目 录

第1章　系统节拍定时(SysTick)

1.1　SysTick 功能简介

SysTick 是一个简单的系统时钟节拍计数器，它属于 ARM Cortex-M3 内核嵌套向量中断控制器 NVIC 里的一个功能单元，而非片内外设。SysTick 常用于操作系统（如：μC/OS-II、FreeRTOS 等）的系统节拍定时。

由于 SysTick 是属于 ARM Cortex-M3 内核里的一个功能单元，因此使用 SysTick 作为操作系统节拍定时，使得操作系统代码在不同厂家的 ARM Cortex-M3 内核芯片上都能够方便地进行移植。

当然，在不采用操作系统的场合下 SysTick 完全可以作为一般的定时/计数器来使用。SysTick 是一个 24 位的计数器，采用倒计时方式。SysTick 设定初值并使能后，每经过 1 个系统时钟周期，计数值就减 1。计数到 0 时，SysTick 计数器自动重装初值并继续运行，同时申请中断，以通知系统下一步做何动作。

1.2　SysTick 基本操作

利用《Stellaris 外设驱动库》操作 SysTick 是非常简单的。无论是配置还是操作，SysTick 的用法都比一般片内外设简单。

表 1　函数 SysTickPeriodSet( )

功能	设置 SysTick 计数器的周期值
原型	void SysTickPeriodSet(unsigned long ulPeriod)
参数	ulPeriod：是 SysTick 计数器每个周期的时钟节拍数，取值 1～16777216
返回	无

表 2　函数 SysTickPeriodGet( )

功能	获取 SysTick 计数器的周期值
原型	unsigned long SysTickPeriodGet(void)
参数	无
返回	1～16777216

表 3　函数 SysTickEnable( )

功能	使能 SysTick 计数器，开始倒计数
原型	void SysTickEnable(void)
参数	无
返回	无

表 4　函数 SysTickDisable( )

功能	关闭 SysTick 计数器，停止计数
原型	void SysTickDisable(void)
参数	无
返回	无

表 5　函数 SysTickValueGet( )

功能	获取 SysTick 计数器的当前值
原型	unsigned long SysTickValueGet(void)
参数	无
返回	SysTick 计数器的当前值，该值的范围是：0～函数 SysTickPeriodSet( )设定的初值－1

程序清单 1.1 是 SysTick 的一个简单应用，能利用其计算一段程序的执行时间，结果通过 UART 输出。在程序中，被计算执行时间的是 SysCtlDelay( )这个函数，延时时间为 50000μs，最终实际运行的结果是 50004μs，误差很小。

程序清单 1.1　SysTick 例程：计算一段程序的执行时间

```
#include   "systemInit.h"
#include   "uartGetPut.h"
#include   <systick.h>
#include   <stdio.h>

//  主函数（程序入口）
int main(void)
{
    unsigned long ulStart, ulStop;
    unsigned long ulInterval;
    char s[40];

    jtagWait( );                                    //  防止 JTAG 失效，重要！
    clockInit( );                                   //  时钟初始化：晶振，6MHz
    uartInit( );                                    //  UART 初始化

    SysTickPeriodSet(6000000UL);                    //  设置 SysTick 计数器的周期值
```

```
    SysTickEnable( );                                   //  使能 SysTick 计数器

    ulStart = SysTickValueGet( );                       //  读取 SysTick 当前值（初值）
    SysCtlDelay(50 * (TheSysClock / 3000));             //  延时一段时间
    ulStop = SysTickValueGet( );                        //  读取 SysTick 当前值（终值）

    SysTickDisable( );                                  //  关闭 SysTick 计数器
    ulInterval = ulStart - ulStop;                      //  计算时间间隔

    sprintf(s, "%ld us\r\n", ulInterval / 6);           //  输出结果，单位：微秒
    uartPuts(s);

    for (;;)
    {
    }
}
```

1.3 SysTick 中断控制

SysTick 的中断控制也非常简单，配置时只需要使能 SysTick 中断和处理器中断，而且进入中断服务函数后硬件会自动清除中断状态，无需手工来清除。

表 6　函数 SysTickIntEnable( )

