青风手把手教你学 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

2.8 定时器输出 PWM 波

2.8.1 原理分析：

假设单片机内部没有硬件 PWM 模块，我们可以采用 Time 定时器输出 PWM 波，通过定时器的比较模式，设定预装载值，可以设计输出不同频率的 PWM 波。并且我们设置不同的翻转量，则可以设置不同的 PWM 占空比，这些运用在驱动电机或者一些相关运用中非常有用。本实验我们采用 TIM1 来产生四路频率相同的，但是占空比不同的 PWM 波。 下面将从软硬件入手，分析如何通过 STM32F0 的定时器输出 PWM 波。首先是硬件方面：

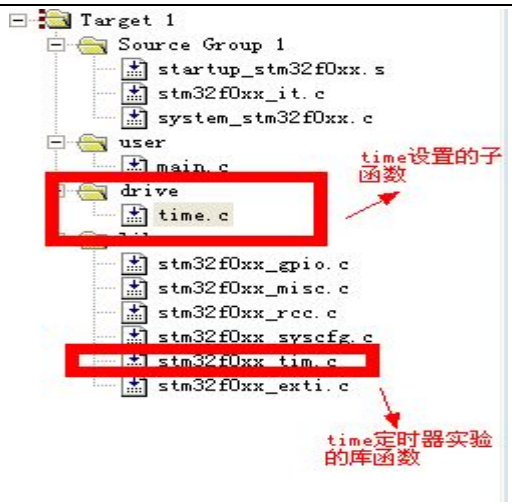
2.8.2 硬件准备：

保证输出 IO 端口如下就可以：

- TIM1_CH1 pin (PA.08)
- TIM1_CH2 pin (PA.09)
- TIM1_CH3 pin (PA.10)
- TIM1_CH4 pin (PA.11)

2.8.3 软件准备：

打开 keil 编译环境，设置系统工程树如图所示：



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。

配置 PWM 波形输出的设置我们分成两个部分完成：

第一步：首先是输出管脚的 IO 口设置，PWM 输出，自然会采用到 IO 口作为输出端口，在 STM32F0 系列中，IO 端口可以复用为 TIM 定时器输出通道，如下表所示：

Pin name	AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7
PA0		USART2_CTS	TIM2_CH1_ETR	TSC_G1_IO1				COMP1_OUT
PA1	EVENTOUT	USART2_RTS	TIM2_CH2	TSC_G1_IO2				
PA2	TIM15_CH1	USART2_TX	TIM2_CH3	TSC_G1_IO3				COMP2_OUT
PA3	TIM15_CH2	USART2_RX	TIM2_CH4	TSC_G1_IO4				

我们采用了 PA0,,PA1，PA2，PA3 的复用功能 AF2 做为 TIM 定时器的 4 路输出通道。那么配置 IO 复用的代码如下：

```
01.  void TIM_Config(void)
02. {
03.   GPIO_InitTypeDef GPIO_InitStructure;
04.
05.   /* 使能 GPIO 时钟 */
06.   RCC_AHBPeriphClockCmd( RCC_AHBPeriph_GPIOA, ENABLE);
07.
08.   /* 配置 GPIO 管脚参数设置*/
09.   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10 |
    GPIO_Pin_11;
10.   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
11.   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
12.   GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
13.   GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP ;
14.   GPIO_Init(GPIOA, &GPIO_InitStructure);
15.
16.   GPIO_PinAFConfig(GPIOA, GPIO_PinSource8, GPIO_AF_2); /* GPIO 管脚复用设置*/
17.   GPIO_PinAFConfig(GPIOA, GPIO_PinSource9, GPIO_AF_2);
18.   GPIO_PinAFConfig(GPIOA, GPIO_PinSource10, GPIO_AF_2);
```

```
19.    GPIO_PinAFConfig(GPIOA, GPIO_PinSource11, GPIO_AF_2);
20. }
```

完成了这一步，也就打开了 PWM 输出的通道。

第二步，设置定时器的参数，配置出频率为 17.57 KHz 的 PWM 波，并且输出四路的占空比分别为：50%，37.5%，25%，12.5%.下面来讨论下如何设置：

首先考虑 time 定时器的时钟频率。如果我们设置分频数为 0，也就是说 time 定时器等于系统时钟，system_stm32f0xx.c 中已经把系统频率设置在 48MHZ，在 startup_stm32f0xx.s 中，首先运行了 systemInit 函数，因此可以确定 time 定时器运行在 48MHZ。

```
IMPORT  SystemInit
        LDR     R0, =SystemInit
        BLX     R0
        LDR     R0, =__main
        BX      R0
        ENDP
```

定时器产生的 PWM 的频率可以按照下面的公式进行计算：

预定标的值 TIM1_Period = (time 定时器频率 / pwm 的频率) - 1

预定标的值实际上就是定时器运行多少次算一个 PWM 周期，这个在设置 pwm 频率中重要的参数。这个参数在结构体中设置。在程序中调用如下：

```
21.    TIM_TimeBaseStructure.TIM_Period = TimerPeriod;
```

