



SimpleGUI

移植范例

摘要

以 Keil MDK 平台下的 STM32 工程为例，演示如何将 SimpleGUI 的演示程序移植到需要的目标平台上。

Polarix

Xuyulin91@163.com

1、环境准备

本说明将在 Keil MDK 环境下，以 STM32 的工程为基础，对 SimpleGUI 的演示例程进行移植，并对 API 的使用做简单描述。

在开始进行移植之前，需要先建立一个可用于编译目标平台的工程，以及在目标平台上适配的，将要使用的屏幕驱动程序，屏幕驱动程序要求至少有读取、写入点的接口。此过程视最终使用的平台不同而有所不同，此处不予以详述。

在本范例中，将使用 Keil MDK 5 下的 STM32F103ZET6 工程为基础，使用 SPI 总线驱动的 SSD1306 主控制器的 0.96 寸 OLED12864 显示屏。其他实验平台请参照描述原理自行分析编写。

2、文件结构

进入 SimpleGUI 的根目录，各目录具体描述如下：

DemoProc	SimpleGUI 应用演示代码
DemoProject	移植演示工程。
Documents	说明文档
GUI	绘图引擎接口实现
HMI	交互引擎接口实现
SimulatorEnv	模拟环境工程

其中演示工程中需要的必要资源为 GUI、MHI 和 DemoProc 三个文件夹。

3、组织工程

打开 Keil MDK 并载入之前准备好的工程，工程需要确保有适配可用的显示屏的驱动程序。驱动程序至少需要具备读写点（像素）的接口，在对效率没有需求的前提下，其他功能均可以通过这两个接口组合实现。

由于演示工程中除了演示最基本的绘图功能，还包含有简单的屏幕更新与交互功能，所以还需要占用目标平台上的一个定时器与一个串口，定时器需要每 10ms 触发一次中断，串口也需要启用接收中断，用以模拟用户的按键操作等。此外，演示工程中还包含系统时钟的相关内容，如果目标平台上包含 RTC 功能及相关电路，且用户想实现相关效果，那么请做好相应的实装并开启 RTC 中断。

4、驱动配置

目前市面上绝大多数的单色点阵显示屏（LCD、OLED 等）都具有串行（SPI 或 IIC）与并行两种方式，很多屏幕在串行驱动模式下，是不支持读操作的，这时就需要在程序中为屏幕显示开辟显示缓存。而且屏幕本身也很少有支持对单一像素点的读写操作，通常以八个像素点为一个页，以页为单位进行操作，而对屏幕寄存器的读操作通常没有写操作的效率高，所以在修改像素点时，修改本地缓存然后写入屏幕通常要比读取屏幕-修改-写入的效率高很多。

以淘宝上常见的 SSD1306 主控制器的 OLED 显示屏模块为例，显示分辨率为 128*64 像素，纵向 8 像素为 1 页，全屏幕共 128*8（1024）个显示寄存器单元。这时候就可以在本地图明一个字节型 8*128 的二维数组作为显示缓存用以支持以像素为单位的屏幕操作，范例如下：

```
uint8_t arruiDisplayCache[128][64];
```

然后，对屏幕上的像素点进行更新时，就可以按照如下方法进行操作。

```

// 位操作宏定义
#define SET_PAGE_BIT(PAGE, Bit)      ((PAGE) = (PAGE) | (0x01 << (Bit)))
#define CLR_PAGE_BIT(PAGE, Bit)      ((PAGE) = (PAGE) & ~(0x01 << (Bit)))

//写点函数
void OLED_SetPixel(uint16_t uiPosX, uint16_t uiPosY, OLED_COLOR eColor)
{
    if((uiPosX < LCD_SIZE_WIDTH) && (uiPosY < LCD_SIZE_HEIGHT))
    {
        // Set point data.
        if(OLED_COLOR_FRG == eColor)
        {
            SET_PAGE_BIT(arruiDisplayCache[uiPosY/8][uiPosX], uiPosY%8);
        }
        else
        {
            CLR_PAGE_BIT(arruiDisplayCache[uiPosY/8][uiPosX], uiPosY%8);
        }
    }
}

//读点函数
uint16_t OLED_GetPixel(uint16_t uiPosX, uint16_t uiPosY)
{
    if((uiPosX < LCD_SIZE_WIDTH) && (uiPosY < LCD_SIZE_HEIGHT))
    {
        return GET_PAGE_BIT(arruiDisplayCache[uiPosY/8][uiPosX], uiPosY%8);
    }
    else
    {
        return 0;
    }
}

