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

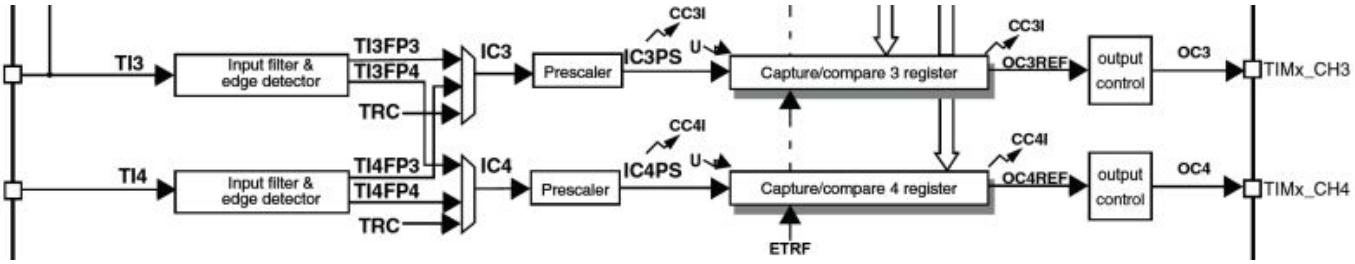
作者： 青风
出品论坛: www.qfv8.com
淘宝店：http://qfv5.taobao.com
QQ 技术群：241364123
硬件平台：QF-STM32F030 开发板

2.7 通用定时器 timer

2.7.1 原理分析

Stm32f0 系列通用定时器由一个 16 位或 32 位的自动装载计数器组成，它由一个可编程的预分频器驱动。它适合多种用途，包含测量输入信号的脉冲宽度(输入捕获)，或者产生输出波形(输出比较和 PWM)。学习 stm32f0 的核心就是认识可编程的预分频器驱动的设置。

