

SOLGUI 用户手册

第一章 SOLGUI 介绍

SOLGUI

SOLGUI 是一种用于嵌入式应用的图形化人机交互支持软件。它为搭载小尺寸单色屏幕的嵌入式设备提供了一种高效易用的人机交互解决方案。SOLGUI 的设计是模块化和层次化的,由在不同层次中的不同模块组成。SOLGUI 适用于所有 CPU,因为它 100%由 ANSI 的 C 语言编写的。

SOLGUI 是开源的,可以在 [github](#) 上关注并获取更新。

git 地址: https://github.com/MaxwellXyao/SOLGUI_V2.git

SOLGUI_V2 新特性

1. 相比于 SOLGUI_V1, SOLGUI_V2 增强了代码间的逻辑层次,方便用户理解使用。
2. 键值输入添加 FIFO 模块,增强在异步事件中操作的准确性。
3. SOLGUI_V2 添加了三种新字体,并且可以与 uCGUI 的字库兼容,用户只需进行简单的修改即可向 SOLGUI_V2 中移植 uCGUI 的字体。
4. 更多控件支持,交互体验更丰富。
5. 更小的 RAM 和 ROM 占用, SOLGUI_V2 所有代码为非阻塞,运行时对系统时间的占用更少。

本文档的目的

本指南描述如何配置和在嵌入式应用中使用 SOLGUI_V2 用户界面。它也说明了软件的内部结构。

在文档中, SOLGUI_V2 会与 SOLGUI 混用。下文出现的 SOLGUI 即指代该软件最新版 SOLGUI_V2。

1.1 需求

要移植 SOLGUI_V2, 你的目标系统中应满足以下条件:

- 有一个 CPU (8/16/32/64 位)
- LCD 或 OLED 等任何类型和任何分辨率显示设备, 小尺寸效果更佳 ([2016-2-28]目前仅支持 12864 类的单色显示屏)
- 最少的 ROM 和 RAM
- 至多有 6 个按键 (如果要使用菜单框架)

储存需求取决于软件的哪些部分被使用以及你的目标编译程序的效率有多高, 我们无法提供精确的数值, 以下的典型配置可以提供参考:

不使用菜单框架（只调用至中间层函数进行简单显示）：

- ROM: 4~5K
- RAM: 1~1.5K

使用菜单框架及控件：

- ROM: 11~13K
- RAM: 5~6K

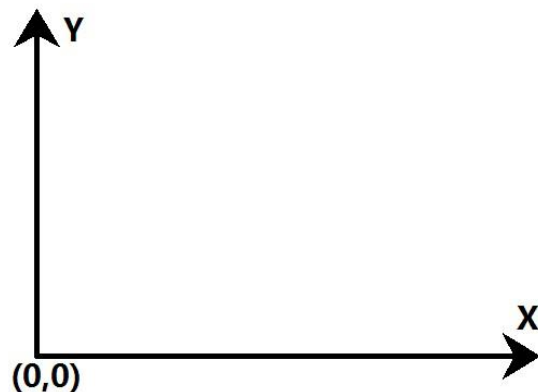
注意，如果你的应用程序需要显示许多波形的话，ROM 的需求将增加。以上所有的数值都是粗略的估计，不能得到保证。

1.2 SOLGUI_V2 特点

- 代码可根据不同的使用需求裁剪，不使用的模块可以配置关闭以节省空间。
- 代码对大小和速度都进行了优化。
- 清晰的结构。
- 所有函数均为非阻塞。
- 自带 4 种字体，用户可自行修改原有字符；字库兼容 uCGUI，只需简单修改即可移植新字体。
- 格式化字符串 SOLGUI_printf 作为屏幕输出函数，功能是 PC 上 printf 的一个子集。支持多种字体的十进制，二进制，十六进制的数值显示。
- 基础图形有多种显示参数可选（实线，点线，虚线，填充，反白等）。
- 选项支持卷轴式滚屏。
- 支持十进制整数和小数的输入和修改。
- 支持字符串的输入和修改。
- 支持谱和波的显示。
- 图像自动调整大小适应控件。

1.3 屏幕和坐标

屏幕由许多能够被单独控制的像素组成。



水平刻度被称作 X 轴，而垂直刻度被称作 Y 轴。一个二维坐标用 X 轴和 Y 轴坐标表示，即值(X,Y)。在程序中需要用到 X 和 Y 坐标时，X 坐标总在前面。

显示屏的左下角为默认的坐标 (0,0)。正的 X 值方向被总是向右；正的 Y 值方向总是向上。上图说明该坐标系和 X 轴和 Y 轴的方向。

另外需要说明的是，在 SOLGUI_V2 中，图形的放置参考点（定位点）也位于左下角。如下图



1.4 数据类型

在大多数情况下，SOLGUI_V2 使用它自己的数据类型，如下表：

数据类型	定义	说明
u8	unsigned char	8 位无符号值
u16	unsigned int	16 位无符号值
u32	unsigned long	32 位无符号值
s8	signed char	8 位有符号值
s16	signed int	16 位有符号值
s32	signed long	32 位有符号值
boolean	enum{False=0,True=!False}	布尔类型

对绝大多数控制器而言，该设置将会正常工作。如果有需要，可以在头文件 SOLGUI_Type.h 中修改它们。

第二章 入门指南

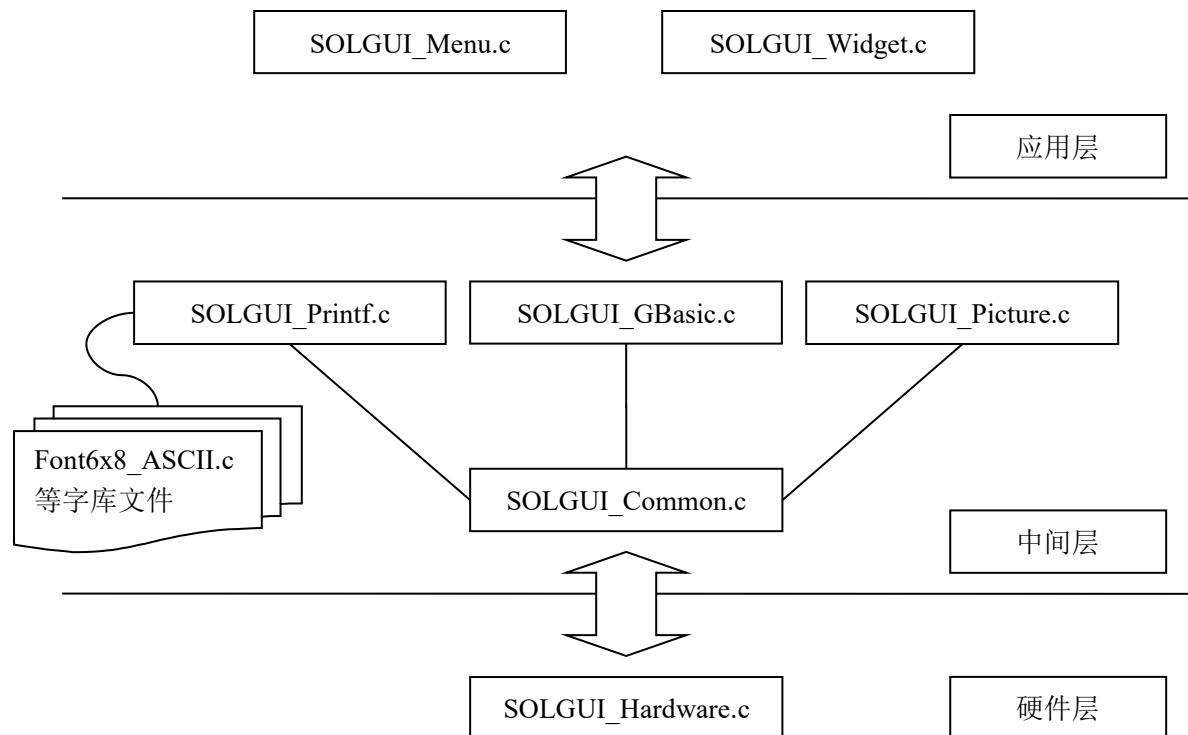
2.1 SOLGUI_V2 文件构成

SOLGUI_V2 的源文件存放在 SOLGUI 文件夹下。SOLGUI 文件夹下除了 SOLGUI_V2 的源文件还有一个 Font 文件夹，用于存放字库。Font 文件夹中的文件也是 SOLGUI_V2 组成的一部分。详细的文件说明：

文件名	路径	说明
README.txt	SOLGUI\	SOLGUI_V2 说明文件
SOLGUI_Common.c	SOLGUI\	共用函数库
SOLGUI_Common.h	SOLGUI\	共用函数库头文件
SOLGUI_Config.h	SOLGUI\	SOLGUI_V2 配置头文件
SOLGUI_GBasic.c	SOLGUI\	基础图形库
SOLGUI_GBasic.h	SOLGUI\	基础图形库头文件
SOLGUI_Hardware.c	SOLGUI\	硬件驱动移植
SOLGUI_Hardware.h	SOLGUI\	硬件驱动移植头文件
SOLGUI_Include.h	SOLGUI\	库索引头文件
SOLGUI_Menu.c	SOLGUI\	菜单框架
SOLGUI_Menu.h	SOLGUI\	菜单框架头文件
SOLGUI_Picture.c	SOLGUI\	图像支持库
SOLGUI_Picture.h	SOLGUI\	图像支持库头文件
SOLGUI_Printf.c	SOLGUI\	格式化字符显示库
SOLGUI_Printf.h	SOLGUI\	格式化字符显示库头文件
SOLGUI_Type.h	SOLGUI\	数据类型定义
SOLGUI_Widget.c	SOLGUI\	控件库
SOLGUI_Widget.h	SOLGUI\	控件库头文件
Font_Macro.h	SOLGUI\Font\	字体替换宏
Font4x6_ASCII.c	SOLGUI\Font\	4x6 字库
Font4x6_ASCII.h	SOLGUI\Font\	4x6 字库头文件
Font6x8_ASCII.c	SOLGUI\Font\	6x8 字库
Font6x8_ASCII.h	SOLGUI\Font\	6x8 字库头文件
Font8x8_ASCII.c	SOLGUI\Font\	8x8 字库
Font8x8_ASCII.h	SOLGUI\Font\	8x8 字库头文件
Font8x10_ASCII.c	SOLGUI\Font\	8x10 字库
Font8x10_ASCII.h	SOLGUI\Font\	8x10 字库头文件

2.2 SOLGUI_V2 代码结构

SOLGUI_V2 的代码结构如图：



SOLGUI_V2 所有代码共分为三层：硬件层，中间层，应用层。

- 硬件层：与屏幕硬件直接关联。`SOLGUI_Hardware.c` 用于将存放屏幕驱动的相关代码。
- 中间层：提供基本的字符，图形，图像等显示 API。该层可以被用户调用，用于构建单页面的数据显示界面。
- 应用层：为应用提供菜单框架，页面，控件等 API。该层可以被用户调用，用于构建复杂的多页面菜单式人机交互界面。

第三章 移植 SOLGUI_V2

这一章将介绍如何移植 SOLGUI_V2 至目标系统中，并完成配置，最后将完成在屏幕上输出 helloworld 的程序范例。

在此说明范例使用的开发环境及硬件参数：

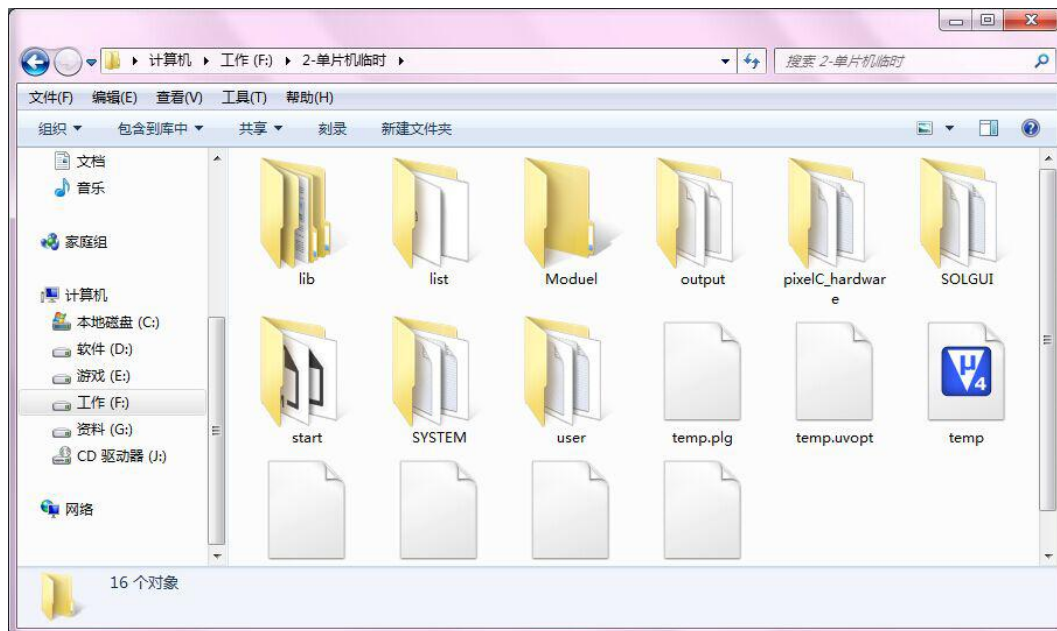
- IDE（集成开发环境）：KEIL MDK-ARM μ Vision V4.10
- 评估板：pixelC2 V1.0（STM32F103C8T6）
- 屏幕：OLED12864（单色）

