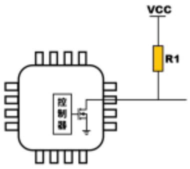
107.HAL库常见函数

2025年2月20日 20:28

1.GPIO闪烁

```c
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
    HAL_GPIO_WritePin(LED_Pin_GPIO_Port, LED_Pin_Pin, GPIO_PIN_SET);
    HAL_Delay(100);
    HAL_GPIO_WritePin(LED_Pin_GPIO_Port, LED_Pin_Pin, GPIO_PIN_RESET);
    HAL_Delay(100);
}
/* USER CODE END 3 */
```

上拉电阻如何取值



控制开关：R1取10K-100K

PWM或者通讯：R1取1K-10K

上拉电阻作用
辅助浮空状态输出高电平。

上拉电阻取值
太小的话漏电流大，太大的话驱动能力弱。

- P1.点亮LED【HAL库复现江协全部STM...
- P2.按键控制LED【HAL库复现江协全部...
- P3.移植江协OLED显示屏【HAL库复现...
- P4.红外对射传感器中断【HAL库复现...
- P5.旋转编码器中断【HAL库复现江协...
- P6.定时器定时中断【HAL库复现江协...
- P7.定时器外部时钟源【HAL库复现江...
- P8.PWM驱动呼吸灯【HAL库复现江协...
- P9.PWM驱动舵机【HAL库复现江协全...
- P10.PWM直流电机【HAL库复现江协...
- P11.定时器输入捕获测频率【HAL库复...
- P12.定时器输入捕获测占空比【HAL库...
- P13.定时器ResetMode【HAL库复现江...
- P14.定时器的门/触发/单脉冲模式【HA...
- 15.定时器编码器接口【HAL库复现江...
- 16.ADC非扫描单次和连续模式【HAL...
- 17.ADC单次非扫描获取多通道【HAL...
- 18.ADC模拟看门狗【HAL库复现江协...
- 19.定时器触发ADC【HAL库复现江协...
- 20.DMA转运数据【HAL库复现江协全...
- 21.扫描模式下的ADC和DMA【HAL库...
- 22.USART发送【HAL库复现江协全部S...
- 23.USART接收【HAL库复现江协全部S...
- 24.USART收发数据包(中断和DMA)【H...
- 25.软件和硬件I2C操作MPU6050【HA...
- 25.软件和硬件I2C操作MPU6050【HA...
- 27.软件SPI操作W25Q64【HAL库复现...
- 28.硬件SPI操作W25Q64【HAL库复现...
- 29.BKP备份寄存器【HAL库复现江协全...
- 30.RTC实时时钟【HAL库复现江协全部...
- 31.PWR睡眠模式【HAL库复现江协全...
- 32.PWR停止模式【HAL库复现江协全...
- 33.PWR待机模式【HAL库复现江协全...
- 33.PWR待机模式【HAL库复现江协全...
- 35.窗口看门狗【HAL库复现江协全部S...
- 36.读写内部Flash闪存【HAL库复现江...

2.Key按键控制

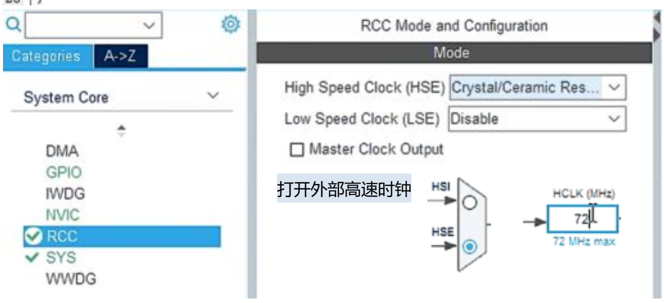
```c
GPIO_PinState ReadButtonState(){
    GPIO_PinState mValue=GPIO_PIN_SET;
    if(HAL_GPIO_ReadPin(LED_Input_GPIO_Port, LED_Input_Pin)==GPIO_PIN_RESET){
        HAL_Delay(20);
        mValue=GPIO_PIN_RESET;
    }
    return mValue;
}
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
    while (1)
    {
      /* USER CODE END WHILE */

      /* USER CODE BEGIN 3 */
        if(ReadButtonState()==GPIO_PIN_RESET){
            HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
        }else {
            HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
        }
    }
    /* USER CODE END 3 */
}
```

3.移植替换JUST的Oled代码块

main.c | OLED.c | OLED.h | OLED_Font.h

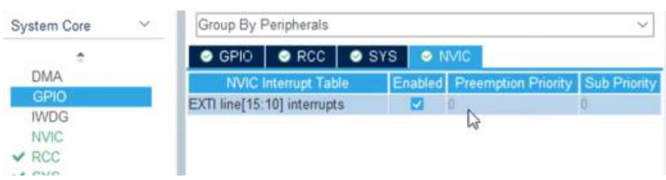
```c
#include "stm32f10x.h"
#include "OLED_Font.h"

/*引脚配置*/
#define OLED_W_SCL(x)        GPIO_WriteBit(GPIOB, GPIO_Pin_8, (BitAction)(x))
#define OLED_W_SDA(x)        GPIO_WriteBit(GPIOB, GPIO_Pin_9, (BitAction)(x))

/*引脚初始化*/
void OLED_I2C_Init(void)
{
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

    GPIO_InitTypeDef GPIO_InitStructure;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_OD;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    OLED_W_SCL(1);
    OLED_W_SDA(1);
}
```

```c
#include "gpio.h"
#include "OLED_Font.h"

/*引脚配置*/
//#define OLED_W_SCL(x)     GPIO_WriteBit(GPIOB, GPIO_Pin_8, (BitAction)(x))
//#define OLED_W_SDA(x)     GPIO_WriteBit(GPIOB, GPIO_Pin_9, (BitAction)(x))
void OLED_W_SCL(GPIO_PinState x){
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8,x);
}
void OLED_W_SDA(GPIO_PinState x){
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9,x);
}

/*引脚初始化*/
void OLED_I2C_Init(void)
{
//    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
//
//  GPIO_InitTypeDef GPIO_InitStructure;
//  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_OD;
//  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
//  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
//  GPIO_Init(GPIOB, &GPIO_InitStructure);
//  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
//  GPIO_Init(GPIOB, &GPIO_InitStructure);
     MX_GPIO_Init();

    OLED_W_SCL(1);
    OLED_W_SDA(1);
}
```

