目录

基于 μC/OS-II 的流水灯按键控制系统设计

尹达恒
（江南大学物联网工程学院，江苏　无锡）

1　设计要求

- 利用 P0 口的一个管脚作为一个按键信号输入，其作用是启动流水灯的开始和停止（第一次按启动，第二次停止，第三次启动，以此类推）；
- 利用 P0 口的一个管脚作为一个按键信号输入，其作用是设置灯亮的时间，分三档 1 秒，2 秒，3 秒（第一次按 1，第二次 2 秒，第三次 3 秒，第四次 1 秒，以此类推）；
- 基于 μC/OS-II 操作系统，完成以上程序设计。

2　设计原理

2.1　LED 流水灯运行原理

LED 流水灯实际上就是八个 LED。如果要点亮一个阴极接地的 LED，需要在其阳极接入高电平；相反，如果要让这个 LED 熄灭，就要阳极接入低电平。因此，要实现流水灯功能，只要在八个 LED 的阳极依次接入高电平，八个 LED 灯便会一亮一暗的做流水灯了。另外，由于人眼的视觉暂留效应以及单片机执行每条指令的时间很短，在控制二极管亮灭的时候应该延时一段时间。

2.2　按键控制原理

按键的基本原理是读取单片机 IO 口的输入状态。对于一个接地按键来说，当 IO 口输入低于某一阈值时表明按键按下，此时在 STM32 的程序中可以读取到对应的 IO 口值为 0。根据不同时刻 IO 口的 01 值即可获得按键状态，并以此控制单片机的行为。

2.3　μC/OS-II 操作系统运行原理

μC/OS-II 由 Micrium 公司提供，是一个可移植、可固化的、可裁剪的、占先式多任务实时内核，它适用于多种微处理器，微控制器和数字处理芯片 μC/OS-II

可以大致分成核心、任务处理、时间处理、任务同步与通信和 CPU 接口 5 个部分：

- 核心部分：操作系统的处理核心，包括操作系统初始化、操作系统运行、中断进出的前导、时钟节拍、任务调度、事件处理等功能代码；
- 任务处理部分：任务处理部分中的内容都是与任务的操作密切相关的。包括任务的建立、删除、挂起、恢复等；
- 时钟部分：μC/OS-II 中的最小时钟单位是 timetick（时钟节拍）。任务延时等操作是在时钟部分完成；
- 任务同步和通信部分：为事件处理部分，包括信号量、邮箱、邮箱队列、事件标志等部分；主要用于任务间的互相联系和对临界资源的访问；
- CPU 接口部分：主要包括中断级任务切换的底层实现、任务级任务切换的底层实现、时钟节拍的产生和处理、中断的相关处理部分等内容。由于 μC/OS-II 是一个通用性的操作系统，所以对于关键问题上的实现，还是需要根据具体 CPU 的具体内容和要求作相应的移植。这部分内容由于牵涉到 SP 等系统指针，所以用汇编语言编写。主要的移植工作都在此部分进行。

3 设计方案

3.1 硬件设计

根据设计要求，流水灯按键控制系统设计方案的硬件主要包含以下部分：

- STM32 单片机最小系统；
- 两个按键；
- 八个 LED。

并使用 STM32 GPIOB 的高八位管脚控制 LED，低八位管脚获取按键状态。

3.2 软件设计

根据设计要求，流水灯按键控制系统的软件包含两个标志位和两个进程：

- 启停标志位：无符号整数，该标志位为奇数时流水灯保持运行，为偶数时流水灯暂停运行；

- 速度标志位：无符号整数，流水灯运行速度为 (速度标志位 mod 3) 秒/位，当速度标志位发生变化时改变流水灯的运行速度；
- 按键进程：每隔一定时间扫描按键状态，并按照按键状态改变标志位值；
- 流水进程：每隔一定时间检查标志位值，并按照标志位值改变 LED 的亮灭状态。

4 系统实现

4.1 硬件实现

开发板使用 PERCHIN HC6800-EM3，其核心板芯片为 STM32F103C8T6，最小系统电路图如附录 A、接地按键电路如图 1、LED 电路如图 2。使用杜邦线将最小系统的 PB0~PB7 和 PB8~PB15 分别与按键电路的 JP5 和 J12 相连，从而使得 GPIOB 的高八位管脚连接 LED，低八位管脚连接按键。

