# 嵌入式系统实验报告



|  |  |
| --- | --- |
| 实验名称： | GPIO—按键与 LED |
| 姓 名： | 江姝潼 |
| 学 号： | 2019211653 |
| 学 院(系)： | 计算机学院 |
| 专 业： | 网络工程 |
| 指导教师： | 刘健培、戴志涛 |

2021年 12 月 1 日

# 实验目的

* 通过 FSM4 实验板了解实验的软硬件环境，熟悉 MDK 开发环境的使用
* 学习查阅文档和数据手册，获取需要的信息。
* 学会使用 C 语言控制 IO 寄存器完成设备控制功能。
* 学会 STM32 GPIO 的基本操作方式（初始化、读、写）。
* 理解开关消抖的原理与机制。
* 理解同时操作多个 IO 的方式。
* 掌握基本的驱动软件编写与调试方式

# 实验环境

* FS-STM32F407开发平台
* ST-Link 仿真器
* RealView MDK5.23集成开发软件
* PC机Window7/8/10 (32/64bit)
* 串口调试工具

# 实验要求

使用实验板上的 4 个按键（K3-K6）控制 4 个 LED 灯（D6-D9）的状态变化。

* 无按键操作时，D6-D9 以跑马灯（1 个灯光在 4 个 LED 上依次滚动亮起，
* 收尾相接）的方式顺时针闪烁。滚动周期 1 秒。  K3——短按，暂停键。第一次短按，暂停跑马灯，第二次短按，继续跑马灯。
* K4——双击，前进键。双击 K4，跑马灯顺时针闪烁。
* K5——双击，后退键。双击 K5，跑马灯逆时针闪烁。
* K6——长按，呼吸灯。长按（超过 2 秒），4 个 LED 灯模拟呼吸灯亮灭(即依次逐渐亮-灭-亮），松开后，呼吸灯停止。

# 实验原理

1. **STM32 GPIO 的配置**

LED 灯的亮灭、蜂鸣器的鸣响、按键电平的读入都需要使用 STM32 芯片的I/O 引脚。STM32 芯片上，I/O 引脚可以被软件设置成各种不同的功能，如输入或输出，所以被称为 GPIO (General-purpose I/O)。而 GPIO 引脚又被分为 GPIOA、GPIOB…GPIOG 不同的组，每组端口分为 0~15，共 16 个不同的引脚。

程序控制 GPIO，包括初始化和主逻辑 2 部分。STM32 GPIO 的初始化需要开启 GPIO 时钟、设置好 GPIO 的配置寄存器两步。如果是复用 GPIO 管脚的外设，则一般需要 4 步：

1) 开启复用管脚的 GPIO 时钟

2) 设置好复用管脚的 GPIO 配置寄存器

3) 开启外设时钟

4) 设置好外设的配置寄存器

主逻辑部分主要是完成程序功能，如本实验的摩斯码编码输出。一般主逻辑可分为 2 部分：

1) 与底层外设无关的逻辑部分。如摩斯码的编解码。

2) 需要通过外设与外部交互的驱动部分。如摩斯码的滴答到 LED/蜂鸣器的映射。

**2. 按键扫描与消抖**

由于机械触点的弹性作用，按键开关在闭合时不会马上稳定的接通，而是有一段时间的抖动，在断开时也不会立即断开。抖动时间由按键的机械特性所决定，一般为 5ms~20ms。如果不作处理这个抖动会给系统带来一些不稳定的因素，甚至是错误的结果，所以在做按键检测时都要加一个消抖的过程。

软件消抖可以有多种方案，例如：

（1）CPU 延时重采样。

延时重采样的意思是，当第一次检测到键值变化（如 1->0）时，再延时一段时间（如 20ms），再次采样，确认是否仍是'0'；若是'0'则认为此时键值为'0'，否则，重新执行检测过程。延时可以采用轮询，也可以采用定时器。该方案的缺陷为，如果延时太短，有可能两次采样时都处于抖动时间，因此可能引起误判；如果延时太长，可能检测不出按键变换；如果采用轮询方式来完成延时，会大大降低 CPU 的效率。

（2）持续采样

持续采样的原理是，当检测到按键处于某电平（如'0'）时，在之后的 N 个时钟周期内连续检测此按键的电平，如果一直不变，则读出此按键的电平值（如'0'）。持续采样的优点为样本足够多，减少误判的可能性；对于按键按下('1'->'0')，按键释放（'0'->'1'）都可以检测。

