# 基于 μC/OS-II 的 LED 控制在 STM32 上的实现①

李 祁[1], 范源远[2], 韩秋枫[1]

[1](海军航空工程学院 计算机教研室, 烟台 264001)
[2](山东航天电子技术研究所霍尔事业部, 烟台 264000)

**摘 要**: 分析了嵌入式实时操作系统 μC/OS-II 的内核结构,并对基于 CoreTex-M3 内核的 STM32 开发板体系结构以及微处理器进行研究,在已经移植了 μC/OS-II2.86 源码的 STM32 平台上实现 LED 闪烁控制的工程设计. 通过实验验证了该方法的可行性以及 μC/OS-II 在处理实时性较强,多任务系统的必要性和优势. 该应用不仅为设计复杂任务提供基础, 也对软件的二次开发具有实践指导意义.

**关键词**: μC/OS-II; STM32; 嵌入式

## Implementation of the LED Control on STM32 Based on μC/OS-II

LI Qi[1], FAN Yuan-Yuan[2], HAN Qiu-Feng[1]

[1](Naval Aeronautical and Astronautical University, Department of Computer, Yantai 264001, China)
[2](Shandong Institute of Aerospace Electronic Technology,Hall Division, Yantai 264000, China)

**Abstract**:The paper analyses the kernel architecture of embedded real-time os μC/OS-II and researches the architecture of STM32 based on CoreTex-M3 kernel.The engineer of LED flashing interval is designed to complete on the STM32 platform which ported the source of μC/OS-II. The feasibility of this method is verified by experiment. Meanwhile μC/OS-II in dealing with strong real-time performance, multitasking system necessity and advantage is displayed. This application not only provides the basis for design of complex task, but also of practical and guiding significance to the secondary development of the software.

**Key words**: μC/OS-II; STM32; embedded

## 1 引言

随着嵌入式系统的广泛应用, 基于嵌入式的操作系统和开发平台越来越受到人们关注, μC/OS-II 作为一款性能稳定、源码开放的实时操作系统, 目前已经被移植到 40 多种不同结构的 CPU 上. 嵌入式系统的设计和开发需要良好的软硬件基础, 本文主要是将 μC/OS-II 应用到测控领域中, 并利用基于 CoreTex-M3 内核的 STM32 硬件平台, 达到能使用 μC/OS-II 的目的.

在各类嵌入式系统设备中, LED 是常见的一种显示方式. 常用于状态指示、定时计数、任务延时等. 例如在具有声光提示双功能的倒车防撞系统中, 利用 LED 闪烁频率来指示障碍物与汽车距离远近[1]; 在智能电网故障分析时, 采用翻牌指示和高亮度 LED 闪烁组合指示故障[2]; 在软件定时器设计中, 被赋予不同优先级和定时值的定时器控制相应的 LED 来显示定时器运行状态[3]. 本文通过实时操作系统 μC/OS-II 控制 STM32 上 LED 闪烁时间间隔, 以检测系统任务管理状态, 本应用实例化可以扩展到智能交通、产品展示等多方领域, 对复杂任务的设计具有实际意义.

## 2 嵌入式实时操作系统 μC/OS-II

嵌入式实时操作系统 μC/OS-II 是由美国工程师 Jean J.Labrosse 所创, 2000 年 μC/OS-II 通过了美国航天管理局(FAA)的安全认证, 可以用于飞机、航天器与人性命攸关的控制系统中. 也就是说, μC/OS-II 具有足够的安全性和稳定性, 用户可以放心使用 μC/OS-II

① 收稿时间:2013-08-30;收到修改稿时间:2013-10-14

研发自己的产品.

μC/OS-II 的源码绝大部分是用移植性很强的 ANSI C 写的. 与微处理硬件相关的部分是用汇编语言写的. 可以在绝大多数 8 位、16 位、32 位以及 64 位处理器、微控制器及数字信号处理器(DSP)上运行; μC/OS-II 具有可裁剪性, 可以通过开关条件编译选项, 定义它的哪些功能模块用于用户程序, 方便控制代码运行所占用的空间及内存; μC/OS-II 可以管理多个任务(实际使用 56 个任务), 并且是完全可剥夺型的实时内核, 它总是运行处于就绪状态下的优先级最高的任务. 图 1 示意 μC/OS-II 的文件结构以及与硬件的关系[4].

