青风带你学 stm32f030 系列教程

---------- 库函数操作版本

出品论坛： www.qfv8.com 青风电子社区

作者： 青风
出品论坛: www.qfv8.com
淘宝店：http://qfv5.taobao.com
QQ 技术群：241364123
硬件平台：QF-STM32F030 开发板

2.1 点亮你的第一个 LED 灯

在讲第一个实例之前，我要先对许多初入 ARM 的朋友说明几个关键的学习问题：

首先是学习资料的准备，在新的处理器出来后，我们要如何入门，如何进行开发，这时相关的技术手册就是必须的了，以后我们的讲解与分享中都会回到技术手册，来分析下如何采用手册做到空手入门，实际上这也是工程师的必经之路。MCU 的设计者设计了非常多的外设寄存器结合处理器的内核构成了一个微控制器，而应用工程师仅仅只需要知道如何操作寄存器，而寄存器的操作实际上往往是傻瓜式的操作，当你认识到这一点的时候就知道 MCU 的控制实际上没有什么难度的，难的地方就是你要知道这些寄存器是干什么的，不同状态下代表什么。对于一个英语功底非常好的朋友读这些手册就相对简单了。

2.1.1 原理分析：

大家之前学习过 51 单片机，也使用过 IO 口。与 ARM CORTEX M0 的 IO 口配置有

点区别，51 不需要设置 IO 口为输入或者输出，而 CORTEX M0 的 IO 口有多种状态需要设置，那么下面我们一一介绍：

首先看看 IO 口的模式，查看 030 参考手册，找到如下寄存器说明，IO 口可以配置为 4 种模式：输入模式，输出模式，复用模式，模拟通道模式。由于 stm32f030 系列多数的 IO 管脚复用了其它的外设功能，比如 I2C, SPI, UART 等，那么此时就就必须要设置 IO 口为复用模式。而模拟通道则作为 AD, DA 的时候使用：

MODERy[1:0]: Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O mode.
00: Input mode (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

如果大家使用库函数编程的时候，可以在 **stm32f0xx_gpio.h** 库文件中找到设置 **IO** 模式的结构体 **GPIOMode_TypeDef**，这里完全是对照参考手册进行编写的 ：

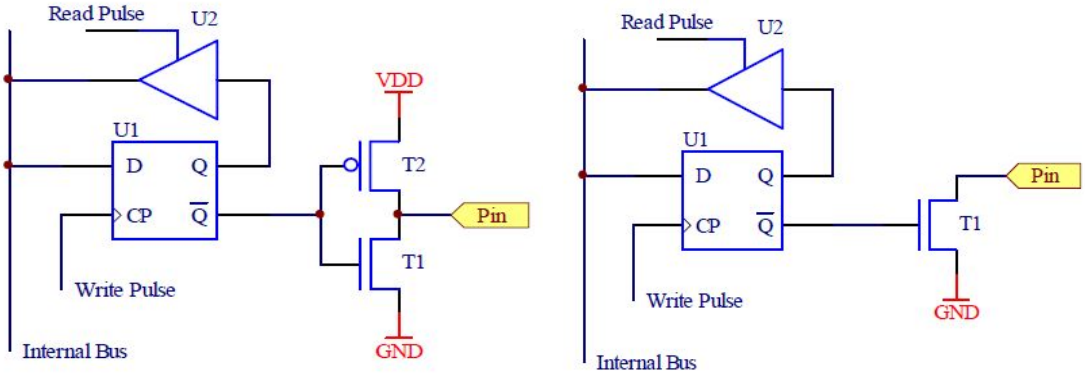
```
01. typedef enum
02. {
03.   GPIO_Mode_IN    = 0x00, /*!< GPIO Input Mode                    */
04.   GPIO_Mode_OUT   = 0x01, /*!< GPIO Output Mode                   */
05.   GPIO_Mode_AF    = 0x02, /*!< GPIO Alternate function Mode */
06.   GPIO_Mode_AN    = 0x03   /*!< GPIO Analog In/Out Mode          */
07. }GPIOMode_TypeDef;
```