这个 TIM_TimeBaseStructure 是定时器的基础设置参数，在 stm32f0xx_tim.h 中通过结构体给出：

```
22.    typedef struct
23. {
24.    uint16_t TIM_Prescaler;            /*!指定用来划分 TIM 时钟预分频值*/
25.    uint16_t TIM_CounterMode;          /*!指定的计数器模式*/
26.    uint32_t TIM_Period;               /*设置时钟周期   */
27.    uint16_t TIM_ClockDivision;        /*设定时钟分频  */
28.    uint8_t TIM_RepetitionCounter;     /*指定重复计数器值  */
29. } TIM_TimeBaseInitTypeDef;
```

上面的结构体参数是设置 TIME 的基础参数，但是输出 PWM 波的占空比我们采用了比较捕获模式，设置在什么情况下发生跳转，还需要使用结构体 **TIM_OCInitTypeDef**，如下面所示：

```
30.    typedef struct
31. {
32.    uint16_t TIM_OCMode;           /*!指定的 TIM 模式  */
33.    uint16_t TIM_OutputState;      /*指定的 TIM 输出比较状态  */
34.    uint16_t TIM_OutputNState;     /*指定 TIM 互补的输出比较状态. */
35.    uint32_t TIM_Pulse;            /*指定的脉冲值被装入到捕获比较寄存器*/
36.    uint16_t TIM_OCPolarity;       /*指定的脉冲值被装入到捕捉比较寄存器  */
37.    uint16_t TIM_OCNPolarity;      /*指定的互补输出极性  */
```

```
38.    uint16_t TIM_OCIdleState;     /*指定在空闲状态下的 TIM 输出比较引脚的状态 */
39.    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. */
40. } TIM_OCInitTypeDef;
```

其中 TIM_Pulse 装载比较寄存器，判断什么时候发生 PWM 翻转 ：

```
41.  TIM_OCInitStructure.TIM_Pulse = Channel1Pulse;
```

Channel1Pulse 的值可以按照下面的公式进行计算：

ChannelxPulse = DutyCycle * (TIM1_Period - 1) / 100

其中 DutyCycle/100 为占空比的值，TIM1_Period 就是我们前面定义的预定标的值。那么四路 PWM 的装载值可以设置为：

```
42.     /*计算预定标 的值，也就是多少个时钟计数为一个周期*/
43.    TimerPeriod = (SystemCoreClock / 17570 ) - 1;
44.    /*计算 CCR1 跳转值 在占空比为 50%时*/
45.    Channel1Pulse = (uint16_t) (((uint32_t) 5 * (TimerPeriod - 1)) / 10);
46.  /*计算 CCR2 跳转值 在占空比为 37.5%时*/
47.    Channel2Pulse = (uint16_t) (((uint32_t) 375 * (TimerPeriod - 1)) / 1000);
48.    /*计算 CCR3 跳转值 在占空比为 25%时*/
49.    Channel3Pulse = (uint16_t) (((uint32_t) 25 * (TimerPeriod - 1)) / 100);
50.  /*计算 CCR4 跳转值 在占空比为 12.5%时*/
51.    Channel4Pulse = (uint16_t) (((uint32_t) 125 * (TimerPeriod- 1)) / 1000);
```

各个数值指标设置好后，我们就按照结构体定义的参数来配置 PWM 的参数，整体的设置函数如下所示：

```
52.  /* TIM1 时钟使能 */
53.    RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1 , ENABLE);
54.    /* Time 定时基础设置*/
55.    TIM_TimeBaseStructure.TIM_Prescaler = 0;
56.    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;   /* Time 定时设置
     为上升沿计算模式*/
57.    TIM_TimeBaseStructure.TIM_Period = TimerPeriod;
58.    TIM_TimeBaseStructure.TIM_ClockDivision = 0;
59.    TIM_TimeBaseStructure.TIM_RepetitionCounter = 0;
60.    TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure);
61.
62.    /* 频道 1，2，3，4 的 PWM 模式设置 */
63.    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
64.    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
65.    TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Enable;
66.    TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
67.    TIM_OCInitStructure.TIM_OCNPolarity = TIM_OCNPolarity_High;
68.    TIM_OCInitStructure.TIM_OCIdleState = TIM_OCIdleState_Set;
69.    TIM_OCInitStructure.TIM_OCNIdleState = TIM_OCIdleState_Reset;
70.
71.    TIM_OCInitStructure.TIM_Pulse = Channel1Pulse;//使能频道 1 配置
72.    TIM_OC1Init(TIM1, &TIM_OCInitStructure);
```

```
73.
74.     TIM_OCInitStructure.TIM_Pulse = Channel2Pulse;//使能频道 2 配置
75.     TIM_OC2Init(TIM1, &TIM_OCInitStructure);
76.
77.     TIM_OCInitStructure.TIM_Pulse = Channel3Pulse;//使能频道 3 配置
78.     TIM_OC3Init(TIM1, &TIM_OCInitStructure);
79.
80.     TIM_OCInitStructure.TIM_Pulse = Channel4Pulse;//使能频道 4 配置
81.     TIM_OC4Init(TIM1, &TIM_OCInitStructure);
82.
83.     /* TIM1 计算器使能*/
84.     TIM_Cmd(TIM1, ENABLE);
85.
86.     /* TIM1 主输出使能 */
87.     TIM_CtrlPWMOutputs(TIM1, ENABLE);
```

主函数的编写就较为简单了，直接调用子函数输出：

```
88.     int main(void)
89. {
90.     TIM_Config();
91.     TIM_PWM_Config();
92.     while (1)
93.     {}
94. }
```