



芯达STM32开发板

STM32 入门系列教程

Systick 的编程

Revision 0.01

(2010-04-25)

提到 systick 不得不抱怨以下 STM32 的用户手册，既然提供了 systick 的功能，为啥手册里却只提了一下？后来笔者上网搜的时候，才发现，抱怨的不仅仅是我一个人。闲话不说，下面将笔者搜集的资料进行总结。

1. systick 介绍

Systick 就是一个定时器而已，只是它放在了 NVIC 中，主要的目的是为了给操作系统提供一个硬件上的中断（号称滴答中断）。没有学过操作系统的同学，可能会很郁闷，啥叫滴答中断？这里来简单地解释一下。操作系统进行运转的时候，也会有“心跳”。它会根据“心跳”的节拍来工作，把整个时间段分成很多小小的时间片，每个任务每次只能运行一个“时间片”的时间长度就得退出给别的任务运行，这样可以确保任何一个任务都不会霸占整个系统不放。这个心跳，可以通过定时器来周期性触发，而这个定时器就是 systick。很明显，这个“心跳”是不允许任何人来随意地访问和修改的。只要不把它在 SysTick 控制及状态寄存器中的使能位清除，就永不停息。

知道 systick 在系统中的地位后，我们来了解 systick 的实现。注意，本期教程并没有讲述 systick 如何在操作系统中的运行，因为这对初学者来说比较复杂。我们这里只是举例说明 systick 的使用。它有四个寄存器，笔者把它列出来：

STK_CSR, 0xE000E010 -- 控制寄存器
STK_LOAD, 0xE000E014 -- 重载寄存器
STK_VAL, 0xE000E018 -- 当前值寄存器
STK_CALRB, 0xE000E01C -- 校准值寄存器

以下部分参考互联网的一篇文章，网址为：

<http://home.eeworld.com.cn/my/space.php?uid=116357&do=blog&id=31714>

感谢作者“416561760 的博客”提供如此详细的寄存器说明的文章。

1、STK_CSR 控制寄存器：寄存器内有 4 个位 t 具有意义

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															COUNT FLAG
															RW
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												CLKSOURCE	TICKINT	ENABLE	
												RW	RW	RW	

第 0 位：ENABLE，Systick 使能位 （0：关闭 Systick 功能；1：开启 Systick 功能）

第 1 位：TICKINT，Systick 中断使能位 （0：关闭 Systick 中断；1：开启 Systick 中断）

第 2 位：CLKSOURCE，Systick 时钟源选择 （0：使用 HCLK/8 作为 Systick 时钟；1：使用 HCLK 作为 Systick 时钟）

第 3 位：COUNTFLAG，Systick 计数比较标志，如果在上次读取本寄存器后，SysTick 已经数到了 0，则该位为 1。如果读取该位，该位将自动清零。

2、STK_LOAD 重载寄存器：

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								RELOAD[23:16]							
								r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RELOAD[15:0]															
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Systick 是一个递减的定时器，当定时器递减至 0 时，重载寄存器中的值就会被重装载，继续开始递减。STK_LOAD 重载寄存器是个 24 位的寄存器最大计数 0xFFFFF。

3、STK_VAL 当前值寄存器：

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								CURRENT[23:16]							
								r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CURRENT[15:0]															
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

也是个 24 位的寄存器，读取时返回当前倒计数的值，写它则使之清零，同时还会清除在 SysTick 控制及状态寄存器中的 COUNTFLAG 标志。

4、STK_CALRB 校准值寄存器：

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
NO REF	SKEW	Reserved						TENMS[23:16]							
								r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TENMS[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

位 31 NOREF : 1=没有外部参考时钟 (STCLK 不可用) 0=外部参考时钟可用

位 30 SKEW: 1=校准值不是准确的 1ms 0=校准值是准确的 1ms

位[23:0] : Calibration value

Indicates the calibration value when the SysTick counter runs on HCLK max/8 as external clock. The value is product dependent, please refer to the Product Reference Manual, SysTick Calibration Value section. When HCLK is programmed at the maximum frequency, the SysTick period is 1ms. If calibration information is not known, calculate the calibration value required from the frequency of the processor clock or external clock.