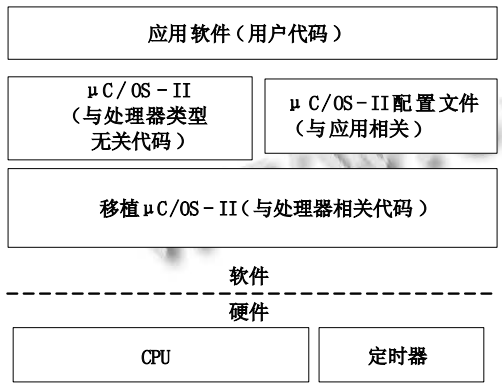


图 1　μC/OS-II 软硬件体系结构

μC/OS-II 的核心工作原理是: 近似地让最高优先级的就绪任务处于运行状态. 首先, 初始化 MCU 再进行操作系统初始化, 主要完成任务控制块 TCB 初始化, TCB 优先级表、TCB 链表初始化, 事件控制块 ECB 链表初始化, 空任务创建等; 然后开始创建新任务, 最后调用 OSSTART()函数启动多任务调度, 随后启动时钟节拍源开始计时, 给系统提供周期性的时钟中断信号, 实现延时和超时确认[5].

## 3　奋斗STM32开发板硬件结构

STM32 是意法半导体(ST)推出的基于 CoreTex-M3 内核的 32 位 ARM,本文选用的 STM32 V5 开发板主系统由四个驱动单元(Cortex-M3 内核 ICode 总线(I-bus)、DCode 总线(D-bus)、系统总线(S-bus)和通用 DMA)和三个被动单元(内部 SRAM、内部闪存存储器、AHB 到 APB 的桥(连接所有的 APB 设备))构成. 这些都是通过一个多级的 AHB 总线构架相互连接的. 如图 1 所示.

该开发板 MCU 采用 STM32F103VET6, 这个芯片属于 STM32F103 系列的高容量芯片, 具有 512K 字节的内部 FLASH, 64K 字节的 SRAM, 外设资源有全速 USB Device, SDIO, SPI, I2C, I2S, FSMC, 定时器, USART, ADC, DAC, CAN 等接口.

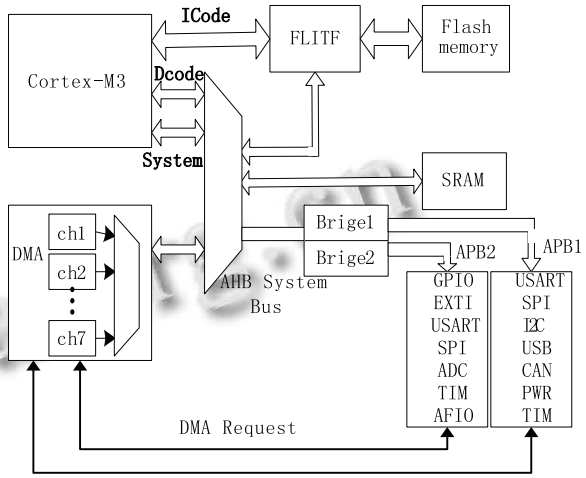


图 2　主系统结构图

其他主要硬件资源: JTAG 调试接口、TTL 异步通信接口、CAN 总线接口、RS-485 总线接口、USB2.0 SLAVE 全速模式接口(可作为供电接口, 也可作为 USB 通信接口)、LCD 接口(和各个尺寸液晶显示模块连接)、SPI 总线控制的 ENC28J60 网络接口(10M)、I2C 总线控制的 FM 收音模块(TEA5767)、复位按键、DC5V 输入接口等.

其中 GPIO(通用 IO)接口, 包含了可以作为普通 IO 或者可具有 PWM, SPI, SDIO 等功能以及其他类型的 IO 接口, 同时包含了电源 5V, 3.3V 接口, 方便用户在这个接口上做二次开发.

