学习 STM32 其实就是学习它的寄存器以及函数的使用，能用单片机的资源实现自己想要的功能，从最开始的点亮一个 led，到使用按键，串口，ADC/DAC，定时器计时计数，输出 PWM，驱动电机，使用 IIC，SPI 进行通讯，驱动各个电子模块等。下面我就 STM32 的标准库寄存器和常用函数做个整理，以下内容均来自《STM32 固件库使用手册中文版》，方便日后查阅。

1、GPIO 相关的寄存器、库函数及举例

在这里插入图片描述

GPIO寄存器结构

寄存器	描述
CRL	端口配置低寄存器
CRH	端口配置高寄存器
IDR	端口输入数据寄存器
ODR	端口输出数据寄存器
BSRR	端口位设置/复位寄存器
BRR	端口位复位寄存器
LCKR	端口配置锁定寄存器
EVCR	事件控制寄存器
MAPR	复用重映射和调试 I/O 配置寄存器
EXTICR	外部中断线路 0-15 配置寄存器

在这里插入图片描述

GPIO库函数

函数名	描述
GPIO_DeInit	将外设 GPIOx 寄存器重设为缺省值
GPIO_AFIODeInit	将复用功能（重映射事件控制和 EXTI 设置）重设为缺省值
GPIO_Init	根据 GPIO_InitStruct 中指定的参数初始化外设 GPIOx 寄存器
GPIO_StructInit	把 GPIO_InitStruct 中的每一个参数按缺省值填入
GPIO_ReadInputDataBit	读取指定端口管脚的输入
GPIO_ReadInputData	读取指定的 GPIO 端口输入
GPIO_ReadOutputDataBit	读取指定端口管脚的输出
GPIO_ReadOutputData	读取指定的 GPIO 端口输出
GPIO_SetBits	设置指定的数据端口位
GPIO_ResetBits	清除指定的数据端口位
GPIO_WriteBit	设置或者清除指定的数据端口位
GPIO_Write	向指定 GPIO 数据端口写入数据
GPIO_PinLockConfig	锁定 GPIO 管脚设置寄存器
GPIO_EventOutputConfig	选择 GPIO 管脚用作事件输出
GPIO_EventOutputCmd	使能或者失能事件输出
GPIO_PinRemapConfig	改变指定管脚的映射
GPIO_EXTILineConfig	选择 GPIO 管脚用作外部中断线路


```
GPIO_DeInit(GPIOA);
GPIO_AFIODeInit();
```

```
GPIO_Init(GPIOE, &GPIO_InitStructure);
GPIO_StructInit(&GPIO_InitStructure);
GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_7);
GPIO_ReadInputData(GPIOC);
GPIO_ReadOutputDataBit(GPIOB, GPIO_Pin_7);
GPIO_ReadOutputData(GPIOC);
GPIO_SetBits(GPIOA, GPIO_Pin_10);
GPIO_ResetBits(GPIOA, GPIO_Pin_10);
GPIO_WriteBit(GPIOA, GPIO_Pin_15, Bit_SET);
GPIO_Write(GPIOA, 0x1101);
GPIO_PinLockConfig(GPIOA, GPIO_Pin_0);
GPIO_EventOutputConfig(GPIO_PortSourceGPIOE, GPIO_PinSource5);
GPIO_EventOutputCmd(ENABLE);
GPIO_PinRemapConfig(GPIO_Remap_I2C1, ENABLE);
GPIO_EXTILineConfig(GPIO_PortSource_GPIOB, GPIO_PinSource8);
```

2、NVIC 相关的寄存器、库函数及举例

NVIC寄存器结构

寄存器	描述
Enable	中断设置使能寄存器
Disable	中断清除使能寄存器
Set	中断设置待处理寄存器
Clear	中断清除待处理寄存器
Active	中断活动位寄存器
Priority	中断优先级寄存器
CPUID	CPU ID 基寄存器
IRQControlStatus	中断控制状态寄存器
ExceptionTableOffset	向量表移位寄存器

NVIC库函数

函数名	描述
NVIC_DeInit	将外设 NVIC 寄存器重设为缺省值
NVIC_SCBDeInit	将外设 SCB 寄存器重设为缺省值
NVIC_PriorityGroupConfig	设置优先级分组：先占优先级和从优先级
NVIC_Init	根据 NVIC_InitStruct 中指定的参数初始化外设 NVIC 寄存器
NVIC_StructInit	把 NVIC_InitStruct 中的每一个参数按缺省值填入
NVIC_SETPRIMASK	使能 PRIMASK 优先级：提升执行优先级至 0
NVIC_RESETPRIMASK	失能 PRIMASK 优先级
NVIC_SETFAULTMASK	使能 FAULTMASK 优先级：提升执行优先级至-1
NVIC_RESETFAULTMASK	失能 FAULTMASK 优先级
NVIC_BASEPRICONFIG	改变执行优先级从 N（最低可设置优先级）提升至 1
NVIC_GetBASEPRI	返回 BASEPRI 屏蔽值
NVIC_GetCurrentPendingIRQChannel	返回当前待处理 IRQ 标识符
NVIC_GetIRQChannelPendingBitStatus	检查指定的 IRQ 通道待处理位设置与否
NVIC_SetIRQChannelPendingBit	设置指定的 IRQ 通道待处理位
NVIC_ClearIRQChannelPendingBit	清除指定的 IRQ 通道待处理位
NVIC_GetCurrentActiveHandler	返回当前活动的 Handler（IRQ 通道和系统 Handler）的标识符
NVIC_GetIRQChannelActiveBitStatus	检查指定的 IRQ 通道活动位设置与否
NVIC_GetCPUID	返回 ID 号码，Cortex-M3 内核的版本号和实现细节
NVIC_SetVectorTable	设置向量表的位置和偏移
NVIC_GenerateSystemReset	产生一个系统复位
NVIC_GenerateCoreReset	产生一个内核（内核+NVIC）复位
NVIC_SystemLPConfig	选择系统进入低功耗模式的条件
NVIC_SystemHandlerConfig	使能或者失能指定的系统 Handler
NVIC_SystemHandlerPriorityConfig	设置指定的系统 Handler 优先级
NVIC_GetSystemHandlerPendingBitStatus	检查指定的系统 Handler 待处理位设置与否
NVIC_SetSystemHandlerPendingBit	设置系统 Handler 待处理位
NVIC_ClearSystemHandlerPendingBit	清除系统 Handler 待处理位
NVIC_GetSystemHandlerActiveBitStatus	检查系统 Handler 活动位设置与否
NVIC_GetFaultHandlerSources	返回表示出错的系统 Handler 源
NVIC_GetFaultAddress	返回产生表示出错的系统 Handler 所在位置的地址


```
NVIC_DeInit();
NVIC_SCBDeInit();
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
EXTI_Init(&EXTI_InitStructure);
NVIC_StructInit(&NVIC_InitStructure);
NVIC_SETPRIMASK();
NVIC_RESETPRIMASK();
NVIC_SETFAULTMASK();
NVIC_RESETPRIMASK();
NVIC_BASEPRICONFIG(10);
NVIC_GetBASEPRI();
NVIC_GetCurrentPendingIRQChannel();
NVIC_GetIRQChannelPendingBitStatus(ADC_IRQChannel);
NVIC_SetIRQChannelPendingBit(SPI1_IRQChannel);
NVIC_ClearIRQChannelPendingBit(ADC_IRQChannel);
NVIC_GetCurrentActiveHandler();
```

