物联网工程技术丛书

物联网技术实验教程

陈邦琼 张小全 李湃 编著

清华大学出版社

物联网工程技术丛书

物联网技术实验教程

陈邦琼 张小全 李湃 编著

清华大学出版社
北京

内 容 简 介

本书是一本物联网应用技术实验教程，包括 μC/OS 基础实验、ZigBee Pro 协议、STM32W108、Wi-Fi 通信、GPRS 通信、物联网应用设计与开发 6 章，共 25 个实验。多数实验包括实验目的、实验设备、实验要求、实验原理、实验步骤和范例路径 6 个部分。全书以北京凯威利亚科技发展有限公司开发的 STM32 型物联网教学/开发/实训平台为基础，以实验的形式，向读者介绍物联网工程技术开发与应用的各技术模块。

本书可作为高等院校、各类职业院校物联网应用技术专业的教材，也可作为物联网相关技术工程人员的参考书。

图书在版编目(CIP)数据

物联网技术实验教程/陈邦琼等编著. --北京：清华大学出版社，2013
物联网工程技术丛书
ISBN 978-7-302-32066-1

Ⅰ. ①物… Ⅱ. ①陈… Ⅲ. ①互联网络－应用 ②智能技术－应用 Ⅳ. ①TP393.4 ②TP18

中国版本图书馆 CIP 数据核字(2013)第 078788 号

责任编辑：孟毅新
封面设计：傅瑞学
责任校对：李　梅
责任印制：杨　艳

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课 件 下 载：http://www.tup.com.cn，010-62795764
印 刷 者：北京富博印刷有限公司
装 订 者：北京市密云县京文制本装订厂
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：13　　**字　　数**：296 千字
版　　次：2013 年 9 月第 1 版　　**印　　次**：2013 年 9 月第1 次印刷
印　　数：1～3000
定　　价：30.00 元

产品编号：050619-01

FOREWORD

前言

本书是根据北京凯威利亚科技发展有限公司开发的物联网教学/开发/实训平台而设计的，主要针对物联网工程技术的综合应用。书中安排的实验均是 STM32 教学实验平台下的功能使用，且附有范例程序。实验内容浅显易懂，读者通过这些实验的学习，可以逐步掌握 STM32 嵌入式教学平台在嵌入式操作系统 μC/OS 下的物联网设计开发与应用。

本书中的所有实验范例代码均经过调试。实验时按照硬件连接说明进行连接后，程序可直接下载运行，使读者节省时间，能够快速入门。

下面对实验目的做一个说明。

了解：初步认识，不作特别要求，也可以不体现出来。

学习：一些比较重要，但短时间掌握不了的，可以作为学习要求。

熟悉：主要是对实验过程和步骤加以说明，要求比较清晰整个过程。

掌握：作为实验的重点知识，要求学生必须有一定的认识。

本书由全国电子行业职业教育能力建设促进基地和全国物联网工学一体化教育工程办公室主持编写，由陈邦琼、张小全、李湃执笔。由于编者水平有限，书中难免有不足之处，恳请读者批评指正，我们将虚心接受并尽快进行更改或再版。

郑重声明：本实验手册和实验产品仅作为教学、学习使用，所有产品不得用于医疗器材、维持生命系统及航空飞行等相关设备。

编　者

2013 年 8 月

CONTENTS

目录

第1章 μC/OS 基础实验

1.1 多任务建立实验

【实验目的】

(1) 熟悉 MDK 集成开发环境的使用。
(2) 熟悉北京凯威利亚科技发展有限公司嵌入式实验箱 STM32。
(3) 掌握 μC/OS 多任务建立。
(4) 掌握 μC/GUI 基本绘图函数的使用。
(5) 通过一个模拟交通灯系统初步了解整个系统基本工作原理。
(6) 学习 μC/GUI 的基本框架。
(7) 掌握多任务建立相关接口函数使用方法。

【实验设备】

(1) 装有 MDK IDE 环境和 J-LINK 驱动的 PC 一台。
(2) 北京凯威利亚科技发展有限公司嵌入式实验箱 STM32 一台。
(3) J-LINK 调试器一块。
(4) 本实验用到实验箱的模块有：LCD 模块、触摸屏模块。