```

以上完成的是对显示缓存内数据的修改，要使修改的内容显示在屏幕上，需要将缓存中修改的内容同步到显示屏中，最简单的办法就是全屏刷新。此外，还可以对修改的单元及范围进行记录，局部更新屏幕以提升屏幕的刷新操作效率。用户可根据目标平台自行定制更新策略，在此不做详述。

5、 移植概要

SimpleGUI 的移植非常简单，下表中列出的是 SimpleGUI 在移植过程中需要用户修改和实现的所有函数：

文件	函数	说明
SGUI_Basic.c	SGUI_Basic_DrawPoint	绘制点：设定屏幕上一个像

		素的状态，需要用户驱动程序支持。
	SGUI_Basic_GetPoint	读取点：获取屏幕上一个像素的状态，需要用户驱动程序支持。
	SGUI_Basic_ClearScreen	清空屏幕：用于清除所有屏幕显示需要用户驱动程序支持，如果没有此接口，可以通过 SGUI_Basic_DrawPoint 接口实现。
	SGUI_Basic_RefreshDisplay	更新屏幕显示：在使用屏幕缓存的场合下，用于将缓存的内容同步到屏幕上。
SGUI_Common.c	SGUI_Common_Allocate	动态内存申请：于堆或内存池中申请一块内存，等同于标准库中的 malloc 函数。
	SGUI_Common_Free	动态释放内存：释放一块已申请的内存，等同于标准库中的 free 函数。
	SGUI_Common_MemoryCopy	内存块复制：复制指定大小的内存块到新地址，等同于标准库中的 memcpy 函数。
	SGUI_Common_MemorySet	设定内存值：设置内存块中所有内存单元的值，等同于标准库中的 memset。
	SGUI_Common_StringLength	测量字符串长度：等同于标准库中的 strlen。
	SGUI_Common_StringCopy	字符串复制：等同于标准库中的 strcpy
	SGUI_Common_StringLengthCopy	复制指定长度字符串：等同于标准库中的 strncpy。
	SGUI_Common_GetNowTime	获取当前时间：如果用户的芯片或电路中有 RTC 支持，可以在此函数中加入对 RTC 驱动的引用，以获取系统时间。
	SGUI_Common_ReadFlashROM	读取 Flash 数据：如果用户将字库等数据信息存储与片外 Flash 上，可以将 Flash 的驱动程序于此处实现，用于读取外部数据。
	SGUI_Common_Delay	延时函数：简单延时，此函数在 SimpleGUI 中没有引用。

通过上表可知，SimpleGUI 需要移植的内容都在 SGUI_Basic.c 和 SGUI_Common.c 两

个文件中，SGUI_Basic.c 中需要移植的是驱动程序，用于 SimpleGUI 的逻辑处理与驱动程序之间的连接，SGUI_Common.c 中需要移植的是一些系统平台的相关函数。为了防止有些情况下不方便或不能使用标准库或微库(MicroLib)，SimpleGUI 中将用到的一些系统函数进行了重新封装，方便用户在必要时自行实现。

