高通汉显®

高通®

GTHMI Engine

用户手册

V1.18

www.hmi.gaotongfont.cn

目录

1.概述

1.1 产品特性

GT-HMI Engine 是一个免费且开源的嵌入式 GUI 引擎，使用 MIT 协议开源，方便用户自由使用和二次开发。该引擎具有丰富的模块化图形组件、高级图形引擎且支持多种显示设备；GT-HMI Engine 支持中英、日韩、东西欧等 180 国多种文字语言，可搭配使用宋仿楷黑、高通优黑、世纪几何体等多种字体；使用 C 编写，支持跨平台操作系统，易于移植使用；且为开发人员提供详实的文档，GT-HMI Engine 高效的嵌入式 UI 引擎，可以帮助你更快地构建、集成和优化 UI 应用程序，轻松实现自己的图形界面想法。使用 GT-HMI Engine，体验高效优质的图形界面开发!

1.2 配置要求

在嵌入式端的环境建议

- 芯片主频 94MHz 及以上
- 芯片最小 RAM：24KB
- 最小 FLASH：128KB

2.控件

什么是控件？控件又称组件或者部件，指用户看到的所有可视化界面以及界面中的元素，例如按钮、标签、文本框等。为方便用户开发，高通 GT-HMI Designer 提供了多种现成的控件以满足用户实现相关功能的需求。而这些控件的实现，离不开 GT-HMI Engine 的底层支持。本文档将通过对各控件的简单调用效果及 GT-HMI Designer 生成的代码(你可以在工程项目文件的 screen 文件夹中找到 gt_init_screen_xxx.c 文件查看)协助你认识 GT-HMI Designer 的控件并理解 GT-HMI Engine 的控件接口函数是如何使用的。

2.1 屏幕控件

屏幕控件是所有其他控件的基础，在创建项目工程时，系统将默认创建一个屏幕控件，该控件背景颜色默认为白色，一个项目工程中需要存在一个以上屏幕控件，当只剩下最后一个屏幕控件时，该屏幕控件将无法删除。

```
gt_obj_st * gt_init_screen_home()
{
  //创建一个屏幕控件
  screen_home = gt_obj_create(NULL);
  //设置屏幕的背景颜色为白色
```

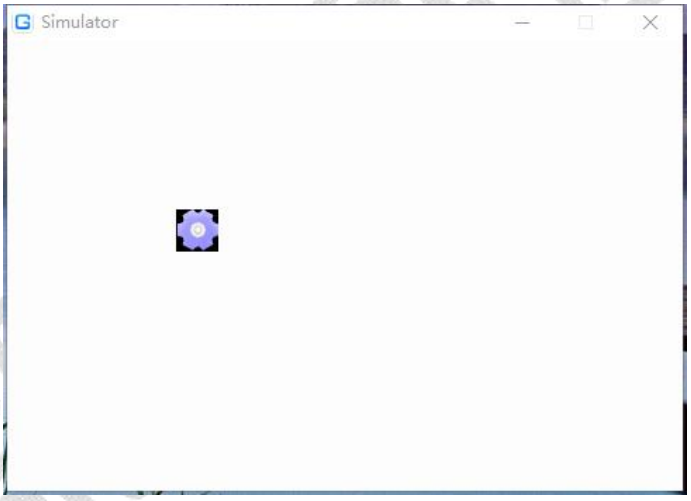
```
    gt_screen_set_bgcolor(screen_home,gt_color_hex(0xFFFFFF));

    return screen_home;
}
```

2.2 图片控件

在创建的屏幕控件中，添加一个图片控件并设置图片控件属性。

```
/*img1*/
//在 screen_home 屏幕控件中创建一个图片控件
img1 = gt_img_create(screen_home);
//设置图片控件的位置为(120,120)
gt_obj_set_pos(img1,120,120);
//设置图片控件的尺寸为图片的大小尺寸(宽度 30 高度 30)
gt_obj_set_size(img1,30,30);
//设置图片控件的图片数据访问路径
gt_img_set_src(img1,".:20.bmp");
```

运行后的图片控件效果如下图所示：



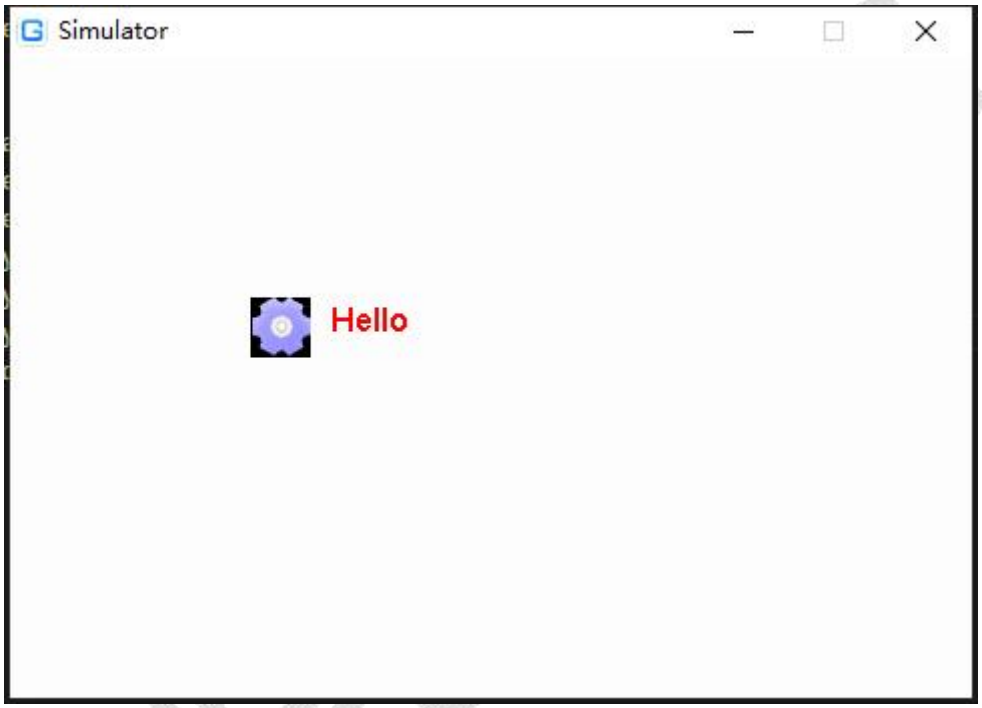
2.3 标签控件

在创建的屏幕控件中再添加一个标签控件并设置标签控件属性。

```
/*lab1*/
//在 screen_home 屏幕控件中创建一个标签控件
lab1 = gt_label_create(screen_home);
//设置标签控件的位置为(120,120)
gt_obj_set_pos(lab1,160,120);
//设置标签控件的尺寸为图片的大小尺寸(宽度 30 高度 30)
gt_obj_set_size(lab1,50,24);
//设置标签控件文本颜色为 0xff0000 红色
gt_label_set_font_color(lab1,gt_color_hex(0xff0000));
```

```
//设置标签控件显示的字体尺寸为 16 点
gt_label_set_font_size(lab1, 16);
//设置对齐方式为左对齐
gt_label_set_font_align(lab1, GT_ALIGN_LEFT);
////设置标签控件显示的文本为"Hello"
gt_label_set_text(lab1,"Hello");
//设置标签控件首行缩进两个字符
gt_label_set_indent(lab1, 4);
```

运行后的标签控件效果如下图所示



2.4 文本区控件

在创建的屏幕控件中添加一个文本区控件并设置文本区控件属性。

```
/*txt1*/
//在 screen_home 屏幕控件中创建一个文本区控件
txt1 = gt_textarea_create(screen_home);
//设置文本区控件的位置为(70,70)
gt_obj_set_pos(txt1,70,70);
//设置文本区控件的尺寸为(宽度 200 高度 90)
gt_obj_set_size(txt1,200,90);
//设置文本区控件文本颜色为 0xffffff 白色
gt_textarea_set_font_color(txt1,gt_color_hex(0xffffff));
//设置文本区控件显示的字体尺寸为 16 点
gt_textarea_set_font_size(txt1, 16);
//设置对齐方式为左对齐
gt_textarea_set_font_align(txt1, GT_ALIGN_LEFT);
```

```
    //文本区控件增加文本 以 0 默认方式增加内容为"Hello, this is a textarea."的 0xffffff 白色文本
    gt_textarea_add_str(txt1,"Hello, this is a textarea.",0,gt_color_hex(0xffffff));
    //设置文本区控件本身背景颜色为 0x000000 黑色
    gt_textarea_set_bg_color(txt1,gt_color_hex(0x000000));
```

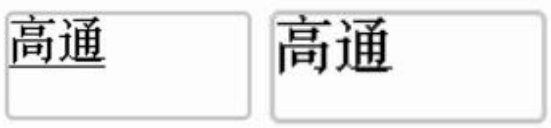
运行后的文本区控件效果如下图所示

当文本区域的文本超长时，可选择单行省略和滚动两种显示方式。

```
    //设置文本区控件为超长省略显示方式
    gt_label_set_omit_single_line(txt1, true);
    //设置文本区控件为滚动显示方式
    gt_label_set_auto_scroll_single_line(txt1, true);
    //设置文本区控件的圆角半径
    gt_obj_set_radius(txt1, 57);
    //设置文本区控件的首行缩进两个字符
    gt_textarea_set_indent(txt1, 4);
```

```
    //设置文本区字体风格为下划线
    gt_textarea_add_str(txt1, "高通", 0 | 1, gt_color_hex(0x000000));
```

左图为下划线风格，右图为默认风格



```
    //设置文本区字体风格为删除线
    gt_textarea_add_str(txt2, "高通", 0 | 2, gt_color_hex(0x000000));
```


```
    //设置文本区字体风格为粗体
    gt_textarea_add_str(txt3, "高通", 0 | 4, gt_color_hex(0x000000));
```

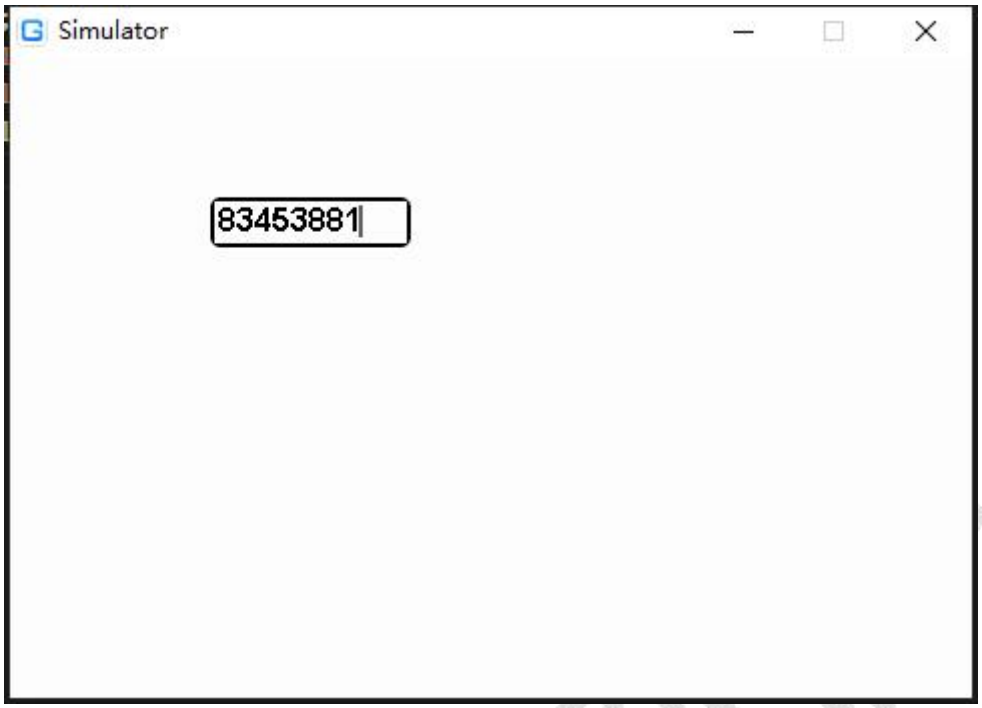

```
    //设置文本区字体风格为斜体
    gt_textarea_add_str(txt4, "高通", 0 | 8, gt_color_hex(0x000000));
```

2.5 输入框控件

在创建的屏幕控件中添加一个输入框控件并设置输入框控件属性。

```
/*input1*/
//在 screen_home 屏幕控件中创建一个输入框控件
input1 = gt_input_create(screen_home);
//设置输入框控件位置为(100,70)
gt_obj_set_pos(input1,100,70);
//设置输入框控件的尺寸为(宽度 100 高度 25)
gt_obj_set_size(input1,100,25);
//设置输入框控件字体颜色为 0x000000 黑色
gt_input_set_font_color(input1,gt_color_hex(0x000000));
//设置输入框控件显示的字体尺寸为 16 点
gt_input_set_font_size(input1, 16);
//设置输入框控件文本为"83453881"
    gt_input_set_value(input1,"83453881");
//设置输入框控件占位符为"placeholder"(输入框无文本时显示)
gt_input_set_placeholder(input1,"placeholder");
    //设置对齐方式为左对齐
gt_input_set_font_align(input1, GT_ALIGN_LEFT);
//设置输入框控件本身背景颜色为 0xffffff 白色
    gt_input_set_bg_color(input1,gt_color_hex(0xffffff));
```

运行后的输入框控件效果如下图所示

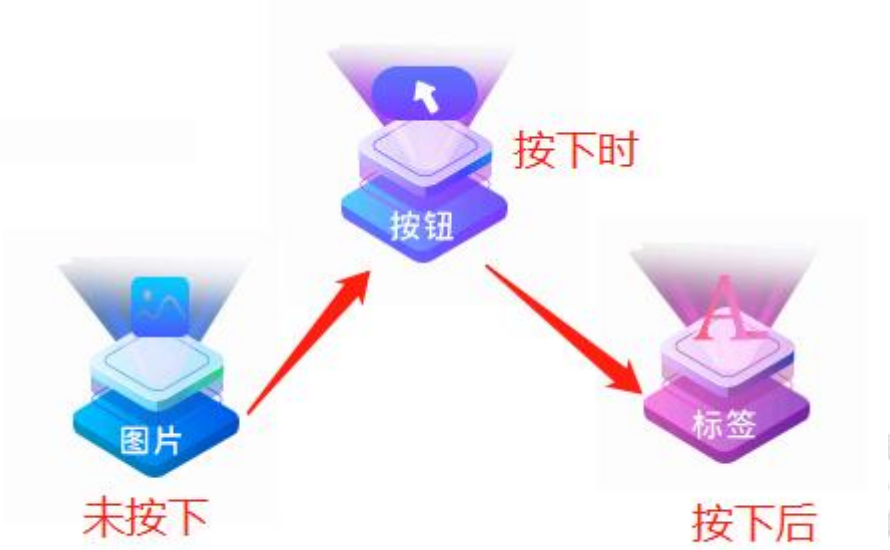


2.6 图片按钮控件

在创建的屏幕控件中添加一个图片按钮控件并设置图片按钮控件属性。

```
/*imgbtn1*/
//在 screen_home 屏幕控件中创建一个图片按钮控件
imgbtn1 = gt_imgbtn_create(screen_home);
//设置图片控件的位置为(150,100)
gt_obj_set_pos(imgbtn1,150,100);
//设置图片控件的尺寸为图片的大小尺寸(宽度 30 高度 30)
gt_obj_set_size(imgbtn1,30,30);
//设置图片按钮控件按下时的图片数据访问路径
gt_imgbtn_set_src_press(imgbtn1,"f:img_97x110_112.png");
//设置图片按钮控件未按下时的图片数据访问路径
gt_imgbtn_set_src(imgbtn1,"f:img_97x110_110.png");
//设置图片控件按下后切换的图片数据访问路径
gt_imgbtn_add_state_item(imgbtn1, "f:img_97x110_113.png");
```

运行后的图片控件效果如下图所示，未按下时显示 110.png，按下不松手时显示 112.png 图片，按下后显示 113.png。



2.7 键盘控件

在创建的屏幕控件中添加一个键盘控件并设置键盘控件属性。

```
    /** keyboard1 */
    //在 screen_home 屏幕控件中创建一个键盘控件
    keyboard1 = gt_keypad_create(screen_home);
    //设置键盘的位置为（91,105）
    gt_obj_set_pos(keyboard1, 91, 105);
    //设置键盘的尺寸为（宽度 300 高度 170）
    gt_obj_set_size(keyboard1, 300, 170);
    //设置键盘字体颜色为 0x000000 黑色
    gt_keypad_set_font_color(keyboard1, gt_color_hex(0x000000));
    //设置键盘字体尺寸为 16 点
    gt_keypad_set_font_size(keyboard1, 16);
    //设置键盘面板颜色为 0x242424 浅黑色
    gt_keypad_set_color_background(keyboard1, gt_color_hex(0x242424));
    //设置键盘普通键颜色为 0x646464 浅灰色
    gt_keypad_set_key_color_background(keyboard1,gt_color_hex(0x646464);
    //设置键盘控制键颜色为 0x3E3E3E 深灰色
    gt_keypad_set_ctrl_key_color_background(keyboard1,gt_color_hex(0x3E
3E3E));
    //设置键盘边框圆角半径为 6
    gt_keypad_set_radius(keyboard1, 6);
    //设置键盘关联到输入框 input1
    gt_keypad_set_target(keyboard1, input1);
```

```
    //修改键盘样式
```

```
gt_keypad_set_default_style(obj, GT_KEYPAD_STYLE_26_KEY);
//设置键盘自动隐藏模式
gt_keypad_set_auto_hide(keyboard1, true);
```

运行后的键盘控件效果如下图所示:


```
//设置键盘控件的风格为下划线
gt_keypad_set_font_style(keyboard1, 1);
```

```
//设置键盘控件的风格为删除线
gt_keypad_set_font_style(keyboard2, 2);
```

```
//设置键盘控件的风格为粗体
gt_keypad_set_font_style(keyboard3, 4);
```

```
//设置键盘控件的风格为斜体
gt_keypad_set_font_style(keyboard4, 8);
```

2.8 开关控件

在创建的屏幕控件中添加一个开关控件并设置开关控件属性。

```
/*switch1*/
//在 screen_home 屏幕控件中创建一个开关控件
switch1 = gt_switch_create(screen_home);
//设置开关控件位置为(150,110)
gt_obj_set_pos(switch1,150,100);
//设置开关控件的尺寸为(宽度 50 高度 20)
gt_obj_set_size(switch1,50,20);
//设置开关控件的初始活跃状态为活跃
gt_obj_set_state(switch1,1);
//设置开关控件活跃颜色为 0x13ce66 的绿色
gt_switch_set_color_act(switch1,gt_color_hex(0x13ce66));
//设置开关控件不活跃颜色为 0xff0000 的红色
gt_switch_set_color_ina(switch1,gt_color_hex(0xff0000));
//设置开关控件开关点的颜色为 0xffff00 的黄色
gt_switch_set_color_point(switch1,gt_color_hex(0xffff00));
//设置开关的装饰线
gt_switch_set_div_line(switch1, true);
```

```
//设置开关控件的装饰线颜色
gt_switch_set_color_divider(switch1, gt_color_hex(0xffffff));
//设置开关控件的装饰线高度和宽度
gt_switch_set_div_line_size(switch1, 5, 30);
```

运行后的开关控件效果如下图所示，点击控件可以切换开关的状态。

2.9 幻灯片控件

在创建的屏幕控件中添加一个幻灯片控件并设置控件属性。

```
/** player1 */
//在 screen_home 屏幕控件中创建一个幻灯片控件
player1 = gt_player_create(screen_home);
//设置幻灯片控件位置为(114,97)
gt_obj_set_pos(player1, 114, 97);
//设置幻灯片控件的尺寸为(宽度 148 高度 77)
gt_obj_set_size(player1, 148, 77);
//添加三张图片项目到幻灯片控件
gt_player_add_item(player1, "f:img_148x77_fh.png", sizeof("f:img_148x77_fh.png"));
gt_player_add_item(player1, "f:img_148x77_113.png", sizeof("f:img_148x77_113.png"));
gt_player_add_item(player1, "f:img_148x77_110.png", sizeof("f:img_148x77_110.png"));
//设置幻灯片控件文件类型为 image 图片
gt_player_set_type(player1, GT_PLAYER_TYPE_IMG);
//设置幻灯片控件播放模式为循环播放
gt_player_set_mode(player1, GT_PLAYER_MODE_LOOP);
//设置幻灯片控件从序号 1 项(0 开始算起)开始播放
gt_player_set_index(player1, 1);
//设置幻灯片控件播放间隔为 500ms
gt_player_set_auto_play_period(player1, 500);
//开始播放
```

```
gt_player_play(player1);
```

运行后的幻灯片控件效果如下图所示，以每张间间隔 500ms 的时间循环切换播放设置的三张图片。

2.10 计数器控件

在创建的屏幕控件中添加一个计数器控件并设置控件属性。

```
/** inputNum1 */
//在 screen_home 屏幕控件中创建一个幻灯片控件
inputNum1 = gt_input_number_create(screen_home);
//设置计数器控件位置为(68,96)
gt_obj_set_pos(inputNum1, 68, 96);
//设置计数器控件的尺寸为(宽度 108 高度 24)
gt_obj_set_size(inputNum1, 108, 24);
//设置字体颜色为 0x000000 的黑色
gt_input_number_set_font_color(inputNum1, gt_color_hex(0x000000));
//设置字号为 16 点
gt_input_number_set_font_size(inputNum1, 16);
//设置对齐方式为左对齐
gt_input_number_set_font_align(inputNum1, GT_ALIGN_LEFT);
//设置计数值为 2
gt_input_number_set_value(inputNum1, 2);
//设置递增/递减的步进值为 200
gt_input_number_set_step(inputNum1, 200);
//设置最小值为 0
```

```
gt_input_number_set_min(inputNum1, 0);
//设置最大值为 1000
```

```
gt_input_number_set_max(inputNum1, 1000);
//开启格式化补齐数值
gt_input_number_set_fill_zero_front(inputNum1, true);
//设置整数部分显示宽度为 3
gt_input_number_set_display_integer_length(inputNum1, 3);
//设置小数部分显示宽度为 1
gt_input_number_set_display_decimal_length(inputNum1, 1);
```

运行后的计数器显示效果如下，2 不足 3 位以 0 补齐为 002，保留 1 位小数显示为 002.0。与 add 按钮配合触发递增事件后，增加设定的步进值 200 变为 202.0。

2.11 按钮控件

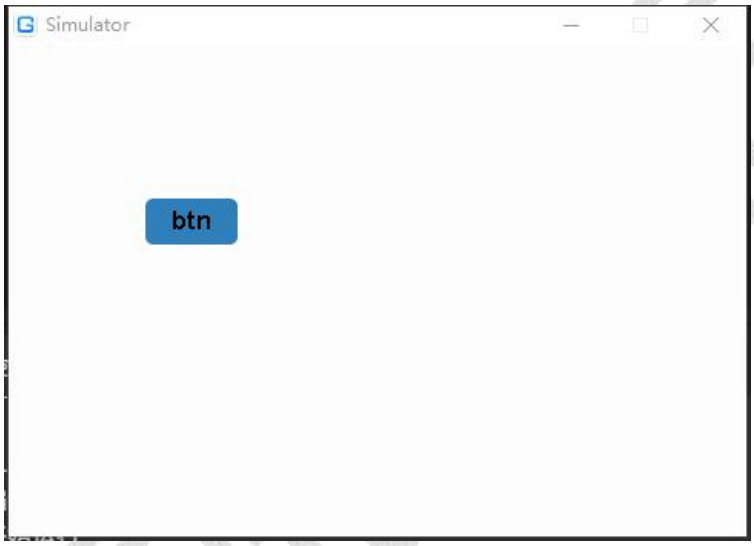
在创建的屏幕控件中，添加一个按钮控件并设置按钮的属性。

```
//设置屏幕的背景颜色为白色
gt_screen_set_bgcolor(screen_home,gt_color_hex(0xFFFFFF));
/*btn1*/
//在 screen_home 屏幕控件中创建一个 Button 按钮控件
btn1 = gt_btn_create(screen_home);
//设置 Button 控件的位置为(100,100)
gt_obj_set_pos(btn1,100,100);
//设置 Button 控件的尺寸(宽度 60 高度 30)
gt_obj_set_size(btn1,60,30);
//设置 Button 控件显示的字体颜色为黑色
gt_btn_set_font_color(btn1,gt_color_hex(0x000000));
//设置 Button 控件显示的字体尺寸为 16 点
gt_btn_set_font_size(btn1, 16);
//设置 Button 控件显示的文本为“btn”
gt_btn_set_text(btn1,"btn");
//设置 Button 控件本身的背景颜色为 R-0x35 G-0x80 B-0xbb 的蓝色
gt_btn_set_color_background(btn1,gt_color_hex(0x3580bb));
//设置 Button 控件圆角矩形圆角的圆半径为 4
gt_btn_set_radius(btn1,4);
//设置中文字体
gt_btn_set_font_family_cn(btn1, 11);
//设置外文字体
  gt_btn_set_font_family_en(btn1, 41);
  //设置数字符号
```

```
gt_btn_set_font_family_numb(btn1, 113);
 //设置对齐方式为居中对齐
```

```
gt_btn_set_font_align(btn1, GT_ALIGN_CENTER);
//设置字 x 间隔为 1，y 间隔为 0
gt_btn_set_space(btn1, 1, 0);
//添加点击时文本内容变化为 123
gt_btn_add_state_content(btn1, "123");
//设置按钮控件首行缩进两个字符
gt_btn_set_indent(btn1, 4);
```

运行后的按钮控件效果如下所示：



2.12 时钟控件

在创建的屏幕控件中添加一个时钟控件并设置控件属性。

```
/** clock1 */
//在 screen_home 屏幕控件中创建一个时钟控件
clock1 = gt_clock_create(screen_home);
//设置时钟控件位置为(168,123)
gt_obj_set_pos(clock1, 168, 123);
//设置时钟控件的尺寸为(宽度 200,高度 24)
gt_obj_set_size(clock1, 200, 24);
//设置字体颜色为 0x000000 的黑色
gt_clock_set_font_color(clock1, gt_color_hex(0x000000));
//设置字号为 24 点
gt_clock_set_font_size(clock1, 24);
//设置中文字体
gt_clock_set_font_family_cn(clock1, 5);
  //设置英文字体
gt_clock_set_font_family_en(clock1, 41);
```

```
//设置当前时间为 0 时 5 分 0 秒
gt_clock_set_time(clock1, 0, 5, 0);
//设置定时时间为 0 时 3 分 0 秒
gt_clock_set_alert_time(clock1, 0, 3, 0);
//设置时钟的工作模式为倒计时
```

```
    gt_clock_set_mode(clock1, GT_CLOCK_MODE_COUNTDOWN);
//设置时钟的显示格式
gt_clock_set_format(clock1, "hh 时 mm 分 ss 秒");
//设置对齐方式为左对齐
gt_clock_set_font_align(clock1, GT_ALIGN_LEFT);
//设置为 12 小时制
gt_clock_set_12_hours_mode(clock1, true);
//设置启用 AM/PM
gt_clock_set_meridiem_mode(clock1, true);
//开启时钟
gt_clock_start(clock1);
```

运行后的时钟控件显示效果如下：在倒计时到 3 分时，将会触发你添加的定时事件。

00时05分00秒

```
  //设置年-月-日 星期 时:分:秒
  gt_clock_set_format(clock1, "yyyy-MM-dd EEE hh:mm:ss");
```

2.13 条形码控件

在创建的屏幕控件中添加一个条形码控件并设置控件属性。

```
   /** barcode1 */
// 在 screen_home 屏幕控件中创建一个条形码控件
barcode1 = gt_barcode_create(screen_home);
// 设置控件坐标
gt_obj_set_pos(barcode1, 300, 100);
// 设置控件尺寸
gt_obj_set_size(barcode1, 138, 80);
// 设置条形码类型为 EAN-13
gt_barcode_set_type(barcode1,GT_FAMILY_BARCODE_TYPE_EAN_13);
// 设置条形码模块宽度为 2 个像素
gt_barcode_set_mode_w(barcode1, 2);
// 设置条形码模块高度为 64 个像素
gt_barcode_set_mode_h(barcode1,64);
// 设置条形码内容
```

```
gt_barcode_set_code_text(barcode1, "1234567890128");
```

运行后的条形码控件效果如下图所示：

2.14 二维码控件

在创建的屏幕控件中添加一个二维码控件并设置控件属性。

```
/** qrcode1 */
// 在 screen_home 屏幕控件中创建一个二维码控件
qrcode1 = gt_qrcode_create(screen_home);
// 设置控件坐标
gt_obj_set_pos(qrcode1, 468, 356);
// 设置控件尺寸
gt_obj_set_size(qrcode1, 37, 37);
// 设置二维码版本
gt_qrcode_set_version(qrcode1,GT_FAMILY_QRCODE_VERSION_8);
// 设置二维码背景色
gt_qrcode_set_background(qrcode1,gt_color_hex(0xFFFFFF));
// 设置二维码前景色
gt_qrcode_set_forecolor(qrcode1,gt_color_hex(0x000000));
// 设置二维码内容
    gt_qrcode_set_str(qrcode1,"https://www.hmi.gaotongfont.cn/");
```