【实验要求】

(1) 实现功能：建立两个任务，分别监听触摸屏消息和交通灯状态维护，拥堵时期可以人为参与，手动实时切换交通通行状态。

(2) 实验现象：两条主干道上车辆通行(绿灯)时间均为 4.8s，停止通行(红灯)时间为 4.8s，红绿灯切换(黄灯)警告时间为 0.8s，并交替循环下去；当出现交通拥堵时，通过按键 tamper 或触摸屏手动实时快速切换通行状态。

【实验原理】

1. 硬件原理

本实验采用具有 SPI 接口的电阻式触摸屏，内含专业高精度触摸屏控制芯片(RSM1843)；LCD 采用 2.8 英寸或 3.2 英寸 TFT 标准屏，分辨率为 320×240 像素，16 位色深，镜面屏，超高清，支持 8/16 位总线接口。硬件原理参考图 1.1。

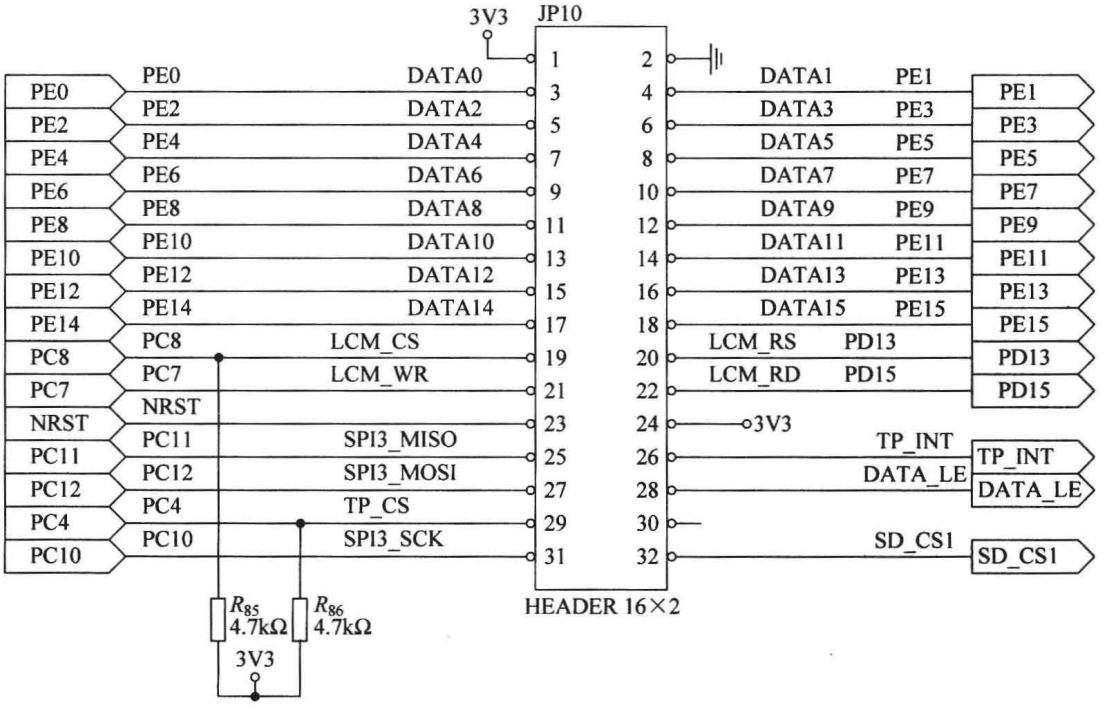

图 1.1 TFT-LCD 硬件连接电路图

LCD 数据线连接到 STM32F107R 的 PE 口，PD 和 PC 部分 I/O 作为控制信号线。

PE0～PE15 <……> DB0～DB15
PD15 <……> nRD
PD14 <……> RS
PD13 <……> nWR
PD12 <……> nCS
PD11 <……> nReset
PC0 <……> BK_LED

由上面可以看出，对 LCD 的操作这里采用的是并行数据传输，对 LCD 寄存器的读写操作时，只要按照读写时序，配置好各 I/O，直接读写 PE 端口即可。

图 1.2 是读取控制器数据时序图。

图 1.3 是写指定地址寄存器时序图。

触摸屏采用 SPI 接口，由于 SPI 时序比较简单，这里既可以采用 I/O 模拟 SPI 时序，也可以采用 STM32F107R 自带 SPI3 控制器，本实验采用自带 SPI3 控制器，并通过查询

