定时器原理及应用

8.1 STM32芯片定时概述

STM32F103RB系列芯片有多达7个定时器：3个普通16位定时器，每个定时器有多达4个用于输入捕获/输出比较/PWM或脉冲计数的通道和增量编码器输入，1个高级16位定时器，用于电机控制的PWM高级控制定时器，2个看门狗定时器，1个系统滴答定时器，24位自减型计数器。

通用定时器（TIMx）

具有3个通用定时器（TIM2、TIM3、TIM4）。特点：

- 可同步运行。
- 通过定时器链接功能与高级定时器共同工作。
- 调试模式下，定时器可被冻结。
- 任一标准定时器都能用于产生PWM输出。
- 每个定时器有独立的DMA请求机制。

通用定时器具有16位向上、向下、向上/向下自动装载定时器，以及16位可编程预分频器，分频系数可以为1 ~ 65536之间的任意数值。

高级控制定时器（TIM1）

可以看成是分配到6个通道的三相PWM发生器，还可以被当成完整的通用定时器。

基本特点与通用定时器相同。

看门狗定时器

看门狗定时器（WDT）实际上是一个计数器，一般给看门狗一个数字，程序开始运行后看门狗开始倒计数。如果程序运行正常，过一段时间CPU应发出指令让看门狗复位，重新开始倒计数。如果看门狗减到0就认为程序没有正常工作，强制整个系统复位。STM32有2种类型的看门狗：独立看门狗和窗口看门狗。

8.2 高级/通用定时器

8.2.2 主要库函数说明

TIM_TimeBaseInit()

void TIM_TimeBaseInit(TIM_TypeDef *TIMx, TIM_TimeBaseInitTypeDef *TIM_TimeBaseInitSturct)

此函数对TIMx的时间基数单位参数初始化。

TIM_TimeBaseInitTypeDef结构体说明：

```
typedef struct
{
    // 设置在下一个更新时间装入活动的自动重装载寄存器周期的值，范围为0x0000 - 0xFFFF
    u16 TIM_Period;
    // 设置了用来作为TIMx时钟频率除数的预分频值，范围为0x0000 - 0xFFFF
    u16 TIM_Perscaler;
    u16 TIM_ClockDivision;      // 时钟分割
    u16 TIM_CounterMode;        // 计数器模式
    u16 TIM_RepetitionCounter;  // 仅高级定时器有用
} TIM_TimeBaseInitTypeDef;
```

TIM_OCxInit()

void TIM_OCxInit(TIM_TypeDef *TIMx, TIM_OCInitTypeDef *TIM_OCInitStruct)

其中x = 1 ~ 4，此函数对TIM的输出通道x外设参数初始化。

TIM_OCInitTypeDef结构体说明：

```
typedef struct
{
    u16 TIM_OCMode;         // 选择定时器模式
    u16 TIM_OutputState;
    u16 TIM_OutputNState;
    // 设置了待装入捕获比较寄存器的脉冲值，范围为0x0000 - 0xFFFF
    u16 TIM_Pulse;
    u16 TIM_OCPolarity;     // 输出极性
    u16 TIM_OCNPolarity;
    u16 TIM_OCIdleState;
    u16 TIM_OCNIdleState;
} TIM_OCInitTypeDef;
```

TIM_ICInit()

void TIM_ICInit(TIM_TypeDef *TIMx, TIM_ICInitTypeDef *TIM_ICInitStruct)

此函数对TIM的输入通道x外设参数初始化。

TIM_ICInitTypeDef结构体说明

```
typedef struct
{
    u16 TIM_Channel;        // 选择通道
    u16 TIM_ICPolarity;     // 输入活动边沿
    u16 TIM_ICSelection;    // 选择输入
    u16 TIM_ICPrescaler;    // 输入捕获预分频器
    u16 TIM_ICFilter;       // 输入比较滤波器
} TIM_ICInitTypeDef;
```

TIM_TIConfig()

void TIM_ITConfig(TIM_TypeDef *TIMx, u16 TIM_IT, FunctionalState NewState)

