**刘禹麟 12211351**

**Write out the registers those are used (configurated or read) and related with the Basic TIM by the project (LED\_BasicTImer) and their meanings one by one.**

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM6, ENABLE);

使能定时器 TIM6 的时钟。

寄存器：RCC\_APB1ENR（RCC APB1 peripheral clock enable register）。

TIM\_TimeBaseStructure.TIM\_Period = BASIC\_TIM\_Period;

设置自动重装载寄存器的值，即计数器计数到该值后会产生一个更新事件。

寄存器：TIM6\_ARR（TIM6 Auto-Reload Register）。

TIM\_TimeBaseStructure.TIM\_Prescaler = BASIC\_TIM\_Prescaler;

设置时钟预分频数，用于控制计数器时钟的频率。

寄存器：TIM6\_PSC（TIM6 Prescaler Register）。

TIM\_TimeBaseInit(TIM6, &TIM\_TimeBaseStructure);

初始化定时器 TIM6 的基本配置。

NVIC\_InitStructure.NVIC\_IRQChannel = TIM6\_IRQn;

设置中断向量为 TIM6\_IRQn，即 TIM6 的中断。

寄存器：NVIC（Nested Vector Interrupt Controller）。

NVIC\_InitStructure.NVIC\_IRQChannelPreemptionPriority = 0;

设置中断的主优先级。

NVIC\_InitStructure.NVIC\_IRQChannelSubPriority = 3;

设置中断的次优先级。

NVIC\_InitStructure.NVIC\_IRQChannelCmd = ENABLE;

使能中断向量。

TIM\_ClearFlag(TIM6, TIM\_FLAG\_Update);

清除计数器中断标志位，以确保在进入中断服务函数之前标志位被清除。

寄存器：TIM6\_SR（TIM6 Status Register）。

TIM\_ITConfig(TIM6, TIM\_IT\_Update, ENABLE);

开启定时器的更新中断。

寄存器：TIM6\_DIER（TIM6 DMA/Interrupt Enable Register）。

TIM\_Cmd(TIM6, ENABLE);

使能定时器 TIM6。

寄存器：TIM6\_CR1（TIM6 Control Register 1）。

**Write out the registers those are used (configurated or read) by the function TIM\_ICInit() & TIMx\_IRQHandler() and their meanings one by one.**

TIM\_ICInitTypeDef TIM\_ICInitStructure;

// 配置输入捕获的通道，需要根据具体的 GPIO 来配置

TIM\_ICInitStructure.TIM\_Channel = GENERAL\_TIM\_CHANNEL\_x;

// 输入捕获信号的极性配置，TIMx\_CCER 寄存器

TIM\_ICInitStructure.TIM\_ICPolarity = GENERAL\_TIM\_STRAT\_ICPolarity;

// 输入通道和捕获通道的映射关系，有直连和非直连两种，TIMx\_CCMR1 寄存器

TIM\_ICInitStructure.TIM\_ICSelection = TIM\_ICSelection\_DirectTI;

// 输入的需要被捕获的信号的分频系数，TIMx\_CCER 寄存器

TIM\_ICInitStructure.TIM\_ICPrescaler = TIM\_ICPSC\_DIV1;

// 输入的需要被捕获的信号的滤波系数，TIMx\_CCMR1 寄存器

TIM\_ICInitStructure.TIM\_ICFilter = 0;

// 定时器输入捕获初始化

TIM\_ICInit(GENERAL\_TIM, &TIM\_ICInitStructure);void GENERAL\_TIM\_INT\_FUN(void)

{

// 当要被捕获的信号的周期大于定时器的最长定时时，定时器就会溢出，产生更新中断

// 这个时候我们需要把这个最长的定时周期加到捕获信号的时间里面去

if ( TIM\_GetITStatus ( GENERAL\_TIM, TIM\_IT\_Update) != RESET )

{

TIM\_ICUserValueStructure.Capture\_Period ++;

TIM\_ClearITPendingBit ( GENERAL\_TIM, TIM\_FLAG\_Update );