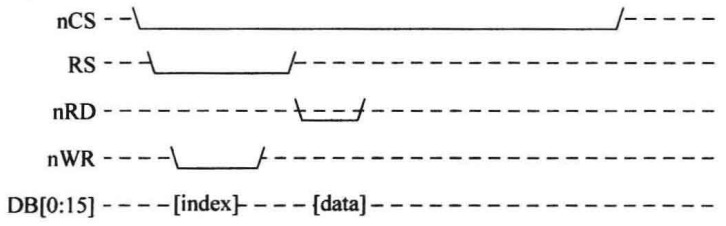

图 1.2　读取控制器数据时序图

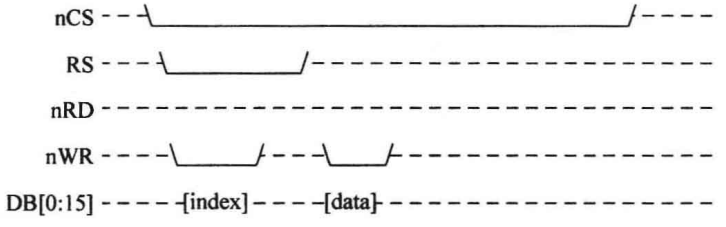

图 1.3　写指定地址寄存器时序图

获取点击状态。

PD5　　<……> TS_CLK
PD6　　<……> TS_nCS
PD7　　<……> TS_DIN
PD8　　<……> TS_BUSY
PD9　　<……> TS_nPENIRQ
PD10　　<……> TS_DOUT

2. 代码分析

主程序首先完成系统相关时钟和硬件资源的初始化，接着创建两个任务，一个用于查询触摸屏状态，一个通过 μC/GUI 提供的 LCD 标准操作接口，维护模拟交通状态图，两个任务共同完成了一个模拟交通灯系统。

(1) 头文件

```
#include "includes.h"
#include "GUI.h"
#include "GUIDEMO.h"
#include "Widget.h"
#include "utilities.h"
```

(2) 主程序

```
int main (void)
{
    INT8U err;
```

```
    NVIC_SetVectorTable(NVIC_VectTab_FLASH, 0x0);
    OSInit(); /* Initialize "μC/OS-II, The Real-Time Kernel" */

    /* Create the start task */
    OSTaskCreateExt(AppTaskStart,
                    (void *)0,
                    (OS_STK *)&AppTaskStartStk[APP_TASK_START_STK_SIZE-1],
                    APP_TASK_START_PRIO,
                    APP_TASK_START_PRIO,
                    (OS_STK *)&AppTaskStartStk[0],
                    APP_TASK_START_STK_SIZE,
                    (void *)0,
                    OS_TASK_OPT_STK_CHK|OS_TASK_OPT_STK_CLR);

    #if (OS_TASK_NAME_SIZE > 13)
        OSTaskNameSet(APP_TASK_START_PRIO, "Start Task", &err);
    #endif
    OSStart();  /* Start multitasking (i.e. give control to μC/OS-Ⅱ) */
}
```

主程序首先设置中断向量表，然后调用 OSInit()完成与 OS 相关的初始化动作，并调用 OSTaskCreateExt()创建一个 AppTaskStart()任务，由该任务完成后继续任务建立和系统初始化工作，最后通过 OSStart()函数启动多任务系统。

(3) 系统初始化任务 AppTaskStart()

```
static  void  AppTaskStart (void *p_arg)
{
    (void)p_arg;

    BSP_Init();                 /* Initialize BSP function   */
    #if (OS_TASK_STAT_EN > 0)
    OSStatInit();               /* Determine CPU capacity    */
    #endif
    AppTaskCreate();            /* Create application tasks */

    while(DEF_TRUE)
    {
        /* Task body, always written as an infinite loop.  */
        OSTaskSuspend(OS_PRIO_SELF);
    }
}
```

