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

2.3 按键控制

按键的输入其实就是对 GPIO 口进行操作。同时引入了中断的概念。实际上按键控制分为两种情况，第一种是按键扫描，这种情况下，CPU 需要不停的工作，来判断 GPIO 口是否被拉低或者置高，效率是比较低的。另一种方式为中断控制，中断控制的效率很高，一旦系统 IO 口出现上升沿或者下降沿电平就会触发执行中断内的程序。

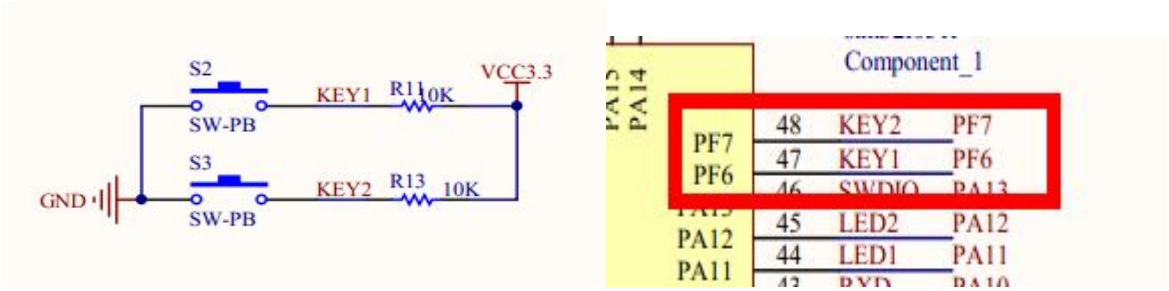
相信按键是所有接触过 MCU 的朋友都知道的，其基本原理我就不啰嗦了，这一节的教程主要针对了实验四和实验三。

下面我就来分别介绍下 STM32F030 的按键控制方式。

首先来学习下按键扫描方式：

2.3.1 硬件准备：

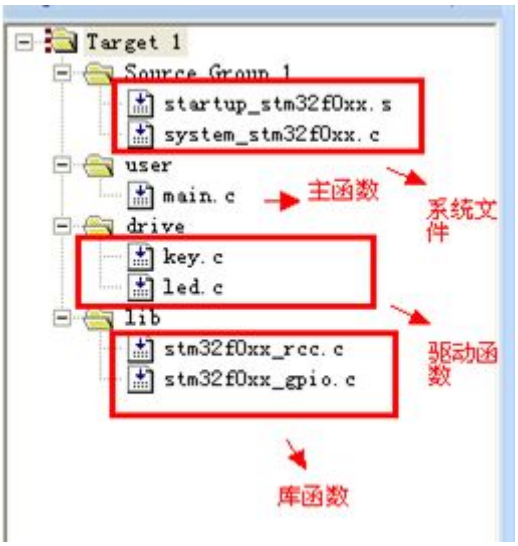
在青风 stm32f030 豪华开发板上设置了 2 个用户按键如下图所示：



Key1 和 Key2 分别和 PF6 和 PF7 两个 IO 管脚相连。这里设置了按键管脚加了 3.3V 的上拉，可以提高驱动能力。当 IO 管脚为低的时候可以判断管脚已经按下。通过 key 的按下来控制 led 的亮灭。

2.3.2 软件准备：

在代码文件中，实验三建立了一个演示历程，我们打开看看需要那些库文件。打开 user 文件夹中的 key 工程：



如上图所示：按照上面的方式添加好相应的函数，其中 **led.c** 和 **key.c** 是用户自己所编写的驱动子文件，用于控制按键和 **LED** 灯。**Lib** 库中只需要添加：**stm32f0xx_rcc.c** 和 **stm32f0xx_gpio.c** 两个文件。

Led.c 和我们第一节所编写的驱动一样，下面我们来看看如何编写 **key.c** 文件。**Key.c** 文件主要是要起到两个作用：第一：初始化开发板上的按键。第二：扫描判断按键是否有按下，按键扫描是通过 **MCU** 不停的判断端口的状态来实现的。完成这两个功能够就可以在 **main.c** 文件中直接调用本驱动了。下面看看代码：

```
01.  #include "key.h"
02.
03. void KEY_Init(void)//首先对 key 进行初始化，也就是设置 GPIO 的模式
04. {
05.  GPIO_InitTypeDef GPIO_InitStruct;
06.  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOF, ENABLE);//设置 GPIO 时钟
07.    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_7 ;
08.    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;//设置管脚为输入模式
09.    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_Level_2;
10.    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;//设置为上拉输入
11.    GPIO_Init(GPIOF, &GPIO_InitStruct);
12.
13.  }
14.
15.  void Delay(uint32_t temp)
16. {
17.    for(; temp!= 0; temp--);
18. }
19.
20. uint8_t KEY_Down(GPIO_TypeDef* GPIOx,uint16_t GPIO_Pin)//按键扫描子函数
```

```
{
    if(GPIO_ReadInputDataBit(GPIOx,GPIO_Pin) == 0 )   //检测是否有按键按下
  {
     Delay(10000);            //延时消抖
    if(GPIO_ReadInputDataBit(GPIOx,GPIO_Pin) == 0 )
                    {
    while(GPIO_ReadInputDataBit(GPIOx,GPIO_Pin) == 0);   //等待按键释放
                          return    0    ;
                    }
          else
          return 1;
    }
 else
    return 1;
}
```