2. systick 编程

现在我们想通过 SysTick 定时器做一个精确的延迟函数，比如让 LED 精确延迟 1 秒钟闪亮一次。

思路：利用 systick 定时器为递减计数器，设定初值并使能它后，它会每个系统时钟周期计数器减 1，计数到 0 时，SysTick 计数器自动重装初值并继续计数，同时触发中断。

那么每次计数器减到 0，时间经过了：系统时钟周期 * 计数器初值。我们使用 72M 作为系统时钟，那么每次计数器减 1 所用的时间是 1/72M，计数器的初值如果是 72000，那么每次计数器减到 0，时间经过 $(1/72M) * 72000 = 0.001$ ，即 1ms。

现在我们做出来的 Delay(1)，就是 1 毫秒延迟。Delay(1000)就是 1 秒。

有了以上的思路后，systick 的编程非常简单。

首先，我们需要有一个 72M 的 systick 系统时钟，那么，使用下面这个时钟就 OK！

SystemInit();

这个函数可以让主频运行到 72M。可以把它作为 systick 的时钟源。

为了配合演示，可以使用 LED 显示来做，于是我们设置了 GPIO_Config(); 初始化函数，初始化了芯达 STM32 开发板上的 LED4 灯。

接着开始配置 systick，实际上配置 systick 的严格过程如下：

使用 ST 的函数库使用 systick 的方法：

- 1、调用 SysTick_CounterCmd() -- 失能 SysTick 计数器
- 2、调用 SysTick_ITConfig () -- 失能 SysTick 中断
- 3、调用 SysTick_CLKSourceConfig() -- 设置 SysTick 时钟源。
- 4、调用 SysTick_SetReload() -- 设置 SysTick 重装载值。

- 5、调用 SysTick_ITConfig () -- 使能 SysTick 中断
- 6、调用 SysTick_CounterCmd() -- 开启 SysTick 计数器

这里大家一定要注意，必须使得当前寄存器的值 VAL 等于 0！

SysTick->VAL = (0x00);

只有当 VAL 值为 0 时，计数器自动重载 RELOAD。

接下来就可以直接调用 Delay(); 函数进行延迟了。延迟函数的实现中，要注意的是，全局变量 TimingDelay 必须使用 volatile，否则可能会被编译器优化。

详细的例程请参考光盘。如果您对 systick 操作还有不明白的地方，请直接到我们的官方网站：ARM 技术交流网 www.arm79.com，进行讨论。我们将会尽快给您做出答复。

以下是一篇 **systick** 的问答篇总结，摘抄于网络，希望对您的理解有帮助。

文章网址：<http://blog.ednchina.com/jielove2003/768642/message.aspx>

Q: 什么是 SYSTick 定时器?

SysTick 是一个 24 位的倒计时定时器, 当计到 0 时, 将从 RELOAD 寄存器中自动重装载定时初值。只要不把它在 SysTick 控制及状态寄存器中的使能位清除, 就永不停息。

Q: 为什么要设置 SysTick 定时器?**(1) 产生操作系统的时钟节拍**

SysTick 定时器被捆绑在 NVIC 中, 用于产生 SYSTICK 异常 (异常号: 15)。在以前, 大多操作系统需要一个硬件定时器来产生操作系统需要的滴答中断, 作为整个系统的时基。因此, 需要一个定时器来产生周期性的中断, 而且最好还让用户程序不能随意访问它的寄存器, 以维持操作系统“心跳”的节律。

(2) 便于不同处理器之间程序移植。

Cortex - M3 处理器内部包含了一个简单的定时器。因为所有的 CM3 芯片都带有这个定时器, 软件在不同 CM3 器件间的移植工作得以化简。该定时器的时钟源可以是内部时钟 (FCLK, CM3 上的自由运行时钟), 或者是外部时钟 (CM3 处理器上的 STCLK 信号)。

不过, STCLK 的具体来源则由芯片设计者决定, 因此不同产品之间的时钟频率可能会大不相同, 你需要检视芯片的器件手册来决定选择什么作为时钟源。SysTick 定时器能产生中断, CM3 为它专门开出一个异常类型, 并且在向量表中有它的一席之地。它使操作系统和其它系统软件在 CM3 器件间的移植变得简单多了, 因为在所有 CM3 产品间对其处理都是相同的。

(3) 作为一个闹铃测量时间。

SysTick 定时器除了能服务于操作系统之外, 还能用于其它目的: 如作为一个闹铃, 用于测量时间等。要注意的是, 当处理器在调试期间被喊停 (halt) 时, 则 SysTick 定时器亦将暂停运作。

Q: Systick 如何运行?