AppTaskStart()首先通过 BSP_Init()对平台时钟、中断、GPIO、LCD、触摸屏等资源进行初始化，然后调用 AppTaskCreate()函数创建用户交互任务，而 AppTaskStart()任务最后通过 OSTaskSuspend(OS_PRIO_SELF)将自己挂起，即进入空闲状态。

(4) 用户交互任务 AppTaskCreate()

```
static  void  AppTaskCreate(void)
{
    INT8U  err;

    OSTaskCreateExt(AppTaskUserIF,
                    (void * )0,
                    (OS_STK * )&AppTaskUserIFStk[APP_TASK_USER_IF_STK_SIZE - 1],
                    APP_TASK_USER_IF_PRIO,
                    APP_TASK_USER_IF_PRIO,
                    (OS_STK * )&AppTaskUserIFStk[0],
                    APP_TASK_USER_IF_STK_SIZE,
                    (void * )0,
                    OS_TASK_OPT_STK_CHK|OS_TASK_OPT_STK_CLR);

    # if (OS_TASK_NAME_SIZE > 8)
        OSTaskNameSet(APP_TASK_USER_IF_PRIO, "User I/F", &err);
    # endif

    OSTaskCreateExt(AppTaskKbd,
                    (void * )0,
                    (OS_STK * )&AppTaskKbdStk[APP_TASK_KBD_STK_SIZE - 1],
                    APP_TASK_KBD_PRIO,
                    APP_TASK_KBD_PRIO,
                    (OS_STK * )&AppTaskKbdStk[0],
                    APP_TASK_KBD_STK_SIZE,
                    (void * )0,
                    OS_TASK_OPT_STK_CHK|OS_TASK_OPT_STK_CLR);

    # if (OS_TASK_NAME_SIZE > 8)
        OSTaskNameSet(APP_TASK_KBD_PRIO, "Keyboard", &err);
    # endif
}
```

在本函数中主要完成了两个用户事件处理任务的创建，AppTaskKbd()用于监听触摸屏，AppTaskUserIF()用于维护模拟交通状态变化，而 μC/GUI 实时根据消息事件更新窗口。

(5) NVIC_SetVectorTable()函数

设置中断向量表，如果要用到中断，就得设置这张表。

(6) OSInit()和 OSStart()函数

OSInit()初始化 μC/OS-Ⅱ，对这个函数的调用必须在调用 OSStart()函数之前，而 OSStart()函数真正开始运行多任务，用户必须在开始多任务调度后(即调用 OSStart()后)允许时钟节拍中断。换句话说，就是用户应该在 OSStart()运行后，μC/OS-Ⅱ启动运行的第一个任务中初始化节拍中断。通常所犯的错误是在调用 OSInit()和 OSStart()之间允许时钟节拍中断。

(7) BSP_Init()函数

这个函数主要完成与硬件相关初始化动作，这是第一个任务必须要做的事，首先调用

SystemInit()初始化系统时钟，然后是 LCD、GPIO、μC/OS 时钟节拍等初始化。

(8) SystemInit()函数

SystemInit()是系统时钟频率初始化函数，应用程序首先得调用这个函数对系统频率进行设置。SystemInit()作为一个库函数，主要进行了一些寄存器必要的初始化后，就调用 SetSysClock()函数。而 SetSysClock()函数则根据 system_stm32f10x.c 中定义的 #define SYSCLK_FREQ_72MHz 72000000 宏，并且调用 SetSysClockTo72()函数，最终将 CPU 主频配置成 72MHz。

