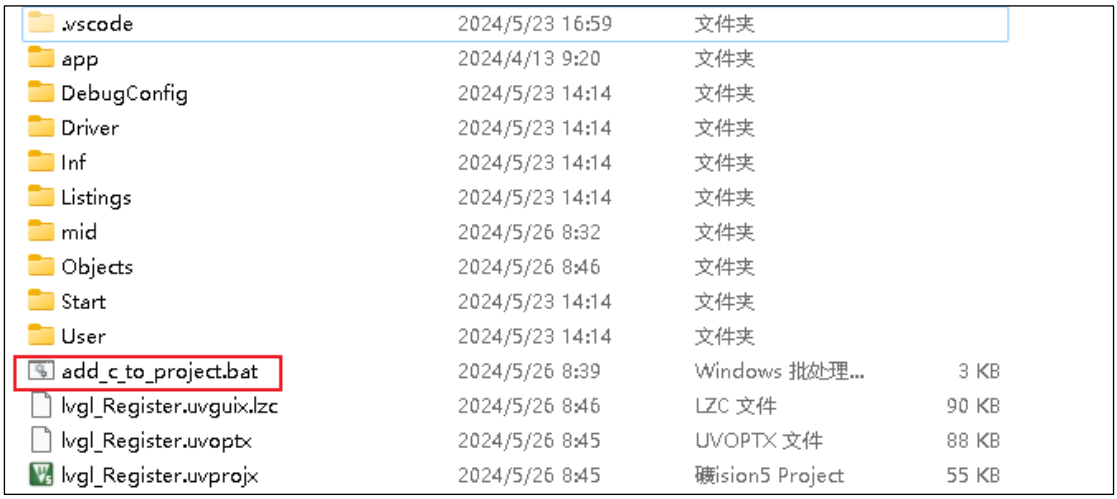
把文件 lv_conf_templete.h 改为 lv_conf.h。

2.2.3 在 keil 中修改项目结构

需要添加所有 c 文件。由于 c 文件太多，逐个添加需要耗费比较多的时间，可以使用下面的脚本添加。

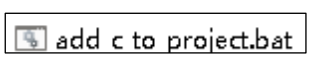
add_c_to_project.bat

1）把脚本放入到项目的根目录

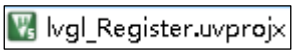


2）使用脚本添加时,目录必须满足下面的结构，名字也不要错

3）双击脚本



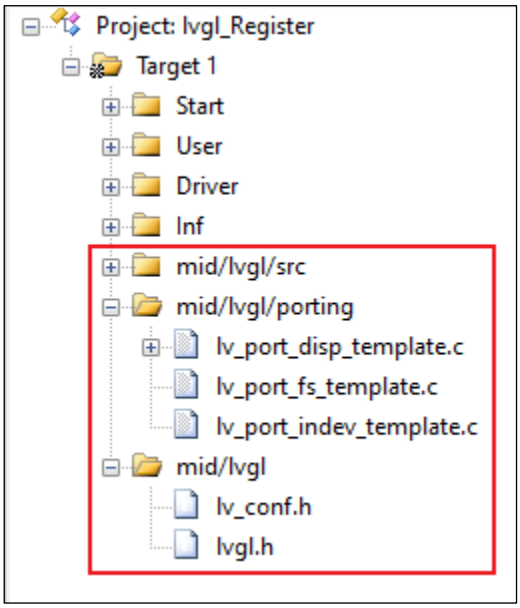
4）使用文本编辑工具打开 lvgl_Register.uvprojx，把首行第一个空格去掉



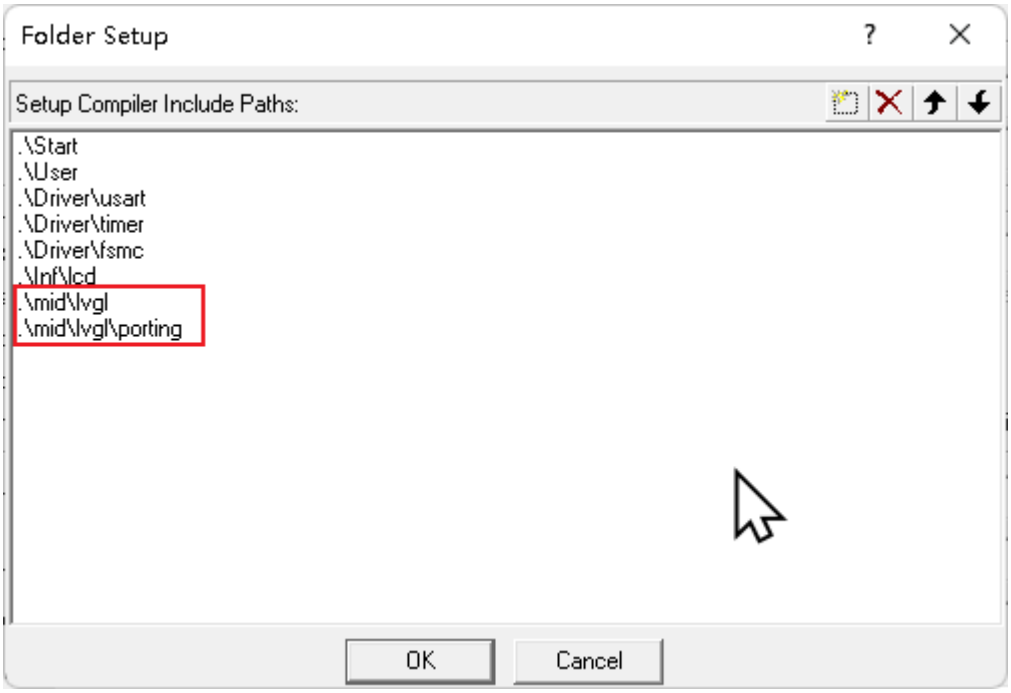

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<Project xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