此函数配置中断。

TIM_ETRClockMode1Config()

void TIM_ETRClockMode1Config(TIM_TypeDef *TIMx, u16 TIM_ExtTRGPrescaler, u16 TIM_ExtTRGPolarity, u16 ExtTRGFilter)

TIM_ExtTRGPrescaler设置外部出发预分频。

TIM_ExtTRGPolarity设置TIMx外部触发极性。

8.2.3 定时器计数器基本说明

时钟源

定时器时钟主要由下述时钟源提供：

1. 内部时钟源（CK_INT）
   内部时钟源作为时钟时，定时器的时钟是来自于输入为AP1或APB2的一个倍频器。
   预分频系数与周期计算（确定TIM_Prescaler和TIM_Period的值）：
   简单来说，定时器的周期为（中断、更新周期）：
   (TIM_Prescaler + 1) * (TIM_Period + 1) / 设备频率
   如，STM32的时钟频率为72MHz，令：

   ```
   TIM_Prescaler = 7200 - 1
   TIM_Period = 10000 - 1
   ```

   则定时器中断、更新周期为：7200 * 10000 / (72 * 10^6) = 1s
2. 外部时钟源
   包括外部时钟模式1：外部输入脚TIx和外部时钟模式2：外部触发输入（ETR）。
3. 内部触发输入（ITRx）
   使用一个定时器作为另一个定时器的预分频器，可以配置一个定时器Timer1作为另一个定时器Timer2的预分频器。

计数模式

1. 向上计数模式：计数器从0计数到自动加载值（TIMx_ARR计数器的值），然后重新从0开始计数并产生一个计数器溢出事件。
2. 向下计数模式：计数器从自动装入的值并开始向下计数到0，然后从自动装入的值重新开始并且产生一个计数器向下溢出事件。
3. 中央对齐模式（向上/向下计数）：在中央对齐模式，计数器从0开始计数到自动加载的值（TIMx_ARR寄存器）-1，产生一个计数器溢出时间，然后向下计数到1并且产生一个计数器下溢事件，然后再从0开始重新计数。

典型的定时器应用

定时器基本功能

```
/*
    通过定时器实现1s的延迟，控制LED1灯每秒闪烁一次。
*/

// 配置TIM1时钟模块使能

RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);

// 初始化定时器

void TIM1_Configuration()
{
    TIM_TimeBaseInitTypeDef tim_init;
    TIM_TimeBaseStructInit(&tim_init);
    tim_init.TIM_Period = 7200 - 1;
    tim_init.TIM_Prescaler = 10000 - 1;
    tim_init.TIM_ClockDivision = TIM_CKD_DIV1;
    tim_init.TIM_CounertMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit(&tim_init);
    TIM_Cmd(TIM1, ENABLE);
}

// 配置定时器中断

void nvic_init()
{
    NVIC_InitTypeDef nvic_init;
    TIM_ITConfig(TIM1, TIM_IT_Update, ENABLE);  // 使能TIM1更新中断
    nvic_init.NVIC_IRQChannel = TIM1_UP_IRQn;
    nvic_init.NVIC_IRQChannelCmd = ENABLE;
    nvic_init.NVIC_IRQChannelPeermptionPriority = 0;
    nvic_init.NVIC_IRQChannelSubPriority = 0;
    NVIC_Init(&nvic_init);
}

void TIM1_UP_IRQHandler()
{
    if(TIM_GetITStatus(TIM1, TIM_IT_Update))
        led_twinkle();

    TIM_ClearITPendingBit(TIM1, TIM_IT_Update);
}
```