另外，SimpleGUI 中的部分机能受以下四个全局宏定义控制：

_SIMPLE_GUI_ENABLE_DYNAMIC_MEMORY_	动态内存使能：若此宏被定义且值大于 0，则 SimpleGUI 将认为所在平台能够支持动态申请与释放内存，列表的列表项动态增减功能将被使能。
_SIMPLE_GUI_ENABLE_BASIC_FONT_	基础字体：SimpleGUI 内部包含了一个 6*8 像素的 ASCII 字库，设计目的是为了在使用外部字库时，如果外部字库发生损坏，还可以通过内部字库和设计保障在屏幕上输出一些错误信息，有助于调试与排查。若此宏被定义且值大于 0，则内部的这组基础 ASCII 字库被设置为有效。
_SIMPLE_GUI_VIRTUAL_ENVIRONMENT_SIMULATOR_	模拟器环境：若此宏被定义且值大于 0，则被认定为运行于 SimpleGUI 模拟器环境中。
_SIMPLE_GUI_ENABLE_ICONV_GB2312_	UTF-8 转码：在模拟器环境中，为了防止在不同语言环境和系统中出现乱码等现象，代码和模拟环境中的资源均使用 UTF-8 编码，若此宏被定义且值大于 0，那么所有字符串在处理前都会被转换成 GB2312 编码。此宏定义仅在 _SIMPLE_GUI_VIRTUAL_ENVIRONMENT_SIMULATOR_ 宏定义有效的前提下有效。

用户可以根据上述信息，根据自己的需要配置和编译 SimpleGUI。例如如果用户对动态内容没有需求，例如不需要列表对列表项进行动态增减，则 SGUI_Common.c 中的 SGUI_Common_Allocate 函数和 SGUI_Common_Free 函数是用不到的，可以不予实现。

6、移植演示工程

明确一直工作需要做的事情后，接下来就可以开始对演示工程进行移植了。

进入 SimpleGUI 根目录下，将 DemoProc、GUI 和 HMI 三个文件夹下的内容。将这三个文件夹复制一份到创建好的 Keil MDK 工程目录中。

名称	修改日期	类型	大小
Core	2016/8/25 19:12	文件夹	
DemoProc	2018/8/12 0:03	文件夹	
Device	2016/8/25 19:12	文件夹	
FWLibrary	2016/8/25 19:12	文件夹	
GUI	2018/8/12 0:03	文件夹	
HMI	2018/8/12 0:03	文件夹	
Listing	2018/8/11 22:26	文件夹	

图 1 复制必要文件到工程目录

然后将 DemoProject\STM32F1\Demo 文件夹下的所有内容复制到 DemoProc 文件夹中，然后将这三个文件中的源码文件全部加入 KeilMDK 的工程。