NVIC_GetIRQChannelActiveBitStatus(ADC_IRQChannel);
NVIC_GetCPUID();
NVIC_SetVectorTable(NVIC_VectTab_FLASH, 0x0);
NVIC_GenerateSystemReset();
NVIC_GenerateCoreReset();
NVIC_SystemLPConfig(SEVONPEND, ENABLE);
NVIC_SystemHandlerConfig(SystemHandler_MemoryManage, ENABLE);
NVIC_SystemHandlerPriorityConfig(SystemHandler_MemoryManage, 2, 8);
NVIC_GetSystemHandlerPendingBitStatus(SystemHandler_MemoryManage);
NVIC_SetSystemHandlerPendingBit(SystemHandler_NMI);
NVIC_ClearSystemHandlerPendingBit(SystemHandler_SysTick);
NVIC_GetSystemHandlerActiveBitStatus(SystemHandler_BusFault);
NVIC_GetFaultHandlerSources(SystemHandler_BusFault);
NVIC_GetFaultAddress(SystemHandler_BusFault);

3.RCC 相关的寄存器、库函数及举例

复位和时钟设置（RCC）

寄存器	描述
CR	时钟控制寄存器
CFGR	时钟配置寄存器
CIR	时钟中断寄存器
APB2RSTR	APB2 外设复位寄存器
APB1RSTR	APB1 外设复位寄存器
AHBENR	AHB 外设时钟使能寄存器
APB2ENR	APB2 外设时钟使能寄存器
APB1ENR	APB1 外设时钟使能寄存器
BDCR	备份域控制寄存器
CSR	控制/状态寄存器

在这里

RCC库函数

函数名	描述
RCC_DeInit	将外设 RCC 寄存器重设为缺省值
RCC_HSEConfig	设置外部高速晶振（HSE）
RCC_WaitForHSEStartUp	等待 HSE 起振
RCC_AdjustHSICalibrationValue	调整内部高速晶振（HSI）校准值
RCC_HSICmd	使能或者失能内部高速晶振（HSI）
RCC_PLLConfig	设置 PLL 时钟源及倍频系数
RCC_PLLCmd	使能或者失能 PLL
RCC_SYSCLKConfig	设置系统时钟（SYSCLK）
RCC_GetSYSCLKSource	返回用作系统时钟的时钟源
RCC_HCLKConfig	设置 AHB 时钟（HCLK）
RCC_PCLK1Config	设置低速 AHB 时钟（PCLK1）
RCC_PCLK2Config	设置高速 AHB 时钟（PCLK2）
RCC_ITConfig	使能或者失能指定的 RCC 中断
RCC_USBCLKConfig	设置 USB 时钟（USBCLK）
RCC_ADCCLKConfig	设置 ADC 时钟（ADCCLK）
RCC_LSEConfig	设置外部低速晶振（LSE）
RCC_LSICmd	使能或者失能内部低速晶振（LSI）
RCC_RTCCLKConfig	设置 RTC 时钟（RTCCLK）
RCC_RTCCLKCmd	使能或者失能 RTC 时钟
RCC_GetClocksFreq	返回不同片上时钟的频率
RCC_AHBPeriphClockCmd	使能或者失能 AHB 外设时钟
RCC_APB2PeriphClockCmd	使能或者失能 APB2 外设时钟
RCC_APB1PeriphClockCmd	使能或者失能 APB1 外设时钟
RCC_APB2PeriphResetCmd	强制或者释放高速 APB（APB2）外设复位
RCC_APB1PeriphResetCmd	强制或者释放低速 APB（APB1）外设复位
RCC_BackupResetCmd	强制或者释放后备域复位
RCC_ClockSecuritySystemCmd	使能或者失能时钟安全系统
RCC_MCOConfig	选择在 MCO 管脚上输出的时钟源
RCC_GetFlagStatus	检查指定的 RCC 标志位设置与否
RCC_ClearFlag	清除 RCC 的复位标志位
RCC_GetITStatus	检查指定的 RCC 中断发生与否
RCC_ClearITPendingBit	清除 RCC 的中断待处理位


```
RCC_DeInit();
RCC_HSEConfig(RCC_HSE_ON);
RCC_WaitForHSEStartUp();
RCC_AdjustHSICalibrationValue(0x1F);
RCC_HSICmd(ENABLE);
RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);
RCC_PLLCmd(ENABLE);
RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
RCC_GetSYSCLKSource() != 0x04
RCC_HCLKConfig(RCC_SYSCLK_Div1);
RCC_PCLK1Config(RCC_HCLK_Div2);
RCC_PCLK2Config(RCC_HCLK_Div1);
```

RCC_ITConfig(RCC_IT_PLLRDY, ENABLE);
RCC_USBCLKConfig(RCC_USBCLKSource_PLLCLK_1Div5);
RCC_ADCCLKConfig(RCC_PCLK2_Div2);
RCC_LSEConfig(RCC_LSE_ON);
RCC_LSICmd(ENABLE);
RCC_RTCCLKConfig(RCC_RTCCLKSource_LSE);
RCC_RTCCLKCmd(ENABLE);
RCC_GetClocksFreq(&RCC_Clocks);
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2 , ENABLE);
RCC_APB1PeriphResetCmd(RCC_APB1Periph_SPI2, DISABLE);
RCC_BackupResetCmd(ENABLE);
RCC_ClockSecuritySystemCmd(ENABLE);
RCC_MCOConfig(RCC_MCO_PLLCLK_Div2);
RCC_GetFlagStatus(RCC_FLAG_PLLRDY);
RCC_ClearFlag();
RCC_GetITStatus(RCC_IT_PLLRDY);
RCC_ClearITPendingBit(RCC_IT_PLLRDY);

4、RTC 相关的寄存器、库函数及举例

实时时钟（RTC）

寄存器	描述
CRH	控制寄存器高位
CRL	控制寄存器低位
PRLH	预分频装载寄存器高位
PRLL	预分频装载寄存器低位
DIVH	预分频分频因子寄存器高位
DIVL	预分频分频因子寄存器低位
CNTH	计数器寄存器高位
CNTL	计数器寄存器低位
ALRH	闹钟寄存器高位
ALRL	闹钟寄存器低位

RTC库函数

函数名	描述
RTC_ITConfig	使能或者失能指定的 RTC 中断
RTC_EnterConfigMode	进入 RTC 配置模式
RTC_ExitConfigMode	退出 RTC 配置模式
RTC_GetCounter	获取 RTC 计数器的值
RTC_SetCounter	设置 RTC 计数器的值
RTC_SetPrescaler	设置 RTC 预分频的值
RTC_SetAlarm	设置 RTC 闹钟的值
RTC_GetDivider	获取 RTC 预分频分频因子的值
RTC_WaitForLastTask	等待最近一次对 RTC 寄存器的写操作完成
RTC_WaitForSynchro	等待 RTC 寄存器(RTC_CNT, RTC_ALR and RTC_PRL)与 RTC 的 APB 时钟同步
RTC_GetFlagStatus	检查指定的 RTC 标志位设置与否
RTC_ClearFlag	清除 RTC 的待处理标志位
RTC_GetITStatus	检查指定的 RTC 中断发生与否
RTC_ClearITPendingBit	清除 RTC 的中断待处理位


```
RTC_ITConfig(RTC_IT_ALR, ENABLE);
RTC_EnterConfigMode();
RTC_ExitConfigMode();
RTC_GetCounter();
RTC_SetCounter(0xFFFF5555);
RTC_SetPrescaler(0x7A12);
RTC_SetAlarm(0xFFFFFFFA);
RTC_GetDivider();
RTC_WaitForLastTask();
RTC_WaitForSynchro();
RTC_GetFlagStatus(RTC_Flag_OW);
RTC_ClearFlag(RTC_FLAG_OW);
RTC_GetITStatus(RTC_IT_SEC);
RTC_ClearITPendingBit(RTC_IT_SEC);
```