计数器功能

```
/*
    实现连接到PA0引脚（TIM2外部输入通道1）外部脉冲进行计数。
*/

// TIM2模块时钟使能

RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

// 初始化计数器
void TIM2_Configuration()
{
    TIM_TimeBaseInitTypeDef tim_init;
    tim_init.TIM_Period = 0xFFFFFF;
    tim_init.TIM_Prescaler = 0;
    tim_init.TIM_ClockDivision = TIM_CKD_DIV1;
    tim_init.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM2, &tim_init);
    TIM_ETRClockMode1Config(TIM2, TIM_ExtTRGPrescaler_OFF, TIM_ExtTRGPolarity_NonInverted, 0);
    TIM_Cmd(TIM2, ENABLE);
}

// 初始化中断

void nvic_init()
{
    NVIC_InitTypeDef nvic_init;
    TIM_ITConfig(TIM2, TIM_IT_Trigger, ENABLE);
    nvic_init.NVIC_IRQChannel = TIM2_IRQn;
    nivc_init.NVIC_IRQChannelCmd = ENABLE;
    nvic_init.NVIC_IRQChannelPreemptionPriority = 0;
    nvic_init.NVIC_IRQChannelSubPriority = 0;
    NVIC_Init(&nvic_init);
}

void TIM2_IRQHandler()
{
    if(TIM_GetITStatus(TIM2, TIM_IT_Trigger))
        usart_send_byte(TIM_GetCounter(TIM2));

    TIM_ClearITPendingBit(TIM2, TIM_IT_Trigger);
}
```

捕获模式功能

输入捕获模式可以用来测量脉冲宽度或者测量频率。以一个具体事例来进行说明：对按钮按下去的时间长度进行测量：

```
/*
    实现对按钮按下去的时间长度进行测量。
*/

// 使能TIM2时钟模块
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

// 初始化TIM2

void TIM2_Configuration()
{
    // 设置每1us计时一次
    TIM_TimeBaseInitTypeDef tim_init;
    tim_init.TIM_Period = 72 - 1;
    tim_init.TIM_Prescaler = 0;
    tim_init.TIM_ClockDivision = TIM_CKD_DIV1;
    tim_init.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM2, &tim_init);

    TIM_ICInitTypeDef tim_ic_init;
    tim_ic_init.TIM_Channel = TIM_Channel_1;
    tim_ic_init.TIM_ICFilter = 0;
    tim_ic_init.TIM_ICPolarity = TIM_ICPolarity_Falling;
    tim_ic_init.TIM_ICPrescaler = TIM_ICPSC_DIV1;
    tim_ic_init.TIM_ICSelection = TIM_ICSelection_DirectT1;
    TIM_ICInit(TIM2, &tim_ic_init);

    NVIC_InitTypeDef nvic_init;
    TIM_ITConfig(TIM2, TIM_IT_Update|TIM_IT_CC1, ENABLE);
    nvic_init.NVIC_IRQChannel = TIM2_IRQn;
    nvic_init.NVIC_IRQChannelCmd = ENABLE;
    nvic_init.NVIC_IRQChannelPeermptionPriority = 0;
    nvic_init.NVIC_IRQChannelSubPriority = 0;
    NVIC_Init(&nvic_init);
    TIM_Cmd(TIM2, ENBALE);
}

extern struct capture_st m_capture;

void TIM2_IRQHandler()
{
    if(TIM_GetITStatus(TIM2, TIM_IT_Update))
    {
        if(m_capture.start == 1)
            m_capture.period_cnt++;
        TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
    }
    if(TIM_GetITStatus(TIM2, TIM_IT_CC1))
    {
        if(m_capture.start == 0)
        {
            m_capture.start = 1;
            m_capture.ccr_value = 0;
            m_capture.period_cnt = 0;
            TIM_OC1PolarityConfig(TIM2, TIM_ICPolarity_Rising);
```

```
            TIM_OC1PolarityConfig(TIM2, TIM_ICPolarity_Rising);
            TIM_SetCounter(TIM2, 0);
        }
        else{
            TIM_ITConfig(TIM2, TIM_IT_CC|TIM_IT_Update, DISABLE);
            m_capture.finish = 1;
            m_capture.start = 0;
            m_capture.ccr_value = TIM2->CCR2;
        }
        TIM_ClearITPendingBit(TIM2, TIM_IT_CC1);
    }
}

int main()
{
    usart_init();
    key_init();
    TIM2_Configuration();
    while(1)
    {
        if(m_capture.finish)
        {
            float us = (m_capture.ccr_value + 1) + m_capture.period_cnt * (0xFFFFF + 1);
            printf("%f\n", us/1e6);
            m_capture.finish = 0;
            TIM_ITConfig(TIM2, TIM_IT_CC1|TIM_IT_Update, ENABLE);
        }
    }
    return 0;
}
```