GUI 目录下的所有文件：

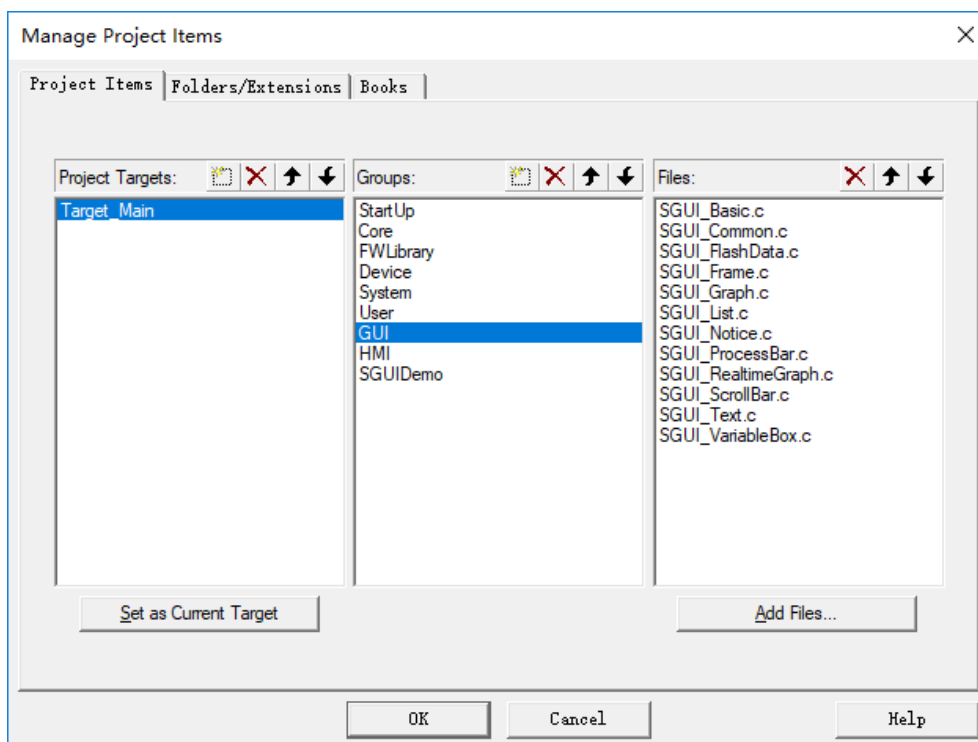


图 2 GUI 文件夹下的源码文件

HMI 目录下的所有文件：

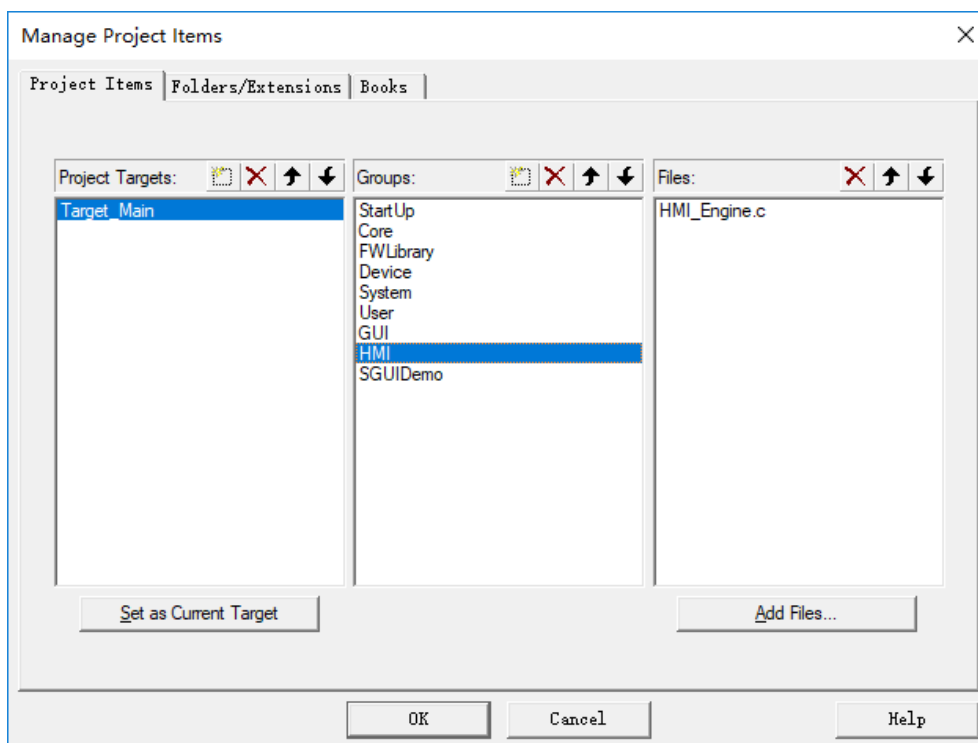


图 3 HMI 文件夹下的源码文件

DemoProc 目录下的所有文件：

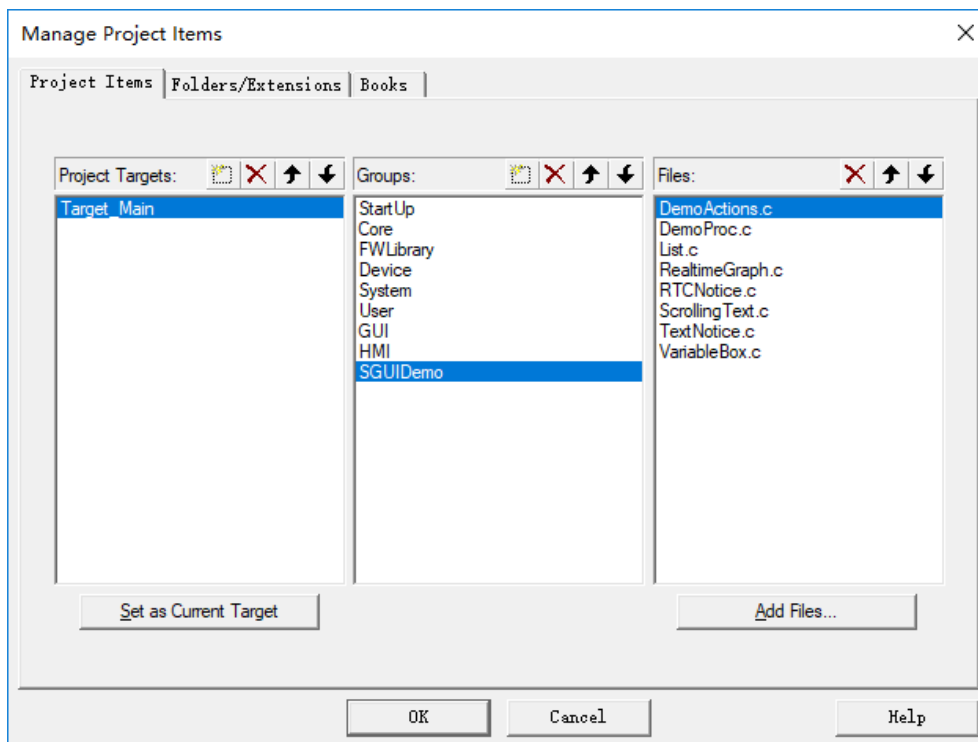


图 4 DemoProc 文件夹下的源码文件

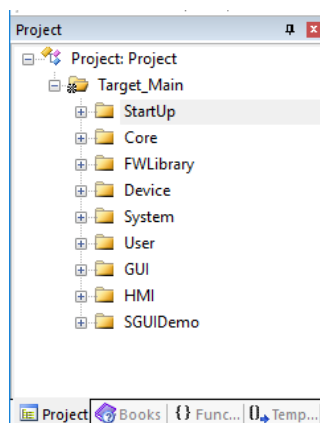


图 5 加入工程的 SimpleGUI 和演示程序

然后进入工程选项中，将添加的头文件路径加入包含列表中，以便 include 能够正常包含演示文件中的头文件。

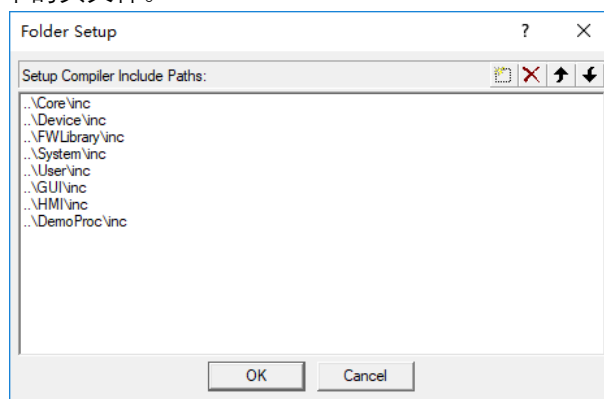


图 6 添加包含路径

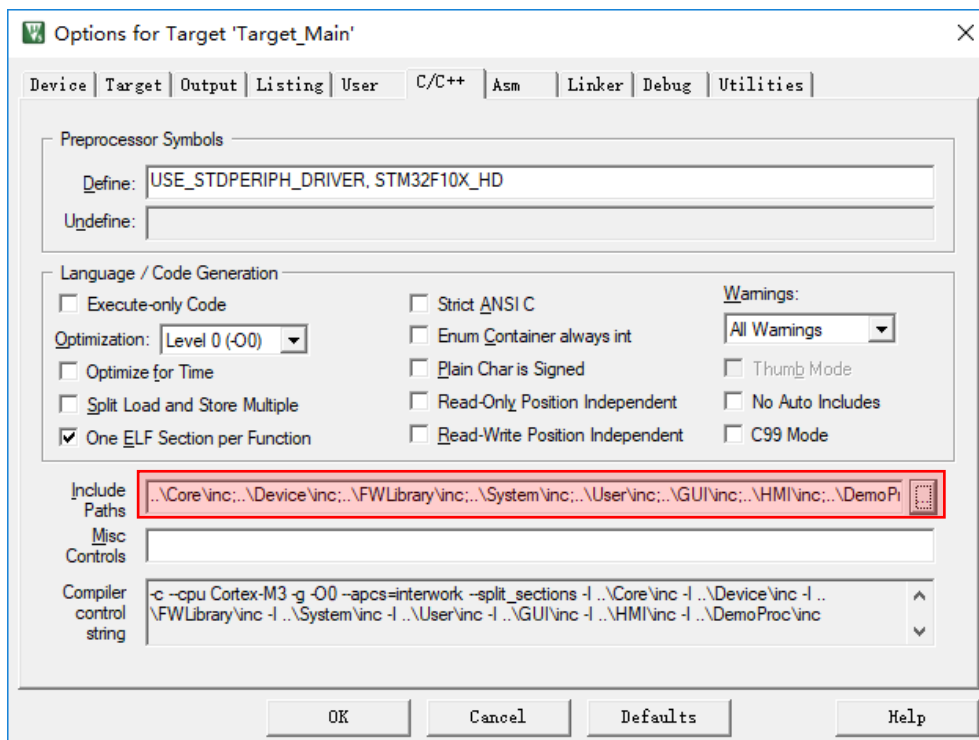


图 7 添加包含路径

至此，演示工程需要的所有文件部署到位，可以尝试第一次编译了。按下键盘上的 F7 键开始编译。

编译后弹出的错误与警告如下。