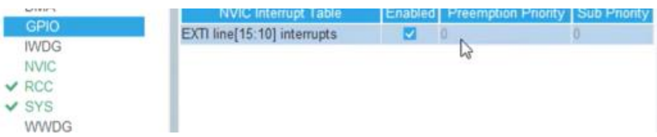


4.红外对射传感器中断

```c
void EXTI15_10_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI15_10_IRQn 0 */

  /* USER CODE END EXTI15_10_IRQn 0 */
    if(__HAL_GPIO_EXTI_GET_FLAG(SensorCount_Pin))
    {
        HAL_GPIO_EXTI_IRQHandler(SensorCount_Pin);
  /* USER CODE BEGIN EXTI15_10_IRQn 1 */
        hhRetCount+=1;
```



```
    /* USER CODE END EXTI15_10_IRQn 0 */
      if(__HAL_GPIO_EXTI_GET_FLAG(SensorCount_Pin))
      {
          HAL_GPIO_EXTI_IRQHandler(SensorCount_Pin);
    /* USER CODE BEGIN EXTI15_10_IRQn 1 */
          hhRetCount+=1;
      }
    /* USER CODE END EXTI15_10_IRQn 1 */
  }

  /* USER CODE BEGIN 1 */
  uint16_t GetCountRet(void){
      return hhRetCount;
  }
```

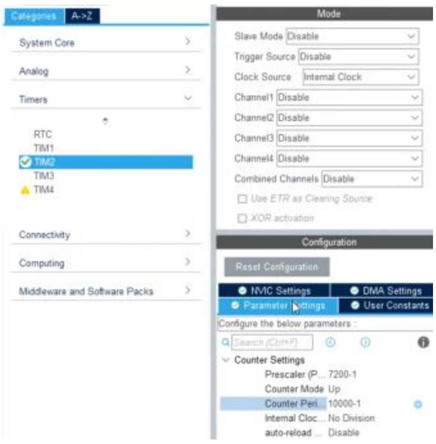


5.旋转编码器中断

```
void EXTI0_IRQHandler(void)
{
    if(__HAL_GPIO_EXTI_GET_FLAG(PB0_EXIT_Pin)){
    if(HAL_GPIO_ReadPin(PB1_EXIT_GPIO_Port,
PB1_EXIT_Pin)==GPIO_PIN_RESET){
            hhRetCount++;
            }
  HAL_GPIO_EXTI_IRQHandler(PB0_EXIT_Pin);
    }
}
void EXTI1_IRQHandler(void)
{
    if(__HAL_GPIO_EXTI_GET_FLAG(PB1_EXIT_Pin)){
    if(HAL_GPIO_ReadPin(PB0_EXIT_GPIO_Port,
PB0_EXIT_Pin)==GPIO_PIN_RESET){
            hhRetCount--;
            }
  HAL_GPIO_EXTI_IRQHandler(PB1_EXIT_Pin);
    }
}
```

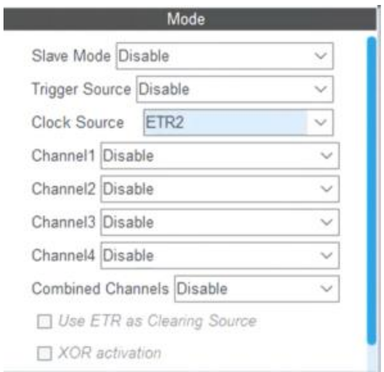
6.定时器定时中断

```
void SystemClock_Config(void);
/* USER CODE BEGIN PFP */
uint16_t hhRetCount=0;
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
        if(htim==&htim2){
            hhRetCount+=1;
        }
}
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  MX_TIM2_Init();
  /* USER CODE BEGIN 2 */
    OLED_Init();
    OLED_ShowString(1, 1, "count:");
       OLED_ShowString(2, 1, "CNT:");
    HAL_TIM_Base_Start_IT(&htim2);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
      OLED_ShowNum(1, 7, hhRetCount,5);
      OLED_ShowNum(2,5, __HAL_TIM_GET_COUNTER(&htim2),5);

  }
  /* USER CODE END 3 */
}
```



7.定时器的外部时钟源选择

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    if(htim==&htim2){
            hhRetCount+=1;
        }
}
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  MX_GPIO_Init();
  MX_TIM2_Init();
  /* USER CODE BEGIN 2 */
    OLED_Init();
       __HAL_TIM_CLEAR_FLAG(&htim2, TIM_FLAG_UPDATE);
    HAL_TIM_Base_Start_IT(&htim2);
```