5、SysTick 相关的寄存器、库函数及举例

Cortex系统定时器（SysTick）

寄存器	描述
CTRL	SysTick 控制和状态寄存器
LOAD	SysTick 重装载值寄存器
VAL	SysTick 当前值寄存器
CALIB	SysTick 校准值寄存器

SysTick库函数

函数名	描述
SysTick_CLKSourceConfig	设置 SysTick 时钟源
SysTick_SetReload	设置 SysTick 重装载值
SysTick_CounterCmd	使能或者失能 SysTick 计数器
SysTick_ITConfig	使能或者失能 SysTick 中断
SysTick_GetCounter	获取 SysTick 计数器的值
SysTick_GetFlagStatus	检查指定的 SysTick 标志位设置与否


```
SysTick_CLKSourceConfig(SysTick_CLKSource_HCLK);
SysTick_SetReload(0xFFFF);
SysTick_CounterCmd(SysTick_Counter_Enable);
SysTick_ITConfig(ENABLE);
SysTick_GetCounter();
SysTick_GetFlagStatus(SysTick_FLAG_COUNT);
```

6、ADC 相关的寄存器、库函数及举例

ADC寄存器

寄存器	描述
SR	ADC 状态寄存器
CR1	ADC 控制寄存器 1
CR2	ADC 控制寄存器 2
SMPR1	ADC 采样时间寄存器 1
SMPR2	ADC 采样时间寄存器 2
JOFR1	ADC 注入通道偏移寄存器 1
JOFR2	ADC 注入通道偏移寄存器 2
JOFR3	ADC 注入通道偏移寄存器 3
JOFR4	ADC 注入通道偏移寄存器 4
HTR	ADC 看门狗高阈值寄存器
LTR	ADC 看门狗低阈值寄存器
SQR1	ADC 规则序列寄存器 1
SQR2	ADC 规则序列寄存器 2
SQR3	ADC 规则序列寄存器 3
JSQR1	ADC 注入序列寄存器
DR1	ADC 规则数据寄存器 1
DR2	ADC 规则数据寄存器 2
DR3	ADC 规则数据寄存器 3
DR4	ADC 规则数据寄存器 4

ADC库函数

函数名	描述
ADC_DeInit	将外设 ADCx 的全部寄存器重设为缺省值
ADC_Init	根据 ADC_InitStruct 中指定的参数初始化外设 ADCx 的寄存器
ADC_StructInit	把 ADC_InitStruct 中的每一个参数按缺省值填入
ADC_Cmd	使能或者失能指定的 ADC
ADC_DMACmd	使能或者失能指定的 ADC 的 DMA 请求
ADC_ITConfig	使能或者失能指定的 ADC 的中断
ADC_ResetCalibration	重置指定的 ADC 的校准寄存器
ADC_GetResetCalibrationStatus	获取 ADC 重置校准寄存器的状态
ADC_StartCalibration	开始指定 ADC 的校准程序
ADC_GetCalibrationStatus	获取指定 ADC 的校准状态
ADC_SoftwareStartConvCmd	使能或者失能指定的 ADC 的软件转换启动功能
ADC_GetSoftwareStartConvStatus	获取 ADC 软件转换启动状态
ADC_DiscModeChannelCountConfig	对 ADC 规则组通道配置间断模式
ADC_DiscModeCmd	使能或者失能指定的 ADC 规则组通道的间断模式
ADC_RegularChannelConfig	设置指定 ADC 的规则组通道，设置它们的转化顺序和采样时间
ADC_ExternalTrigConvConfig	使能或者失能 ADCx 的经外部触发启动转换功能
ADC_GetConversionValue	返回最近一次 ADCx 规则组的转换结果
ADC_GetDuelModeConversionValue	返回最近一次双 ADC 模式下的转换结果
ADC_AutoInjectedConvCmd	使能或者失能指定 ADC 在规则组转化后自动开始注入组转换
ADC_InjectedDiscModeCmd	使能或者失能指定 ADC 的注入组间断模式
ADC_ExternalTrigInjectedConvConfig	配置 ADCx 的外部触发启动注入组转换功能
ADC_ExternalTrigInjectedConvCmd	使能或者失能 ADCx 的经外部触发启动注入组转换功能
ADC_SoftwareStartinjectedConvCmd	使能或者失能 ADCx 软件启动注入组转换功能
ADC_GetsoftwareStartinjectedConvStatus	获取指定 ADC 的软件启动注入组转换状态
ADC_InjectedChannleConfig	设置指定 ADC 的注入组通道，设置它们的转化顺序和采样时间
ADC_InjectedSequencerLengthConfig	设置注入组通道的转换序列长度
ADC_SetinjectedOffset	设置注入组通道的转换偏移值
ADC_GetInjectedConversionValue	返回 ADC 指定注入通道的转换结果
ADC_AnalogWatchdogCmd	使能或者失能指定单个/全体，规则/注入组通道上的模拟看门狗
ADC_AnalogWatchdongThresholdsConfig	设置模拟看门狗的高/低阈值
ADC_AnalogWatchdongSingleChannelConfig	对单个 ADC 通道设置模拟看门狗
ADC_TampSensorVrefintCmd	使能或者失能温度传感器和内部参考电压通道
ADC_GetFlagStatus	检查制定 ADC 标志位置 1 与否
ADC_ClearFlag	清除 ADCx 的待处理标志位
ADC_GetITStatus	检查指定的 ADC 中断是否发生
ADC_ClearITPendingBit	清除 ADCx 的中断待处理位


```
ADC_DeInit(ADC2);
ADC_Init(ADC1, &ADC_InitStructure);
ADC_StructInit(&ADC_InitStructure);
ADC_Cmd(ADC1, ENABLE);
ADC_DMACmd(ADC2, ENABLE);
ADC_ITConfig(ADC2, ADC_IT_EOC | ADC_IT_AWD, ENABLE);
ADC_ResetCalibration(ADC1);
ADC_GetResetCalibrationStatus(ADC2);
```