运行后的二维码控件效果如下图所示：

2.15 滑窗控件

在创建的屏幕控件中添加一个滑窗控件并设置控件属性。

```
//在 screen_home 屏幕控件中创建一个滑窗控件
viewPager1 = gt_view_pager_create(screen_home);
//设置滑窗控件的位置(0,0)
gt_obj_set_pos(viewPager1, 0, 0);
//设置滑窗控件的尺寸(宽度 480,高度 320 默认为屏幕尺寸)
gt_obj_set_size(viewPager1, 480, 320);
//设置滑窗控件的透明度
gt_obj_set_opa(viewPager1, GT_OPA_0);
//设置滑窗控件的页数
gt_view_pager_set_fragment_count(viewPager1, 2);
//设置滑窗控件背景的玻璃效果
gt_view_pager_set_glass(viewPager1, 0);
```

与之对应的在其他控件设置所属的滑窗，以添加控件到滑窗内部。

```
//为滑窗控件的第一页(从 0 开始)添加图片控件 1
gt_view_pager_fragment_add_widget(viewPager1, 0, img1);
//为滑窗控件的第二页添加图片控件 4
gt_view_pager_fragment_add_widget(viewPager1, 1, img4);
```

运行后的滑窗结果如下图所示：

2.16 分组控件

在创建的屏幕控件中添加一个分组控件。分组组件为逻辑抽象控件，并不会在屏幕上

有显示效果，无需更改样式属性。通常与单选框，复选框组合使用。

```
/*group1*/
//在 screen_home 屏幕控件中创建一个分组控件
group1 = gt_group_create(screen_home);
```

2.17 单选框控件

在创建的分组 1 控件中添加一个单选框控件并设置单选框控件属性。

```
/*radio1*/
//在创建的分组 1 控件中创建一个单选框控件
radio1 = gt_radio_create(group1);
//设置单选框控件位置为(83,155)
gt_obj_set_pos(radio1,83,155);
//设置单选控件的尺寸为(宽度 60 高度 24)
gt_obj_set_size(radio1,60,24);
//设置单选框控件字体颜色为 0x000000 黑色
gt_radio_set_font_color(radio1,gt_color_hex(0x000000));
//设置单选框控件显示的字体尺寸为 16 点
gt_radio_set_font_size(radio1, 16);
//设置对齐方式为左对齐
gt_radio_set_font_align(radio1, GT_ALIGN_LEFT);
//设置单选框控件文本为"one"
gt_radio_set_text(radio1,"one");
//设置单选框状态为选中状态
 gt_radio_set_selected(radio1);
```

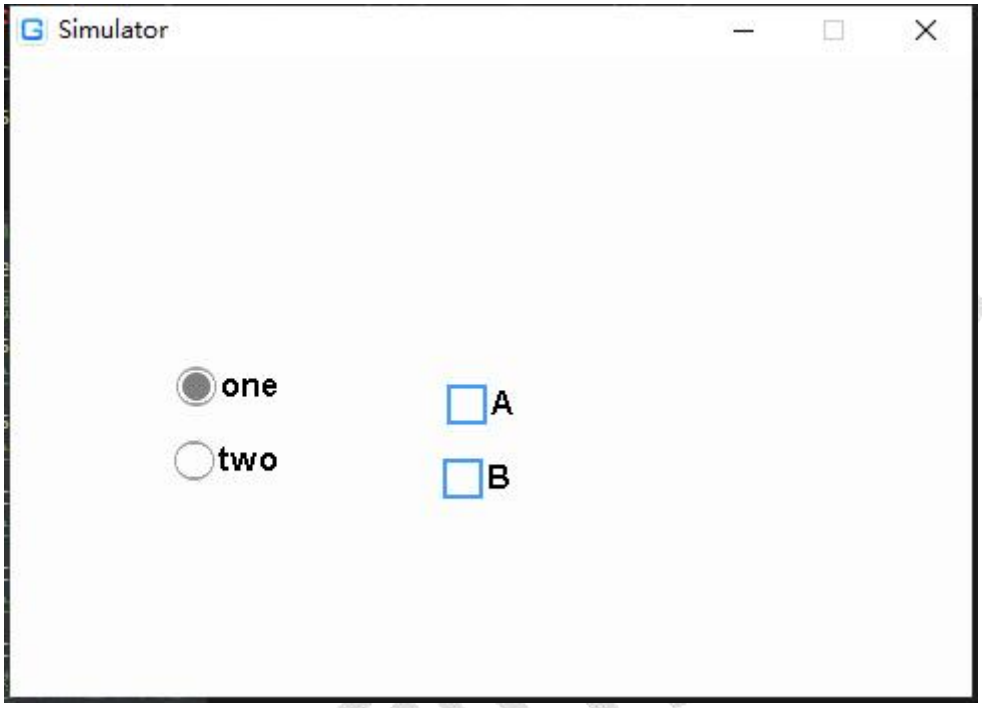
2.18 复选框控件

在创建的分组 2 中添加一个复选框控件并设置复选框控件属性。

```
/*cbox1*/
//在创建的分组 2 控件中创建一个复选框控件
cbox1 = gt_checkbox_create(group2);
//设置复选框位置为(218,164)
gt_obj_set_pos(cbox1,218,164);
//设置复选框控件的尺寸为(宽度 84 高度 24)
gt_obj_set_size(cbox1,84,24);
//设置复选框控件字体颜色为 0x000000 黑色
gt_checkbox_set_font_color(cbox1,gt_color_hex(0x000000));
//设置复选框控件显示的字体尺寸为 16 点
gt_checkbox_set_font_size(cbox1, 16);
//设置对齐方式为左对齐
gt_checkbox_set_font_align(cbox1, GT_ALIGN_LEFT);
```

```
//设置复选框控件文本为"83453881"
gt_checkbox_set_text(cbox1,"A");
//设置复选框状态为未选中状态
gt_obj_set_state(cbox1,0);
```

下图为添加两个单选框到分组 1，两个复选框到分组 2 的运行效果。

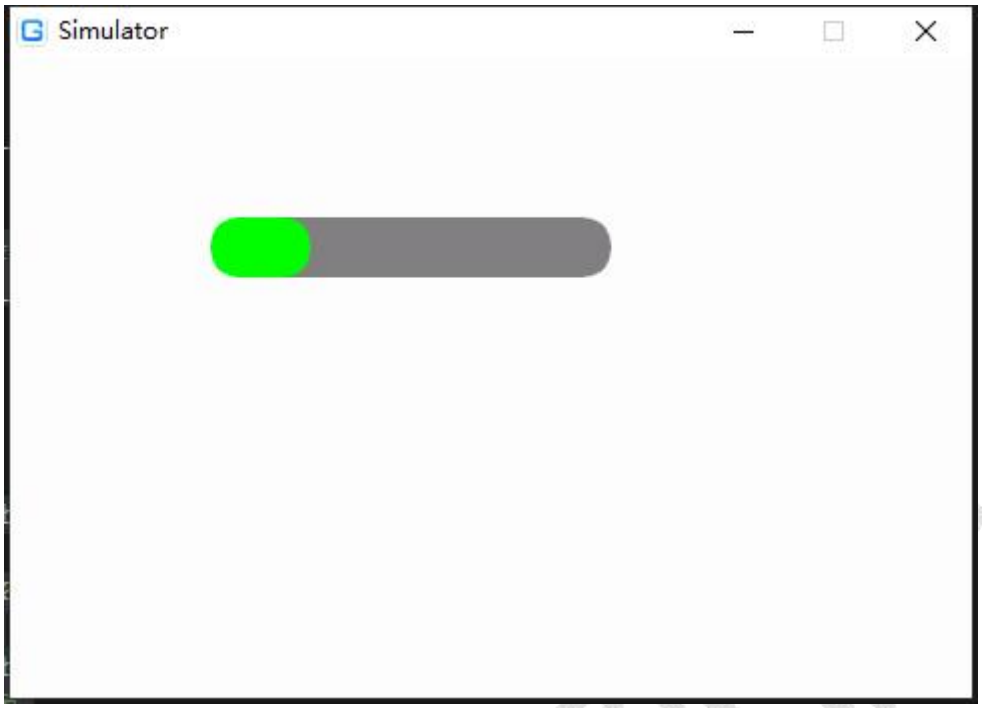


2.19 进度条控件

在创建的屏幕控件中添加一个进度条控件并设置进度条控件属性。

```
/*bar1*/
//在 screen_home 屏幕控件中创建一个进度条控件
bar1 = gt_progress_bar_create(screen_home);
//设置进度条的位置为(100,80)
gt_obj_set_pos(bar1,100,80);
//设置进度条的尺寸为(宽度 200 高度 30)
gt_obj_set_size(bar1,200,30);
//设置进度条的起始值为-20 结束值为 180
gt_progress_bar_set_start_end(bar1,-20,180);
//设置进度条的当前值为 30
gt_progress_bar_set_pos(bar1,30);
//设置进度条的活跃颜色为 0x00ff00 的绿色
gt_progress_bar_set_color_act(bar1,gt_color_hex(0x00ff00));
//设置进度条的不活跃颜色为 0x808080 的灰色
gt_progress_bar_set_color_ina(bar1,gt_color_hex(0x808080));
//设置进度条的方向为从左到右
gt_progress_bar_set_dir(bar1,GT_BAR_DIR_HOR_L2R);
```

运行后的进度条控件效果如下图所示。



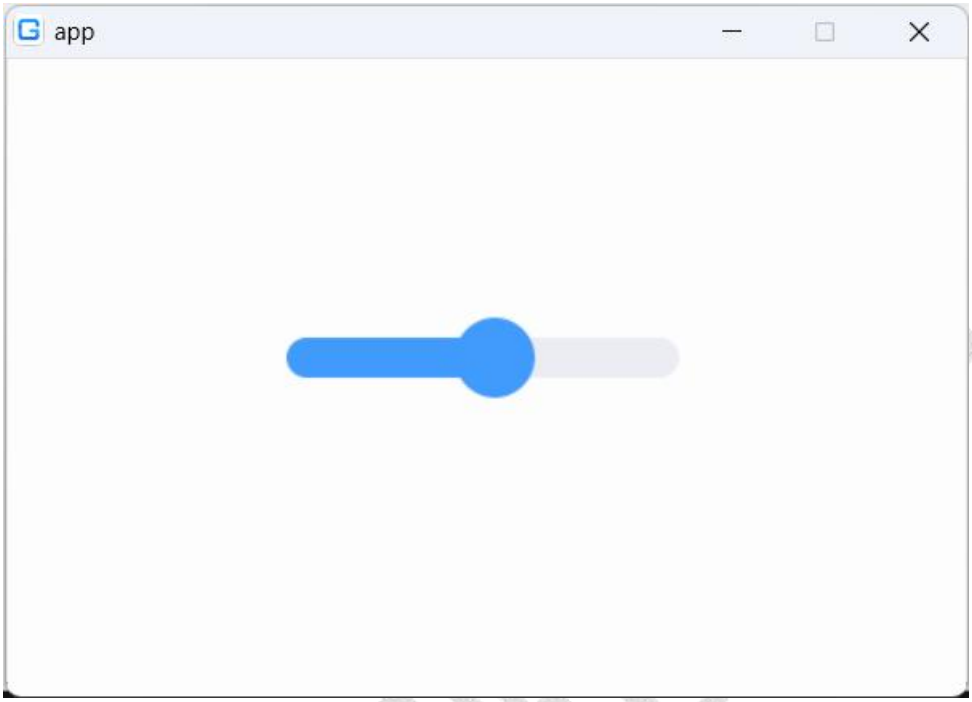
2.20 滑动条控件

在创建的屏幕控件中添加一个滑动条控件并设置滑动条控件属性。

```
  /** slider1 */
//在 screen_home 屏幕控件中创建一个滑动条控件
slider1 = gt_slider_create(screen_home);
//设置滑动条的位置为（120,130）
gt_obj_set_pos(slider1, 120, 130);
//设置滑动条的尺寸为（宽度 236 高度 40）
gt_obj_set_size(slider1, 236, 40);
//设置滑动条的起始值为 0，结束值为 100
gt_slider_set_start_end(slider1, 0, 100);
//设置滑动条的当前值为 50
gt_slider_set_pos(slider1, 50);
//设置滑动条的活跃颜色为 0x409eff 蓝色
gt_slider_set_color_act(slider1, gt_color_hex(0x409eff));
//设置滑动条的不活跃颜色为 0xebeef5 浅灰色
gt_slider_set_color_ina(slider1, gt_color_hex(0xebeef5));
//设置滑动条的刻度点可见
gt_slider_set_tag_visible(slider1, 1);
//设置滑动条的滑动方向为从左到右
gt_slider_set_dir(slider1, GT_BAR_DIR_HOR_L2R);
//设置滑动条的长条宽度为 20
```

```
gt_slider_set_thickness(slider1, 20);
//设置滑动条的刻度点大小为 40
gt_slider_set_tag_size(slider1, 40);
```

运行后的滑动条控件效果如下图所示，可以拖动设置滑动条的值。



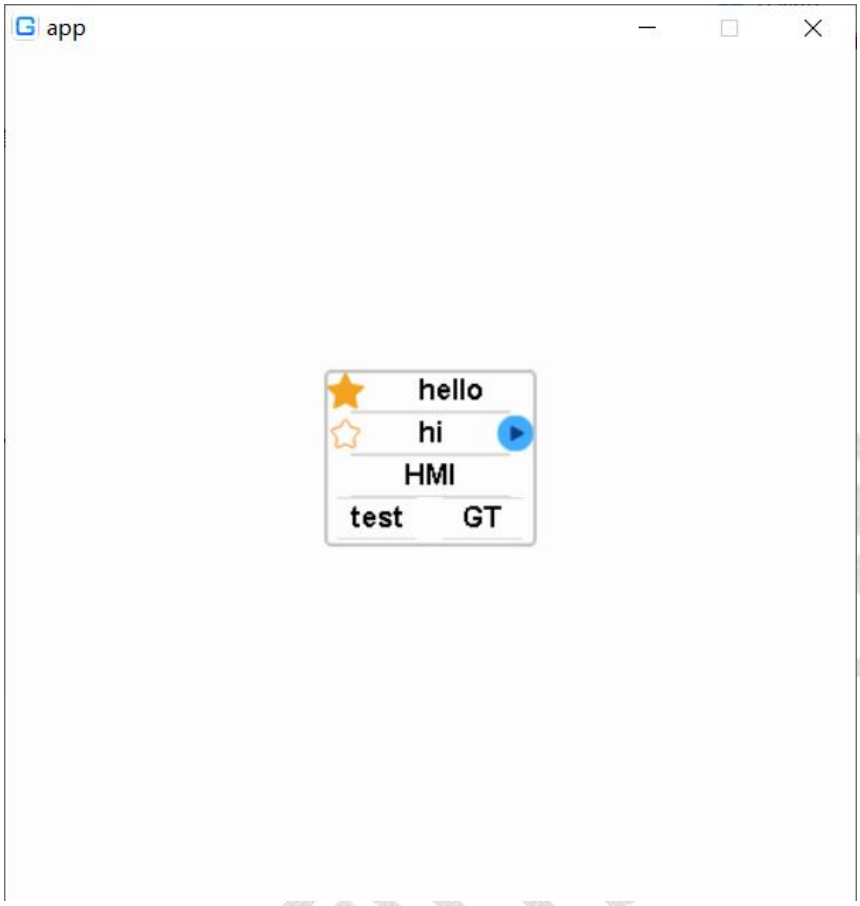
2.21 列表控件

在创建的屏幕控件中添加一个列表控件并设置列表控件属性。

```
/** listv1 */
//在 screen_home 屏幕控件中创建一个列表控件
listv1 = gt_listview_create(screen_home);
//设置列表的位置为（180,180）
gt_obj_set_pos(listv1, 180, 180);
//设置列表的尺寸为（宽度 120 高度 100）
gt_obj_set_size(listv1, 120, 100);
//设置列表字体颜色为 0x000000 黑色
gt_listview_set_font_color(listv1, gt_color_hex(0x000000));
//设置列表字体尺寸为 16 点
gt_listview_set_font_size(listv1, 16);
//设置列表对齐方式为居中对齐
gt_listview_set_font_align(listv1, GT_ALIGN_CENTER_MID);
//设置列表每项高度为 24
gt_listview_set_item_height(listv1, 24);
//设置列表边框颜色为 0xc7c7c7 灰色
gt_listview_set_border_color(listv1,gt_color_hex(0xc7c7c7));
//设置列表边框宽度为 2
```

```
gt_listview_set_border_width(listv1, 2);
//设置列表每项分割线
gt_listview_set_septal_line(listv1, 1);
//设置列表选中高亮模式
gt_listview_set_highlight_mode(listv1, 1);
//设置列表背景颜色为 0xffffff 白色
gt_listview_set_bg_color(listv1, gt_color_hex(0xffffff));
//设置列表每项背景颜色为 0xffffff 白色
gt_listview_set_item_bg_color(listv1, gt_color_hex(0xffffff));
//设置列表显示每项背景颜色
gt_listview_show_item_bg(listv1, true);
//设置列表每项内边距为 2
gt_listview_set_item_reduce(listv1, 2);
//设置列表每项边框半径为 10
gt_listview_set_item_radius(listv1, 10);
//设置列表中单图标样式比例为文本 80：图标 20
gt_listview_set_scale(listv1, 20, 80);
//设置列表中单图标样式比例为左图标：20 文本 60：右图标 20
gt_listview_set_scale_triple(listv1, 20, 60, 20);
//设置列表一行显示列数为单列
gt_listview_set_next_row_item_count(listv1, 1);
//设置列表第一行图标 1.png，文本为 hello
gt_listview_add_item_icon(listv1, ".:1.png", "hello");
//设置列表第二行左图标 2.png，文本为 hi，右图标为 3.png
gt_listview_add_item_icons(listv1, ".:2.png", "hi", ".:3.png");
//设置列表第三行文本为 HMI
gt_listview_add_item(listv1, "HMI");
//设置列表一行显示列数为双列
gt_listview_set_next_row_item_count(listv1, 2);
//设置列表第四行左文本为 test，右文本为 GT
gt_listview_add_item(listv1, "test");
gt_listview_add_item(listv1, "GT");
```

运行后的列表控件效果如下图所示：



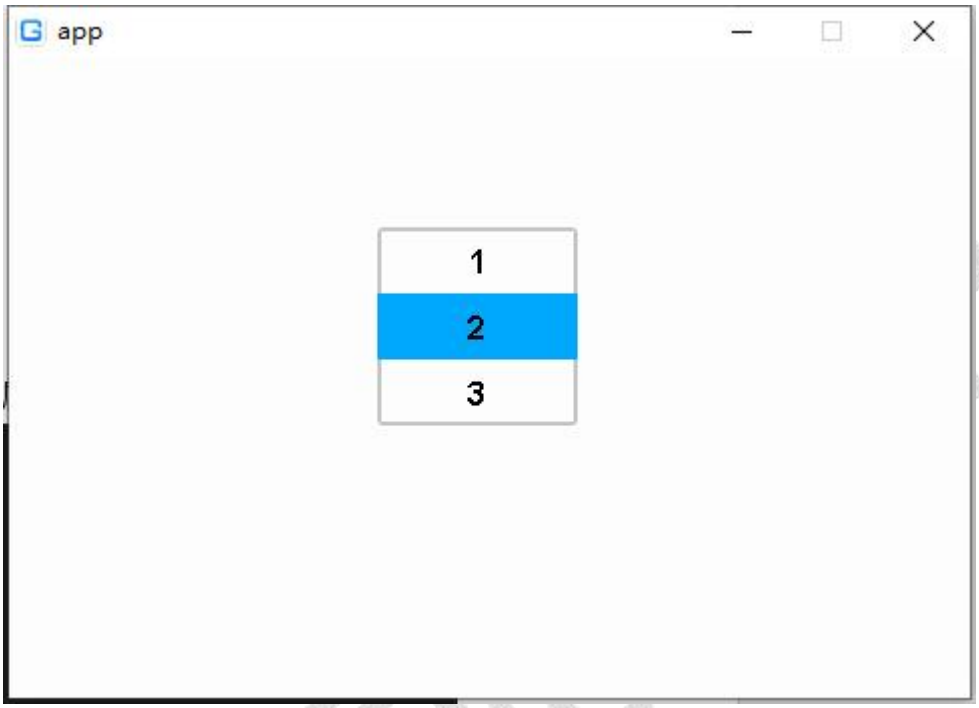
2.22 滚轮控件

在创建的屏幕控件中添加一个滚轮控件并设置滚轮控件属性。

```
//在 screen_home 屏幕控件中创建一个滚轮控件
roller1 = gt_roller_create(screen_home);
//设置矩形的位置为(184,85)
gt_obj_set_pos(roller1, 184, 85);
//设置滚轮的尺寸为(宽度 100 高度 100)
gt_obj_set_size(roller1, 100, 100);
//设置滚轮的字体颜色为黑色
gt_roller_set_font_color(roller1, gt_color_hex(0x000000));
 //设置字体的大小为 16
 gt_roller_set_font_size(roller1, 16);
//设置对齐方式为居中对齐
gt_roller_set_font_align(roller1, GT_ALIGN_CENTER_MID);
//设置滚轮显示元素数量
gt_roller_set_display_item_count(roller1, 3);
//设置滚轮选项
gt_roller_set_options(roller1, "1\n2\n3\n4\n5\n6\n7\n8", GT_ROLLER_MODE_NORMAL);
```

```
    //设置滚轮元素行间距
gt_roller_set_line_space(roller1, 17);
    //设置滚轮当前选中第几项
    gt_roller_set_selected(roller1, 1);
```

运行后的滚轮控件效果如下图所示：



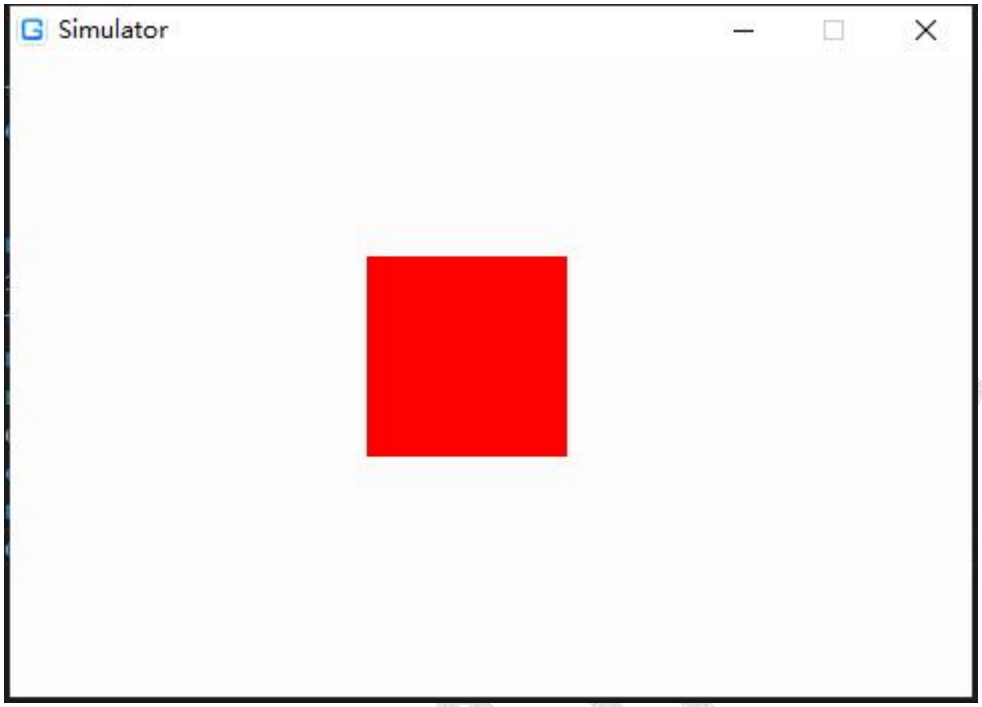
2.23 矩形控件

在创建的屏幕控件中添加一个矩形控件并设置矩形控件属性。

```
/*rect1*/
//在 screen_home 屏幕控件中创建一个矩形控件
rect1 = gt_rect_create(screen_home);
//设置矩形的位置为(178,100)
gt_obj_set_pos(rect1,178,100);
//设置矩形的尺寸为(宽度 100 高度 100)
gt_obj_set_size(rect1,100,100);
   //设置四端角的半径为 0 即直角
   gt_rect_set_radius(rect1,0);
//设置矩形的背景颜色为 0xff0000 红色
gt_rect_set_bg_color(rect1,gt_color_hex(0xff0000));
//设置矩形的边框颜色为 0xff0000 红色
gt_rect_set_color_border(rect1,gt_color_hex(0xff0000));
//设置矩形为填充
 gt_rect_set_fill(rect1,1);
```

```
//设置矩形边框宽度为 0 即无边框
gt_rect_set_border(rect1,0);
```

运行后的矩形控件效果如下图所示：

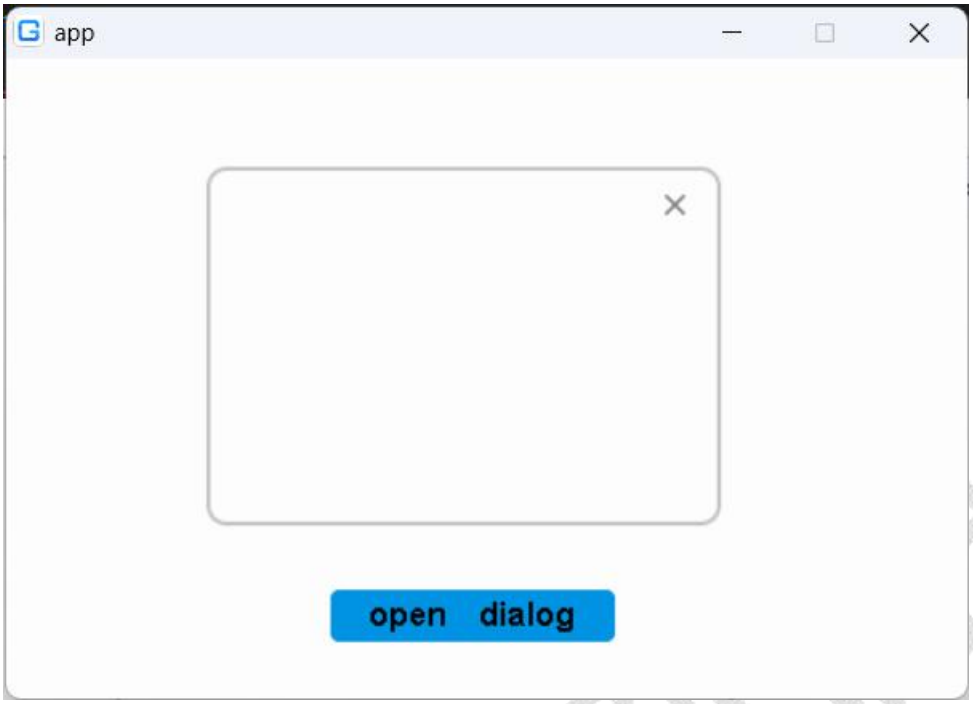


2.24 对话框控件

在创建的屏幕控件中添加一个对话框控件并设置对话框控件属性。

```
/** dialog1 */
//在 screen home 屏幕控件中创建一个对话框控件
dialog1 = gt_dialog_create(true);
//设置对话框的位置为（100,54）
gt_obj_set_pos(dialog1, 100, 54);
//设置对话框的尺寸为（宽度 257 高度 179）
gt_obj_set_size(dialog1, 257, 179);
//设置对话框的背景颜色为 0xffffff 白色
gt_dialog_set_bgcolor(dialog1, gt_color_hex(0xffffff));
//设置对话框的边框颜色为 0x000000 黑色
gt_dialog_set_border_color(dialog1, gt_color_hex(0x000000));
//设置对话框的边框宽度为 2
gt_dialog_set_border_width(dialog1, 2);
//设置对话框的边框圆角半径为 10
gt_dialog_set_border_radius(dialog1, 10);
```

运行后的点击“open dialog”按钮，显示对话框控件效果如下图所示：

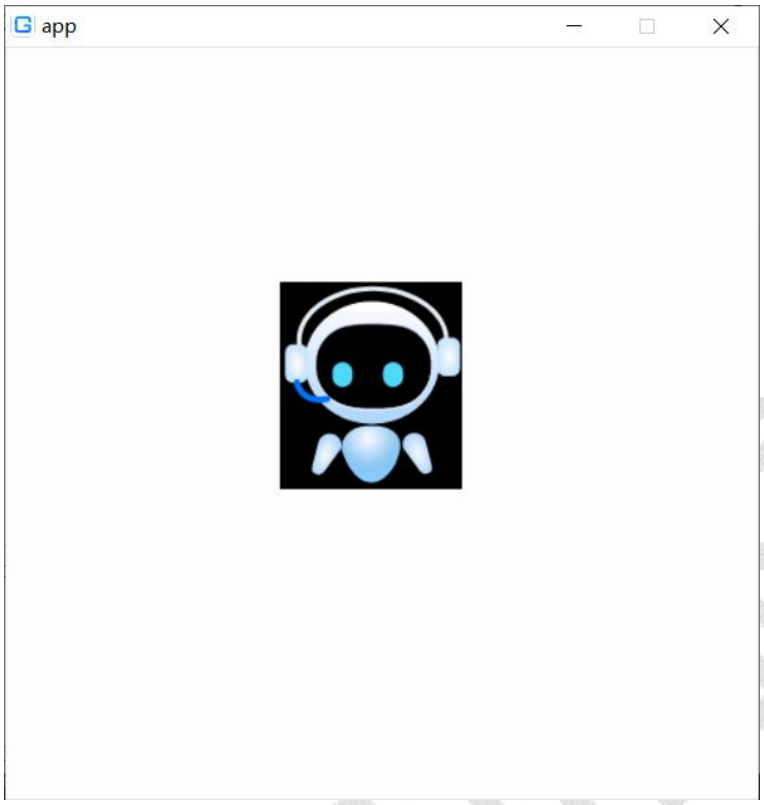


2.25 GIF 控件

在创建的屏幕控件中添加一个 GIF 控件并设置 GIF 控件属性。

```
/** gif1 */
//在 screen_home 屏幕控件中创建一个 GIF 控件
gif1 = gt_gif_create(screen_home);
//设置 GIF 控件的位置为（175，150）
gt_obj_set_pos(gif1, 175, 150);
//设置 GIF 控件的大小为（宽度 116，高度 132）
gt_obj_set_size(gif1, 116, 132);
//设置 GIF 文件为“img_116x132_1.gif”
gt_gif_set_src(gif1, ".:img_116x132_1.gif");
//播放 GIF 动图
gt_gif_play(gif1);
```