```c
MX_GPIO_Init();
MX_TIM2_Init();
/* USER CODE BEGIN 2 */
  OLED_Init();
    __HAL_TIM_CLEAR_FLAG(&htim2, TIM_FLAG_UPDATE);
  HAL_TIM_Base_Start_IT(&htim2);
  OLED_ShowString(1, 1, "count:");
     OLED_ShowString(2, 1, "CNT:");
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
    OLED_ShowNum(1, 7, hhRetCount,5);
    OLED_ShowNum(2,5, __HAL_TIM_GET_COUNTER(&htim2),5);
}
/* USER CODE END 3 */
}
```

8.PWM驱动呼吸灯

已知定时器的时基单元使用的是内部RCC时钟，假如需要生成一个1KHZ，占空比为50%，分辨率为1%的PWM波形。求自动重装寄存器ARR、输出比较寄存器CRR、时基单元预分频器PSC的值。

- 分辨率为1%再由分辨率公式可得ARR的值为99
- 占空比为50%再由占空比公式可得CCR的值为50
- 频率为1KHZ,而定时器是使用RCC内部使用所以CK_PSC是72MHZ。从而算得PSC为719

```c
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM2_Init();
   OLED_Init();
   OLED_Clear();
   HAL_TIM_PWM_Start(&htim2,TIM_CHANNEL_1);
  while (1)
  {
        for(int i=0;i<100;i++){
            __HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_1,i);
            HAL_Delay(10);
        }
        for(int i=0;i<100;i++){
            __HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_1,100-
i);
            HAL_Delay(10);
        }
  }
}
```

9.PWM驱动舵机

```c
uint16_t GetCCRFromAngle(float InputAngle){
    float Ret=InputAngle/180 * 2000 +500;
   return Ret;
}
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  MX_GPIO_Init();
  MX_TIM2_Init();
  /* USER CODE BEGIN 2 */
    OLED_Init();
    HAL_TIM_PWM_Start(&htim2,TIM_CHANNEL_2);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
      for(int i=0;i<180;i++){
            __HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_2,GetCCRFromAngle(i));
            OLED_ShowNum(1,7,i,5);
            HAL_Delay(10);
        }
        for(int i=180;i>0;i--){
            __HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_2,GetCCRFromAngle(i));
            OLED_ShowNum(1,7,i,5);
            HAL_Delay(10);
        }
  }
  /* USER CODE END 3 */
}
```

10.PWM直流电机

```c
void SetSpeed(int8_t Speed){
        if (Speed >= 0)
    {
        /*GPIO_SetBits(GPIOA, GPIO_Pin_4);
        GPIO_ResetBits(GPIOA, GPIO_Pin_5);
        hh_SetTimrCCR(Speed);*/
        HAL_GPIO_WritePin(GPIOA, PA4_Motor_Pin, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOA, PA5_Motor_Pin, GPIO_PIN_RESET);
        __HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_3,Speed);
    }
    else
    {
        /*GPIO_ResetBits(GPIOA, GPIO_Pin_4);
        GPIO_SetBits(GPIOA, GPIO_Pin_5);
        hh_SetTimrCCR(-Speed);*/
        HAL_GPIO_WritePin(GPIOA, PA4_Motor_Pin,GPIO_PIN_RESET );
        HAL_GPIO_WritePin(GPIOA, PA5_Motor_Pin, GPIO_PIN_SET);
        __HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_3,-Speed);
    }
}

int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick.
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM2_Init();
  /* USER CODE BEGIN 2 */
    OLED_Init();
    HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_3);
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
        //从0速度正转到最大速度
        for(int i=0;i<100;i++){
            SetSpeed(i);
            HAL_Delay(10);
        }
        //从正转最大速度到0
        for(int i=0;i<100;i++){
            SetSpeed(100-i);
            HAL_Delay(10);
        }
        //从0速度正转到最大速度
        for(int i=0;i<100;i++){
            SetSpeed(-i);
            HAL_Delay(10);
        }
        //从正转最大速度到0
        for(int i=0;i<100;i++){
            SetSpeed(i-100);
            HAL_Delay(10);
        }
```

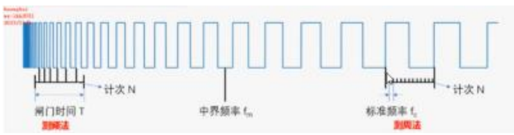
	引脚	定义
	VM	驱动电压输入端（4.5-10V）
	VCC	逻辑电平输入端（2.7-5.5V）
	GND	电源地端
	STBY	正常工作/待机状态控制输入端
1路电机	PWMA	PWM信号输入端
	AIN1	电机控制模式输入端
	AIN2	
	A01	电机驱动输出端
	A02	
2路电机	PWMB	PWM信号输入端
	BIN1	电机控制模式输入端
	BIN2	
	B01	电机驱动输出端
	B02	

输入				输出		
IN1	IN2	PWM	STBY	O1	O2	模式状态
H	H	H/L	H	L	L	制动
L	H	H	H	L	H	反转
L	H	L	H	L	L	制动
H	L	H	H	H	L	正转
H	L	L	H	L	L	制动
L	L	H	H	OFF		停止
H/L	H/L	H/L	L	OFF		待机

11.定时器输入捕获测频率

```c
__weak void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(htim);

  /* NOTE : This function should not be modified, when the callback is needed,
            the HAL_TIM_IC_CaptureCallback could be implemented in the user file
   */
}
```

弱定义(函数值可以修改)




```c
int32_t freq;
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim){
    if (htim->Instance == TIM3) {
        uint32_t capture =HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1)+1;//这里跟keil中一样也要加1
        uint32_t aa=HAL_RCC_GetPCLK1Freq();
        uint32_t bb=(htim->Instance->PSC+1);
        uint32_t cc=HAL_RCC_GetPCLK1Freq()/(htim->Instance->PSC+1);
        freq=1000000/capture;
    }
}
```