ADC_StartCalibration(ADC2);
ADC_GetCalibrationStatus(ADC2);
ADC_SoftwareStartConvCmd(ADC1, ENABLE);
ADC_GetSoftwareStartConvStatus(ADC1);
ADC_DiscModeChannelCountConfig(ADC1, 2);
ADC_DiscModeCmd(ADC1, ENABLE);
ADC_RegularChannelConfig(ADC1,ADC_Channel_2,1,ADC_SampleTime_7Cycles5);
ADC_ExternalTrigConvCmd(ADC1, ENABLE);
ADC_GetConversionValue(ADC1);
ADC_GetDualModeConversionValue();
ADC_AutoInjectedConvCmd(ADC2, ENABLE);
ADC_InjectedDiscModeCmd(ADC2, ENABLE);
ADC_ExternalTrigInjectedConvConfig(ADC1,ADC_ExternalTrigConv_T1_CC4);
ADC_ExternalTrigInjectedConvCmd(ADC1, ENABLE);
ADC_SoftwareStartInjectedConvCmd(ADC2, ENABLE);
ADC_GetSoftwareStartInjectedConvStatus(ADC1);
ADC_InjectedChannelConfig(ADC2, ADC_Channel_4, 11,ADC_SampleTime_71Cycles5);
ADC_InjectedSequencerLengthConfig(ADC1, 4);
ADC_SetInjectedOffset(ADC1, ADC_InjectedChannel_3, 0x100);
ADC_GetInjectedConversionValue(ADC1,ADC_InjectedChannel_1);
ADC_AnalogWatchdogCmd(ADC2,ADC_AnalogWatchdog_AllRegAllInjecEnable);
ADC_AnalogWatchdogThresholdsConfig(ADC1, 0x400, 0x100);
ADC_AnalogWatchdogSingleChannelConfig(ADC1, ADC_Channel_1);
ADC_TempSensorVrefintCmd(ENABLE);
ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC);
ADC_ClearFlag(ADC2, ADC_FLAG_STRT);
ADC_GetITStatus(ADC1, ADC_IT_AWD);
ADC_ClearITPendingBit(ADC2, ADC_IT_JEOC);

7、IIC 相关的寄存器、库函数及举例

I2C寄存器

寄存器	描述
CR1	I2C 控制寄存器 1
CR2	I2C 控制寄存器 2
OAR1	I2C 自身地址寄存器 1
OAR2	I2C 自身地址寄存器 2
DR	I2C 数据寄存器
SR1	I2C 状态寄存器 1
SR2	I2C 状态寄存器 2
CCR	I2C 时钟控制寄存器
TRISE	I2C 上升时间寄存器

I2C库函数

函数名	描述
I2C_DeInit	将外设 I2Cx 寄存器重设为缺省值
I2C_Init	根据 I2C_InitStruct 中指定的参数初始化外设 I2Cx 寄存器
I2C_StructInit	把 I2C_InitStruct 中的每一个参数按缺省值填入
I2C_Cmd	使能或者失能 I2C 外设
I2C_DMACmd	使能或者失能指定 I2C 的 DMA 请求
I2C_DMALastTransferCmd	使下一次 DMA 传输为最后一次传输
I2C_GenerateSTART	产生 I2Cx 传输 START 条件
I2C_GenerateSTOP	产生 I2Cx 传输 STOP 条件
I2C_AcknowledgeConfig	使能或者失能指定 I2C 的应答功能
I2C_OwnAddress2Config	设置指定 I2C 的自身地址 2
I2C_DualAddressCmd	使能或者失能指定 I2C 的双地址模式
I2C_GeneralCallCmd	使能或者失能指定 I2C 的广播呼叫功能
I2C_ITConfig	使能或者失能指定的 I2C 中断
I2C_SendData	通过外设 I2Cx 发送一个数据
I2C_ReceiveData	返回通过 I2Cx 最近接收的数据
I2C_Send7bitAddress	向指定的从 I2C 设备传送地址字
I2C_ReadRegister	读取指定的 I2C 寄存器并返回其值
I2C_SoftwareResetCmd	使能或者失能指定 I2C 的软件复位
I2C_SMBusAlertConfig	驱动指定 I2Cx 的 SMBusAlert 管脚电平为高或低
I2C_TransmitPEC	使能或者失能指定 I2C 的 PEC 传输
I2C_PECPositionConfig	选择指定 I2C 的 PEC 位置
I2C_CalculatePEC	使能或者失能指定 I2C 的传输字 PEC 值计算
I2C_GetPEC	返回指定 I2C 的 PEC 值
I2C_ARPCmd	使能或者失能指定 I2C 的 ARP
I2C_StretchClockCmd	使能或者失能指定 I2C 的时钟延展
I2C_FastModeDutyCycleConfig	选择指定 I2C 的快速模式占空比
I2C_GetLastEvent	返回最近一次 I2C 事件
I2C_CheckEvent	检查最近一次 I2C 事件是否是输入的事件
I2C_GetFlagStatus	检查指定的 I2C 标志位设置与否
I2C_ClearFlag	清除 I2Cx 的待处理标志位
I2C_GetITStatus	检查指定的 I2C 中断发生与否
I2C_ClearITPendingBit	清除 I2Cx 的中断待处理位

ST 的软件 IIC 不好用，一般都用硬件模拟，所以就不展开库函数的举例了。

8、SPI 相关的寄存器、库函数及举例

SPI寄存器

寄存器	描述
CR1	SPI 控制寄存器 1
CR2	SPI 控制寄存器 2
SR	SPI 状态寄存器
DR	SPI 数据寄存器
CRCPR	SPI CRC 多项式寄存器
RxCRCR	SPI 接收 CRC 寄存器
TxCRCR	SPI 发送 CRC 寄存器

SPI库函数

函数名	描述
SPI_DeInit	将外设 SPIx 寄存器重设为缺省值
SPI_Init	根据 SPI_InitStruct 中指定的参数初始化外设 SPIx 寄存器
SPI_StructInit	把 SPI_InitStruct 中的每一个参数按缺省值填入
SPI_Cmd	使能或者失能 SPI 外设
SPI_ITConfig	使能或者失能指定的 SPI 中断
SPI_DMACmd	使能或者失能指定 SPI 的 DMA 请求
SPI_SendData	通过外设 SPIx 发送一个数据
SPI_ReceiveData	返回通过 SPIx 最近接收的数据
SPI_DMALastTransferCmd	使下一次 DMA 传输为最后一次传输
SPI_NSSInternalSoftwareConfig	为选定的 SPI 软件配置内部 NSS 管脚
SPI_SSOutputCmd	使能或者失能指定的 SPI SS 输出
SPI_DataSizeConfig	设置选定的 SPI 数据大小
SPI_TransmitCRC	发送 SPIx 的 CRC 值
SPI_CalculateCRC	使能或者失能指定 SPI 的传输字 CRC 值计算
SPI_GetCRC	返回指定 SPI 的发送或者接受 CRC 寄存器值
SPI_GetCRCPolynomial	返回指定 SPI 的 CRC 多项式寄存器值
SPI_BiDirectionalLineConfig	选择指定 SPI 在双向模式下的数据传输方向
SPI_GetFlagStatus	检查指定的 SPI 标志位设置与否
SPI_ClearFlag	清除 SPIx 的待处理标志位
SPI_GetITStatus	检查指定的 SPI 中断发生与否
SPI_ClearITPendingBit	清除 SPIx 的中断待处理位


```
SPI_DeInit(SPI2);
SPI_Init(SPI1,&SPI_InitStructure);
SPI_StructInit(&SPI_InitStructure);
SPI_Cmd(SPI1, ENABLE);
SPI_ITConfig(SPI2, SPI_IT_TXE, ENABLE);
SPI_DMACmd(SPI2, SPI_DMAReq_Rx, ENABLE);
SPI_SendData(SPI1, 0xA5);
SPI_ReceiveData(SPI2);
SPI_NSSInternalSoftwareConfig(SPI2, SPI_NSSInternalSoft_Reset);
SPI_SSOutputCmd(SPI1, ENABLE);
SPI_DataSizeConfig(SPI2, SPI_DataSize_16b);
SPI_TransmitCRC(SPI1);
SPI_CalculateCRC(SPI2, ENABLE);
CRCValue = SPI_GetCRC(SPI1, SPI_CRC_Tx);
SPI_GetCRCPolynomial(SPI2);
SPI_BiDirectionalLineConfig(SPI_Direction_Tx);
```

