stm32f103c8t6系统配置

标准库函数整理：

stm32.docx 34.7KB

第一：配置GPIO模块

1.实现步骤（主思路）

- 使能GPIO对应端口时钟
- 设置GPIO结构体，配置包括：工作模式，对应引脚，传输速度。
- 设置对应引脚的高/低电平，读取对应引脚的高/低电平

2.所涉函数整理（按配置顺序）

a.使能GPIO端口时钟函数

该函数使能APB2的外设时钟

```
1.  void RCC_APB2PeriphClockCmd(uint32_t RCC_APB2Periph, FunctionalState
NewState);
```

- 第一个参数：连接APB2总线的外设有GPIO(A--E)，定时器，UASRT，EXTID等
- 第二个参数：设置使能端ENABLE or DISABLE（使能/失能）
- 开启对应时钟，使用对应总线的时钟配置函数，具体看函数库文件

b.设置GPIO结构体

- 定义一个类型为初始化GPIO的结构体
- 设置结构体的内容，包括工作模式，对应引脚，传输速度（不可管）
- 使用GPIO初始函数，将配置内容进行设置

配置包括：工作模式，对应引脚，传输速度

```
    //2.使用GPIO_Init函数初始化GPIO
```

```
    GPIO_InitTypeDef GPIO_InitStructture;
    GPIO_InitStructture.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructture.GPIO_Pin = GPIO_Pin_0;
    GPIO_InitStructture.GPIO_Speed =GPIO_Speed_50MHz;
    GPIO_Init(GPIOA,&GPIO_InitStructture);
```

c.初始化GPIO函数

将配置内容进行设置

```
void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct);
```

- 第一个参数：选择对应的GPIO
- 第二个参数：先设置结构体，再把地址传入，结构体类型GPIO_InitTypeDef

d.将对应端口置1或置0函数

将对应的GPIOx，写1

```
void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
```

- 第一个参数：选择对应的GPIOx
- 第二个参数：选择对应GPIO_Pin_0~~ GPIO_Pin_12 ，置1

将对应的GPIOx，写0

```
void GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
```

- 第一个参数：选择对应的GPIOx
- 第二个参数：选择对应GPIO_Pin_0~~ GPIO_Pin_12 ，置0

将对应的GPIOx，对应的端口，写1 或 写0

```
void GPIO_WriteBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, BitAction BitVal);
```

- 第一个参数：选择对应的GPIOx
- 第二个参数：选择对应GPIO_Pin_0~~ GPIO_Pin_12
- 第三个参数：输入置1/0选择，Bit_RESET（置0），Bit_SET（置1）

将对应的GPIOx，全部端进行设置

```
void GPIO_Write(GPIO_TypeDef* GPIOx, uint16_t PortVal);
```

- 第一个参数：选择对应的GPIOx
- 第二个参数：输入对应寄存器的值（16进制数）

读取输入寄存器的值（一位）

```
uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
```

- 第一个参数：选择对应的GPIOx
- 第二个参数：选择对应的GPIO引脚

读取输入寄存器的值（全部）

```
uint16_t GPIO_ReadInputData(GPIO_TypeDef* GPIOx);
```

- 第一个参数：选择对应的GPIOx（一个GPIO的全部端口）

读取输出寄存器的值（一位）

```
uint8_t GPIO_ReadOutputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
```