..\GUI\src\SGUI_Common.c(36): error: #5: cannot open source input file "RTC.h": No such file or directory
..\DemoProc\src\List.c(35): warning: #177-D: variable "s_arrszEnumedValue" was declared but never referenced
..\DemoProc\src\DemoActions.c(9): error: #5: cannot open source input file "RTC.h": No such file or directory

第一个错误出现在 SGUI_Common.c 文件夹中，提示 RTC.h 文件未被包含。这是由于演示工程中有显示时间的相关代码，需要片上或片外时钟设备支持。如果用户的平台上没有此类功能或电路，可以修改此文件中的 SGUI_Common_GetNowTime 函数的实现，将输出值固定化以避免此问题。

由于我们使用的 STM32 单片机是具有片上 RTC 的，可以用于获取系统时间，所以我们添加 RTC 相关功能的实现函数，使此功能有效化，关于 STM32 片上 RTC 的相关配置，请自行在网络上搜索并实现，或参考配套例程，此处不再赘述。

第二个是一个警告，这是因为模拟环境与示例工程公用一套演示工程，但使用的资源文件编码并不一致，所以在此有所区分。此处是由于宏定义导致的警告，可以无视。

第三处的错误与第一处相同，第一处处理完成后，第三处也不会再出现了。

以上错误修正后，可以重新按下 F7 键，再次尝试编译。

编译后回弹出以下错误。

..\DemoProc\src\DemoActions.c(48): error: #140: too many arguments in function call

此处错误是由于示例工程自定义了串口初始化函数，由于示例工程使用了串口来代替按键输入来进行交互上的模拟，所以需要串口的支持，所以在这里需要您自行实现串口的初始化操作，并正确配置串口中断。

修正完串口的初始化与实现后，再次编译，已经没有错误，这说明基本框架已经没有了，接下来开进行整合。

首先，需要将显示屏的读点、写点函数对应到 SimpleGUI 的接口中去。这些接口位于 SGUI_Basic.c 文件中，包括 SGUI_Basic_DrawPoint、SGUI_Basic_GetPoint 和 SGUI_Basic_ClearScreen 三个函数，分别对应写像素点读像素点和清屏幕三个操作，如果没有专门的清屏幕操作，可以通过写点操作实现。