本节介绍输出比较，通过 TM3 的比较寄存器 CCR3 和 CCR4 产生 2 路中断，通过定时器产生不同的中断时间。其结构如下图所示：



当我们设置系统时钟位 48MHZ，TIM3 的计数时钟速率为 6 MHz 时，那么预定标器可以按下面算式进行计算：

Prescaler = (TIM3CLK / TIM3 counter clock) - 1

Prescaler = (PCLK1 /6 MHz) - 1

那么比较寄存器 CC3 和 CC4 的更新率可以如下计算：

CC3 update rate = TIM3 counter clock / CCR3_Val = 439.4 Hz

CC4 update rate = TIM3 counter clock / CCR4_Val = 878.9 Hz

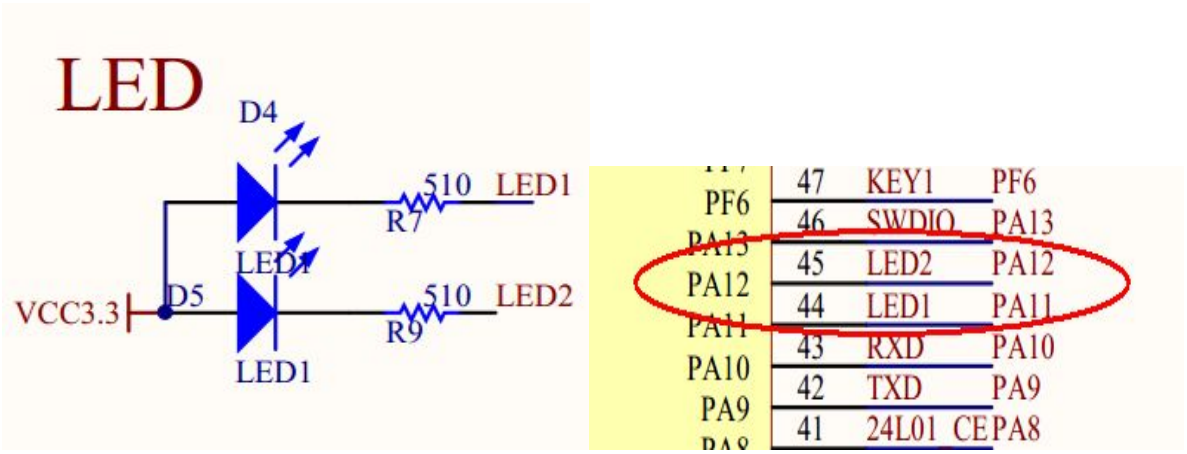
因此中断的产生反转频率为：

CC3 update rate/2 和 CC4 update rate/2

按照这个设计方案准备设计方案：

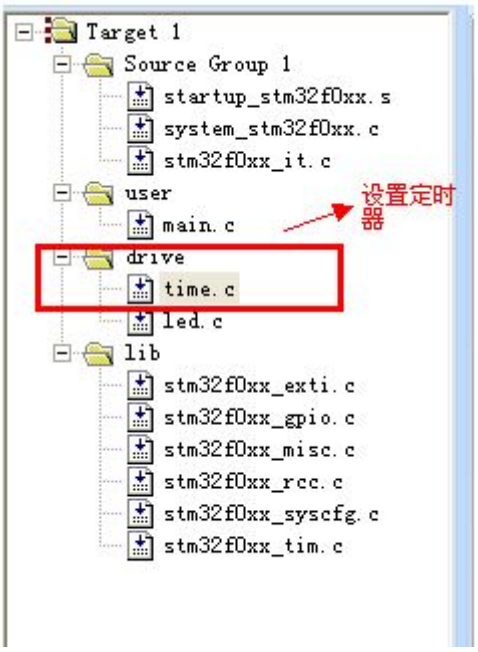
2.7.2 硬件准备：

只需要连接 2 个 LED 灯，就可以实现 TIM 定时了：



2.7.3 软件准备：

Time 定时器可以进行精确定时，并且通过 TIME 进行中断触发，在精确控制方面具有很好的优势。本实验采用了 TIM3 作为定时器，控制 2 路 LED 灯翻转。下面将从软硬件入手，分析如何通过 STM32F0 的定时器进行定时触发中断，从而控制 LED 灯的亮灭。



如上图所示，在 lib 库函数调用了 stm32f0xx.tim.c 函数库，我们在驱动函数 time.c 中编写定时器输出的相关参量设置。而中断执行函数则在 stm32f0xx_it.c 函数中进行编写；

配置 TIM3 的定时中断我们可以分成两个部分完成：

第一步：首先是配置 TIM 的中断嵌套，代码如下所示：

```
01.  void TIM_INT_Config(void)
02. {
03.     NVIC_InitTypeDef NVIC_InitStructure;
04.   /* TIM3 时钟使能 */
```

```
05.    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
06.
07.    /*  TIM3 中断嵌套设计*/
08.    NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;
09.    NVIC_InitStructure.NVIC_IRQChannelPriority = 0;
10.    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
11.    NVIC_Init(&NVIC_InitStructure);
12. }
```

上面函数中，设置中断通道为 TIM3 中断，频道优先级设为 0，并且使能频道。这样就配置好了 TIM3 中断嵌套。当然中断要执行的操作要在 stm32f0xx_it.c 进行编写，这个等下我们再讲，我们先把 TIM3 的参数配置进行讨论：

首先来看看 TIM 定时器的基础配置参数，这个参数的配置要求在文件 stm32f0xx_tim.c 中进行了描述，通过如下的结构体单元进行了归纳：

```
13.    typedef struct
14. {
15.    uint16_t TIM_Prescaler;          /*!指定用来划分 TIM 时钟预分频值*/
16.    uint16_t TIM_CounterMode;        /*!指定的计数器模式*/
17.    uint32_t TIM_Period;             /*设置时钟周期   */
18.    uint16_t TIM_ClockDivision;      /*设定时钟分频  */
19.    uint8_t TIM_RepetitionCounter;   /*指定重复计数器值  */
20. } TIM_TimeBaseInitTypeDef;
```

上面的结构体参数就是设置 TIME 的基础参数，下面我们就来确定这几个参数的设置：

```
21.      /* Time 定时器基础设置  */
22.    TIM_TimeBaseStructure.TIM_Period = 65535;
23.    TIM_TimeBaseStructure.TIM_Prescaler = 0;
24.    TIM_TimeBaseStructure.TIM_ClockDivision = 0;
25.    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
26.    TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);
```