```c
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM2_Init();
  MX_TIM3_Init();
  HAL_TIM_PWM_Start(&htim2,TIM_CHANNEL_1);
  HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_1);
   OLED_Init();
   OLED_Clear();
   OLED_ShowString(1,1,"Freq:000000Hz");

  while (1)
```

1. 设置TI1FP1为ResetMode,即清空计数
2. 使用内部时钟
3. 通道1设置为输入捕获模式
4. 设置预分频器为72
5. 自动重装寄存器为65535
6. 输入捕获上升沿

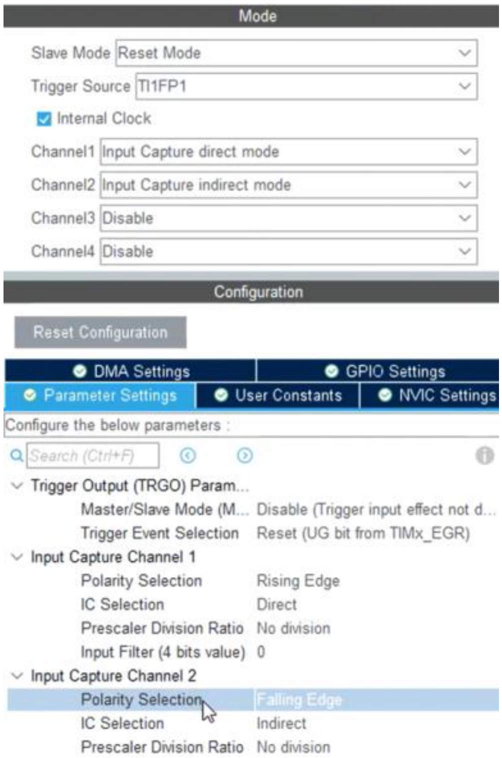
```c
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM2_Init();
  MX_TIM3_Init();
  HAL_TIM_PWM_Start(&htim2,TIM_CHANNEL_1);
  HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_1);
   OLED_Init();
   OLED_Clear();
   OLED_ShowString(1,1,"Freq:000000Hz");

  while (1)
  {
      OLED_ShowSignedNum(1,6,freq,5);
  }
}
```

4. 设置预分频器为72
5. 自动重装寄存器为65535
6. 输入捕获上升沿

12.定时器输入捕获测占空比

```c
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim){
    if (htim->Instance == TIM3) {
        if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1) {
            // 上升沿触发的中断

             capture =HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1)+1;
            uint32_t aa=HAL_RCC_GetPCLK1Freq();
            uint32_t bb=(htim->Instance->PSC+1);
            uint32_t cc=HAL_RCC_GetPCLK1Freq()/(htim->Instance->PSC+1);
            freq=1000000/capture;
        } else if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_2) {
            // 下降沿触发的中断
            uint32_t capture2 =HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_2)+1;
            DutyResult=capture2 *100 / capture;
        }
    }
}
```

```c
  MX_GPIO_Init();
  MX_TIM2_Init();
  MX_TIM3_Init();
  /* USER CODE BEGIN 2 */
    OLED_Init();
    HAL_TIM_PWM_Start(&htim2,TIM_CHANNEL_1);
   HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_1);
   HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_2);
   OLED_Init();
   OLED_Clear();
   OLED_ShowString(1,1,"Freq:000000Hz");
    OLED_ShowString(2,1,"Duty:00%");
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
      OLED_ShowSignedNum(1,6,freq,5);
      OLED_ShowNum(2,6,DutyResult,2);
  }
  /* USER CODE END 3 */
}
```



13.定时器ResetMode

```c
void SystemClock_Config(void);
/* USER CODE BEGIN PFP */
uint32_t hhRetCount=0;
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    if(htim==&htim3){
        if(__HAL_TIM_GET_FLAG(htim,TIM_FLAG_TRIGGER)==SET){
            //从模式Reset的中断
            __HAL_TIM_CLEAR_FLAG(htim,TIM_FLAG_TRIGGER);//需手动清空标志位
        }else{
            //自动重载的中断
            hhRetCount+=1;
        }
    }
}
```

```c
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM3_Init();
  /* USER CODE BEGIN 2 */
    OLED_Init();
```

1. 设置从模式为Reset模式
2. 使用的触发源是TI1FP1,但选择这个后右边芯片图中的PA6引脚会自动创建标签
3. 使用内部时钟
4. 设置PSC为36000-1，自动重装寄存器为2000-1.计数器从0增加到2000，溢出周期为1秒。
5. 设置从模式的Reset模式用上升沿
6. 滤波值设置15

```c
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM3_Init();
  /* USER CODE BEGIN 2 */
    OLED_Init();
    HAL_TIM_Base_Start_IT(&htim3);
       OLED_ShowString(2, 1, "CNT:");
       OLED_ShowString(1, 1, "count:");
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
      OLED_ShowNum(1, 7, hhRetCount,5);
      OLED_ShowNum(2,5, __HAL_TIM_GET_COUNTER(&htim3),5);
  }
  /* USER CODE END 3 */
}
```

6. 滤波值设置15

14.定时器的门/触发/单脉冲模式