去掉首行的第一个空格

5）使用 keil 打开项目，最终项目结构如图所示

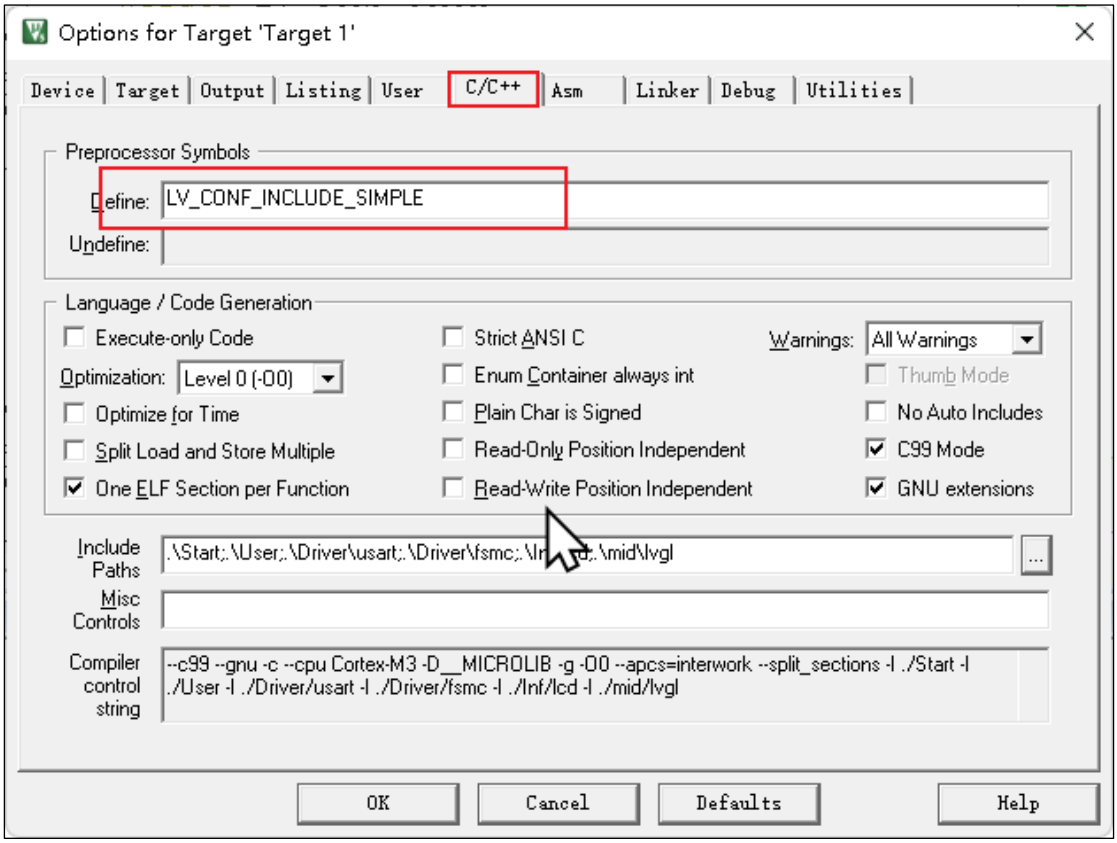


2.2.4 在 keil 中增加 include path

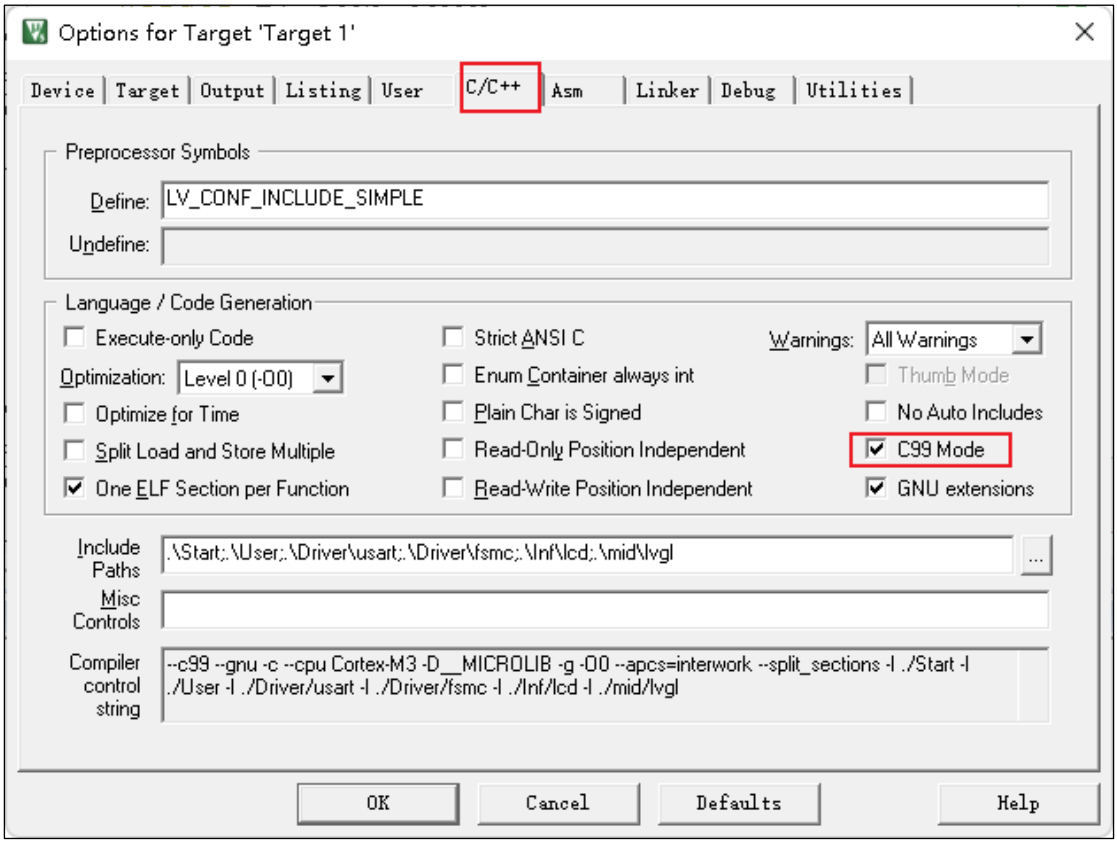


2.2.5 在 keil 中添加一个宏定义

LV_CONF_INCLUDE_SIMPLE



2.2.6 工程务必支持 C99



2.3 修改源码

2.3.1 打开配置开关

打开 lv_conf.h，把 #if 0 改成 #if 1。

```
/* clang-format off */
#if 1 /*Set it to "1" to enable content*/
```

2.3.2 给 lvgl 配置底层驱动

2.3.2.1 LCD 显示驱动

（1）打开 lv_port_disp_template.h 和 lv_port_disp_template.c，分别把文件开头部分的 #if 0 改为 #if 1

```
#if 1 /*Set it to "1" to enable content*/
```

（2）导入 LCD 驱动

打开 lv_port_disp_template.c

```
#include "Inf_LCD.h"
```

（3）设置屏幕的宽和高，并把中间的 #warning 注释掉

```
16  /*********************
17   *      DEFINES
18   *********************/
19  #ifndef MY_DISP_HOR_RES
20      //#warning Please define or replace the macro MY_DISP_HOR_RES with the actual scre
21      #define MY_DISP_HOR_RES    320   屏幕宽度
22  #endif
23
24  #ifndef MY_DISP_VER_RES
25      //#warning Please define or replace the macro MY_DISP_HOR_RES with the actual scre
26      #define MY_DISP_VER_RES    480   屏幕高度
27  #endif
```

（4）注释掉显示缓冲 2 和缓冲 3，只保留显示缓冲 1

```
/* Example for 1) */
static lv_disp_draw_buf_t draw_buf_dsc_1;
static lv_color_t         buf_1[MY_DISP_HOR_RES * 10];                      /*
lv_disp_draw_buf_init(&draw_buf_dsc_1, buf_1, NULL, MY_DISP_HOR_RES * 10); /*

/* Example for 2) */
// static lv_disp_draw_buf_t draw_buf_dsc_2;
// static lv_color_t         buf_2_1[MY_DISP_HOR_RES * 10];
// static lv_color_t         buf_2_2[MY_DISP_HOR_RES * 10];
// lv_disp_draw_buf_init(&draw_buf_dsc_2, buf_2_1, buf_2_2, MY_DISP_HOR_RES *

/* Example for 3) also set disp_drv.full_refresh = 1 below*/
// 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];
//  static lv_color_t buf_3_2[MY_DISP_HOR_RES * MY_DISP_VER_RES];
// lv_disp_draw_buf_init(&draw_buf_dsc_3, buf_3_1, buf_3_2,
//                       MY_DISP_VER_RES * LV_VER_RES_MAX);   /*Initialize the
```

全部注释

（5）在函数 disp_init 中调用 LCD 的初始化

```
/*Initialize your display and the required peripherals.*/
static void disp_init(void)
{
    /*You code here*/
    Inf_LCD_Init();
}
```