(9) OSTaskCreateExt()函数

OSTaskCreate()可以创建一个新任务，但推介使用 OSTaskCreateExt()，与 OSTaskCreate()不同的是，OSTaskCreateExt()允许用户设置更多的细节内容。任务的建立可以在多任务环境启动之前，也可以在正在运行的任务中建立，但中断处理程序中不能建立新任务。一个任务必须为无限循环结构，且不能有返回点，函数原型如下：

```
INT8U OSTaskCreateExt(void ( * task)(void * pd), void * pdata, OS_STK * ptos,
                      INT8U prio, INT16U id, OS_STK * pbos,
                      INT32U stk_size, void * pext, INT16U opt);
```

下面是各参数的意义。

task：是指向任务代码的指针。

pdata：指针指向一个数据结构，该结构用来在建立任务时向任务传递参数。

ptos：为指向任务堆栈栈顶的指针。如果初始化常量 OS_STK_GROWTH 设为 1，堆栈被设为向低端增长(从内存高地址向低地址增长)。此时 ptos 应该指向任务堆栈空间的最高地址。反之，如果 OS_STK_GROWTH 设为 0，堆栈将从低地址向高地址增长。

prio：为任务的优先级。每个任务必须有一个唯一的优先级作为标识。数字越小，优先级越高。

id：是任务的标识，目前这个参数没有实际的用途，但保留在 OSTaskCreateExt()中供今后扩展，应用程序中可设置 id 与优先级相同。

pbos：为指向堆栈底端的指针。如果初始化常量 OS_STK_GROWTH 设为 1，堆栈被设为从内存高地址向低地址增长。此时 pbos 应该指向任务堆栈空间的最低地址。反之，如果 OS_STK_GROWTH 设为 0，堆栈将从低地址向高地址增长。pbos 应该指向堆栈空间的最高地址。参数 pbos 用于堆栈检测函数 OSTaskStkChk()。

stk_size：指定任务堆栈的大小。其单位由 OS_STK 定义：当 OS_STK 的类型定义为 INT8U、INT16U、INT32U 时，stk_size 的单位为分别为字节(8 位)、字(16 位)和双字(32 位)。

pext：是一个用户定义数据结构的指针，可作为 TCB 的扩展。例如，当任务切换时，用户定义的数据结构中可存放浮点寄存器的数值、任务运行时间、任务切入次数等信息。

opt：存放与任务相关的操作信息。opt 的低 8 位由 μC/OS 保留，用户不能使用。用户可以使用 opt 的高 8 位。每一种操作由 opt 中的一位或几位指定，当相应的位被置位时，表示选择某种操作。当前的 μC/OS 版本支持下列操作。

OS_TASK_OPT_STK_CHK：决定是否进行任务堆栈检查。

OS_TASK_OPT_STK_CLR：决定是否清空堆栈。

OS_TASK_OPT_SAVE_FP：决定是否保存浮点寄存器的数值。此项操作仅当处理器有浮点硬件时有效。保存操作由硬件相关的代码完成。

返回值的含义如下。

OS_NO_ERR：函数调用成功。

OS_PRIO_EXIST：具有该优先级的任务已经存在。

OS_PRIO_INVALID：参数指定的优先级大于 OS_LOWEST_PRIO。

OS_NO_MORE_TCB：系统中没有 OS_TCB 可以分配给任务了。

(10) GUI_Init()函数

程序 GUI_Init()初始化 LCD 和 μC/GUI 的内部数据结构，在其他 μC/GUI 函数运行之前必须被调用。如果忽略了这个调用，整个图形系统将不会得到初始化，从而无法准备下一步的动作。

(11) WM_SetCallback()函数

为窗口设置一个回调处理函数，函数原型如下：

```
WM_CALLBACK * WM_SetCallback (WM_HWIN hWin, WM_CALLBACK * cb);
```

函数返回原先回调函数的指针，用户可以保存上一个窗口状态，在回调函数中一般处理各种消息事件(如绘制窗口、触摸屏事件、按键消息等)。

参数意义如下。

hWin：窗口的句柄。

cb：回调函数的指针。

(12) GUI_DrawBitmap()函数

在当前视窗的指定位置绘一幅位图，函数原型如下：

```
void GUI_DrawBitmap(const GUI_BITMAP * pBM, int x, int y);
```