SPI_ClearFlag(SPI2, SPI_FLAG_OVR);
SPI_GetITStatus(SPI1, SPI_IT_OVR);
SPI_ClearITPendingBit(SPI2, SPI_IT_CRCERR);

9、DMA 相关的寄存器、库函数及举例

DMA寄存器

寄存器	描述
ISR	DMA 中断状态寄存器
IFCR	DMA 中断标志位清除寄存器
CCRx	DMA 通道 x 设置寄存器
CNDTRx	DMA 通道 x 待传输数据数目寄存器
CPARx	DMA 通道 x 外设地址寄存器
CMARx	DMA 通道 x 内存地址寄存器

DMA库函数

函数名	描述
DMA_DeInit	将 DMA 的通道 x 寄存器重设为缺省值
DMA_Init	根据 DMA_InitStruct 中指定的参数初始化 DMA 的通道 x 寄存器
DMA_StructInit	把 DMA_InitStruct 中的每一个参数按缺省值填入
DMA_Cmd	使能或者失能指定的通道 x
DMA_ITConfig	使能或者失能指定的通道 x 中断
DMA_GetCurrDataCounte	返回当前 DMA 通道 x 剩余的待传输数据数目
DMA_GetFlagStatus	检查指定的 DMA 通道 x 标志位设置与否
DMA_ClearFlag	清除 DMA 通道 x 待处理标志位
DMA_GetITStatus	检查指定的 DMA 通道 x 中断发生与否
DMA_ClearITPendingBit	清除 DMA 通道 x 中断待处理标志位

DMA_DeInit(DMA_Channel2);
DMA_Init(DMA,&DMA_InitStructure);
DMA_StructInit(&DMA_InitStructure);
DMA_Cmd(DMA_Channel7, ENABLE);
DMA_ITConfig(DMA_Channel5, DMA_IT_TC, ENABLE);
DMA_GetCurrDataCounter(DMA_Channel2);
DMA_GetFlagStatus(DMA_FLAG_HT6);
DMA_ClearFlag(DMA_FLAG_TE3);
DMA_GetITStatus(DMA_IT_TC7);
DMA_ClearITPendingBit(DMA_IT_GL5);

10、USART 相关的寄存器、库函数及举例

USART寄存器

寄存器	描述
SR	USART 状态寄存器
DR	USART 数据寄存器
BRR	USART 波特率寄存器
CR1	USART 控制寄存器 1
CR2	USART 控制寄存器 2
CR3	USART 控制寄存器 3
GTPR	USART 保护时间和预分频寄存器

USART库函数

函数名	描述
USART_DeInit	将外设 USARTx 寄存器重设为缺省值
USART_Init	根据 USART_InitStruct 中指定的参数初始化外设 USARTx 寄存器
USART_StructInit	把 USART_InitStruct 中的每一个参数按缺省值填入
USART_Cmd	使能或者失能 USART 外设
USART_ITConfig	使能或者失能指定的 USART 中断
USART_DMACmd	使能或者失能指定 USART 的 DMA 请求
USART_SetAddress	设置 USART 节点的地址
USART_WakeUpConfig	选择 USART 的唤醒方式
USART_ReceiverWakeUpCmd	检查 USART 是否处于静默模式
USART_LINBreakDetectLengthConfig	设置 USART LIN 中断检测长度
USART_LINCmd	使能或者失能 USARTx 的 LIN 模式
USART_SendData	通过外设 USARTx 发送单个数据
USART_ReceiveData	返回 USARTx 最近接收到的数据
USART_SendBreak	发送中断字
USART_SetGuardTime	设置指定的 USART 保护时间
USART_SetPrescaler	设置 USART 时钟预分频
USART_SmartCardCmd	使能或者失能指定 USART 的智能卡模式
USART_SmartCardNackCmd	使能或者失能 NACK 传输
USART_HalfDuplexCmd	使能或者失能 USART 半双工模式
USART_IrDAConfig	设置 USART IrDA 模式
USART_IrDACmd	使能或者失能 USART IrDA 模式
USART_GetFlagStatus	检查指定的 USART 标志位设置与否
USART_ClearFlag	清除 USARTx 的待处理标志位
USART_GetITStatus	检查指定的 USART 中断发生与否
USART_ClearITPendingBit	清除 USARTx 的中断待处理位


```
USART_DeInit(USART1);
USART_Init(USART,&USART_InitStructure);
USART_StructInit(&USART_InitStructure);
USART_Cmd(USART1, ENABLE);
USART_ITConfig(USART1, USART_IT_Transmit ENABLE);
USART_DMACmd(USART2, USART_DMAReq_Rx | USART_DMAReq_Tx, ENABLE);
USART_SetAddress(USART2, 0x5);
USART_WakeUpConfig(USART1, USART_WakeUpIdleLine);
```

USART_ReceiverWakeUpCmd(USART3, DISABLE);
USART_LINBreakDetectLengthConfig(USART1,USART_LINDetectLength_10b);
USART_LINCmd(USART2, ENABLE);
USART_SendData(USART3, 0x26);
USART_ReceiveData(USART2);
USART_SendBreak(USART1);
USART_SetGuardTime(0x78);
USART_SetPrescaler(0x56);
USART_SmartCardCmd(USART1, ENABLE);
USART_SmartCardNACKCmd(USART1, ENABLE);
USART_HalfDuplexCmd(USART2, ENABLE);
USART_IrDAConfig(USART2,USART_IrDAMode_LowPower);
USART_IrDACmd(USART1, ENABLE);
USART_GetFlagStatus(USART1, USART_FLAG_TXE);
USART_ClearFlag(USART1,USART_FLAG_OR);
USART_GetITStatus(USART1, USART_IT_OverrunError);
USART_ClearITPendingBit(USART1,USART_IT_OverrunError);

11、通用定时器 TIM3 相关的寄存器、库函数及举例

通用定时器（TIM）

寄存器	描述
CR1	控制寄存器 1
CR2	控制寄存器 2
SMCR	从模式控制寄存器
DIER	DMA/中断使能寄存器
SR	状态寄存器
EGR	事件产生寄存器
CCMR1	捕获/比较模式寄存器 1
CCMR2	捕获/比较模式寄存器 2
CCER	捕获/比较使能寄存器
CNT	计数器寄存器
PSC	预分频寄存器
APR	自动重装载寄存器
CCR1	捕获/比较寄存器 1
CCR2	捕获/比较寄存器 2
CCR3	捕获/比较寄存器 3
CCR4	捕获/比较寄存器 4
DCR	DMA 控制寄存器
DMAR	连续模式的 DMA 地址寄存器