功能	使能 SysTick 中断
原型	void SysTickIntEnable(void)
参数	无
返回	无

表 7　函数 SysTickIntDisable( )

功能	禁止 SysTick 中断
原型	void SysTickIntDisable(void)
参数	无
返回	无

表 8　函数 SysTickIntRegister( )

功能	注册一个 SysTick 中断的中断服务函数
原型	void SysTickIntRegister(void (*pfnHandler)(void))
参数	pfnHandler：指向 SysTick 中断产生时被调用的函数的指针
返回	无

表 9　函数 SysTickIntUnregister( )

功能	注销一个 SysTick 中断的中断服务函数
原型	void SysTickIntUnregister(void)
参数	无
返回	无

程序清单 1.2 是 SysTick 中断的简单示例。在 SysTick 中断服务函数 SysTick_ISR( )里不需要手工清除中断状态，直接执行用户代码即可。程序运行后，LED 会不断闪烁。

程序清单 1.2　SysTick 例程：中断操作

```
#include   "systemInit.h"
#include   <systick.h>

//   定义 LED
#define   LED_PERIPH          SYSCTL_PERIPH_GPIOG
#define   LED_PORT            GPIO_PORTG_BASE
#define   LED_PIN             GPIO_PIN_2

//   主函数（程序入口）
int main(void)
{
      jtagWait( );                                          //   防止 JTAG 失效，重要！
      clockInit( );                                         //   时钟初始化：晶振，6MHz

      SysCtlPeriEnable(LED_PERIPH);                         //   使能 LED 所在的 GPIO 端口
      GPIOPinTypeOut(LED_PORT, LED_PIN);                    //   设置 LED 所在管脚为输出

      SysTickPeriodSet(3000000UL);                          //   设置 SysTick 计数器的周期值
      SysTickIntEnable( );                                  //   使能 SysTick 中断
      IntMasterEnable( );                                   //   使能处理器中断
      SysTickEnable( );                                     //   使能 SysTick 计数器

      for (;;)
      {
      }
}

//   SysTick 计数器的中断服务函数
void SysTick_ISR(void)
{
      unsigned char ucVal;

      //   硬件会自动清除 SysTick 中断状态
```

```
    ucVal = GPIOPinRead(LED_PORT, LED_PIN);                    //  反转 LED
    GPIOPinWrite(LED_PORT, LED_PIN, ~ucVal);
}
```

1.4 SysTick 应用：模拟 PC 按键重复特性

PC 按键具有如下特性：当按下某个键时，系统立即响应，如果未松手，则键值会重复出现，直到松手时才停止。如图 1.1 所示的 Windows 键盘属性设置，有两个重要的按键参数：重复延迟和重复率。重复延迟是指从按下键不松手开始，到第 1 次键值重复开始时的时间间隔；重复率是指以后连续重复的速率，每两次重复之间的时间间隔越短重复越快。