（6）实现显示刷新函数

```
static void disp_flush(lv_disp_drv_t *disp_drv, const lv_area_t *area, lv_color_t
*color_p)
{
    if(disp_flush_enabled)
    {
        /*The most simple case (but also the slowest) to put all pixels to the
screen one-by-one*/
        Inf_LCD_FillColor(area->x1,
                          area->y1,
                          area->x2 - area->x1 + 1,
                          area->y2 - area->y1 + 1,
                          (uint16_t *)color_p);
    }
    /*IMPORTANT!!!
```

```
    *Inform the graphics library that you are ready with the flushing*/
    lv_disp_flush_ready(disp_drv);
}
```

2.3.2.2 触摸屏驱动

打开 lv_port_indev_template.c 和 lv_port_indev_template.c 文件。

（1）打开开关

```
#if 1
```

（2）修改 lvgh.h 的 include 写法

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

（3）修改 lv_port_indev_init 函数

这个函数是用来初始化输入设备的，我们的输入设备只有触摸屏，只保留触摸屏的初始化代码，其他的全部删除。

```
void lv_port_indev_init(void)
{
    /**
     * Here you will find example implementation of input devices supported by
LittelvGL:
     *  - Touchpad
     *  - Mouse (with cursor support)
     *  - Keypad (supports GUI usage only with key)
     *  - Encoder (supports GUI usage only with: left, right, push)
     *  - Button (external buttons to press points on the screen)
     *
     *  The `..._read()` function are only examples.
     *  You should shape them according to your hardware
     */

    static lv_indev_drv_t indev_drv;

    /*------------------
     * Touchpad
     * -----------------*/
```

```
    /*Initialize your touchpad if you have*/
    touchpad_init();
    /*Register a touchpad input device*/
    lv_indev_drv_init(&indev_drv);
    indev_drv.type = LV_INDEV_TYPE_POINTER;
    indev_drv.read_cb = touchpad_read;
    indev_touchpad = lv_indev_drv_register(&indev_drv);
}
```

（4）实现 touchpad_init 函数

我们的输入设备是触摸屏，这个函数就是具体的触摸屏的初始化实现函数。

```
static void touchpad_init(void)
{
    /*Your code comes here*/
    Inf_FT5x16_Init();
}
```

（5）实现 touchpad_is_pressed 函数

当触摸屏被按下时，这个函数应该返回 true，表示发生了触摸事件，否则返回 false。

```
/*Return true is the touchpad is pressed*/
static bool touchpad_is_pressed(void)
{
    /*Your code comes here*/
    if(Inf_FT5x16_IsPressed())
    {
        return true;
    }
    return false;
}
```

（6）实现 touchpad_get_xy 函数

如果发生了触摸事件，此函数返回 x 和 y 坐标。

```
/*Get the x and y coordinates if the touchpad is pressed*/
static void touchpad_get_xy(lv_coord_t *x, lv_coord_t *y)
{
    /*Your code comes here*/
```

```
    Inf_FT5x16_ReadXY((uint16_t *)x, (uint16_t *)y);
    /* (*x) = 0;
    (*y) = 0; */
}
```

2.3.3 适配“心跳”

LVGL 的任务都是基于时间的，包含绘制，响应触摸等等；所以呢，我们需要给它一个时间基准。

LVGL 提供了一个 lv_tick_inc 的函数，我们需要在系统中去周期性调用他，告诉 LVGL 的时间基准；可以用普通 Timer，也可以使用 SystemTick；

这里我们使用是基本定时器 6，每 1ms（时间基准）产生一次更新中断。在定时器 6 的 ISR 中添加下面的代码。

```
extern void lv_tick_inc(uint32_t tick_period);
void TIM6_IRQHandler(void)
{
    /* 1. 先清除中断标志位 */
    TIM6->SR &= ~TIM_SR_UIF;

    /* 如果每 1ms 调用一次,就传入 1;  如果每 2ms 调用一次就传入 2 ... */
    lv_tick_inc(1);
    /* 必须: 周期性的调用, 用于处理与 lvgl 相关的任务.ms 级别的调用即可 */
    lv_timer_handler();
}
```

2.3.4 修改内存

（1）在启动文件 startup_stm32f10x_hd.s 中修改堆栈大小：栈建议 8k，堆建议 2k

```
Stack_Size      EQU     0x00002000
Heap_Size       EQU     0x00000800
```

（2）在 lv_conf.h 中修改动态堆的申请大小为 16k（>=2k）

```
/*Size of the memory available for `lv_mem_alloc()` in bytes (>= 2kB)*/
#define LV_MEM_SIZE (16U * 1024U)          /*[bytes]*/
```

第 3 章 快速验证移植是否成功

3.1 main.c

```
#include "Common_Debug.h"
#include "Delay.h"
#include "App_Display.h"

int main()
{
    debug_start();
    debug_printfln("\r\n"
                   "lvgl 项目模板: \r\n"
                   "    1. 该模板支持尚硅谷 STM32 开发板\r\n"
                   "    2. 该模板已经集成了 LCD 驱动(Inf_LCD.c)\r\n"
                   "    3. 该模板已经集成了触摸屏驱动(Inf_FT5x16.c)");
    App_Display_Init();

    /* 实例 1: 创建一个按钮 */
    App_Display_CreateButton();

    while(1)
    {
    }
}
```

3.2 App_Display.h

```
#ifndef __APP_DISPLAY_H
#define __APP_DISPLAY_H

#include "Common_Debug.h"
#include "Driver_TIM6.h"
#include "lv_port_disp_template.h"
#include "lv_port_indev_template.h"
void App_Display_Init(void);
void App_Display_CreateButton(void);
#endif
```

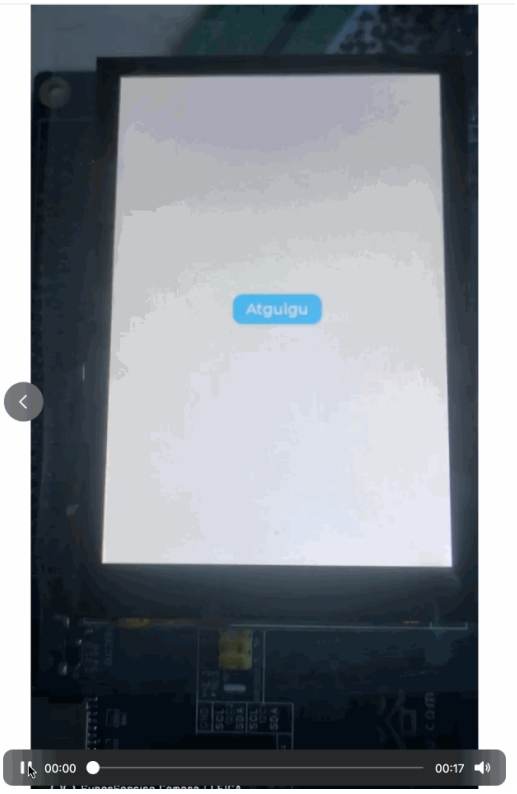
3.3 App_Display.c

注意:lvgl 支持的编码是 utf-8,为了后面能显示中文,把这个文件编码修改为 **utf-8** 编码

```
#include "App_Display.h"


void App_Display_Init(void)
{
    /* 0. 启动定时器,用于给 lvgl 提供时基础 */
    Driver_TIM6_Init();
    Driver_TIM6_Start();

    /* 1. 初始化 lvgl 库 */
    lv_init();
    /* 2. 初始化显示设备 */
    lv_port_disp_init();
    /* 3. 初始化输入设备(触摸设备) */
    lv_port_indev_init();
}


void btn_cb(lv_event_t *e)
{
    static uint8_t cnt = 0;
    lv_obj_t *obj   = lv_event_get_current_target(e);
    lv_obj_t *label = lv_obj_get_child(obj, 0);
    lv_label_set_text_fmt(label, "Atguigu %d", cnt++);
}


/* 创建一个按钮 */
void App_Display_CreateButton(void)
{
    lv_obj_t *btn = lv_btn_create(lv_scr_act());
    lv_obj_align(btn, LV_ALIGN_CENTER, 0, 0);
    lv_obj_set_height(btn, 30);
    lv_obj_add_event_cb(btn, btn_cb, LV_EVENT_PRESSED, NULL);
```

```
    lv_obj_t *label;
    label = lv_label_create(btn);
    lv_obj_align(label, LV_ALIGN_CENTER, 0, 0);
    lv_label_set_text(label, "Atguigu");

    static lv_style_t style_btn;
    lv_style_init(&style_btn);
    lv_style_set_radius(&style_btn, 10);
    lv_style_set_border_color(&style_btn, lv_color_black());
    lv_style_set_border_opa(&style_btn, LV_OPA_30);
    lv_obj_add_style(btn, &style_btn, LV_STATE_DEFAULT);
}
```