TIM库函数

函数名	描述
TIM_DeInit	将外设 TIMx 寄存器重设为缺省值
TIM_TimeBaseInit	根据 TIM_TimeBaseInitStruct 中指定的参数初始化 TIMx 的时间基数单位
TIM_OCInit	根据 TIM_OCInitStruct 中指定的参数初始化外设 TIMx
TIM_ICInit	根据 TIM_ICInitStruct 中指定的参数初始化外设 TIMx
TIM_TimeBaseStructInit	把 TIM_TimeBaseInitStruct 中的每一个参数按缺省值填入
TIM_OCStructInit	把 TIM_OCInitStruct 中的每一个参数按缺省值填入
TIM_ICStructInit	把 TIM_ICInitStruct 中的每一个参数按缺省值填入
TIM_Cmd	使能或者失能 TIMx 外设
TIM_ITConfig	使能或者失能指定的 TIM 中断
TIM_DMAConfig	设置 TIMx 的 DMA 接口
TIM_DMACmd	使能或者失能指定的 TIMx 的 DMA 请求
TIM_InternalClockConfig	设置 TIMx 内部时钟
TIM_ITRxExternalClockConfig	设置 TIMx 内部触发为外部时钟模式
TIM_TIxExternalClockConfig	设置 TIMx 触发为外部时钟
TIM_ETRClockMode1Config	配置 TIMx 外部时钟模式 1
TIM_ETRClockMode2Config	配置 TIMx 外部时钟模式 2
TIM_ETRConfig	配置 TIMx 外部触发
TIM_SelectInputTrigger	选择 TIMx 输入触发源
TIM_PrescalerConfig	设置 TIMx 预分频
TIM_CounterModeConfig	设置 TIMx 计数器模式
TIM_ForcedOC1Config	置 TIMx 输出 1 为活动或者非活动电平
TIM_ForcedOC2Config	置 TIMx 输出 2 为活动或者非活动电平
TIM_ForcedOC3Config	置 TIMx 输出 3 为活动或者非活动电平
TIM_ForcedOC4Config	置 TIMx 输出 4 为活动或者非活动电平
TIM_ARRPreloadConfig	使能或者失能 TIMx 在 ARR 上的预装载寄存器
TIM_SelectCCDMA	选择 TIMx 外设的捕获比较 DMA 源
TIM_OC1PreloadConfig	使能或者失能 TIMx 在 CCR1 上的预装载寄存器
TIM_OC2PreloadConfig	使能或者失能 TIMx 在 CCR2 上的预装载寄存器
TIM_OC3PreloadConfig	使能或者失能 TIMx 在 CCR3 上的预装载寄存器
TIM_OC4PreloadConfig	使能或者失能 TIMx 在 CCR4 上的预装载寄存器
TIM_OC1FastConfig	设置 TIMx 捕获比较 1 快速特征
TIM_OC2FastConfig	设置 TIMx 捕获比较 2 快速特征
TIM_OC3FastConfig	设置 TIMx 捕获比较 3 快速特征
TIM_OC4FastConfig	设置 TIMx 捕获比较 4 快速特征
TIM_ClearOC1Ref	在一个外部事件时清除或者保持 OCREF1 信号
TIM_ClearOC2Ref	在一个外部事件时清除或者保持 OCREF2 信号
TIM_ClearOC3Ref	在一个外部事件时清除或者保持 OCREF3 信号
TIM_ClearOC4Ref	在一个外部事件时清除或者保持 OCREF4 信号
TIM_UpdateDisableConfig	使能或者失能 TIMx 更新事件
TIM_EncoderInterfaceConfig	设置 TIMx 编码界面
TIM_GenerateEvent	设置 TIMx 事件由软件产生
TIM_OC1PolarityConfig	设置 TIMx 通道 1 极性
TIM_OC2PolarityConfig	设置 TIMx 通道 2 极性
TIM_OC3PolarityConfig	设置 TIMx 通道 3 极性
TIM_OC4PolarityConfig	设置 TIMx 通道 4 极性
TIM_UpdateRequestConfig	设置 TIMx 更新请求源
TIM_SelectHallSensor	使能或者失能 TIMx 霍尔传感器接口

```
TIM_DeInit(TIM2);
TIM_TimeBaseInit(TIM3,&TIM_TimeBaseInitStruct);
TIM_OCInit(TIM3,&TIM_OCInitStructure);
TIM_ICInit(TIM3,&TIM_ICInitStructure);
TIM_TimeBaseStructInit(& TIM_TimeBaseInitStructure);
TIM_OCStructInit(& TIM_OCInitStructure);
TIM_ICStructInit(& TIM_ICInitStructure);
TIM_Cmd(TIM2, ENABLE);
TIM_ITConfig(TIM2, TIM_IT_CC1, ENABLE );
TIM_DMAConfig(TIM2, TIM_DMABase_CCR1, TIM_DMABurstLength_1Byte);
TIM_DMACmd(TIM2, TIM_DMA_CC1, ENABLE);
TIM_InternalClockConfig(TIM2);
TIM_ITRxExternalClockConfig(TIM2, TIM_TS_ITR3);
TIM_TIxExternalClockConfig(TIM2,TIM_TS_TI1FP1,TIM_ICPolarity_Rising,0);
TIM_ExternalCLK1Config(TIM2,TIM_ExtTRGPSC_DIV2,TIM_ExtTRGPolarity_NonInverted, 0x0);
TIM_ETRConfig(TIM2,TIM_ExtTRGPSC_DIV2,TIM_ExtTRGPolarity_NonInverted, 0x0);
TIM_SelectInputTrigger(TIM2, TIM_TS_ITR3);
TIM_PrescalerConfig(TIM2, TIMPrescaler,TIM_PSCReloadMode_Immediate);
TIM_CounterModeConfig(TIM2, TIM_Counter_CenterAligned1);
TIM_ForcedOC1Config(TIM2, TIM_ForcedAction_Active);
TIM_ForcedOC2Config(TIM2, TIM_ForcedAction_Active);
TIM_ForcedOC3Config(TIM2, TIM_ForcedAction_Active);
TIM_ForcedOC4Config(TIM2, TIM_ForcedAction_Active);
TIM_ARRPreloadConfig(TIM2, ENABLE);
TIM_SelectCCDMA(TIM2, ENABLE);
TIM_OC1PreloadConfig(TIM2, TIM_OCPreload_Enable);
TIM_OC2PreloadConfig(TIM2, TIM_OCPreload_Enable);
TIM_OC3PreloadConfig(TIM2, TIM_OCPreload_Enable);
TIM_OC4PreloadConfig(TIM2, TIM_OCPreload_Enable);
TIM_OC1FastConfig(TIM2, TIM_OCFast_Enable);
TIM_OC2FastConfig(TIM2, TIM_OCFast_Enable);
TIM_OC3FastConfig(TIM2, TIM_OCFast_Enable);
TIM_OC4FastConfig(TIM2, TIM_OCFast_Enable);
TIM_ClearOC1Ref(TIM2, TIM_OCClear_Enable);
TIM_ClearOC2Ref(TIM2, TIM_OCClear_Enable);
TIM_ClearOC3Ref(TIM2, TIM_OCClear_Enable);
TIM_ClearOC4Ref(TIM2, TIM_OCClear_Enable);
TIM_UpdateDisableConfig(TIM2, DISABLE);
TIM_EncoderInterfaceConfig(TIM2,TIM_EncoderMode_TI1,TIM_ICPolarity_Rising,
TIM_ICPolarity_Rising);
TIM_GenerateEvent(TIM2, TIM_EventSource_Trigger);
TIM_OC1PolarityConfig(TIM2, TIM_OCPolarity_High);
TIM_OC2PolarityConfig(TIM2, TIM_OCPolarity_High);
TIM_OC3PolarityConfig(TIM2, TIM_OCPolarity_High);
```

```
TIM_OC4PolarityConfig(TIM2, TIM_OCPolarity_High);
TIM_UpdateRequestConfig(TIM2, TIM_UpdateSource_Regular);
TIM_SelectHallSensor(TIM2, ENABLE);
TIM_SelectOnePulseMode(TIM2, TIM_OPMode_Single);
TIM_SelectOutputTrigger(TIM2, TIM_TRGOSource_Update);
TIM_SelectSlaveMode(TIM2, TIM_SlaveMode_Gated);
TIM_SelectMasterSlaveMode(TIM2, TIM_MasterSlaveMode_Enable);TIM_SetCounter
(TIM2, TIMCounter);
TIM_SetAutoreload(TIM2, TIMAutoreload);
TIM_SetCompare1(TIM2, TIMCompare1);
TIM_SetCompare2(TIM2, TIMCompare2);
TIM_SetCompare3(TIM2, TIMCompare3);
TIM_SetCompare4(TIM2, TIMCompare4);
TIM_SetIC1Prescaler(TIM2, TIM_ICPSC_Div2);
TIM_SetIC2Prescaler(TIM2, TIM_ICPSC_Div2);
TIM_SetIC3Prescaler(TIM2, TIM_ICPSC_Div2);
TIM_SetIC4Prescaler(TIM2, TIM_ICPSC_Div2);
TIM_SetClockDivision(TIM2, TIM_CKD_DIV4);
TIM_GetCapture1(TIM2);//获得 TIM2 输入捕获 1 的值
TIM_GetCapture2(TIM2);//获得 TIM2 输入捕获 2 的值
TIM_GetCapture3(TIM2);//获得 TIM2 输入捕获 3 的值
TIM_GetCapture4(TIM2);//获得 TIM2 输入捕获 4 的值
TIM_GetCounter(TIM2);
TIM_GetPrescaler(TIM2);//获得 TIM2 的预分频值
TIM_ClearFlag(TIM2, TIM_FLAG_CC1);
TIM_GetITStatus(TIM2, TIM_IT_CC1) == SET； //判断定时器TIMx的中断类型TIM_IT
                                          // 是否发生中断
TIM_ClearITPendingBit(TIM2, TIM_IT_CC1);//清除定时器TIMx的中断TIM_IT标志位
```

