按键检测组件

1、按键检测组件简介

在嵌入式系统或单片机程序开发中，经常会使用到外部按键的触发，可能很多人一看到按键，感觉很简单，不就是电平触发，然后单片机去检测按键触发电平嘛，然后再用一个延时对按键进行消抖，然后编写按键触发需要处理的事物逻辑。对，实际的确如此，可是当出现一下现象和要求时，你改作何处理：

1）要求系统不堵塞，实时性要好，那按键的消抖就不能直接使用死循环延时来解决，尽管 5-10 毫秒的延时可能不多，如果出现矩阵键盘，你作何处理，这会严重影响系统的实时性。

2）尽量降低按键和整个工程的耦合性。大家在写程序的工程时应该会出现写着写着，发现程序变成了一团乱糊，每一个功能模块之间都有关联，每一个功能模块都不能独立存在，导致一旦出现 Bug，就很难针对性模块化的去查找和核对问题所在的地方。也许可以迅速定位问题出现的点，由于各个功能模块耦合性差，所以修改起来让你无从下手。

3）当工程要求添加新的按键功能时，由于每一个功能模块耦合性差，彼此都有干涉，导致你无从下手。

4）当要求同一个按键同时实现单击、双击、长安的功能时，你是不是要疯掉了。。。

5）当客户要求将高电平触发变成低电平触发时，你是不是开始郁闷的大片大片的修改按键的初始化、按键的触发电平，不难，但改的挺多。

针对以上情况，再使用那种比较“笨拙”、“死板”的按键检测方法，还能完美应对吗？我想是否定的。因此我们针对实际开发中遇见的种种的情况和要求，设计开发了一套能够兼容各种平台的按键检测组件。

我们设计开发的按键检测按键主要借鉴了面向对象的编程思路（比如在使用 C#编写一个 winform 上的按键响应事件时，你会发现双击按键会生成一个按键点击事件，而这个事件被绑定在了该按键的单击触发的委托函数上，它并没有重头到尾将这个按键单击事件重新写一边，可能表述并不完全妥当准确），嵌入式（C）没法使用面向对象的类和继承的属性去写嵌入式按键驱动，但是我们可以开一个注册按键的结构体数组，将要驱动的按键注册到这个按键的结构数组里面；嵌入式里面也没有委托，我们可以使用回调函数（函数指针）绑定注册的按键来实现面向对象的按键的委托；通过计数来实现按键的消抖；通过对按键注册不同的按键模式来实现同一个按键的多中模式的触发；通过修改注册按键的触发模式，来修改按键的电平触发模式。这样的一个设计同样也能使整个按键检测组件完全独立，是组件的耦合性降低到最低。

2、按键检测组件函数接口介绍

1	char Init_Key_Struct(void (*Update_Key_CallBack)(void),void (*Debug_CallBack)(unsigned char *debug_mess));	**初始化按键**
2	char Reg_Key(unsigned char *key_s,const unsigned short count, Trig_Mode_TypeDef Trig_Mode_E, Key_Mode_TypeDef Key_Mode_E, void (*Key_Click_CallBack)(void));	**添加注册按键（注：如果按键已经注册过，那么再次注册会覆盖之前注册过的相同的按键）**
3	char Key_Detect(void);	**按键检测**
4	char *Get_Version_Mess(void);	**打印 Key_Detect 组件版本信息**

2.1、初始化按键

在使用这个组件时，必须调用 Init_Key_Struct 函数接口初始化一下这个按键检测组件，不初始化按键组件，是没法成功注册按键的。该函数有一个返回值，返回 0 表示按键检测组件初始化成功；返回 1，表示按键检测组件初始化失败。

Init_Key_Struct 接口函数的入口参数配置如下：

1）void (*Update_Key_CallBack)(void)是一个函数指针，功能是用来更新注册了的按键的电平状态的，这个不能为 NULL，一定要添加。

2）void (*Debug_CallBack)(unsigned char *debug_mess)是一个函数指针，功能是用来打印按键组件的异常信息的，如果不需要打印按键组件异常信息，可以直接写 NULL。

2.2、添加注册按键（注：如果按键已经注册过，那么再次注册会覆盖之前注册过的相同的按键）

初始完按键组件后，就可以注册按键了。调用 Reg_Key 函数接口来注册按键。该函数有一个返回值，返回 0 表示注册按键成功；返回 1（表示注册的按键的触发回调函数为 NULL）或 2（表示按键注册数量已经达到上线，请修改宏定义 Key_Num_Max 的值），表示注册按键失败。

Reg_Key 接口函数的入口参数配置如下：

1）unsigned char *key_s 是要注册的按键的状态指针，将要注册的按键的地址写在这里便好。

2）const unsigned short count 是按键被按下后计数多少次进行按键触发，这个参数根据按键检测频率以及要求的手感进行编写。

3）Trig_Mode_TypeDef Trig_Mode_E 是按键电平触发模式，它是一个枚举变量（Trig_Mode_TypeDef），枚举变量定义如下：

```
/**
  * @brief 按键触发模式状态枚举
  */
typedef enum
{
    N_Trig = 0,          /*!< 0 空 */
    L_Trig ,             /*!< 1 低电平触发 */
    H_Trig,              /*!< 2 高电平触发 */
}Trig_Mode_TypeDef;
```