持续采样的缺点则为持续检测的时间太长（大于按键按下和释放的时间差），则可能无法检测按键的变换。从理论上来说，延时（如 20ms）重采样的准确率低于持续采样。

（3）使用外部中断

按键按动时会产生高低电平的变化，可以将按键管脚设置为边缘触发中断。但按键抖动可能触发多次中断，所以处理中断时还需要以上的延迟或者重采样思想来消抖。 该方法的优点是：CPU 效率高，实时性强。缺点是需要用掉一个额外的外部中断资源。编程比上述同步方法要复杂一些。

（4）使用 ADC 转换器

当按键接通时，通过 ADC 转换后对按键的电压值比较判断，这是一种滤波法。该方法的优点是：累加电压值对抖动不敏感，准确性较高。缺点是：需要用掉额外的 ADC 资源，硬件成本太高。

（5）软件数字滤波法

滤波算法有很多种，比如中位值滤波法、算术平均滤波法、加权递推平均滤波法、消抖滤波法等，通过对历史数据与新数据的特性进行判别，得到当前的稳定值。不同滤波算法与不同的特性。

实验中使用了持续采样+延迟重采样的轮询采样方法和递推平均滤波法。

**3. 呼吸灯与 PWM 控制原理**

呼吸灯，就是指灯光设备的亮度随着时间由暗到亮逐渐增强，再由亮到暗逐渐衰减，很有节奏感地一起一伏，就像是在呼吸一样，因而被广泛应用于手机、电脑等电子设备的指示灯中。呼吸的特性是一种类似图 指数曲线 中的指数曲线过程，吸气是指数上升过程，呼气是指数下降过程，成年人吸气呼气整个过程持续约 3 秒。

若把拟合的波形改成呼吸特性曲线，即可得到控制呼吸灯使用的 PWM 波形，要生成拟合的 PWM 波形，通常使用计算法和调制法：

(1) 计算法：根据拟合波形的频率、幅值和半周期脉冲数，准确计算 PWM 波各脉冲宽度和间隔， 据此控制开关器件的通断，就可得到所需 PWM 波形；

(2) 调制法：拟合波形作调制信号，进行调制得到期望的 PWM 波；该方法一般采用等腰三角波为载波，其任一点水平宽度和高度成线性关系且左右对称。载波(等腰三角波)与平缓变化的调制信号波(即要拟合的波形)相交， 在载波与信号波的交点控制器件通断，就得宽度正比于信号波幅值的脉冲，符合 PWM 的要求，相对于计算法，其处理过程计算简单。

在本章的实验中，实现使用计算法得到的呼吸曲线 PWM 波和 SPWM 波。在本实验中，PWM 波的生成使用软件延迟模拟，即：根据占空比，先输出高电平，然后程序延迟一段时间（task\_sleep 函数），再输出低电平，延迟一段时间。