参数意义如下。

pBM：需显示位图的指针。

x：位图在屏幕上位置的左上角 x 坐标。

y：位图在屏幕上位置的左上角 y 坐标。

位图数据必须定义为像素×像素，每个像素等同于一位。最高有效位(MSB)定义第一个像素；图片数据以位流进行说明，以第一个字节的 MSB 作为起始。新的一行总是在一个偶数地址开始，而位图的第 n 行在地址偏移量 $n \times$ BytesPerLine 处开始。位图可以在用户区中任意一点显示，位图转换器 uC-GUI-BitmapConvert. exe 用于产生位图。

(13) GUI_FillCircle()函数

在当前视窗指定坐标以指定尺寸绘制一个填充圆，函数原型如下：

```
void GUI_FillCircle(int x0, int y0, int r);
```

参数意义如下。

x0：在用户视窗中圆心的 x 轴坐标(以像素为单位)。

y0：在用户视窗中圆心的 y 轴坐标(以像素为单位)。

r：圆的半径，最小值为 0，最大值为 180。

当然 μC/GUI 还提供了很多类似的绘图接口供用户使用，在下面的实验中再进行介绍，读者也可以参考 μC/GUI 相关手册了解更多。

(14) GUI_SetColor()函数

设置当前前景色，函数原型为如下：

```
void GUI_SetColor(GUI_COLOR Color);
```

参数意义如下。

Color：前景颜色，24 位 RGB 数值。

【实验步骤】

(1) 进入本实验程序源码，路径如下：\1. uCOS 基础实验\11\11. uCOSII_task，双击 uCOSDemo. uvproj 文件  ，即可打开工程，如图 1.4 所示。