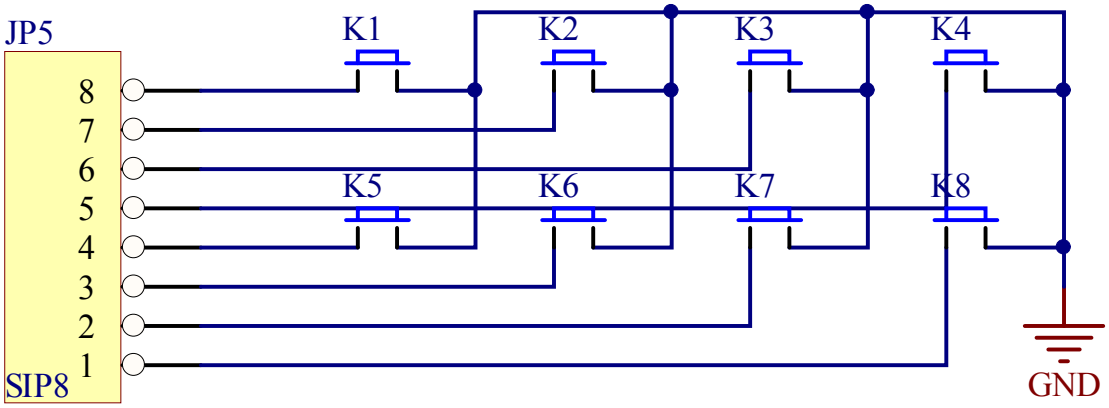


图 1. 按键电路原理图（使用其中的 K7、K8 按键）

4.2 软件实现

4.2.1 μC/OS-II 移植

μC/OS-II 的详细移植步骤如下：

1、 构造一个裸机 Demo，实现基本的流水灯；

2、 至 Micrium 官网下载并解压 STM32F107 官方移植的 μC/OS，其中 Keil MDK4 的版本是 μC/OS 2.92.07；

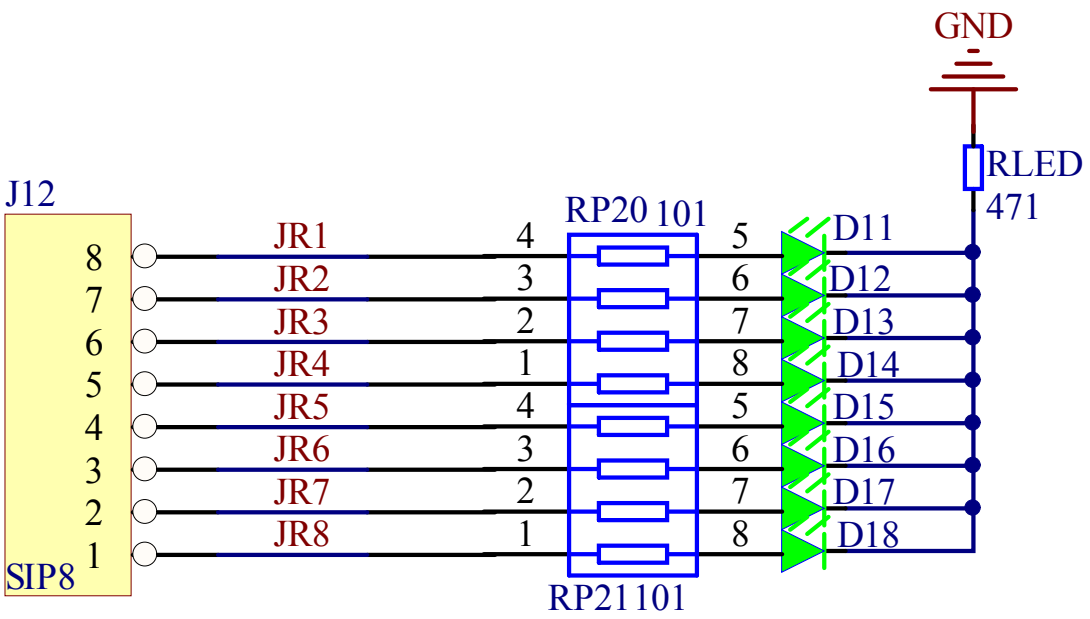


图 2. LED 电路原理图

3、 将下列文件夹中的文件复制到裸机 Demo 下：

- Micrium/Software/uCOS-II/Ports
- Micrium/Software/uCOS-II/Source
- Micrium/Software/EvalBoards/Micrium/uC-Eval-STM32F107/uCOS-II 中的 os_cfg.h、app_cfg.h、app_hooks.c、includes.h。