通用定时器的内容真多，但是，高级定时器比通用定时器的内容更多，因为通用定时器有的功能，高级定时器都有。而高级定时器有的功能，通用定时器不一定有。

12、高级定时器 TIM1 相关的寄存器、库函数及举例

高级控制定时器（TIM1）

寄存器	描述
CR1	控制寄存器 1
CR2	控制寄存器 2
SMCR	从模式控制寄存器
DIER	DMA/中断使能寄存器
SR	状态寄存器
EGR	事件产生寄存器
CCMR1	捕获/比较模式寄存器 1
CCMR2	捕获/比较模式寄存器 2
CCER	捕获/比较使能寄存器
CNT	计数器寄存器
PSC	预分频寄存器
APR	自动重装载寄存器
RCR	周期计数寄存器
CCR1	捕获/比较寄存器 1
CCR2	捕获/比较寄存器 2
CCR3	捕获/比较寄存器 3
CCR4	捕获/比较寄存器 4
BDTR	刹车和死区寄存器
DCR	DMA 控制寄存器
DMAR	连续模式的 DMA 地址寄存器

TIM1 库函数

函数名	描述
TIM1_DeInit	将外设 TIM1 寄存器重设为缺省值
TIM1_TIM1BaseInit	根据 TIM1_TIM1BaseInitStruct 中指定的参数初始化 TIM1 的时间基数单位
TIM1_OC1Init	根据 TIM1_OCInitStruct 中指定的参数初始化 TIM1 通道 1
TIM1_OC2Init	根据 TIM1_OCInitStruct 中指定的参数初始化 TIM1 通道 2
TIM1_OC3Init	根据 TIM1_OCInitStruct 中指定的参数初始化 TIM1 通道 3
TIM1_OC4Init	根据 TIM1_OCInitStruct 中指定的参数初始化 TIM1 通道 4
TIM1_BDTRConfig	设置刹车特性，死区时间，锁电平，OSSI，OSSR 状态和 AOE（自动输出使能）
TIM1_ICInit	根据 TIM1_ICInitStruct 中指定的参数初始化外设 TIM1
TIM1_PWMIConfig	根据 TIM1_ICInitStruct 中指定的参数设置外设 TIM1 工作在 PWM 输入模式
TIM1_TIM1BaseStructInit	把 TIM1_TIM1BaseInitStruct 中的每一个参数按缺省值填入
TIM1_OCStructInit	把 TIM1_OCInitStruct 中的每一个参数按缺省值填入
TIM1_ICStructInit	把 TIM1_ICInitStruct 中的每一个参数按缺省值填入
TIM1_BDTRStructInit	把 TIM1_BDTRInitStruct 中的每一个参数按缺省值填入
TIM1_Cmd	使能或者失能 TIM1 外设
TIM1_CtrlPWMOutputs	使能或者失能 TIM1 外设的主输出
TIM1_ITConfig	使能或者失能指定的 TIM1 中断
TIM1_DMAConfig	设置 TIM1 的 DMA 接口
TIM1_DMACmd	使能或者失能指定的 TIM1 的 DMA 请求
TIM1_InternalClockConfig	设置 DMA 内部时钟
TIM1_ETRClockMode1Config	配置 TIM1 外部时钟模式 1
TIM1_ETRClockMode2Config	配置 TIM1 外部时钟模式 2
TIM1_ETRConfig	配置 TIM1 外部触发
TIM1_ITRxExternalClockConfig	设置 TIM1 内部触发为外部时钟模式
TIM1_TIxExternalClockConfig	设置 TIM1 触发为外部时钟
TIM1_SelectInputTrigger	选择 TIM1 输入触发源
TIM1_UpdateDisableConfig	使能或者失能 TIM1 更新事件
TIM1_UpdateRequestConfig	设置 TIM1 更新请求源
TIM1_SelectHallSensor	使能或者失能 TIM1 霍尔传感器接口
TIM1_SelectOnePulseMode	设置 TIM1 单脉冲模式
TIM1_SelectOutputTrigger	选择 TIM1 触发输出模式
TIM1_SelectSlaveMode	选择 TIM1 从模式
TIM1_SelectMasterSlaveMode	设置或者重置 TIM1 主/从模式
TIM1_EncoderInterfaceConfig	设置 TIM1 编码界面
TIM1_PrescalerConfig	设置 TIM1 预分频
TIM1_CounterModeConfig	设置 TIM1 计数器模式
TIM1_ForcedOC1Config	置 TIM1 输出 1 为活动或者非活动电平
TIM1_ForcedOC2Config	置 TIM1 输出 2 为活动或者非活动电平
TIM1_ForcedOC3Config	置 TIM1 输出 3 为活动或者非活动电平
TIM1_ForcedOC4Config	置 TIM1 输出 4 为活动或者非活动电平
TIM1_ARRPreloadConfig	使能或者失能 TIM1 在 ARR 上的预装载寄存器
TIM1_SelectCOM	选择 TIM1 外设的通讯事件
TIM1_SelectCCDMA	选择 TIM1 外设的捕获比较 DMA 源
TIM1_CCPreloadControl	设置或者重置 TIM1 捕获比较控制位
TIM1_OC1PreloadConfig	使能或者失能 TIM1 在 CCR1 上的预装载寄存器
TIM1_OC2PreloadConfig	使能或者失能 TIM1 在 CCR2 上的预装载寄存器
TIM1_OC3PreloadConfig	使能或者失能 TIM1 在 CCR3 上的预装载寄存器
TIM1_OC4PreloadConfig	使能或者失能 TIM1 在 CCR4 上的预装载寄存器
TIM1_OC1FastConfig	设置 TIM1 捕获比较 1 快速特征
TIM1_OC2FastConfig	设置 TIM1 捕获比较 2 快速特征
TIM1_OC3FastConfig	设置 TIM1 捕获比较 3 快速特征

```
TIM1_DeInit();
TIM_TimeBaseInit(TIM1,&TIM_TimeBaseInitStruct);
TIM1_OC1Init(&TIM1_OCInitStructure);
TIM1_OC2Init(&TIM1_OCInitStructure);
TIM1_OC3Init(&TIM1_OCInitStructure);
TIM1_OC4Init(&TIM1_OCInitStructure);
TIM1_BDTRConfig(&TIM1_BDTRInitStructure);
TIM1_ICInit(&TIM1_ICInitStructure);
TIM1_PWMIConfig(&TIM1_ICInitStructure);
TIM1_TimeBaseStructInit(& TIM1_TimeBaseInitStructure);
TIM1_OCStructInit(& TIM1_OCInitStructure);
TIM1_ICStructInit(& TIM1_ICInitStructure);