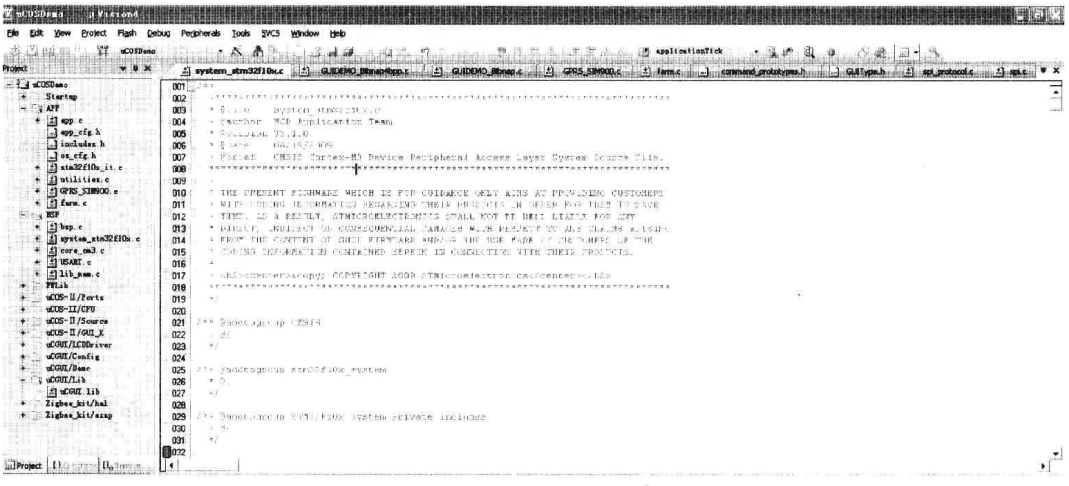

图 1.4 打开工程

(2) 使用 J-LINK 与实验箱相连，打开目标实验箱电源开关，如图 1.5 所示。

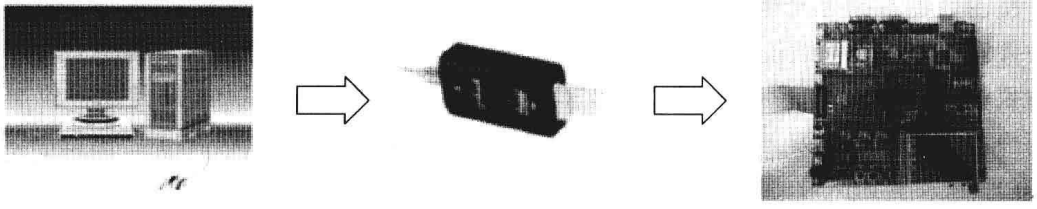

图 1.5 打开目标实验箱电源开关

(3) 通过魔术棒选择 J-LINK 和 Flash 类型，具体方法参见附录。

(4) 第一次选择 Rebuild all target files 命令，以后就可以直接按 F7 键编译当前修改过的文件。

(5) 单击 Download 按钮下载运行，也可以选择按 Ctrl＋F5 键进入单步调试。

注：本实验用到的图片资源在源码目录的 tools 下面。

实验现象如下。

程序源码下载成功后，开发板上屏幕出现如图 1.6 所示界面。

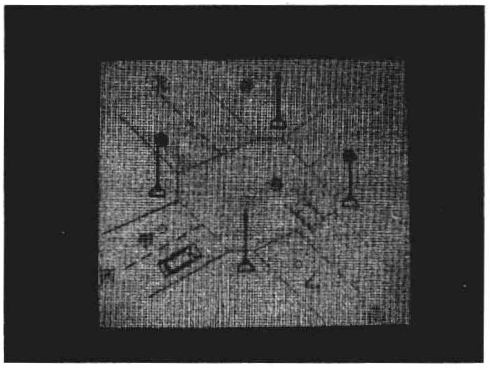

图 1.6　开发板屏幕

两条主干道上车辆通行(绿灯)时间均为 4.8s，如图 1.7 所示。

停止通行(红灯)时间为 4.8s，如图 1.8 所示。

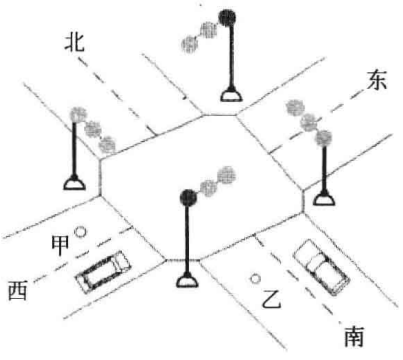

图 1.7　主干道

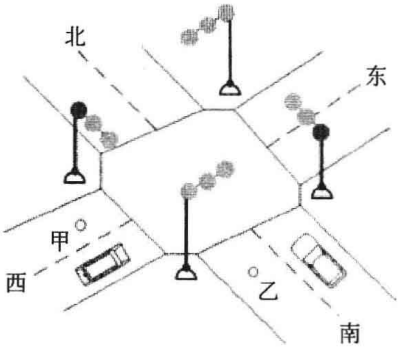

图 1.8　停止通行

红绿灯切换(黄灯)警告时间为 0.8s，如图 1.9 所示。

当出现交通拥堵时，通过按键 tamper 或触摸屏手动实时快速切换通行状态，此时屏幕左上角会出现提示信息，如图 1.10 所示。

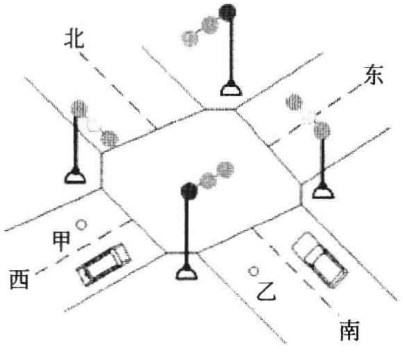

图 1.9　红绿灯切换

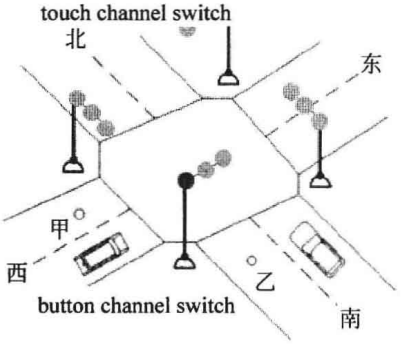

图 1.10　提示信息

假如出现车辆拥堵情况，通过触摸屏幕手动或者通过按键 tamper 控制，行驶的车辆会立即退回原来位置，以达到实时切换通行的效果。

思考：在本模拟交通灯系统中，手动切换通行状态下的按键切换是怎样实现的，以及车辆通行时间在现实环境中时应该怎样正确设置，如果要将本系统应用到现实生活中还要做哪些调整，加入哪些控制单元？