完成本范例需要以下步骤：

- 源码加入工程文件夹下
- 在工程中加入 SOLGUI_V2
- 添加 SOLGUI_V2 的头文件路径
- 屏幕硬件驱动的移植

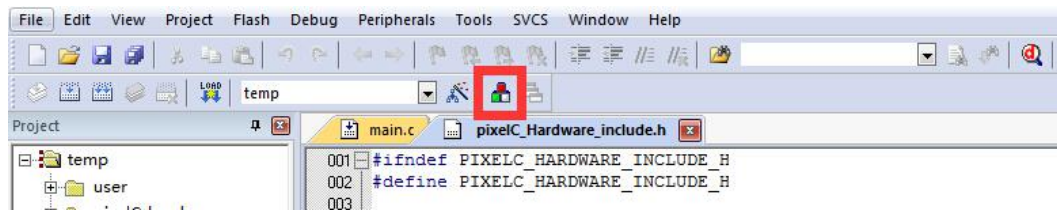
3.1 源码加入工程文件夹下

将存放有 SOLGUI_V2 所有源码的文件夹添加至工程文件夹下。推荐将 SOLGUI_V2 源码单独存放于一个文件夹中，这是一个好的习惯，这样方便代码管理。

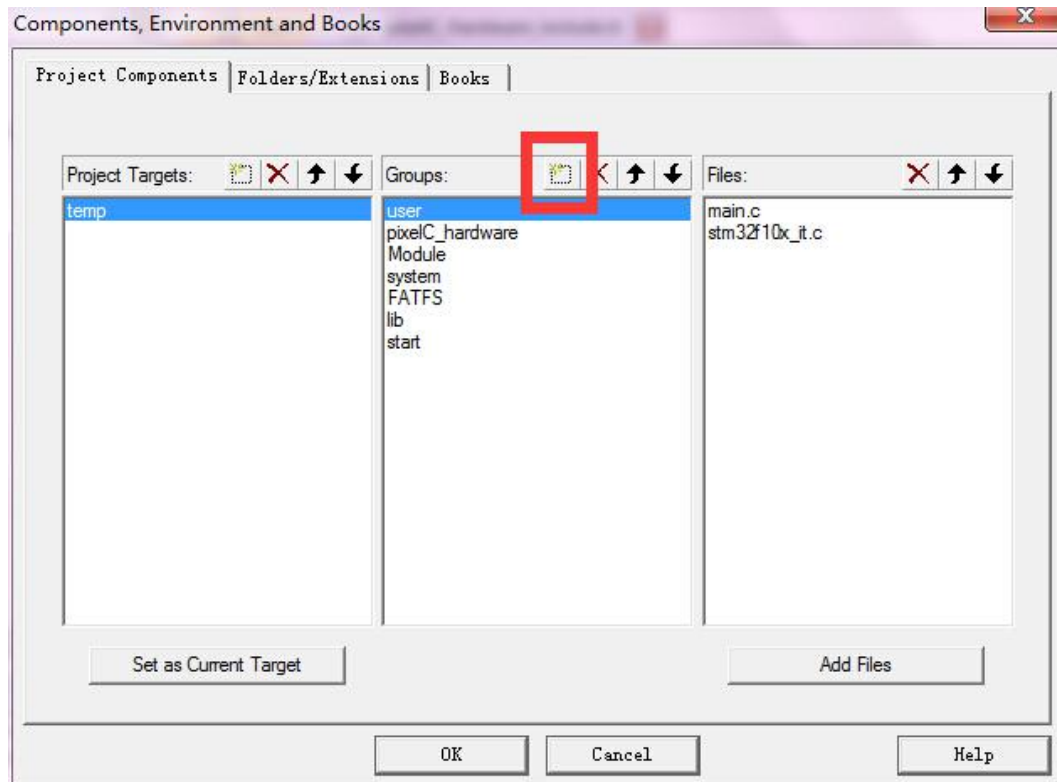


3.2 在工程中加入 SOLGUI_V2

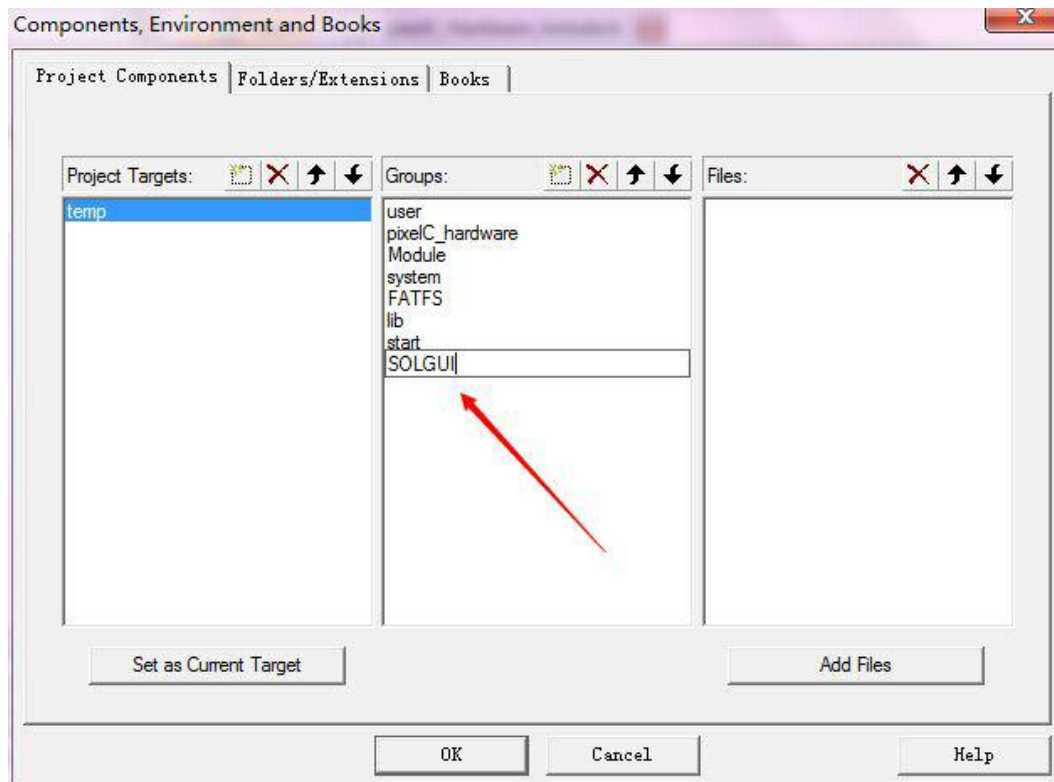
1. 点击文件管理图标



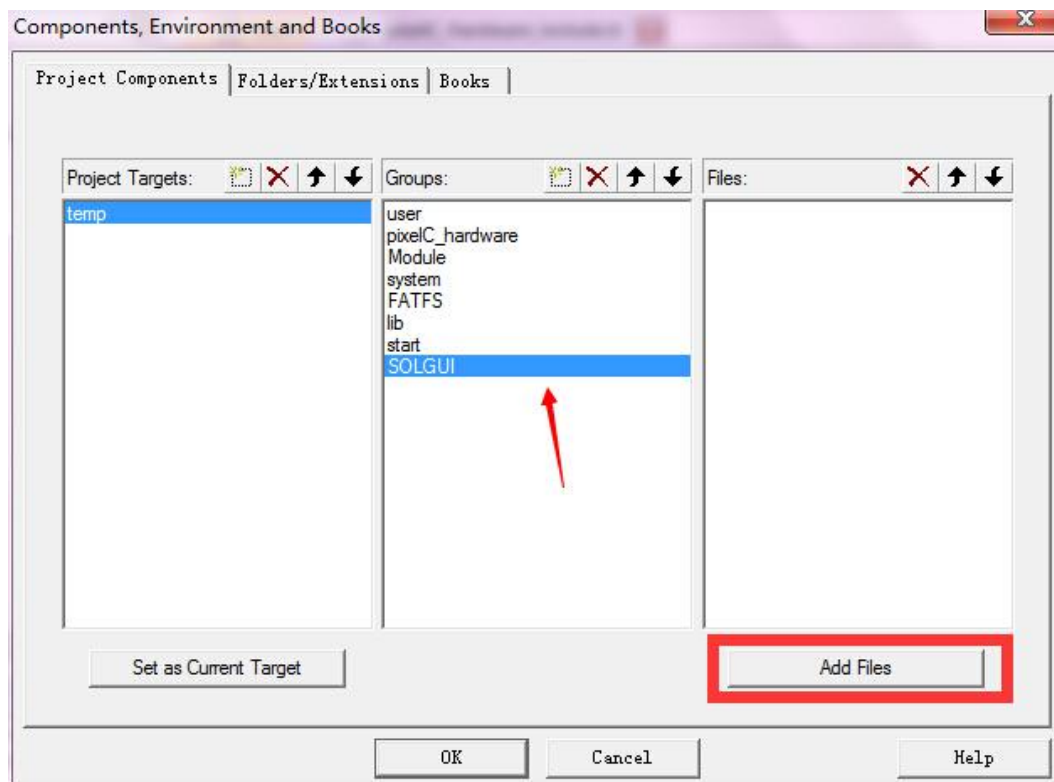
2. 新建一个 GROUP 组



3. 将新建的组命名为“SOLGUI”