```c
void SystemClock_Config(void);
/* USER CODE BEGIN PFP */
uint32_t hhRetCount=0;
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    if(htim==&htim3){
            hhRetCount+=1;
    }
}
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM3_Init();
  /* USER CODE BEGIN 2 */    __HAL_TIM_CLEAR_FLAG(&htim3, TIM_FLAG_UPDATE);
    OLED_Init();
    HAL_TIM_Base_Start_IT(&htim3);
       OLED_ShowString(2, 1, "CNT:");
       OLED_ShowString(1, 1, "count:");
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
      OLED_ShowNum(1, 7, hhRetCount,5);
      OLED_ShowNum(2,5, __HAL_TIM_GET_COUNTER(&htim3),5);
  }
  /* USER CODE END 3 */
}
```

可以控制寄存器的暂停和启动

一、门模式

1、从模式下的门模式

它的作用是TIF接收到信号时(高电平或低电平)，会暂停计数器计数。会设置触发器中断标志位，但是**不会触发触发器中断。**

2、实例

1、设置定时器相关参数

1. 设置从模式为Gated模式
2. 使用的触发源是TI1FP1,但选择这个后右边芯片图中的PA6引脚会自动创建标签
3. 使用内部时钟
4. 设置PSC为36000-1，自动重装寄存器为2000-1.计数器从0增加到2000，溢出周期为1秒。
5. 设置从模式的Gated为低电平有效
6. 滤波值设置15

二、触发(Trigger)模式

1、从模式下的触发(Trigger)模式

门模式可以在任何时候控制定时器的停止/启动计数，Trigger模式可以控制定时器的启动。若定时器从模式设置了这个模式，定时器执行 `HAL_TIM_Base_Start_IT(&htim3);` 后，是不会计数的。需要从模式TIF检测到指定信号才会计数。它只生效一次，即定时器启动计数后，后面就会一直计数到重装载，自动清空又重新计数。

2、实例

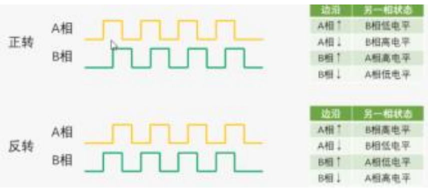
在门模式基础上只需要修改TriggerSource为TI1FP1即可。其他包括代码都不用修改

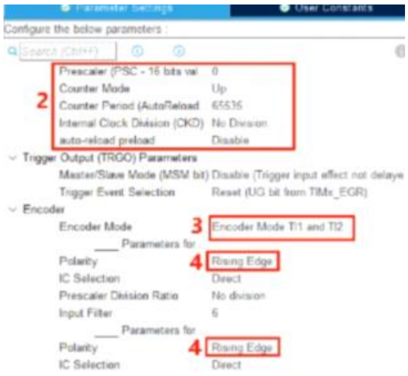
14.定时器编码器接口

```c
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM3_Init();
  HAL_TIM_Encoder_Start(&htim3, TIM_CHANNEL_ALL);
   OLED_Init();
   OLED_Clear();
   OLED_ShowString(2, 1, "CNT:");
  while (1)
  {
      OLED_ShowSignedNum(2,5,__HAL_TIM_GET_COUNTER(&htim3),5);
      HAL_Delay(100);
```

1. 打开编码器模式
2. 设置定时器的相关参数。由于是检测速度，所以这里就不分频，尽可能让定时器快点。计数器设置最大，防止输入信号频率太高溢出。
3. 设置计数模式，TI1计数和TI2计数
4. 设置信号不反转

```c
    while (1)
    {
        OLED_ShowSignedNum(2,5,__HAL_TIM_GET_COUNTER(&htim3),5);
        HAL_Delay(100);
    }
}
```






```c
int16_t hhGetEncoderSpeedCountAndReset(){
    int16_t Tmp;
    Tmp=__HAL_TIM_GET_COUNTER(&htim3);
    __HAL_TIM_SET_COUNTER(&htim3,0);
    return Tmp;
}
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM3_Init();
  HAL_TIM_Encoder_Start(&htim3, TIM_CHANNEL_ALL);
   OLED_Init();
   OLED_Clear();
   OLED_ShowString(2, 1, "CNT:");
  while (1)
  {
      OLED_ShowSignedNum(2,5,hhGetEncoderSpeedCountAndReset(),5);
      HAL_Delay(1000);
  }
}
```

15.ADC非扫描单次和连续模式

五、ADC的4种模式

1、单次转换非扫描模式

2、连续转换非扫描模式

3、单次转换扫描模式

4、连续转换扫描模式

- 单次/连续：转换后是否继续自动转换，单次就不继续自动转换
- 扫描/非扫描：是否支持多通道，扫描就可以多通道

可知，单次非扫描就是给指令就开始转换，转换需要等待完成，转换完成后需要再次给指令且一次只能转换一个通道。


```c
HAL_StatusTypeDef HalState;
uint16_t Ret;
uint16_t StartAndGetOneResult(){
    //[illegible] ADC_SoftwareStartConvCmd(ADC1,ENABLE);
    HAL_ADC_Start(&hadc1);
    //[illegible] while(ADC_GetFlagStatus(ADC1,ADC_FLAG_EOC)==RESET);
    HalState= HAL_ADC_PollForConversion(&hadc1, 10);//[illegible]
    if(HalState == HAL_OK){
            Ret=HAL_ADC_GetValue(&hadc1);
    } else{
        Ret=0;
    }
    //HAL_ADC_Stop(&hadc1);
    return  Ret;
}
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
```

用电位器模拟测电压

```c
SystemClock_Config();

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_ADC1_Init();
/* USER CODE BEGIN 2 */
  OLED_Init();
  HAL_ADCEx_Calibration_Start(&hadc1);
  OLED_ShowString(1,1,"ADValue:");
  OLED_ShowString(2,1,"Voltage:0.00V");
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
      ADValue= StartAndGetOneResult();
      OLED_ShowNum(1,9,ADValue,4);
      Voltage=(float) ADValue/ 4095 *3.3;
      OLED_ShowNum(2,9,(uint32_t)Voltage,1);
      OLED_ShowNum(2,11,((uint16_t)(Voltage * 100)) % 100,2);
      HAL_Delay(100);

}
```

Continuous Conversion Mode Enabled

上为单次非扫描模式 | | 下为连续非扫描

```c
HAL_StatusTypeDef HalState;
uint16_t Ret;
uint16_t StartAndGetOneResult(){
    Ret=HAL_ADC_GetValue(&hadc1);
    return  Ret;
}
uint16_t ADValue;
float Voltage;
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{