- 第一个参数：选择对应的GPIOx
- 第二个参数：选择对应的GPIO引脚

读取输出寄存器的值（全部）

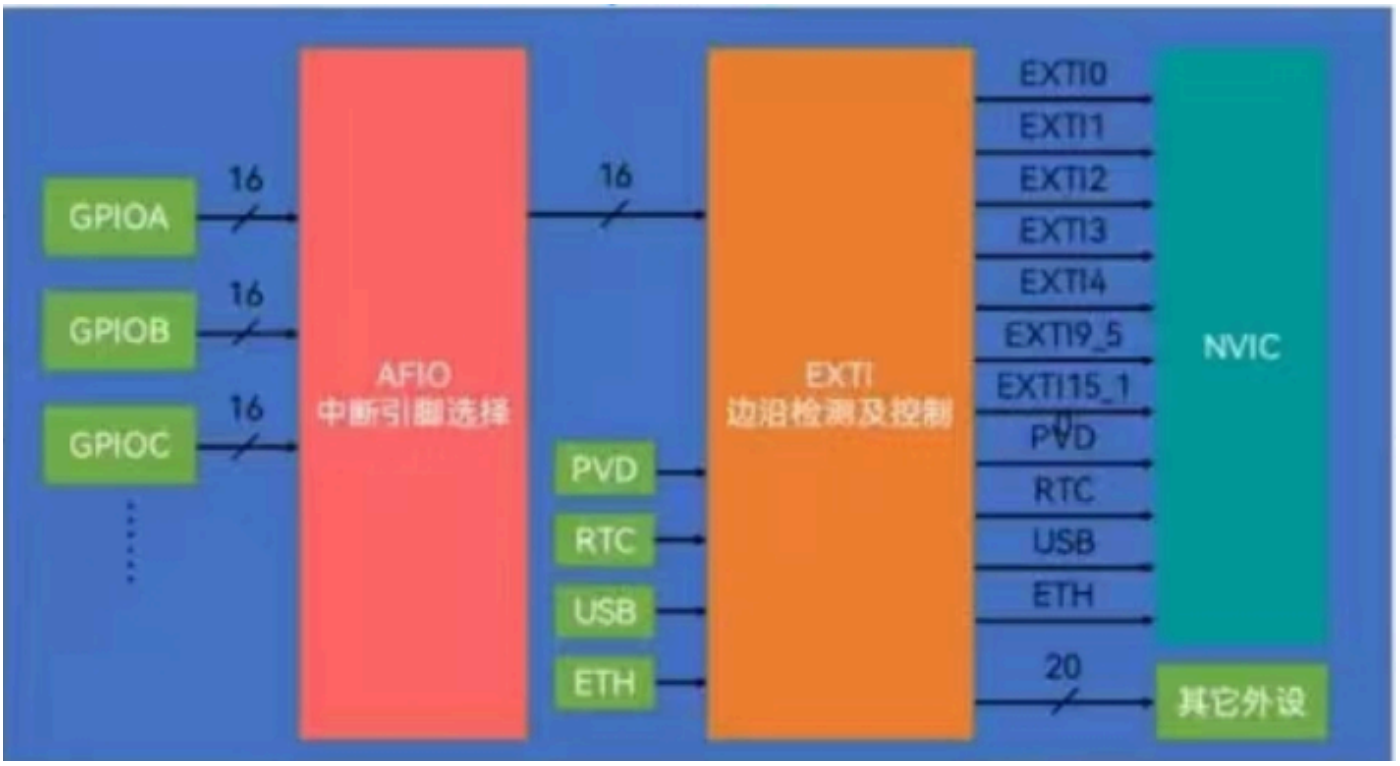
```
9.  uint16_t GPIO_ReadOutputData(GPIO_TypeDef* GPIOx);
```

- 第一个参数：选择对应的GPIOx（一个GPIO的全部端口）

3.理解操作寄存器

第二：配置外部中断模式

1.实现步骤（主思路）



- 使能对应功能模块的时钟：AFIO，GPIO，而EXTI，NVIC无需设置
- 配置GPIO设置相应功能：上拉输入，合适的端口
- 通过AFIO将GPIO端口设置外部中断模式，并连接外部中断外设
- EXTI外部中断配置外设：（可设置）中断方式，请求外部中断，中断屏蔽器，产生外部中断事件
- NVIC中断管理外设：设置各种中断的优先级（将其分组，并设置抢占优先级，响应优先级）

2.所涉函数整理（按配置顺序）

a.使能GPIO,AFIO时钟

时钟使能GPIO/AFIO

```
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);
```

b.通过AFIO设置GPIO模式外部中断

中断线路配置（选择对应GPIO设置外部中断）

```
GPIO_EXTILineConfig(GPIO_PortSourceGPIOB,GPIO_PinSource5);
```

c.初始化EXIT外部中断配置外设

- 定义一个EXTI_InitTypeDe类型的结构体EXTI_InitStructure
- 设置结构体内容：设置中端路线（EXTI_LIine5为GPIOx共有的），中断使能，设置中断模式（更新中断，中断事件），中断触发方式（上沿，下沿，上下沿）
- 将EXIT进行初始化

配置EXIT

```
    EXTI_InitTypeDef EXTI_InitStructure;
    EXTI_InitStructure.EXTI_Line = EXTI_Line5 ;
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
    EXTI_Init(&EXTI_InitStructure);
```

d.设置NVIC配置优先级的工作方式

- 该函数设置NVIC的优先级模式
- NVIC优先级模式（5种），区分与响应优先级和抢占优先级取值范围

NVIC优先级模式

```
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
```

f.初始化NVIC中断优先级模式

- 定义一个NVIC_InitTypeDef类型的结构体NVIC_InitStructure
- 设置结构体内容：设置好请求中断源（5-9端口的中断源EXTI9_5_IRQn）（也就是EXIT到NVIC的路线），NVIC使能，设置响应和抢占优先级
- 将NVIC进行初始化