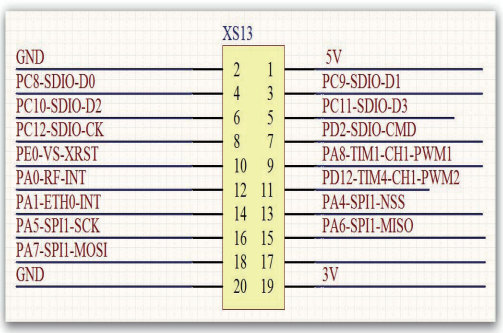


图 3　GPIO 接口原理图

开发板 LED 配置 1 个电源 LED(橙色), 3 个状态 LED(蓝色). 状态 LED 指示灯, 主要用于用户编程状态

指示. 高电平灯亮. 低电平灯灭. 原理图如图 3 所示.

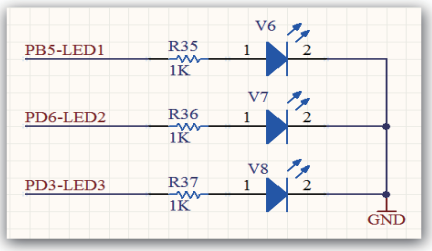


图 4 状态 LED 原理图

## 4 基于 μC/OS-II的LED控制在STM32上的实现

STM32 V5 开发板采用 STM32F103VET6 作为 MCU, 内置 512K FLASH 64KSRAM, 非常适合短小精悍的 μC/OS-II 作为操作系统; 而 μC/OS-II 是实时操作系统, 也极适合 STM32 所面对的嵌入式微控领域. 开发板选用了已经被移植到 STM32 平台上的 μC/OS-II2.86 源码. 经广泛测试此源代码在 STM32 上可靠运行, 以下是在 STM32 开发板上实现对 LED 闪烁控制, 通常是任务驱动 LED 闪烁, 而本文是在 μC/OS-II 操作系统中建立任务, 并通过串口中断响应处理任务, 实现对 LED 的控制.

1) 功能要求

开发板上电后, LED1-3 会按照默认的 500ms 间隔, 明暗闪烁, 此时可以通过串口助手 SSCOM3.2 发出指令, 设置 LED1,LED2,LED3 的闪烁间隔时间. 间隔范围是 1-65535ms. 可以设置任意一个 LED 的闪烁间隔时间.

根据功能要求, 对这个例程进行了工程策划, 选用 MDK3.80a 作为工程编译环境. JLINK V8 作为下载仿真器, 三个 LED 的闪烁分别采用建立 3 个任务. 功能里的串口接收指令, 表明例程会用到串口 1 中断, 还需建立一个串口接收任务. 再采用一个任务作为初始化时的主任务, 用于建立以上的 4 个用户任务. 根据实时响应的重要程度, 将各个任务的优先级经行了设置.

为了兼顾实时效率及CPU 的负荷, 将 μC/OS-II 的时钟节拍设置为 10ms, μC/OS-II 需要提供周期性信号源, 用于实现时间延时和确认超时, 时钟节拍的含义就是任务和任务之间最短切换时间. 这个节拍也不能设置的过短, 否则 CPU 负荷过大, 任务执行兼顾不周. 某些高优先级任务总是在执行, 而有些低优先级任务得不到执行. 但节拍也不能设置的过长, 这会造成任务执行的实时性变差. 一般设置在 10-100ms 之间.

表 1 任务及优先级表列

| 任务名 | 优先级 | 说明 |
|---|---|---|
| APP_TASK_START_PRIO | 2 | 主任务 |
| Task_Com1_PRIO | 4 | COM1 通信任务 |
| Task_Led1_PRIO | 7 | LED1 闪烁任务 |
| Task_Led2_PRIO | 8 | LED2 闪烁任务 |
| Task_Led3_PRIO | 9 | LED3 闪烁任务 |