    OLED_Init();
    HAL_ADCEx_Calibration_Start(&hadc1);
    HAL_ADC_Start(&hadc1);//只执行一次开始转换
    OLED_ShowString(1,1,"ADValue:");
    OLED_ShowString(2,1,"Voltage:0.00V");
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
        ADValue= StartAndGetOneResult();
        OLED_ShowNum(1,9,ADValue,4);
        Voltage=(float) ADValue/ 4095 *3.3;
        OLED_ShowNum(2,9,(uint32_t)Voltage,1);
        OLED_ShowNum(2,11,((uint16_t)(Voltage * 100)) % 100,2);
        HAL_Delay(100);

  }
  /* USER CODE END 3 */
}
```

17.ADC单次非扫描获取多通道

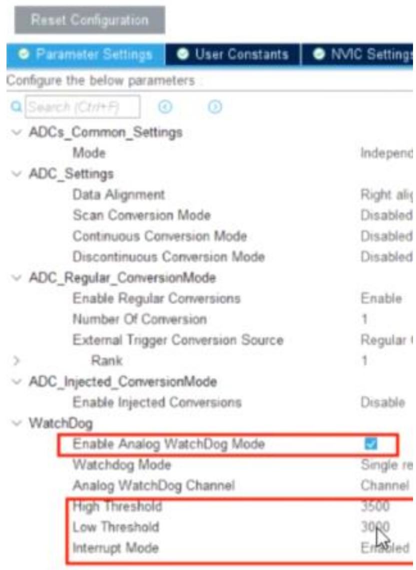
```c
HAL_StatusTypeDef HalState;
uint16_t Ret;
ADC_ChannelConfTypeDef sConfig = {0};
uint16_t StartAndGetOneResult(uint32_t ADC_Channel){

      sConfig.Channel = ADC_Channel;
      sConfig.Rank = ADC_REGULAR_RANK_1;
      sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
      if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
      {
            Ret=0;
      }
    //开始转换对应标准库：ADC_SoftwareStartConvCmd(ADC1,ENABLE);
    HAL_ADC_Start(&hadc1);
    //等待转换完成对应标准库：while(ADC_GetFlagStatus(ADC1,ADC_FLAG_EOC)==RESET);
    HalState= HAL_ADC_PollForConversion(&hadc1, 10);//第二个参数是等待超时时间
    if(HalState == HAL_OK){
```

```c
HAL_StatusTypeDef HalState;
uint16_t Ret;
ADC_ChannelConfTypeDef sConfig = {0};
uint16_t StartAndGetOneResult(uint32_t ADC_Channel){

      sConfig.Channel = ADC_Channel;
      sConfig.Rank = ADC_REGULAR_RANK_1;
      sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
      if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
      {
            Ret=0;
      }
    //开始转换对应标准库：ADC_SoftwareStartConvCmd(ADC1,ENABLE);
    HAL_ADC_Start(&hadc1);
    //等待转换完成对应标准库：while(ADC_GetFlagStatus(ADC1,ADC_FLAG_EOC)==RESET);
    HalState= HAL_ADC_PollForConversion(&hadc1, 10);//第二个参数是等待超时时间
    if(HalState == HAL_OK){
          Ret=HAL_ADC_GetValue(&hadc1);
    } else{
        Ret=0;
    }
    //HAL_ADC_Stop(&hadc1);
    return  Ret;
}
uint16_t ADValue;
float Voltage;
uint16_t AD1;
uint16_t AD2;
uint16_t AD3;
uint16_t AD4;
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  MX_GPIO_Init();
  MX_ADC1_Init();
  /* USER CODE BEGIN 2 */
    OLED_Init();
    HAL_ADCEx_Calibration_Start(&hadc1);
    OLED_ShowString(1,1,"AD1:");
    OLED_ShowString(2,1,"AD2:");
    OLED_ShowString(3,1,"AD3:");
    OLED_ShowString(4,1,"AD4:");
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
        AD1= StartAndGetOneResult(ADC_CHANNEL_0);
        AD2= StartAndGetOneResult(ADC_CHANNEL_1);
        AD3= StartAndGetOneResult(ADC_CHANNEL_2);
        AD4= StartAndGetOneResult(ADC_CHANNEL_3);
        OLED_ShowNum(1,5,AD1,4);
        OLED_ShowNum(2,5,AD2,4);
        OLED_ShowNum(3,5,AD3,4);
        OLED_ShowNum(4,5,AD4,4);
        HAL_Delay(100);
  }
  /* USER CODE END 3 */
}
```

18.ADC模拟看门狗

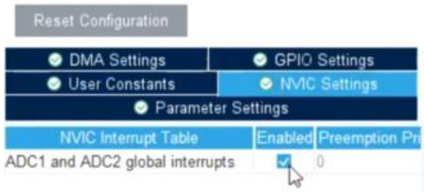
ADC的模拟看门狗，其功能时指定一个阈值范围。假如ADC转换出来的结果不在这个范围内，看门狗就会"叫"(触发中断)

```c
void ADC1_2_IRQHandler(void)
{
  /* USER CODE BEGIN ADC1_2_IRQn 0 */
      if (__HAL_ADC_GET_FLAG(&hadc1, ADC_FLAG_AWD)) {
            hhRetCount++;
            __HAL_ADC_CLEAR_FLAG(&hadc1, ADC_FLAG_AWD);
      }
  /* USER CODE END ADC1_2_IRQn 0 */
  HAL_ADC_IRQHandler(&hadc1);
  /* USER CODE BEGIN ADC1_2_IRQn 1 */