# 实验步骤

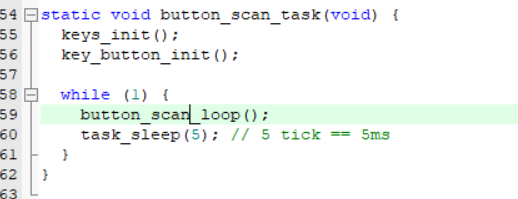
本实验使用轮询方式定期采样按键状态、定期更新 LED 状态。

创建按键采样 button\_scan\_task 和 LED 控制 led\_blink\_task2 个任务。2 者通

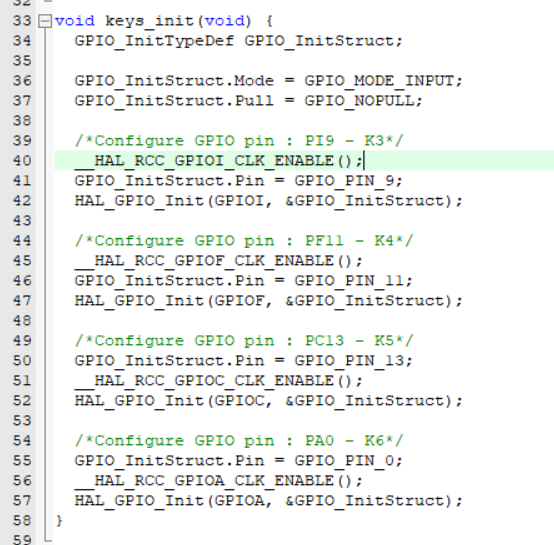
过一个 LED 的全局状态变量 led\_state 进行通信。

1. 设置按键 K3-K6的 GPIO 口。

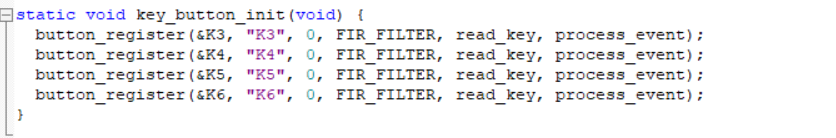
首先创建按键采样 button\_scan\_task这个任务，执行task\_create\_simple(button\_scan\_task, 0)，查看button\_scan\_task函数，发现里面包括初始化、循环扫描两部分：



在初始化中，首先绑定GPIO引脚，包括初始化时钟，绑定引脚两部分，这里初始化按键K3、K4、K5、K6分别对应PI9、PF11、PC13、PA0端口。



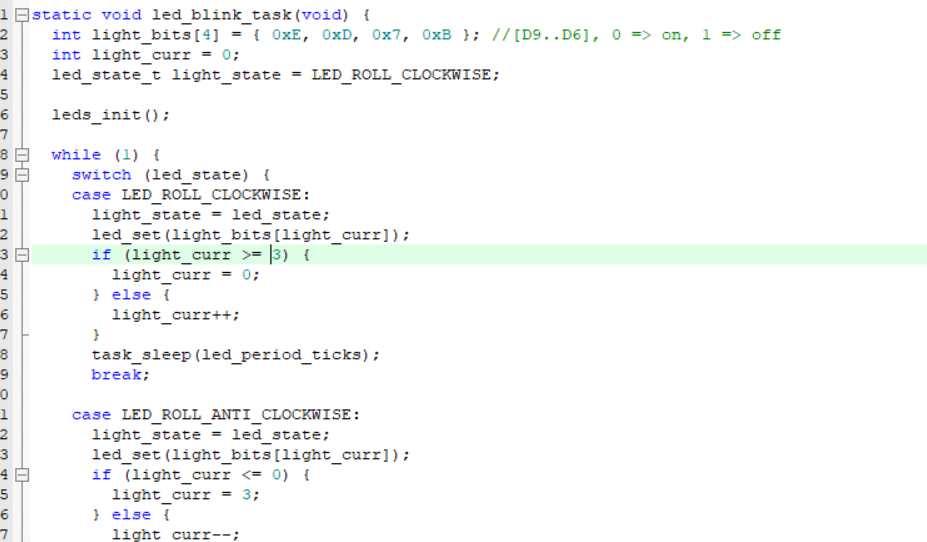
在key\_button\_init()中对读入的按键状态确定处理方式，即采用了process\_event函数，改变存储LED状态的变量led\_state：



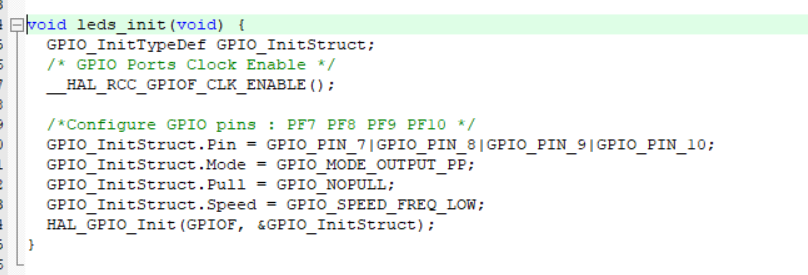
之后button\_scan\_task 每隔 5ms 读取一次按键状态，根据按键输出设置 LED 状态变量 led\_state。