NVIC优先级模式

```
    NVIC_InitTypeDef NVIC_InitStructure;
    NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
```

```
6     NVIC_Init(&NVIC_InitStructure);
```

e.外部中断标志位读取和清除

读取标志位

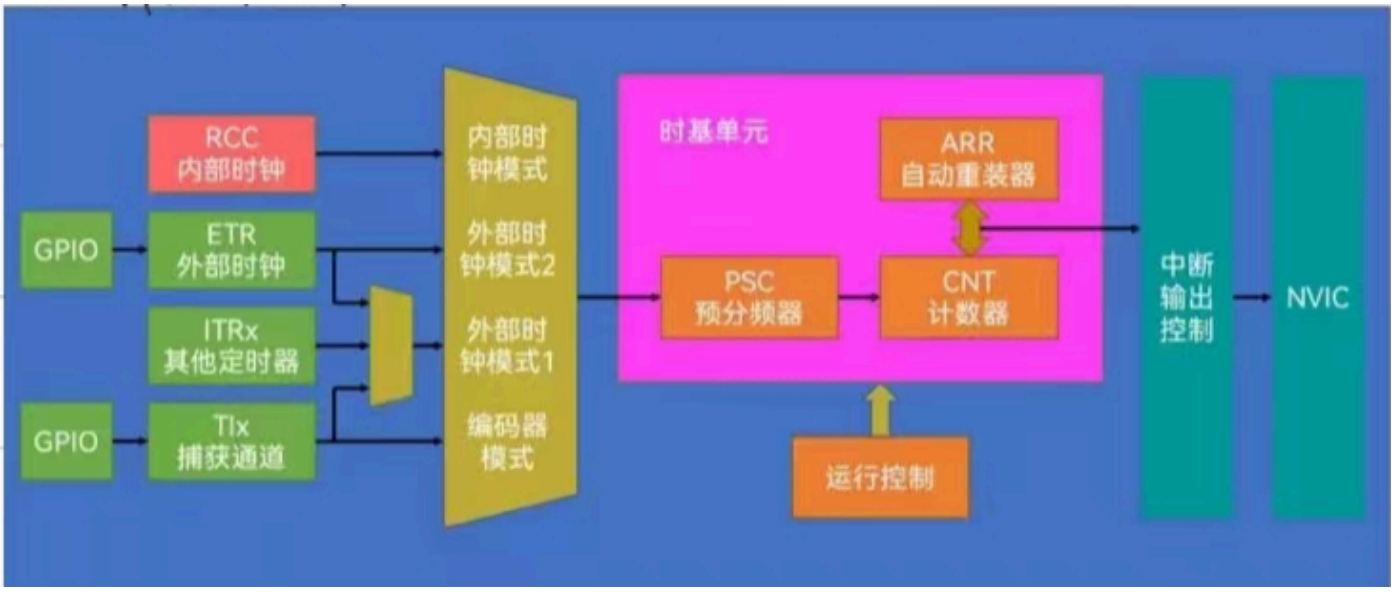
```
1 ITStatus EXTI_GetITStatus(uint32_t EXTI_Line);
```

清除标志位

```
1 EXTI_ClearITPendingBit(EXTI_Line5);
```

第三：配置定时器（定时，捕获，比较）

1.实现定时功能步骤（主思路）



- 使能对应功能模块的时钟：如：TIM2时钟使能
- 选择TIM2计数时钟源，通常使用的时内部时钟模式作为计数时钟（72Mhz），当然还可以使用外部或其他定时器时钟
- 配置定时器基本时基单元：PSC(预分频器)，ARR(自动重装值)，CNT(计数器)
- 当CNT值到达ARR值时，产生更新中断和中断事件，请求中断管理NVIC产生中断
- 设置NVIC配置，定时TIM2中断优先级

2.定时所涉函数整理（按配置顺序）

a.使能TIM时钟

使能TIM2时钟

```
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);
```

b.配置TIM计数时钟的选择（采用内部时钟）

TIM2配置成内部时钟源

```
TIM_InternalClockConfig(TIM2);
```

c.初始化TIM2基本单元结构体

- 定义一个TIM_TimeBaseInitTypeDef类型的结构体TIM_TimeBaseInitStructure
- 设置结构体内容：TIM_ClockDivision（与（TIM）基本框架无关，用滤波分频）TIM_CounterMode（定时器计数方式），TIM_Period（设置ARR值），TIM_Prescaler（设置PSC值，TIM_RepetitionCounter（无关）
- 初始化定时器TIM2基本单元结构

配置TIM2基本单元结构体内容

```
TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;
TIM_TimeBaseInitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
TIM_TimeBaseInitStructure.TIM_CounterMode = TIM_CounterMode_Up ;
TIM_TimeBaseInitStructure.TIM_Period = 10000-1;
TIM_TimeBaseInitStructure.TIM_Prescaler = 7200-1;
TIM_TimeBaseInitStructure.TIM_RepetitionCounter = 0 ;
TIM_TimeBaseInit(TIM2,&TIM_TimeBaseInitStructure);
```

d.开启更新中断与NVIC的通路

向NVIC请求中断

```
TIM_ITConfig(TIM2,TIM_IT_Update,ENABLE);
```

f.启动定时器

使能定时TIM2

```
TIM_Cmd(TIM2,ENABLE);
```

g.TIM中断标志位和清除函数

读取TIM中断标志位

```
TIM_GetITStatus(TIM2,TIM_IT_Update);
```

清除TIM中断标志位

```
TIM_ClearITPendingBit(TIM2,TIM_IT_Update);
```

h.定时中断函数

写入中断程序

```
void TIM2_IRQHandler(void)
{
    if(TIM_GetITStatus(TIM2,TIM_IT_Update) == SET)
    {
            TIM_ClearITPendingBit(TIM2,TIM_IT_Update);
    }
}