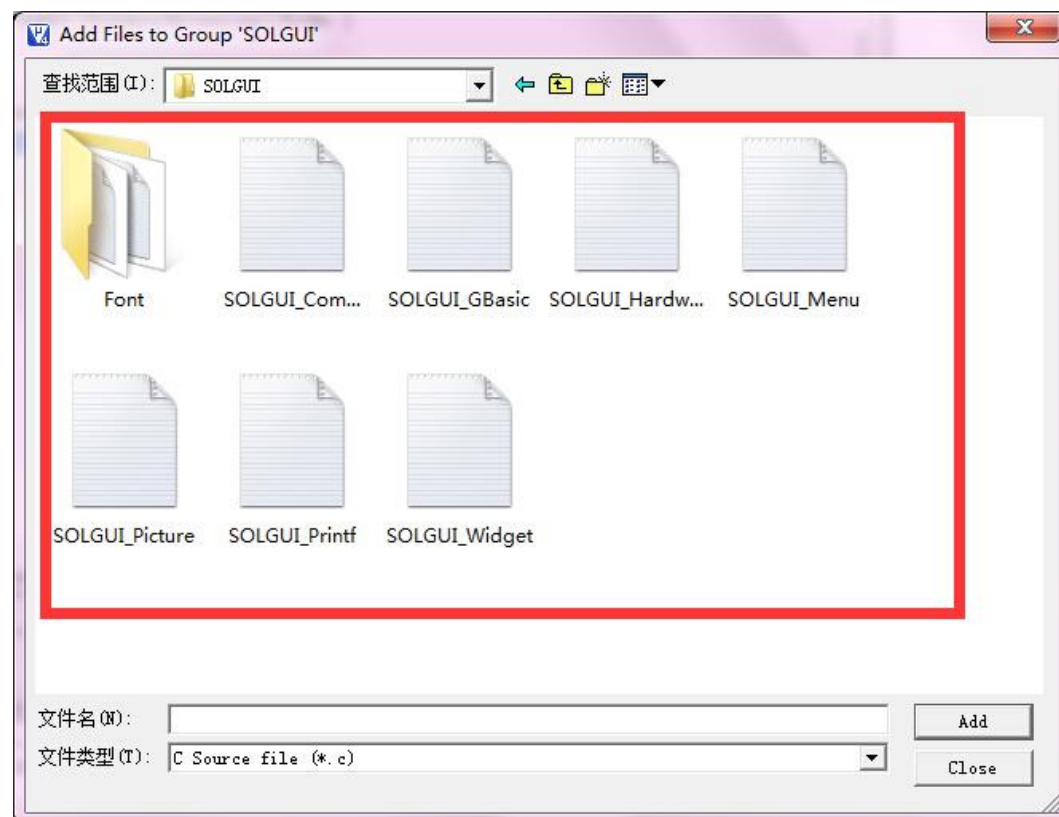


4. 在“SOLGUI”组下添加源文件

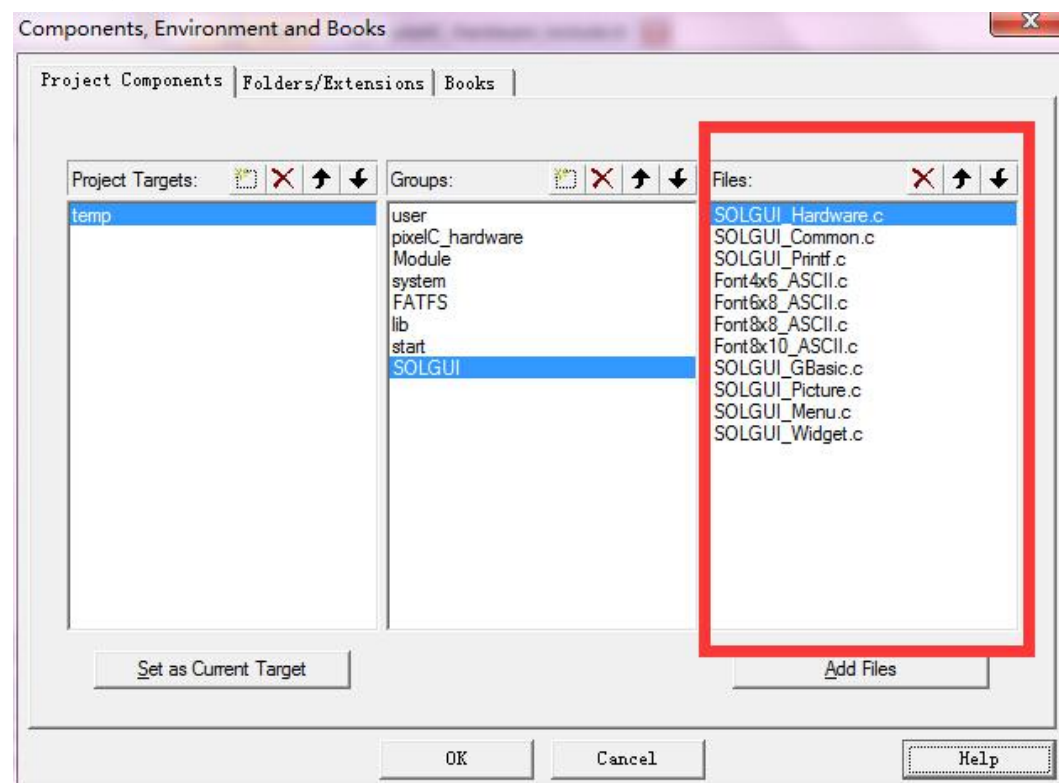


5. 找到工程文件夹下的 SOLGUI 文件夹，将文件夹中所有的源文件进行添加（包括 Font

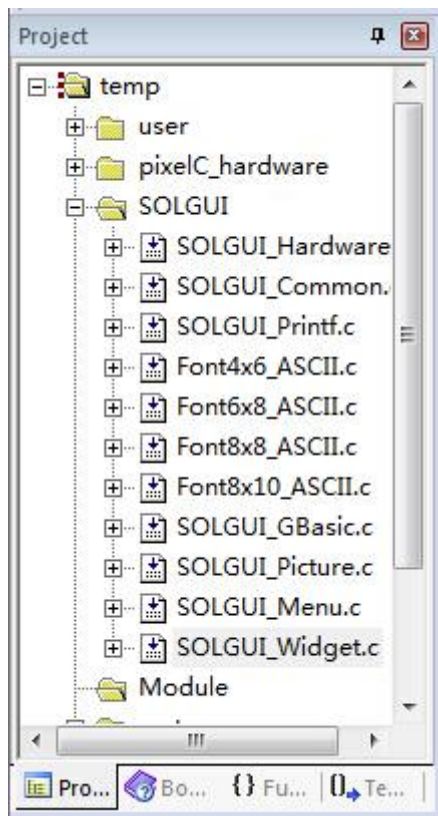
文件夹下的源文件)。



6. 添加完成，确认



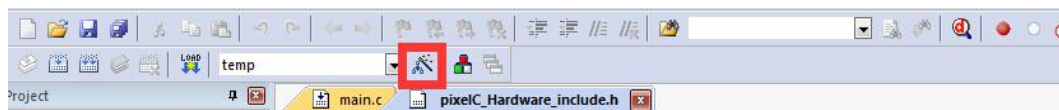
7. 在侧边的工程栏下，可以看到已经添加好的 SOLGUI_V2 源文件。



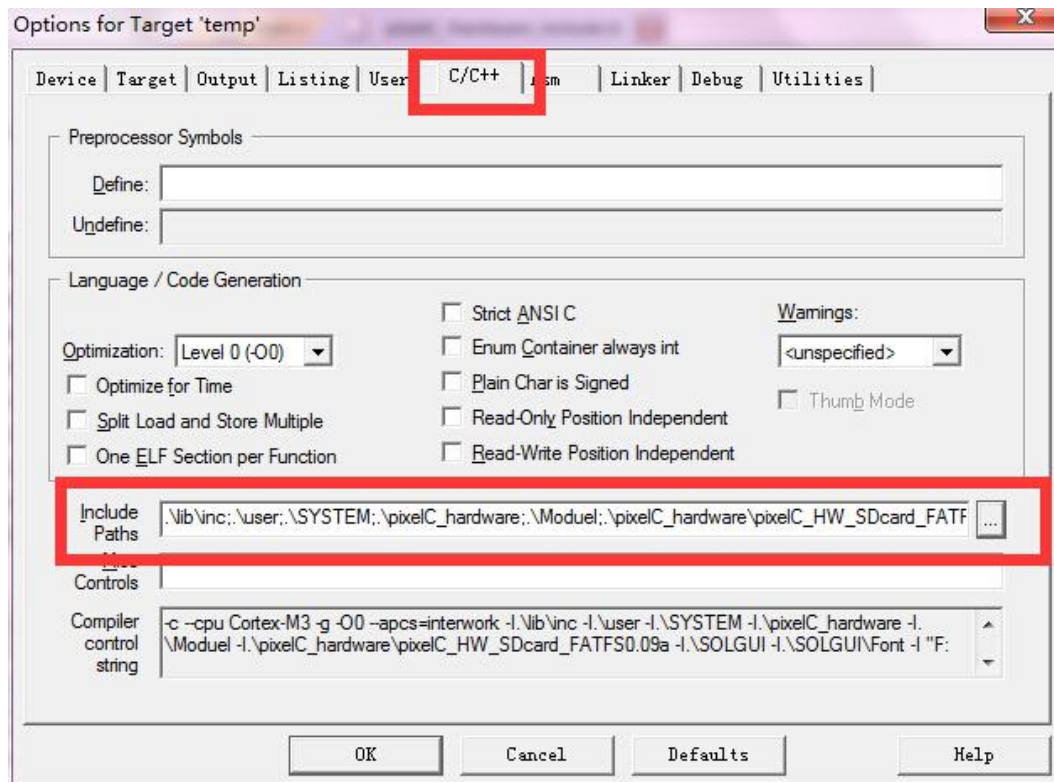
3.3 添加 SOLGUI_V2 的头文件路径

需要我们手动添加头文件路径以便编译器可以找到头文件。

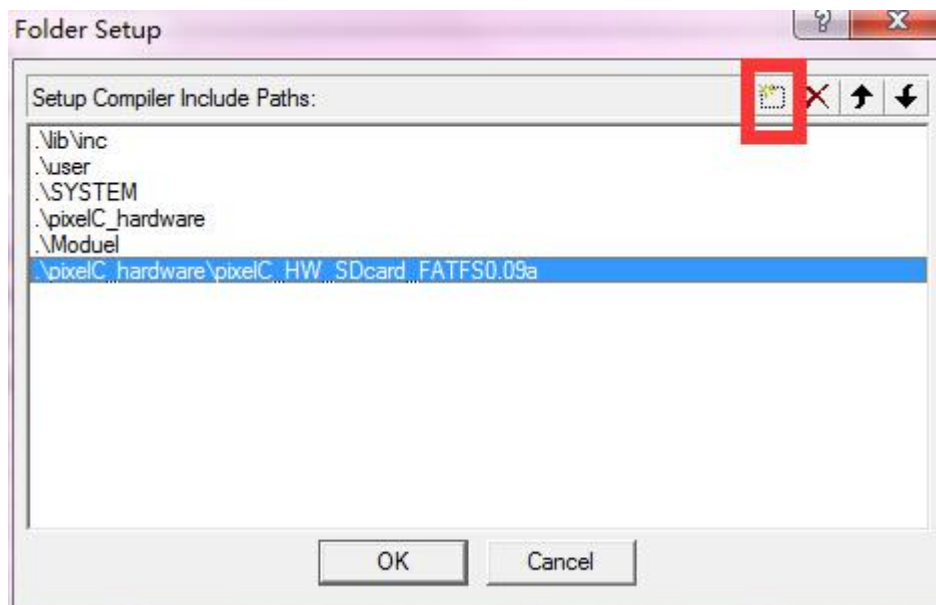
1. 点击属性图标



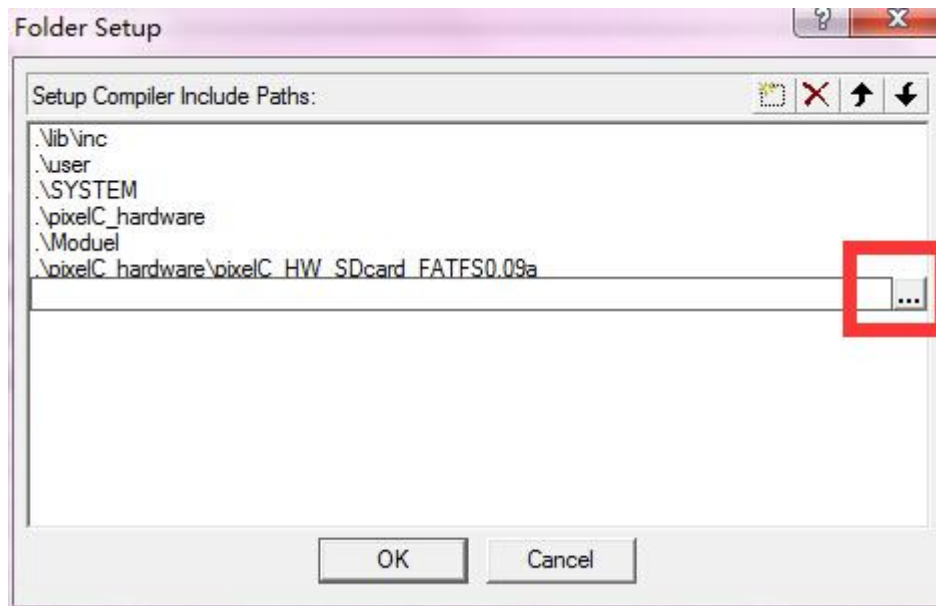
2. 选择 C/C++选项卡，点击下方的 Include Path 栏的右侧按钮添加路径



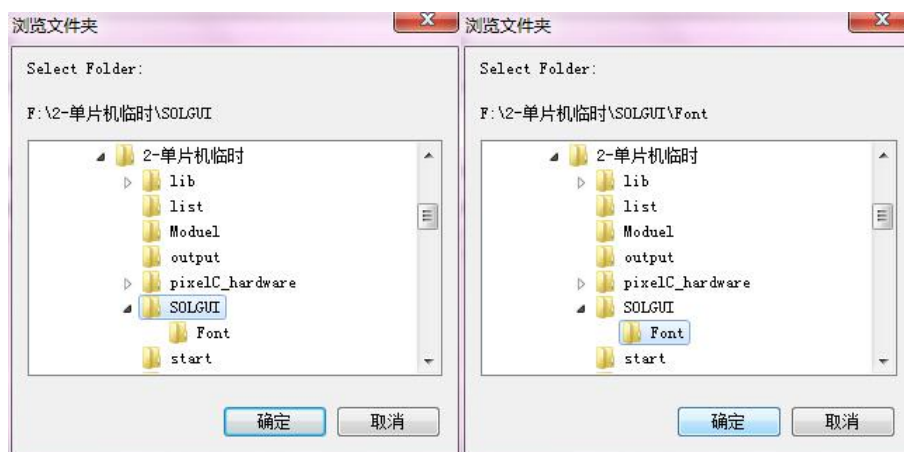
3. 新建一个文件路径



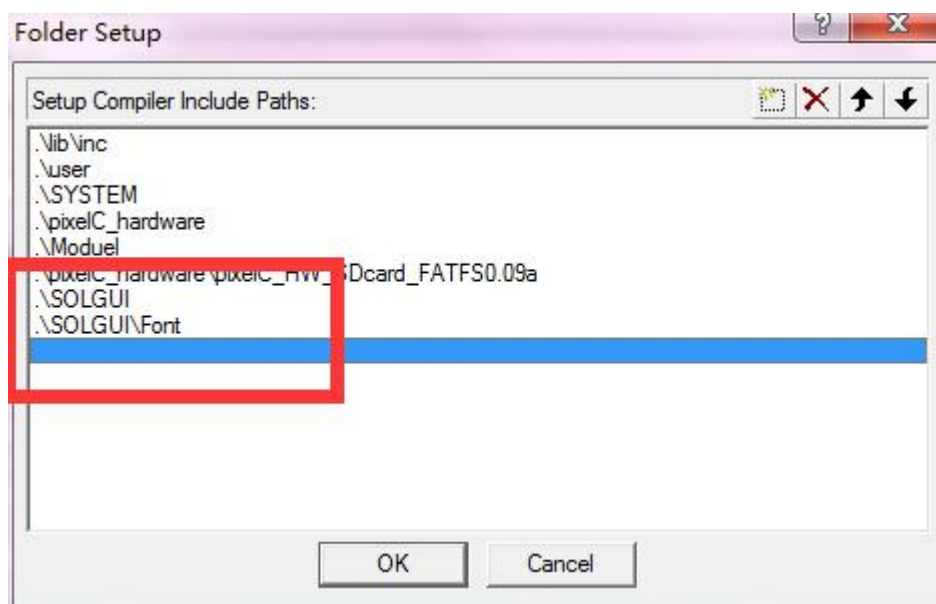
4. 点击添加路径



5. SOLGUI 和 SOLGUI\Font 两个文件夹路径都要添加，因此要再次重复上面的 3~4 步。



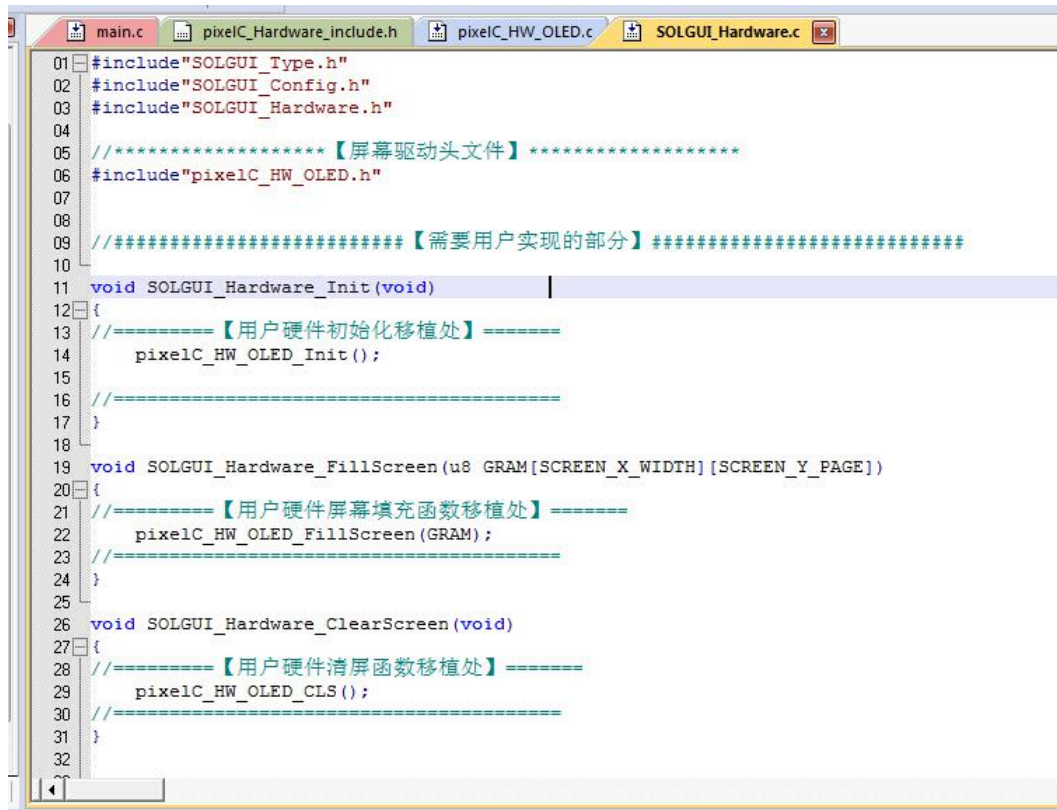
6. 添加完成两个路径，确认



至此我们就完成了 SOLGUI 在工程中部署。

3.4 屏幕硬件驱动的移植

现在开始将要进行的是程序的移植和编写。在此之前用户需确保屏幕驱动已经调试通过。打开 SOLGUI_Hardware.c 文件，如下



```
01 #include "SOLGUI_Type.h"
02 #include "SOLGUI_Config.h"
03 #include "SOLGUI_Hardware.h"
04
05 //*****【屏幕驱动头文件】*****
06 #include "pixelC_HW_OLED.h"
07
08 //*****【需要用户实现的部分】*****
09
10
11 void SOLGUI_Hardware_Init(void)
12 {
13     //=====【用户硬件初始化移植处】=====
14     pixelC_HW_OLED_Init();
15
16     //=====
17 }
18
19 void SOLGUI_Hardware_FillScreen(u8 GRAM[SCREEN_X_WIDTH][SCREEN_Y_PAGE])
20 {
21     //=====【用户硬件屏幕填充函数移植处】=====
22     pixelC_HW_OLED_FillScreen(GRAM);
23     //=====
24 }
25
26 void SOLGUI_Hardware_ClearScreen(void)
27 {
28     //=====【用户硬件清屏函数移植处】=====
29     pixelC_HW_OLED_CLS();
30     //=====
31 }
32
```

按照提示我们需要移植三个与硬件有关的函数：

- 用户硬件初始化：即屏幕硬件初始化函数。
- 用户硬件屏幕填充函数：这个需要根据屏幕实际情况来进行编写。SOLGUI_V2 使用一个二维数组作为屏幕显示缓存，用户需要在屏幕驱动中构造一个函数来将二维数组中的内容映射在屏幕上，可以形象的理解为将 GRAM 以类似图片的形式显示在屏幕上。GRAM[][] 的两个下标分别是屏幕的长宽像素数，需要在 SOLGUI_Config.h 文件中提前配置。（本范例配置为 SCREEN_X_WIDTH 128，SCREEN_Y_WIDTH 64。由于 OLED12864 是单色屏，因此取数组宽为 SCREEN_Y_PAGE (SCREEN_Y_WIDTH/8)，即每一个比特代表一个像素）。
- 用户硬件清屏函数：即屏幕硬件清屏函数。如果没有也可以不写，SOLGUI_V2 中会使用软件清屏来实现该功能。

其他的设置，用户需要打开 SOLGUI_Config.h，并在【硬件层】栏目下进行修改相应的参数来适配目标硬件。

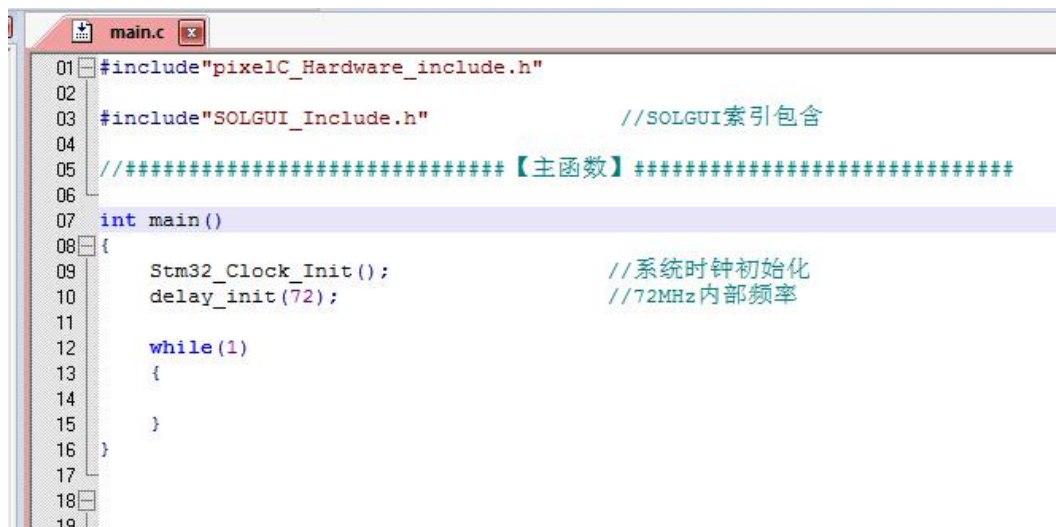

```

03 //*****【硬件层】*****
04 //-----【屏幕长宽像素数】-----
05 #define SCREEN_X_WIDTH      128      //屏幕的x轴像素数
06 #define SCREEN_Y_WIDTH      64       //屏幕的y轴像素数
07
08 //-----【屏幕长宽字数】-----
09 #define SCREEN_X_PAGE      (SCREEN_X_WIDTH/6)      //支持多少个6x8字宽（一般不需要修改，默认即可）
10 #define SCREEN_Y_PAGE      (SCREEN_Y_WIDTH/8)      //支持多少个6x8字高（一般不需要修改，默认即可）
11
12

```

3.5 程序编写

为了使用 SOLGUI_V2,用户需将库索引头文件 SOLGUI_Include.h 包含进代码中



```

main.c
01 #include "pixelC_Hardware_include.h"
02
03 #include "SOLGUI_Include.h"      //SOLGUI索引包含
04
05 //*****【主函数】*****
06
07 int main()
08 {
09     Stm32_Clock_Init();          //系统时钟初始化
10     delay_init(72);              //72MHz内部频率
11
12     while(1)
13     {
14
15     }
16 }
17
18
19

```

为了在屏幕上显示“Helloworld”，在代码中用户需要调用三条语句：

- ①SOLGUI_Init(): SOLGUI 初始化。SOLGUI_Init()在运行时会将硬件一同初始化，因此不必再重复初始化硬件；
- ②SOLGUI_printf(): 在屏幕缓存上格式化输出字符串。前面两个参数表示参考点（定位）的坐标；参数 F6X8 表示使用宽 6 像素高 8 像素的字体，SOLGUI_V2 中预设了 4 种字体，分别为 F4X6, F6X8, F8X8, F8X10；后面的参数就是格式化字符串。（具体使用请参见第五章 API）
- ③SOLGUI_Refresh(): 屏幕更新函数。由于输出图形都会预先写到屏幕缓存上，因此为了在屏幕上显示出字符，必须在完成全部绘图后调用该函数更新一次屏幕。