}

// 上升沿捕获中断

if ( TIM\_GetITStatus (GENERAL\_TIM, GENERAL\_TIM\_IT\_CCx ) != RESET)

{

// 第一次捕获

if ( TIM\_ICUserValueStructure.Capture\_StartFlag == 0 )

{

// 计数器清 0

TIM\_SetCounter ( GENERAL\_TIM, 0 );

// 自动重装载寄存器更新标志清 0

TIM\_ICUserValueStructure.Capture\_Period = 0;

// 存捕获比较寄存器的值的变量的值清 0

TIM\_ICUserValueStructure.Capture\_CcrValue = 0;

// 当第一次捕获到上升沿之后，就把捕获边沿配置为下降沿

GENERAL\_TIM\_OCxPolarityConfig\_FUN(GENERAL\_TIM, TIM\_ICPolarity\_Falling);

// 开始捕获标准置 1

TIM\_ICUserValueStructure.Capture\_StartFlag = 1;

}

// 下降沿捕获中断

else // 第二次捕获

{// 获取捕获比较寄存器的值，这个值就是捕获到的高电平的时间的值

TIM\_ICUserValueStructure.Capture\_CcrValue =

GENERAL\_TIM\_GetCapturex\_FUN (GENERAL\_TIM);

// 当第二次捕获到下降沿之后，就把捕获边沿配置为上升沿，好开启新的一轮捕

获

GENERAL\_TIM\_OCxPolarityConfig\_FUN(GENERAL\_TIM, TIM\_ICPolarity\_Rising);

// 开始捕获标志清 0

TIM\_ICUserValueStructure.Capture\_StartFlag = 0;

// 捕获完成标志置 1

TIM\_ICUserValueStructure.Capture\_FinishFlag = 1;

}

TIM\_ClearITPendingBit (GENERAL\_TIM,GENERAL\_TIM\_IT\_CCx);

}

}

用到的寄存器：

TIMx\_CR1 - Timer Control Register 1：

TIM\_ICInitStructure.TIM\_Channel 配置捕获通道时可能涉及。

TIM\_ICInitStructure.TIM\_CounterMode 在时基结构体初始化时可能涉及。

TIMx\_CCMR1 - Capture/Compare Mode Register 1：

TIM\_ICInitStructure.TIM\_ICSelection 和 TIM\_ICInitStructure.TIM\_ICPrescaler 配置输入捕获模

式时涉及。

TIMx\_CCER - Capture/Compare Enable Register：

TIM\_ICInitStructure.TIM\_ICPolarity 配置输入捕获信号的极性时涉及。

TIMx\_SR - Status Register：

TIM\_ClearITPendingBit 和 TIM\_GetITStatus 用于清除和检查中断标志位。

TIMx\_CCR1 - Capture/Compare Register 1：

TIM\_ICUserValueStructure.Capture\_CcrValue 存储捕获比较寄存器的值。

TIMx\_EGR - Event Generation Register：

TIM\_ClearITPendingBit 用于清除中断标志位。

**Sort out the programme flow of TIMx\_IRQHandler( ).**

void GENERAL\_TIM\_INT\_FUN(void)

{

// 当要被捕获的信号的周期大于定时器的最长定时时，定时器就会溢出，产生更新中断

// 这个时候我们需要把这个最长的定时周期加到捕获信号的时间里面去

if ( TIM\_GetITStatus ( GENERAL\_TIM, TIM\_IT\_Update) != RESET ){

TIM\_ICUserValueStructure.Capture\_Period ++;

TIM\_ClearITPendingBit ( GENERAL\_TIM, TIM\_FLAG\_Update );

}

// 上升沿捕获中断

if ( TIM\_GetITStatus (GENERAL\_TIM, GENERAL\_TIM\_IT\_CCx ) != RESET)

{

// 第一次捕获

if ( TIM\_ICUserValueStructure.Capture\_StartFlag == 0 )

{

// 计数器清 0

TIM\_SetCounter ( GENERAL\_TIM, 0 );