那么在主函数中就可以直接调用 **key,c 和 led.c** 两个子文件，代码如下：

```
#include "stm32f0xx.h"
#include "led.h"
#include "key.h" //调用子函数的头文件

int main(void)
{
   LED_Init();//led 初始化
 KEY_Init();//按键初始化
   GPIO_ResetBits(GPIOA,GPIO_Pin_11 );//点亮一个 led 等

   while(1)
   {
      if( KEY_Down(GPIOF,GPIO_Pin_7) ==0)//判定按键是否按下
 {

          GPIO_WriteBit(GPIOA, GPIO_Pin_11,
      (BitAction)((1-GPIO_ReadOutputDataBit(GPIOA, GPIO_Pin_11))));//反转 led 灯
         GPIO_WriteBit(GPIOA, GPIO_Pin_12,
                    (BitAction)((1-GPIO_ReadOutputDataBit(GPIOA, GPIO_Pin_12))));
 }
   }
}
```

那么更加好的按键控制方法是使用中断控制，这样可以大大节省了 **cpu** 的占有率。中断是指由于接收到来自外围硬件（相对于中央处理器和内存）的异步信号或来自软件的同步信号，而进行相应的硬件／软件处理。发出这样的信号称为进行中断请求（**interrupt request，IRQ**）。硬件中断导致处理器通过一个上下文切换（**context switch**）来保存执行状态（以程序计数器和程序状态字等寄存器信息为主）；软件中断则通常作

为 CPU 指令集中的一个指令，以可编程的方式直接指示这种上下文切换，并将处理导向一段中断处理代码。中断在计算机多任务处理，尤其是实时系统中尤为有用，这样的系统，包括运行于其上的操作系统，也被称为“中断驱动的”。简单的来说就比如某个人正在做某事，突然来了个电话，他就要停下手中的事情去接电话，中断相当于这个电话。触发中断后跳出原来运行的程序去执行中断处理。

在使用 stm32f051 库函数来完成中断，需要设置如下几个地方：

第一：中断嵌套的设置。

第二：外部 GPIO 中断的设置。

首先是中断嵌套的问题：在 stm32f03xx 的参考手册第 157 页有一个中断嵌套列表，列出了各类中断的地址，在库文件 stm32f0xx_misc.c 里对嵌套进行了设置。并且在 stm32f0xx.h 文件中给出了中断列表中各类中断的中断频道号，并且封装成 IRQn_Type 结构体。

我们设置中断的类型只要设置如下结构体的参数就行：

```
58.    typedef struct
59.  {
60.    uint8_t NVIC_IRQChannel;     /*!< Specifies the IRQ channel to be enabled or disabled.
61.                                        This parameter can be a value of @ref IRQn_Type
62.                                      (For the complete STM32 Devices IRQ Channels list,
63.                                           please refer to stm32f0xx.h file) */
64.
65.    uint8_t NVIC_IRQChannelPriority;   /*!< Specifies the priority level for the IRQ channel
66.                                       Specified   in NVIC_IRQChannel. This parameter can be a
67.                                            value   between 0 and 3.   */
68.
69.    FunctionalState NVIC_IRQChannelCmd;   /*!< Specifies whether the IRQ channel defined in
70.                                                NVIC_IRQChannel   will be enabled or disabled.
71.                                      This parameter can be set either to ENABLE or DISABLE */
72.  } NVIC_InitTypeDef;
```

其中 NVIC_IRQChannel 在 stm32f0xx_misc.c 里对嵌套进行了设置，NVIC_IRQChannelPriority 可以在 1 到 3 直接进行设置。 NVIC_IRQChannelCmd 频道命令主要是使能和禁能的作用。

第二个问题是外部 IO 中断的设置，主要是 stm32030 中断各种外部中断所使用的。在库函数中，在 stm32f0xx_exti.c 文件中进行了详细设置，我们只需要配置下面一个结构体就可以实现外部中断的控制了：