3.4 测试效果



第 4 章 LVGL 基本使用

4.1 一些基本概念

4.1.1 Objects（对象）

4.1.1.1 对象的概念

在 LVGL 中，用户界面的 基本组成部分 是对象，也称为 Widgets。例如，一个 按钮、标签、图像、列表、图表 或者 文本区域都可称为一个对象。

所有的对象都使用 lv_obj_t 指针作为句柄进行引用。之后可以使用该指针来设置或获取对象的属性。

4.1.1.2 对象的 Attributes（属性）

1）Basic attributes（基本属性）

所有的对象类型都有一些通用的基本属性：位置，大小，父级，样式，事件处理程序等等。

可以使用 lv_obj_set_... 和 lv_obj_get_... 函数设置或者获取这些属性。

```
/*设置基本对象属性*/
lv_obj_set_size(btn1, 100, 50);   /*设置按钮的大小*/
lv_obj_set_pos(btn1, 20,30);      /*设置按钮的位置*/
```

2）Specific attributes（特殊属性）

对象类型也有特殊的属性。例如，滑块有：最小值和最大值，当前值。

针对这些特殊属性，每个对象类型可能有独特的 API 函数。例如，对于滑块。

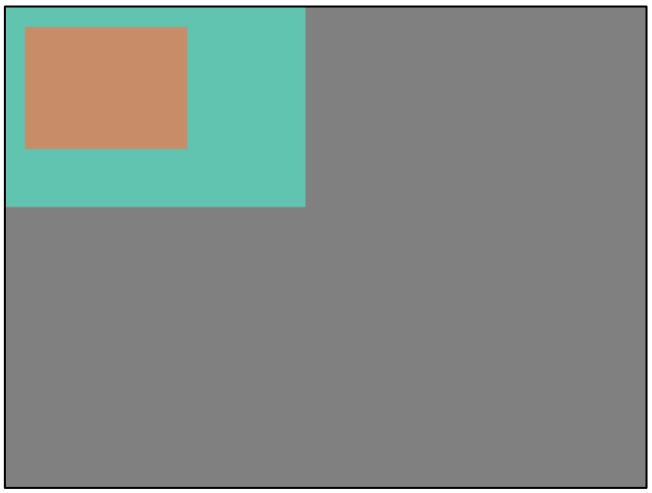
```
/*Set slider specific attributes*/
lv_slider_set_range(slider1, 0, 100);   /*Set the min. and max. values*/
lv_slider_set_value(slider1, 40, LV_ANIM_ON); /*Set the current value (position)*/
```

4.1.1.3 对象的 Parent-child structure（父子结构）

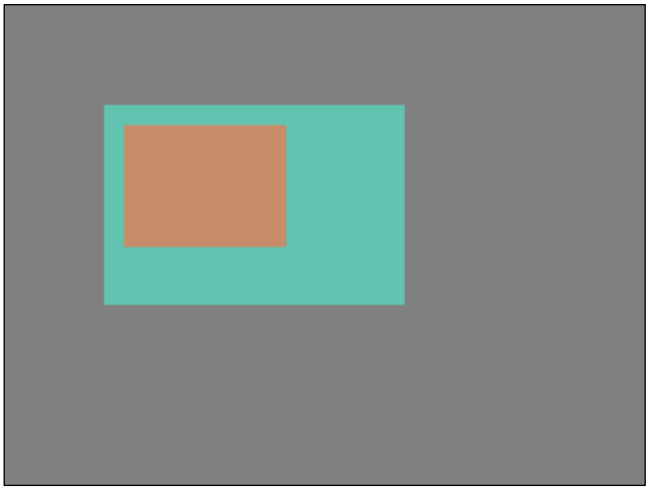
一个父对象可以被视为其子对象的容器。每个对象都必须会有且仅有一个父对象（屏幕除外），但一个父对象可以有任意数量的子对象。父对象的类型没有限制，但是有些对象一般是父对象（例如按钮）或者是子对象（例如标签）。

4.1.1.4 父子 Moving together（一起移动）

如果父对象的位置改变，子对象也会随之移动。因此，所有子对象的位置都是相对于父对象而言的。



移动父对象后。你会发现在子对象会随着父对象进行移动。

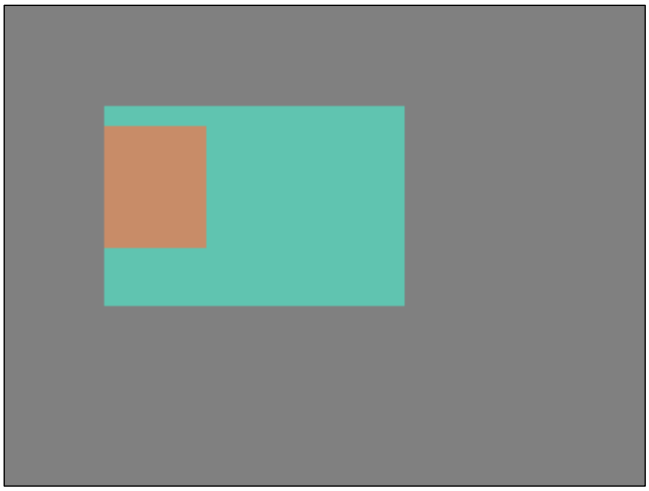


4.1.1.5 Visibility only on the parent（仅在父对象上可见）

如果一个子对象部分或完全超出父对象，则超出部分将不可见。

比如将子对象移出父对象一点点。

```
lv_obj_set_x(obj1, -30);    /*将子对象移出父对象一点点*/
```



4.1.1.6 对象的 Create and delete （创建和删除）

在 LVGL 中，可以在运行时动态创建或删除对象。这也就是说，直到当对象被创建之后才会消耗内存资源。

因此，可以在点击按钮准备打开新界面（屏幕）时再创建新界面（屏幕），并在加载新界面（屏幕）时删除旧界面（屏幕）。

每个控件都有自己的 create 函数，函数原型如下：

```
lv_obj_t * lv_<widget>_create(lv_obj_t * parent, <如果有其他参数>);
```

通常，创建函数只有一个 parent 参数，指示在哪个对象上创建该控件。返回值是指向创建出来的控件的指针，类型为 `lv_obj_t *`。

有一个通用的 delete 函数适用于所有对象类型。它删除对象及其所有子对象。

```
void lv_obj_delete(lv_obj_t * obj);
```

lv_obj_del() 会立即删除对象。如果出于任何原因无法立即删除对象，可以使用 lv_obj_delete_async(obj) ，它会在下一次调用 lv_timer_handler() 时执行删除操作。 这在子对象的 LV_EVENT_DELETE 处理程序中删除父对象时很有用（现在不能马上删除父对象，下一次运行 lv_timer_handler 时再删除）。