```

i.单独设置和读取CNT值

读取CNT值

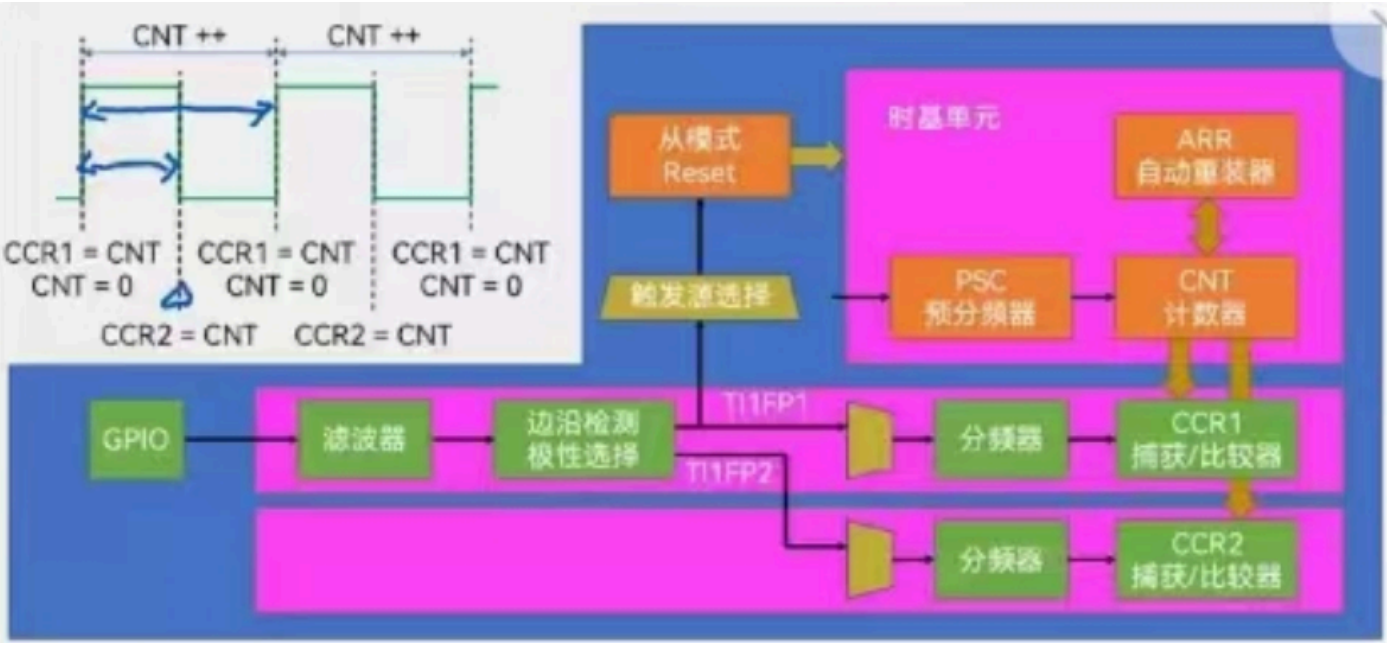
```
uint16_t TIM_GetCounter(TIM_TypeDef* TIMx)
```

设置CNT值

```
void TIM_SetCounter(TIM_TypeDef* TIMx, uint16_t Counter)
```

3.定时理解操作寄存器

4.实现捕获功能步骤（主思路）



- 使能对应功能模块的时钟：TIM，GPIO
- 这里GPIO作为输入端，所以选择对应的输入捕获端口，将其配置成上拉输入，端口的默认功能是有输入捕获功能的，下一步的是滤波和边沿检测电路。
- 配置定时器基本时基单元：PSC(预分频器)，ARR(自动重装值)，CNT(计数器)，输入捕获只检测两个边沿的时间，所以将PSC = 72 -1，ARR设置成最大的即可，CNT加一次，1/1000000s，可得到两个边沿的时间
- 设置定时器IC输入捕获功能的配置：TIM_Channel（选择对应的4个输入通道），TIM_ICFilter（捕获的滤波器效果），TIM_ICPolarity（边沿检测的极性选择），TIM_ICPrescale（分频选择，为1不分频，为2触发时间减半），TIM_ICSelection（直连通道，交叉通道）
- 需不要设置双通道（交叉通道），可以一个输入捕获通道，可测周期和占空比
- 设置主从模式，其作用是完成硬件的自动化，重置CNT的值，循环输入捕获
- 可根据公式编写直接获取周期和占空比