1）Key_Mode_TypeDef Key_Mode_E 是按键模式，它是一个枚举变量（Key_Mode_TypeDef），枚举变量定义如下：

```
/**
  * @brief 按键模式状态枚举
  */
typedef enum
{
    N_Click = 0,     /*!< 0 空 */
    S_Click ,        /*!< 1 单击 */
    D_Click,         /*!< 2 双击 */
```

```
    L_Press,          /*!< 3 长按 */
}Key_Mode_TypeDef;
```

2） void (*Key_Click_CallBack)(void)是按键触发回调函数，它是一个函数指针，主要功能是用来存放按键要处理的事件代码。

2.3、按键检测

Key_Detect 主要是用来检测按键的，使用该函数接口时，需要周期性的调用，比如使用定时器器，以 5 毫秒的频率调用，或者放在大循环里以一个固定的周期调用。该函数接口有一个返回值，如果返回 0，表示该函数接口一直在正常运行，如果返回 1，表示更新按键电平状态的回调函数 Update_Key_CallBack 为 NULL。

2.4、下面贴出该按键检测组件测试的 DEMO

```
/**
  ******************************************************************************
  * @file    main.c
  * @author  wr
  * @version V1.0
  * @date    2018-4-28
  * @brief   功能：按键检测组件测试 DEMO
  ******************************************************************************
  * @attention
  *
  * 参考资料：参考了魔电物联的源码和它的框架
  *
  ******************************************************************************
  */

#include "public.h"
#include "usart.h"
#include "key.h"
#include "key_detect.h"


//打印组件调试信息
void Print_Debug_mess(unsigned char *debug_Mess)
{
    USART_Send_Str(USART1, debug_Mess);
}

//按键 0 回调函数:打印按键以及打印按键版本号
void key0_CallBack(void)
{
    static char i=0;
```

```
	Print(USART1,"key0_CallBack:KEY0 is triggered.\r\n",Get_Version_Mess);
	if(i==0)
	{
		i=1;
		Reg_Key(&KEY0, 100, L_Trig, L_Press, key0_CallBack);//双击
	}
}

void key0_S_CallBack(void)
{
	Print(USART1,"key0_S_CallBack:KEY0 is triggered.\r\n",Get_Version_Mess);
}

void key0_L_CallBack(void)
{
	Print(USART1,"key0_L_CallBack:KEY0 is triggered.\r\n",Get_Version_Mess);
}

//按键 1 回调函数
void key1_CallBack(void)
{
	Print(USART1,"KEY1 is triggered.\r\n");
}

//按键 2 回调函数
void key2_CallBack(void)
{
	Print(USART1,"KEY2 is triggered.\r\n");
}

//按键 3 回调函数
void key3_CallBack(void)
{
	Print(USART1,"KEY3 is triggered.\r\n");
}

//更新按键状态
void Update_Key_CallBack(void)
{
	KEY0=GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_1);	//读取按键 0 状态
	KEY1=GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_4);	//读取按键 1 状态
	KEY2=GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_5);	//读取按键 2 状态
	KEY3=GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_0);	//读取按键 3 状态
}
```

```
int main(void)
{
	Init_Delay();//初始化延时函数，即系统滴答定时器（本程序中的延时是使用系统滴答定时器进行延时的）
	NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);//设置 NVIC 中断分组 2:2 位抢占优先级，2 位响应优先级
	Init_USART1(115200); //初始化串口 1，波特率：115200
	Init_KEY();

	//初始化按键结构体
	Init_Key_Struct(Update_Key_CallBack,Print_Debug_mess);

	//低电平触发
	Reg_Key(&KEY3, 3, L_Trig, S_Click, key3_CallBack);//单击
	Reg_Key(&KEY0, 50, L_Trig, D_Click, key0_CallBack);//双击
	Reg_Key(&KEY0, 3, L_Trig, S_Click, key0_S_CallBack);//单击
	Reg_Key(&KEY0, 100, L_Trig, L_Press, key0_L_CallBack);//双击
	Reg_Key(&KEY1, 100, L_Trig, L_Press, key1_CallBack);//长按
	Reg_Key(&KEY2, 3, L_Trig, S_Click, key2_CallBack);//单击

	//高电平触发
//	Reg_Key(&KEY3, 3, H_Trig, S_Click, key3_CallBack);//单击
//	Reg_Key(&KEY0, 50, H_Trig, D_Click, key0_CallBack);//双击
//	Reg_Key(&KEY1, 100, H_Trig, L_Press, key1_CallBack);//长按
//	Reg_Key(&KEY2, 3, H_Trig, S_Click, key2_CallBack);//单击

	while(1)
	{
		Delay_ms(5);//因为没有其他任务，所以就用延时来创建模拟一个 6 毫秒的调度频率来调用按键检测 Key_Detect
函数接口
		Key_Detect();
	}
}
```