运行后显示 GIF 控件效果如下图所示：



2.26 状态栏控件

在创建的屏幕控件中添加一个状态栏控件并设置状态栏控件属性。

```
/** status_bar1 */
    //在 gt_ui_init UI 初始化时创建一个状态栏控件
    gt_status_bar_create(true);
    //设置状态栏控件的高度为 30
    gt_status_bar_set_height(30);
    //设置状态栏控件内元素宽度比例为（左：30，标题：40，右：30）
    gt_status_bar_set_scale(30, 40, 30);
    //设置状态栏控件显示状态栏背景
    gt_status_bar_show_bg(true);
    //设置状态栏控件背景颜色为 0x000000 黑色
    gt_status_bar_set_bg_color(gt_color_hex(0x000000));
    //设置状态栏控件的左边文本为“GT”，文本宽度为 30
    gt_status_bar_left_add_text("GT", 30);
    //设置状态栏控件的中间文本为“12:17”
    gt_status_bar_center_set_text("12:17");
    //设置状态栏控件的右边第一个图标为 1.png
    gt_status_bar_right_add_icon(".:1.png");
    //设置状态栏控件的右边第二个图标为 2.png
```

```
gt_status_bar_right_add_icon(".:2.png");
//设置状态栏控件的文本字体为矢量英文黑正
gt_status_bar_set_font_family_en(90);
//设置状态栏控件的文本字体大小为 16
gt_status_bar_set_font_size(16);
//设置状态栏控件的中字体使用 1Bit 灰度字体
gt_status_bar_set_font_gray(1);
//设置状态栏控件的文本颜色为 0xffffff 白色
gt_status_bar_set_font_color(gt_color_hex(0xffffffff));
//设置状态栏控件的中文字体粗细为 22
gt_status_bar_set_font_thick_cn(22);
//设置状态栏控件的英文字体粗细为 22
gt_status_bar_set_font_thick_en(22);
```

运行后显示状态栏控件效果如下图所示:

2.27 图表控件

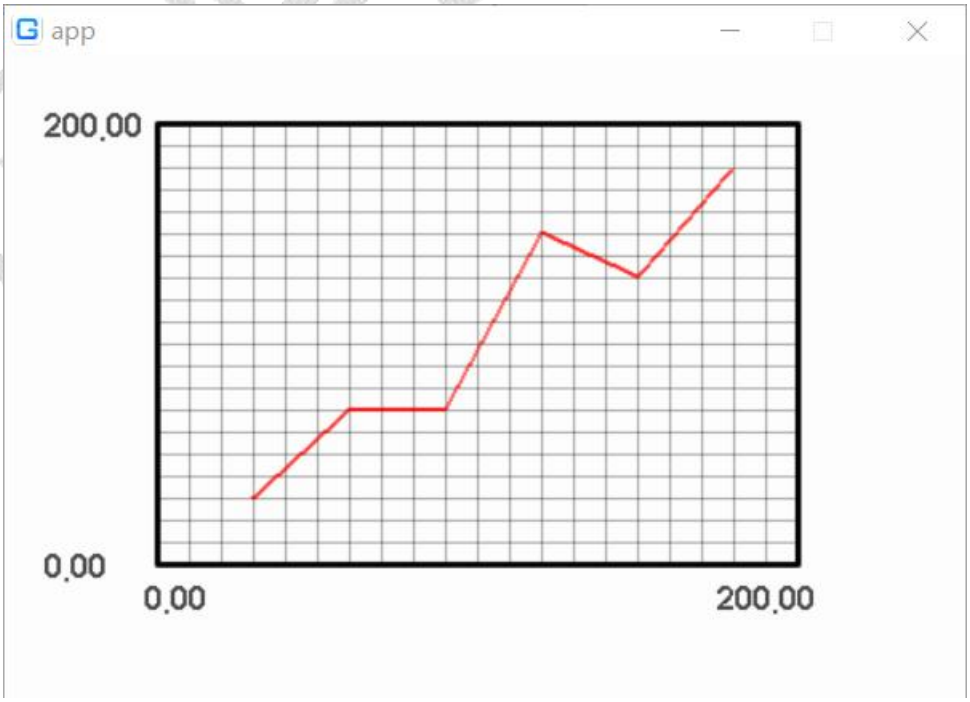
在创建的屏幕控件中添加一个图表控件并设置图表控件属性。

```
/** graph1 */
//在 screen_home 屏幕控件中创建一个图表控件
graph1 = gt_graphs_create(screen_home);
//设置图表控件的位置为（16，13）
gt_obj_set_pos(graph1, 16, 13);
//设置图表控件的尺寸为（宽 409，高 283）
gt_obj_set_size(graph1, 409, 283);
```

```
//初始化坐标集
gt_point_f_st graph1Rand0[]  = {
   { 30, 30 },
   { 60, 70 },
   { 90, 70 },
   { 120, 150 },
   { 150, 130 },
   { 180, 180 }
};
//初始化线图配置结构体
gt_axis_st graph1Axis = {
    //水平轴
    .hor = {
        .start = 0,
        .end = 200,
    },
    //垂直轴
    .ver = {
        .start = 0,
        .end = 200,
    },
    //标尺水平跨度为 10
    .hor_unit = 10,
    //标尺垂直跨度为 10
    .ver_unit = 10,
    //外边框设置
    .scale = {
        //外边框线宽为 3
        .width = 3,
        //外边框颜色为 0x000000
        .color = gt_color_hex(0x000000),
        //外边框透明度为 70
        .opa = GT_OPA_70,
    },
    //标尺设置
    .grid = {
        //标尺线宽为 1
         .width = 1,
        //标尺颜色为 0x000000
         .color = gt_color_hex(0x000000),
        //标尺透明度为 30
         .opa = GT_OPA_30,
    },
};
```

```
    //用线图配置结构体配置图表控件
    gt_graphs_set_axis(graph1, &graph1Axis);
    //设置图表控件关闭横坐标反转
    gt_graphs_set_anti_axis_hor_dir(graph1, false);
    //设置图表控件关闭纵坐标反转
    gt_graphs_set_anti_axis_ver_dir(graph1, false);
    //初始化线设置配置结构体
    gt_line_st graph1Line0 = {
        //线宽为 2
        .width = 2,
        //线颜色为 0xff0000
        .color = gt_color_hex(0xff0000),
        //线透明度为 80
        .opa = GT_OPA_80,
        //图表类型为折线图
        .type = GT_GRAPHS_TYPE_LINE,
    };
    //用线设置配置结构体配置图表控件
    gt_series_points_st * graph1Ser0 = gt_graphs_add_series_line(graph1,
&graph1Line0);
    //设置图表控件的坐标集为 graph1Rand0
    gt_graphs_series_add_point_list_by(graph1, 0, graph1Rand0, 6);
    //设置图表控件线图为关闭自适应最值
    gt_graphs_set_self_adaptive(graph1, false);
    //设置图表控件显示坐标系数
    gt_graphs_set_show_ruler(graph1, true);
```

运行后图表控件效果如下图所示，还可以将图表控件设置为散点图、曲线图和柱状图。



2.28 布局控件

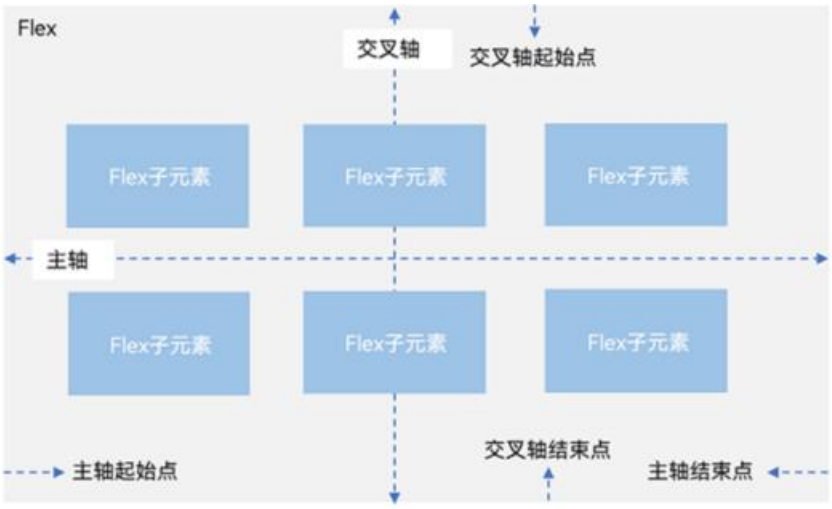
在创建的屏幕控件中添加一个自适应布局控件并设置自适应布局控件件属性。

```
/** layout1 init */
//在 screen_home 屏幕控件中创建一个自适应布局控件
gt_obj_st * layout1 = gt_obj_create(screen_1);
//初始化自适应布局的配置结构体
gt_obj_container_st layout1container = {
    //左右边距为 0
    .gap = 0,
    //溢出时不压缩
    .layout_type = GT_LAYOUT_TYPE_FLEX,
    //主轴方向为水平向右
    .flex_direction = GT_LAYOUT_FLEX_DIR_ROW,
    //主轴对齐方式为居中对齐
    .justify_content = GT_LAYOUT_JUSTIFY_CONTENT_CENTER,
    //交叉轴对齐方式为顶部对齐
    .align_items = GT_LAYOUT_ALIGN_ITEMS_START,
};
//用配置结构体配置自适应布局控件
gt_layout_init(layout1, &layout1container);
//设置自适应布局控件的位置为（148，85）
gt_obj_set_pos(layout1, 148, 85);
//设置自适应布局控件的尺寸为（宽 230，高 137）
gt_obj_set_size(layout1, 230, 137);

//需要添加到自适应布局控件中的其他控件，应修改其所属布局
input1 = gt_input_create(layout1);
```

主轴和交叉轴的基本概念：

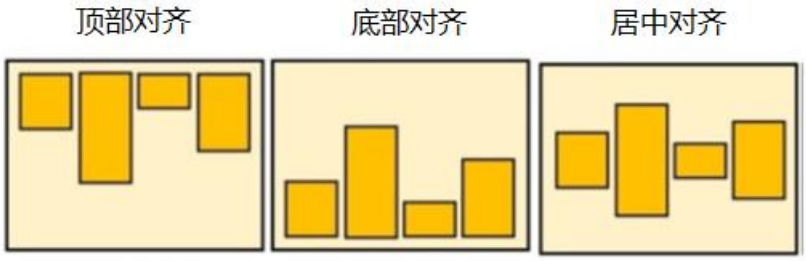
主轴：Flex 组件布局方向的轴线，子元素默认沿着主轴排列。主轴开始的位置称为主轴起始点，结束位置称为主轴结束点。

交叉轴：垂直于主轴方向的轴线，交叉轴开始的位置称为交叉轴起始点，结束位置称为交叉轴结束点。主轴为水平方向的 Flex 容器示意图如下图所示：



主轴方向为水平向右时不同的主轴对齐方式的结构示意图如下图所示：

主轴方向为水平向右时不同的交叉轴对齐方式的结构示意图如下图所示：



2.29 Media Player 播放器流控件

在创建的屏幕控件中添加一个 Media Player 播放器流控件并设置 Media Player 播放器流控件属性。

```
/** media_player1 */
//初始化播放图片格式结构体
gt_img_raw_st raw_st = {
    .buffer = raw_buf,          //图片颜色数据
    .opa = NULL,                //图片透明度
    .width = 138,               //图片宽度
    .height = 131,              //初始图片高度
};
```

```
    //在 screen_home 屏幕控件中创建一个 Media Player 播放器流控件
    media_player1 = gt_media_player_create(screen_home);
    //设置 Media Player 播放器流控件的位置为（29,39）
    gt_obj_set_pos(media_player1, 29, 39);
    //设置 Media Player 播放器流控件的大小为（宽 240，高 240）
    gt_obj_set_size(media_player1, 240, 240);
    //设置 Media Player 播放器流控件的总时长为 MEDIA_NUM_MAX
    gt_media_player_set_total_time(media_player1, MEDIA_NUM_MAX);
    //设置 Media Player 播放器流控件的当前进度为 0
    gt_media_player_set_current_time(media_player1, 0);

    //设置 Media Player 播放器流控件的播放图片格式
    gt_media_player_set_raw(media_player1, &raw_st);
    //设置 Media Player 播放器流控件的播放回调函数为 play_cb
    gt_media_player_set_play_cb(media_player1, play_cb, NULL);
    //设置 Media Player 播放器流控件的暂停回调函数为 stop_cb
    gt_media_player_set_stop_cb(media_player1, stop_cb, NULL);
    //设置 Media Player 播放器流控件通过滑动条来设置当前时间
    gt_media_player_set_slider_change_cb(media_player1, value_change_cb,
NULL);
    //设置播放器控件的圆角半径为 40
    gt_obj_set_radius(media_player1, 40);
    //设置 Media Player 播放器流控件开始播放
    gt_anim_init_start();
```

还需要定义一个动画播放函数。

```
static void gt_anim_init_start(void)
{
    //新建一个动画对象
    gt_anim_st anim_del;
    //初始化动画对象
    gt_anim_init(&anim_del);
    //设置动画的时间间隔为 1000ms
    gt_anim_set_time(&anim_del, 1000);
    //设置动画播放的次数为无限循环
    gt_anim_set_repeat_count(&anim_del, GT_ANIM_REPEAT_INFINITE);
    //设置动画播放的对象
    gt_anim_set_target(&anim_del, media_player1);
    //设置动画每一帧播放的回调函数
    gt_anim_set_exec_cb(&anim_del, _scr_anim_del_ready_cb);
    //开始播放
    anim_del_p = gt_anim_start(&anim_del);
    if (NULL == anim_del_p) {
        GT_CHECK_PRINT(anim_del_p = gt_anim_start(&anim_del));
    }
```

```
}

//每一帧的播放回调函数
static void _scr_anim_del_ready_cb(struct gt_anim_s * anim)
{
    //读取一帧图片的数据
    bsp_qspi_flash_read(media_current*IMG_BUFF_SIZE , IMG_BUFF_SIZE ,
raw_buf);
    //设置 Media Player 播放器流控件的当前进度
    gt_media_player_set_current_time(media_player1, media_current);
    //显示新的一帧内容
    gt_disp_invalid_area(media_player1);
    //帧数加一
    ++media_current;
    //到最后一帧后，回到第一帧
    if(media_current == MEDIA_NUM_MAX)
    {
        media_current = 0;
    }
}
```

2.30 下拉框控件

```
    //在 screen_home 控件中创建一个下拉框控件
    select1 = gt_select_create(screen_4);
    //设置下拉框控件 x 坐标和 y 坐标
    gt_obj_set_pos(select1, 139, 107);
    //设置下拉框控件的宽度和高度
    gt_obj_set_size(select1, 153, 41);
    //设置下拉框中文字的的颜色
    gt_select_set_font_color(select1, gt_color_hex(0x000000));
    //设置下拉框中文字的大小
    gt_select_set_font_size(select1, 24);
    //设置中文字体
    gt_select_set_font_family_cn(select1, 5);
    //设置文字居中对齐
    gt_select_set_font_align(select1, GT_ALIGN_CENTER_MID);
    //设置每条选项的高度
    gt_select_set_option_height(select1, 30);
    //添加下拉框的选项
    gt_select_add_option(select1, "中文");
    gt_select_add_option(select1, "英文");
    gt_select_add_option(select1, "阿拉伯文");
```

3.控件事件

事件是对各种控件由于应用程序内部或者外部产生的事情或动作处理的通称，在高通 GT-HMI 中常见的事件为加载屏幕，背景、字体颜色变化，位置、大小变化等。事件通常使用 gt_obj_add_event_cb 函数接口添加，以下为事件举例说明。

3.1 加载屏幕

加载屏幕事件为屏幕控件特有事件，主要用于屏幕页面间跳转使用。所有页面都由页栈进行管理。

首先，通过枚举变量给每个页面注册页面 ID；

```
typedef enum {
        GT_ID_SCREEN_HOME = 0,
        GT_ID_SCREEN_1 = 1,
        GT_ID_SCREEN_2 = 2,
        GT_ID_SCREEN_3 = 3,
        GT_ID_SCREEN_4 = 4,
} gt_scr_id_et;
```

其次，创建页面对象；

```
/** screen_1 */
gt_obj_st * screen_1 = NULL;

gt_obj_st * gt_init_screen_1(void)
{
    //创建页面 screen_1
    screen_1 = gt_obj_create(NULL);
    //设置页面背景为 0xffffff 白色
    gt_screen_set_bgcolor(screen_1, gt_color_hex(0xFFFFFF));

    /*****************************************/
    //添加控件到页面 screen_1
    /*****************************************/
```

```
    //返回页面 screen_1 对象
    return screen_1;
}
```

然后，将页面 ID 与对应的页面绑定；

```
static const gt_scr_list_st gt_scr_list[] = {
    {GT_ID_SCREEN_HOME, gt_init_screen_home},
    {GT_ID_SCREEN_1, gt_init_screen_1},
    {GT_ID_SCREEN_2, gt_init_screen_2},
    {GT_ID_SCREEN_3, gt_init_screen_3},
    {GT_ID_SCREEN_4, gt_init_screen_4},
};
```

最后，初始化页栈并设置主页。

```
void gt_ui_init(void) {
    //初始化页栈，并设置页栈深度为 20
    gt_scr_stack_register_id_list(gt_scr_list, sizeof(gt_scr_list) /
sizeof(gt_scr_list[0]), 20);
    //设置 screen_home 为主页，并设置为主页可释放
    gt_scr_stack_set_home_scr_id(GT_ID_SCREEN_HOME, false);
    //显示主页，并将主页的页面 ID 加入栈顶
    gt_disp_stack_load_scr(GT_ID_SCREEN_HOME);
}
```

以下示例为在 screen_home 中的 btn1 按钮添加加载屏幕事件实现点击按下 btn1 控件时跳转到 screen_1 界面。

```
static void btn1_0_cb(gt_event_st * e) {
    //无过渡动画显示 screen_1 页面，并将 screen_1 页面的页面 ID 加入到栈顶
    gt_disp_stack_load_scr_anim(GT_ID_SCREEN_1, GT_SCR_ANIM_TYPE_NONE,
500, 0, true);
}

  //为 btn1 按钮控件添加事件 触发条件为 PRESSED 点击按下 按下时回调处理 action1_cb
函数内容 触发时不带用户数据 NULL
  gt_obj_add_event_cb(btn1, btn1_0_cb, GT_EVENT_TYPE_INPUT_PRESSED,NU
LL);
```

加载屏幕时可以为其添加过渡动画以下为 screen_1 屏幕控件添加延时 0ms 执行，类型为向左移动，持续时间为 500ms 的过渡动画，同时将页面 ID 加入到栈顶。

```
   gt_disp_stack_load_scr_anim(GT_ID_SCREEN_1, GT_SCR_ANIM_TYPE_MOVE_LEF
T, 500, 0, true);
```

返回页面时可选择返回层数，当选择返回层数超过页栈当前页栈深度时，便默认返回主页。以下示例为返回上一页。

```
    gt_disp_stack_go_back(1);
```

3.2 移动动画

移动动画事件主要用于实现平滑移动控件，以下为 btn1 按钮添加移动动画事件,当滑动 btn1 控件时将触发 btn2 按钮控件(原坐标为 100,,10)平滑地移动到(10,10)的位置。

```
static void btn1_0_cb (gt_event_st * e)
{
    static gt_anim_param_st param;
  //初始化动画基础值
  gt_anim_param_init(&param);
  //设置动画延时 0ms 后执行 持续时间 100ms
  gt_anim_param_set_time(&param, 0, 100);
  //设置移动终点为(10,10)
  gt_anim_param_set_dst_point(&param, 10, 10);
  //设置动画移动类型
  gt_anim_param_set_path_type(&param, GT_ANIM_PATH_TYPE_OVERSHOOT);
  //为 btn2 添加该动画
  gt_anim_pos_move(btn2, &param);
}

  //为 btn1 按钮控件添加事件 触发条件为 SCROLL 滑动 滑动时回调处理 btn1_0_cb 函数内容 触发时不带用户数据 NULL
  gt_obj_add_event_cb(btn1, btn1_0_cb, GT_EVENT_TYPE_INPUT_SCROLL, NULL);
```

3.3 背景颜色变化

背景颜色变化主要用于实现改变控件本身的背景颜色。事件添加代码与上述重复仅展示回调函数 btn1_0_cb 差异如下所示。

```
static void btn1_0_cb(gt_event_st * e)
{
//改变 btn2 背景颜色为 0xff0000 红色
gt_btn_set_color_background(btn2,gt_color_hex(0xff0000));
}
```

3.4 手势返回

手势返回主要用于实现页面切换，以下为 screen_home 屏幕添加手势返回事件，当在 screen_home 屏幕控件中右滑时将返回上一页。

```
static void screen_home_0_cb(gt_event_st * e) {
    //显示上一页，并将当前页面 ID 从页栈中出栈
    gt_disp_stack_go_back(1);
}
    //为 screen_home 屏幕控件添加事件 触发条件为 GT_EVENT_TYPE_INPUT_HOME_GES
```

```
TURE_RIGHT 手势右滑 滑动时回调处理 screen_home_0_cb 函数内容 触发时不带用户数据
NULL
    gt_obj_add_event_cb(screen_home, screen_home_0_cb, GT_EVENT_TYPE_INP
UT_HOME_GESTURE_RIGHT, NULL);
```

3.5 控件坐标独立变化事件

```
//设置文本区控件的 x 轴和 y 轴同时步进移动事件
static GT_ATTRIBUTE_RAM_TEXT void txt1_0_cb(gt_event_st * e) {
    gt_obj_set_pos_step(txt1, 10, 0);
}

//设置文本区控件的 x 轴方向过渡移动事件，平移到 x=10 坐标处
static GT_ATTRIBUTE_RAM_TEXT void txt2_0_cb(gt_event_st * e) {
    gt_obj_set_x_anim(txt2, 10);
}

//设置列表控件步进移动
static GT_ATTRIBUTE_RAM_TEXT void listv3_0_cb(gt_event_st * e) {
    gt_obj_set_x_step(listv3, 10);
}

//设置列表控件过渡移动
static GT_ATTRIBUTE_RAM_TEXT void listv1_0_cb(gt_event_st * e) {
    gt_obj_set_pos_anim(listv1, 200, 0);
}
```

3.6 控件宽度或高度独立变化事件

```
//设置文本区控件宽度，步进效果增加控件宽度
static GT_ATTRIBUTE_RAM_TEXT void txt5_0_cb(gt_event_st * e) {
    gt_obj_set_w_step(txt5, 10);
}
//设置文本区控件高度，步进效果增加控件高度
static GT_ATTRIBUTE_RAM_TEXT void txt5_0_cb(gt_event_st * e) {
    gt_obj_set_h_step(txt5, 10);
}
```

3.7 控件圆角变化事件

```
//设置文本区控件圆角变化事件
static GT_ATTRIBUTE_RAM_TEXT void txt3_0_cb(gt_event_st * e) {
```

```
    gt_obj_set_radius(txt3, 40);
}
```

3.8 控件隐藏事件

```
//按钮控件控制文本区控件隐藏和显示的效果
static GT_ATTRIBUTE_RAM_TEXT void btn2_0_cb(gt_event_st * e) {
    bool status = gt_obj_get_visible(txt4);
    gt_obj_set_visible(txt4, !status);

}
//设置文本区控件隐藏事件，点击控件触发隐藏效果
static GT_ATTRIBUTE_RAM_TEXT void txt4_0_cb(gt_event_st * e) {
    bool status = gt_obj_get_visible(txt4);
    gt_obj_set_visible(txt4, !status);
}
//按钮控制列表控件的的隐藏效果
static GT_ATTRIBUTE_RAM_TEXT void btn2_0_cb(gt_event_st * e) {
    bool status = gt_obj_get_visible(listv2);
    gt_obj_set_visible(listv2, !status);
}
```

3.9 控件透明度变化事件

```
//设置文本区控件的透明度变化
static GT_ATTRIBUTE_RAM_TEXT void txt1_0_cb(gt_event_st * e) {
    gt_obj_set_opa(txt1, GT_OPA_20);
}
```

3.7 其他事件

其他诸如字体颜色变化、位置变化、大小变化、透明度变化、文本变化、隐藏等事件与背景颜色变化事件类似，皆为在回调函数 btn1_0_cb 中调用控件函数接口实现，本文档不再缀叙。

4.高通字库

在工程的 gt_gui_driver.h 中，可以看到所有需要使用的字体类型定义在 font_option_et 枚举。字体类型分为灰度字库（推荐）、点阵字库和矢量字库等。

需要注意的是，在工程中使用字库时需要根据实际使用的字库来修改 gt_conf.h 中的宏 GT_FONT_FAMILY_OLD_ENABLE，不然在编译时会出现报错。这个宏为 1 时使用旧版字库体系，为 0 时使用新版字库体系。

```
#ifndef GT_USE_WIDGET_LAYOUT
    /**
     * @brief Set the widget layout function:
     *      0: no layout function, improve performance
     *      1: layout function
     * [Default: 1]
     */
    #define GT_USE_WIDGET_LAYOUT        01
#endif

#ifndef GT_FONT_FAMILY_OLD_ENABLE
    /**
     * @brief use old font family
     *
     */
    #define GT_FONT_FAMILY_OLD_ENABLE  01
#endif

/* default font style and size */
#define GT_CFG_DEFAULT_FONT_FAMILY          (0xFFFF)
#define GT_CFG_DEFAULT_FONT_SIZE            (16)

#if (defined(GT_FONT_FAMILY_OLD_ENABLE) && (GT_FONT_FAMILY_OLD_ENABLE == 1))
#define GT_CFG_DEFAULT_FONT_FAMILY_CN       0xFFFE
#define GT_CFG_DEFAULT_FONT_FAMILY_EN       GT_CFG_DEFAULT_FONT_FAMILY
#define GT_CFG_DEFAULT_FONT_FAMILY_FL       GT_CFG_DEFAULT_FONT_FAMILY
#define GT_CFG_DEFAULT_FONT_FAMILY_NUMB     GT_CFG_DEFAULT_FONT_FAMILY
#endif


#ifndef GT_USE_MODE_SRC
    #define GT_USE_MODE_SRC             01
#endif
```

4.1 矢量字库

矢量字库是高通专为嵌入式产品打造的精品矢量字库。具有丰富多样的字体，占用资源小（RAM 5-8K），支持文字大小无极缩放，支持边缘平滑的灰度显示，语言覆盖 180 多个国家，广泛应用在显示质量要求较高的彩屏和高分辨率电子产品上。

4.2 灰度字库

灰度字库融合点阵与矢量字库之精髓，无需 MCU 渲染，以超低资源消耗，呈现媲美手机与 PC 的细腻字体显示。包含中文的 GB2312 和 GBK 编码的 16 点黑体、20 点黑体、24 点黑体和 16 点正楷。

4.3 点阵字库

点阵字库以像素点阵的形式存储字符信息，相比于矢量字库，点阵字库通常占用更少的存储空间，适合在资源受限的嵌入式系统中使用。

由于点阵字库以像素点阵形式存储字符信息，渲染速度较快。在一些对速度要求较高的场景下，点阵字库能够提供更快的显示效果。点阵字库可以比较灵活地适应不同的显示设备和分辨率，因为它们是以像素为单位存储的。这使得点阵字库在不同的显示环境下都能够提供较好的显示效果。

5.支持的第三方库

5.1 PNG 解码库

PNG 解码器可以让我们在 GT-HMI Engine 中使用 PNG 图像。该功能使用 lodepng 库实现。

在 gt_conf_widgets.h 文件中，如果配置使能了 GT_CFG_ENABLE_IMG 功能，并且在 gt_conf.h 文件中配置使能了 GT_USE_SJPG 功能，那么 GT-HMI Engine 将会把 PNG 的图像解码软解库注册到 PNG 默认配置中，之后 PNG 格式的文件就可以当成图像源进行调用。PNG 解码器会对整个 PNG 图像解码，所以在解码过程中将额外增加 RAM 的占用，大小为：图像宽度 x 图像高度 x 4 字节。