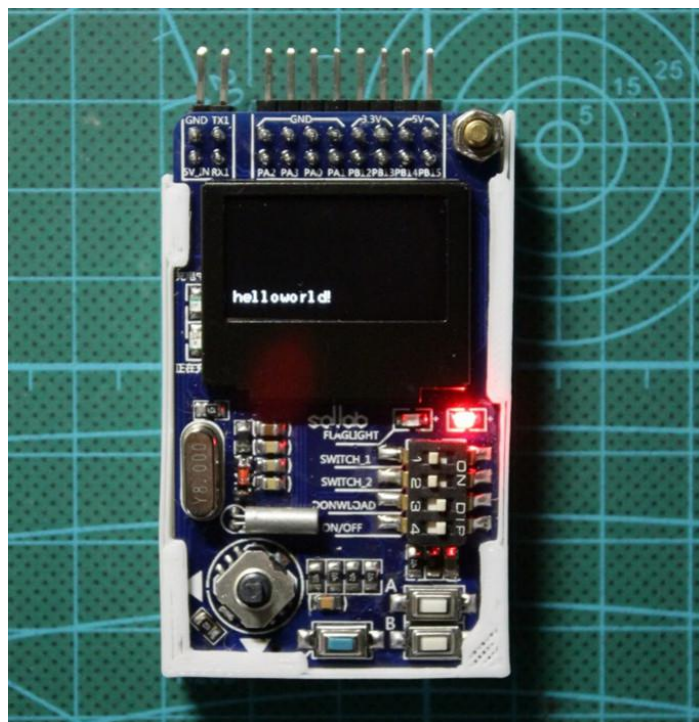


```

04 //*****【主函数】*****
05
06
07 int main()
08 {
09     Stm32_Clock_Init();          //系统时钟初始化
10     delay_init(72);              //72MHz内部频率
11
12     SOLGUI_Init();                //SOLGUI初始化 ①
13     SOLGUI_printf(0,0,F6X8,"helloworld!"); //②
14     SOLGUI_Refresh();             //屏幕更新 ③
15
16     while(1)
17     {
18
19     }
20 }
21

```

编译，下载到评估板中通电运行，效果如下。



3.6 额外的说明

- 使用和不使用菜单框架 SOLGUI 初始化函数 SOLGUI_Init() 会有不同的参数！此处是不使用菜单框架的情况，使用菜单框架与否可以在文件 SOLGUI_Config.h 中【应用层】栏目下修改（允许 1，不允许 0）。本例未使用菜单框架，该项置 0。

```
26
27
28 //*****【应用层】*****
29 //-----【SOLGUI菜单框架前台使用开关】
30 /*-----关闭后不能调用应用层中函数，即无法使用菜单框架-----*/
31 #define MENU_FRAME_EN 0 //允许SOLGUI菜单框架作为前台
32
33 //-----【Widget开关】
34 /*-----可根据实际使用情况开关，节约空间-----*/
35 #define WIDGET_GOTOPAGE_EN 1 //允许使用控件：GotoPage页面跳转
36 #define WIDGET_SPIN_EN 1 //允许使用控件：Spin数字旋钮
37 #define WIDGET_OPTIONTEXT_EN 1 //允许使用控件：OptionText选项文本
```

- 关于屏幕更新函数 SOLGUI_Refresh()：本例中的字符式静态显示，因此只需要单次调用屏幕更新函数；如果需要动态的更新屏幕信息，如数值变化，就必须周期性的调用。一个常见的方式是将该函数写在 while(1) 主循环中或在一个定时器中周期更新屏幕。

- 具体代码可见例程“1_在屏幕上输出 helloworld”

- 更详细的配置方法可以查看附录的“SOLGUI_Config.h 配置介绍”。

第四章 菜单及多页面

这一章将介绍如何在 SOLGUI_V2 下进行多页面菜单的编写及控件使用介绍。最后将完成一个多页面菜单的程序范例。

在此之前，假设用户已经完成第三章的移植工作，并且调试通过。本章的范例将基于第三章的工程进行修改得到。

4.1 对环境的要求——输入设备

菜单和多页面涉及到用户的输入，因此系统中必须至多有 6 个独立的按键以对应执行 6 个操作行为（此处假设每个按键输出唯一的键值。如果按键可以输出多个键值如短按、长按时，意味着实际可以用更少的按键进行控制）。SOLGUI_V2 操作菜单 6 个行为分别是：上，下，左，右，确认和返回。用户需要 SOLGUI_Config.h 文件中的【应用层】下的【系统返回的按键键值】配置预先设置触发行为的键值对应关系。

评估板 pixelC2 V1.0 上搭载有一个五向按键和三个独立按键，每个按键可以输出两个键值（短按，长按）。因此满足 SOLGUI_V2 对环境的要求。这里我们约定，五向按键按方向定义为上键，下键，左键，右键和中间的确认键（OK），取蓝色键帽的独立按键为返回键（BACK）。每个按键只取短按键值输入 SOLGUI_V2。因此，在评估板上的设置如下：

```
53 //-----【系统返回的按键键值】
54 /*-----用户需根据系统按键返回的键值来进行设定-----*/
55 #define SOLGUI_KEY_UP          0x50          //上键键值
56 #define SOLGUI_KEY_DOWN        0x20          //下键键值
57 #define SOLGUI_KEY_LEFT        0x30          //左键键值
58 #define SOLGUI_KEY_RIGHT       0x10          //右键键值
59 #define SOLGUI_KEY_OK          0x40          //OK键键值
60 #define SOLGUI_KEY_BACK        0x60          //返回键键值
61
62
```

4.2 相关概念及约定

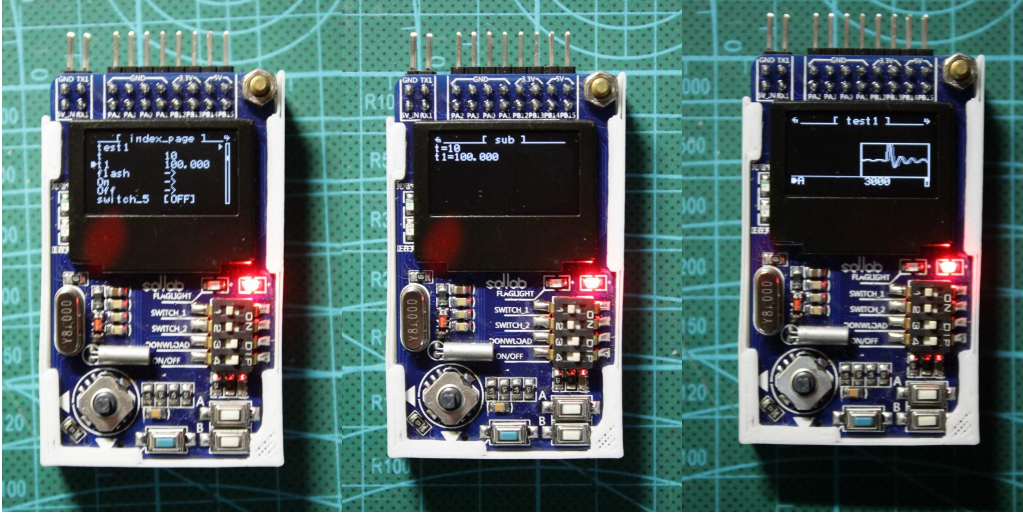
- 行(Row)：横向的格子叫“行”，第几行表示从下往上数的行位置，最底端的为第 0 行，往上依次递增。

...
第4行
第3行
第2行
第1行
第0行

在 SOLGUI_V2 中，行高为一个默认字体高，而默认字体为 F6X8（6 像素宽，8 像素高），因此行高默认为 8 个像素。评估板 pixelC2 V1.0 上搭载有一块 12864OLED，因此按照此划分，屏上 y 坐标 0~7 为第 0 行，y 坐标 8~15 为第 1 行……以此类推。因此 12864 屏幕共有

8 行，为第 0 行到第 7 行（页面中第 7 行会被标题占用）。

• 页面（Page）：页面是承载控件的容器，显示时将占用整块屏幕。一个页面下可以有十多个控件，依靠光标选择；也可以没有光标控制，只有做数据显示界面。下方是页面显示的几个实例：



页面结构体：

```
typedef struct __MENU_PAGE{
    const u8 *pageTitle;           //页面的标题
    struct __MENU_PAGE *parentPage; //父页面指针
    void (*pageFunc)();           //页面函数指针
}MENU_PAGE;
```

页面的制作实际上是在页面函数下添加光标函数和控件函数完成的。页面函数中除了放置光标函数和控件函数以外，还可以在其中编写一些只在本页面类运行的逻辑代码。页面的切换则通过相应功能的控件完成。

• 光标（Cursor）：光标用于选择并触发控件。光标可以上下选择控件，被选中的控件在按下确认键后会执行控件所设定的行为：触发函数，修改数值等。一个页面中可以没有光标。光标由光标函数产生。

• 控件（Widget）：控件则是页面中显示的具体内容，是丰富人机交互体验的重要部分。控件分为两大类：选项式控件和自由式控件。

I .选项式控件：以选项的形式出现，一个选项式控件会占据屏幕上的一行。其出现位置只能在行当中。每个选项式控件的函数都有一个 8 位无符号参数，USN，称为显示序号。USN 确定了在本页面下该选项的显示位置，显示时，选项会按照 USN 从小到大排列显示。同一页面下，选项式控件的 USN 不能相同。选项式控件会响应当前键值，因此常作为信息输入使用。**选项式控件必须要添加光标函数才能使用。**下表是 SOLGUI_V2 可提供的所有选项式控件

控件名	说明
GotoPage	页面跳转
Spin	旋钮

OptionText	选项文本
Button	按键
Switch	开关
Edit	文本编辑器

II.自由式控件：可以在页面中自由布置，位置依靠 X 和 Y 坐标确定（定位点在左下角）。自由式控件的大小可以自由设定。自由式控件不能响应键值，因此常作为数据输出使用。下表是 SOLGUI_V2 可提供的所有自由式控件

控件名	说明
Text	文本
Bar	条
Spectrum	谱
Oscillogram	波形
Picture	图像

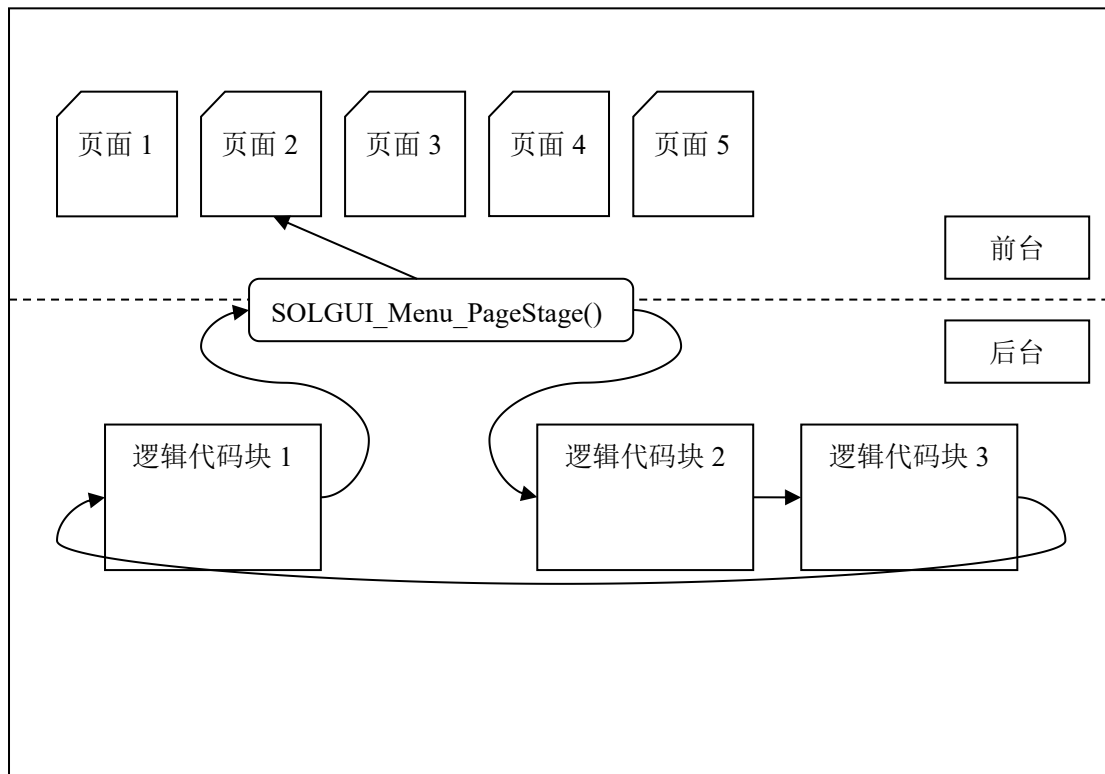
4.3 运行模型

将系统中的关于显示的代码与逻辑部分的代码分离，可以更好的将代码层次化和模块化。

- 前台：系统中有关显示的代码部分称为前台。
- 后台：系统中的逻辑运算、操作控制的代码部分称为后台。

前台和后台之间的数据交换可以通过全局变量或邮箱（多任务）来传递完成。

SOLGUI_V2 的菜单部分在系统中是作为前台来使用。一个典型的执行模型便是在单任务系统（超级循环）中使用。SOLGUI_V2 的一个关键的函数是 SOLGUI_Menu_PageStage(), 它是所有页面的切换器,同一时刻只有一个页面能够被运行。它必须被循环调用。下图是运行中的示意模型，其中后台的代码是一个超级循环，此时系统前台中运行的是页面 2 的页面函数。



单任务系统（超级循环）

所有的程序运行在一个超级循环中。通常情况下，所有软件单元被有秩序地调用。没有使用实时内核，因此软件的实时功能必须要依靠中断来完成。这类系统用于小的系统或是实时行为并不是很必须的情况下。

范例：一个典型的单任务系统（超级循环）

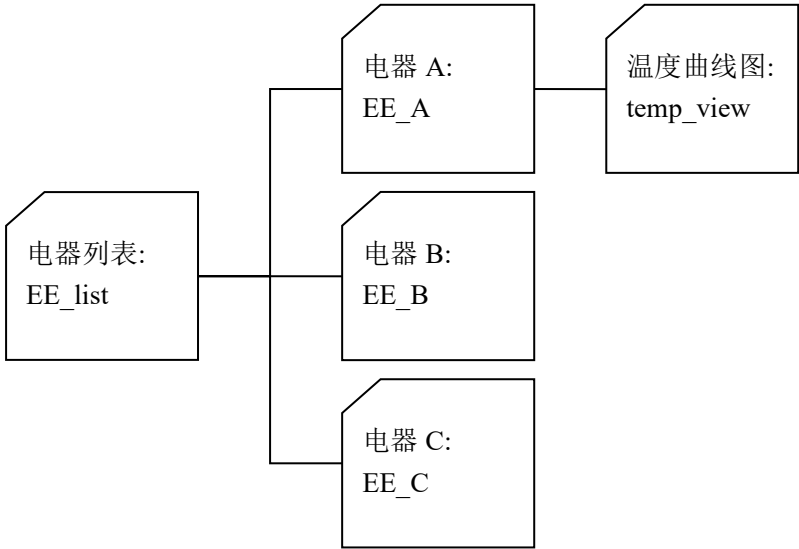
```
void main(void)
{
    HARDWARE_Init();
    XXX_Init();
    YYY_Init();
    SOLGUI_Init();           //SOLGUI 初始化
    /* 超级循环：有秩序地调用所有的软件单元*/
    while(1)
    {
        /* 运行所有的软件单元 */
        XXX_Exec();
        YYY_Exec();
        SOLGUI_Menu_PageStage();    //前台
    }
}
```

4.4 编写程序

在编写程序之前我们先进行场景的设置，再提出需求：假设的场景是需要编写一个多功

能遥控器的界面。能够遥控三种及以上的电器，每个电器又需要进行一些数值的设置及功能的开关，某些电器还需要输入字符串命令。在我们的遥控器中还需要能显示电器反馈的参数甚至波形。

需求分析：界面的编写使用 SOLGUI_V2。从需求上看需要一个列表式的根页面选择操作何种电器，每个一级页面下需要设置一些控件完成数值设置、功能开关和字符串的输入。另外某些电器下还需要单独的子页面来显示参数和曲线。可以大概的画出页面结构图：



1. 在 SOLGUI_Config.h 文件中【应用层】栏目下打开“菜单框架前台使用开关”

```
27
28 //*****【应用层】*****
29 //-----【SOLGUI菜单框架前台使用开关】-----
30 /*-----关闭后不能调用应用层中函数，即无法使用菜单框架-----*/
31 #define MENU_FRAME_EN 1 //允许SOLGUI菜单框架作为前台
32
```

2. 为了使用菜单框架，在代码中用户需要调用以下几个函数：

- ① SOLGUI_Init(): SOLGUI 初始化。与上一章不同，打开菜单框架前台使用开关后，函数调用时必须带上一个参数。这个参数是首页的页面（进入系统后第一个见到的页面）指针。
- ② SOLGUI_InputKey(): 从系统中获取当前按键键值存入键池。通过这个函数向 SOLGUI_V2 中输入键值，使用前要参考 4.1 节的内容进行键值设置。
- ③ SOLGUI_Menu_PageStage(): SOLGUI 页面前台。页面函数在该函数中选择并被运行。由于它是非阻塞的，因此必须被循环调用。
- ④ SOLGUI_Refresh(): 更新屏幕函数。将屏幕缓存中的内容更新到屏幕上。