TIM1_BDTRStructInit(& TIM1_BDTRInitStructure);
TIM1_Cmd(ENABLE);
TIM1_CtrlPWMOutputs(ENABLE);
TIM1_ITConfig(TIM1_IT_CC1, ENABLE );
TIM1_DMAConfig(TIM1_DMABase_CCR1, TIM1_DMABurstLength_1Byte)
TIM1_DMACmd(TIM1_DMA_CC1, ENABLE);
TIM1_InternalClockConfig(TIM2);
TIM1_ExternalCLK1Config(TIM1_ExtTRGPSC_DIV2,TIM1_ExtTRGPolarity_NonInverted, 0x0);
TIM1_ExternalCLK2Config(TIM1_ExtTRGPSC_DIV2,TIM1_ExtTRGPolarity_NonInverted, 0x0);
TIM1_ETRConfig(TIM1_ExtTRGPSC_DIV2,TIM1_ExtTRGPolarity_NonInverted,0x0);
TIM1_ITRxExternalClockConfig(TIM1_TS_ITR3);
TIM1_TIxExternalClockConfig(TIM1_TS_TI1FP1, TIM1_ICPolarity_Rising, 0);
TIM1_SelectInputTrigger(TIM1_TS_ITR3);
TIM1_UpdateDisableConfig(DISABLE);
TIM1_UpdateRequestConfig(TIM1_UpdateSource_Regular);
TIM1_SelectHallSensor(ENABLE);
TIM1_SelectOnePulseMode(TIM1_OPMode_Single);
TIM1_SelectOutputTrigger(TIM1_TRGOSource_Update);
TIM1_SelectSlaveMode(TIM1_SlaveMode_Gated);
TIM1_SelectMasterSlaveMode(TIM2, TIM1_MasterSlaveMode_Enable);
TIM1_EncoderInterfaceConfig(TIM1_EncoderMode_1,TIM1_ICPolarity_Rising,TIM1_ICPolarity_Ri
sing);
TIM1_PrescalerConfig(0xFF00, TIM1_PSCReloadMode_Update);
TIM1_CounterModeConfig(TIM1_Counter_CenterAligned1);
TIM1_ForcedOC1Config(TIM1_ForcedAction_Active);
TIM1_ForcedOC2Config(TIM1_ForcedAction_Active);
TIM1_ForcedOC3Config(TIM1_ForcedAction_Active);
TIM1_ForcedOC4Config(TIM1_ForcedAction_Active);
TIM1_ARRPreloadConfig(ENABLE);
TIM1_SelectCOM(ENABLE);
TIM1_SelectCCDMA(ENABLE);
TIM1_CCPreloadControl(ENABLE);
```

```
TIM1_OC1PreloadConfig(TIM1_OCPreload_Enable);
TIM1_OC2PreloadConfig(TIM1_OCPreload_Enable);
TIM1_OC3PreloadConfig(TIM1_OCPreload_Enable);
TIM1_OC4PreloadConfig(TIM1_OCPreload_Enable);
TIM1_OC1FastConfig(TIM1_OCFast_Enable);
TIM1_OC2FastConfig(TIM1_OCFast_Enable);
TIM1_OC3FastConfig(TIM1_OCFast_Enable);
TIM1_OC4FastConfig(TIM1_OCFast_Enable);
TIM1_ClearOC1Ref(TIM1_OCClear_Enable);
TIM1_ClearOC2Ref(TIM1_OCClear_Enable);
TIM1_ClearOC3Ref(TIM1_OCClear_Enable);
TIM1_ClearOC4Ref(TIM1_OCClear_Enable);
TIM1_GenerateEvent(TIM1_EventSource_Trigger);
TIM1_OC1PolarityConfig(TIM1_OCPolarity_High);
TIM1_OC1NPolarityConfig(TIM1_OCNPolarity_High);
TIM1_OC2PolarityConfig(TIM1_OCPolarity_High);
TIM1_OC2NPolarityConfig(TIM1_OCNPolarity_High);
TIM1_OC3PolarityConfig(TIM1_OCPolarity_High);
TIM1_OC3NPolarityConfig(TIM1_OCNPolarity_High);
TIM1_OC4PolarityConfig(TIM1_OCPolarity_High);
TIM1_CCxCmd(TIM1_Channel_4, ENABLE);
TIM1_CCxNCmd(TIM1_Channel_3, ENABLE);
TIM1_SelectOCxM(TIM1_Channel_1, TIM1_OCMode_PWM2);
TIM1_SetCounter(TIM1Counter);
TIM1_SetAutoreload(TIM1Autoreload);
TIM1_SetCompare1(TIM1Compare1);
TIM1_SetCompare2(TIM1Compare2);
TIM1_SetCompare1(TIM1Compare3);
TIM1_SetCompare1(TIM1Compare4);
TIM1_SetIC1Prescaler(TIM1_ICPSC_Div2);
TIM1_SetIC2Prescaler(TIM1_ICPSC_Div2);
TIM1_SetIC3Prescaler(TIM1_ICPSC_Div2);
TIM1_SetIC4Prescaler(TIM1_ICPSC_Div2);
TIM1_SetClockDivision(TIM1_CKD_DIV4);
TIM1_GetCapture1();
TIM1_GetCapture2();
TIM1_GetCapture3();
TIM1_GetCapture4();
TIM1_GetCounter();
TIM1_GetPrescaler();
TIM1_GetFlagStatus(TIM1_FLAG_CC1) == SET;
TIM1_ClearFlag(TIM1_FLAG_CC1);
TIM1_GetITStatus(TIM1_IT_CC1) == SET;
TIM1_ClearITPendingBit(TIM1_IT_CC1);
```