以下为以本范例使用平台为基础的实现：

```
//写像素点
void SGUI_Basic_DrawPoint(SGUI_UINT uiCoordinateX, SGUI_UINT uiCoordinateY,
SGUI_COLOR eColor)
{
    if((uiCoordinateX < LCD_SIZE_WIDTH) && (uiCoordinateY < LCD_SIZE_HEIGHT))
    {
        if(SGUI_COLOR_FRGCLR == eColor)
        {
            OLED_SetPixel(uiCoordinateX, uiCoordinateY, OLED_COLOR_FRG);
        }
        else if(SGUI_COLOR_BKGCLR == eColor)
        {
            OLED_SetPixel(uiCoordinateX, uiCoordinateY, OLED_COLOR_BKG);
        }
    }
}

//读像素点
SGUI_COLOR SGUI_Basic_GetPoint(SGUI_UINT uiCoordinateX, SGUI_UINT
uiCoordinateY)
{
    if((uiCoordinateX < LCD_SIZE_WIDTH) && (uiCoordinateY < LCD_SIZE_HEIGHT))
    {
        uiPixValue = OLED_GetPixel(uiCoordinateX, uiCoordinateY);
        if(0 == uiPixValue)
        {
            eColor = SGUI_COLOR_BKGCLR;
        }
        else
        {
            eColor = SGUI_COLOR_FRGCLR;
        }
    }
    return eColor;
}

//清屏幕
void SGUI_Basic_ClearScreen(void)
{

```

```
OLED_ClearDisplay();  
}
```

此处移植完成后，SimpleGUI 与显示屏设备之间的连接就基本完成了。由于对像素点的操作是通过对显示缓存中的数据进行位操作来完成，所以还需要实现缓存同步到屏幕的 SGUI_Basic_RefreshDisplay 接口，在用户全部修改完要显示的屏幕内容后，调用此接口以将修改的内容同步显示到屏幕上。

至此，SimpleGUI 的相关接口已经可以正常通过屏幕显示内容了。

接下来，在 main 函数中添加以下代码：

```
int main(void)  
{  
    //初始化系统  
    SystemInit();  
    //初始化串口  
    Initialize_Serial(115200);  
    //初始化显示屏  
    OLED_Initialize();  
  
    //初始化 HMI 引擎  
    InitializeEngine();  
    printf("HMI engine Initialized.\r\n");  
  
    //模拟触发事件  
    while(1)  
    {  
        DemoAction_TimerEvent();  
    }  
}
```