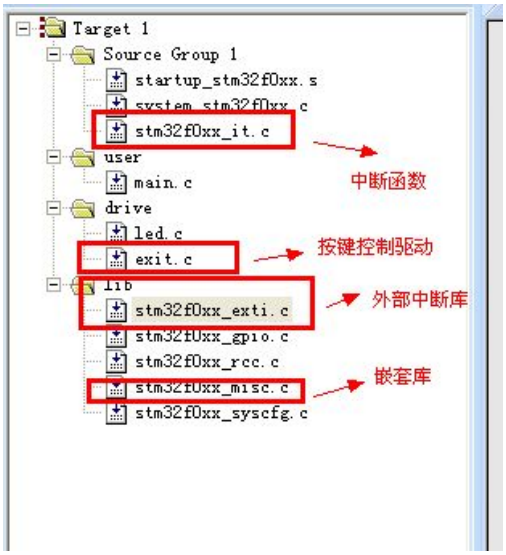
```
73.   typedef struct
74.  {
75.    uint32_t EXTI_Line;        //外部中断线程
76.    EXTIMode_TypeDef EXTI_Mode;     //外部中断模式
77.    EXTITrigger_TypeDef EXTI_Trigger; //外部中断触发配置
78.    FunctionalState EXTI_LineCmd;        //外部中断使能命令
79.  }EXTI_InitTypeDef;
```

按照上面的配置要求，我们编写 exit.c 外部中断子函数，如下面的代码：

```
80.   #include "exit.h"
```

```
81. void EXIT_KEY_Init(void)
82. {
83.     GPIO_InitTypeDef GPIO_InitStruct;
84.    EXTI_InitTypeDef EXTI_InitStruct;
85.     NVIC_InitTypeDef NVIC_InitStruct;
86.     RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
87.     RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOF, ENABLE);
88.
89.       /* Configyre P[A|B|C|D|E]0   NIVC   中断嵌套配置*/
90.       NVIC_InitStruct.NVIC_IRQChannel = EXTI4_15_IRQn;//配置 4—15 通道
91.       NVIC_InitStruct.NVIC_IRQChannelPriority = 0x00;//优先级设为 0
92.       NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;//使能通道
93.       NVIC_Init(&NVIC_InitStruct);
94.
95.       /* EXTI line gpio config(PF7) *外部 IO 端口配置*/
96.       GPIO_InitStruct.GPIO_Pin = GPIO_Pin_7;
97.  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;
98.  GPIO_InitStruct.GPIO_Speed = GPIO_Speed_Level_2;
99.      GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP; //  上拉输入
100.      GPIO_Init(GPIOF, &GPIO_InitStruct);
101.
102.          /* EXTI line(PF7) mode config 外部中断配置  */
103.      SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOF, EXTI_PinSource7);//配置外部中断源
104.      EXTI_InitStruct.EXTI_Line = EXTI_Line7; //外部中断线程
105.      EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;//外部中断模式
106.          EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Falling; //下降沿中断
107.          EXTI_InitStruct.EXTI_LineCmd = ENABLE;//使能外部中断
108.          EXTI_Init(&EXTI_InitStruct);
109.      }
```

整个文件工程应该调动的文件如下图所示：



其中 exit.c 就是上面我们编写的按键中断初始化的驱动函数。而 Lib 树下文件我们之间添加 ST 公司提供的库函数。这里面，我们需要在 stm32f0xx_it.c 中加入相应的中断后执行函数，也就是发生按键中断后，我们执行反转 LED 的命令：

```
110.      void EXTI4_15_IRQHandler(void)
111. {
112.    if(EXTI_GetITStatus(EXTI_Line7) != RESET)
113.    {
114.      GPIO_WriteBit(GPIOA, GPIO_Pin_2,
115.                    (BitAction)((1-GPIO_ReadOutputDataBit(GPIOA, GPIO_Pin_2))));
116.      GPIO_WriteBit(GPIOA, GPIO_Pin_3,
117.                    (BitAction)((1-GPIO_ReadOutputDataBit(GPIOA, GPIO_Pin_3))));
118.      //翻转 led 灯
119.      EXTI_ClearITPendingBit(EXTI_Line7);//清除中断标志
120.    }
121. }
122.
```

通过上面几个步骤，下面的主函数就相当的简单了，我们直接调用 exit.h 函数就可以完成按键中断的初始化了。看看下面代码，是不是非常简单不解释：

```
123.      #include "stm32f0xx.h"
124. #include   "led.h"
125. #include   "exit.h"
126.
127. int main(void)
128. {
129.      SystemInit();
130.      LED_Init();//led 灯初始化
131.      GPIO_ResetBits(GPIOA,GPIO_Pin_2);
132.      EXIT_KEY_Init();//按键中断初始化
133.      while(1)
134.      {
135.          }
136.      }
```

实验下载到 QF-STM32F030 开发板后的实验现象如下：