4、 在 Keil 工程中添加上一步中复制的文件；

5、 修改 includes.h：删去 bsp(Board Support Pack, 开发板支持) 和 lib 相关的 include 引用；

6、 修改 stm32f10x_it.c：

- 在 SysTick 中断处理函数 SysTick_Handler() 中调用操作系统的 SysTick 函数 OS_CPU_SysTickHandler()；
- 删去原有的 PendSV 异常处理函数 PendSV_Handler()，防止下一步中修改 uCOS 的 os_cpu_a.asm 后出现重定义错误；

7、 修改 os_cpu_a.asm：

- 将操作系统原有的 PendSV 处理函数 OS_CPU_PendSVHandler 重命名为 PendSVHandler；
- 将文件开头的 EXPORT OS_CPU_PendSVHandler 改为 EXPORT PendSVHandler。

4.2.2 按键进程函数

- 编写函数 KEY_gets()，使用一个全局变量保存按键状态，在 KEY_gets() 中调用 GPIO_ReadInputData 读取当前按键状态（8 位二进制数），若当前按键状态与保存的按键状态不同且为按下状态，即表明按键经历了一个按下的过程，此时 KEY_gets() 返回值对应位置 1，否则返回 0；
- 编写按键进程函数 LED_change()，该函数的 while 循环每隔 10 毫秒运行一次，在循环中调用 KEY_gets() 函数获取按键状态并判定按下的按键，如果按下了按键 K8，则改变启停标志位值；如果按下了按键 K7，则改变速度标志位值。