首先我们来介绍下输入和输出模式也就是 **MODE_IN** 和 **MODE_OUT**，其中输出模式寄存器定义如下图所示，分为推挽输出和**开漏输出**。这时候大家就要回忆下模拟电路的的课程了：

Bits 15:0 **OTy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O output type.
0: Output push-pull (reset state)
1: Output open-drain

我使用下面一个等效图说明一下推挽输出和开漏输出。如左图所示：**U1** 是输出锁存器，执行 **GPIO** 管脚写操作时，在写脉冲（**Write Pulse**）的作用下，数据被锁存到 **Q** 和/**Q**。**T1** 和 **T2** 构成 **CMOS** 反相器，**T1** 导通或 **T2** 导通时都表现出较低的阻抗，但 **T1** 和 **T2** 不会同时导通或同时关闭，最后形成的是推挽输出。

如右图所示，为 **GPIO** 管脚在开漏输出模式下的等效结构示意图。开漏输出和推挽输出相比结构基本相同，但只有下拉晶体管 **T1** 而没有上拉晶体管。同样，**T1** 实际上也是多组可编程选择的晶体管。开漏输出的实际作用就是一个开关，输出“**1**”时断开、输出“**0**”时连接到 **GND**（有一定内阻）。



同样大家使用库函数编程的时候，可以在 在 **stm32f0xx_gpio.h** 库函数文件中定义了输出模式的结构体 **GPIOOType_TypeDef** ：

```
typedef enum
{
  GPIO_OType_PP = 0x00    //推挽
  GPIO_OType_OD = 0x01    //开漏
}GPIOOType_TypeDef;
```

IO 端口有输出速度的区别，大体上我们可以设置 IO 端口输出的速度在库函数中通过结构体来解决：

```
typedef enum
{
  GPIO_Speed_Level_1   = 0x01, /*!< Medium Speed */
  GPIO_Speed_Level_2   = 0x02, /*!< Fast Speed    */
  GPIO_Speed_Level_3   = 0x03   /*!< High Speed    */
}GPIOSpeed_TypeDef;
```

再来谈谈输入模式，输入的模式可以分为上拉和下拉模式，这就比较简单了，寄存器的设置如下图：

PUPDRy[1:0]: Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O pull-up or pull-down
00: No pull-up, pull-down
01: Pull-up
10: Pull-down
11: Reserved

同样大家使用库函数编程的时候，可以在 **stm32f0xx_gpio.h** 文件中找到设置输入模式的结构体 GPIOOPuPd_TypeDef ：

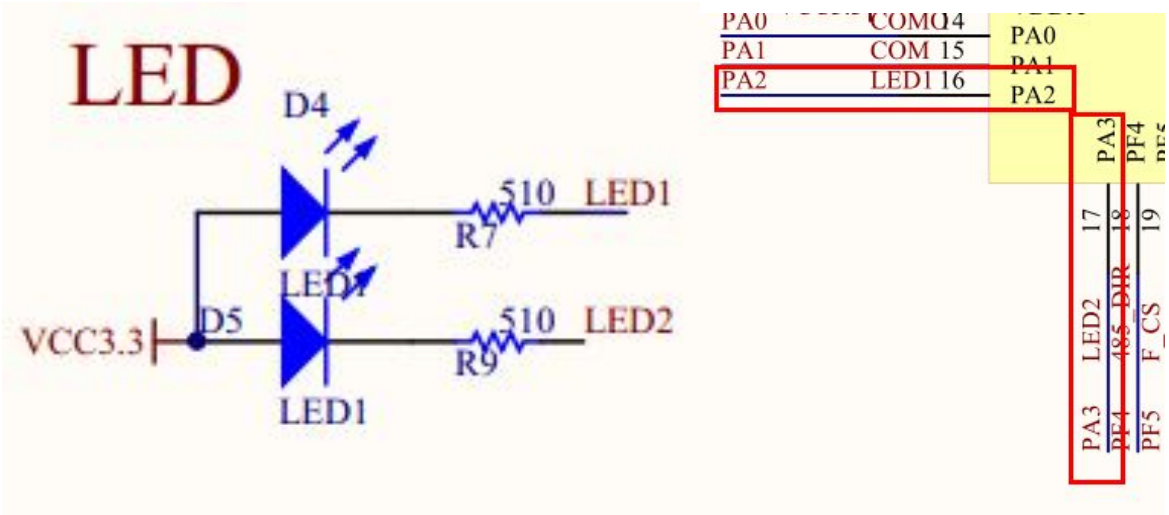
```
typedef enum
{
  GPIO_PuPd_NOPULL = 0x00,    //无上拉下拉
  GPIO_PuPd_UP       = 0x01,    //上拉
  GPIO_PuPd_DOWN     = 0x02    //下拉
}GPIOPuPd_TypeDef;