// 自动重装载寄存器更新标志清 0

TIM\_ICUserValueStructure.Capture\_Period = 0;

// 存捕获比较寄存器的值的变量的值清 0

TIM\_ICUserValueStructure.Capture\_CcrValue = 0;

// 当第一次捕获到上升沿之后，就把捕获边沿配置为下降沿

GENERAL\_TIM\_OCxPolarityConfig\_FUN(GENERAL\_TIM, TIM\_ICPolarity\_Falling);

// 开始捕获标准置 1

TIM\_ICUserValueStructure.Capture\_StartFlag = 1;

}

// 下降沿捕获中断

else // 第二次捕获

{

// 获取捕获比较寄存器的值，这个值就是捕获到的高电平的时间的值

TIM\_ICUserValueStructure.Capture\_CcrValue =

GENERAL\_TIM\_GetCapturex\_FUN (GENERAL\_TIM);

// 当第二次捕获到下降沿之后，就把捕获边沿配置为上升沿，好开启新的一轮捕

获

GENERAL\_TIM\_OCxPolarityConfig\_FUN(GENERAL\_TIM, TIM\_ICPolarity\_Rising);

// 开始捕获标志清 0

TIM\_ICUserValueStructure.Capture\_StartFlag = 0;

// 捕获完成标志置 1

TIM\_ICUserValueStructure.Capture\_FinishFlag = 1;

}

TIM\_ClearITPendingBit (GENERAL\_TIM,GENERAL\_TIM\_IT\_CCx);

}

}

计时器清零，捕获上升沿，存储，捕获下降沿，存储

**Write out the registers those are used(configurated or read) by the function TIM\_OCxInit() for General Tim and their meanings one by one.**

TIM\_OCInitTypeDef TIM\_OCInitStructure;

// 配置为 PWM 模式 1

TIM\_OCInitStructure.TIM\_OCMode = TIM\_OCMode\_PWM1;

// 输出使能

TIM\_OCInitStructure.TIM\_OutputState = TIM\_OutputState\_Enable;

// 输出通道电平极性配置

TIM\_OCInitStructure.TIM\_OCPolarity = TIM\_OCPolarity\_High;

// 输出比较通道 1

TIM\_OCInitStructure.TIM\_Pulse = GENERAL\_TIM\_CCR1;

TIM\_OC1Init(GENERAL\_TIM, &TIM\_OCInitStructure);

TIM\_OC1PreloadConfig(GENERAL\_TIM, TIM\_OCPreload\_Enable);

// 输出比较通道 2

TIM\_OCInitStructure.TIM\_Pulse = GENERAL\_TIM\_CCR2;

TIM\_OC2Init(GENERAL\_TIM, &TIM\_OCInitStructure);

TIM\_OC2PreloadConfig(GENERAL\_TIM, TIM\_OCPreload\_Enable);

// 输出比较通道 3

TIM\_OCInitStructure.TIM\_Pulse = GENERAL\_TIM\_CCR3;

TIM\_OC3Init(GENERAL\_TIM, &TIM\_OCInitStructure);

TIM\_OC3PreloadConfig(GENERAL\_TIM, TIM\_OCPreload\_Enable);

// 输出比较通道 4

TIM\_OCInitStructure.TIM\_Pulse = GENERAL\_TIM\_CCR4;

TIM\_OC4Init(GENERAL\_TIM, &TIM\_OCInitStructure);

TIM\_OC4PreloadConfig(GENERAL\_TIM, TIM\_OCPreload\_Enable);

// 使能计数器

TIM\_Cmd(GENERAL\_TIM, ENABLE);

TIMx->CCMR1：捕获/比较模式寄存器 1，用于配置输出比较通道的模式。

TIMx->CCER：捕获/比较使能寄存器，用于配置输出比较通道的使能状态。

TIMx->CCR1、TIMx->CCR2、TIMx->CCR3、TIMx->CCR4：捕获/比较寄存器，分别配置输出比

较通道 1 到 4 的比较值。