按键进程的运行流程可以用如图 3所示的流程图表示。核心代码如附录 C。

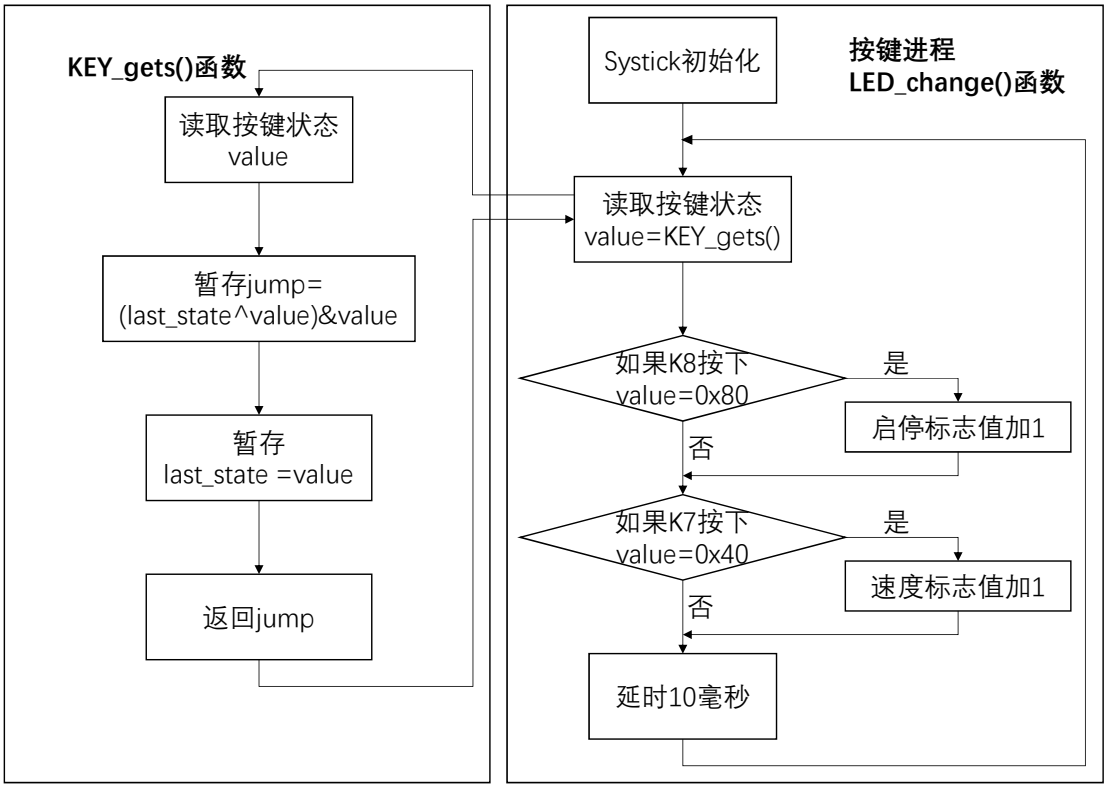


图 3. 按键进程流程图

4.2.3 流水进程函数

流水进程由一个 while 循环构成，每次循环中都会进行如下操作：

- 读取启停标志位的值，若该值为奇数，则调用 LED_Sets 函数亮起流水灯中的下一个灯，否则不进行任何操作；

- 读取速度标志位的值,以该值的模 3 余数加一为秒数,调用 OSTimeDlyHMSM 函数设置下一次循环的运行时间。

流水进程的运行流程可以用如图 4所示的流程图表示。核心代码如附录 D。

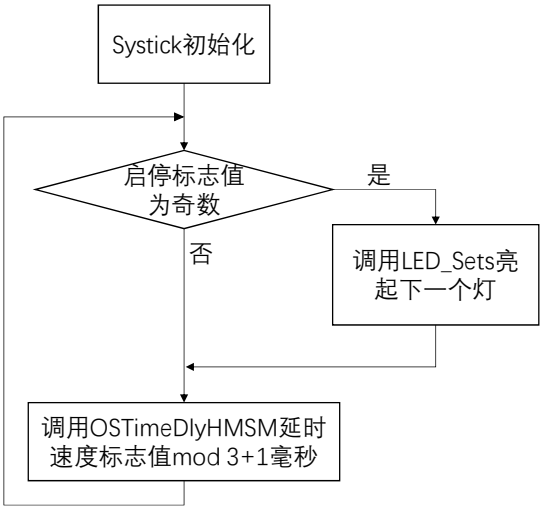


图 4. 流水灯进程流程图

4.2.4 主函数

在主函数中需要进行系统时钟、管脚和操作系统的初始化操作以及操作系统的任务创建和启动，相关核心代码如附录 E所示，各部分实现流程如下：

- 系统时钟初始化：使用系统固件库函数 RCC_GetClocksFreq 获取到系统时钟源 HCLK 的频率；调用 SysTick 设置函数 SysTick_Config 设置 systick 重装定时器的值 =HCLK 频率/操作系统每秒钟的 Tick 数 OS_TICKS_PER_SEC，使操作系统获取到准确的系统时钟；
- 管脚的初始化：根据 3.1节中的设计方案，需要将 GPIOB 管脚高八位设置为推挽输出,低八位设置为上拉输入。使用 GPIO_InitStructure 结构和 GPIO_Init 函数进行设置；
- 操作系统系统的初始化：μC/OS-II 操作系统的初始化通过调用函数 OSInit() 完成；
- 操作系统任务的创建：根据 3.2节中的设计方案，需要调用 OSTaskCreate 函数将 4.2.2节中编写的按键进程函数和 4.2.3流水进程函数分别创建为两个进程，且使流水进程的优先级为最高，从而获得较为准确的定时时间；

- 操作系统的启动：μC/OS-II 操作系统的启动通过调用函数 OSStart() 函数完成。

5 设计结果与总结

5.1 设计结果

编译工程并下载至开发板芯片中查看实现效果，其结果完全符合设计要求，当按下按键 K8 时，流水灯以每 2 秒一次的速度启动，此时按键 K8 控制流水灯的暂停和继续；流水灯的速度受到按键 K7 的控制，每次按下 K7 按键时，流水灯的速度都会在 1 秒一次 $\rightarrow$2 秒一次 $\rightarrow$3 秒一次三个状态之间循环。