2. 定义页面。按照需求分析的内容定义 5 个页面。

- ① 声明所有将要用到的页面。
- ② 在页面宏下编写页面函数。这里的 `__M_PAGE` 是一个宏（注意不是函数！不能再程序中调用！），可以用来快速定义页面。使用时格式为（注意不要缺少标点符号！）

```
__M_PAGE(页面变量名,页面标题,父页面指针,
{
    ///////////此处添加控件//////////
});
```

当然也可以不使用宏而用更易理解的传统方式定义一个页面，效果是一样的：

```
void 页面函数名()
{
    ///////////此处添加控件//////////
}
MENU_PAGE 页面变量名={
    页面标题,
    父页面指针,
    页面函数指针
};
```

具体的定义可以 `SOLGUI_Menu.h` 头文件中找到。


```

main.c* x pixelC_Hardware_include.h SOLGUI_Include.h SOLGUI_Config.h SOLGUI_Widget.h
04 //*****【页面函数】*****
05 MENU_PAGE EE_List,EE_A,temp_view,EE_B,EE_C; //页面声明 ①
06 //-----【EE_List页】
07 _M_PAGE(EE_List,"list",PAGE_NULL, ②
08 {
09
10 });
11
12 //-----【EE_A页】
13 _M_PAGE(EE_A,"EEA",&EE_List,
14 {
15
16 });
17
18 //-----【temp_view页】
19 _M_PAGE(temp_view,"temp_view",&EE_A,
20 {
21
22 });
23
24 //-----【EE_B页】
25 _M_PAGE(EE_B,"EEB",&EE_List,
26 {
27
28 });
29
30 //-----【EE_C页】
31 _M_PAGE(EE_C,"EEC",&EE_List,
32 {
33
34 });
35

```

3. 编写电器列表 EE_list 的页面函数。

- ① SOLGUI_Cursor(): 光标函数。用于产生一个光标来选择并触发选项式控件。
- ② SOLGUI_Widget_GotoPage(): 页面跳转控件。这是一个选项式控件，特征是他的定位参数是一个数值（即 USN）而不是一对坐标值。GotoPage 控件的功能是当被选中并按下确认键触发该控件后，当前页面会跳转至参数中的页面。
- ③ 同上
- ④ 同上

```

014
015 //-----【EE_List页】
016 _M_PAGE(EE_List,"list",PAGE_NULL,
017 {
018     SOLGUI_Cursor(6,0,3); //光标（移动范围：第0行至第6行，选项数:3）①
019     SOLGUI_Widget_GotoPage(0,&EE_A); //页面跳转 ②
020     SOLGUI_Widget_GotoPage(1,&EE_B); //页面跳转 ③
021     SOLGUI_Widget_GotoPage(2,&EE_C); //页面跳转 ④
022 });
023
024

```

4. 编写电器 A EE_A 的页面函数。

- ① 使用全局变量来传递参数。此处定义了 4 个全局变量，会被页面函数中的 Spin 旋钮控件修改数值。这是个全局变量会在后台的逻辑代码中被使用。
- ② 按键控件的触发函数。在页面中使用了 Button 按键控件，按键控件当被选中并按下确认键触发时，会立即执行参数中所指向的函数，即_SendParam()。此处的触发函

数的功能是评估板板载标志灯闪烁一次，表示将参数发送出去。

- ③ SOLGUI_Widget_Spin(): 旋钮控件。这是一个选项式控件，该控件用于在指定范围内修改变量的数值。当被光标选中并按下确认键触发时，该控件进入数值编辑状态，左右键调整步进值，上下键加减，再次确认则离开数值编辑状态。此处的四个旋钮控件可以修改上面全局变量的值。
- ④ 同上，但是区别在于参数 INTEGRAL 表示修改值是整数型的，DECIMAL 表示修改值为小数（浮点）型的。
- ⑤ SOLGUI_Widget_Button(): 按键控件。这是一个选项式控件，当被光标选中并按下确认键触发时，会执行参数中所指向的函数_SendParam()一次。
- ⑥ 页面跳转控件。跳转至温度曲线图页面 temp_view。

```
033
034 //-----【EE_A页】
035 unsigned char pa1=0; ①
036 int pa2=0;
037 int pa3=0;
038 double pa4=0;
039
040 void _SendParam() ②
041 {
042     pixelC_HW_flagLight_Twinkle(); //发送参数
043 }
044
045 _M_PAGE(EE_A,"EEA",&EE_List,
046 {
047     SOLGUI_Cursor(6,0,6); //光标（移动范围：第0行至第6行，选项数：6）
048     SOLGUI_Widget_Spin(0,"Param_A1",INTEGRAL,0,255,&pa1); //数字旋钮 ③
049     SOLGUI_Widget_Spin(1,"Param_A2",INTEGRAL,-1000,1000,&pa2); //数字旋钮
050     SOLGUI_Widget_Spin(2,"Param_A3",INTEGRAL,-1000,1000,&pa3); //数字旋钮
051     SOLGUI_Widget_Spin(3,"Param_A4",DECIMAL,-10,10,&pa4); //数字旋钮 ④
052     SOLGUI_Widget_Button(4,"SendParam",_SendParam); //按键 ⑤
053     SOLGUI_Widget_GotoPage(5,&temp_view); //页面跳转 ⑥
054 });
055
056
```

5. 编写温度曲线图 temp_view 页面函数

可以看到，temp_view 页面中没有光标函数，因为该页中没有需要控制的选项式控件。

- ① 使用宏为波形储存块申请空间。在这个页面中，需要使用 Oscillogram 波形显示控件。因此需要开辟空间来储存波的一段数值。这里的__M_WMB_MALLOC 是一个宏，用来申请一个 s32 数组作为数值储存空间。使用格式：

__M_WMB_MALLOC(波形储存块名,空间大小)

其中的空间大小是 s32 数组的元素个数。申请的空间越大，波形的显示细节越完整。当然也可以不使用宏而用更易理解的传统方式定义申请空间，效果是一样的：

```
s32 pa_wave_mem[64];
WaveMemBlk _pa_wave={64,pa_wave_mem};
WaveMemBlk *pa_wave=&_pa_wave;
```

具体的定义可以 SOLGUI_Widget.h 头文件中找到。

- ② SOLGUI_Widget_Text(): 文字显示控件。这是一个自由式控件，功能和用法与中间层的 SOLGUI_printf() 完全相同。区别在于在菜单框架（应用层）中应该使用 SOLGUI_Widget_Text() 而不是 SOLGUI_printf()。（层与层之间函数最好不要混用！）
- ③ SOLGUI_Widget_Oscillogram(): 波形显示控件。这是一个自由式控件，功能是在屏幕上生成一个指定大小的波形窗显示波形。参数中要求输入一个波形储存块的地址，即我们上方申请的 pa_wave。

- ④ SOLGUI_Oscillogram_Probe(): 波探针。它是波形显示控件的一个附件。如果把波形显示控件比作电压表, 波探针函数相当于表笔。该函数要求输入一个待测量的整型变量, 需要被循环调用。红框中的代码是模拟信号而产生的正弦波, 第二幅图中的代码是位于后台的逻辑代码段中。

```

030 //-----【temp_view页】
039 int w2=0;
041
042 __M_WMB_MALLOC(pa_wave,64); //为波形储存块申请内存空间: 波形储存块名pa_wave, 大小64*4字节 ①
043
044 __M_PAGE(temp_view,"temp_view",&EE_A,
045 {
046     SOLGUI_Widget_Text(0,8,F6X8,"PA1=%d PA2=%d ",pa1,pa2); //文字 ②
047     SOLGUI_Widget_Text(0,0,F6X8,"PA3=%d PA4=%f",pa3,pa4); //文字
048     SOLGUI_Widget_Oscillogram(0,16,128,40,255,-255,pa_wave); //波显示 ③
049 });
050
051
052
053
054
055
056
057
058
059
060
061
062
063
064
065
066
067
068
069
070
071
072
073
074
075
076
077
078
079
080
081
082
083
084
085
086
087
088
089
090
091
092
093
094
095
096
097
098
099
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

6. 编写电器 B EE_B 的页面函数。

- ① 开关变量寄存器定义。这个页面里我们将使用到 switch 开关控件。开关变量即只有 2 个值, 0 和 1, 占用 1bit 空间。这里定义的开关变量是一个 u32 型的变量, 4 字节共 32 位可以作为开关变量储存使用。使用时需位运算取得某一位的值来控制代码与否。
- ② SOLGUI_Widget_Switch(): 开关控件。这是一个选项式控件, 参数中需输入一个 u32 型的指针, 最后一个参数表示被控制的位 (例如 SOLGUI_Widget_Switch(1,"Func1_SWCH",&EEB_REG,0)即表示控制 EEB_REG[0] 位)。当被光标选中并按下确认键触发时, 会翻转该位的值 (由 1 变成 0 或由 0 变成 1)

```

050 //-----【EE_B页】
051 u32 EEB_REG=0; //开关变量寄存器 ①
052 int pbl=0;
053
054
055 void _SendParam1()
056 {
057     pixelC_HW_flagLight_Twinkle(); //发送参数
058 }
059
060 __M_PAGE(EE_B,"EEB",&EE_List,
061 {
062     SOLGUI_Cursor(6,0,6); //光标 (移动范围: 第0行至第6行, 选项数:6)
063     SOLGUI_Widget_Spin(0,"Param_B1",INTEGRAL,0,255,&pbl); //数字旋钮
064     SOLGUI_Widget_Switch(1,"Func1_SWCH",&EEB_REG,0); //变量开关 ②
065     SOLGUI_Widget_Switch(2,"Func2_SWCH",&EEB_REG,1); //变量开关
066     SOLGUI_Widget_Switch(3,"Func3_SWCH",&EEB_REG,2); //变量开关
067     SOLGUI_Widget_Switch(4,"Func4_SWCH",&EEB_REG,3); //变量开关
068     SOLGUI_Widget_Button(5,"SendParam",_SendParam1); //按键
069 });
070
071
072
073
074
075
076
077
078
079
080
081
082
083
084
085
086
087
088
089
090
091
092
093
094
095
096
097
098
099
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

7. 编写电器 C EE_C 的页面函数。

- ① 命令缓存数组。对于某些使用字符命令控制的电器, 此页中将使用文本编辑控件。

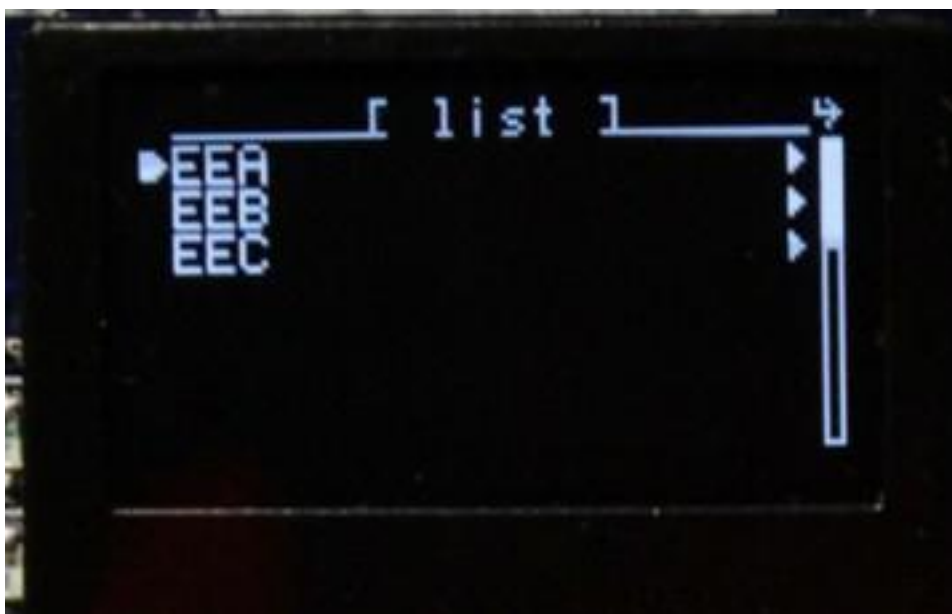
命令缓存数组用于缓存命令字符串，然后在后台的逻辑代码中通过其他方式传送出去。

- ② SOLGUI_Widget_Edit(): 文本编辑控件。这是一个选项式控件，参数中需要输入可编辑的字符长度及字符缓存的指针。当被光标选中并按下确认键触发时，会进入字符编辑窗，上下左右选择需要编辑的字符，再次确认进入单字符编辑状态，左右选择字符集（大写，小写，数字，符号，控制字符），上下选择字符集中的字符。再次确认则退出单字符编辑状态，回到返回位置确认则字符编辑窗。（具体操作详见第五章）

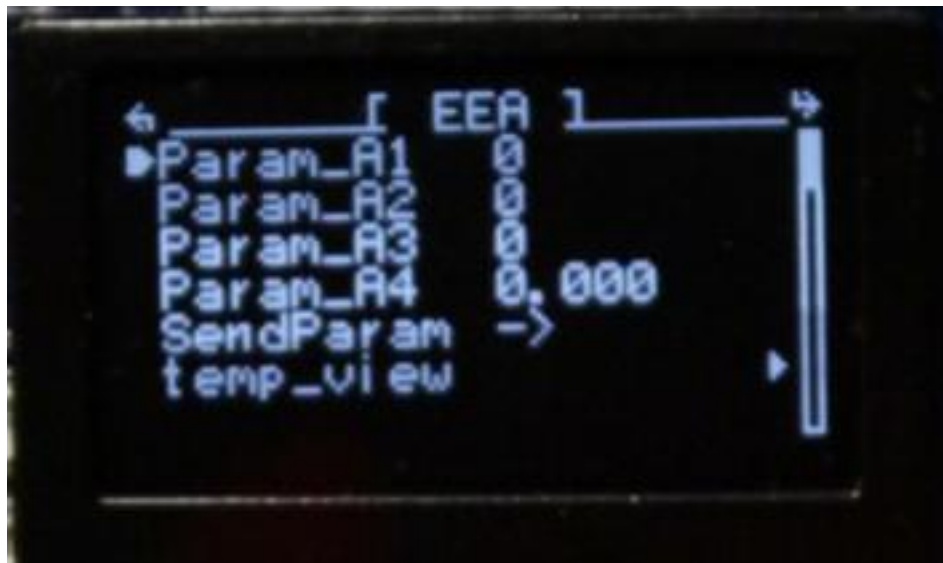
```
071
072 //-----【EE_C页】
073
074 u8 cmd_buf[33];           //命令字符串缓存 ①
075
076 void _SendCmd()
077 {
078     pixelC_HW_flagLight_Twinkle(); //发送参数
079 }
080
081 _M_PAGE(EE_C, "EEC", &EE_List,
082 {
083     SOLGUI_Cursor(6,0,3); //光标（移动范围：第0行至第6行，选项数:3）
084     SOLGUI_Widget_Spin(0,"Param_B1", INTEGRAL,0,255,&pb1); //数字旋钮
085     SOLGUI_Widget_Edit(1,"command",32,cmd_buf); //文本编辑器 ②
086     SOLGUI_Widget_Button(2,"SendCmd",_SendCmd); //按键
087 });
088
089
```