5.2 JPEG 解码库

JPEG 解码器可以让我们在 GT-HMI Engine 中使用 JPEG 图像。该功能使用 tjgpd 库实现。

在 gt_conf_widgets.h 文件中，如果配置使能了 GT_CFG_ENABLE_IMG 功能，并且在 gt_conf.h 文件中配置使能了 GT_USE_PNG 功能，那么 GT-HMI Engine 将会把 JPEG 的图像解码软解库注册到 JPEG 默认配置中，之后 JPEG 格式的文件就可以当成图像源进行调用。

5.3 GIF 解码库

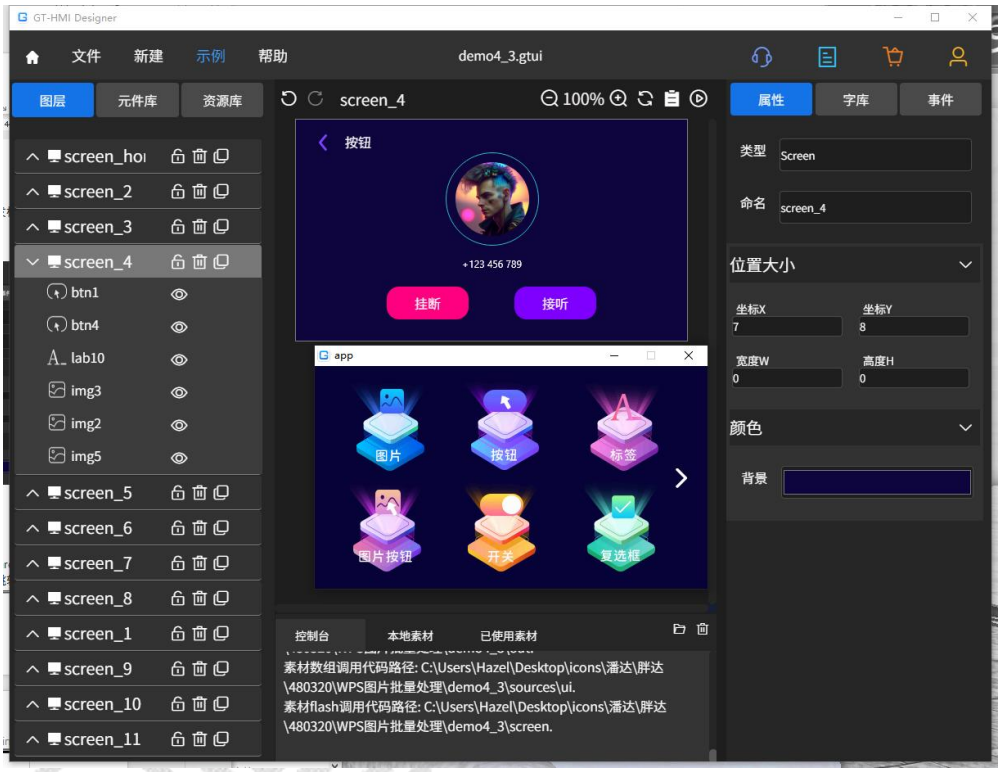
GIF 解码器可以让我们在 GT-HMI Engine 中使用 GIF 图像。该功能使用 gifdec 库实现。

在 gt_conf_widgets.h 文件中，如果配置使能了 GT_CFG_ENABLE_IMG 功能，并且在 gt_conf.h 文件中配置使能了 GT_USE_GIF 功能，那么 GT-HMI Engine 将会把 GIF 的图像解码软解库注册到 GIF 默认配置中，之后 GIF 格式的文件就可以当成图像源进行调用。

6.GT-HMI Designer 程序移植说明

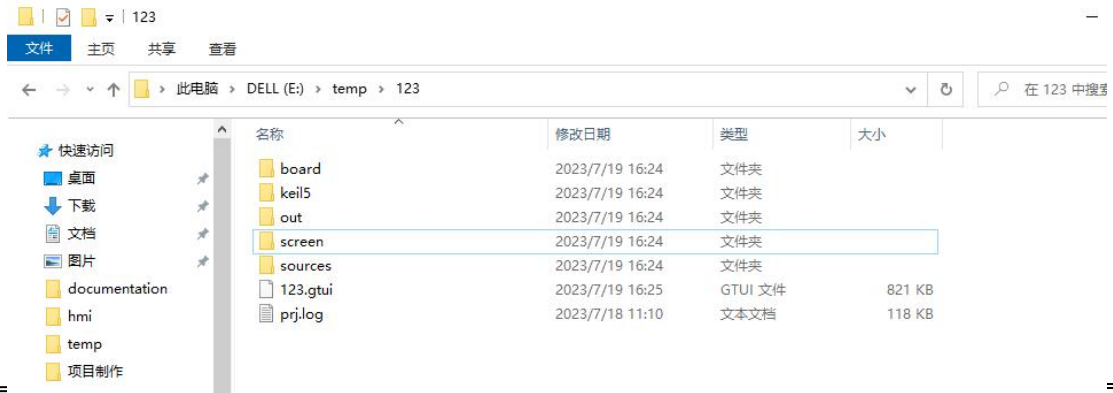
6.1UI 设计完成

当使用 GT-HMI Designer 完成对 UI 交互的开发之后。需要点击编译环境设置设定开发板卡和字库配置，然后点击  仿真运行一次。以确保编译生成需要的文件。

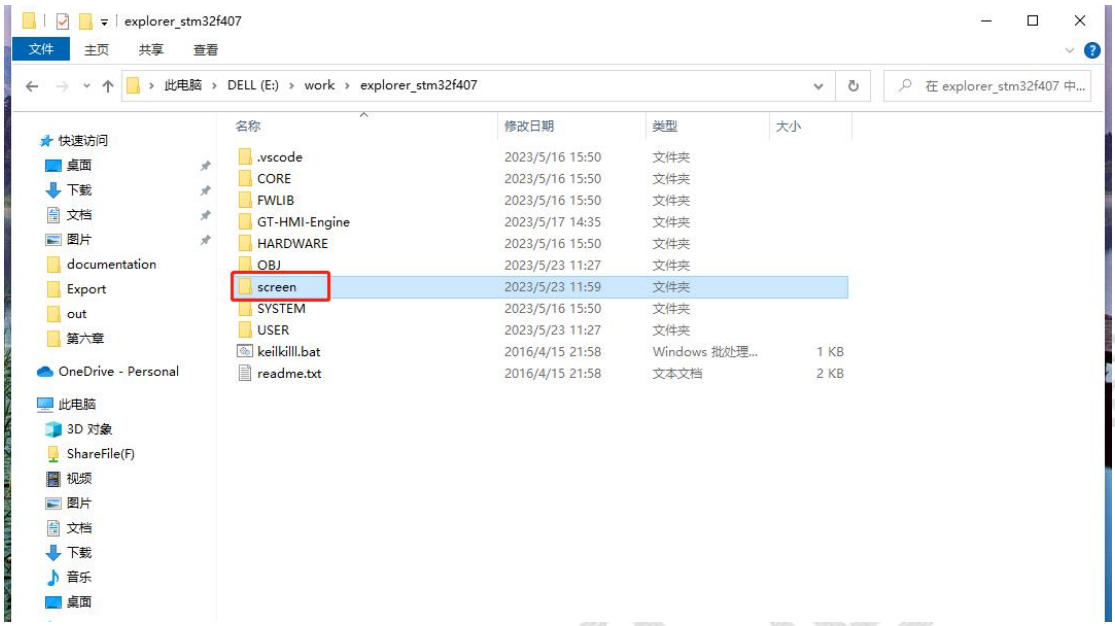


6.2 复制生成的 UI 文件

找到工程项目所在路径，假如你使用的是 GT-HMI 模块，则你可以直接使用 keil5/board 文件夹中 keil 工程生成程序文件下载使用，无需再进行以下的代码移植操作步骤，直接跳转到 6.2 将素材导入到板子中即可，更加方便快捷。

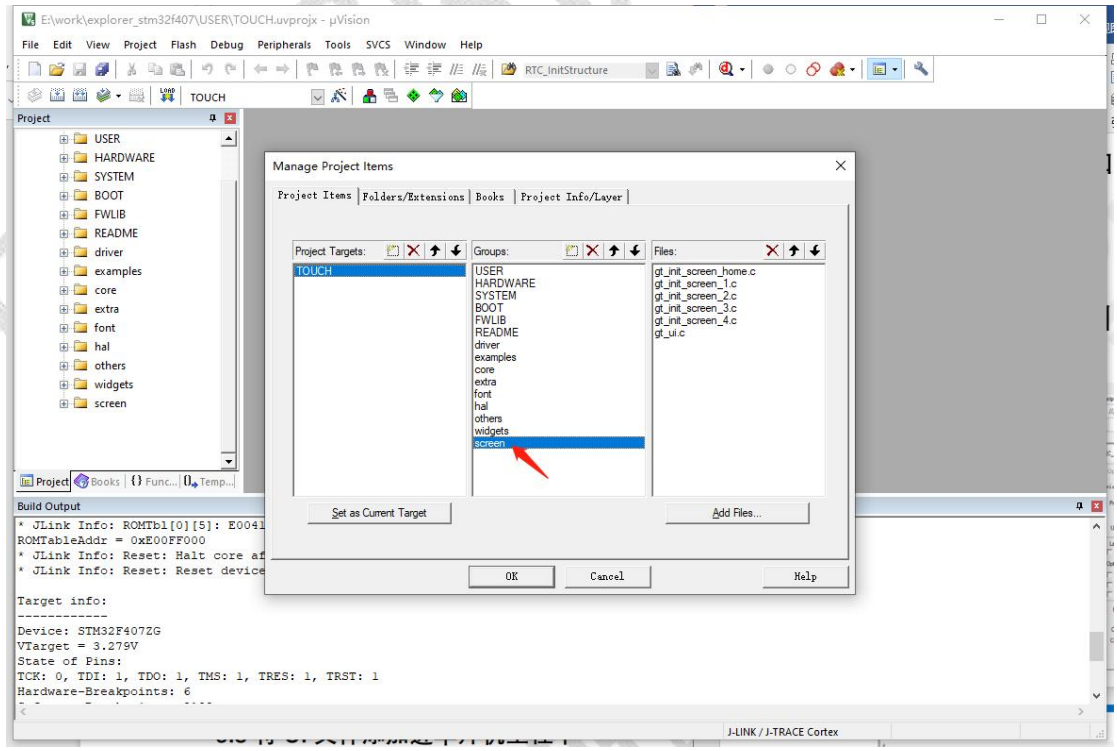


我们来说明使用非 GT-HMI 模块的移植过程，复制当前路径下的 screen 文件夹。将此文件夹直接复制到已经移植好 GT-HMI Engine 的工程文件夹下。如 stm32f407 的工程，移植过程可以参考 P 6.1 移植下位机平台流程

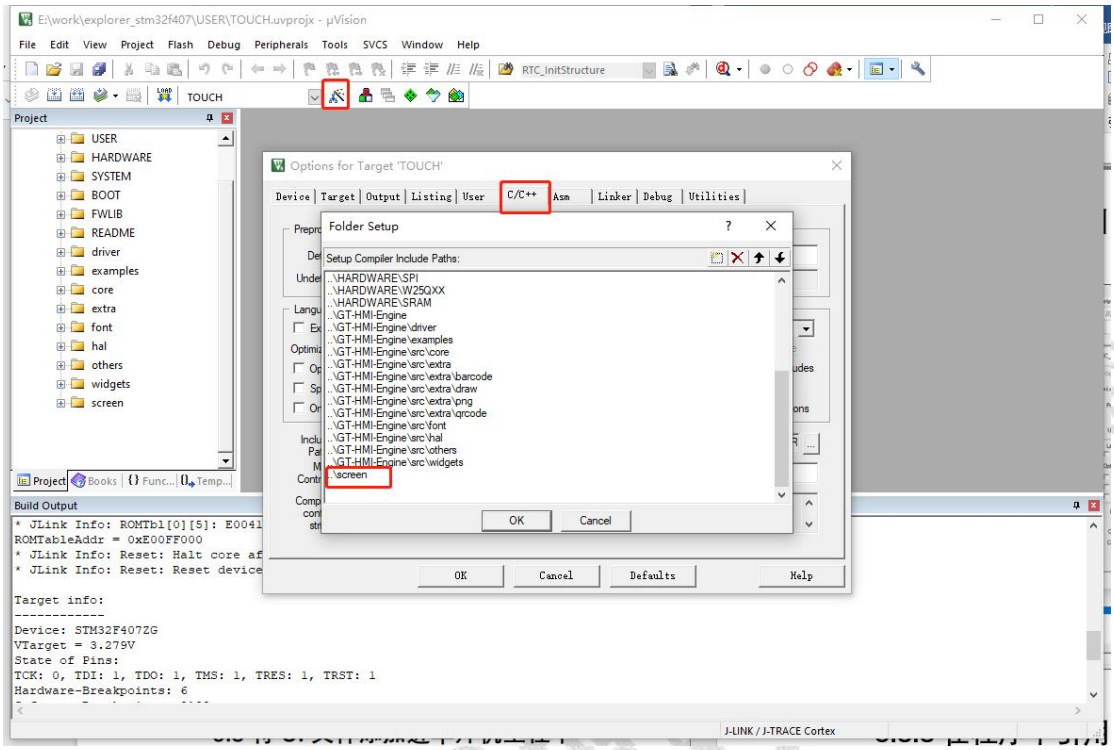


6.3 将 UI 文件添加进单片机工程中

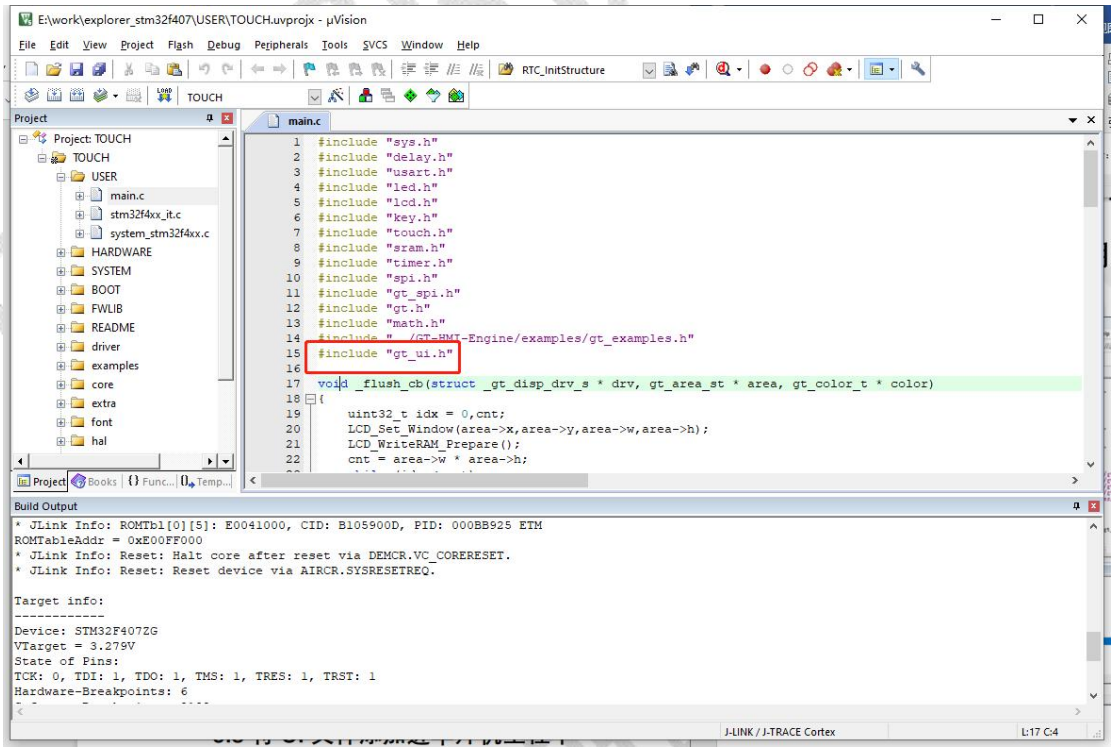
6.3.1 在工程中添加 screen 文件夹及内部包含的文件



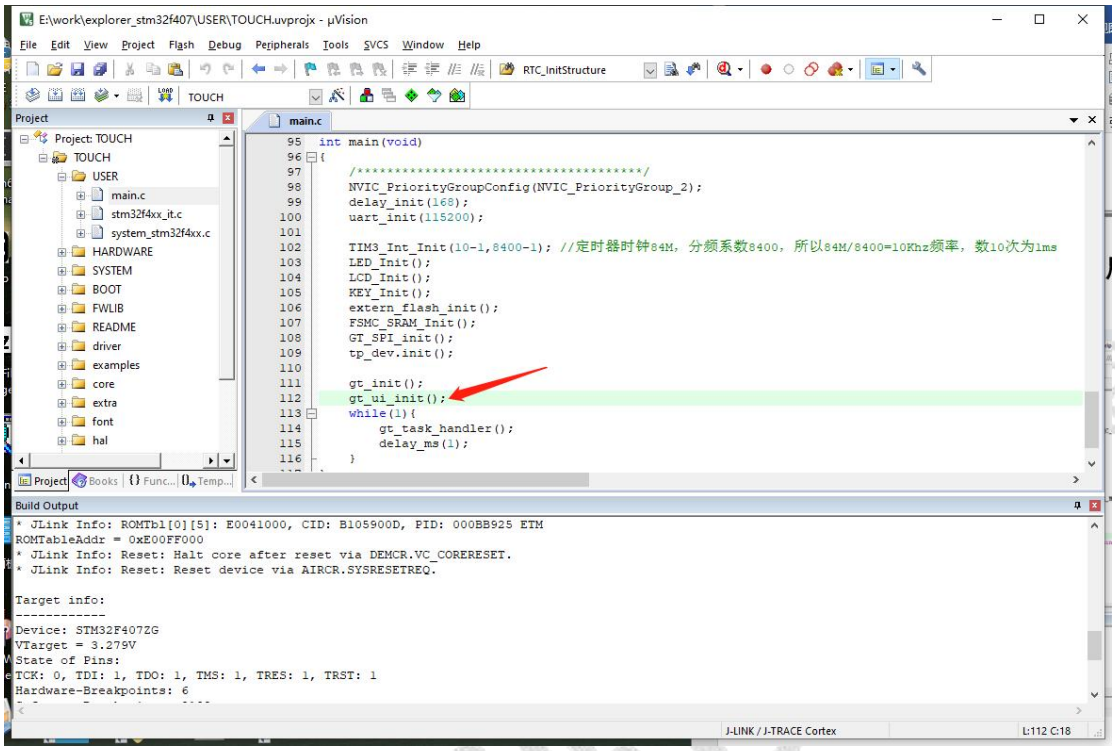
6.3.2 添加头文件引用路径



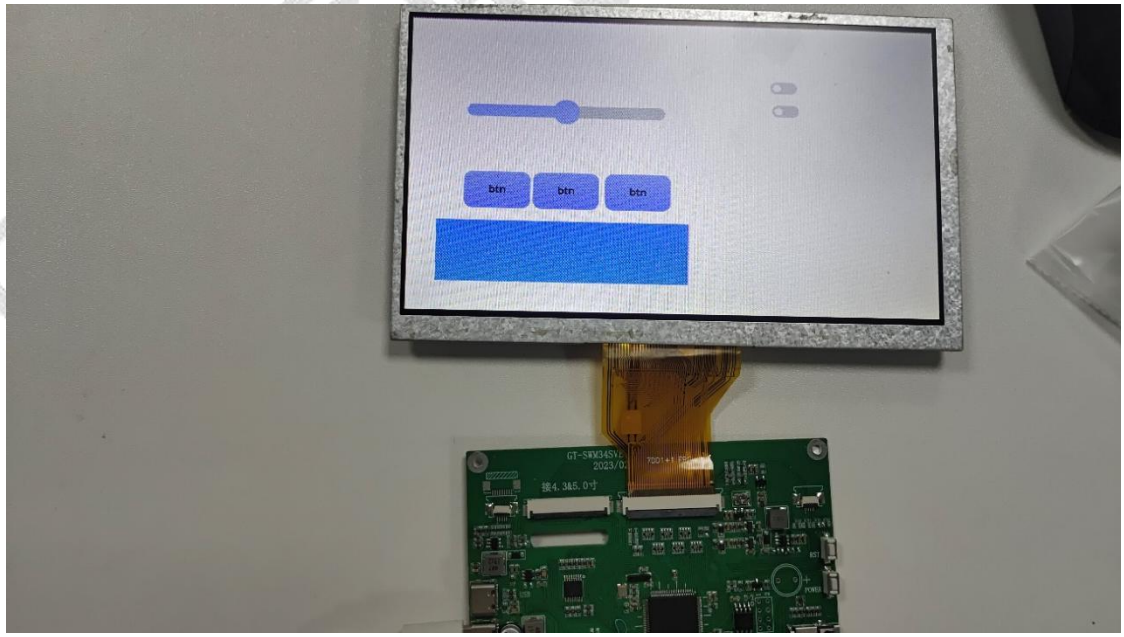
6.3.3 在程序中引用 gt_ui.h 这个头文件



6.3.4 在 gt_init()之后调用 gt_ui_init()函数



6.3.5 移植完成

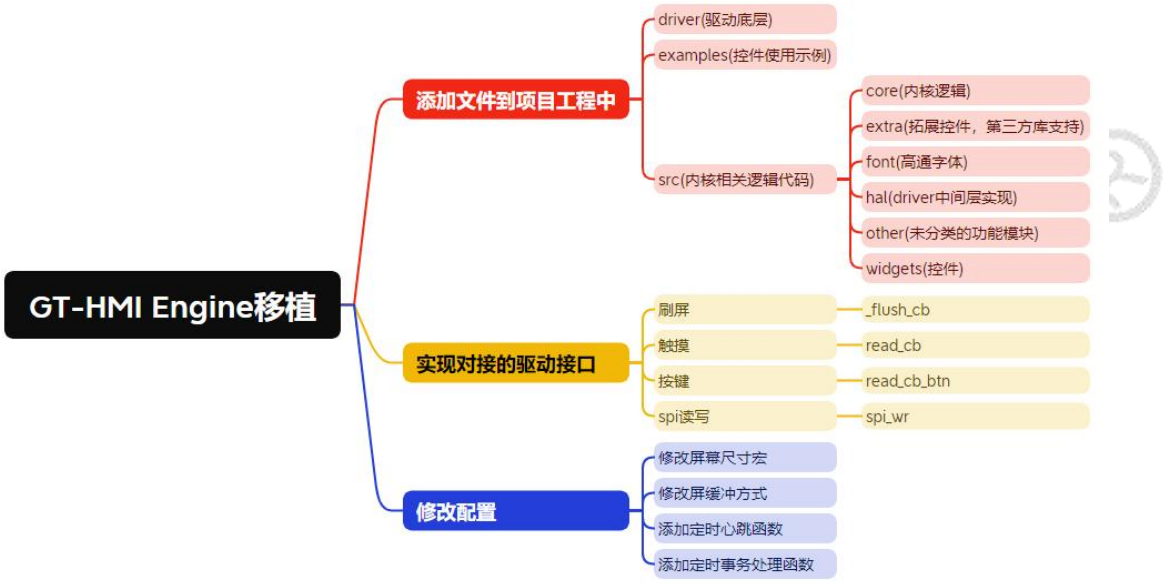


注意以上移植仅移植各页面的调用文件，若使用图片和字库等需额外进行导入素材操作，请参考 P6.2,P6.3 将素材和素材调用文件导入工程。后续素材不变动，仅 UI 部分调整的情况下，只需将 screen 文件夹内的文件更换为上位机新编译出来的文件即可。

7.GT-HMI Engine 移植操作流程

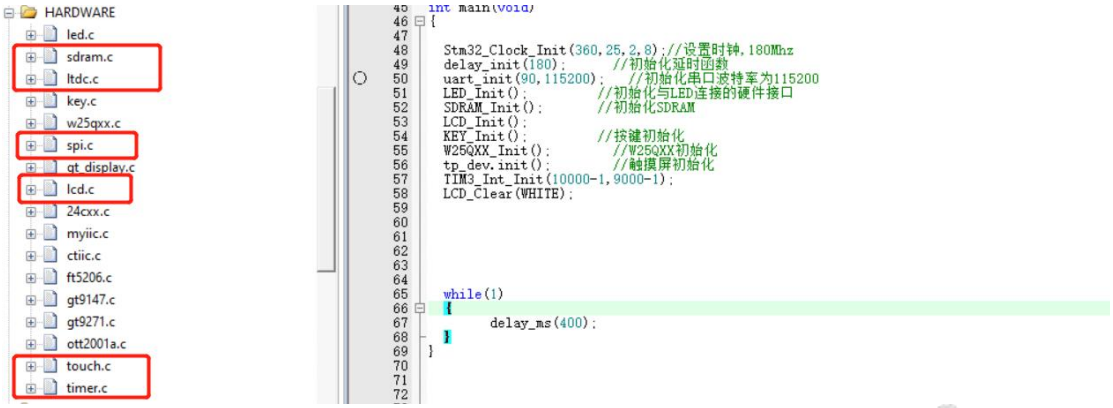
7.1GT-HMI Engine 移植平台流程

移植 GT_GUI 下位机流程导图如下所示(本手册以正点原子 STM32F429 阿波罗开发板示例)



7.1.1 准备工作

首先需要一份已经实现对应外设驱动（显示屏驱动,触摸驱动,定时器驱动,sdram 或 sram 驱动,如果需要使用到字库内容或 flash 文件系统，则还需要实现对应的 spi 驱动以读取 flash）并确保触摸屏例程使用正常的工程。这里准备的是主控为 STM32F429 的工程。



实现对应驱动的示例

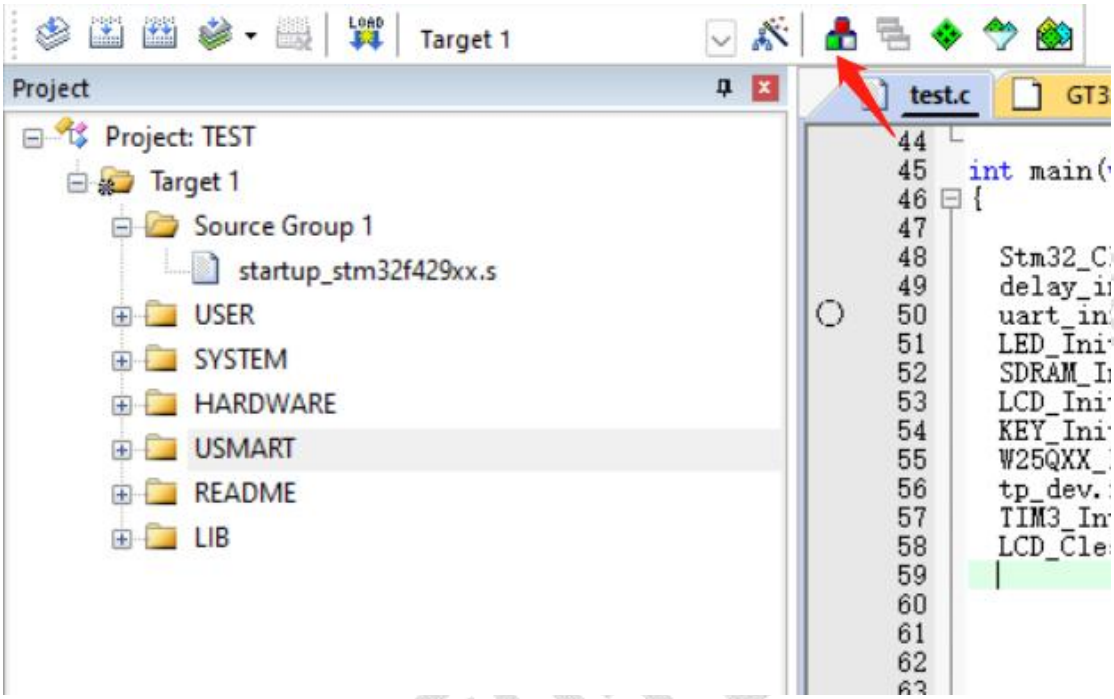
然后还需要准备 GUI 下位机的 GT-HMI-Engine 平台引擎包 (官网下载或仓库克隆，我这里将文件包改名为 gui)