输出PWM模式

```
/*
    利用PWM输出实现呼吸灯效果。
*/

// 初始化时钟模块使能

RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

// 初始化定时器

void TIM2_Configuration()
{
    TIM_TimeBaseInitTypeDef tim_init;
    TIM_OCInitTypeDef tim_oc_init;
    tim_init.TIM_Period = 1024 - 1;
    tim_init.TIM_Prescaler = 200 - 1;
    TIM_TimeBaseInit(TIM2, &tim_init);

    tim_oc_init.TIM_OCMode = TIM_OCMode_PWM1;
    tim_oc_init.TIM_OCPolarity = TIM_OCPolarity_Low;  // 低电平点亮
    tim_oc_init.TIM_OutputState = TIM_OutputState_Enable;
    tim_oc_init.TIM_Pulse = 0;
    TIM_OC2Init(TIM2, &tim_oc_init);

    NVIC_InitTypeDef nvic_init;
    TIM_ClearFlag(TIM2, TIM_IT_Update);
    TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
    nvic_init.NVIC_IRQChannel = TIM2_IRQn;
    nvic_init.NVIC_IRQChannelCmd = ENABLE;
    nvic_init.NVIC_IRQChannelPreemptionPriority = 0;
    nvic_init.NVIC_IRQChannelSubPriority = 0;
    NVIC_Init(&nvic_init);
    TIM_Cmd(TIM2, ENABLE);
}

void TIM2_IRQHandler()
{
    if(TIM_GetITStatus(TIM2, TIM_IT_Update) == SET)
    {
        if(!--period_class)
        {
            period_class = 10;
            if(++data_index >= 110)
                data_index = 0;
            TIM_SetCompare2(TIM2, data[data_index]);
        }
        TIM_ClearITPendingBit(TIM2, TIM_IT_Update);

    }
}
```

PWM输入捕获模式

```
/*
    通过PA6引脚作为输出产生PWM，然后PA8对PA6产生的PWM波形进行测量。
*/

// GPIO配置

void gpio_init()
{
    GPIO_InitTypeDef gpio_init;
    gpio_init.GPIO_Pin = GPIO_Pin_6;
    gpio_init.GPIO_Mode = GPIO_Mode_AF_PP;
    gpio_init.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &gpio_init);
    gpio_init.GPIO_Pin = GPIO_Pin_8;
    gpio_init.GPIO_Mode = GPIO_Mode_IPD;
    GPIO_Init(GPIOA, &gpio_init);
}

// 定时器配置

void TIM_Configuration()
{
    TIM_TimeBaseInitTypeDef tim_init;
    TIM_ICInitTypeDef tim_ic_init;
    TIM_OCInitTypeDef tim_oc_init;

    // 配置TIM3通道1产生PWM波形
    tim_init.TIM_Period = 8 - 1;
    tim_init.TIM_Prescaler = 72 - 1;
    tim_init.TIM_ClockDivision = TIM_CKD_DIV1;
    tim_init.TIM_CounterMode = TIM_CounterMode_Up;
    tim_init.TIM_RepetitionCounter = 0;
    TIM_TimeBaseInit(TIM3, &tim_init);
    tim_oc_init.TIM_OCIdleStatel = TIM_OCIdleState_Reset;
    tim_oc_init.TIM_OCMode = TIM_OCMode_PWM1;
    tim_oc_init.TIM_OCPolarity = TIM_OCPolarity_High;
    tim_oc_init.TIM_OutputState = TIM_OutputState_Enable;
    tim_oc_init.TIM_OutputNState = TIM_OutputNState_Disable;
    tim_oc_init.TIM_Pulse = 4;
    TIM_OC1Init(TIM3, &tim_oc_init);
    TIM_Cmd(TIM3, ENABLE);

    // 配置TIM1对PWM波形进行测量
    TIM_TimBaseInit(TIM1, &tim_init);
    tim_ic_init.TIM_Channel = TIM_Channel_1;
    tim_ic_init.TIM_ICFilter = 0;
    tim_ic_init.TIM_ICPolarity = TIM_ICPolarity_Rising;
    tim_ic_init.TIM_ICPrescaler = TIM_ICPSC_DIV1;
    tim_ic_init.TIM_ICSelection = TIM_ICSelection_DirectTI;
    TIM_PWMIConfig(TIM1, &tim_ic_init);

    TIM_SelectInputTrigger(TIM1, TIM_TS_TI1FP1);
```

```
    TIM_SelectSlaveMode(TIM1, TIM_SlaveMode_Enable);
    TIM_SelectMasterSlaveMode(TIM1, TIM_MasterSlaveMode_Enable);
    TIM_ClearFlag(TIM1, TIM_FLAG_CC1);
    nvic_config();
    TIM_ITConfig(TIM1, TIM_IT_CC1, ENABLE);

    TIM_Cmd(TIM1, ENABLE);
}

void nvic_config(void)
{
    NVIC_InitTypeDef nvic_init;
    nvic_init.NVIC_IRQChannel = TIM1_CC_IRQn;
    nvic_init.NVIC_IRQChannelCmd = ENABLE;
    nvic_init.NVIC_IRQChannelPreemptionPriority = 0;
    nvic_init.NVIC_IRQChannelSubPriority = 0;
    NVIC_Init(&nvic_init);
}

void TIM1_CC_IRQHandler()
{
    uint16_t ic1 = TIM_GetCapture1(TIM1);
    uint16_t ic2 = TIM_GetCapture2(TIM1);

    if(ic1)
        printf("%d %d\n", ic1 + 1, ic2 + 1);

    TIM_ClearITPendingBit(TIM1, TIM_IT_CC1);
}
```