5.捕获所涉函数整理（按配置顺序）

a.使能TIM，GPIO时钟

使能TIM，GPIO时钟

```
1   RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3,ENABLE);
2   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
```

b.设置TIM基本单元

设置TIM基本单元

```
TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;
TIM_TimeBaseInitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
TIM_TimeBaseInitStructure.TIM_CounterMode = TIM_CounterMode_Up ;
TIM_TimeBaseInitStructure.TIM_Period = 65536-1;
TIM_TimeBaseInitStructure.TIM_Prescaler = 72-1;
TIM_TimeBaseInitStructure.TIM_RepetitionCounter = 0 ;
TIM_TimeBaseInit(TIM3,&TIM_TimeBaseInitStructure);
```

c.配置定时器IC单元功能

- 定义一个TIM_ICInitTypeDef类型的结构体TIM_ICInitStructure
- 设置结构体内容：TIM_Channel（选择对应的4个输入通道），TIM_ICFilter（捕获的滤波器效果），TIM_ICPolarity（边沿检测的极性选择），TIM_ICPrescale（分频选择，为1不分频，为2触发时间减半），TIM_ICSelection（直连通道，交叉通道）
- 初始化定时器TIM2的IC单元结构

配置IC单元结构

```
TIM_ICInitTypeDef TIM_ICInitStructure;
TIM_ICInitStructure.TIM_Channel= TIM_Channel_1;
TIM_ICInitStructure.TIM_ICFilter = 0xF;
TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Rising;
TIM_ICInitStructure.TIM_ICPrescaler =TIM_ICPSC_DIV1;
TIM_ICInitStructure.TIM_ICSelection =TIM_ICSelection_DirectTI;
TIM_ICInit(TIM3,&TIM_ICInitStructure);
```

d.开启双通道（交叉通道）

交叉通道

```
TIM_PWMIConfig(TIM3,&TIM_ICInitStructure);
```

e.设置主从模式（实现硬件自动化）

主从模式和启动定时器

```
    TIM_SelectInputTrigger(TIM3,TIM_TS_TI1FP1);
    TIM_SelectSlaveMode(TIM3,TIM_SlaveMode_Reset);
    TIM_Cmd(TIM3,ENABLE);
```

f.编写周期和占空比

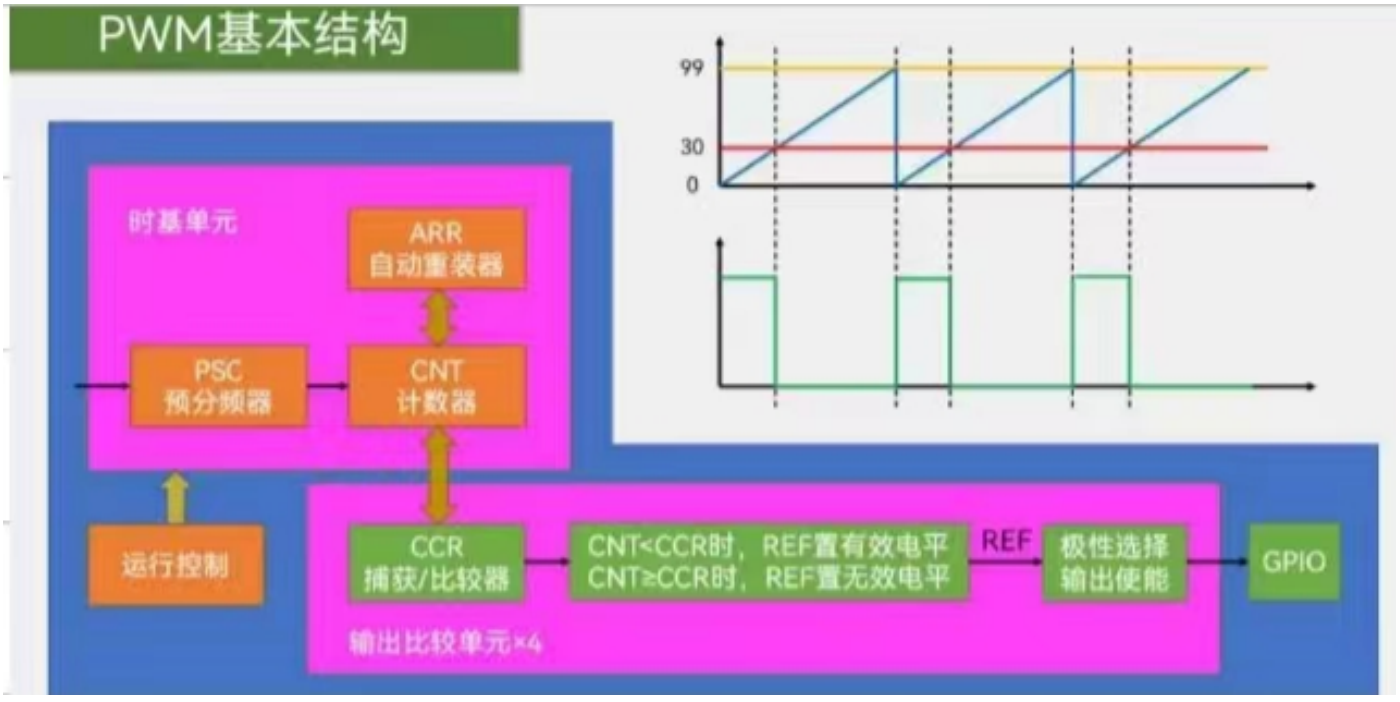
获取周期和占空比

```c
uint32_t IC_GetFreq(void)
{
    return 1000000 / (TIM_GetCapture1(TIM3)+1);
}

 uint32_t IC_GetDuty(void)
{
    return (TIM_GetCapture2(TIM3) + 1) * 100 / (TIM_GetCapture1(TIM3) + 1);
}
```