可以使用 lv_obj_clean(obj) 删除对象的所有子对象（但不包括对象本身）。

可以使用 lv_obj_delete_delayed(obj, 1000) 在经过一定时间后再删除对象，以毫秒为单位。

4.1.1.7 部件（Parts）

一个控件（Widget）由多个部件（Part）组成。例如，一个 Base object 使用主要部分和滚动条部分，而一个 Slider 使用主要部分、指示器部分和旋钮部分。部件类似于 CSS 中的 伪元素。

在 lvgl 中由如下内置部件：

（1）LV_PART_MAIN：类似矩形的背景。

（2）LV_PART_SCROLLBAR：滚动条（一个或多个）。

（3）LV_PART_INDICATOR：指示器，例如滑块、条形图、开关或复选框的勾选框。

（4）LV_PART_KNOB：类似于把手（旋钮），用于调整值。

（5）LV_PART_SELECTED：指示当前选定的选项或部分。

（6）LV_PART_ITEMS：如果部件有多个类似的元素（例如表格单元格）则可用。

（7）LV_PART_CURSOR：标记特定位置，例如文本区域或图表的光标。

（8）LV_PART_CUSTOM_FIRST：可以从这里添加自定义部分。

4.1.2 Screens（屏幕）

屏幕是一种特殊的对象，它们没有父对象。

4.1.2.1 Create screens（创建屏幕）

```
lv_obj_t * scr1 = lv_obj_create(NULL);
```

屏幕可以使用任何对象类型创建。例如，可以使用 Base object 或者图像控件来创建壁纸。

4.1.2.2 Get the active screen（获取活动屏幕）

每个显示器上都会存在一个活动屏幕。默认情况下，库会为每个显示器创建和加载一个名为“Base object”的屏幕。

```
lv_scr_act()   // 获取活动屏幕
```

4.1.2.3 Load screens（加载屏幕）

使用 lv_scr_load() 加载屏幕

4.1.2.4 Layers（图层）

使用 lv_scr_load()加载屏幕会自动生成 2 个图层：顶层（top layer），系统层（system layer）

它们与屏幕独立，将显示在每个屏幕上。顶层位于屏幕上每个对象之上，系统层位于顶层之上。您可以自由地向顶层 添加任何弹出窗口。但是，系统层受到系统级别的限制。

层级：活动屏幕（screen_active） < 顶层（top layer） < 系统层（system layer）

lv_layer_top()和 lv_layer_sys()函数返回指向顶层和系统层的指针。

4.2 Widgets（控件）

4.2.1 Base object（基础对象）

基础对象可以直接用作一个简单的控件：它只不过是一个矩形。在 HTML 术语中，将其视为 。

```
/**
 * @description: 创建一个基础对象
 */
void App_Display_CreateBasicObj(void)
```

```
{
    /* 1. 创建一个基础对象 */
    lv_obj_t *obj1 = lv_obj_create(lv_scr_act());
    /* 2. 设置基础对象的大小 */
    lv_obj_set_size(obj1, 100, 50);
    /* 3. 设置基础对象的对齐方式 */
    lv_obj_align(obj1, LV_ALIGN_CENTER, 0, 0);

    /* 4. 给基础对象设置样式 */
    static lv_style_t style_shadow;
    lv_style_init(&style_shadow);
    lv_style_set_shadow_width(&style_shadow, 10);
    lv_style_set_shadow_spread(&style_shadow, 5);
    lv_style_set_shadow_color(&style_shadow, lv_palette_main(LV_PALETTE_BLUE));
    lv_obj_add_style(obj1, &style_shadow, LV_PART_MAIN);
}
```

4.2.2 Core widgets（核心组件）

4.2.2.1 Label（标签）（lv_label）

标签是用来显示文本的基本对象类型。

```
void App_Display_CreateLable(void)
{
    /* 1. 创建标签对象 */
    lv_obj_t *label1 = lv_label_create(lv_scr_act());

    /* 2.设置标签对象对齐方式 */
    lv_obj_align(label1, LV_ALIGN_CENTER, 0, -60);

    /* 3. 设置长模式: 长文本自动换行 */
    lv_label_set_long_mode(label1, LV_LABEL_LONG_WRAP); /*Break the long lines*/
    /* 4. 对文本重新着色 */
    lv_label_set_recolor(label1, true); /*Enable re-coloring by commands in the
text*/
    /* 5. 设置文本 */
```

```
    lv_label_set_text(label1, "#0000ff Atguigu# #ff00ff Embedded# #ff0000 systems
discipline# is a new field "
                              "and has been very successful,"
                              "with prospects for even greater success in the
future.");
    /* 6. 设置宽度 */
    lv_obj_set_width(label1, 220); /*Set smaller width to make the lines wrap*/
    /* 7. 设置文本对齐方式 */
    lv_obj_set_style_text_align(label1, LV_TEXT_ALIGN_CENTER, 0);

    lv_obj_t *label2 = lv_label_create(lv_scr_act());
    lv_obj_align(label2, LV_ALIGN_CENTER, 0, 60);
    lv_label_set_long_mode(label2, LV_LABEL_LONG_SCROLL_CIRCULAR); /*Circular
scroll*/
    lv_obj_set_width(label2, 150);
    lv_label_set_text(label2, "Atguigu Embedded systems discipline is a new field "
                              "and has been very successful,"
                              "with prospects for even greater success in the
future.");
}
```

4.2.2.2 Button（按钮）（lv_btn）

```
void App_Display_CreateButton_1(void)
{
    lv_obj_t *btn1 = lv_btn_create(lv_scr_act());
    lv_obj_align(btn1, LV_ALIGN_CENTER, 0, -40);

    lv_obj_t *label = lv_label_create(btn1);
    lv_label_set_text(label, "Button");
    lv_obj_center(label);

    lv_obj_t *btn2 = lv_btn_create(lv_scr_act());
    lv_obj_align(btn2, LV_ALIGN_CENTER, 0, 40);
    lv_obj_add_flag(btn2, LV_OBJ_FLAG_CHECKABLE);
    lv_obj_set_height(btn2, LV_SIZE_CONTENT);
```

```
    label = lv_label_create(btn2);
    lv_label_set_text(label, "Toggle");
    lv_obj_center(label);
}
```

4.2.2.3 Button matrix（按钮矩阵）（lv_btnmatrix）

矩阵按钮（lv_btnmatrix）控件是一种在行和列中显示多个按钮的轻量级实现方式。

按钮不是实际创建出来的，而是实时绘制出来的，所以轻量级，因为这样一个按钮仅使用 8 个字节的内存，而不是普通 Button 控件那样：~100-150 字节再加上 Label 控件的内存占用。

```
static const char *btnm_map[] = {"1", "2", "3", "4", "5", "\n", "6", "7", "8", "9",
"0", "\n", "+", "-", "*", "/", "\n", "=", "other"};
/* 创建按钮矩阵 */
void App_Display_CreateButtonmatrix(void)
{
    lv_obj_t *btnm1 = lv_btnmatrix_create(lv_scr_act());
    lv_obj_align(btnm1, LV_ALIGN_CENTER, 0, 0);

    lv_btnmatrix_set_map(btnm1, btnm_map);
    lv_btnmatrix_set_btn_width(btnm1, 14, 2); /*Make "=" twice as wide as "Other"*/
    lv_btnmatrix_set_btn_ctrl(btnm1, 14, LV_BTNMATRIX_CTRL_CHECKABLE);
    lv_btnmatrix_set_btn_ctrl(btnm1, 15, LV_BTNMATRIX_CTRL_CHECKED);
    // lv_obj_add_event_cb(btnm1, event_handler, LV_EVENT_ALL, NULL);
}
```