1. 设置LED6-9的 GPIO 口。

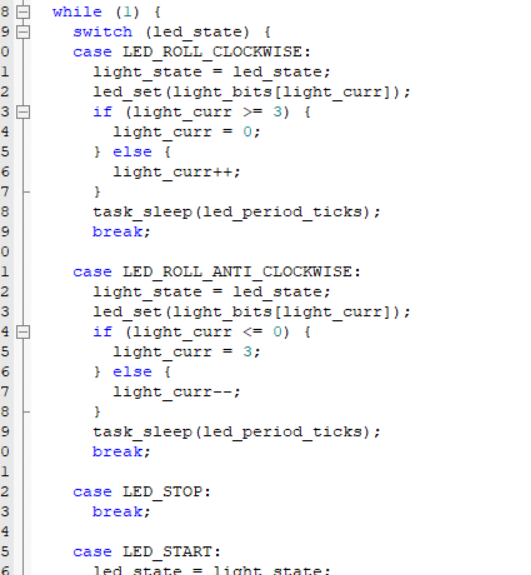
之后创建LED采样led\_blink\_task这个任务，执行task\_create\_simple(led\_blink\_task, 0)，查看led\_blink\_task函数，发现里面包括初始化、状态处理两部分：



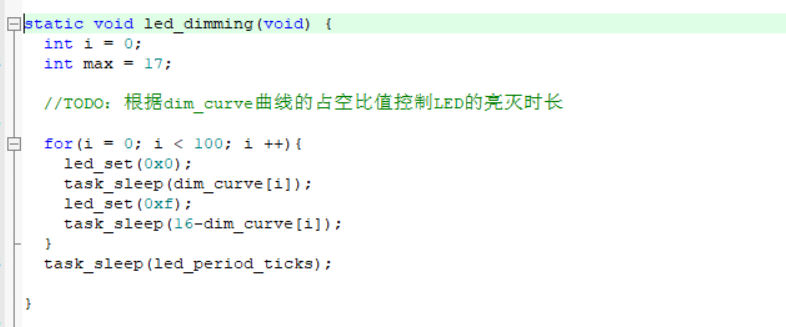
在初始化中，绑定LED对应的引脚，先初始化时钟，然后分别配置PF7、PF8、PF9、PF10引脚。



然后进入while(1)循环，每1ms读取一次LED的状态led\_state，然后根据状态执行暂停、前进、后退等操作。



当状态为LED\_DIMMING时，调用led\_dimming函数，控制4个灯一并为呼吸灯闪烁。实现方式为，用light\_bits[4]控制4个灯的亮灭，0表示灯亮, 1表示灯灭，先设置为0x0使得所有灯为亮的状态，再设置0xf灭掉所有的灯，并控制亮灭之间的时间间隔来改变占空比，进而实现呼吸灯的效果。



3. 测试输出结果

在 sscom 中输入 prj 2命令，将调用 project2\_main 函数。长按K6，观察4个led灯，可以观察到，4 个 LED 灯依次逐渐亮-灭-亮，松开后，呼吸灯停止，符合实验预期。

# 实验方案与实现

## 软件结构

如：**框图**说明软件系统结构，文字说明系统各模块重要的细节定义（常量变量定义、数据结构、主要函数等）。

在MDK环境中，点击启动调试并打开串口监视器查看串口输出。首先执行reset\_handler来启动开发板，并调用systemInit进入main初始化各外设，接下来初始化并启动多任务调度，在使用控制台程序中等待成用户输入命令。在本实验中我们输入命令prj 2,来执行project2\_main函数，在project2\_main函数中初始化led和按键的gpio口，并注册按键事件处理函数button\_register。接下来并行扫描button和led的状态，分别延迟5ms和1ms，根据扫描到的值修改状态变量，再执行对应的操作。

