用定时器控制 LED 闪烁

RT-Thread 评估板 RealTouch 裸机例程

版本号：1.0.0

日期：2012-08-06

修订记录

日期	作者	描述
2012-08-06	Heyuanjie87	创建文档

实验目的

闪灯程序在嵌入式学习中犹如“Hello World!”在C/C++语言学习中一样经典。它以简单的方式引导了无数的嵌入式爱好者。通过本节的学习你可以基本了解STM32的GPIO以及基本定时器的使用。

硬件说明

本例程需要一个定时器和一个LED，其中LED就是扩展板上的红色LED接在PD3上且正极接在高电平上,定时器选用基本定时器7。

1. STM32 GPIO 简介

- GPIO主要特性
  - 输出状态可选推挽、开漏、上拉或下拉
  - 可为每个I/O选择速度
  - 输入状态可选则悬空、上拉/下拉、模拟
  - 每个I/O引脚都有复用功能
  - 可对每个输出引脚进行位操作

STM32的每个GPIO都有4个32位配置寄存器：模式选择寄存器、输出类型配置寄存器、输出速度配置寄存器、上拉/下拉电阻配置寄存器；2个32位数据寄存器：数据输入寄存器、数据输出寄存器；1个32位锁定寄存器和2个32位复用功能选择寄存器。无论你选择某个I/O作为输入还是输出，都可以给根据需求选择是否使用上拉或下拉电阻。总的来讲，每个I/O有8中模式可供选择：输入悬空、带上拉输入、来下拉输入、带上拉或下拉开漏输出、带上拉或下拉推挽输出、模拟输入、推挽且带上拉或下拉的复用功能、开漏且带上拉或下拉的复用功能。

1.1 I/O 模式选择

每个 I/O 引脚都有 4 种用途模式可供选择。GPIOx_MODER(x = A..I)是一个 32 位寄存器，每两位配置一个引脚，位[1:0]配置引脚 0 以此类推。其取值及含义如表 1.1 所示。

表 1.1 I/O 用途模式设置

MODER[1:0	描述
B00	输入模式（初始值）
B01	通用输出模式
B10	复用功能模式
B11	模拟信号模式

1.2 输出类型选择

根据输出需求你可在 GPIOx_OTYPER 中设置推挽或开漏输出。这个寄存器只有低 16 位有效，取值及定义如表 1.2 所示。

表 1.2 输出类型设置

OT[0]	描述
B0	推挽输出（初始值）
B1	开漏输出

1.3 输出速度设置

表 1.3 端口输出速度设置

OSPEEDER[1： 0]	描述
B00	2MHz 低速
B01	25MHz 中速
B10	50MHz 快速

B11	100MHz 高速

1.4 上拉下拉电阻设置

表 1.4 上拉下拉电阻设置

PUPDR[1:0]	描述
B00	无上拉或下拉电阻
B01	上拉电阻连接
B10	下拉电阻连接
B11	保留

1.5 数据输入和输出

当 GPIO 设置为通用输入时，读取寄存器 GPI Ox_IDR)(x = A..I)可得到端口的输入状态且这个寄存器是只读的；GPIOx_ODR(x = A..I)是一个可读写寄存器，写数据到这个寄存器可控制端口输出电平，从这个寄存器读数据可判断端口的输出状态。

1.6 复用功能选择

STM32 中有 16 种复用功能，一个引脚会对应其中几种。共有两个寄存器 GPIOx_AFRL 与 GPIOx_AFRH 用来设置引脚的复用功能，其中每 4 位对应一个引脚。

表 1.6 复用功能配置

AFR[3:0]	描述
0x1	AF1(TIM1/TIM2)
0x2	AF2(TIM3…5)
0x4	AF4(I2C1…3)

0xD	AF14(DCMI)

2. STM32 基本定时器简介

STM32 的定时器非常强大，根据功能可分为高级控制定时器、通用定时器、基本定时，其中定时器 6、7 为基本定时器。在这里我们主要对基本定时器给予简单介绍。它具有一个 16 位自动重载加法计数器和一个 16 位可编程预分频器。预分频器对输入时钟进行分频，然后提供给计数器作为计数时钟使用。STM32 的自动重载寄存器（TIMx_ARR）在物理上对应两个寄存器，一个是咱们可以随便读写的，另一个则只能被定时器读取。这个咱们无法操作的寄存器被称作影子寄存器。当 TIMx_CR1 中的 ARPE 位等于 0 时改变 TIMx_ARR 的值就可马上改变影子寄存器里的值。当 ARPE 等于 1 时 TIMx_ARR 的值被缓存起来了，只有当更新事件发生后才会将新的值传送到影子寄存器中。