总结：定时器的配置

1. TIM_TimeBaseInitTypeDef配置并初始化：
   时钟模块使能：RCC_APByPeriphClockCmd(RCC_APByPeriph_TIMx, ENABLE);
   TIM_Period、TIM_Prescaler、TIM_ClockDivision、TIM_CounterMode。
   TIM_TimeBaseInit(TIMx, &tim_init);
   TIM_Cmd(TIMx, ENABLE);
   中断设置：TIM_ITConfig(TIMx, TIM_IT_XXXXX);
2. 需要捕捉外部触发计数时（计数器功能）：
   TIM_ETRClockMode1Config(TIMx, TIM_ExtTRGPSC_OFF, TIM_ExtTRGPolarity_NonInverted, 0);
3. 捕获脉冲时，需要使用TIM_ICInitTypeDef：
   设置相关字段：TIM_Channel、TIM_ICFilter、TIM_ICPolarity、TIM_ICPrescaler、TIM_ICSelection等。
   TIM_ICInit(TIMx, &tim_ic_init);
4. 输出PWM时，需要使用TIM_OCInitTypeDef：
   设置相关字段：TIM_OCMode、TIM_OCPolarity、TIM_OutputState、TIM_Pulse等。

8.3 系统嘀嗒定时器（Systick）

一个定时器产生周期性的中断，以维持操作系统为多个任务许以不同数目的时间片，来保证没有一个任务能够霸占操作

8.4 看门狗定时器（WatchDog）

8.4.1 独立看门狗

使用独立的时钟，即使主时钟故障它仍然有效，能够完全独立工作，对时间精度要求低的场合。

8.4.2 窗口看门狗

由APB1分频后产生的时钟驱动，精确度较高，常用来检测由外部干扰或不可见的逻辑条件造成的应用程序产生的软件故障。

1. 独立看门狗没有中断，窗口看门狗有中断
2. 独立看门狗有硬件软件之分，窗口看门狗只能软件控制
3. 独立看门狗只有下限，窗口看门狗又下限和上限
4. 独立看门狗Iwdg——独立于系统之外，因为有独立时钟，所以不受系统影响的系统故障探测器。主要用于监视硬件错误；窗口看门狗wwdg——系统内部的故障探测器，时钟与系统相同。

8.5 实时时钟（RTC）

RTC（Real-Time Clock)实时时钟为操作系统提供了一个可靠的时间，并且在断电的情况下，RTC实时时钟也可以通过电池供电，一直运行下去。RTC模块之所以具有实时时钟功能，是因为它内部维持了一个独立的定时器，通过配置，可以让它准确地每秒钟中断一次。