4.2.2.4 Bar（进度条）（lv_bar）

进度条对象有一个背景和一个指示器。指示器的宽度根据进度条的当前值自动设置。如果设置进度条的宽度小于其高度，就可以创建出垂直摆放的进度条。不仅可以设置结束，还可以设置进度条的起始值，从而改变指标的起始位置。

```
void App_Display_CreateBar(void)
{
    lv_obj_t * bar1 = lv_bar_create(lv_scr_act());
    lv_obj_set_size(bar1, 200, 20);
```

```
    lv_obj_center(bar1);
    lv_bar_set_range(bar1, 0, 100);
    lv_bar_set_value(bar1, 70, LV_ANIM_OFF);
}
```

4.2.2.5 其他组件

其他组件参考官方文档，不在此一一展示了。工作种需要的时候，参考官方示例即可。

第 5 章 LVGL 高级使用

5.1 给组件添加事件

事件用于通知用户某个对象发生了某些事情。可以将一个或多个事件注册给一个对象，如果该对象被单击、释放、拖动、删除等将被调用。

可以通过函数 lv_obj_add_event_cb 来给对象注册事件回调函数。

lvgl 支持的所有事件如下：

```
typedef enum {
    LV_EVENT_ALL = 0,
    /** Input device events 输入设备事件*/
    LV_EVENT_PRESSED,            /**< The object has been pressed*/
    LV_EVENT_PRESSING,           /**< The object is being pressed (called
continuously while pressing)*/
    LV_EVENT_PRESS_LOST,         /**< The object is still being pressed but slid
cursor/finger off of the object */
    LV_EVENT_SHORT_CLICKED,      /**< The object was pressed for a short period of
time, then released it. Not called if scrolled.*/
    LV_EVENT_LONG_PRESSED,       /**< Object has been pressed for at least
`long_press_time`.  Not called if scrolled.*/
    LV_EVENT_LONG_PRESSED_REPEAT, /**< Called after `long_press_time` in every
`long_press_repeat_time` ms.  Not called if scrolled.*/
    LV_EVENT_CLICKED,            /**< Called on release if not scrolled (regardless
to long press)*/
    LV_EVENT_RELEASED,           /**< Called in every cases when the object has
been released*/
    LV_EVENT_SCROLL_BEGIN,       /**< Scrolling begins. The event parameter is a
pointer to the animation of the scroll. Can be modified*/
```

```
    LV_EVENT_SCROLL_END,          /**< Scrolling ends*/
    LV_EVENT_SCROLL,              /**< Scrolling*/
    LV_EVENT_GESTURE,             /**< A gesture is detected. Get the gesture with
`lv_indev_get_gesture_dir(lv_indev_get_act());` */
    LV_EVENT_KEY,                 /**< A key is sent to the object. Get the key with
`lv_indev_get_key(lv_indev_get_act());`*/
    LV_EVENT_FOCUSED,             /**< The object is focused*/
    LV_EVENT_DEFOCUSED,           /**< The object is defocused*/
    LV_EVENT_LEAVE,               /**< The object is defocused but still selected*/
    LV_EVENT_HIT_TEST,            /**< Perform advanced hit-testing*/

    /** Drawing events*/
    LV_EVENT_COVER_CHECK,        /**< Check if the object fully covers an area. The
event parameter is `lv_cover_check_info_t *`.*/
    LV_EVENT_REFR_EXT_DRAW_SIZE, /**< Get the required extra draw area around the
object (e.g. for shadow). The event parameter is `lv_coord_t *` to store the
size.*/
    LV_EVENT_DRAW_MAIN_BEGIN,    /**< Starting the main drawing phase*/
    LV_EVENT_DRAW_MAIN,          /**< Perform the main drawing*/
    LV_EVENT_DRAW_MAIN_END,      /**< Finishing the main drawing phase*/
    LV_EVENT_DRAW_POST_BEGIN,    /**< Starting the post draw phase (when all
children are drawn)*/
    LV_EVENT_DRAW_POST,          /**< Perform the post draw phase (when all children
are drawn)*/
    LV_EVENT_DRAW_POST_END,      /**< Finishing the post draw phase (when all
children are drawn)*/
    LV_EVENT_DRAW_PART_BEGIN,    /**< Starting to draw a part. The event parameter
is `lv_obj_draw_dsc_t *`. */
    LV_EVENT_DRAW_PART_END,      /**< Finishing to draw a part. The event parameter
is `lv_obj_draw_dsc_t *`. */

    /** Special events*/
    LV_EVENT_VALUE_CHANGED,       /**< The object's value has changed (i.e. slider
moved)*/
```

```
    LV_EVENT_INSERT,              /**< A text is inserted to the object. The event
data is `char *` being inserted.*/
    LV_EVENT_REFRESH,             /**< Notify the object to refresh something on it
(for the user)*/
    LV_EVENT_READY,               /**< A process has finished*/
    LV_EVENT_CANCEL,              /**< A process has been cancelled */

    /** Other events*/
    LV_EVENT_DELETE,              /**< Object is being deleted*/
    LV_EVENT_CHILD_CHANGED,       /**< Child was removed, added, or its size,
position were changed */
    LV_EVENT_CHILD_CREATED,       /**< Child was created, always bubbles up to all
parents*/
    LV_EVENT_CHILD_DELETED,       /**< Child was deleted, always bubbles up to all
parents*/
    LV_EVENT_SCREEN_UNLOAD_START, /**< A screen unload started, fired immediately
when scr_load is called*/
    LV_EVENT_SCREEN_LOAD_START,   /**< A screen load started, fired when the screen
change delay is expired*/
    LV_EVENT_SCREEN_LOADED,       /**< A screen was loaded*/
    LV_EVENT_SCREEN_UNLOADED,     /**< A screen was unloaded*/
    LV_EVENT_SIZE_CHANGED,        /**< Object coordinates/size have changed*/
    LV_EVENT_STYLE_CHANGED,       /**< Object's style has changed*/
    LV_EVENT_LAYOUT_CHANGED,      /**< The children position has changed due to a
layout recalculation*/
    LV_EVENT_GET_SELF_SIZE,       /**< Get the internal size of a widget*/

    _LV_EVENT_LAST,               /** Number of default events*/

    LV_EVENT_PREPROCESS = 0x80,   /** This is a flag that can be set with an event
so it's processed
                                     before the class default event processing */
} lv_event_code_t;
```

5.1.1 案例 1：给按钮添加点击事件

```
void btn1_cb(lv_event_t *e)
{
    debug_printfln("用户点击了按钮");

    /* 1. 获取发生事件的对象: 按钮 */
    lv_obj_t *btn1  = lv_event_get_target(e);
    /* 2. 获取按钮的第 0 个子对象: label */
    lv_obj_t *label = lv_obj_get_child(btn1, 0);
    /* 3. 设置 lable 的值 */
    static uint8_t cnt = 0;
    lv_label_set_text_fmt(label, "%d", cnt++);
}
void App_Display_CreateButton_1(void)
{

    lv_obj_t *btn1 = lv_btn_create(lv_scr_act());
    lv_obj_align(btn1, LV_ALIGN_CENTER, 0, -40);
    /* 给按钮添加点击事件 */
    lv_obj_add_event_cb(btn1, btn1_cb, LV_EVENT_CLICKED, NULL);

    lv_obj_t *label = lv_label_create(btn1);
    lv_label_set_text(label, "Button");
    lv_obj_center(label);

    lv_obj_t *btn2 = lv_btn_create(lv_scr_act());
    lv_obj_align(btn2, LV_ALIGN_CENTER, 0, 40);
    lv_obj_add_flag(btn2, LV_OBJ_FLAG_CHECKABLE);
    lv_obj_set_height(btn2, LV_SIZE_CONTENT);

    label = lv_label_create(btn2);
    lv_label_set_text(label, "Toggle");
    lv_obj_center(label);
}
```