5.2 遇到的问题

汇编函数 OS_CPU_PendSVHandler 的调用问题

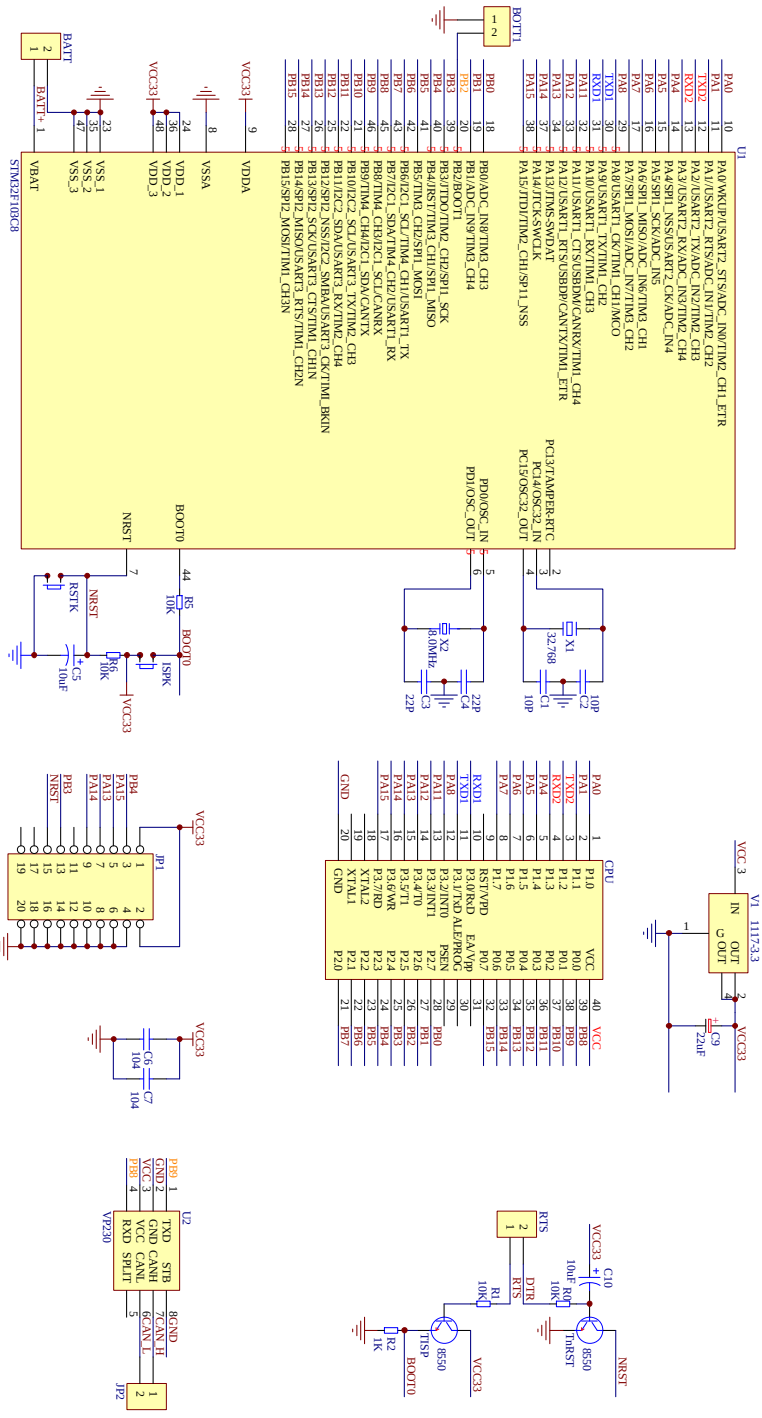
- 问题描述:在 4.2.1节介绍的移植过程中,os_cpu_a.asm 中的 OS_CPU_PendSVHandler 函数需要在作为可挂起系统中断函数在 PendSV 中断时进行调用，但若将其放入 stm32f10x_it.c 文件的 PendSV_Handler 中系统无法正常工作。
- 问题原因：OS_CPU_PendSVHandler 的作用是在时间片轮转和系统出错时保存现场，因此使用汇编语言编写，而若将其放在 PendSV_Handler 中作为一个函数进行调用，在编译过后，其保存的现场为 PendSV_Handler 中断处理函数的现场，而不是正常程序执行时的现场，因此系统无法正常工作。
- 问题解决：直接在 os_cpu_a.asm 中将 OS_CPU_PendSVHandler 函数名称修改为 PendSV_Handler，使得 PendSV 中断发生时直接调用 os_cpu_a.asm 的处理函数。