2) 程序流程

启动文件初始化, 首先在 C 代码里, 执行 main(), 在启动 μC/OS-II 内核前先禁止 CPU 的中断 CPU_IntDis(), 防止启动过程中系统崩溃, 然后对 μC/OS-II 内核进行初始化 OSInit(), 以上两个函数都不用特意修改, 在 STM32 平台上已经被移植好了, 对开发板上的一些用到的外设进行初始化设置 BSP_Init(), 这个函数包含了对系统时钟的设置 RCC_Configuration(), 将系统时钟设置为 72MHz, 对 LED 闪烁控制端口进行设置 GPIO_Configuration(), 对中断源进行配置 NVIC_Configuration(), 对串口 1 进行配置 USART_Config(USART1,115200), 开发板上的外设初始化完毕后, 默认 3 个 LED 的闪烁间隔分别是 500ms、500ms、500ms, 通过串口 1 格式化输出例程信息.

```
USART_OUT(USART1," STM32 开发板 LED 闪烁 \r\n");
USART_OUT(USART1,"\r\n");
USART_OUT(USART1,"LED1 闪烁间隔 : 1ms--65535ms 指令 L1 1F--L1 65535F \r\n");
USART_OUT(USART1,"LED2 闪烁间隔 : 1ms--65535ms 指令 L2 1F--L1 65535F \r\n");
USART_OUT(USART1,"LED3 闪烁间隔 : 1ms--65535ms 指令 L3 1F--L1 65535F \r\n");
USART_OUT(USART1,"\r\n");
```

建立主任务, 该任务是为了在内核启动后, 建立另外 4 个用户任务. 主任务的任务名为 App_TaskStart, 主任务有自己的堆栈, 堆栈尺寸为 APP_TASK_START_STK_SIZE*4(字节), 然后执行 μC/OS-II 内部函数 OSTimeSet(0), 将节拍计数器清 0, 节拍计数器范围是 0-4294967295, 对于节拍频率 100hz 时, 每隔 497 天就重新计数, 调用内部函数

OSStart(), 启动 μC/OS-II 内核, 此时 μC/OS-II 内核开始运行. 对任务表进行监视, 主任务因为已经处于就绪状态, 于是开始执行主任务 App_TaskStart(), μC/OS-II 的任务结构规定必须为无返回的结构, 也就是无限循环模式.

```
static void App_TaskStart(void* p_arg)
{
  (void) p_arg;
//这个语句是防止没有引用的参数会编译错误或警告
  OS_CPU_SysTickInit();
//时钟节拍初始化为 100Hz (10ms)
  #if (OS_TASK_STAT_EN > 0)
//使能 μC/OS-II 的统计任务
   OSStatInit();
   //----统计任务初始化函数
  #endif
       App_TaskCreate();
    //建立其他 4 个用户任务
       while (1)
       {
          OSTimeDlyHMSM(0, 0,1, 0);
    //1 秒一次循环,也可采用挂起
       }
    }
static void App_TaskCreate(void)
{
Com1_MBOX=OSMboxCreate((void *) 0);
//建立串口 1 中断的消息邮箱
OSTaskCreateExt(Task_Com1,void *)0;
//任务开始执行时, 传递给任务的参数指针
//分配给任务的堆栈栈顶指针从顶向下递减
(OS_STK*)&Task_Com1Stk[Task_Com1_STK_SIZE-1]
;
Task_Com1_PRIO; //分配给任务的优先级
(OS_STK *)&Task_Com1Stk[0];//指向任务堆栈栈底的指针, 用于堆栈的检验
Task_Com1_STK_SIZE; //指定堆栈的容量, 用于堆栈的检验
(void *)0;//指向用户附加的数据域的指针, 用来扩展任务的任务控制块
//选项, 指定是否允许堆栈检验, 是否将堆栈清 0,任务是否要进行浮点运算等.
OS_TASK_OPT_STK_CHK|OS_TASK_OPT_STK_CL
R);
... ...//LED1\LED2\LED3\LED4 闪烁任务
}
```

4 个用户任务各自有自己的堆栈空间, 为了接收串口的信息事件, 建立了一个信息邮箱, 来传递串口进来的信息. 闪烁任务是通过精确延时来实现任务就绪及挂起的. 函数原型可以参考 μC/OS-II 手册, 最大精度是 10ms, 延时 105ms, 其实是延时了 110ms. 以下为 LED1 闪烁任务:

```
static void Task_Led1(void* p_arg)
{
    (void) p_arg;
    while (1)
    {
    LED_LED1_ON();
    OSTimeDlyHMSM(0, 0, 0, milsec1);
    LED_LED1_OFF();
    OSTimeDlyHMSM(0, 0, 0, milsec1);
    }
}
```