5.1.2 案例 2：给滑块添加值变化事件

```
lv_obj_t   *slider_label;
static void slider_event_cb(lv_event_t *e)
{
    lv_obj_t *slider = lv_event_get_target(e);
    lv_label_set_text_fmt(slider_label, "%d%%", lv_slider_get_value(slider));
    lv_obj_align_to(slider_label, slider, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);
}

void App_Display_CreateSlide(void)
{
    /*Create a slider in the center of the display*/
    lv_obj_t *slider = lv_slider_create(lv_scr_act());
    lv_obj_center(slider);
    lv_obj_add_event_cb(slider, slider_event_cb, LV_EVENT_VALUE_CHANGED, NULL);



    /*Create a label below the slider */
    slider_label = lv_label_create(lv_scr_act());
    lv_label_set_text(slider_label, "0%");
    lv_obj_align_to(slider_label, slider, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);
}
```

5.2 添加动画

可以使用动画自动更改变量的值，该值在开始值和结束值之间变化。动画将通过定期调用具有相应值参数的“animator”函数来完成。

5.2.1 案例 1：在一个事件上添加动画

```
/* 动画 1 */
static void anim_x_cb(void *var, int32_t v)
{
    lv_obj_set_x(var, v);
}

static void sw_event_cb(lv_event_t *e)
{
```

```
    lv_obj_t *sw    = lv_event_get_target(e);
    lv_obj_t *label = lv_event_get_user_data(e);

    if(lv_obj_has_state(sw, LV_STATE_CHECKED))
    {
        lv_anim_t a;

        lv_anim_init(&a);
        lv_anim_set_var(&a, label);
        lv_anim_set_values(&a, lv_obj_get_x(label), 100);
        lv_anim_set_time(&a, 500);
        lv_anim_set_exec_cb(&a, anim_x_cb);
        lv_anim_set_path_cb(&a, lv_anim_path_overshoot);
        lv_anim_start(&a);
    }
    else
    {
        lv_anim_t a;
        lv_anim_init(&a);
        lv_anim_set_var(&a, label);
        lv_anim_set_values(&a, lv_obj_get_x(label), -lv_obj_get_width(label));
        lv_anim_set_time(&a, 500);
        lv_anim_set_exec_cb(&a, anim_x_cb);
        lv_anim_set_path_cb(&a, lv_anim_path_ease_in);
        lv_anim_start(&a);
    }
}
void App_Display_Anim_1(void)
{
    lv_obj_t *label = lv_label_create(lv_scr_act());
    lv_label_set_text(label, "Hello Atguigu!");
    lv_obj_set_pos(label, 100, 10);

    lv_obj_t *sw = lv_switch_create(lv_scr_act());
```

```
    lv_obj_center(sw);
    lv_obj_add_state(sw, LV_STATE_CHECKED);
    lv_obj_add_event_cb(sw, sw_event_cb, LV_EVENT_VALUE_CHANGED, label);
}
```

5.2.2 案例 2：播放动画

```
/* 动画 2 */

static void anim_size_cb(void *var, int32_t v)
{
    lv_obj_set_size(var, v, v);
}

/**
 * Create a playback animation
 */
void App_Display_Anim_2(void)
{

    lv_obj_t *obj = lv_obj_create(lv_scr_act());
    lv_obj_set_style_bg_color(obj, lv_palette_main(LV_PALETTE_RED), 0);
    lv_obj_set_style_radius(obj, LV_RADIUS_CIRCLE, 0);

    lv_obj_align(obj, LV_ALIGN_LEFT_MID, 10, 0);

    lv_anim_t a;
    lv_anim_init(&a);
    lv_anim_set_var(&a, obj);
    lv_anim_set_time(&a, 1000);
    lv_anim_set_playback_delay(&a, 100);
    lv_anim_set_playback_time(&a, 300);
    lv_anim_set_repeat_delay(&a, 500);
    lv_anim_set_repeat_count(&a, LV_ANIM_REPEAT_INFINITE);
    lv_anim_set_path_cb(&a, lv_anim_path_ease_in_out);
```

```
    lv_anim_set_values(&a, 10, 50);
    lv_anim_set_exec_cb(&a, anim_size_cb);
    lv_anim_start(&a);

    lv_anim_set_values(&a, 10, 240);
    lv_anim_set_exec_cb(&a, anim_x_cb);
    lv_anim_start(&a);
}
```