8. 编译，下载到评估板中。

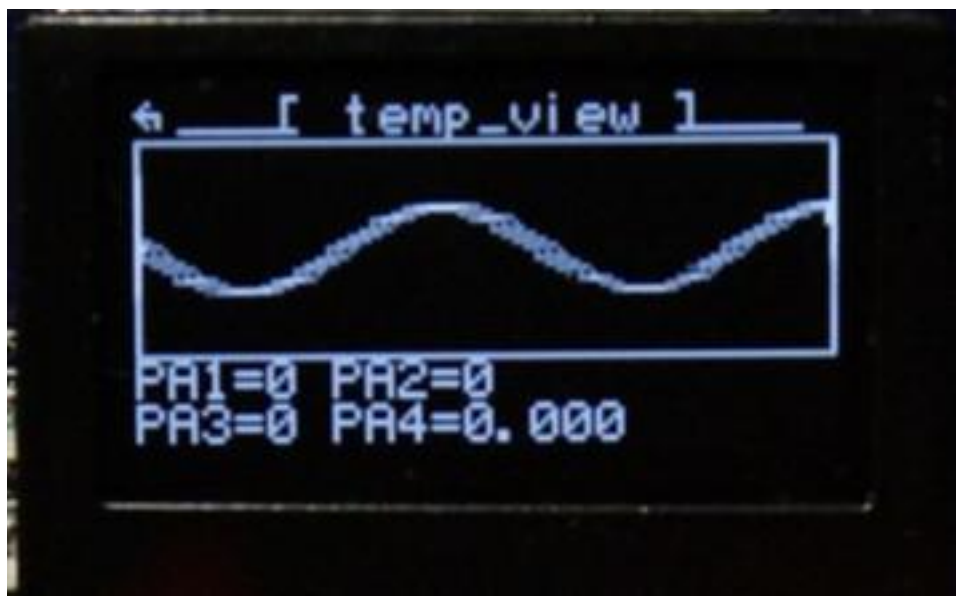
(1) EE_list 页面



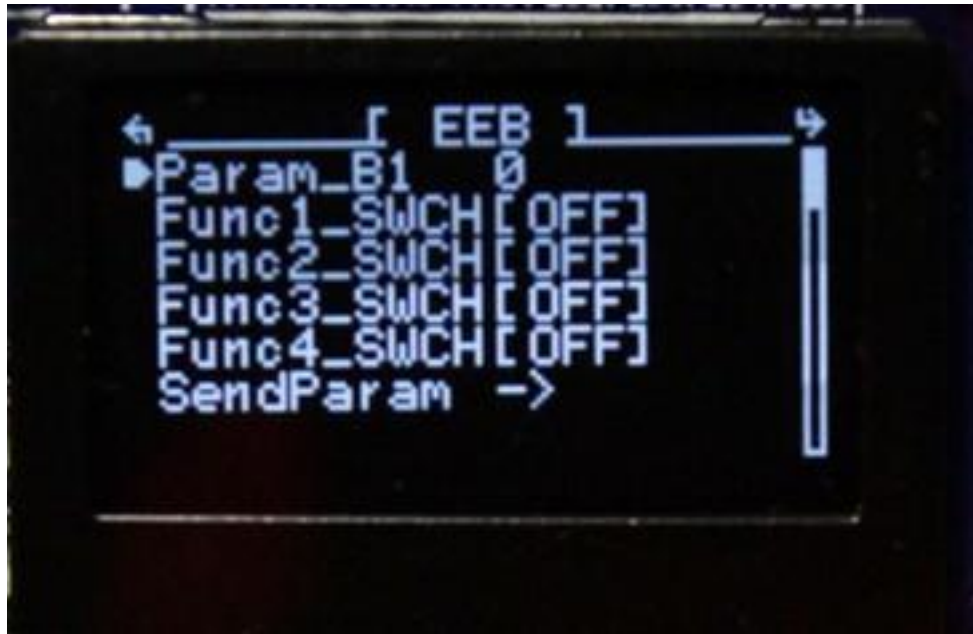
(2) EE_A 页面



(3) temp_view 页面



(4) EE_B 页面，开关控件关闭。



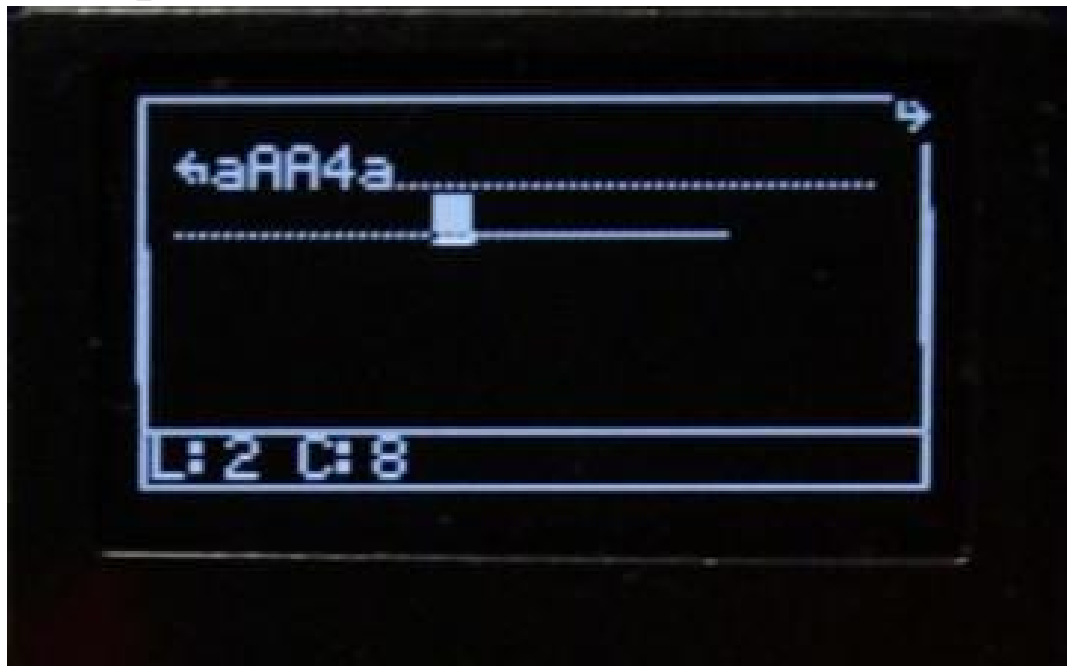
(5) EE_B 页面，开关控件部分开启。



(6) EE_C 页面



(7) EE_C 页面界面下的字符编辑窗



4.5 额外的说明

- 上面的步骤只是一个页面开发流程的参考，并未详细说明各控件的参数及使用方法，具体的控件使用及调用方法需参考第五章 API。
- 具体代码可见例程“2_多页面菜单”

第五章 API

这一章将介绍 SOLGUI_V2 中所用可以被用户调用的函数及使用方法，为用户提供参考。

5.1 通用函数

SOLGUI_Init()

描述

SOLGUI_V2 初始化。

函数原型

(1) 不使用菜单框架时：

void SOLGUI_Init(void)

• 参数

无

• 返回值

无

(2) 使用菜单框架时：

void SOLGUI_Init(MENU_PAGE *home_page)

• 参数

MENU_PAGE *home_page	首页的页面指针
----------------------	---------

• 返回值

无

源位置

SOLGUI_Common.c

附加信息

使用 SOLGUI_V2 必须调用该函数初始化。该函数分为使用菜单框架和不使用菜单框架两种情况。菜单框架的使用与否可以在 SOLGUI_Config.h 中的【应用层】栏目下“SOLGUI 菜单框架前台使用开关”中设置。

范例

(1) 不使用菜单框架时： SOLGUI_V2 初始化

SOLGUI_Init();

(2) 使用菜单框架时： SOLGUI_V2 初始化，并以 home 页面作为首页页面

SOLGUI_Init(&home);

SOLGUI_Refresh()

描述

将屏幕显示缓存的内容更新到屏幕上。

函数原型

void SOLGUI_Refresh(void)

- 参数

无

- 返回值

无

源位置

SOLGUI_Common.c

附加信息

由于输出图形都会预先写到屏幕显示缓存上，必须在完成全部绘图后调用该函数更新一次屏幕，屏幕上才会显示出字符。

范例

在完成所有绘图后更新屏幕

```
SOLGUI_Refresh();
```

SOLGUI_Clean()

描述

清空屏幕显示缓存。

函数原型

void SOLGUI_Clean(void)

- 参数

无

- 返回值

无

源位置

SOLGUI_Common.c

附加信息

绘图过程中使用此函数软清屏，会比使用硬件清屏速度快。

可以代替硬件清屏：调用此函数清空屏幕显示缓存后再更新屏幕。

范例

清空屏幕显示缓存

```
SOLGUI_Clean();
```

SOLGUI_DrawPoint()

描述

在屏幕上指定坐标绘制一个点

函数原型

void SOLGUI_DrawPoint(u32 x,u32 y,u8 t)

- 参数

u32 x	该点的 x 坐标
u32 y	该点的 y 坐标

u8 t	该点是点亮（1）还是熄灭（0）
------	-----------------

• 返回值

无

源位置

SOLGUI_Common.c

附加信息

这是显示所有图形的基本函数。用户可以自行调用并绘制自定义的图形。

范例

在坐标(64,32)处绘制一个点亮的点

SOLGUI_DrawPoint(64,32,1);

5.2 中间层函数

SOLGUI_printf()

描述

在屏幕的指定坐标位置格式化输出字符串。

函数原型

void SOLGUI_printf(u32 x,u32 y,u8 mode,const u8* str,...)

• 参数

u32 x	x 坐标
u32 y	y 坐标
u8 mode	字体参数。 自带有 4 种字体： F4X6, F6X8, F8X8, F8X10; 对应的反白显示参数： R4X6, R6X8, R8X8, R8X10
const u8* str	格式化字符串
...	变长参数。变量值

• 返回值

无

源位置

SOLGUI_Printf.c

附加信息

此处的 SOLGUI_printf 功能是 PC 上 printf 的一个子集
因此格式用法和 PC 的 printf 几乎相同：

0x01:格式符：

%s,%S:字符串

%c,%C:单个字符
%f,%F:浮点数（默认留 3 位小数）
%b,%B:整型二进制
%o,%O:整型八进制
%d,%D:整型十进制
%u,%U:整型十进制
%x,%X:整型十六进制

0x02:格式参数（只有这三类）
%07d: 补零（数字最小显示长读 7 格，不足则用 0 在前面补齐）
%-7d: 左对齐（数字最小显示长度 7 格，不足则用空格在后面补齐）
%.4f: 保留小数位数（保留 4 位小数，最多保留 7 位）

可以在 SOLGUI_Config.h 中【中间层】栏目下的开关自带的字体。

PS: F4X6 等字体参数可以可以直接取反得到相应的反白字体。如~F4X6 表示反白显示 F4X6 字体，效果和 R4X6 相同。

范例

显示字符串 helloworld: SOLGUI_printf(0,0,F6X8,"helloworld");
以 F8X8 字体显示整型变量值: SOLGUI_printf(0,0,F8X8,"Param=%d",10);
以 F4X6 字体反白显示浮点数，并保留 5 为小数：
SOLGUI_printf(0,0,~F4X6,"f=%.5f",-13.442321);

SOLGUI_GBasic_Line()

描述

在屏幕上两个指定坐标间绘制一条线。

函数原型

void SOLGUI_GBasic_Line(u32 x0,u32 y0,u32 xEnd,u32 yEnd,u8 mode)

• 参数

u32 x0	起点的 x 坐标
u32 y0	起点的 y 坐标
u32 xEnd	终点的 x 坐标
u32 yEnd	终点的 y 坐标
u8 mode	线型参数。 可选择的线型有： DELETE（消除），ACTUAL（实线）， DOTTED（点线），DASHED（虚线）

• 返回值

无

源位置

SOLGUI_GBasic.c

附加信息

范例

在点(0,0)与点(127,32)之间连接一条实线
SOLGUI_GBasic_Line(0, 0,127,32, ACTUAL);

GUI_GBasic_Rectangle ()

描述

在屏幕上指定两个坐标为顶点，绘制一个矩形。

函数原型

void SOLGUI_GBasic_Rectangle(u32 x0,u32 y0,u32 x1,u32 y1,u8 mode)

• 参数

u32 x0	顶点 1 的 x 坐标
u32 y0	顶点 1 的 y 坐标
u32 x1	顶点 2 的 x 坐标
u32 y1	顶点 2 的 y 坐标
u8 mode	参数。 可选择的边线型有： ACTUAL（实线），DOTTED（点线）， DASHED（虚线） 可选择的填充参数有： DELETE（消除），FILL（填充）

• 返回值

无

源位置

SOLGUI_GBasic.c

附加信息

线型和填充不可以叠加使用。

范例

以点(8,16)和点(63,63)为顶点绘制一个虚线边的矩形
SOLGUI_GBasic_Line(8,16, 63,63, DASHED);
以点(64,0)和点(127,63)为顶点绘制一个填充的矩形
SOLGUI_GBasic_Line(64,0,127,63, FILL);

SOLGUI_GBasic_MultiLine ()

描述

多个点之间的连续连线。从第一点连到第二点，再连到第三点...

函数原型

void SOLGUI_GBasic_MultiLine(const u32 *points,u8 num,u8 mode)

• 参数

const u32 *points	多个点坐标数据的指针，数据排列为(x0,y0)、(x1,y1)、(x2,y2)...
u8 num	点数目，至少要大于 1
u8 mode	线型参数。 可选择的线型有： DELETE（消除），ACTUAL（实线）， DOTTED（点线），DASHED（虚线）

• 返回值

无

源位置

SOLGUI_GBasic.c

附加信息

point 指针可以使用一维或二维数组。

范例

以点数组 point1 为顶点绘制一段折线（实线）

```
u32 point1[4]={0,0,127,44,33,23,16,8};
```

```
SOLGUI_GBasic_Line(point1, 4, ACTUAL);
```

以点数组 point2 为顶点绘制一段折线（点线）

```
u32 point2[4][2]={ {0,0}, {127,44}, {33,23}, {16,8} };
```

```
SOLGUI_GBasic_Line(point2,4, DOTTED);
```

SOLGUI_GBasic_Circle ()

描述

指定圆心位置及半径，画圆。

函数原型

```
void SOLGUI_GBasic_Circle(u32 x0,u32 y0,u32 r,u8 mode)
```

• 参数

u32 x0	圆心的 x 坐标
u32 y0	圆心的 y 坐标
u32 r	圆半径
u8 mode	参数。 可选择的边线型有： ACTUAL（实线） 可选择的填充参数有： FILL（填充）

• 返回值

无

源位置

SOLGUI_GBasic.c

附加信息

线型和填充只有两个参数可选。

范例

```
以点(64,32)为圆心，10 为半径绘制一个实线边的圆
    SOLGUI_GBasic_Circle(64,32,10,ACTUAL);
以点(64,32)为圆心，10 为半径绘制一个填充的圆
    SOLGUI_GBasic_Line(64,32,10,FILL);
```

SOLGUI_Pictrue ()

描述

在屏幕上指定位置显示单色位图（不包含图像头数据）。

函数原型

```
void SOLGUI_Pictrue(u32 x0,u32 y0,const u8 *pic,u32 x_len,u32 y_len,u8 mode)
```

• 参数

u32 x0	参考点的 x 坐标
u32 y0	参考点的 y 坐标
const u8 *pic	图片指针
u32 x_len	图片 x 方向长
u32 y_len	图片 y 方向长
u8 mode	显示参数。 可选择的显示方式有： NORMAL（默认），REVERSE（反白）

• 返回值

无

源位置

```
SOLGUI_Picture.c
```

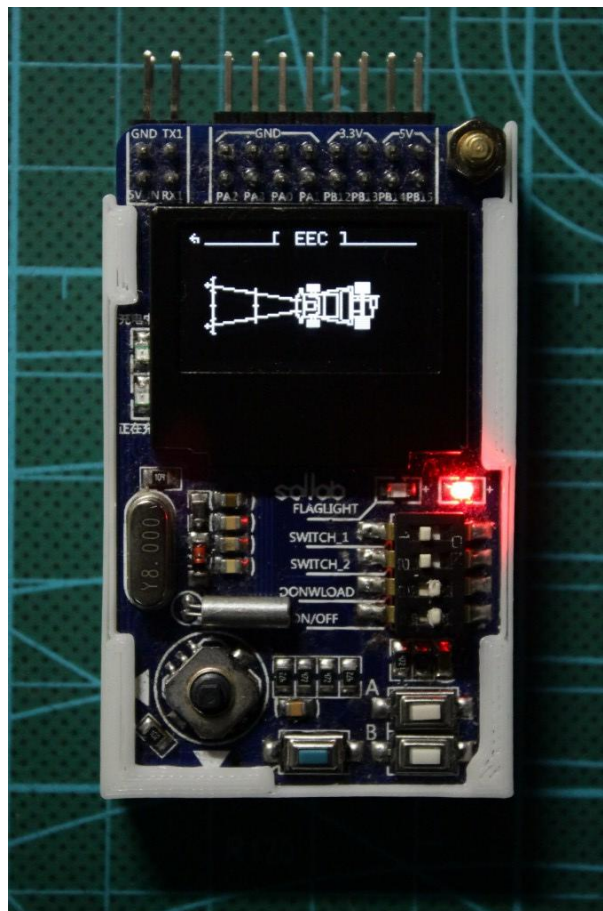
附加信息

图片应储存为**一维数组**，取模方式列行式：**数据水平，字节垂直，从左到右，从上到下扫描**。使用附带的 Image2Lcd 取模软件取模时应按下图设置：（最大宽度和高度需要自行设定）



范例

正常显示 94x31 的位图图片



5.3 应用层函数

SOLGUI_InputKey()

描述

从系统中获取当前按键键值存入 GUI 键值 FIFO。

函数原型

void SOLGUI_InputKey(u8 key_value)

• 参数

u8 key_value	当前系统的键值
--------------	---------

• 返回值

无

源位置

SOLGUI_Menu.c

附加信息

使用菜单框架必须调用该函数从系统中取得键值，光标才能被操作。该函数除了可以放在 while(1) 超级循环中，还可以放在定时器中断中，在扫描得到键值后输入 SOLGUI。