COM1 通信任务中, 通过 msg=(unsigned char *)OSMboxPend(Com1_MBOX,0,&err); 等待串口接收指令成功的邮箱信息, 一旦接收到改变 LED 闪烁延时的指令, 串口中断例程就通过 OSMboxPost(Com1_MBOX,(void*)&msg) 将接收到的信息用消息邮箱传递给串口接收指令任务, 串口接收任务接收到指令后, 对指令进行解析, 以获得各 LED 的闪烁间隔时间. LED 闪烁任务在延时等待结束后, 会用新的延时值重新运行.

3) 实验运行显示

运行 MDK3.50 开发环境, 打开 STM32 奋斗板 LED 闪烁工程文件. 编译后生成 HEX 类型文件,通过 ISP 或者 JTAG 将 HEX 文件下载到开发板中. 在 PC 端运行 SCOM 串口助手软件, 并初始化设置如下:

给开发板重新上电或复位, SCOM 将接收到数据, 按照串口提示可以设置任意一个 LED 灯的闪烁间隔, 比如设置 LED1 的闪烁时间间隔为 50ms, 可以在字符串输入框中输入 L1 50F, 点击发送, 开发板上的 LED1 灯将按照设置的 50ms 间隔时间闪烁, 同理其余

的 LED 也执行同样的操作, 同时开发板正确收到设置指令后, 会将该指令回送到串口助手.


图 5 SCOM 串口助手初始化设置


图 6 SCOM 串口助手设置闪烁间隔时间等待状态


图 7 SCOM 串口助手显示设置指令返回状态

通过串口助手显示可以判断, 串口能够正确接收任务并显示定义的闪烁间隔时间, 此时开发板上的 3 个 LED 按照定义间隔闪烁时间正常运行, 证明此设计可行, 通过串口中断响应处理任务可实现对 LED 的控制. 另一方面, 该实验也体现了 μC/OS-II 在处理多任务时, 可剥夺型内核的优势, 表现出良好的实时性能.

## 5 结语

随着微控领域应用日趋复杂化, 对 MCU 资源以及嵌入式实时操作系统要求越来越高. 本文通过研究在 μC /OS- II 嵌入式操作系统在奋斗 STM32 开发板上对 LED 闪烁控制方法, 验证了该方法的可行性以及 μC/OS-II 在处理实时性较强,多任务系统必要性和优势, 也为更加复杂的微控应用提供了实践基础和开发思路.

## 参考文献

1 周超.具有声光提示双功能的倒车防撞系统设计.传感器与微系统, 2011,5 (30).
2 张宏波,袁钦成.故障指示器在智能电网中的应用,能源技术经济, 2011,1(23).
3 王中良,梅静静,胡敏. μC/OS-II 中软件定时器的研究与改进. 单片机与嵌入式系统应用.2011,4.
4 Labrosse JJ,邵贝贝等译.嵌入式实时操作系统 μC/OS-II.北京:北京航空航天大学出版社,2012:72–73.
5 魏春杰.嵌入式实时操作系统 μC/OS-II 应用技术研究[学位论文].大连:大连海事大学,2004.

(上接第 167 页)

10 张建华,冯冲,刘毅,等.用于 MIMO OFDM 系统的定时同步算法.北京邮电大学学报,2009,32(1):118–121.
11 王一蓉,王文博.一种改进的分布式 MIMO-OFDM 系统同步方案.电子测量技术,2007,30(11):173–176.
12 Feng C, Zhang J, Zhang Y, Xia M. A novel timing synchronization method for MIMO OFDM systems. Vehicular Technology Conference. 2008. VTC. Spring, 2008. IEEE. May 2008.913–917.
13 柏刚, MIMO-OFDM 系统同步技术研究[硕士学位论文].西安:西安电子科技大,2011.1.