6.捕获理解操作寄存器

7.实现比较功能步骤（主思路）



- 使能对应功能模块的时钟：TIM，GPIO
- 配置GPIO作为PWM波输出引脚，因此GPIO工作模式为复用推挽模式，即输出电平不取决于GPU，而是外设，这里配置，不需要动AFIO，将对应端口设置复用推挽模式，端口默认复用有TIM即可
- 配置定时器基本时基单元：PSC(预分频器)，ARR(自动重装值)，CNT(计数器)，该单元确定整个PWM波的周期
- 设置定时器TIM的OC（输出比较功能）的配置，PWM工作模式（作用于REF的意义），极性选择（PWM是否翻转），使能操作，设置比较值（CCR）（这个CCR值可以用单独的函数设置，实现PWM可调）

8.比较所涉函数整理（按配置顺序）

a.使能TIM，GPIO时钟

a.使能TIM，GPIO时钟

```
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
```

b.设置TIM基本单元

- 该单元确定整个PWM波的周期，配合比较值CCR，可调控PWM波的高低电平的时间
- 生成PWM波，不需要产生中断，所以开启定时器，不配置中断

c.配置定时器的OC单元功能

- 定义一个TIM_OCInitTypeDef类型的结构体TIM_OCInitSttucture
- 设置结构体内容：TIM_OCMode（设置PWN的工作模式），TIM_OCPolarity（配置输出极性），TIM_OutputState（使能控制端），TIM_Pulse（设置CCR的参数）
- 初始化定时器TIM2的OC单元结构

设置PWM功能的结构体

```
    TIM_OCInitTypeDef TIM_OCInitSttucture;
    TIM_OCInitSttucture.TIM_OCMode = TIM_OCMode_PWM1;
    TIM_OCInitSttucture.TIM_OCPolarity = TIM_OCPolarity_High;
    TIM_OCInitSttucture.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCInitSttucture.TIM_Pulse = 0;
    TIM_OC2Init(TIM2,&TIM_OCInitSttucture);
```

d.设置OC单元的CCR比较值

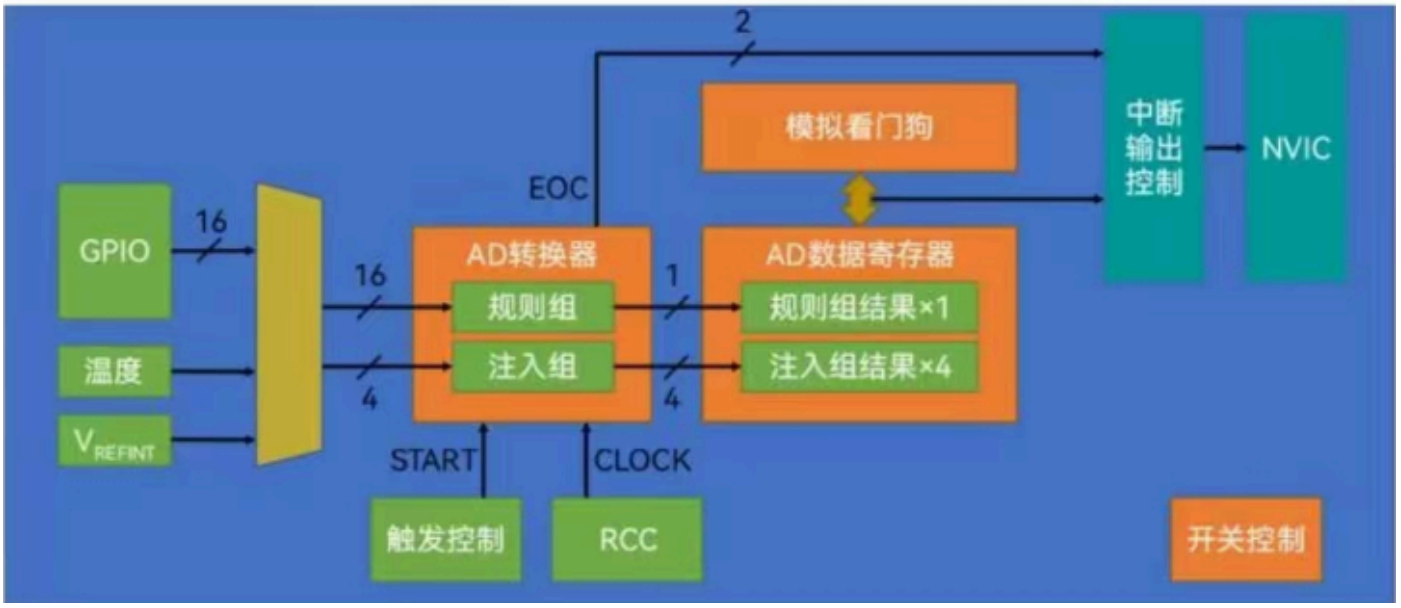
设置CCR值

```
TIM_SetCompare2(TIM2,Compare);
```