可以在 SOLGUI_Config.h 中【应用层】栏目下的设置 FIFO 键值缓存大小。

范例

超级循环中获取系统键值：

```
while(1)
{
    //////////////////////////////////其他逻辑代码////////////////////////////////////
    kv=pixelC_HW_Key_GetValue();    //按键键值输出
    SOLGUI_InputKey(kv);            //从系统中获取当前按键键值存入键池
    //////////////////////////////////其他逻辑代码////////////////////////////////////
}
```

SOLGUI_Menu_PageStage ()

描述

SOLGUI_V2 页面切换器。

函数原型

void SOLGUI_Menu_PageStage(void);

• 参数

无

• 返回值

无

源位置

SOLGUI_Menu.c

附加信息

使用菜单框架时必须被调用。页面函数在该函数中选择并被运行。由于它是非阻塞的，因此必须被循环调用。一个典型的使用方式是放在 while(1)超级循环中。

范例

在 while(1)超级循环中调用页面切换器

```
140
141 while(1)
142 {
143     kv=pixelC_HW_Key_GetValue(); //按键键值输出
144     SOLGUI_InputKey(kv); //从系统中获取当前按键键值存入键池
145     SOLGUI_Menu_PageStage(); //SOLGUI前台页面切换器
146     SOLGUI_Refresh(); //更新屏幕
147 }
```

SOLGUI_GetCurrentKey ()

描述

从 SOLGUI 的键值 FIFO 中获得当前系统的键值。

函数原型

u8 SOLGUI_GetCurrentKey(void)

• 参数

无

• 返回值

u8	当前系统的键值
----	---------

源位置

SOLGUI_Menu.c

附加信息

该函数常用于在某一页面中取得当前键值，用户可以自行解析按键的行为。

注意：不可使用返回键键值。

范例

在页面中上下左右移动图片。

```
107
108 _M_PAGE(EE_C, "EEC", &EE_List,
109 {
110     static int x=24;
111     static int y=10;
112     u8 kv=0;
113     kv=SOLGUI_GetCurrentKey();
114     switch(kv)
115     {
116         case SOLGUI_KEY_UP: if(y<63) y++; break;
117         case SOLGUI_KEY_DOWN: if(y>0) y--; break;
118         case SOLGUI_KEY_LEFT: if(x>0) x--; break;
119         case SOLGUI_KEY_RIGHT: if(x<127) x++; break;
120         default;;
121     }
122     SOLGUI_Pictrue(x,y,gImage_pic94x31,94,31,NORMAL);
123 });
124
125
```

SOLGUI_Cursor ()

描述

在页面指定行范围内生成光标。

函数原型

void SOLGUI_Cursor(u8 rowBorder_Top,u8 rowBorder_Bottom,u8 option_num)

• 参数

u8 rowBorder_Top	光标运动上边界行
u8 rowBorder_Bottom	光标运动下边界行
u8 option_num	该光标统治下的选项个数

• 返回值

无

源位置

SOLGUI_Widget.c

附加信息

每页只允许有一个光标。

在选项的显示也会被限制在行范围内，选项显示时会卷轴式滚屏。

可以在 SOLGUI_Config.h 中【应用层】栏目下的设置选项最大数目。

范例

在第 0 行到第 3 行间生成光标，光标统治 10 个选项。

SOLGUI_Cursor(0,3,10);

在第 0 行生成光标，光标统治 12 个选项。

SOLGUI_Cursor(0,0,10);

SOLGUI_Widget_GotoPage ()

描述

页面跳转控件

函数原型

void SOLGUI_Widget_GotoPage(u8 USN,MENU_PAGE *page)

• 参数

u8 USN	显示序号
MENU_PAGE *page	目标页面指针

• 返回值

无

源位置

SOLGUI_Widget.c

附加信息

选项名为目标页面标题。当被光标选中并按下确认键触发该控件后，当前页面会跳

转至目标页面。

范例

```
添加一个可以跳转至 sub1 页面的选项
    SOLGUI_Widget_GotoPage(0,&sub1);
```

SOLGUI_Widget_Spin ()

描述

旋钮控件

函数原型

```
void SOLGUI_Widget_Spin( u8 USN,
                        const u8 *name,
                        u8 type,
                        double max,
                        double min,
                        void* value);
```

• 参数

u8 USN	显示序号
const u8 *name	选项名（不超过 10 个字符）
u8 type	被调整数类型。 INT8（s8 型）， UINT8（u8 型）， INT16（u16,s16 型）， INT32（u32,s32 型）， FLT16（float 型）， FLT32（double 型）
double max	可调整上限
double min	可调整下限
void* value	被调整数指针

• 返回值

无

源位置

```
SOLGUI_Widget.c
```

附加信息

该控件用于在指定范围内修改变量的数值。被调整数可以是 double 类型，float 类型或整形。被调整数类型选择：s8 型选择 INT8，u8 型选择 UINT8，u16 或 s16 型选择 INT16，u32 或 s32 型选择 INT32，float 型选择 FLT16，double 型选择 FLT32

当被光标选中并按下确认键触发时，该控件进入数值编辑状态，选项名变成反白显示的步进值。左右键调整步进值，上下键使目标数加减步进值，再次按下确认键则离开数值编辑状态。

全局变量中定义后最好初始化(置 0 或其他)，防止出现意想不到的错误。

可以在 SOLGUI_Config.h 中【应用层】栏目下的设置最大步进值和最小步进值。

范例

添加一组用于设置 PID 三参数的选项。

```
SOLGUI_Widget_Spin( 0,"Speed_kp", FLT16,100,-100, &S_kp);
SOLGUI_Widget_Spin( 1,"Speed_ki", FLT16,10000,-10, &S_ki);
SOLGUI_Widget_Spin( 2,"Speed_kd", FLT16,10000,-10000, &S_kd);
```

SOLGUI_Widget_OptionText ()

描述

选项文本控件

函数原型

```
void SOLGUI_Widget_OptionText(u8 USN,const u8* str,...)
```

• **参数**

u8 USN	显示序号
const u8* str	格式化字符串（不可超过 20 个字符）
...	变长参数。变量值

• **返回值**

无

源位置

SOLGUI_Widget.c

附加信息

以选项的形式显示字符串或者变量值，可以在一页中显示更多的信息。用法与 SOLGUI_printf 完全相同。

范例

以选项的形式显示一系列变量值。

```
SOLGUI_Widget_OptionText(0,"+----[ADC Value]----+");
SOLGUI_Widget_OptionText(1,"ADC_F0 :  %d", ADC_F0);
SOLGUI_Widget_OptionText(2,"ADC_F1 :  %d", ADC_F1);
SOLGUI_Widget_OptionText(3,"ADC_F2 :  %d", ADC_F2);
SOLGUI_Widget_OptionText(4,"ADC_F3 :  %d", ADC_F0);
SOLGUI_Widget_OptionText(5,"ADC_F4 :  %d", ADC_F4);
SOLGUI_Widget_OptionText(6,"ADC_F5 :  %d", ADC_F5);
SOLGUI_Widget_OptionText(7,"ADC_F6 :  %d", ADC_F6);
SOLGUI_Widget_OptionText(8,"ADC_F7 :  %d", ADC_F7);
SOLGUI_Widget_OptionText(9,"ADC_B0 :  %d", ADC_B0);
SOLGUI_Widget_OptionText(10,"ADC_B1 :  %d", ADC_B1);
SOLGUI_Widget_OptionText(11,"+-----+");
```

SOLGUI_Widget_Button ()

描述

按键控件

函数原型

```
void SOLGUI_Widget_Button(u8 USN,const u8 *name,void (*func)(void))
```

• 参数

u8 USN	显示序号
const u8 *name	选项名（不超过 10 个字符）
void (*func)(void)	无返回值无参数函数的指针

• 返回值

无

源位置

SOLGUI_Widget.c

附加信息

当被光标选中并按下确认键触发该控件后，会立刻执行参数中所指向的函数一次。

范例

添加一个”开启电机”选项按键

```
SOLGUI_Widget_Button(0,"M_[ON]",M_On);
```

其对应触发的函数为

```
void M_On ()
{
    _M_set(&_PID_cur);
    _M_TurnOn();
}
```

SOLGUI_Widget_Switch ()

描述

开关控件

函数原型

```
void SOLGUI_Widget_Switch(u8 USN,const u8 *name,u32 *mem_value,u8 L_shift)
```

• 参数

u8 USN	显示序号
const u8 *name	选项名（不超过 10 个字符）
u32 *mem_value	用于存放开关值的 u32 变量指针
u8 L_shift	控制的位（0~31）

• 返回值

无

源位置

SOLGUI_Widget.c

附加信息

当被光标选中并按下确认键触发时，会翻转选项指定位的值（由 1 变成 0 或由 0

变成 1)。

范例

添加一个控制”ADC”选项开关。
(假设在 ADC_CRR 寄存器中,ADC_CRR[15]用于控制 ADC 开启或关闭)
SOLGUI_Widget_Switch(0,”ADC_Swch”,ADC_CRR,15);

SOLGUI_Widget_Edit ()

描述

文本编辑器控件

函数原型

void SOLGUI_Widget_Edit(u8 USN,const u8 *name,u16 char_num, u8 *buf)

• **参数**

u8 USN	显示序号
const u8 *name	选项名（不超过 10 个字符）
u16 char_num	可修改的字符数
u8 *buf	字符缓存指针

• **返回值**

无

源位置

SOLGUI_Widget.c

附加信息

当被光标选中并按下确认键触发时，会进入字符编辑窗。上下左右选择需要编辑的字符，再次确认进入单字符编辑状态，左右选择字符集（有大写，小写，数字，符号，控制字符），上下选择该字符集中的字符。再次确认则退出单字符编辑状态，回到返回位置（”\n”返回符号）按确认则字符编辑窗回到正常页面中。
可以在 SOLGUI_Config.h 中【应用层】栏目下的设置最大可编辑的长度和选项省略信息的长度。

范例

添加一个编辑串口发送信息的编辑器控件,允许编辑 40 个字符。
SOLGUI_Widget_Edit(0,”Send_edit”, 40, usartSend_Buf);

SOLGUI_Widget_Text ()

描述

在页面的指定坐标位置格式化输出字符串。

函数原型

void SOLGUI_Widget_Text(u32 x0,u32 y0,u8 mode,const u8* str,...)

• **参数**

u32 x0	x 坐标
u32 y0	y 坐标

u8 mode	字体参数。 自带有 4 种字体： F4X6, F6X8, F8X8, F8X10; 对应的反白显示参数： R4X6, R6X8, R8X8, R8X10
const u8* str	格式化字符串
...	变长参数。变量值

• 返回值

无

源位置

SOLGUI_Widget.c

附加信息

作用和用法与 SOLGUI_printf 完全相同。用于在应用层使用。

在应用层应避免中间层的函数（在页面中显示字符应使用 SOLGUI_Widget_Text() 函数而非 SOLGUI_printf()函数）。

范例

显示字符串 helloworld: SOLGUI_Widget_Text (0,0,F6X8,"helloworld");
以 F8X8 字体显示整型变量值: SOLGUI_Widget_Text (0,0,F8X8,"Param=%d",10);
以 F4X6 字体反白显示浮点数, 并保留 5 为小数:
SOLGUI_Widget_Text (0,0,~F4X6,"f=%.5f",-13.442321);

SOLGUI_Widget_Bar ()

描述

在页面上指定位置显示条（进度条，刻度条）

函数原型

```
void SOLGUI_Widget_Bar( u32 x0,
                        u32 y0,
                        u32 xsize,
                        u32 ysize,
                        s32 max,
                        s32 min,
                        s32 value,
                        u8 mode)
```

• 参数

u32 x0	x 坐标
u32 y0	y 坐标
u32 xsize	控件条 x 方向长
u32 ysize	控件条 y 方向长
s32 max	变量的最大值
s32 min	变量的最小值
s32 value	变量

u8 mode	外观参数： PROGBAR（进度条），SCALEBAR（刻度条） 生长方向参数： DIREC_X（x 方向生长），DIREC_Y（y 方向生长）
---------	---

- 返回值
无

源位置
SOLGUI_Widget.c

附加信息
在指定位置生成一个条，用于直观地显示数值的相对大小。外观分为进度条和刻度条两种（可见附图）。生长方向分为 x 方向生长（数值增长向 X 轴正方向）和 y 方向生长（数值增长向 Y 轴正方向）。这两个参数可叠加使用。

范例
在(8,0)点生成一个 y 方向生长的进度条。
SOLGUI_Widget_Bar(8,0,5,40,100,0,23 ,DIREC_Y|PROGBAR);
在(16,0)点生成一个 x 方向生长的刻度条。
SOLGUI_Widget_Bar(16,0,40,4,100,0,65,DIREC_X| SCALEBAR);



SOLGUI_Widget_Spectrum ()

描述
在页面上指定位置显示谱图

函数原型
void SOLGUI_Widget_Spectrum(u32 x0,
u32 y0,
u32 xsize,
u32 ysize,
s32 max,
s32 min,

u16 val_num,
s32 value[])

• 参数

u32 x0	x 坐标
u32 y0	y 坐标
u32 xsize	控件 x 方向长
u32 ysize	控件 y 方向长
s32 max	变量的最大值
s32 min	变量的最小值
u16 val_num	变量个数
s32 value[]	变量数组

• 返回值

无

源位置

SOLGUI_Widget.c

附加信息

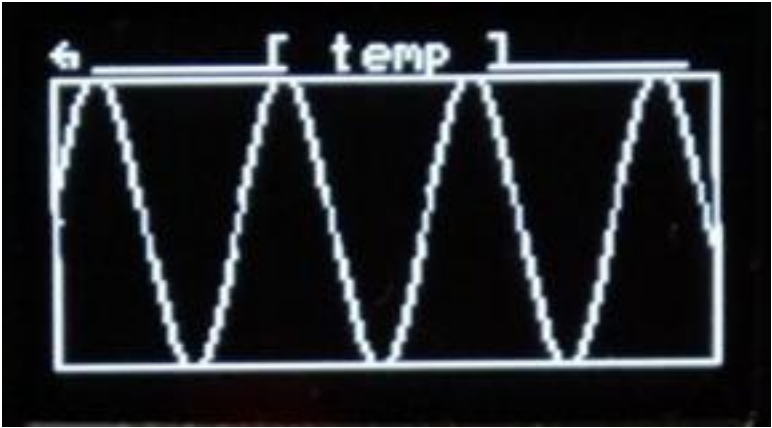
谱用于将数组中的值曲线化。例如显示线型 CCD 采集的亮度数组，FFT（快速傅里叶变化）频域数组等。

若要显示浮点型的数组，需要将其转为 s32 型数据后输入参数。

范例

在(0,0)位置，长 128 高 56 的谱图，显示 val_s 数组。

```
数组定义:   int val_s[128];
数组产生:   for(i=0;i<128;i++){
              val_s[i]=100*sin(10*i*3.1415926/180+0);
            }
谱显示:     SOLGUI_Widget_Spectrum(0,0,128,56,100,-100,128,(s32*)val_s);
```



SOLGUI_Widget_Oscillogram ()

描述

在页面上指定位置显示波形图

函数原型

```
void SOLGUI_Widget_Oscillogram (  u32 x0,
                                   u32 y0,
                                   u32 xsize,
                                   u32 ysize,
                                   s32 max,
                                   s32 min,
                                   WaveMemBlk *wmb)
```

• 参数

u32 x0	x 坐标
u32 y0	y 坐标
u32 xsize	控件 x 方向长
u32 ysize	控件 y 方向长
s32 max	变量的最大值
s32 min	变量的最小值
WaveMemBlk *wmb	波形数据储存块指针

• 返回值

无

源位置

```
SOLGUI_Widget.c
```

附加信息

用于显示变量值的波形。需配合附件 SOLGUI_Oscillogram_Probe()波探针使用。变量的值用波探针输入。

范例

```
在(0,16)位置，长 128 高 40 的波形图，显示 pa_wave 的波形数据。  
(变量由 SOLGUI_Oscillogram_Probe()波探针输入 pa_wave 波形数据储存块)  
SOLGUI_Widget_Oscillogram(0,16,128,40,255,-255,pa_wave); //波显示
```



SOLGUI_Oscillogram_Probe ()

描述

波探针

函数原型

void SOLGUI_Oscillogram_Probe(WaveMemBlk *wmb,s32 value)