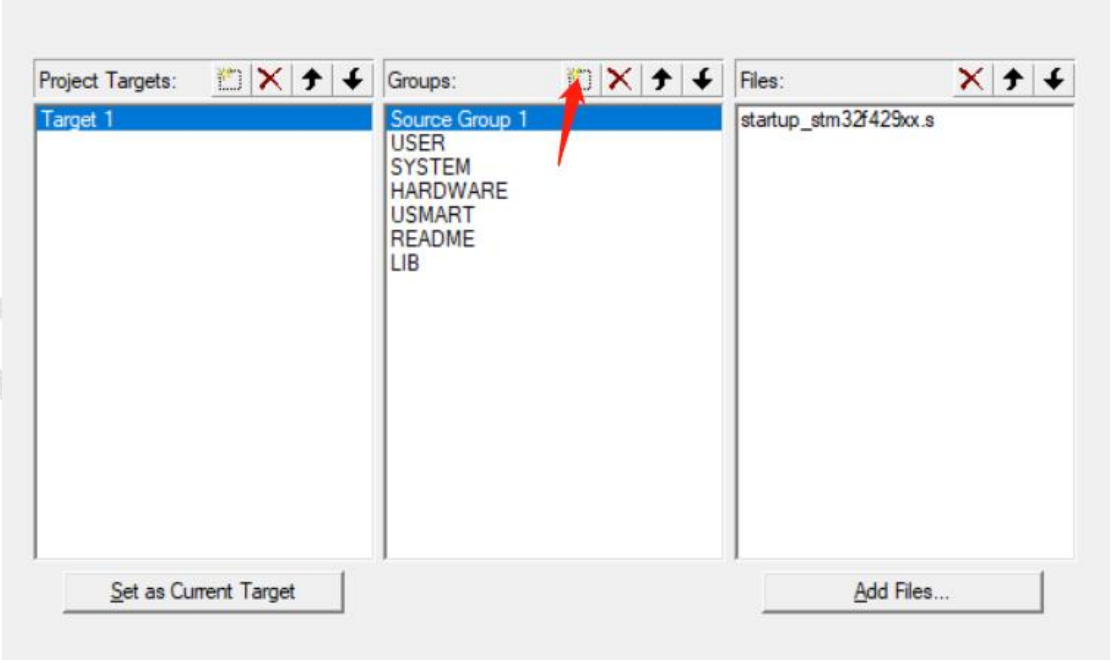
GT-HMI-Engine 引擎包的目录结构

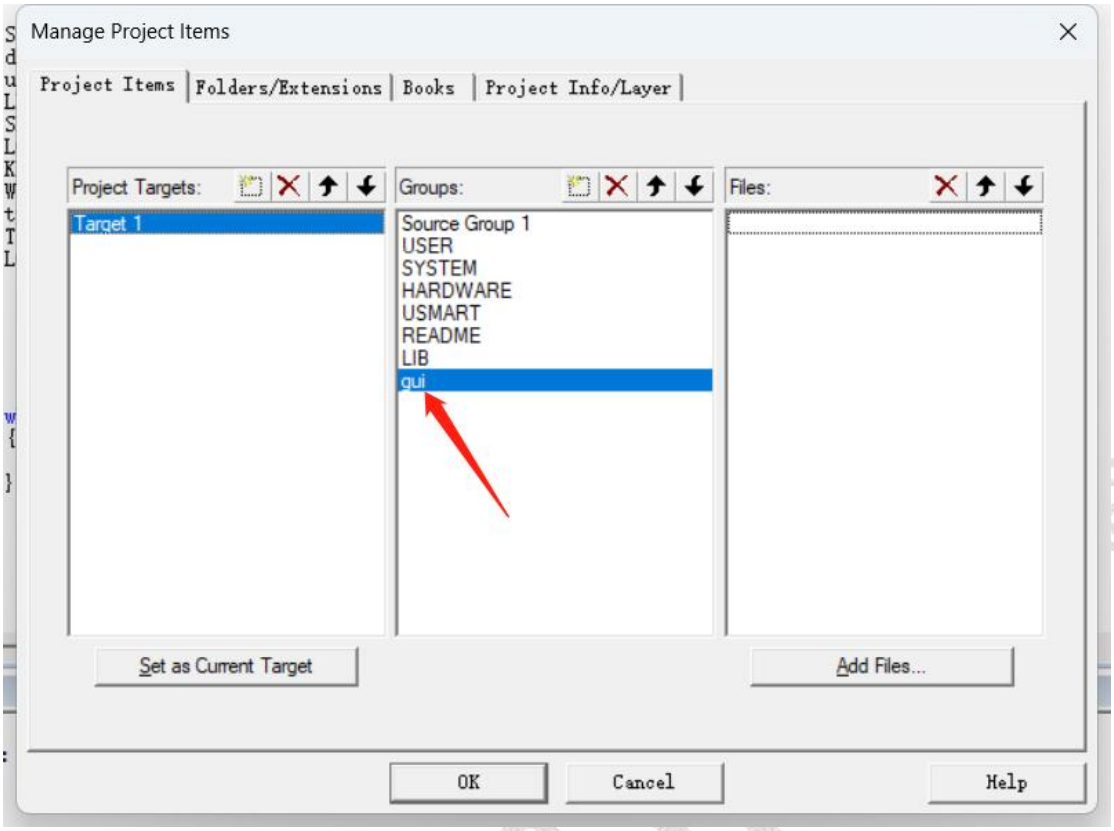
直接将整个文件夹放入 STM32F429 工程的根目录。

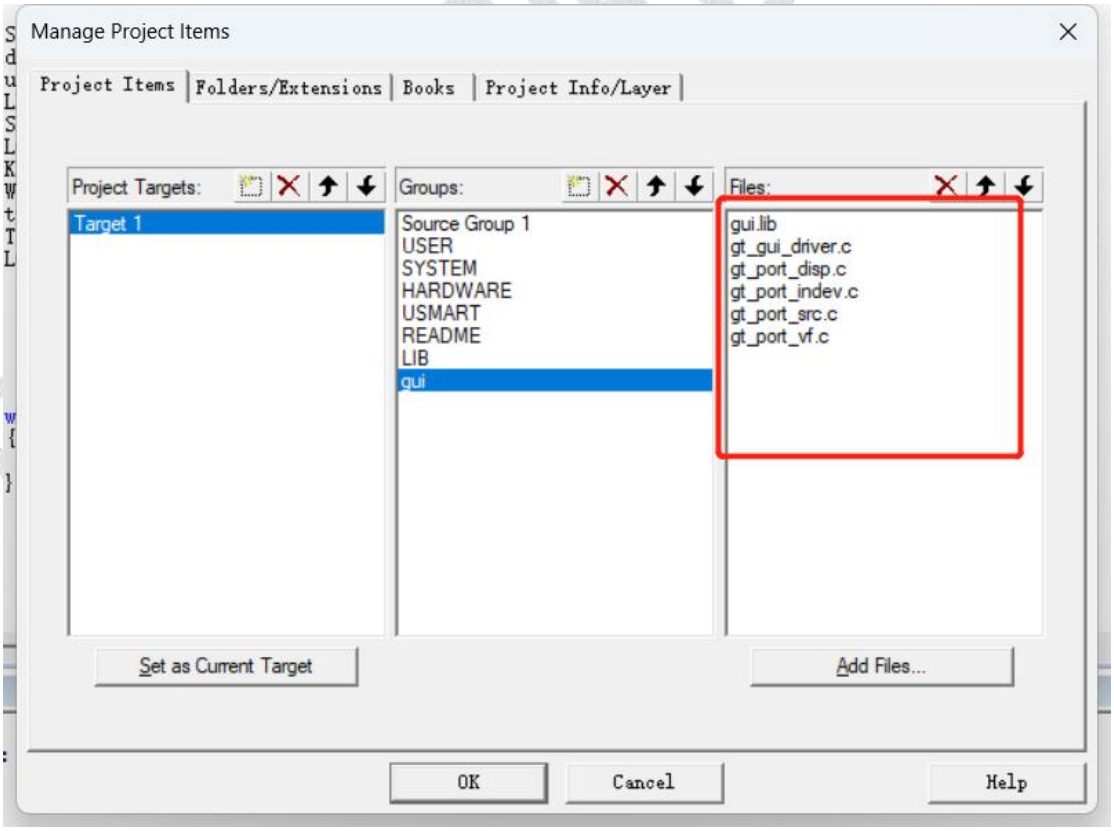
7.1.2 移植编译

在 keil 工程中新建 gui 的 groups 文件夹，并将 driver 和 src 文件中的 C 文件添加进去,example 文件夹为控件示例可以添加也可以不添加。

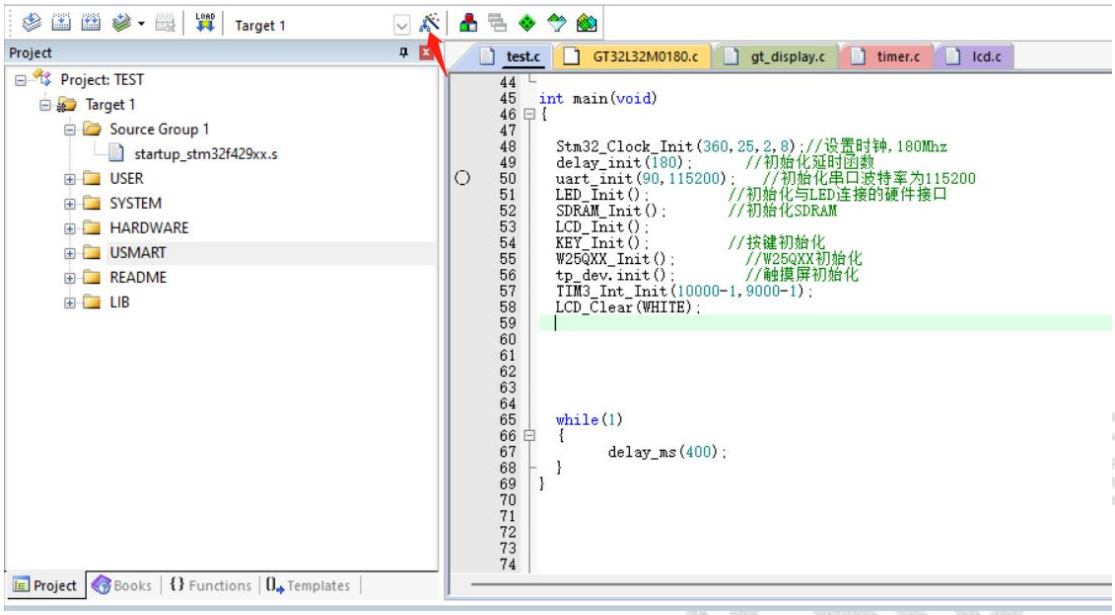


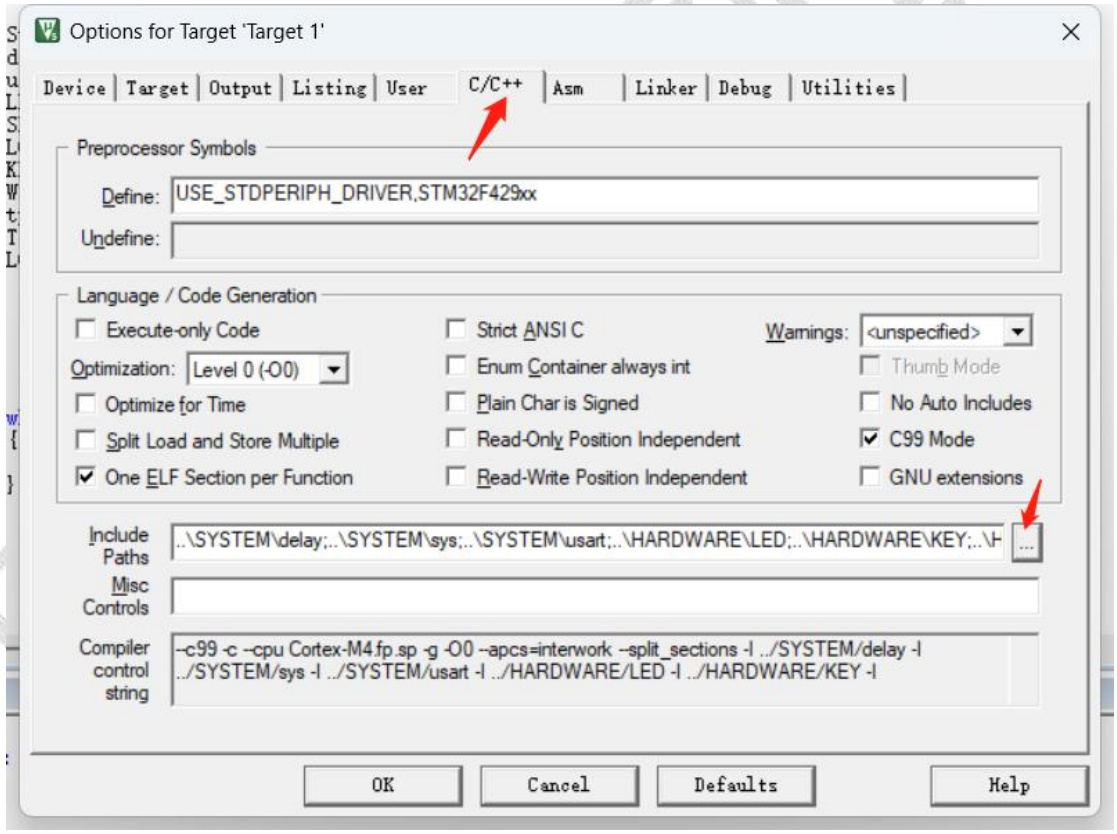


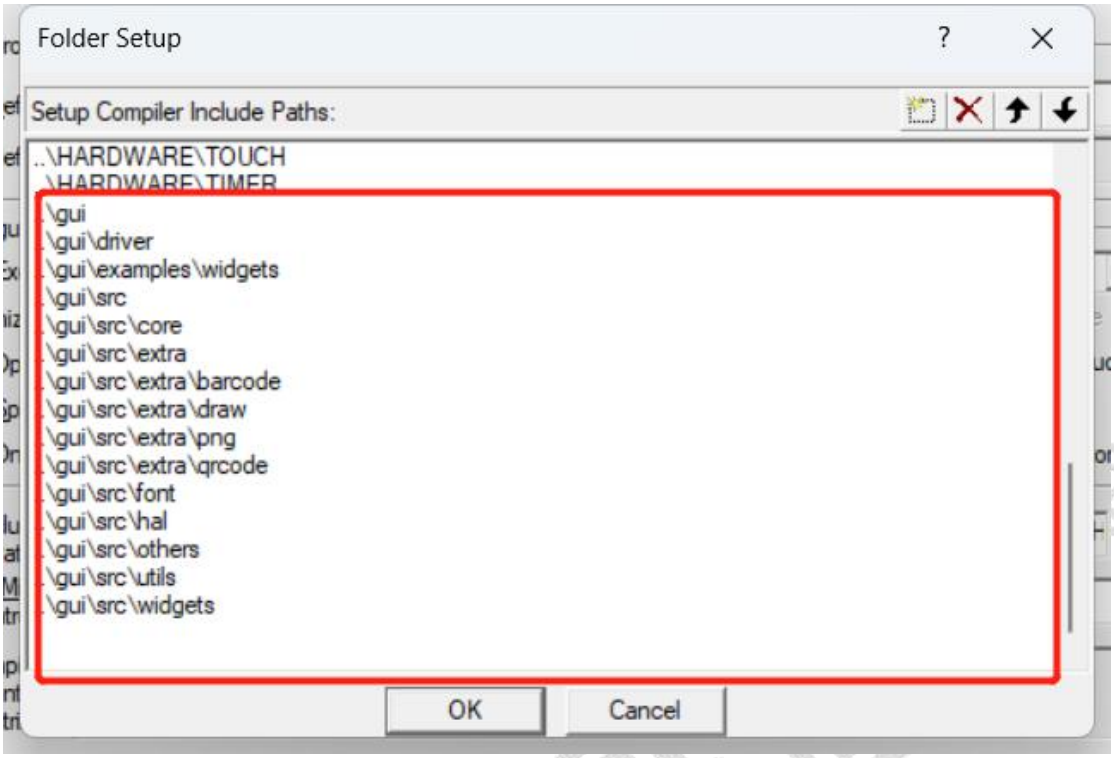

Manage Project Items
Project Items
Folders/Extensions
Books
Project Info/Layer
Project Targets:
Target 1
Groups:
Source Group 1
USER
SYSTEM
HARDWARE
USMART
README
LIB
gui
Files:
Set as Current Target
Add Files...
OK
Cancel
Help


Manage Project Items
Project Items
Folders/Extensions
Books
Project Info/Layer
Project Targets:
Target 1
Groups:
Source Group 1
USER
SYSTEM
HARDWARE
USMART
README
LIB
gui
Files:
gui.lib
gt_gui_driver.c
gt_port_disp.c
gt_port_indev.c
gt_port_src.c
gt_port_vf.c
Set as Current Target
Add Files...
OK
Cancel
Help

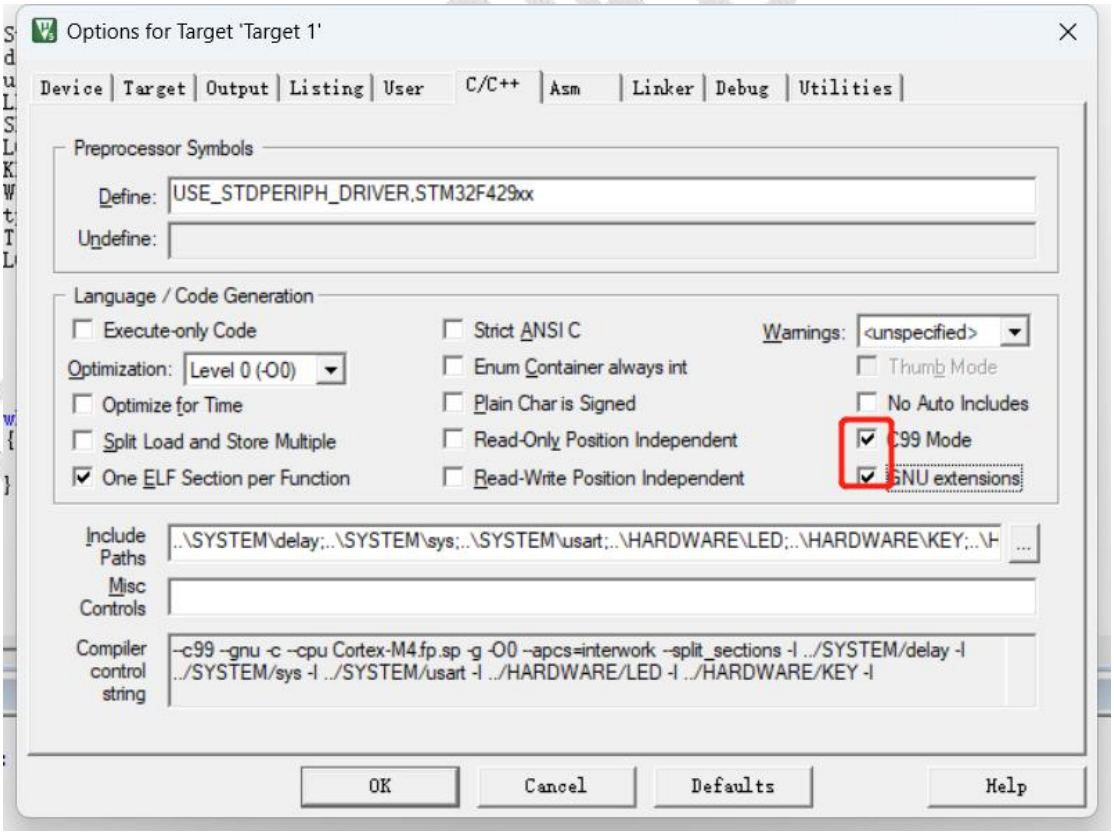
添加 c 文件后，还需要在工程配置中将对应的头文件路径一一添加进来。







添加完成如上图头文件路径后还需要打开 C99 和 GUN 模式编译选项，以防编译报错。



设置完成后编译，会提示四个错误。

```
Build Output
compiling gt_port_disp.c...
linking...
..\OBJ\TEST.axf: Error: L6218E: Undefined symbol _flush_cb (referred from gt_port_disp.o).
..\OBJ\TEST.axf: Error: L6218E: Undefined symbol read_cb (referred from gt_port_indev.o).
..\OBJ\TEST.axf: Error: L6218E: Undefined symbol read_cb_btn (referred from gt_port_indev.o).
..\OBJ\TEST.axf: Error: L6218E: Undefined symbol spi_wr (referred from gt_port_vf.o).
Not enough information to list image symbols.
Not enough information to list load addresses in the image map.
Finished: 2 information, 0 warning and 4 error messages.
"..\OBJ\TEST.axf" - 4 Error(s), 0 Warning(s).
Target not created.
Build Time Elapsed:  00:00:00
```

原因是还没有实现对应的刷屏，触摸，按键和 spi 读写驱动接口。

在 main.c 文件中分别实现这些接口函数:(注意函数名与各自外部声明的位置一致)

_flush_cb 刷屏函数(声明在 gt_port_disp.c 中)

```
void _flush_cb(struct _gt_disp_drv_s * drv, gt_area_st * area, gt_color_t * color){

  LTDC_Color_Fill(area->x, area->y, area->w+area->x-1, area->h+area->y-1, color);

}
```

read_cb 触摸函数(声明在 gt_port_indev.c 中)

```
void read_cb(struct _gt_indev_drv_s * indev_drv, gt_indev_data_st * data){
  if(!tp_dev.scan(1)) {
    data->state = GT_INDEV_STATE_RELEASED;
    return ;
  }
  data->point.x = tp_dev.x[0];
  data->point.y = tp_dev.y[0];
  data->state = GT_INDEV_STATE_PRESSED;
}
```

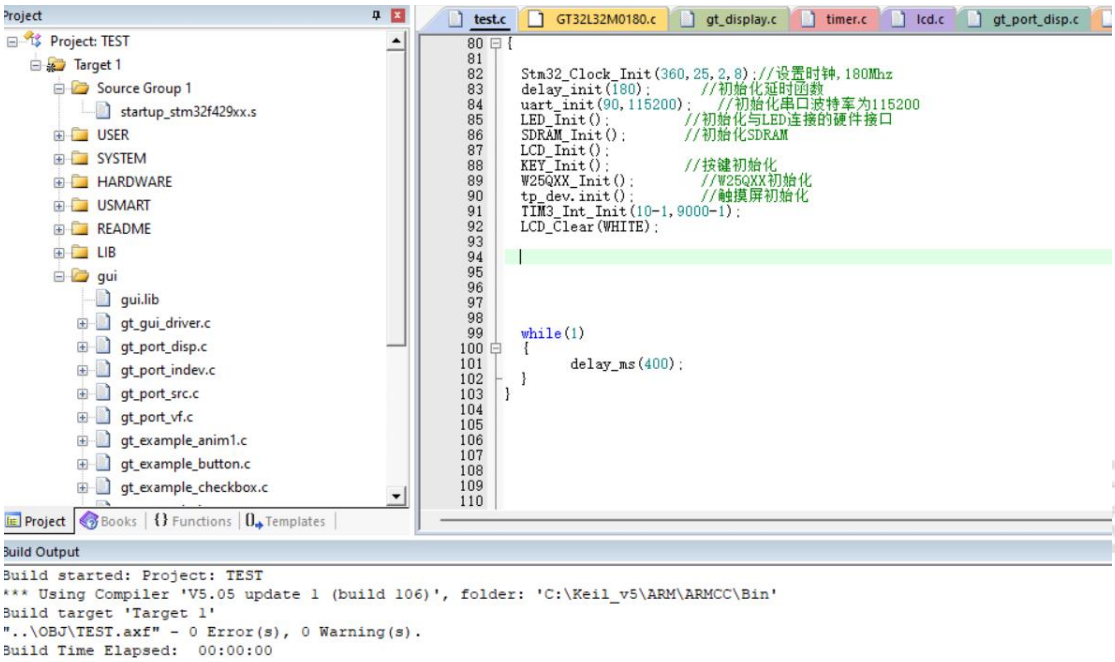
read_cb_btn 按键函数(声明在 gt_port_indev.c 中)

```
void read_cb_btn(struct _gt_indev_drv_s * indev_drv, gt_indev_data_st * data)
{
  uint8_t status = 0;
  status = KEY_Scan(1);
  if( status ){
    data->btn_id = status;
    data->state = GT_INDEV_STATE_PRESSED;
  }else{
    data->state = GT_INDEV_STATE_RELEASED;
  }
}
```

spi_wr 函数(声明在 gt_port_vf.c 中,这里没使用字库，可以为空不实现功能，查看 6.2 实现。)

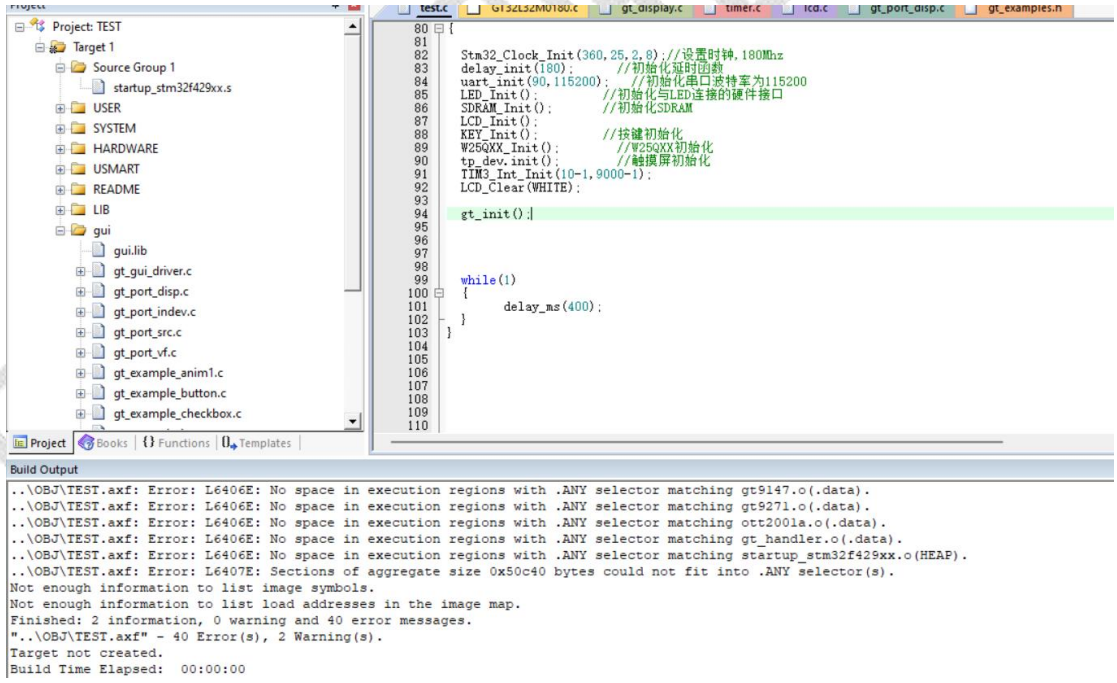
```
void spi_wr(uint8_t *dw, uint32_t lw, uint8_t *dr, uint32_t lr)
{
  return ;
}
```

再次编译,会提示成功未出现报错。

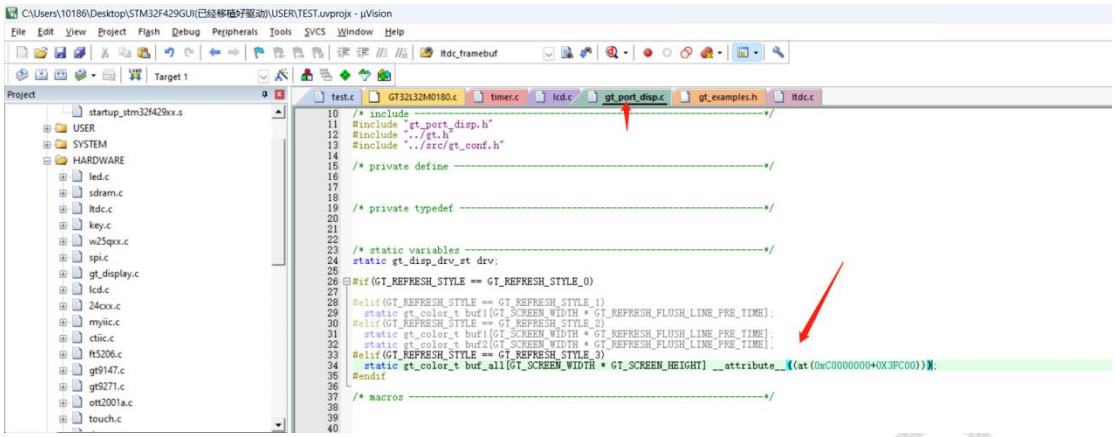


在 main.c 文件中 include 包含 gt.h，然后调用里面声明的 gt_init 函数,再次编译,可能会提示 No space xxx 的错误。

```
#include "gt.h"
#include "../GT-HMI-Engine/examples/gt_examples.h"
```

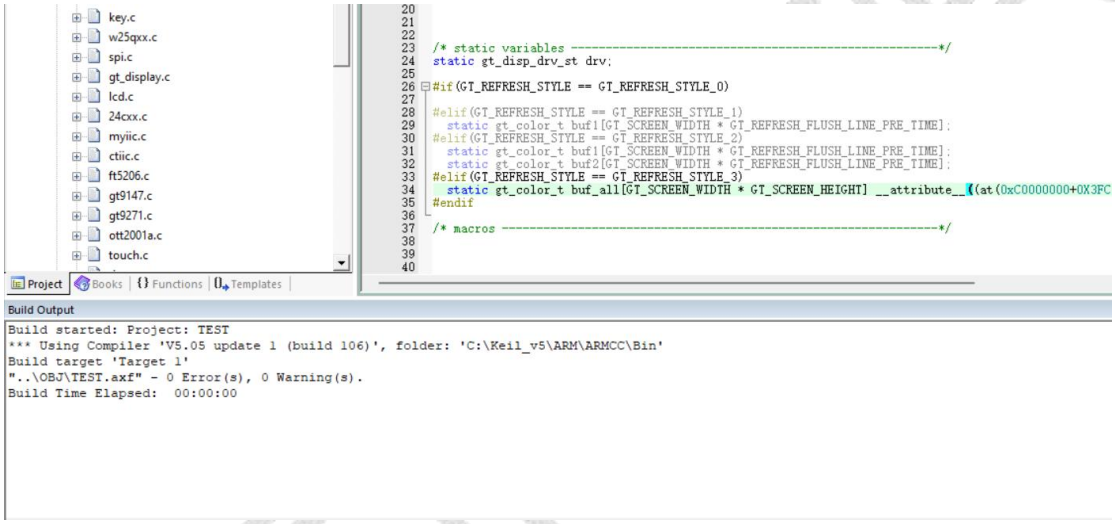


原因是 gt_port_disp.c 中定义的刷屏数组太大，超过片内的 ram 空间而报错，将其定义在片外的 sram 中即可，其定义的地址需要自己参考对应主控芯片的情况以及对 sram 的使用情况来决定。