操作定会器前需要先打开输入时钟，然后可设置重预分频系数寄存器（TIMx_PSC）和自动载值寄存器。因为我们要让定时器产生更新中断，因此必需使能 TIMx_DIER 中的 UIE 位以及设置 NVIC 相关寄存器。初始化完成后设置 TIMx_CR1 的 CEN 位即可开启定时器。

实验原理及程序结构

实验设计

利用 STM32 的基本定时器产生更新中断，在中断处理函数中控制 LED 引脚的电平，带给大家一个闪灯的效果。本例程会涉及到 GPIO_DO3 以及基本定时器 7 的初始化和一个定时器 7 中断服务例程等。

源程序说明

下面让我们来看看 blink.c 是如何实现的。

LED 初始化

```
/* blink.c */
void led_init(void)
{
    GPIO_InitTypeDef  GPIO_InitStructure;

    /* 使能 GPIOD 时钟 */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);

    /* 配置 GPIO_LED 引脚 */
    GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_3;
    GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_NOPULL;

    GPIO_Init(GPIOD, &GPIO_InitStructure);
}
```

LED 的初始化比较简单，选择好引脚并将引脚用途模式设置为输出即可。需要注意的是 STM32 的每个外设的时钟都可单独控制，在操作前需要打开时钟而且要从 STM32 用户手册中确认清楚，你要用到的外设具体挂载在哪个总线上的（AHB1、APB1 等）。

LED 控制

```
/* blink.c */
void led_on(void)
{
    /* 设置 PD3 为低电平 */
    GPIO_ResetBits(GPIOD, GPIO_Pin_3);
}

void led_off(void)
{
```

```
        /* 设置 PD3 为高电平 */
    GPIO_SetBits(GPIOD, GPIO_Pin_3);
}

void led_toggle(void)
{
        /* 切换 PD3 电平状态 */
    GPIO_ToggleBits(GPIOD, GPIO_Pin_3);
}
```

这三个 LED 控制函数都使用了位操作，由于 LED 的正极是接在高电平上的，led_on 是让 PD3 输出低电平从而点亮 LED。led_toggle 将会在定时器的中断服务例程中调用，它会把引脚在高低电平之间切换达到闪灯的果。

定时器初始化

```
/* blink.c */
void TIM7_Configuration(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseInitStruct;

    /* 操作定时器前先使能输入时钟 */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM7, ENABLE);
    /**** Time base configuration ****/

    /* 设置自动重装载周期,计数到 5000 为 1000ms */
    TIM_TimeBaseInitStruct.TIM_Period = 5000;
    /* 设置预分频值,即定时器计数频率为 10Khz */
    TIM_TimeBaseInitStruct.TIM_Prescaler =(8400-1);
    TIM_TimeBaseInitStruct.TIM_ClockDivision = 0;
    /* 向上计数模式 */
    TIM_TimeBaseInitStruct.TIM_CounterMode = TIM_CounterMode_Up;

    /* 根据 TIM_TimeBaseInitStruct 中指定的参数初始化 TIM7 的时基单位 */
    TIM_TimeBaseInit(TIM7, &TIM_TimeBaseInitStruct);
```

```
    /* 使能 TIM7 更新中断 */
    TIM_ITConfig(TIM7, TIM_IT_Update, ENABLE);

    /* Configure the NVIC Preemption Priority Bits */
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
    /* Enable the TIM7 global Interrupt */
    NVIC_InitStructure.NVIC_IRQChannel = TIM7_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    /* 启动定时器 */
             TIM_Cmd(TIM7, ENABLE);
}
```

这里使用的是最简单的基本定时器，打开输入时钟、设置预分频器和计数重载值后配置好相关中断寄存器后就可启动定时器了。在 ST 库代码的 stm32f4xx_it.C 中定义了所有中断入口，我们需要在里面添加如下代码：

定时器 7 中断服务例程

```
/* stm32f4xx_it.C */
void TIM7_IRQHandler(void)
{
    if (TIM_GetITStatus(TIM7, TIM_IT_Update) != RESET)
    {
        /* 清除中断标志 */
        TIM_ClearITPendingBit(TIM7, TIM_IT_Update  );
       /* 切换 I/O 状态 */
        led_toggle();
     }
}
```

关于闪灯的驱动部分就是这些了，详细代码请参见示例工程。