• 参数

WaveMemBlk *wmb	波形数据储存块指针
s32 value	变量

• 返回值

无

源位置

SOLGUI_Widget.c

附加信息

SOLGUI_Widget_Oscillogram()波形图控件附件。用于变量的输入。

范例

在逻辑代码中加入波探针。

```
088
089
090     w2=100*sin(10*i*3.1415926/180+0);
091     i++;
092     if(i>=360)i=0;
093     SOLGUI_Oscillogram_Probe(pa_wave,w2); //波探针 ④
094
095
096
```

SOLGUI_Widget_Picture ()

描述

在页面上指定位置显示单色位图（不包含图像头数据）。

函数原型

void SOLGUI_Widget_Picture(u32 x0,
 u32 y0,
 u32 xsize,
 u32 ysize,
 const u8 *pic,
 u32 x_len,
 u32 y_len,
 u8 mode)

• 参数

u32 x0	x 坐标
u32 y0	y 坐标
u32 xsize	控件 x 方向长
u32 ysize	控件 y 方向长
const u8 *pic	图片指针
u32 x_len	图片 x 方向长

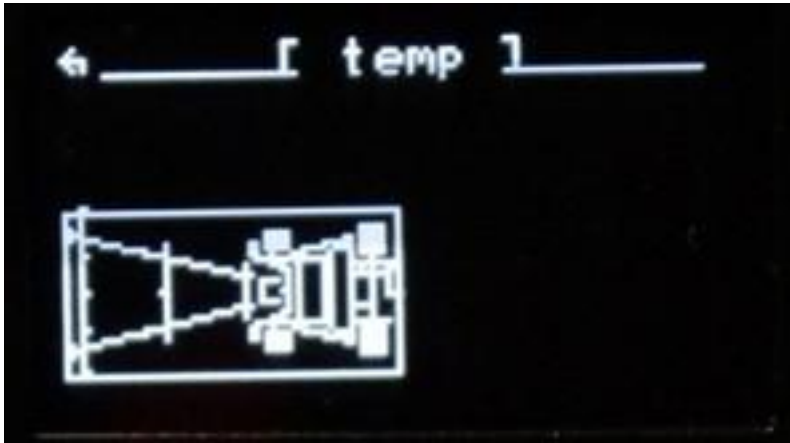
u32 y_len	图片 y 方向长
u8 mode	显示参数。 可选择的显示方式有： NML（默认），REV（反白） 附加参数： FRM（加边框）

• 返回值
无

源位置
SOLGUI_Widget.c

附加信息
与 SOLGUI_Pictrue()使用完全相同，因此取模方式也一样，详见 SOLGUI_Pictrue()。此处的图像控件多了个加边框的附加参数，可以与显示方式参数叠加使用。在图像尺寸大，控件尺寸小时，该控件还会将图像缩小以适应控件大小。

范例
在(0,0)点，长 64 高 32 的图片显示控件显示 94x31 的单色位图图片，正常显示加边框。
SOLGUI_Widget_Picture(0,0,64,32,gImage_pic94x31,94,31,NML|FRM);



5.4 宏

SOLGUI_V2 在定义页面和为波形数据储存块申请空间时都可以使用宏来快速完成。这些宏不是函数，不可以被调用，而是用于定义页面和波形数据储存块(都是全局变量）时使用。

__M_PAGE ()

描述
页面定义及页面函数。
原型

__M_PAGE(name,pageTitle,parentPage,Program)

• 参数

name	页面变量名
pageTitle	页面标题字符串
parentPage	该页父页面指针
Program	页面函数块

源位置

SOLGUI_Menu.h

附加信息

使用该宏可以快速定义页面并编写页面函数，效果与传统方式定义相同。

传统方式

```
188
189 void _sub()
190 {
191     SOLGUI_Cursor(0,1,8);
192     SOLGUI_Widget_GotoPage(0,&test1);
193     SOLGUI_Widget_Edit(2,"edit",1,test_str);
194     SOLGUI_Widget_Spin(1,"t",INTEGRAL,-1000,1000,&t);
195     SOLGUI_Widget_Spin(3,"t1",DECIMAL,1000,-1000,&t1);
196 }
197 MENU_PAGE sub={"sub_page",&index,_sub};
198
199
```

宏方式

```
174
175
176 __M_PAGE(sub, "sub_page",&index,{
177     SOLGUI_Cursor(0,1,8);
178     SOLGUI_Widget_GotoPage(0,&test1);
179     SOLGUI_Widget_Edit(2,"edit",1,test_str);
180     SOLGUI_Widget_Spin(1,"t",INTEGRAL,-1000,1000,&t);
181     SOLGUI_Widget_Spin(3,"t1",DECIMAL,1000,-1000,&t1);
182 });
183
```

注意：定义页面后要将页面变量添加到页面声明中。

范例

创建 list 页面并填写页面函数。

```
032
033 MENU_PAGE EE_List,EE_A,temp_view,EE_B,EE_C;    //页面声明
034
035 //-----【EE_List页】
036 __M_PAGE(EE_List,"list",PAGE_NULL,
037 {
038     SOLGUI_Cursor(6,0,3);    //光标（移动范围：第0行至第6行，选项数:3）
039     SOLGUI_Widget_GotoPage(0,&EE_A);    //页面跳转
040     SOLGUI_Widget_GotoPage(1,&EE_B);    //页面跳转
041     SOLGUI_Widget_GotoPage(2,&EE_C);    //页面跳转
042 });
043
```

__M_WMB_MALLOC ()

描述

为波形数据储存块定义空间

原型

__M_WMB_MALLOC(name,size)

• 参数

name	波形数据储存块名
size	申请空间大小（size*4 字节）

源位置

SOLGUI_Widget.h

附加信息

该宏用于开辟空间来储存波的数值。其实便是全局定义了一个 s32 类型的数组，数组长度为 size，然后取指针。申请的空间越大，波形的显示细节越多。

传统方式

```
173  
174 __M_WMB_MALLOC(pa_wave,64)  
175
```

宏方式

```
165  
166 s32 pa_wave_mem [64];  
167 WaveMemBlk _ pa_wave ={64, pa_wave_mem};  
168 WaveMemBlk * pa_wave =&_ pa_wave;  
169
```

在使用 SOLGUI_Widget_Oscillogram()波形控件和 SOLGUI_Oscillogram_Probe()波探针时。参数中需要使用定义好的波形数据储存块。

范例

需要储存 360 个点，定义波形数据储存块 pa_wave。

```
164  
165 __M_WMB_MALLOC(pa_wave,360)  
166
```

第六章 附录

6.1 SOLGUI_Config.h 文件设置

1.对硬件层的设置。需要在移植屏幕驱动时进行设置。

像素

SCREEN_X_WIDTH: 屏幕的 X 轴像素数。用户根据实际情况自行设置。

SCREEN_Y_WIDTH: 屏幕的 Y 轴像素数。用户根据实际情况自行设置。

字数

SCREEN_X_PAGE: 支持多少个 6x8 字宽。即屏幕一行可容纳字符数。此处不需要用户设置。

SCREEN_Y_PAGE: 支持多少个 6x8 字高。即计算屏幕行数。此处也不需要用户设置。

```
03 //*****【硬件层】*****
04 //-----【屏幕长宽像素数】
05 #define SCREEN_X_WIDTH      128      //屏幕的x轴像素数
06 #define SCREEN_Y_WIDTH      64       //屏幕的y轴像素数
07
08 //-----【屏幕长宽字数】
09 #define SCREEN_X_PAGE      (SCREEN_X_WIDTH/6)    //支持多少个6x8字宽（一般不需要修改，默认即可）
10 #define SCREEN_Y_PAGE      (SCREEN_Y_WIDTH/8)    //支持多少个6x8字高（一般不需要修改，默认即可）
11
12
```

2.对中间层的设置。该层有控制程序编译的功能开关，用户可根据需要自行开关裁剪，节约空间。

字体

FONT4X6_EN: 允许使用 4x6 字体。

FONT8X8_E: 允许使用 8x8 字体。

FONT8X10_EN: 允许使用 8x10 字体。

这三个字体不是必须的，因此在空间紧张的情况下可以关闭。

基本图形

GBASIC_LINE_EN: 允许使用直线绘制。

GBASIC_RECTANGLE_EN: 允许使用矩形绘制。

GBASIC_CIRCLE_EN : 允许使用圆形绘制。

```

12
13 //*****【中间层】*****
14 //-----【字库开关】-----
15 /*-----默认使用6x8字体-----*/
16 /*-----可根据实际使用情况开关，节约空间-----*/
17 #define FONT4X6_EN 0 //允许使用4x6字体
18 #define FONT8X8_EN 0 //允许使用8x8字体
19 #define FONT8X10_EN 0 //允许使用8x10字体
20
21 //-----【基础图形库开关】-----
22 #define GBASIC_LINE_EN 1 //允许使用直线绘制
23 #define GBASIC_RECTANGLE_EN 1 //允许使用矩形绘制
24 #define GBASIC_CIRCLE_EN 1 //允许使用圆形绘制
25
26

```

2.对应用层的设置。

菜单框架使用

MENU_FRAME_EN: 使用菜单框架开关。关闭后不能调用应用层中的函数，无法使用页面。开启后会无视用户在中间层基础图形处设置的功能开关，即中间层基础图形处的设置不影响菜单框架开启。

```

28 //*****【应用层】*****
29 //-----【SOLGUI菜单框架前台使用开关】-----
30 /*-----关闭后不能调用应用层中函数，即无法使用菜单框架-----*/
31 #define MENU_FRAME_EN 1 //允许SOLGUI菜单框架作为前台
32
33

```

控件开关

WIDGET_GOTOPAGE_EN: 允许使用控件: GotoPage 页面跳转

WIDGET_SPIN_EN: 允许使用控件: Spin 数字旋钮

WIDGET_OPTIONTEXT_EN: 允许使用控件: OptionText 选项文本

WIDGET_BUTTON_EN: 允许使用控件: Button 按钮

WIDGET_SWITCH_EN: 允许使用控件: Switch 复选开关

WIDGET_EDIT_EN: 允许使用控件: Edit 文本编辑

WIDGET_TEXT_EN: 允许使用控件: Text 文字

WIDGET_BAR_EN: 允许使用控件: Bar 条

WIDGET_SPECTRUM_EN: 允许使用控件: Spectrum 谱

WIDGET_OSCILLOGRAM_EN: 允许使用控件: Oscillogram 波形

WIDGET_PICTURE_EN: 允许使用控件: Picture 图

关闭不使用的控件，尤其是 edit 控件，空间紧张的情况下可以大大节省空间。

```

32 //-----【Widget开关】-----
33 /*-----可根据实际使用情况开关，节约空间-----*/
34
35 #define WIDGET_GOTOPAGE_EN 1 //允许使用控件: GotoPage页面跳转
36 #define WIDGET_SPIN_EN 1 //允许使用控件: Spin数字旋钮
37 #define WIDGET_OPTIONTEXT_EN 1 //允许使用控件: OptionText选项文本
38 #define WIDGET_BUTTON_EN 1 //允许使用控件: Button按钮
39 #define WIDGET_SWITCH_EN 1 //允许使用控件: Switch复选开关
40 #define WIDGET_EDIT_EN 1 //允许使用控件: Edit文本编辑
41
42 #define WIDGET_TEXT_EN 1 //允许使用控件: Text文字
43 #define WIDGET_BAR_EN 1 //允许使用控件: Bar条
44 #define WIDGET_SPECTRUM_EN 1 //允许使用控件: Spectrum谱
45 #define WIDGET_OSCILLOGRAM_EN 1 //允许使用控件: Oscillogram波形
46 #define WIDGET_PICTURE_EN 1 //允许使用控件: Picture图
47

```

选项及按键 FIFO 参数设置

OPTIONS_MAX: 每个页面可以容纳的最大选项数目。

FIFOBUF_SIZE: 可以缓存的键值数。

这两个参数默认即可。如果需要每页显示更多的选项，可以设置大些，但不宜太大，否则操作会很麻烦。设置时需要根据目标硬件的具体情况设置。

```
48 //-----【选项最大数目】
49 #define OPTIONS_MAX          16          //每页最大可以存放的选项数（不可设置太大）
50
51 //-----【FIFO键值缓存大小】
52 #define FIFOBUF_SIZE         5          //可以缓存的键值数（默认即可）
53
```

键值对应行为设置

此处需根据目标系统按键返回的键值来设置。一个键值对应一个行为，如果目标系统按键可以输出多键值（短按，长按），那么实际的按键数可以大大减少。

```
54 //-----【键值对应行为设置】
55 /*-----用户需根据系统按键返回的键值来进行设定-----*/
56 #define SOLGUI_KEY_UP        0x50        //上
57 #define SOLGUI_KEY_DOWN      0x20        //下
58 #define SOLGUI_KEY_LEFT     0x30        //左
59 #define SOLGUI_KEY_RIGHT     0x10        //右
60 #define SOLGUI_KEY_OK        0x40        //确认
61 #define SOLGUI_KEY_BACK      0x60        //返回
62
```

图标设置

某些选项式控件会有图标，此处为图标在字符数组中对应的下标，不需要修改。具体的图标样式可以在 Font6x8_ASCII.c 中设置。

```
62
63 //-----【系统图标设置】
64 /*-----如果用户不添加自定义图标，此处默认即可，不需修改-----*/
65 #define ICON_UP              0x80        //上（可以在Font6x8_ASCII.c中设置）
66 #define ICON_DOWN           0x81        //下（可以在Font6x8_ASCII.c中设置）
67 #define ICON_LEFT           0x82        //左（可以在Font6x8_ASCII.c中设置）
68 #define ICON_RIGHT          0x83        //右（可以在Font6x8_ASCII.c中设置）
69 #define ICON_OK              0x84        //OK（可以在Font6x8_ASCII.c中设置）
70 #define ICON_BACK           0x85        //返回（可以在Font6x8_ASCII.c中设置）
71 #define ICON_CURSOR          0x86        //光标（可以在Font6x8_ASCII.c中设置）
72 #define ICON_WIDGET_GOTOPAGE 0x87        //页面跳转控件（可以在Font6x8_ASCII.c中设置）
73 #define ICON_WIDGET_EDIT     0x88        //文本编辑控件（可以在Font6x8_ASCII.c中设置）
74 #define ICON_OTHER_HIDE      0x89        //省略号（可以在Font6x8_ASCII.c中设置）
75
```

SPIN 控件和 EDIT 控件参数设置

SPIN_DIGIT_MAX: 可在 SPIN 中可设置的上限位数。即 SPIN 控件可步进的最大值，此处设置为 5，表示最大步进值为 100000。超过 5 时会出错。

SPIN_DIGIT_MIN: 可在 SPIN 中可设置的下限位数。即 SPIN 控件可步进的最小值，此处设置为-3，表示最小步进值为 0.001（ 10^{-3} ）。

EDIT_BUF_SIZE: 在 EDIT 中字符缓存的最大长度。EDIT 控件在修改字符串时会将字符串预先复制到一个字符缓存中，此处即修改字符缓存的大小。此处的（40+1）表示字符缓存大小为“40 个可修改字符+‘\0’”，‘\0’预先占位防止修改掉结束位后字符串操作进入死循环。

EDIT_THUMBNAIL_SIZE: 在 EDIT 选项中缩略信息缓存的最大长度。EDIT 控件在显示为选项时会把字符缓存的缩略的显示出来。此处为缩略信息长度，默认即可。

```

75
76 //-----【SPIN控件：上下限位数设置】
77 #define SPIN_DIGIT_MAX      5          //可在SPIN中可设置的上限位数（超过5位会出错）
78 #define SPIN_DIGIT_MIN      (-3)       //可在SPIN中可设置的下限位数（精度10^-3）
79
80 //-----【EDIT控件：缓存大小设置】
81 #define EDIT_BUF_SIZE       (40+1)     //在EDIT中字符缓存的最大长度（128*64下最大94）
82 #define EDIT_THUMBNAI_SIZE  7          //在EDIT选项中缩略信息缓存的最大长度
83

```

6.2 文档版本管理

文档版本号	SOLGUI 版本号	时间	说明
V0.0.0001	V2.0.0004	2016-3-4	文档完成
V0.0.0002	V2.0.0005	2016-3-13	SPIN 控件 bug 修复
V0.0.0003	V2.0.0006	2016-4-12	
V0.0.0004	V2.0.0007	2016-4-17	OPTIONTEXT 控件 bug 修复
V0.0.0005	V2.0.0008	2016-5-24	Printf 函数 bug 修复

注：SOLGUI 版本号含义说明：

SOLGUI 版本号使用 A.B.CCCC 格式，A 为核心版本号，改变时表示重大更改。B 为子版本号，改变时表示为功能添加完善，优化等，其中偶数代表开发版，技术代表稳定版。CCCC 为次版本号，表示小幅度修改。