```

关于其数字的复用功能会在相应外设里进行介绍，我们这里只是谈通用 IO 口，也就是 GPIO，现在就来点亮一个 LED 灯，学过 51 的同学可以回忆下，51 是通过设置 IO 口输出 0 或者 1 来驱动 LED 灯的亮灭。在 STM32F030 中，我们通过设置 IO 端口为输出来点亮 LED 等。

2.1.2 硬件准备：

如上图所示： 青风 STM32F030 豪华开发板上，通过管脚 PA11 和管脚 PA12 连接 2 个 LED 灯，我们下面的任务首先来点亮它。IO 管脚接分别接一个发光二级管，然后接高电平，因此当把 IO 管脚定义为输出低电平的时候，就可以点亮发光二极管了。



2.1.3 软件准备以及编写：

按照前哨篇里的介绍，首先建立一个工程项目，采用库函数来在驱动 IO 口首先要添加几个驱动库，如下图所示：

上图红色框框中的几个文件都是 ST 官方给我们编好的库函数。那边用户在使用中，只需要编写 led.c 驱动文件和 main.c 主函数就 OK，整个工程项目大家如果加入分层的思想那么就对之后的移植非常有利。打个比方:底层和应用程隔离。底层驱动和应用层无关，main.c 使用的函数在 led.c 驱动中已经些好，这些才和硬件有关，这是需要移植到不同硬件时，main 主函数是可以不做任何修改的，只需要修改和底层相关的 led.c

驱动。

下面来分析下 led .c 的驱动编写：

```
#include "led.h"//首先设置.h 文件

void LED_Init(void)   //led 灯的初始化话，这里设置 IO 端口的模式
{
   GPIO_InitTypeDef GPIO_InitStruct; //调用 IO 定义的结构体
   RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);//时能 IO 端口时钟
   GPIO_InitStruct.GPIO_Pin = GPIO_Pin_2|GPIO_Pin_3 ;//设置连接 LED 等的 IO 端口
   GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT;//设置端口为输出模式
   GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;//设置输出推挽
   GPIO_InitStruct.GPIO_Speed =GPIO_Speed_Level_3;//设置输出速度为第三级
   GPIO_Init(GPIOA, &GPIO_InitStruct);   //把设置参数运用到结构体中
   GPIO_SetBits(GPIOA, GPIO_Pin_2 | GPIO_Pin_3);//熄灭 LED 灯
}

void LED_Open(void)         //点亮 led 灯子函数
{
GPIO_ResetBits(GPIOA, GPIO_Pin_2);
}

void LED_Close(void)        //熄灭 Led 灯子函数
{
GPIO_SetBits(GPIOA, GPIO_Pin_2);
}
```