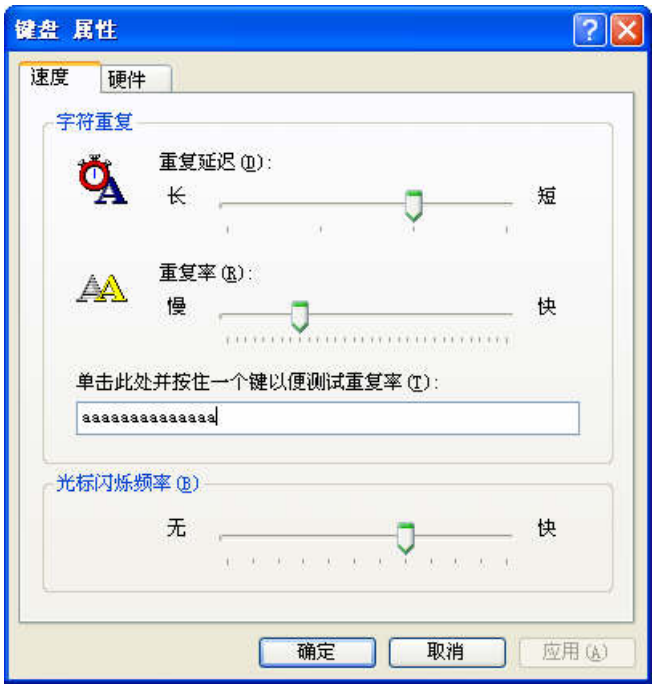

图 1.1　Windows 键盘属性设置对话框

程序清单 1.3 是 SysTick 的一个应用示例，利用其定时中断扫描按键 KEY 的输入，并模拟出 PC 按键的重复特性。在程序里，SysTick 设置的定时中断周期是 10ms，可自动地消除按键抖动，同时还能保证每秒 100 次的按键采样速率。重复延时参数是 KEY_DELAY，即按下 KEY 后若不松手则等待 10×KEY_DELAY 毫秒后开始重复。重复速率参数是 KEY_SPEED，即开始重复后每两次重复之间的时间间隔为 10×KEY_SPEED 毫秒。

在 SysTick 中断服务函数 SysTick_ISR( )里，采用状态处理的方法，实现了预定的功能，并没有出现任何无谓的等待。

程序清单 1.3　SysTick 应用：模拟 PC 按键重复特性

```
#include   "systemInit.h"
#include   "uartGetPut.h"
#include   <systick.h>

//  定义 KEY
#define   KEY_PERIPH          SYSCTL_PERIPH_GPIOD
#define   KEY_PORT            GPIO_PORTD_BASE
```

```
#define    KEY_PIN                GPIO_PIN_1

//   设置 KEY 重复延迟和重复速率参数
#define    KEY_DELAY              75
#define    KEY_SPEED              10

//   定义按键缓冲区
char KEY_Buf = '\0';

//   主函数（程序入口）
int main(void)
{
       jtagWait( );                                          //   防止 JTAG 失效，重要！
       clockInit( );                                         //   时钟初始化：晶振，6MHz
       uartInit( );                                          //   UART 初始化
       uartPuts("\r\n");

       SysCtlPeriEnable(KEY_PERIPH);                         //   使能 KEY1 所在的 GPIO 端口
       GPIOPinTypeIn(KEY_PORT, KEY_PIN);                     //   设置 KEY1 所在管脚为输出

       SysTickPeriodSet(10 * (TheSysClock / 1000));          //   设置 SysTick 周期，定时 10ms
       SysTickIntEnable( );                                  //   使能 SysTick 中断
       IntMasterEnable( );                                   //   使能处理器中断
       SysTickEnable( );                                     //   使能 SysTick 计数器

       for (;;)
       {
              SysCtlSleep( );                                //   进入睡眠省电模式

              if (KEY_Buf != '\0')                           //   如果 KEY 缓冲区不空
              {
                     uartPutc(KEY_Buf);                      //   显示 KEY 值
                     KEY_Buf = '\0';                         //   清空 KEY 缓冲区
              }
       }
}

//   SysTick 计数器的中断服务函数
void SysTick_ISR(void)
{
       static tBoolean bStatus = false;                      //   KEY 状态：false 松开，true 按下
       static unsigned short usDelayCnt = KEY_DELAY;         //   重复延时计数器
       static unsigned short usSpeedCnt = KEY_SPEED;         //   重复速率计数器
```

```
    if (bStatus)                                        //  如果原先 KEY 是按下的
    {
        if (GPIOPinRead(KEY_PORT, KEY_PIN) == 0x00)     //  如果 KEY 仍为按下状态
        {
            if (usDelayCnt == 0)                        //  如果重复延时已经结束
            {
                if (--usSpeedCnt == 0)                  //  执行重复动作
                {
                    usSpeedCnt = KEY_SPEED;
                    KEY_Buf = 'K';
                }
            }
            else
            {
                usDelayCnt--;
            }
        }
        else                                            //  如果 KEY 已松开
        {
            bStatus = false;
        }
    }
    else                                                //  如果原先 KEY 是松开的
    {
        if (GPIOPinRead(KEY_PORT, KEY_PIN) == 0x00)     //  如果 KEY 按下
        {
            bStatus = true;
            KEY_Buf = 'K';
            usDelayCnt = KEY_DELAY;
            usSpeedCnt = KEY_SPEED;
        }
    }
}
```