9.比较理解操作寄存器

第三：配置AD（模数转化）

1.实现步骤（主思路）



- 使能对应功能模块的时钟：GPIO,ADC
- 这里的GPIO作为模拟电压输入，即不以0/1，而是连续信号进行输入，即GPIO配置成模拟输入
- 配置ADC的内容：先设置ADC的时钟（该时钟是ADC转化的速度重要参数），ADC_Mode（ADC的工作模式），ADC_DataAlign（数据对齐模式），ADC_ExternalTrigConv（转换模式），ADC_ScanConvMode（扫描转换模式），ADC_NbrOfChannel（列表的个数）
- 再配置是以单次非扫描为主体，通过程序实现（单次扫描，多次扫摸等模式），想要扫描（循环调用触发ADC转化的函数即可），想要多通道（先配置一个通道的，通过不断更改规则组的通道表，实现多通道的）
- 该模块需要复位校验

2.所涉函数整理（按配置顺序）

a.使能ADC，GPIO时钟

使能ADC，GPIO时钟

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1,ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
```

b.设置ADC基本单元(设置成单次非扫描模式，规则组菜单为1)

- 定义一个ADC_InitTypeDef类型的结构体ADC_InitStructure

- 设置结构体内容：工作模式（分独立模式），数据对齐模式（左对齐），触发控制的触发源（软件触发模式），转换模式（单次转换模式），扫描转换模式（非扫描模式），列表的个数（1个）
- 初始化ADC单元结构
- 这里的转换模式，扫描转换模式，是可以设置成连续转换模式，扫描模式的，即要改列表个数，和ADC规则组菜单内容的

ADC基本单元

```
  ADC_InitTypeDef ADC_InitStructure;
  ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
  ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
  ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
  ADC_InitStructure.ADC_ScanConvMode = DISABLE;
  ADC_InitStructure.ADC_NbrOfChannel = 1;
  ADC_Init(ADC1, &ADC_InitStructure);
```

c.复位校验

复位校验

```
  ADC_ResetCalibration(ADC1);
  while (ADC_GetResetCalibrationStatus(ADC1) == SET);
  ADC_StartCalibration(ADC1);
  while (ADC_GetCalibrationStatus(ADC1) == SET);
```

d.上电使能ADC

上电使能ADC

```
  ADC_Cmd(ADC1, ENABLE);
```

e.ADC规则组菜单

单通道

```
ADC_RegularChannelConfig(ADC1, ADC_Channel_0, 1, ADC_SampleTime_55Cycles5);
```

多通道

```
uint16_t AD_GetValue(uint8_t ADC_Channel)
{
    ADC_RegularChannelConfig(ADC1, ADC_Channel, 1, ADC_SampleTime_55Cycles5);
```