  /* USER CODE END ADC1_2_IRQn 1 */
}

/* USER CODE BEGIN 1 */
uint16_t GetCountRet(void){
      return hhRetCount;
}
```



main.c中重写回调函数就可以了，上面的中断函数重写可以不使用

```c
__weak void HAL_ADC_LevelOutOfWindowCallback(ADC_HandleTypeDef* hadc)
```




```c
HAL_StatusTypeDef HalState;
uint16_t Ret;
uint16_t StartAndGetOneResult(){
    //启动转换对应标准库：ADC_SoftwareStartConvCmd(ADC1,ENABLE);
    HAL_ADC_Start(&hadc1);
    //等待转换完成对应标准库：while(ADC_GetFlagStatus(ADC1,ADC_FLAG_EOC)==RESET);
    HalState= HAL_ADC_PollForConversion(&hadc1, 10);//第二个参数是等待超时时间
    if(HalState == HAL_OK){
        Ret=HAL_ADC_GetValue(&hadc1);
    } else{
        Ret=0;
    }
    //HAL_ADC_Stop(&hadc1);
    return  Ret;
}
uint16_t ADValue;
float Voltage;
uint16_t hhRetCount;
void HAL_ADC_LevelOutOfWindowCallback(ADC_HandleTypeDef* hadc){
      if (__HAL_ADC_GET_FLAG(&hadc1, ADC_FLAG_AWD)) {
          hhRetCount++;
          __HAL_ADC_CLEAR_FLAG(&hadc1, ADC_FLAG_AWD);
      }
}
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  uint16_t ADValue;
  float Voltage;
  MX_GPIO_Init();
  MX_ADC1_Init();
   OLED_Init();
   OLED_Clear();
   HAL_ADCEx_Calibration_Start(&hadc1);
    OLED_ShowString(1,1,"ADValue:");
    OLED_ShowString(2, 1, "count:");
  while (1)
  {
        ADValue= StartAndGetOneResult();
        OLED_ShowNum(1,9,ADValue,4);
        OLED_ShowNum(2, 7,GetCountRet(),5);
        HAL_Delay(100);
  }
}
```

19.定时器触发ADC

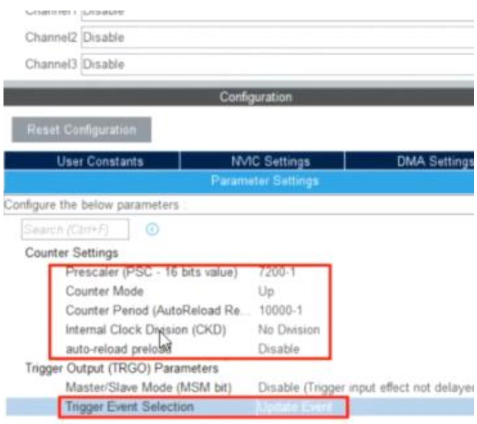
```c
HAL_StatusTypeDef HalState;
uint16_t Ret;
uint16_t StartAndGetOneResult(){
        HalState= HAL_ADC_PollForConversion(&hadc1, 10);//第二个参数是等
        if(HalState == HAL_OK){
              Ret=HAL_ADC_GetValue(&hadc1);
        }
    return  Ret;
}
uint16_t ADValue;
float Voltage;
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
```

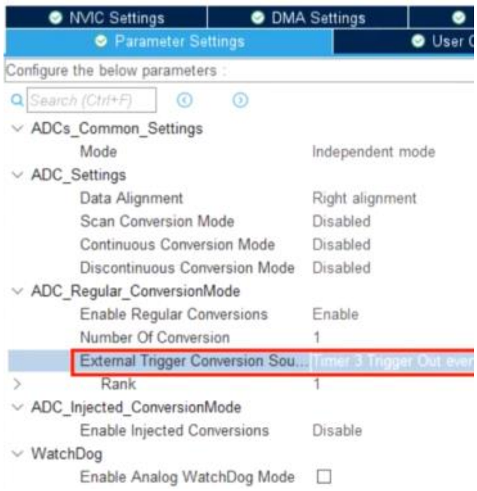
前面的例子都是使用软件触发，下面演示使用TIM3的TRGO事件来触发ADC。

1、配置定时器

1. 使用内部时钟
2. 设置定时器相关参数，定时1秒钟。对应Keil中的 `TIM_TimeBaseInit(TIM3,&hhTimeBaseStruc);`
3. 打开更新事件。对应Keil中的 `TIM_SelectOutputTrigger(TIM3, TIM_TRGOSource_Update);`

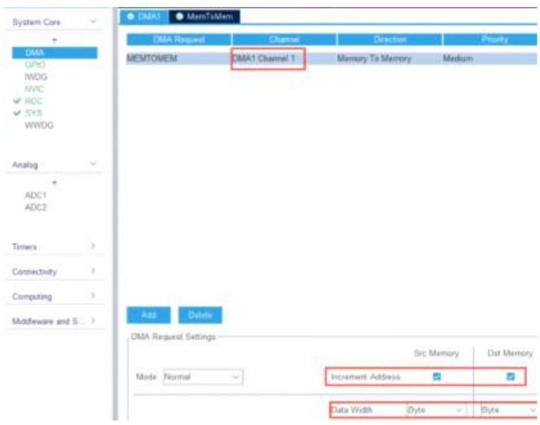
```c
HAL_Init();
SystemClock_Config();
MX_GPIO_Init();
MX_ADC1_Init();
MX_TIM3_Init();
/* USER CODE BEGIN 2 */
 OLED_Init();
 OLED_Clear();
 HAL_ADCEx_Calibration_Start(&hadc1);
 HAL_TIM_Base_Start(&htim3); // 启动TIM3
 HAL_ADC_Start(&hadc1);
  OLED_ShowString(1,1,"ADValue:");
  OLED_ShowString(2,1,"Voltage:0.00V");
/* USER CODE END 2 */