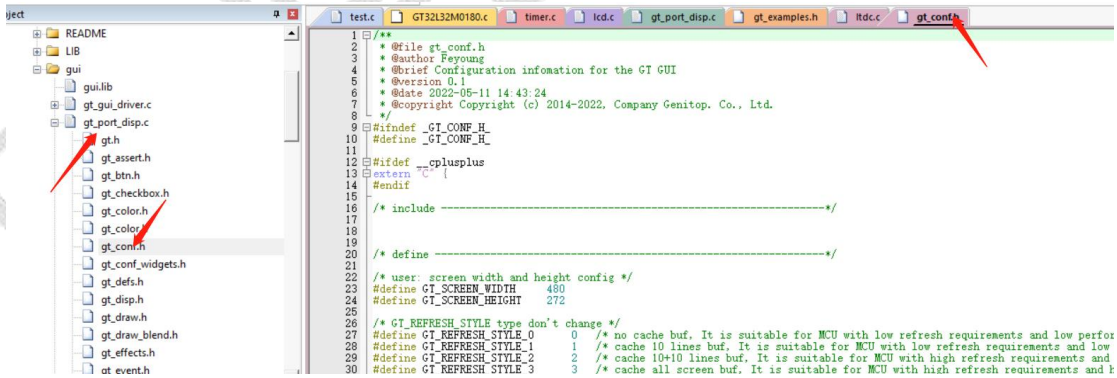


重定义 gt_port_disp.c 中的刷屏数组

再次编译即可通过。



之后修改 gt_conf.h 中的屏幕尺寸和内存空间参数，以适配开发板环境。

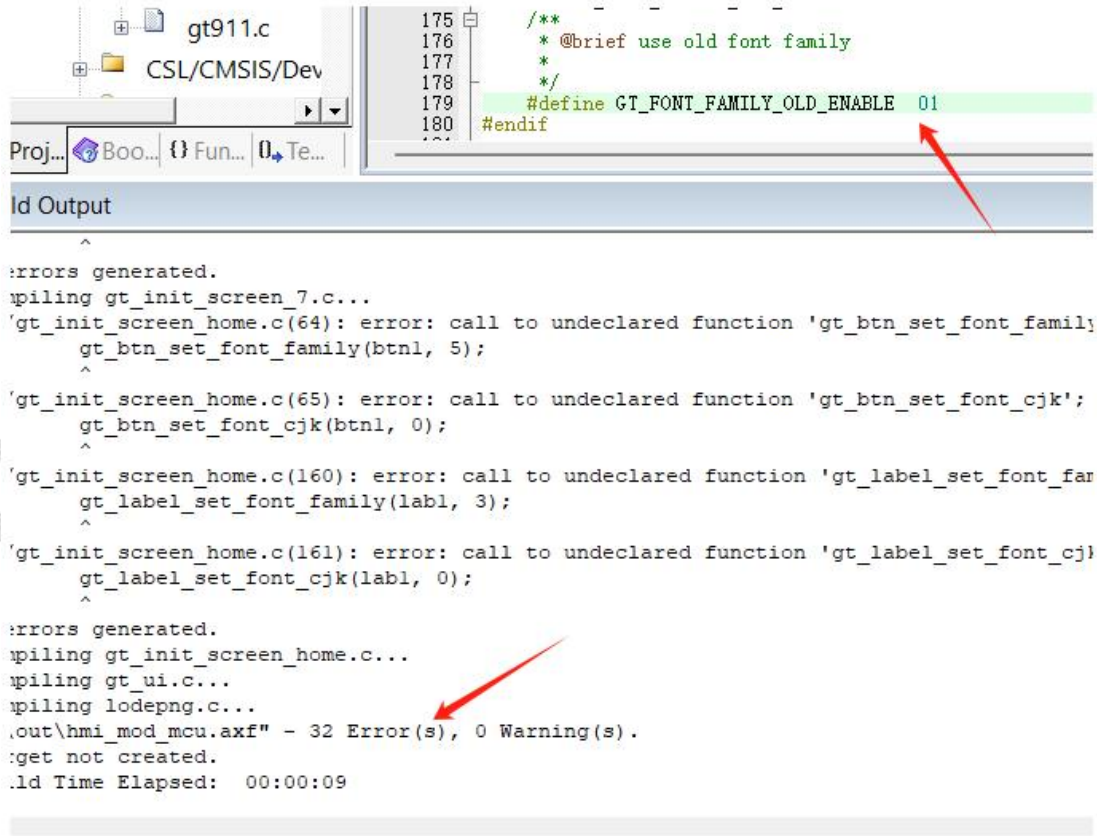


```
 * @date 2022-05-11 14:43:24
 * @copyright Copyright (c) 2014-2022, Company Genitop. Co., Ltd.
 */
#ifndef _GT_CONF_H_
#define _GT_CONF_H_

#ifdef __cplusplus
extern "C" {
#endif

/* include ------------------------------------------------------------*/



/* define -------------------------------------------------------------*/

/* user: screen width and height config */
#define GT_SCREEN_WIDTH     480
#define GT_SCREEN_HEIGHT    272

/* GT_REFRESH_STYLE type don't change */
#define GT_REFRESH_STYLE_0      0   /* no cache buf, It is suitable for MCU with low refresh requirements and low performance */
#define GT_REFRESH_STYLE_1      1   /* cache 10 lines buf, It is suitable for MCU with low refresh requirements and low performance
#define GT_REFRESH_STYLE_2      2   /* cache 10+10 lines buf, It is suitable for MCU with high refresh requirements and high perfor
#define GT_REFRESH_STYLE_3      3   /* cache all screen buf, It is suitable for MCU with high refresh requirements and high perform
```

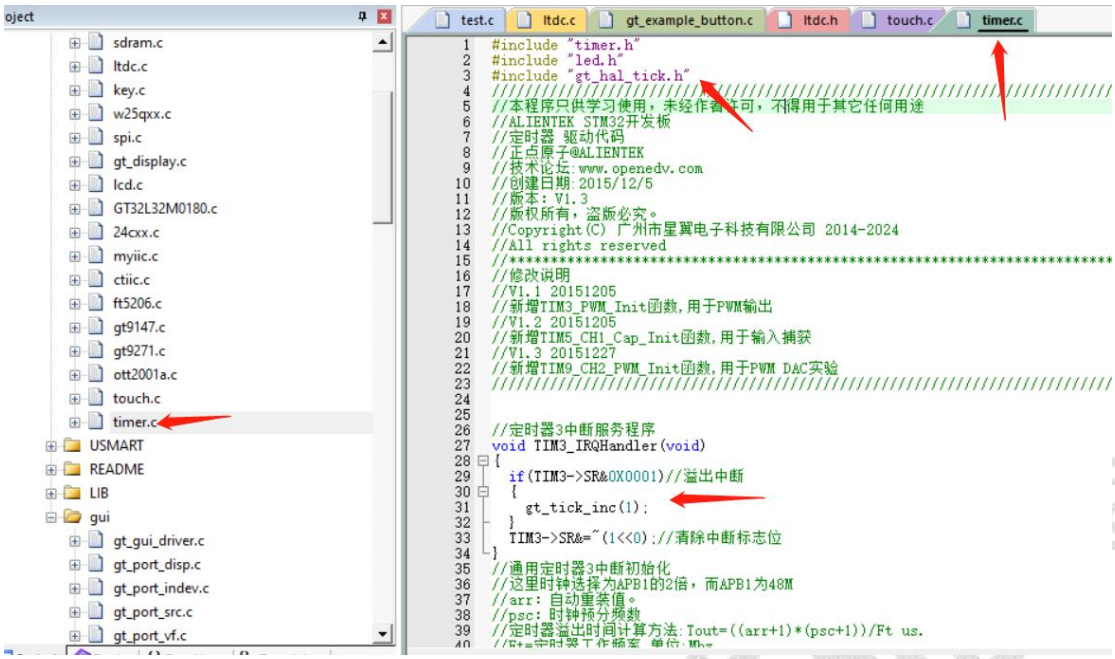
如果使用控件很多，可适当加大 GT_MEM_SIZE 的大小。

```
#if GT_MEM_CUSTOM
    #define GT_MEM_CUSTOM_INCLUDE   "../others/gt_tlsf.h"

    #define GT_MEM_SIZE             (44 * 1024U)    //Byte

    #define gt_tlsf_assert(_expr)   ( (void)0 )
#else
    #define GT_MEM_CUSTOM_INCLUDE   "stdlib.h"
```

需要根据实际使用的字库来修改 GT_FONT_FAMILY_OLD_ENABLE，为 1 时使用旧版字库体系，为 0 时使用新版字库体系，不然在编译时会有报错。



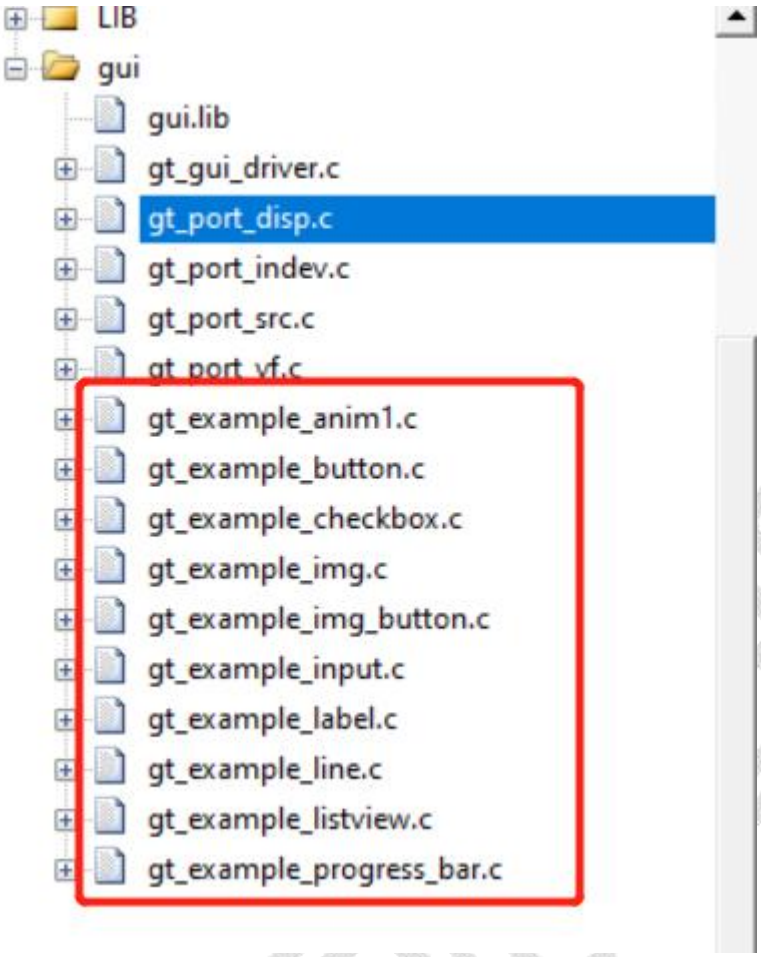
在开发板的定时器驱动中加入 1ms 的心跳函数 gt_tick_inc(1)，保证 GUI 持续运行。



在 main 函数中设置定时器 1ms 触发一次中断并在 while 循环中加入 gui 的事务处理函数 gt_task_handler 使其每 1ms 循环执行一次。

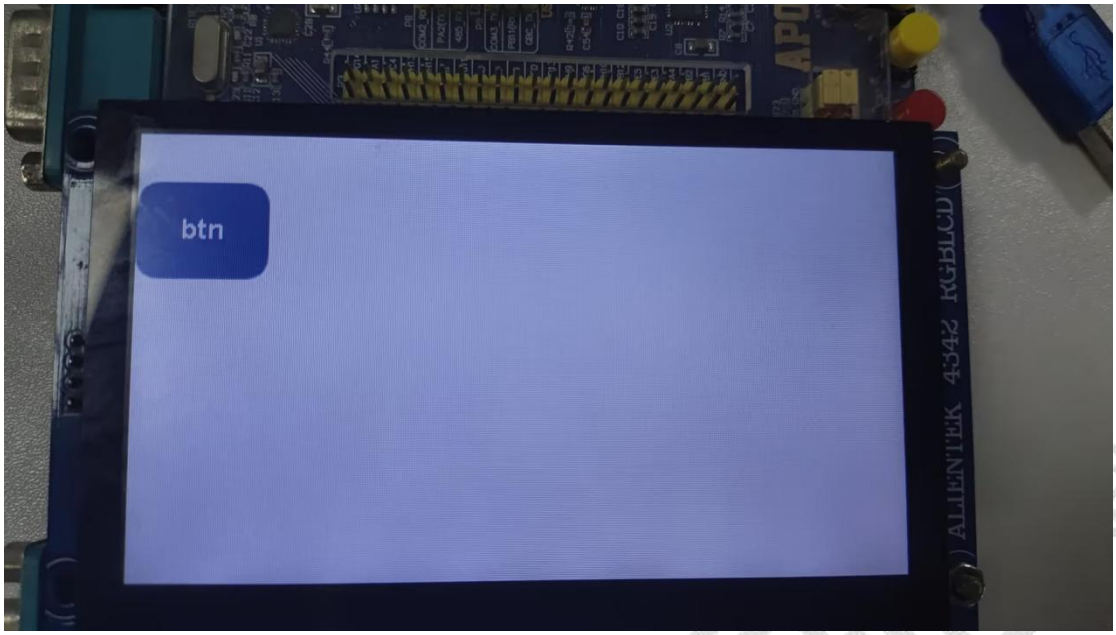
```
int main(void)
{
  u16 i=0;
  u8 a;
  unsigned char DZ_Data[512]={0};
  unsigned char pBits[512]={0};
  unsigned int n=0;
  u32 addr=0;
  Stm32_Clock_Init(360,25,2,8);//设置时钟,180Mhz
  delay_init(180);       //初始化延时函数
  uart_init(90,115200);    //初始化串口波特率为115200
  usmart_dev.init(90);     //初始化USMART
  LED_Init();          //初始化与LED连接的硬件接口
  SDRAM_Init();        //初始化SDRAM
  LCD_Init();
  KEY_Init();          //按键初始化
  W25QXX_Init();         //W25QXX初始化
  tp_dev.init();         //触摸屏初始化
  TIM3_Int_Init(10-1,9000-1);
  LCD_Clear(WHITE);

  gt_init();

  gt_examples_button();

  while(1)
  {
      gt_task_handler();
      delay_ms(1);
  }
}
```

导入 example 文件夹中的例程，调用控件的例程进行测试。

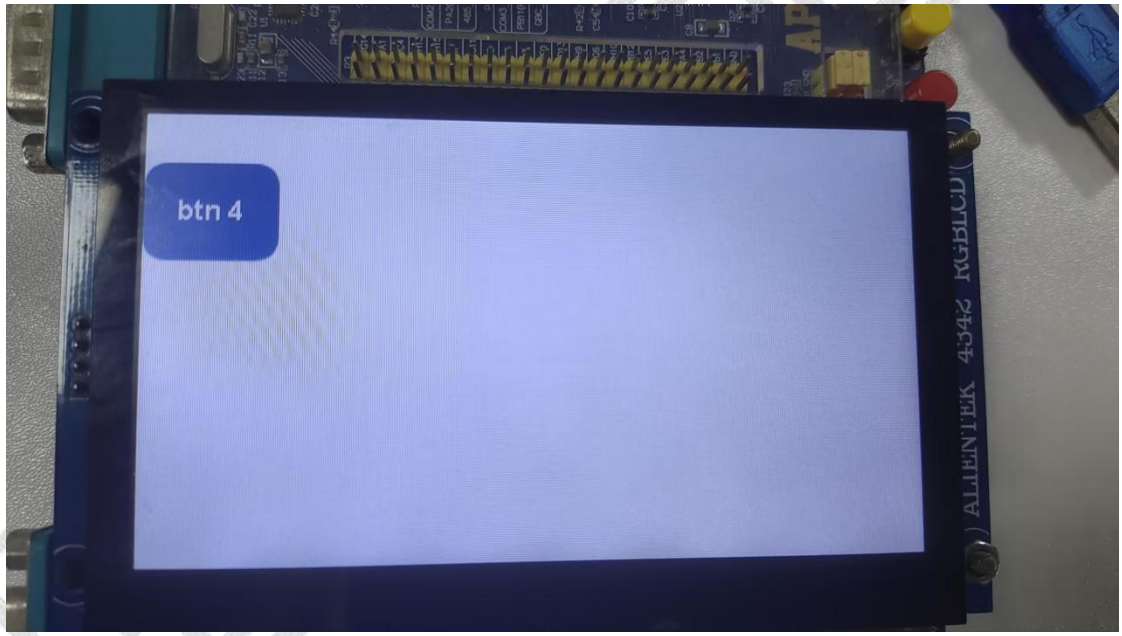



```
int main(void)
{

  Stm32_Clock_Init(360,25,2,8);//设置时钟,180Mhz
  delay_init(180);        //初始化延时函数
  uart_init(90,115200);    //初始化串口波特率为115200
  LED_Init();           //初始化与LED连接的硬件接口
  SDRAM_Init();         //初始化SDRAM
  LCD_Init();
  KEY_Init();           //按键初始化
  W25QXX_Init();          //W25QXX初始化
  tp_dev.init();          //触摸屏初始化
  TIM3_Int_Init(10-1,9000-1);
  LCD_Clear(WHITE);

  gt_init();
  gt_examples_button();


  while(1)
  {
        gt_task_handler();
        delay_ms(1);
  }
}
```

编译并烧录进开发板，可以看到 btn 控件的生成。



点击按钮会有对应事件出现

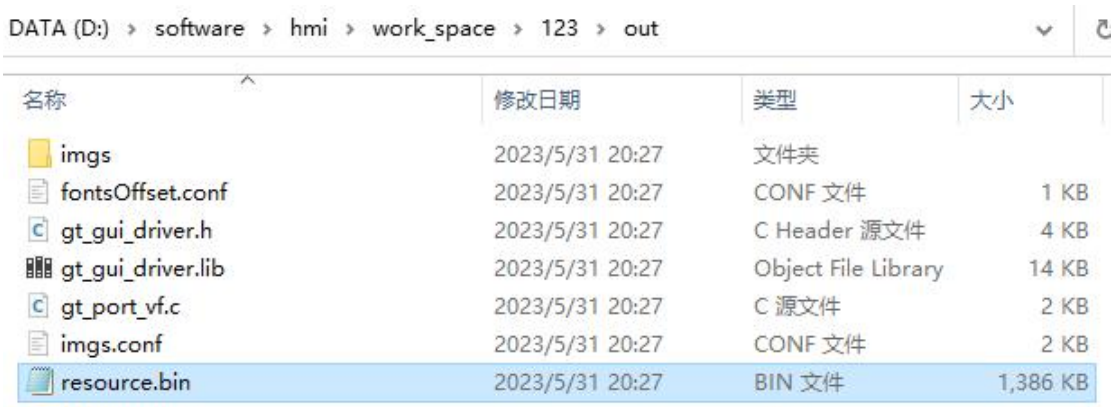


至此移植完成。

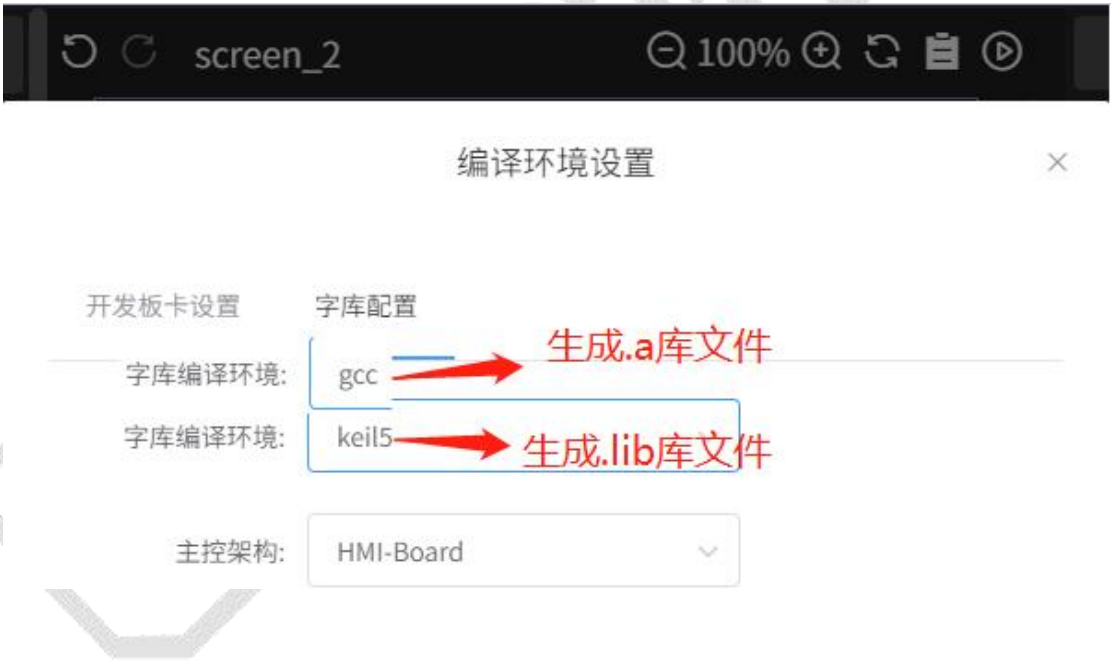
7.2 素材导入流程(bin 文件方式)

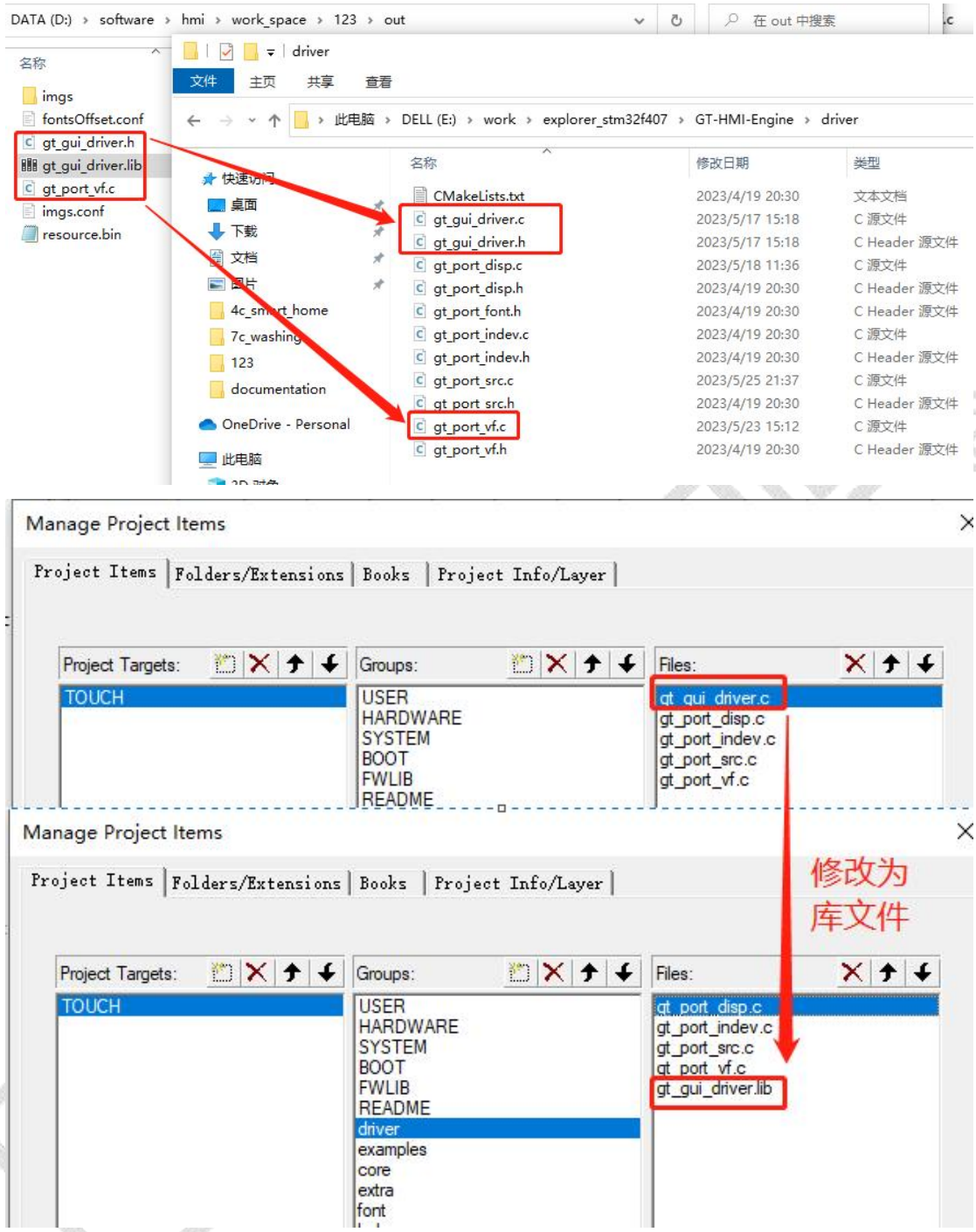
在 GUI 平台上如果使用自定义的图片或字库等素材，采用 Bin 文件形式将素材导入到下位机板子中。

GT-HMI Designer 工程文件编译成功后，会在工程目录下生成 out 文件夹(使用 GT-HMI 的模块时请使用 board 文件夹中的 resource.bin 文件)，out 目录中的 bin 文件是 GT-HMI Designer 工程中用到的图片和字库等素材转换来的，如下图所示：



将 resource.bin 文件烧录到您所使用的我司提供的 flash 芯片中。然后复制生成的 gt_port_vf.c 文件，假如使用字库则会额外生成 gt_gui_driver.h 和库文件（此处与 Designer 软件中编译环境设置有关，设置为 keil5 则生成 gt_gui_driver.lib 库文件，设置为 gcc 编译则生成 libgt_gui_driver.a 库文件），将生成的文件替换掉 STM32F429 工程 GT-HMI-Engine 引擎包 driver 中的 gt_port_vf.c 文件，gt_gui_driver.h 文件和 gt_gui_driver.c 文件。注意使用库.lib 文件替换掉.c 源文件时，工程中的配置也要相应改变。





实现 gt_gui_driver.h 中外部声明的 r_dat_bat 函数和之前移植中未实现的 spi_wr()函数，其中 datawrite 为需写入数据的数组，len_write 为写入的数据长度，data_read 为存放读取数据的数组,len_read 为读取数据的长度。参考实现如下:

gt_gui_driver.h

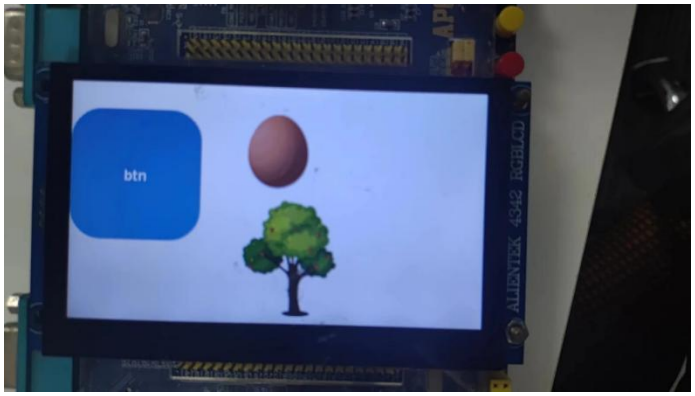
```
11
12  /* 外部函数声明 */
13  extern unsigned long r_dat_bat(unsigned long address, unsigned long DataLen, unsigned char* pBuff);
```

```
unsigned long r_dat_bat(unsigned long address,unsigned long DataLen,unsigned char *pBuff)
{
    unsigned long i;
    unsigned char addrHigh;
    unsigned char addrMid;
    unsigned char addrLow;

    addrHigh=address>>16;
    addrMid=address>>8;
    addrLow=(unsigned char)address;

    Rom_csL;          //片选选中芯片

    SPI1_ReadWriteByte(0x03);   //普通读取首先送0X03,然后发送地址高八位addrHigh，中八位addrMid，低八位addrLow。
    SPI_Address(addrHigh,addrMid,addrLow);

    for(i=0;i<DataLen;i++)
    {
        pBuff[i]=SPI1_ReadWriteByte(0x00);
    }
    Rom_csH;

    return 0;
}
```

```
uint32_t spi_wr(uint8_t * data_write, uint32_t len_write, uint8_t * data_read, uint32_t len_read)
{
    uint32_t i;
    spi_cs_set(0);
    for(i = 0;i < len_write;i++){
        SPI1_ReadWriteByte(data_write[i]);
    }
    for(i = 0;i < len_read;i++){
        data_read[i] = SPI1_ReadWriteByte(0xFF);
    }
    spi_cs_set(1);
    return 0;
}
```