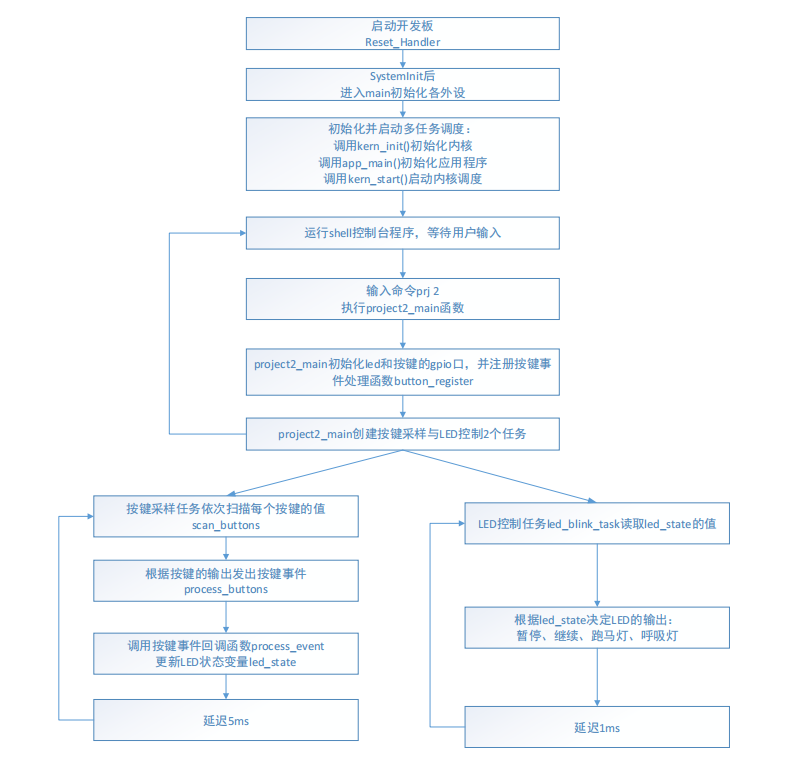


图2-1 执行流程

## 源代码

static uint8\_t dim\_curve[] = { 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2,

        2, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 7, 7, 8, 8, 9, 9,

        10, 10, 11, 11, 12, 13, 14, 14, 15, 16, 16, 15, 14, 14, 13, 12, 11, 11,

        10, 10, 9, 9, 8, 8, 7, 7, 6, 6, 6, 5, 5, 5, 5, 4, 4, 4, 4, 3, 3, 3, 3,

        3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1 };

static uint16\_t dim\_curve\_len = sizeof(dim\_curve) / sizeof(dim\_curve[0]);

static void led\_dimming(void) {

    int i = 0;

    int max = 17;

    //TODO：根据dim\_curve曲线的占空比值控制LED的亮灭时长

    for(i = 0; i < 100; i ++){

        led\_set(0x0);

        task\_sleep(dim\_curve[i]);

        led\_set(0xf);

        task\_sleep(16-dim\_curve[i]);

    }

    task\_sleep(led\_period\_ticks);

}

# 实验结果与分析

1.请描述从按下按键到 LED 灯改变状态的程序运行流程。

每隔5ms会扫描一次按键button的状态，若识别出摁下了按键，则会根据按键号对应地修改存储LED灯的状态变量led\_state。每隔1msLED控制任务会读取led\_state的值，根据值的内容改变LED灯的状态，如为LED\_ROLL\_CLOCKWISE时跑马灯顺时针闪烁，为LED\_ROLL\_ANTI\_CLOCKWISE则逆时针闪烁。

2.本实验通过轮询方式（不间断扫描）检查按键状态，因此会一直占用 CPU资源，这对低功耗应用场景不友好。如果要降低正常工作模式下的功耗，该如何处理？

可以选择不适用轮询的方式，而是采用外部中断的方法，按键按动时会产生高低电平的变化，可以将按键管脚设置为边缘触发中断（上边缘、下边缘或双边）。但按键抖动可能触发多次中断，所以处理中断时还需要以上的延迟或者重采样思想来消抖。该方法的优点是：CPU 效率高，实时性强，缺点则是：需要用掉一个额外的外部中断资源。编程比上述同步方法要复杂一些。

# 实验总结

此次实验是我们第一次尝试使用GPIO外设端口，通过使用GPIO引脚绑定LED、button等外设，了解了STM32 GPIO 的基本操作方式，包括初始化时钟，如何读写等操作，并学会了使用 C 语言控制 IO 寄存器完成设备控制功能。

和我们在课下做的7个小作业实验不用，这个实验是在硬件实验板上实现的，由于和硬件相关联，在实际应用中要考虑消抖，因为机械触点的弹性作用使得按键开关在闭合时不会马上稳定的接通，而是有一段时间的抖动，在断开时也不会立即断开，若不作处理这个抖动可能会造成错误的按键状态判断，故在做按键检测时需要加一个消抖的过程。本次实验中我们采用的是轮询的方式，每隔特定的时间查询一次状态，这样实时性虽然较好，但是会造成持续占用CPU，对低功耗应用场景不友好。