5.3 显示中文

要在 lvgl 中使用中文显示，我们需要用到两个东西：字体文件和字体转换器。

字体文件我们可以使用开源的字体或者自己制作出来，准备好了字体文件之后使用字体转换器即可转换成可以在 lvgl 上使用的字体格式。

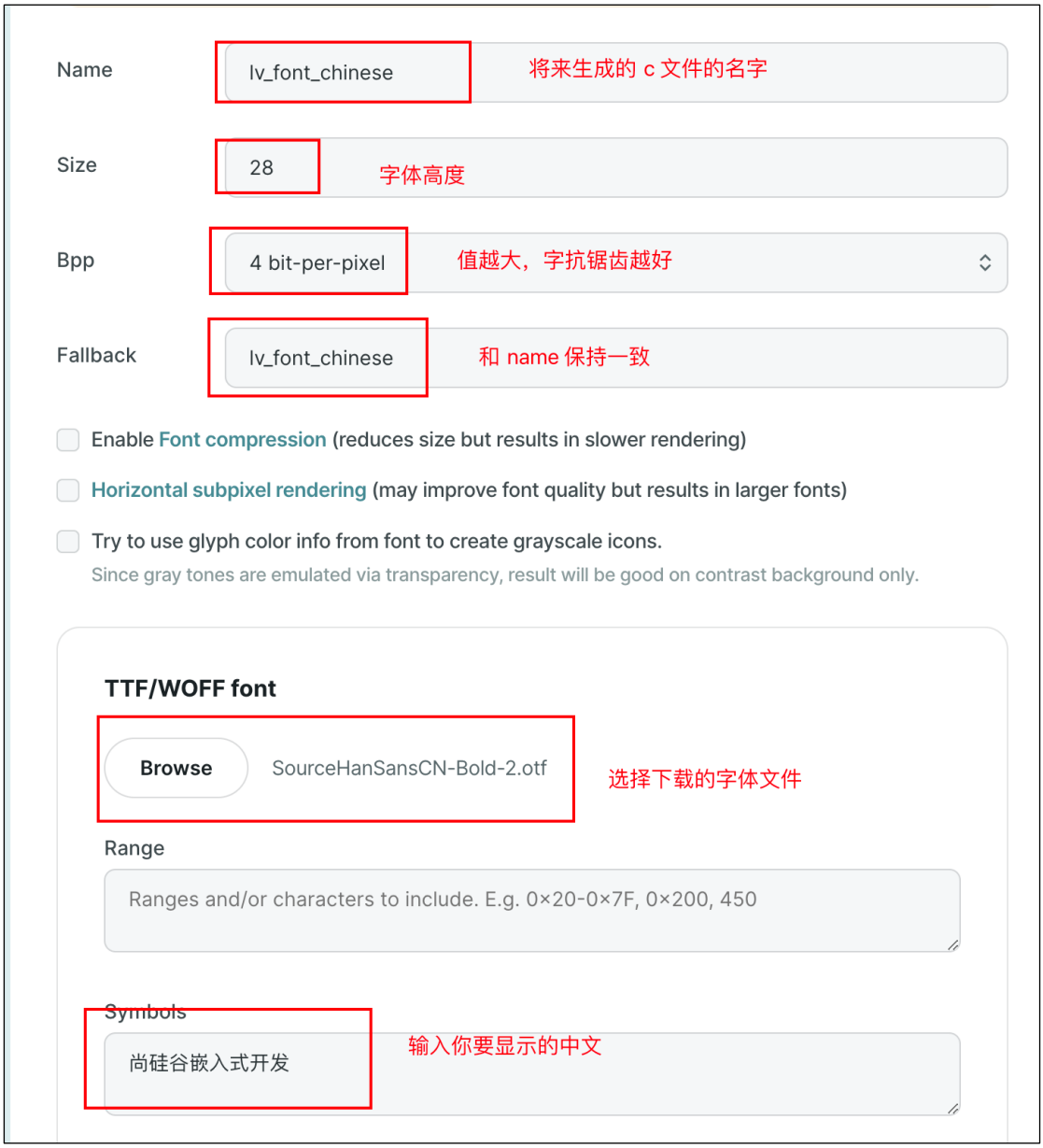
（1）下载免费字体文件

https://lvgl.100ask.net/master/extra/fonts-zh-source.html#id14

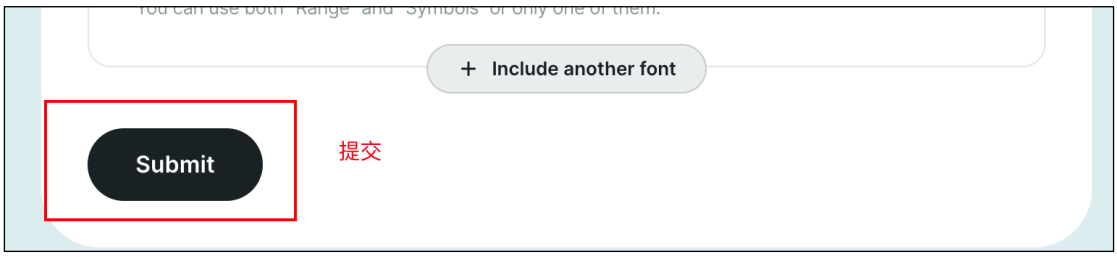
（2）打开 lvgl 字体转换器网站

https://lvgl.io/tools/fontconverter

（3）在页面进入相应的配置



（4）提交

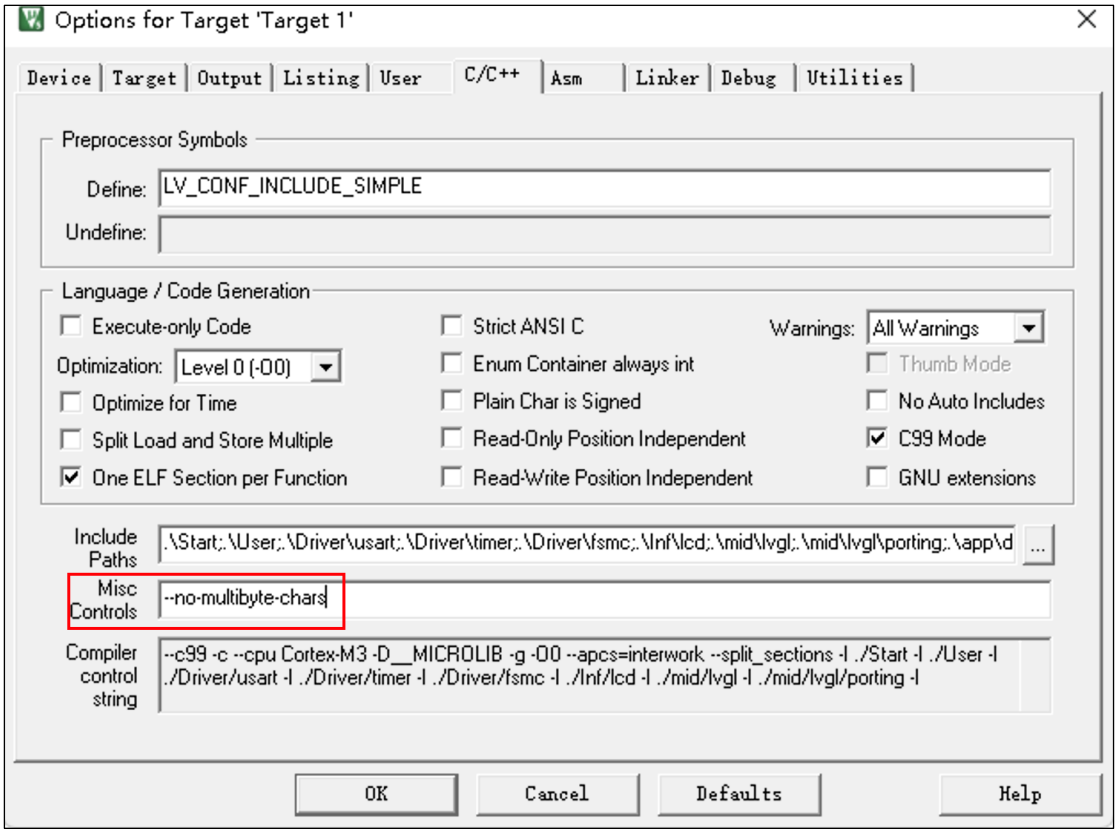


（5）把生成的 c 文件放入工程中

（6）示例代码. 注意,使用中文的 c 文件的编码必须选择 utf-8 格式.并且在 keil 中需要添加一项编译配置 --no-multibyte-chars 才能编译通过.

```
263  void App_Display_ShowChines(void)
264  {
265      LV_FONT_DECLARE(lv_font_chinese);
266
267      lv_obj_t *label1 = lv_label_create(lv_scr_act());
268
269      lv_obj_align(label1, LV_ALIGN_CENTER, 0, 0);
270      lv_label_set_long_mode(label1, LV_LABEL_LONG_WRAP); /*Break the long lines*/
271      lv_label_set_recolor(label1, true); /*Enable re-coloring by commands in the tex
272
273      lv_label_set_text(label1, "尚硅谷嵌入式开发");
274      lv_obj_set_width(label1, 220); /*Set smaller width to make the lines wrap*/
275      lv_obj_set_style_text_align(label1, LV_TEXT_ALIGN_CENTER, 0);
276      lv_obj_set_style_text_font(label1, &lv_font_chinese, LV_PART_MAIN);
277  }
```

```
OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                  powershell

PS W:\embedded_copy\atguigu_new\lvgl_demo\lvgl_register>

Ln 278, Col 1  Spaces: 4  UTF-8  CRLF  {}  C  Target
```




```
void App_Display_ShowChines(void)

{

    LV_FONT_DECLARE(lv_font_chinese); // 包含中文字体文件


    lv_obj_t *label1 = lv_label_create(lv_scr_act());
```

```
    lv_obj_align(label1, LV_ALIGN_CENTER, 0, 0);
    lv_label_set_long_mode(label1, LV_LABEL_LONG_WRAP); /*Break the long lines*/
    lv_label_set_recolor(label1, true); /*Enable re-coloring by commands in the
text*/

    lv_label_set_text(label1, "尚硅谷嵌入式开发");
    lv_obj_set_width(label1, 220); /*Set smaller width to make the lines wrap*/
    lv_obj_set_style_text_align(label1, LV_TEXT_ALIGN_CENTER, 0);
    // 设置字体样式
    lv_obj_set_style_text_font(label1, &lv_font_chinese, LV_PART_MAIN);
}
```