实现上述两个函数之后，调用文字或者 img 图片控件，设置图片路径，可以看到文字和图片显示情况，根据需求做排版布局调整。

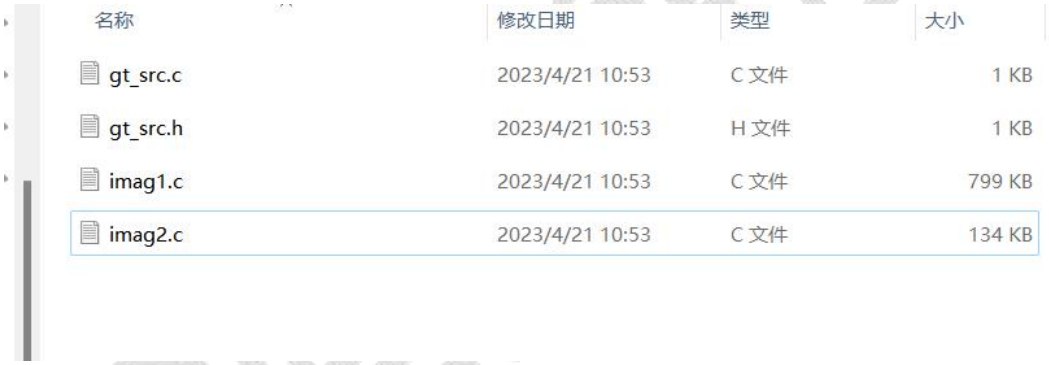
```
    gt_obj_st * screen;
    screen = gt_obj_create(NULL);

    gt_obj_st * btn = gt_btn_create(screen);
    gt_obj_style_set_pos(btn, 0, 30);
    gt_obj_style_set_size(btn, 150, 150);
    gt_btn_style_set_text(btn, "btn");
    gt_obj_add_event_cb(btn, btn_click_cb, GT_EVENT_TYPE_INPUT_PROCESSED, NULL);

    gt_obj_st * img1 = gt_img_create(screen);      //创建控件
    gt_obj_style_set_pos(img1,200,30);             //设置控件位置
    gt_img_style_set_src(img1,"f:img3.png");       //设置图片路径

    gt_obj_st * img2 = gt_img_create(screen);      //创建控件
    gt_obj_style_set_pos(img2,200,130);            //设置控件位置
    gt_img_style_set_src(img2,"f:img4.png");       //设置图片路径


    gt_disp_load_scr(screen);

  while(1)
  {
        gt_task_handler();
        delay_ms(1);
  }
```



7.3 素材导入流程(C 文件数组方式)

在 GUI 平台上如果使用自定义的图片或字库等素材，采用 C 文件数组形式将素材导入到下位机板子中。

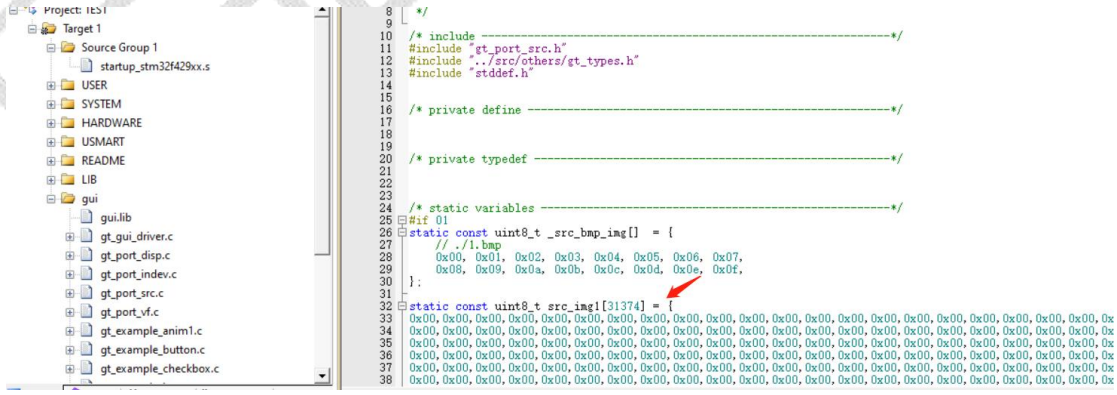
GT-HMI Designer 工程文件编译后，图片素材同样会以数组形式导出来，存放路径为“工程存放路径/sources/imgsCode”。Imag1 是图片 1 转换出来的数组数据，imag2 是图片 2 转换出来的数组数据。



名称	修改日期	类型	大小
gt_src.c	2023/4/21 10:53	C 文件	1 KB
gt_src.h	2023/4/21 10:53	H 文件	1 KB
imag1.c	2023/4/21 10:53	C 文件	799 KB
imag2.c	2023/4/21 10:53	C 文件	134 KB

图片素材和 C 文件

将各自图片 C 文件中的数组直接复制到工程的 gt_port_src.c 文件中



使用 imgsCode/gt_src.c 文件中数组 user_src_list 列举出的文件路径数据替换掉 gt_port_src.c 中的_src_icon_sys 结构体数组数据。user_src_list 如图 1 所示，_src_icon_sys 如图二所示。

```
const gt_src_st user_src_list[] = {
    {".:img_148x77_fh.png", 148, 77, src_img_148x77_fh, 34188, 1},
    {".:img_148x77_113.png", 97, 110, src_img_148x77_113, 32010, 1},
    {".:img_97x110_112.png", 97, 110, src_img_97x110_112, 32010, 1},
    {".:img_148x77_110.png", 97, 110, src_img_148x77_110, 32010, 1},
    {NULL, 0, 0, NULL, 0, 0},
};
```

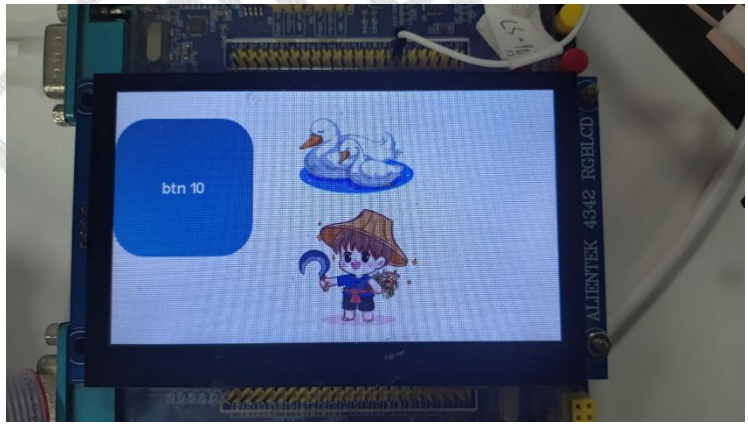
图 1

```
static const gt_src_st _src_icon_sys[] =
{
    {".:img_148x77_fh.png", 148, 77, src_img_148x77_fh, 34188, 1},
    {".:img_148x77_113.png", 97, 110, src_img_148x77_113, 32010, 1},
    {".:img_97x110_112.png", 97, 110, src_img_97x110_112, 32010, 1},
    {".:img_148x77_110.png", 97, 110, src_img_148x77_110, 32010, 1},
    {NULL, 0, 0, NULL, 0, 0},
};
```

图 2

调用 img 控件并赋予文件路径即可显示

```
gt_obj_st * screen;
screen = gt_obj_create(NULL);

gt_obj_st * btn = gt_btn_create(screen);
gt_obj_style_set_pos(btn, 0, 30);
gt_obj_style_set_size(btn, 150, 150);
gt_btn_style_set_text(btn, "btn");
gt_obj_add_event_cb(btn, btn_click_cb, GT_EVENT_TYPE_INPUT_PROCESSED, NULL);

gt_obj_st * img1 = gt_img_create(screen);      //创建控件
gt_obj_style_set_pos(img1,200,30);             //设置控件位置
gt_img_style_set_src(img1,".:img1.png");       //设置图片路径

gt_obj_st * img2 = gt_img_create(screen);      //创建控件
gt_obj_style_set_pos(img2,200,130);            //设置控件位置
gt_img_style_set_src(img2,".:img2.png");       //设置图片路径

gt_disp_load_scr(screen);
```



7.4 需要实现的硬件适配代码

移植 GUI 所需适配的底层硬件函数及示例。
刷屏函数
_flush_cb(struct _gt_disp_drv_s * drv, gt_area_st * area, gt_color_t * color)
参数说明: _gt_disp_drv_s drv 此参数不需用户适配
gt_area_st 结构体,area->x 为绘图的起始 x 坐标
area->y 为绘图的起始 y 坐标
area->w 为绘图区域的宽度
area->h 为绘图区域的高度
color 为图片数据数组
例程中调用的为正点原子的区域填充代码,其余方式的实现函数只需要能够在指定区域绘制指定大小的图片均可。

```
void _flush_cb(struct _gt_disp_drv_s * drv, gt_area_st * area, gt_color_t * color){
    LCD_Color_Fill(area->x,area->y,area->w+area->x-1,area->h+area->y-1,color);
}
```

正点原子该函数实现示例

```
//在指定区域内填充指定颜色块
//(sx,sy),(ex,ey):填充矩形对角坐标,区域大小为:(ex-sx+1)*(ey-sy+1)
//color:要填充的颜色
void LCD_Color_Fill(u16 sx,u16 sy,u16 ex,u16 ey,u16 *color)
{
    u16 height,width;
    u16 i,j;
    if(lcdltdc.pwidth!=0)    //如果是 RGB 屏
    {
        LTDC_Color_Fill(sx,sy,ex,ey,color);
    }else
    {
        width=ex-sx+1;              //得到填充的宽度
        height=ey-sy+1;             //高度
        for(i=0;i<height;i++)
        {
            LCD_SetCursor(sx,sy+i);      //设置光标位置
            LCD_WriteRAM_Prepare();    //开始写入 GRAM
            for(j=0;j<width;j++)LCD->LCD_RAM=color[i*width+j];//写入数据
        }
    }
}
```

如果是 RGB 屏使用 DMA 方式传输

```
//在指定区域内填充指定颜色块,DMA2D 填充
//此函数仅支持 u16,RGB565 格式的颜色数组填充.
//(sx,sy),(ex,ey):填充矩形对角坐标,区域大小为:(ex-sx+1)*(ey-sy+1)
//注意:sx,ex,不能大于 lcddev.width-1;sy,ey,不能大于 lcddev.height-1!!!
//color:要填充的颜色数组首地址
void LTDC_Color_Fill(u16 sx,u16 sy,u16 ex,u16 ey,u16 *color)
{
    u32 psx,psy,pex,pey;    //以 LCD 面板为基准的坐标系,不随横竖屏变化而变化
    u32 timeout=0;
    u16 offline;
    u32 addr;
    //坐标系转换
    if(lcdltdc.dir)  //横屏
    {
        psx=sx;psy=sy;
        pex=ex;pey=ey;
    }else            //竖屏
    {
        psx=sy;psy=lcdltdc.pheight-ex-1;
        pex=ey;pey=lcdltdc.pheight-sx-1;
    }
    offline=lcdltdc.pwidth-(pex-psx+1);
    addr=((u32)ltdc_framebuf[lcdltdc.activelayer]+lcdltdc.pixsize*(lcdltdc.pwidth*psy+psx));
    RCC->AHB1ENR|=1<<23;                //使能 DM2D 时钟
    DMA2D->CR&=~(1<<0);                 //先停止 DMA2D
    DMA2D->CR=0<<16;                    //存储器到存储器模式
    DMA2D->FGPFCCR=LCD_PIXFORMAT;       //设置颜色格式
    DMA2D->FGOR=0;                      //前景层行偏移为 0
    DMA2D->OOR=offline;                 //设置行偏移
    DMA2D->FGMAR=(u32)color;            //源地址
    DMA2D->OMAR=addr;                   //输出存储器地址
    DMA2D->NLR=(pey-psy+1)|((pex-psx+1)<<16);   //设定行数寄存器
    DMA2D->CR|=1<<0;                    //启动 DMA2D
    while((DMA2D->ISR&(1<<1))==0)       //等待传输完成
    {
        timeout++;
        if(timeout>0X1FFFFF)break; //超时退出
    }
    DMA2D->IFCR|=1<<1;                  //清除传输完成标志
}
```

读取触摸函数

```
void read_cb(struct _gt_indev_drv_s * indev_drv, gt_indev_data_st * data)
```

参数说明: _gt_indev_drv_s indev_drv 此参数不需用户适配
gt_indev_data_st 结构体 data->state 有无输入事件发生
data->point.x 触摸点的 x 坐标
data->point.y 触摸点的 y 坐标

在此函数中，判断有无触摸事件发生，如果没有发生则返回，如果发生触摸函数则将触摸点的 x 坐标和 y 坐标传入 data 结构体的 point.x 和 point.y，并将 state 设置为 GT_INDEV_STATE_PRESSED 即可。其余的实现方式只需要测试触摸屏幕时，能否读取到正确的触摸状态和对应的 x y 坐标均可。

```
void read_cb(struct _gt_indev_drv_s * indev_drv, gt_indev_data_st * data){
    if(!tp_dev.scan(1)) {
        data->state = GT_INDEV_STATE_RELEASED;   //如果没有触摸事件则返回
        return ;
    }
    data->point.x = tp_dev.x[0];
    data->point.y = tp_dev.y[0];
    data->state = GT_INDEV_STATE_PRESSED;
}
```

主要需要实现 scan 函数,正点例程中为了区分不同的触摸设备设计了不同的 scan 函数。
此处选用其中一个作为示例

```
//扫描触摸屏(采用查询方式)
//mode:0,正常扫描.
//返回值:当前触屏状态.
//0,触屏无触摸;1,触屏有触摸
u8 GT9147_Scan(u8 mode)
{
    u8 buf[4];
    u8 i=0;
    u8 res=0;
    u8 temp;
    u8 tempsta;
    static u8 t=0; //控制查询间隔,从而降低 CPU 占用率
    t++;
    if((t%10)==0||t<10)//空闲时,每进入 10 次 CTP_Scan 函数才检测 1 次,从而节省 CPU 使用
率
    {
        GT9147_RD_Reg(GT_GSTID_REG,&mode,1);    //读取触摸点的状态
        if(mode&0X80&&((mode&0XF)<6))
        {
            temp=0;
            GT9147_WR_Reg(GT_GSTID_REG,&temp,1);//清标志
        }
```

```
if((mode&0XF)&&((mode&0XF)<6))
{
    temp=0XFF<<(mode&0XF);      //将点的个数转换为 1 的位数,匹配 tp_dev.sta
定义
    tempsta=tp_dev.sta;         //保存当前的 tp_dev.sta 值
    tp_dev.sta=(~temp)|TP_PRES_DOWN|TP_CATH_PRES;
    tp_dev.x[4]=tp_dev.x[0];    //保存触点 0 的数据
    tp_dev.y[4]=tp_dev.y[0];
    for(i=0;i<5;i++)
    {
        if(tp_dev.sta&(1<<i))   //触摸有效
        {
            GT9147_RD_Reg(GT9147_TPX_TBL[i],buf,4);    //读取 XY 坐标值
        if(lcddev.id==0X5510)  //4.3 寸 800*480 MCU 屏
        {
            if(tp_dev.touchtype&0X01)//横屏
            {
                tp_dev.y[i]=((u16)buf[1]<<8)+buf[0];
                tp_dev.x[i]=800-(((u16)buf[3]<<8)+buf[2]);
            }else
            {
                tp_dev.x[i]=((u16)buf[1]<<8)+buf[0];
                tp_dev.y[i]=((u16)buf[3]<<8)+buf[2];
            }
        }else if(lcddev.id==0X4342) //4.3 寸 480*272 RGB 屏
        {
            if(tp_dev.touchtype&0X01)//横屏
            {
                tp_dev.x[i]=(((u16)buf[1]<<8)+buf[0]);
                tp_dev.y[i]=(((u16)buf[3]<<8)+buf[2]);
            }else
            {
                tp_dev.y[i]=((u16)buf[1]<<8)+buf[0];
                tp_dev.x[i]=272-(((u16)buf[3]<<8)+buf[2]);
            }
        }else if(lcddev.id==0X4384) //4.3 寸 800*480 RGB 屏
        {
            if(tp_dev.touchtype&0X01)//横屏
            {
                tp_dev.x[i]=(((u16)buf[1]<<8)+buf[0]);
                tp_dev.y[i]=(((u16)buf[3]<<8)+buf[2]);
            }else
            {
                tp_dev.y[i]=((u16)buf[1]<<8)+buf[0];
```

```
                    tp_dev.x[i]=480-(((u16)buf[3]<<8)+buf[2]);
                }
            }
                    //printf("x[%d]:%d,y[%d]:%d\r\n",i,tp_dev.x[i],i,tp_dev.y[i]);
                }
            }
            res=1;
            if(tp_dev.x[0]>lcddev.width||tp_dev.y[0]>lcddev.height)//非法数据(坐标超出了)
            {
                if((mode&0XF)>1)        //有其他点有数据,则复第二个触点的数据到第一
个触点.
                {
                    tp_dev.x[0]=tp_dev.x[1];
                    tp_dev.y[0]=tp_dev.y[1];
                    t=0;                //触发一次,则会最少连续监测 10 次,从而提高命
中率
                }else                   //非法数据,则忽略此次数据(还原原来的)
                {
                    tp_dev.x[0]=tp_dev.x[4];
                    tp_dev.y[0]=tp_dev.y[4];
                    mode=0X80;
                    tp_dev.sta=tempsta;    //恢复 tp_dev.sta
                }
            }else t=0;                  //触发一次,则会最少连续监测 10 次,从而提高命
中率
        }
    }
    if((mode&0X8F)==0X80)//无触摸点按下
    {
        if(tp_dev.sta&TP_PRES_DOWN)  //之前是被按下的
        {
            tp_dev.sta&=~(1<<7); //标记按键松开
        }else                       //之前就没有被按下
        {
            tp_dev.x[0]=0xffff;
            tp_dev.y[0]=0xffff;
            tp_dev.sta&=0XE0;      //清除点有效标记
        }
    }
    if(t>240)t=10;//重新从 10 开始计数
    return res;
}
```

读取按键函数

void read_cb_btn(struct _gt_indev_drv_s * indev_drv, gt_indev_data_st * data)
此函数与读触摸函数类似,需要实现读取按键的函数。如果有按键被按下则将 data->state 设置为 GT_INDEV_STATE_PRESSED，同时将 btn_id 设置为按键键值。

```
void read_cb_btn(struct _gt_indev_drv_s * indev_drv, gt_indev_data_st * data)
{
    uint8_t status = 0;
    status = KEY_Scan(1);
    if( status ){
        data->btn_id = status;
        data->state = GT_INDEV_STATE_PRESSED;
    }else{
        data->state = GT_INDEV_STATE_RELEASED;
    }
}
```

SPI 读写函数
uint32_t spi_wr(uint8_t * data_write, uint32t len_write, uint8_t * data_read, uint32_t len_read)
参数说明:data_write 此指针存放着 24 位的地址值。*(data_write+1)为高八位地址
*(data_write+2)为中八位地址
*(data_write+3)为第八位地址

len_write 参数无需用户适配
data_read 为存放读取数据的指针
len_read 为读取数据的长度

```
uint32_t spi_wr(uint8_t * data_write, uint32_t len_write, uint8_t * data_read, uint32_t len_read)
{
    unsigned long ReadAddr;

   ReadAddr = *(data_write+1)<<16;                    //高八位地址
   ReadAddr +=  *(data_write+2)<<8;                       //中八位地址
   ReadAddr +=  *(data_write+3);                          //低八位地址

   r_dat_bat(ReadAddr,len_read,data_read);
}
```

其余实现方式只需要能读取在指定的 24 位地址读取指定数据的长度并存入 data_read 中均可。

定时器配置
GUI 需要一个 1ms 一次的心跳脉冲。故而可以通过定时中断来实现。

```
//定时器 3 中断服务程序
void TIM3_IRQHandler(void)
```

```
{
    if(TIM3->SR&0X0001)//溢出中断
    {
        gt_tick_inc(1);
    }
    TIM3->SR&=~(1<<0);//清除中断标志位
}
//通用定时器 3 中断初始化
//这里时钟选择为 APB1 的 2 倍，而 APB1 为 48M
//arr：自动重装值。
//psc：时钟预分频数
//定时器溢出时间计算方法:Tout=((arr+1)*(psc+1))/Ft us.
//Ft=定时器工作频率,单位:Mhz
//这里使用的是定时器 3!
void TIM3_Int_Init(u16 arr,u16 psc)
{
    RCC->APB1ENR|=1<<1;    //TIM3 时钟使能
    TIM3->ARR=arr;  //设定计数器自动重装值
    TIM3->PSC=psc;  //预分频器
    TIM3->DIER|=1<<0;  //允许更新中断
    TIM3->CR1|=0x01;   //使能定时器 3
  MY_NVIC_Init(1,3,TIM3_IRQn,2);      //抢占 1，子优先级 3，组 2

}
```

在定时中断的终端服务函数中加入 GUI 的心跳脉冲。gt_tick_inc(1)此函数声明在 gt_hal_tick.h 头文件中。在初始化硬件的时候，将 TIM3 的定时中断定为 1ms 触发一次。

定时器的自动重装值和时钟预分频数需要根据各自定时器配置。
TIM3_Int_Init(10-1,9000-1);   //1ms 定时中断
程序中仅供参考。

SDRAM 配置
如果 MCU 内部 RAM 无法满足 GUI 刷屏数组的定义，则需要将 gt_port_disp.c 中的刷屏数组定义到 SRAM 或者 SDRAM 中，地址需要根据项目对 SDRAM 的使用情况自由调整。

```
#elif(GT_REFRESH_STYLE == GT_REFRESH_STYLE_3)
  static gt_color_t buf_all[GT_SCREEN_WIDTH * GT_SCREEN_HEIGHT] __attribute__((at(0xC0000000+0X3FC00)));
#endif
```

使用 __attribute__ 关键字修饰该数组。

其余配置，如果需要使用 GUI 的 log 打印功能，则需要用户自己实现 printf 函数，使其能在串口格式化打印内容。

8.示例

8.1 串口示例

8.1.1 串口驱动实现

GUI 可以支持使用串口来控制各个控件，实现串口屏的需求。若想实现用串口操作控件，需要先实现串口的正常收发。

8.1.2 协议制定

实现串口的正常收发之后，可以通过制定不同的通信协议来实现不同的人机交互效果。下图为简单和复杂的两种协议制定示例:

数据头	数据长度	功能选择
0xAA	0x03	0x01:车载界面 0x02:洗衣机 0x03: 3D打印

	A	B	C	D	E	F	G	H	I	J	K	L	M
	数据头	数据长度	A/C功能	AUTO功能	加湿功能	左座椅加热	右座椅加热	空气循环	左侧温度	右侧温度	风速	返回	校验位
	0xAB	0x13	0x01:打开 0x00:关闭	0x01:打开 0x00:关闭	0x01:打开 0x00:关闭	0x01:打开 0x00:关闭	0x01:打开 0x00:关闭	0x01:打开 0x00:关闭	数值 0x10-0x1E	数值 0x10-0x1E	数值 0x00-0x0A	0x00不返回 0x01返回	0x00

对应 GUI 示例工程中的两个界面

8.1.3 解析协议并操作控件

之后只需要对应的 UI 界面实现对应的协议解析代码。通过解析不同的协议值，来改变显示的 UI 界面。

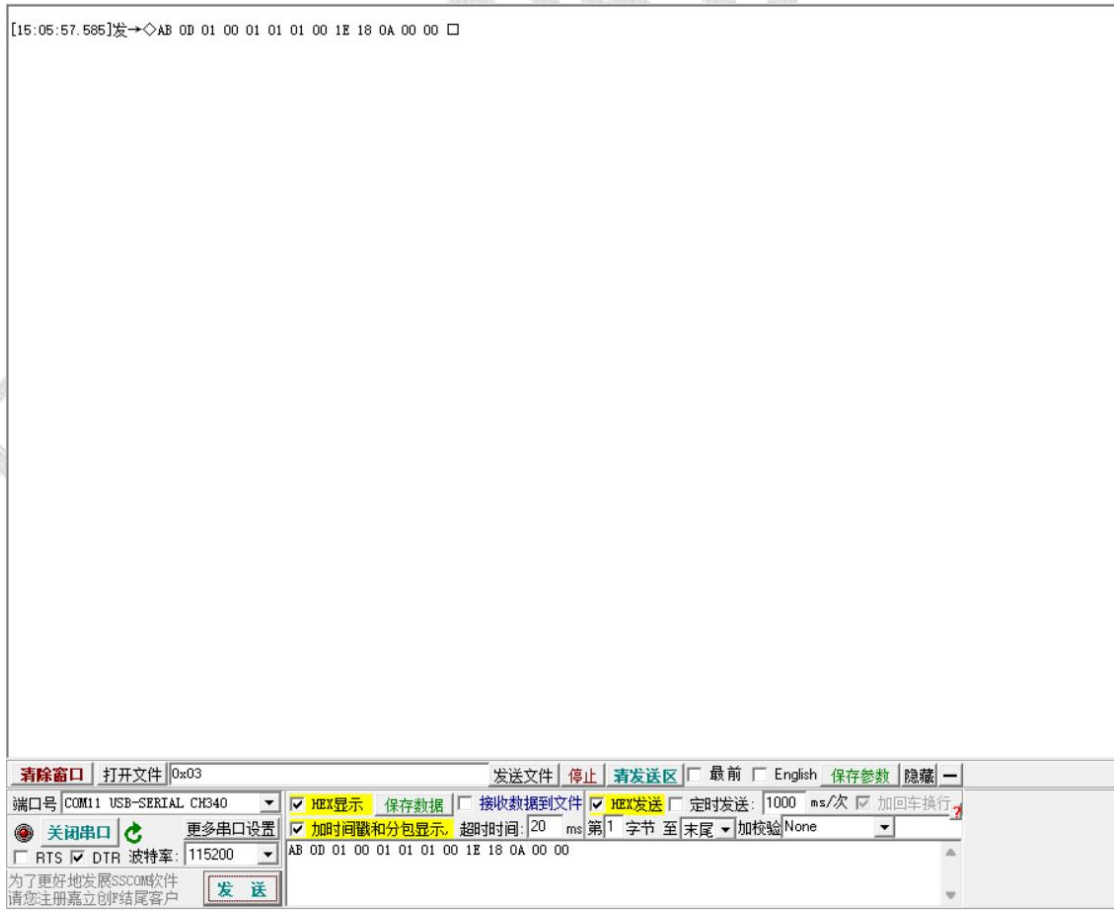
如图为车载界面协议解析示例:

```
//车载界面协议:数据头AB+长度0x0D+6个功能状态(0x00或0x01)+3个数值(左右温度和风速)+是否返回(01)+校验位
else if(_uart3_rx_buf[0] == 0xAB && _uart3_rx_buf[1] == 0x0D)
{
    memcpy(data,_uart3_rx_buf,13);
    car_data(data);
}
```

在串口线程中读取到 0xAB 的数据头后将 13 个数据长度发送到解析函数

```
/**
 * @brief 车载协议解析
 * 数据头0xAA 长度 A/C AUTO 加湿 左加热 右加热 空气循环 左温度 右温度 风控 返回 校验位
 */
void car_data(uint8_t *data)
{
    //   A/C
    if(data[2]==1){
        acstate = 0;
        gt_img_style_set_src(ac, "f:1_p.bmp");
    }else{
        acstate = 1;
        gt_img_style_set_src(ac, "f:1_84x63(122x417).bmp");
    }

    //  AUTO
    if(data[3]==1){
        autostate = 0;
        gt_img_style_set_src(automode, "f:2_p.bmp");
    }else{
        autostate = 1;
        gt_img_style_set_src(automode, "f:2_84x63(219x417).bmp");
    }

    //  加湿
    if(data[4]==1){
        defoggstate = 0;
        gt_img_style_set_src(defogging, "f:3_p.bmp");
    }else{
        defoggstate = 1;
        gt_img_style_set_src(defogging, "f:3_84x63(316x417).bmp");
    }
```

在解析函数中对协议中每一位做不同的处理。



在示例中通过串口发送打开 AC 关闭 AUTO 打开加湿 左右座椅加热 左侧温度 30 右侧温度 24，风速为 10，不返回的一条通信协议。
此时 UI 对协议做出反应:

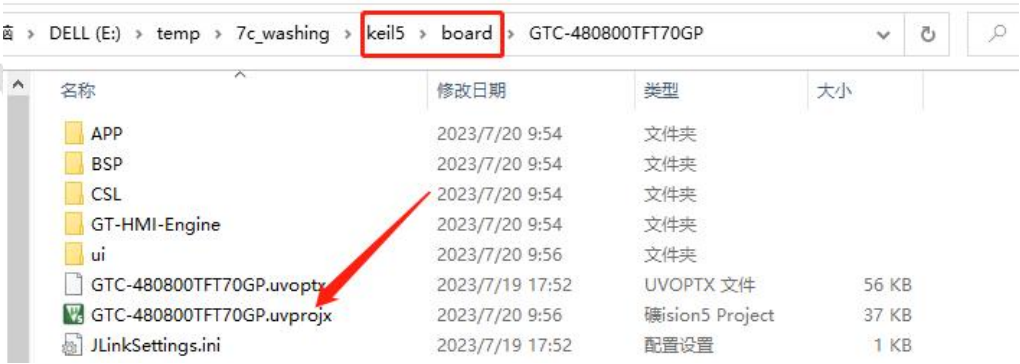
GT_GUI 的协议支持用户自己定义并灵活实现。本例仅供参考。

8.2 示例使用注意事项

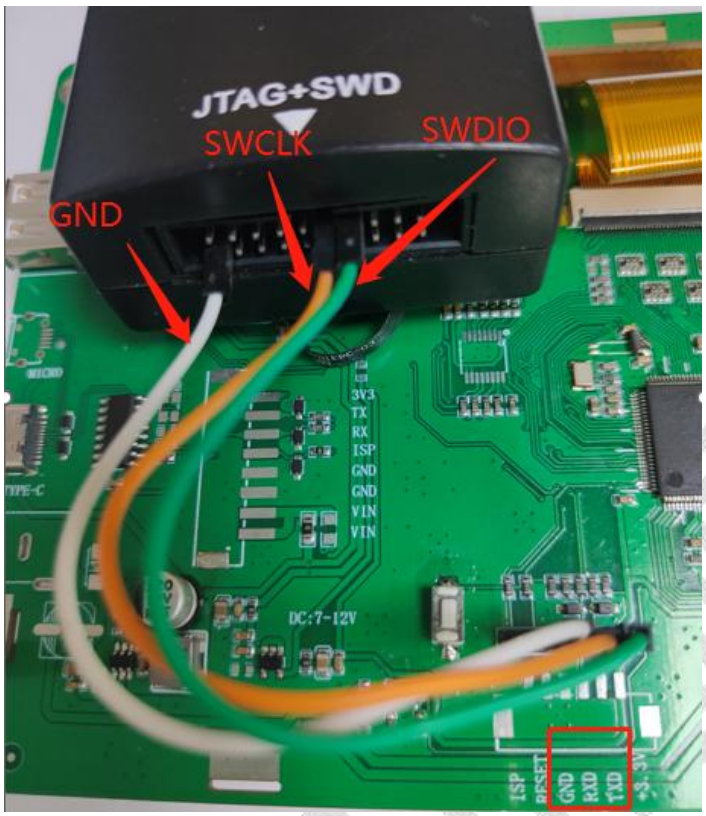
8.2.1.示例下载

打开 GT-HMI Designer 的软件，选择相应的示例工程打开，在进行仿真编译前另存为到自定义路径，务必设置好对应的编译环境板卡选项与字库配置，确保编译产物正确生成。

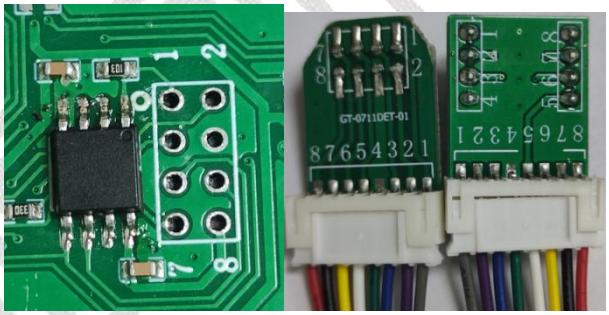
本章以 GT-HMI7 寸模块演示，打开另存为的自定义工程路径，直接打开 keil5 文件夹中 board 文件夹内的 keil 工程，keil 软件没有 Pack 包支持时请登录华芯微特官网下载，网址：https://www.synwit.cn/wendang455/。

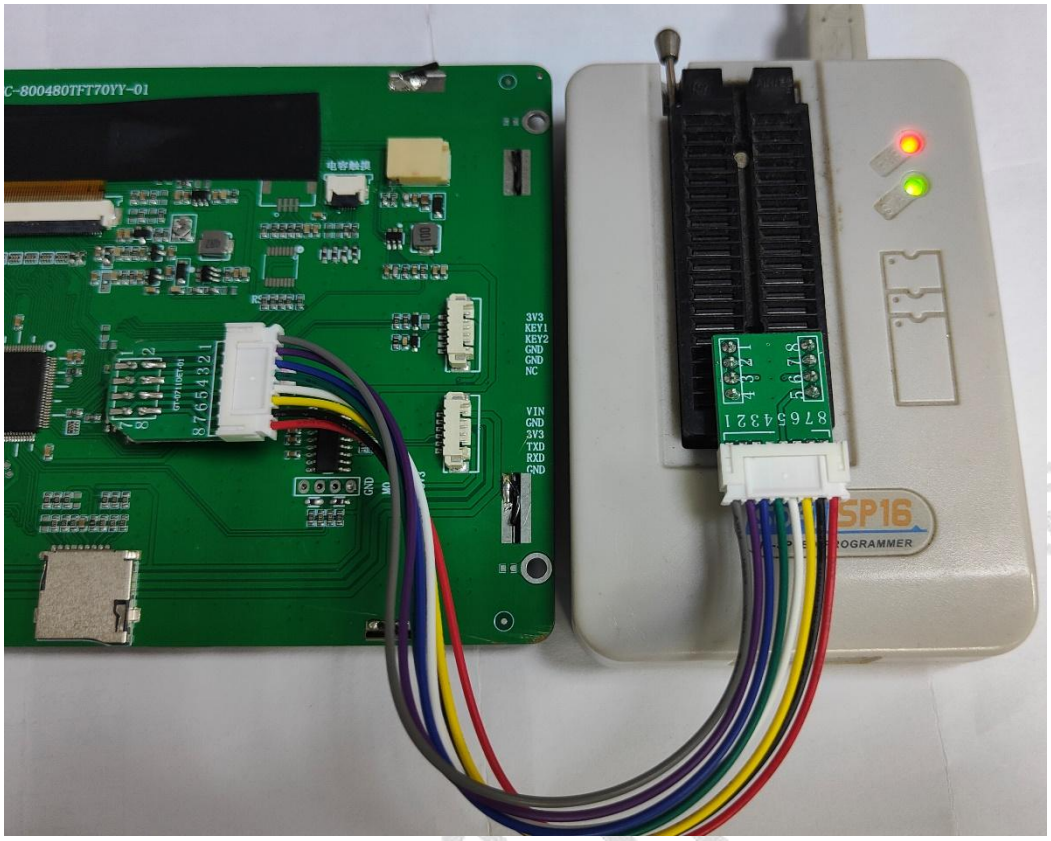


点击 keil 软件左边的编译按钮开始编译，编译完成后再点击右边的下载按钮功能，将示例代码下载到开发板当中，板子端口与烧录器连接请参考相应模块手册，下图为 7 寸模块板与 J-LINK 以 SWD 方式连接。



之后连接烧录器与板子，打开烧录器软件，加载 board 文件夹中的资源 bin 文件，将其烧录到板子上的 flash 中，以下为 7 寸模块 flash 端口，连接转换线与烧录器的连接图。

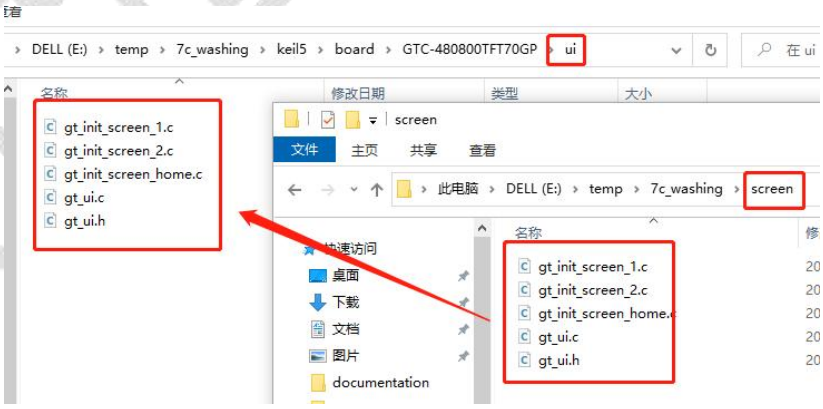




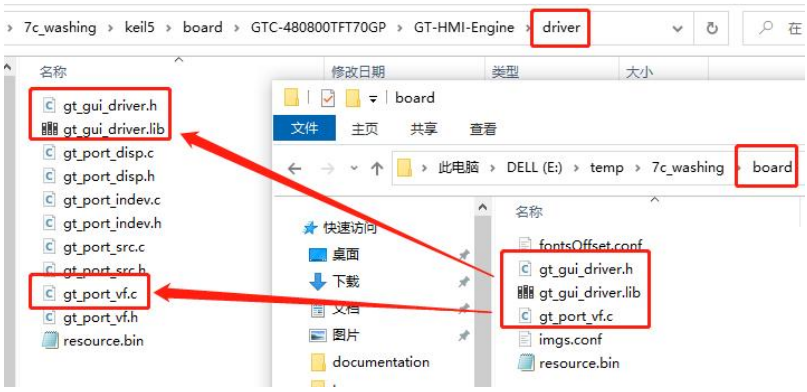
8.2.2.工程修改移植

如果已经在 keil 工程上实现多种第三方功能，在工程修改后不便于使用编译时新生成的 keil 工程，则可以对旧工程进行移植操作，移植过程与非 GT-HMI 模块类似，替换掉 screen 中生成的界面调用文件和 board 中生成的资源文件代码，重新烧写素材 bin 文件。具体操作如下：

第一步：替换掉 keil5 工程中 ui 目录下的 gt_init_screen_xx.c 文件。



第二步：替换掉 keil5 工程中 GT-HMI-Engine/driver 目录下的 gt_port_vf.c，gt_port_driver.lib 和 gt_port_driver.h 文件



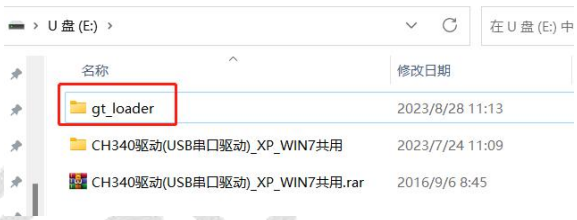
第三步：替换完成后重新编译下载程序到板子 mcu 中。

第四步：把 resource.bin 素材文件烧入 flash 中。

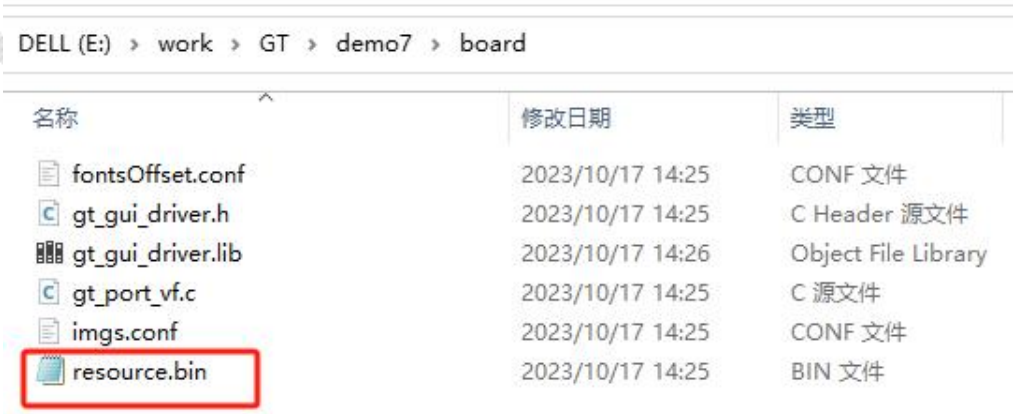
8.2.3.SD 卡升级

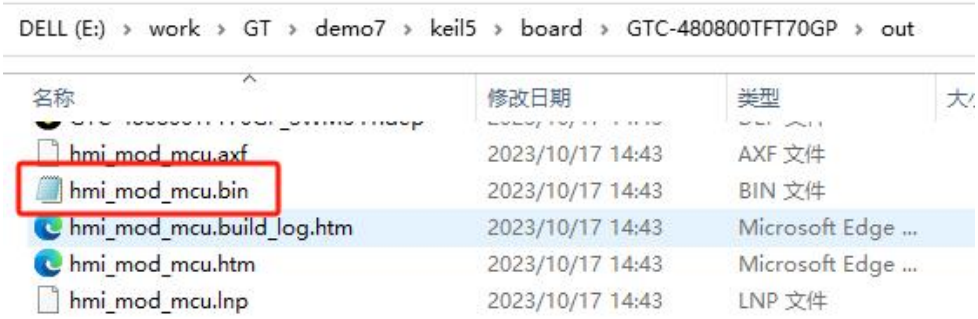
上述示例下载操作需要仿真器和烧录器的硬件支持，才能分别将程序文件与素材资源文件烧录到板子上，可以使用另一种方式 SD 卡进行升级。

- 准备 1 张 Mirco sd 卡（通称 TF 卡）,确保其容量足够存储固件文件。
- 在 SD 卡根目录中新建 gt_loader 文件夹，（注意名称小写保持一致，否则升级识别不到）



- 将 GT-HMI Designer 上位机软件仿真编译生成的工程路径下的\board\resource.bin 文件和 keil 工程编译生成的程序.bin 文件一起拷贝到 gt_loader 文件夹下，其中 resource.bin 文件是素材资源文件，升级到板载字库芯片当中，keil 生成的.bin 文件程序文件，升级到板载 MCU 当中，也可以只复制素材资源文件和程序文件其中一个文件，单独升级其中一个文件使用。

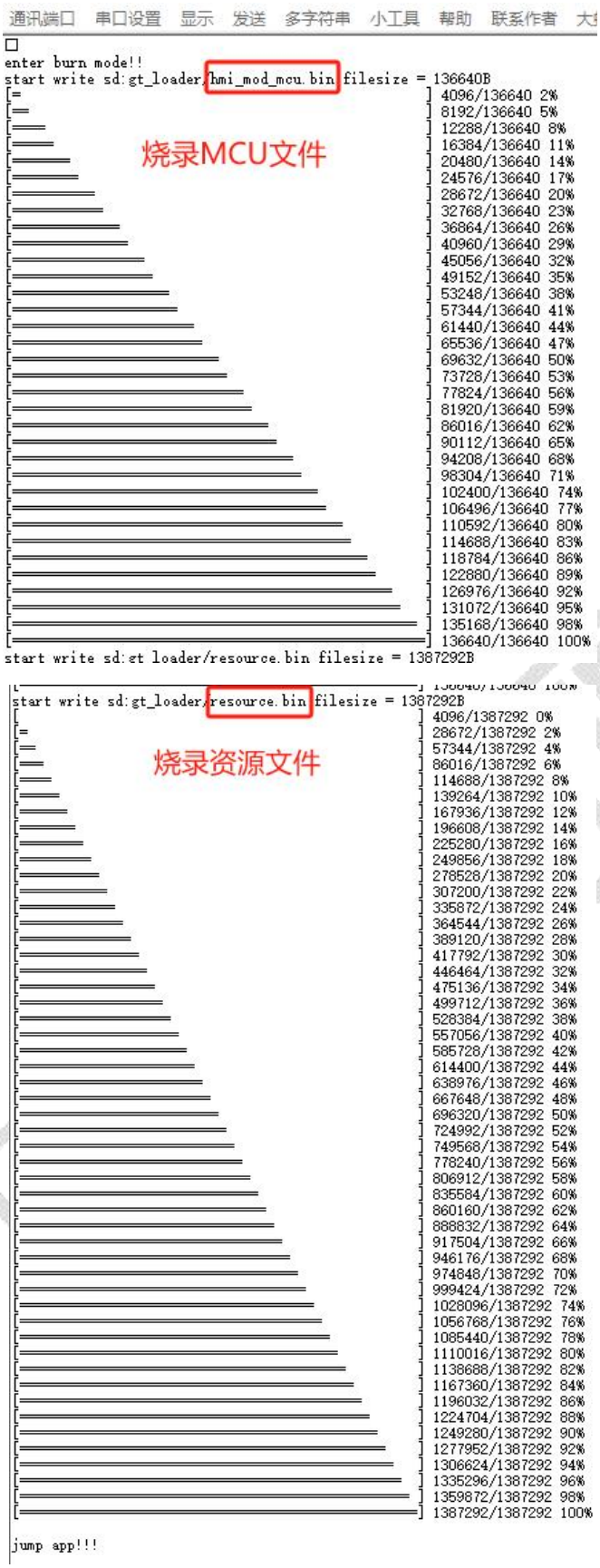




- 使用 4 寸，4.3 寸，7 寸或者 10.1 寸等 hmi 模块时 keil 工程所生成的 mcu 程序.bin 文件需要改名为 hmi_mod_mcu.bin。（注意名称小写保持一致，否则升级识别不到）

- 使用 0.9 寸，1.9 寸，2.8 寸，或者 3.5 寸等 GUI-LCD 时 mcu 程序.bin 文件需要改名为 gui_lcd_mcu.bin。（注意名称小写保持一致，否则升级识别不到）

- 从电脑弹出 SD 卡，插入板子中，按动板子上的 reset 按键开始升级，板子上面的灯开始闪烁代表正在升级中，或者使用 USB 数据线连接电脑打开串口助手查看升级打印信息(见下图)。升级完成后，灯停止闪烁，程序升级完成，拔出 Mirco sd 卡。


通讯端口 串口设置 显示 发送 多字符串 小工具 帮助 联系作者
enter burn mode!!
start write sd:gt_loader/hmi_mod_mcu.bin filesize = 136640B
烧录MCU文件
] 4096/136640 2%
] 8192/136640 5%
] 12288/136640 8%
] 16384/136640 11%
] 20480/136640 14%
] 24576/136640 17%
] 28672/136640 20%
] 32768/136640 23%
] 36864/136640 26%
] 40960/136640 29%
] 45056/136640 32%
] 49152/136640 35%
] 53248/136640 38%
] 57344/136640 41%
] 61440/136640 44%
] 65536/136640 47%
] 69632/136640 50%
] 73728/136640 53%
] 77824/136640 56%
] 81920/136640 59%
] 86016/136640 62%
] 90112/136640 65%
] 94208/136640 68%
] 98304/136640 71%
] 102400/136640 74%
] 106496/136640 77%
] 110592/136640 80%
] 114688/136640 83%
] 118784/136640 86%
] 122880/136640 89%
] 126976/136640 92%
] 131072/136640 95%
] 135168/136640 98%
] 136640/136640 100%
start write sd:gt_loader/resource.bin filesize = 1387292B
start write sd:gt_loader/resource.bin filesize = 1387292B
烧录资源文件
] 4096/1387292 0%
] 28672/1387292 2%
] 57344/1387292 4%
] 86016/1387292 6%
] 114688/1387292 8%
] 139264/1387292 10%
] 167936/1387292 12%
] 196608/1387292 14%
] 225280/1387292 16%
] 249856/1387292 18%
] 278528/1387292 20%
] 307200/1387292 22%
] 335872/1387292 24%
] 364544/1387292 26%
] 389120/1387292 28%
] 417792/1387292 30%
] 446464/1387292 32%
] 475136/1387292 34%
] 499712/1387292 36%
] 528384/1387292 38%
] 557056/1387292 40%
] 585728/1387292 42%
] 614400/1387292 44%
] 638976/1387292 46%
] 667648/1387292 48%
] 696320/1387292 50%
] 724992/1387292 52%
] 749568/1387292 54%
] 778240/1387292 56%
] 806912/1387292 58%
] 835584/1387292 60%
] 860160/1387292 62%
] 888832/1387292 64%
] 917504/1387292 66%
] 946176/1387292 68%
] 974848/1387292 70%
] 999424/1387292 72%
] 1028096/1387292 74%
] 1056768/1387292 76%
] 1085440/1387292 78%
] 1110016/1387292 80%
] 1138688/1387292 82%
] 1167360/1387292 84%
] 1196032/1387292 86%
] 1224704/1387292 88%
] 1249280/1387292 90%
] 1277952/1387292 92%
] 1306624/1387292 94%
] 1335296/1387292 96%
] 1359872/1387292 98%
] 1387292/1387292 100%
jump app!!!

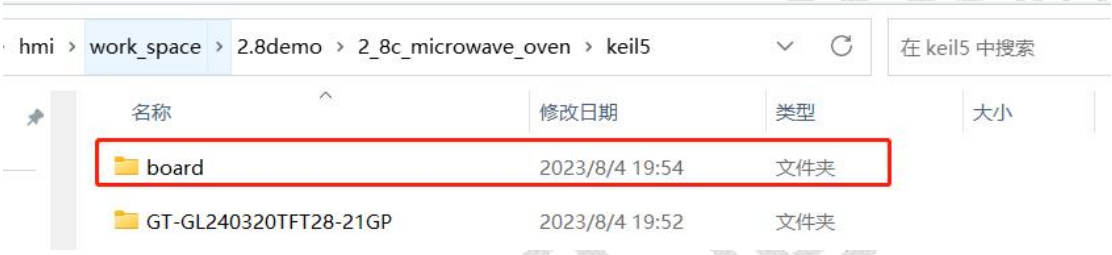
9 常见问题解答

(1) 素材一般都选择烧写在 flash 中，以数组的形式存放在数组里面会占用大量的内存资源，这样会导致芯片内存资源很紧缺。

(2) 存储素材一般都选择 flash，而不是内存卡，因为芯片有 sfc 通信接口，读取数据速度比 SDIO 读内存卡速度快。

(3) 如果在之前做好的非 GT-HMI 模块工程文件基础做素材添加修改，需要替换掉代码调用文件并重新烧写 bin 文件，参考上述 7.2.2 工程修改移植，但必须使用 out 文件夹内生成的文件。

(4) 使用 GT-HMI 模块，可使用 hmi 工程下 board 目录里得 keil5 工程。



(5) 如果工程用到的图片素材很多，GT-HMI Designer 编译后需要等一会才会导出素材 bin 文件和数组文件。

(6) 在编译前需要设置开发板设置和字库设置，开发板设置主要是针对 hmi 模块，设置开发板设置会在 hmi 工程直接生成 keil5 工程，字库设置主要是导出字库库文件，字库设置需要指定 keil5 的路径。

(7) hmi 工程文件有 board 目录和 out 目录，board 目录是针对 hmi 模块的，可以使用 hmi 上所有的字体，如使用非 hmi 模块需要使用 out 目录下的编译产物。

(8) 如果在 hmi 上使用未购买字库，out 目录不会导出字库库文件。

10.版本更新日志

版本	更新内容	日期
V1.0	初始版本	2023-04-12
V1.1	添加控件事件	2023-04-17
V1.2	更新移植说明	2023-04-21
V1.3	新增幻灯片控件和计数器控件 更新程序移植说明及操作流程	2023-05-31
V1.4	新增时钟、倒计时控件;	2023-06-25
V1.5	增加工程编译设置对话框自动弹出提示效果,修复部分已知问题	2023-07-20
V1.6	增加一维码、二维码控件	2023-08-07
V1.7	修复已知 BUG	2023-09-01
V1.8	更新 SD 卡升级及屏幕过渡动画	2023-11-03
V1.9	新增滑窗控件	2023-11-30
V1..10	新增滚轮控件	2023-12-29
V1.11	新增对话框控件，滑动条增加分级吸附效果	2024-03-29
V1.12	新增 GIF 控件、手势返回功能	2024-04-30
V1.13	新增状态栏控件，文本控件增加单行省略和滚动显示方式	2024-05-31
V1.14	新增灰度字库	2024-06-28
V1.15	新增图表控件、布局控件	2024-08-07
V1.16	新增 Media Player 播放器流控件	2024-09-13
V1.17	更新封面	2024-10-29
V1.18	新增公共属性圆角半径，仅文本区控件，播放器控件可以使用 新增下拉框控件 时钟控件新增年月日星期属性 键盘控件新增自动隐藏属性， 新增键盘风格：下划线、删除线、粗体、斜体 新增首行缩进的控件：文本区控件、按钮控件、标签控件 新增公共事件：x 轴和 y 轴坐标变化事件；宽度和高度的变化事件；圆角变化事件；隐藏和显示事件 事件新增效果：步进效果，过渡效果 列表控件新增事件：位置变化事件、x 坐标变化事件、y 坐	2024-12-26

	标变化事件、隐藏/显示事件 开关控件新增装饰线自定义宽度和高度 新增字体风格/文本区字体风格	

11.联系信息

深圳高通半导体有限公司
地址：深圳市福田区车公庙泰然九路金润大厦 12C
电话： 0755-83453881 83453855
技术支持： www.hmi.gaotongfont.cn
Call：0755-83453881

高通 GT-HMI 交流群： 加入技术交流群一起学习!

12.版权说明

13.免责声明

- 1、对于本软件产品和服务，本公司仅作下述有限保证，该有限保证取代任何文档、包装，或其他资料中的任何其他明示或默示的保证（如果有）。本公司仅以“现有状况且包含所有错误”的形式提供相关的产品、软件或程序及任何支持服务，并仅保证：

（1）本软件应用所提供的产品和服务能基本符合本软件正式公布的要求；

（2）本软件应用所提供的相关产品和服务基本与本软件正式公布的服务承诺相符；

（3）本软件仅在合理范围内尽力解决本软件在提供产品和服务过程中所遇到的问题，但不排除本公司会在较长的时间内，对软件产生的部分问题，统一进行修复。

（4）在适用法律允许的最大范围内，本公司明确表示不提供任何其他类型的保证，不论是明示的或默示的，包括但不限于本软件的适用性、可靠性、准确性、完整性、无病毒以及无错误的任何明示或默示保证和责任。

- 2、在适用法律允许的最大范围内，本公司并不担保本软件所提供的产品和服务一定能满足用户的需求，也不担保提供的产品和服务不会被中断，并且对产品和服务的及时性、安全性、出错发生均不作任何担保。
- 3、在适用法律允许的最大范围内，本公司不就因用户使用本软件的产品和服务引起的，或在任何方面与本软件的产品和服务有关的任何意外的、非直接的、特殊的，或间接

的损害或请求承担任何责任。

- 4、在适用法律允许的最大范围内，本软件上的控件样式设计灵感均来源开源社区提供的风格样式，本公司并不承担客户在未同意本协议的情况下，使用本软件控件样式带来的原著作人的法律诉讼等任何责任。
- 5、每个用户都要对以其用户名进行的所有活动和事件负全责。对于由于用户自身原因导致用户账号、密码泄密或招致其他损失，深圳高通不承担任何责任。
- 6、深圳高通对于由于不可抗力所导致的用户数据丢失、服务停顿或其他错误，将不承担任何法律或经济责任。不可抗力情况包括但不限于：

（1）深圳高通的服务器或其他计算机或网络设施遭受电脑病毒破坏而致使用户资料受损；

（2）深圳高通的服务器或其他计算机或网络设施遭受电脑黑客入侵致使用户资料受损或被窃取；

（3）因计算机设施或操作系统软件本身固有的技术缺陷而引起的用户资料数据受损或被窃取；

（4）因维护、检修、升级系统的需要或相关的第三方为维护、检修、升级系统所使用的主机、电力设备、公共网络等相关配套设施需要而造成的故障、损失或错误；

（5）国家或地方法律法规或政府政策发生的变化以及司法机关或政府行政机关的要求造成的服务停止、中断或改变；

（6）其他无法预见、无法避免且无法克服的客观情况。

14.License 开源及免费许可

特此免费授予任何人获得本软件和相关文档文件（"软件"），以不受限制地处理本软件的权利，包括但不限于使用、复制、修改、合并、出版、分发、再许可和/或销售本软件的副本，并允许从本软件派生的软件的人获取这些权利，但须遵守以下条件：

- 上述版权声明和本许可声明应包含在所有复制或重要部分中。
- 本软件按“原样”提供，不作任何明示或暗示保证，包括但不限于适销性、适用性和非侵权性的保证。在任何情况下，作者或版权持有人均不应对任何索赔、损害或其他责任承担任何责任，无论是在合同、侵权或其他方面，由本软件引起或与之相关的，或与本软件使用或其他交互相关的，即使已被告知此类损失的可能性。
- 对于源代码和二进制代码的修改，应注明更改的日期和作者，并以明确的方式提示新的版本。
- 本许可证的条款和条件应受当地法律管辖，任何有关本许可证的争议均应受有管辖权的法院解决。
- 该软件是免费和开源的，你可以自由使用并对其进行修改，但这并不意味着作者或版权持有人对软件的质量承担责任。希望您在使用本软件时谨慎处理。

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

- The above copyright notice and this permission notice shall be included in all copies or

substantial portions of the Software.

- The software is offered on an "as is" basis, with no guarantee of any kind, express or implied, including but not limited to warranties of merchantability, fitness for a particular purpose, and non-infringement. In no event shall the authors or copyright holders be held liable for any claims, damages or other liabilities, whether in contract, tort or otherwise, arising from, out of, or in connection with the software or the use or other dealings in the software.
- For source and binary code modifications, the date and author of the change should be indicated and new versions prompted in an explicit manner.
- The terms and conditions of this permit shall be governed by local law and any dispute relating to this permit shall be settled by a court of competent jurisdiction.
- The software is free and open source, and you can freely use and modify it, but this does not mean that the author or copyright holder is responsible for the quality of the software. I hope that you will be careful when using this software.