上面的函数中 GPIO_ResetBits 函数和 GPIO_SetBits 函数在 stm32f0xx_gpio.c 驱动文件中所定义了。分别表示复位和置位相关 IO 管脚。

那么主函数的编写就比较简单了，我们需要调用下面 2 个头文件，才能够直接使用我们定义的子函数。如下使用 LED_Open()函数就能够点亮一个 LED 灯了，是不是很简单。

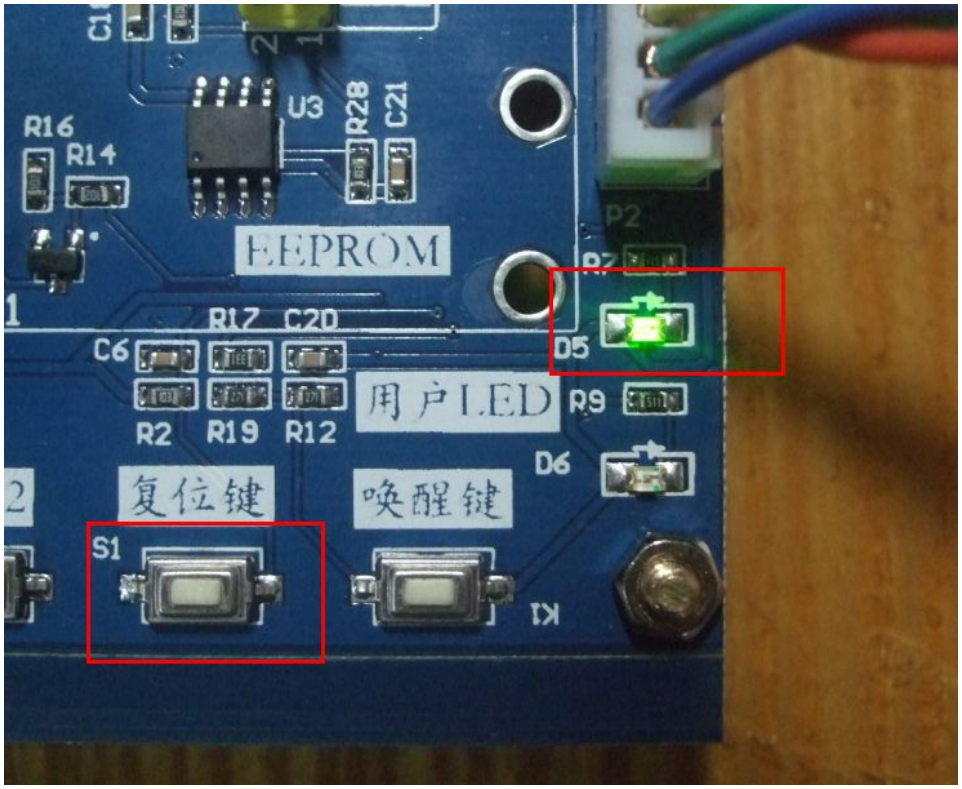
```
#include "stm32f0xx.h"
#include "led.h"
int main(void)
{
                LED_Init();//初始化
                while(1)
                {
                 LED_Open();//点亮
                }
}
```

那么加入一个小的延迟 delay 函数和打开与关闭 LED 子函数相结合，就可以实现 LED 闪烁的功能了，写一个软件延迟，函数如下所示：

```
#include "stm32f0xx.h"
#include "led.h"
void delay()     //延迟函数
{
                int i,j;
   for(i=0;i<1000;i++)
                  {
                     for(j=0;j<1000;j++);
                  }
}

int main(void)
{
                LED_Init();
                while(1)
                {
                 LED_Open(); //点亮
                 delay();
                 LED_Close();//熄灭
      delay();
                }
}
```

下载到青风 STM32F030 开发板上运行后的效果如下图所示，按下复位键，LED 开始闪烁：



如图所示，按下复位键后，上方的用户 led 灯不停闪烁