然后编译工程并烧录到单片机，就已经可以看到初步效果了。

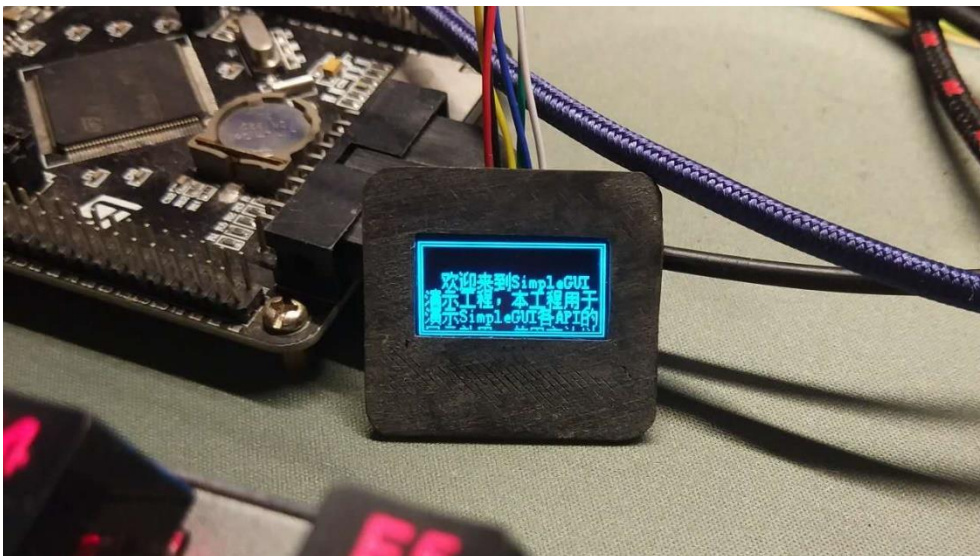


图 8 初步移植效果

此时可以看到欢迎屏幕上的滚动文字，但是交互功能依然不能演示，接下来就需要

对交互功能进行移植。

在移植交互功能之前，还需要用户根据自身平台，启用一个定时器并配置相应中断，中断需要可以查询到触发状态，以便主程序可以根据触发状态进行相应动作。

在本演示程序中，配置并使用 STM32F103 的 Timer3 定时器，定时周期 10ms，提供 GetTimerTriggered 和 ResetTimerTriggered 两个接口函数用于查询和重置定时器的触发状态。添加定时器的相关处理后，主程序变成如下的样子。

```
int main(void)
{
    //初始化系统
    SystemInit();
    //初始化串口
    Initialize_Serial(115200);
    //初始化显示屏
    OLED_Initialize();

    //初始化 HMI 引擎
    InitializeEngine();
    printf("HMI engine Initialized.\r\n");

    while(1)
    {
        //定时器事件
        rue == GetTimerTriggered()
        {
            DemoAction_TimerEvent();
            ResetTimerTriggered();
        }
    }
}
```

接下来就可以开始配置串口中断了。范例程序中，用串口输入模拟用户输入，每次 1 字节，每字节高四位为控制键码，低四位为主键码。本范例中针对模拟简码的定义位于 DemoActions.h 文件中，定义如下

```
// 主键码.
#define KEY_VALUE_TAB (0x01)
#define KEY_VALUE_ENTER (0x02)
#define KEY_VALUE_ESC (0x03)
#define KEY_VALUE_SPACE (0x04)
#define KEY_VALUE_LEFT (0x05)
#define KEY_VALUE_UP (0x06)
#define KEY_VALUE_RIGHT (0x07)
#define KEY_VALUE_DOWN (0x08)

//控制键码，用于模拟 ALT/CTRL/SHIFT
#define KEY_OPTION_CTRL (0x10)
```

```
#define KEY_OPTION_ALT (0x20)
#define KEY_OPTION_SHIFT (0x40)
```

与定时器的处理类似，串口接收中断配置完成后，也需要一个查询和重置的接口。本示例定义 GetReveivedByte 和 ResetReveivedByte 接口用于获取最后一个接收的字节和重置接收变量。添加串口相关处理后，主程序代码如下。

```
int main(void)
{
    //初始化系统
    SystemInit();
    //初始化串口
    Initialize_Serial(115200);
    //初始化显示屏
    OLED_Initialize();

    //初始化 HMI 引擎
    InitializeEngine();
    printf("HMI engine Initialized.\r\n");

    while(1)
    {
        //定时器事件
        if(true == GetTimerTriggered())
        {
            DemoAction_TimerEvent();
            ResetTimerTriggered();
        }
        //串口接收事件
        cbReceivedByte = GetReveivedByte();
        if(KEY_NONE != cbReceivedByte)
        {
            printf("Received virtual key value 0x%02X.\r\n", cbReceivedByte);
            DemoAction_UsartReceiveEvent(cbReceivedByte);
            ResetReveivedByte();
        }
    }
}
```

至此，示例程序基本完成，重新编译工程，烧录后即可看到效果，打开串口助手，发送 0x04，画面即变更至列表演示画面，其他操作可以根据 DemoActions.h 文件中对键码的定义逐一实验。

7、 驱动优化

未完待续。

8、 联系开发者

首先，感谢您对 SimpleGUI 的赏识与支持。

虽然最早仅仅作为一套 GUI 接口库使用，但我最终希望 SimpleGUI 能够为您提供一套完整的单色屏 GUI 及交互设计解决方案，如果您有新的需求、提议亦或想法，可以联系 QQ 326684221 或电子邮件 xuyulin91@163.com，也可以在以下地址留言：

SimpleGUI@开源中国：<https://www.oschina.net/p/simplegui>

SimpleGUI@码云：<https://gitee.com/Polarix/simplegui>

本人并不是全职的开源开发者，依然有工作及家庭的琐碎事务要处理，所以对于大家的需求和疑问反馈的可能并不及时，多有怠慢，敬请谅解。

最后，再次感谢您的支持。