TIM_Prescaler 设置预分频为 0，也就是不预分频。那么系统时钟我们设置为 48MHZ，那么 TIM 定时器也跑在了 48MHZ。而 TIM_ClockDivision 我们设为 0，也就是不进行时钟分频。TIM _CounterMode 设置为向上计数。TIM 计数时钟为 6MHZ，那么翻转率按照下面公式继续计算

CC3 翻转率 = TIM3 counter clock / CCR3_Val

CC4 翻转率= TIM3 counter clock / CCR4_Val

配置完基础配置后，CC3 和 CC4 的翻转要通过输入捕获实现定时器的翻转，而定时器输入比较模式通过下面的结构体进行配置：

```
27.    typedef struct
28. {
29.    uint16_t TIM_OCMode;          /*!指定的 TIM 模式  */
30.    uint16_t TIM_OutputState;     /*指定的 TIM 输出比较状态  */
31.    uint16_t TIM_OutputNState;    /*指定 TIM 互补的输出比较状态. */
32.    uint32_t TIM_Pulse;           /*指定的脉冲值被装入到捕获比较寄存器*/
33.    uint16_t TIM_OCPolarity;      /*指定的脉冲值被装入到捕捉比较寄存器  */
```

```
34.    uint16_t TIM_OCNPolarity;     /*指定的互补输出极性 */
35.    uint16_t TIM_OCIdleState;     /*指定在空闲状态下的 TIM 输出比较引脚的状态 */
36.    uint16_t TIM_OCNIdleState;    /*指定在空闲状态下的互补 TIM 输出比较引脚的状态. */
37. } TIM_OCInitTypeDef;
```

对上面的产生配置如下代码：

```
38.   /* 输出比较时序模式配置设置 */
39.    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Timing;
40.    TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
41.    /* 输出比较时序模式配置: 频道 3*/
42.    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
43.    TIM_OCInitStructure.TIM_Pulse = CCR3_Val;
44.    TIM_OC3Init(TIM3, &TIM_OCInitStructure);
45.    TIM_OC3PreloadConfig(TIM3, TIM_OCPreload_Disable);
46.    /* 输出比较时序模式配置: 频道 4 */
47.    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
48.    TIM_OCInitStructure.TIM_Pulse = CCR4_Val;
49.    TIM_OC4Init(TIM3, &TIM_OCInitStructure);
50.    TIM_OC4PreloadConfig(TIM3, TIM_OCPreload_Disable);
51.    /* TIM 中断使能 */
52.    TIM_ITConfig(TIM3, TIM_IT_CC3 | TIM_IT_CC4, ENABLE);
53.    /* TIM3 使能 */
54.    TIM_Cmd(TIM3, ENABLE);
```

那么中断我们则执行 LED 是翻转工作：

```
55.
56. void TIM3_IRQHandler(void)
57. {
58.    if (TIM_GetITStatus(TIM3, TIM_IT_CC3) != RESET)
59.    {
60.      TIM_ClearITPendingBit(TIM3, TIM_IT_CC3);
61.
62.      /* LED3 toggling with frequency = 219.7 Hz */
63.      LED1_Toggle();
64.      capture = TIM_GetCapture3(TIM3);
65.      TIM_SetCompare3(TIM3, capture + CCR3_Val);
66.    }
67.    else
68.    {
69.      TIM_ClearITPendingBit(TIM3, TIM_IT_CC4);
70.
71.      /* LED4 toggling with frequency = 439.4 Hz */
72.      LED2_Toggle();
73.      capture = TIM_GetCapture4(TIM3);
74.      TIM_SetCompare4(TIM3, capture + CCR4_Val);
```

```
75.     }
76. }
```

主函数的编写就较为简单了，直接调用子函数输出：

```
77.     int main(void)
78. {
79.                     LED_Init();
80.                     LED1_Open();
81.                     LED2_Open();
82.                     TIM_INT_Config();
83.                     TIM_OUT_Config();
84.                       /*无限循环  */
85.     while (1)
86.     {
87.     }
88. }
89.
```

实验现象：

两个 led 灯按照不同速率进行翻转