首先设置计数器时钟源, CTRL->CLKSOURCE (控制寄存器)。设置重载值 (RELOAD 寄存器), 清空计数寄存器 VAL (就是下图的 CURRENT)。置 CTRL->ENABLE 位 开始计时。

如果是中断则允许 SysTick 中断, 在中断例程中处理。如采用查询模式则不断读取控制寄存器的 COUNTFLAG 标志位, 判断是否计时至零。或者采取下列一种方法

当 SysTick 定时器从 1 计到 0 时, 它将把 COUNTFLAG 位置位; 而下述方法可以清零之:

1. 读取 SysTick 控制及状态寄存器 (STCSR)
2. 往 SysTick 当前值寄存器 (STCVR) 中写任何数据

只有当 VAL 值为 0 时, 计数器自动重载 RELOAD。

Q: 如何使用 SysTicks 作为系统时钟?

SysTick 的最大使命，就是定期地产生异常请求，作为系统的时基。OS 都需要这种“滴答”来推动任务和时间管理。如欲使能 SysTick 异常，则把 STCSR.TICKINT 置位。另外，如果向量表被重定位到 SRAM 中，还需要为 SysTick 异常建立向量，提供其服务例程的入口地址。

Q: 如何使用 SysTick 完成一段延时?

查询方式 参考: <http://blog.ednchina.com/atom6037/188271/message.aspx>

中断方式 参考:

初始化函数 SysTick_Configuration(void)放在 while()循环外，执行一次:

[view plaincopy to clipboardprint?](#)

```
1.      void SysTick_Configuration(void)
2.      {
3.          /* Select AHB clock(HCLK) as SysTick clock source 设置 AHB 时钟为 SysTick 时
        钟*/
4.          SysTick_CLKSourceConfig(SysTick_CLKSource_HCLK);
5.
6.          /* Set SysTick Priority to 3 设置 SysTicks 中断抢占优先级 3，从优先级 0*/
7.          NVIC_SystemHandlerPriorityConfig(SystemHandler_SysTick, 3, 0);
8.
9.          /* SysTick interrupt each 1ms with HCLK equal to 72MHz 每 1ms 发生一次 SysTi
        ck 中断*/
10.         SysTick_SetReload(72000);
11.
12.         /* Enable the SysTick Interrupt */
13.         SysTick_ITConfig(ENABLE);
14.     }
```

延时函数，需要延时处调用:

[view plaincopy to clipboardprint?](#)

```
1.      void Delay(u32 nTime)
2.      {
3.          /* Enable the SysTick Counter 允许 SysTick 计数器*/
4.          SysTick_CounterCmd(SysTick_Counter_Enable);
5.
6.          TimingDelay = nTime;
7.
8.          while(TimingDelay != 0)
9.              ; //等待计数至 0
```

```
10.  
11.      /* Disable the SysTick Counter 禁止 SysTick 计数器*/  
12.      SysTick_CounterCmd(SysTick_Counter_Disable);  
13.      /* Clear the SysTick Counter 清零 SysTick 计数器*/  
14.      SysTick_CounterCmd(SysTick_Counter_Clear);  
15.  }
```

中断函数，定时器减至零时调用，放在 `stm32f10x_it.c` 文件中
[view plaincopy to clipboardprint?](#)

```
1.      void SysTickHandler(void)  
2.      {  
3.          TimingDelay--;  
4.      }
```

附：

福州芯达工作室简介

福州芯达工作室成立于 2009 年 9 月，我们专注于嵌入式产品的研发与推广，目前芯达产品涉及 ARM9 系列、STM32 系列。

芯达团队成员均硕士研究生毕业，具有一定研发实力。我们的愿景在于把福州芯达打造成国内一流的嵌入式品牌。或许我们现在做的还不够，但是我们真的努力在做，希望通过我们的努力，能够在您学习和使用芯达产品的过程中带来或多或少的帮助。

这是芯达为了配合 STM32 开发板而推出的入门系列教程。如果您在看了我们的教程后，理清了思路，我们都会倍感欣慰！让我们一起学习，共同进步，在征服嵌入式领域的道路上风雨同行！

官方网站：<http://www.arm79.com/>

官方淘宝：<http://shop36353570.taobao.com/>