```
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);
    while (ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
    return ADC_GetConversionValue(ADC1);
}
```

f.触发ADC转化

- 单次非扫描，将ADC_SoftwareStartConvCmd(ADC1, ENABLE);不放在循环里即可
- 扫摸模式，将ADC_SoftwareStartConvCmd(ADC1, ENABLE);放在循环里即可

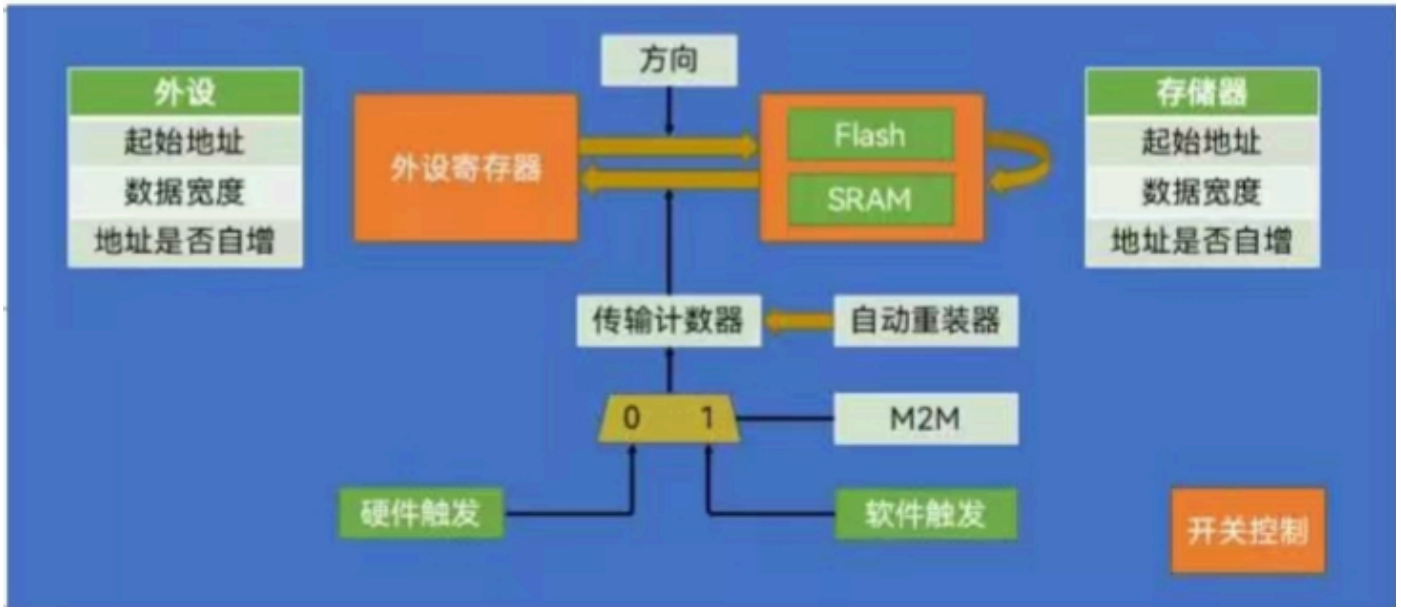
e.获取数字转化结果函数

函数

```
ADC_GetConversionValue(ADC1);
```

第四：配置DMA数据转运

1.实现步骤（主思路）



- 该模块配置基本单元内容，即可
- 配置ADC的结构体内容：DMA_PeripheralBaseAddr（外设站点的基地址(32位的地址)，也就是数据传输点）DMA_PeripheralDataSize（指定数据宽度，以字节方式传输），DMA_PeripheralInc（指定外设地址是否自增）DMA_MemoryBaseAddr（存储站点的初始地址，数据的接受点，类型uint32_t），DMA_MemoryDataSize（指定数据宽度，以字节方式传输），DMA_MemoryInc（指定存储地址是否自增），DMA_DIR（传输方向），

DMA_BufferSize（传输次数），DMA_Mode（判断是否自动重装），DMA_M2M（是否软件触发），DMA_Priority（配置对的优先级）

- 需要注意的，确定好外设和存储的地址，转运方向，设置好转运次数，不需要开启自动重装，因为不断使能，重新设置转运此时，实现自动重装。（这个转运次数，是通过传参进行的）
- 格外注意一下，配置的通到序列

2.所涉函数整理（按配置顺序）

a.使能DMA1时钟

使能DMA1时钟

```
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1,ENABLE);
```

b.设置DMA基本单元

整体配置

```
void MyDMA_Init(uint32_t AddrA,uint32_t AddrB,uint16_t Size)
{
    MyDMA_Size = Size;
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1,ENABLE);
    DMA_InitTypeDef DMA_InitStructure;
    DMA_InitStructure.DMA_PeripheralBaseAddr = AddrA;
    DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;
    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Enable ;
    DMA_InitStructure.DMA_MemoryBaseAddr =AddrB;
    DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Byte;
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
    DMA_InitStructure.DMA_BufferSize = Size;
    DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
    DMA_InitStructure.DMA_M2M = DMA_M2M_Enable;
    DMA_InitStructure.DMA_Priority = DMA_Priority_Medium;
    DMA_Init(DMA1_Channel1,&DMA_InitStructure);
    DMA_Cmd(DMA1_Channel1,DISABLE);
}
```

c.上电使能DMA1

上电使能DMA1

```
1   DMA_Cmd(DMA1_Channel1,DISABLE);
```

d.更改转运次数

更改转运次数

```
1 DMA_SetCurrDataCounter(DMA1_Channel1,MyDMA_Size);
```

e.读取和清除DMA完成标志位

读取DMA完成标志位

```
1 while (DMA_GetFlagStatus(DMA1_FLAG_TC1) == RESET);
```

清除DMA完成标志位

```
1 DMA_ClearFlag(DMA1_FLAG_TC1);
```

f.实现自动重装

自动重装

```
1 Dvoid MyDMA_Transfer(void)
2 {
3     DMA_Cmd(DMA1_Channel1,DISABLE);
4     DMA_SetCurrDataCounter(DMA1_Channel1,MyDMA_Size);
5     DMA_Cmd(DMA1_Channel1,ENABLE);
6     while(DMA_GetFlagStatus(DMA1_FLAG_TC1 == RESET));
7     DMA_ClearFlag(DMA1_FLAG_TC1);
8 }
```