【范例路径】

在北京凯威利亚科技发展有限公司提供的 STM32 实验箱配套源码中提供本实验的参考程序，路径如下：\1. uCOS 基础实验\11\11. uCOSII_task。

1.2 互斥信号量实验

【实验目的】

(1) 进一步熟悉北京凯威利亚科技发展有限公司嵌入式实验箱 STM32。

(2) 掌握 μC/OS 防止优先级反转的方法——互斥信号量的使用。

(3) 学习 μC/GUI 文本显示及字体设置方法。

(4) 掌握通过 GPIO 的置位/复位寄存器来控制相应端口的电平状态。

(5) 掌握端口驱动程序相关固件库的使用。

【实验设备】

(1) 装有 MDK IDE 环境和 J-LINK 驱动的 PC 一台。

(2) 北京凯威利亚科技发展有限公司嵌入式实验箱 STM32 一台。

(3) J-LINK 调试器一块。

(4) 本实验用到实验箱的模块有：LCD 模块、LED 模块。

【实验要求】

(1) 实现功能：建立 3 个任务分别控制 3 个灯泡(这里由 LCD 模拟)的亮灭，但每个任务在调度前就得获取互斥锁，然后才能点亮灯泡并在 LCD 屏上显示当前任务的信息。

(2) 实验现象如下。

① 开发板 LED 点高，表示每个任务都能获得信号量，并点亮自己的灯泡，1 秒后释放信号量给其他任务，如此交替进行。

② 开发板 LED 熄灭，则表示 task1 得到信号量后并未释放，即其他任务无法获得该信号量。

③ 通过 KEY1 按键，可以选择 task1 是否永不释放得到的信号量。

④ 由于 task1 任务优先级最高，所以只要有因获取信号量被加入等待队列的任务时，一旦信号量被释放 task1 会首先得到。

【实验原理】

1. 硬件原理

本实验采用 GPIOD14 连接到 LED，通过设置 GPIOD 端口的高低电平决定 LED 的亮灭。硬件原理图如图 1.11 所示。每个通用输入/输出端口有 2 个 32 位配置寄存器 GPIOx_CRL，GPIOx_CRH，2 个 32 位数据寄存器 GPIOx_IDR，GPIOx_ODR，一个 32 位置位/复位寄存器 GPIOx_BSRR，一个 16 位复位寄存器 GPIOx_BRR 和一个 32 位锁定寄存器 GPIOx_LCKR。

通用输入/输出的每个端口位可以由软件单独的配置成以下几种模式。

- 输入浮动；
- 输入上拉；
- 输入下拉；
- 模拟输入；
- 输出开漏；
- 输出推拉模式；
- 备用功能推拉；
- 备用功能开漏。

每个 I/O 端口位可以自由编程，尽管 I/O 端口寄存器必须以 32 位字的方式访问（不允许以半字或者字节的方式访问）。GPIOx_BSRR 和 GPIOx_BRR 寄存器的目的就是用来允许对 GPIO 寄存器进行原子的读/写操作。在这种方式下，当 IRQ（中断请求）发生在读写操作之间时就不会带来风险。在 STM32F107R 中新增加了 G、F 这两个口的 GPIO，即 GPIO 由原来 STM32F103 的 80 个增加到 112 个。

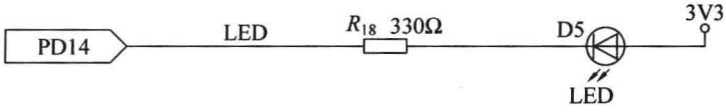

图 1.11　LED 硬件连接电路图

2. 代码分析

驱动代码主要实现了对 GPIOD 口的初始化及其读写操作，应用程序通过相关操作接口实现对 LED 的置位控制。

(1) 头文件

```
#include <includes.h>
#include "GUI.h"
#include "GUIDEMO.h"
#include "Widget.h"
```