5.3 心得体会

- 巩固了单片机和嵌入式系统的课程知识，更加深入地理解了单片机和嵌入式系统的工作原理；
- 实际看到了一个简单操作系统的实现代码，更加深入地理解了操作系统的工作流程和实现方法，尤其是任务调度和保存现场的过程；
- 进一步精进了代码水平，对嵌入式系统的开发更加熟练。

附录

A STM32F103C8T6 最小系统原理图（PRECHIN STM32 核心板）



B 标志位代码

```
uint8_t isflow=0;
uint8_t speed=0;
```

C 按键进程代码

```
uint8_t last_state=0x00;
uint8_t KEY_Gets()
{
  uint8_t value = (uint8_t)(GPIO_ReadInputData(GPIOB)&0x00f0);
  uint8_t jump = (last_state^value)&value;
  last_state = value;
  return jump;
}
void LED_change()
{
  uint8_t value=0x00;
  SystickInit();
  while(1)
  {
    value=KEY_Gets();
    if(value&0x80)
      isflow+=1;
    if(value&0x40)
      speed+=1;
    OSTimeDlyHMSM(0,0,0,10);
  }
}
```

D 流水进程代码

```
void LED_Sets(uint8_t k)
{
  uint16_t setValue;
  setValue = GPIO_ReadOutputData(GPIOB)&0x00ff;
  setValue |= (uint16_t)(1<<(k%8)) << 8;
  GPIO_Write(GPIOB,setValue);
}
void LED_flow()
{
  uint8_t count = 0;
  SystickInit();
```

```
  while(1)
  {
    if(isflow%2==0)
      LED_Sets(count++);
    OSTimeDlyHMSM(0,0,speed%3+1,0);
  }
}
```

E 主函数代码

系统时钟初始化

```
static void SystickInit()
{
  RCC_ClocksTypeDef  rcc_clocks;
  RCC_GetClocksFreq(&rcc_clocks);
  SysTick_Config(rcc_clocks.HCLK_Frequency / OS_TICKS_PER_SEC);
}
```

管脚的初始化

```
#define RCC_LED   RCC_APB2Periph_GPIOB
#define PIN_KEY (GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3|\
  GPIO_Pin_4|GPIO_Pin_5|GPIO_Pin_6|GPIO_Pin_7)
#define PIN_LED (GPIO_Pin_8|GPIO_Pin_9|GPIO_Pin_10|GPIO_Pin_11|\
  GPIO_Pin_12|GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15)
void LED_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;
  GPIO_InitStructure.GPIO_Pin = PIN_LED;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  RCC_APB2PeriphClockCmd(RCC_LED, ENABLE);
  GPIO_Init(GPIOB, &GPIO_InitStructure);
}
void KEY_Init(void)
{
  GPIO_PinRemapConfig(GPIO_Remap_SWJ_Disable, ENABLE);
  GPIO_InitTypeDef GPIO_InitStructure;
  GPIO_InitStructure.GPIO_Pin=PIN_KEY;
  GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IPU;
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);
  GPIO_Init(GPIOB,&GPIO_InitStructure);
```

```
}
void BSP_Init()
{
  LED_Init();
  KEY_Init();
}
```

操作系统系统的初始化、系统任务的创建和启动

```
static OS_STK task_LED_flow[STARTUP_TASK_STK_SIZE];
static OS_STK task_LED_change[STARTUP_TASK_STK_SIZE];
int main(void)
{
  BSP_Init();
  OSInit();
  OSTaskCreate(LED_flow,(void *)0,
    &task_LED_flow[STARTUP_TASK_STK_SIZE-1],
    STARTUP_TASK_PRIO);
  OSTaskCreate(LED_change,(void *)0,
    &task_LED_change[STARTUP_TASK_STK_SIZE-1],
    STARTUP_TASK_PRIO-1);
  OSStart();
}
```