/* USER CODE BEGIN WHILE */
while (1)
{
      ADValue= StartAndGetOneResult();
      OLED_ShowNum(1,9,ADValue,4);
      Voltage=(float) ADValue/ 4095 *3.3;
      OLED_ShowNum(2,9,(uint32_t)Voltage,1);
      OLED_ShowNum(2,11,((uint16_t)(Voltage * 100)) % 100,2);
      HAL_Delay(100);
}
}
```





20.DMA转运数据

```c
int main(void)
{
  HAL_Init();

  SystemClock_Config();
  MX_GPIO_Init();
  MX_DMA_Init();
  /* USER CODE BEGIN 2 */
   OLED_Init();
   OLED_Clear();
    //显示转运前数据DataA、DataB
    uint8_t DataA[]={0x01,0x02,0x03,0x04};
    uint8_t DataB[]={0,0,0,0};
    OLED_ShowHexNum(1,1,DataA[0],2);
    OLED_ShowHexNum(1,4,DataA[1],2);
    OLED_ShowHexNum(1,7,DataA[2],2);
    OLED_ShowHexNum(1,10,DataA[3],2);

    OLED_ShowHexNum(2,1,DataB[0],2);
    OLED_ShowHexNum(2,4,DataB[1],2);
    OLED_ShowHexNum(2,7,DataB[2],2);
    OLED_ShowHexNum(2,10,DataB[3],2);

     // 启动DMA传输
      HAL_DMA_Start(&hdma_memtomem_dma1_channel1, (ui

      // 等待传输完成
      HAL_DMA_PollForTransfer(&hdma_memtomem_dma1_cha
HAL_MAX_DELAY);

        OLED_ShowHexNum(1,1,DataA[0],2);
        OLED_ShowHexNum(1,4,DataA[1],2);
        OLED_ShowHexNum(1,7,DataA[2],2);
        OLED_ShowHexNum(1,10,DataA[3],2);
```



```c
    // 启动DMA传输
    HAL_DMA_Start(&hdma_memtomem_dma1_channel1, (ui

    // 等待传输完成
    HAL_DMA_PollForTransfer(&hdma_memtomem_dma1_cha
HAL_MAX_DELAY);

        OLED_ShowHexNum(1,1,DataA[0],2);
        OLED_ShowHexNum(1,4,DataA[1],2);
        OLED_ShowHexNum(1,7,DataA[2],2);
        OLED_ShowHexNum(1,10,DataA[3],2);

        OLED_ShowHexNum(2,1,DataB[0],2);
        OLED_ShowHexNum(2,4,DataB[1],2);
        OLED_ShowHexNum(2,7,DataB[2],2);
        OLED_ShowHexNum(2,10,DataB[3],2);
  while (1)
  {
  }
}
```

上面是单次DMA转运数据 || 下面是单次非扫描AD通道，DMA转运，仅在main函数之前增加一个单通道数切换的函数

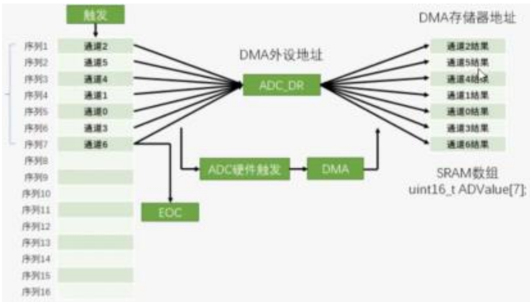
```c
void ReDMA(){
    __HAL_DMA_DISABLE(&hdma_memtomem_dma1_channel1);
    hdma_memtomem_dma1_channel1.Instance->CNDTR=4;
      __HAL_DMA_ENABLE(&hdma_memtomem_dma1_channel1);
    // 等待传输完成
    HAL_DMA_PollForTransfer(&hdma_memtomem_dma1_channel1, HAL_DMA_FULL_TRANSFER, HAL_MAX_DELAY);

}
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  MX_GPIO_Init();
  MX_DMA_Init();
  /* USER CODE BEGIN 2 */
    OLED_Init();
  /* USER CODE END 2 */
    //显示转运前的数据DataA. DataB
        uint8_t DataA[]={0x01,0x02,0x03,0x04};
        uint8_t DataB[]={0,0,0,0};

        // 启动DMA传输
        HAL_DMA_Start(&hdma_memtomem_dma1_channel1, (uint32_t)&DataA, (uint32_t)&DataB, 4);

            /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    DataA[0]++;
    DataA[1]++;
    DataA[2]++;
    DataA[3]++;
    //显示DataA
    OLED_ShowHexNum(2,1,DataA[0],2);
    OLED_ShowHexNum(2,4,DataA[1],2);
    OLED_ShowHexNum(2,7,DataA[2],2);
    OLED_ShowHexNum(2,10,DataA[3],2);
    //显示DataB
    OLED_ShowHexNum(4,1,DataB[0],2);
    OLED_ShowHexNum(4,4,DataB[1],2);
    OLED_ShowHexNum(4,7,DataB[2],2);
    OLED_ShowHexNum(4,10,DataB[3],2);
    HAL_Delay(1000);
    //执行转运数据
    ReDMA();
    //显示DataA
    OLED_ShowHexNum(2,1,DataA[0],2);
    OLED_ShowHexNum(2,4,DataA[1],2);
    OLED_ShowHexNum(2,7,DataA[2],2);
    OLED_ShowHexNum(2,10,DataA[3],2);
    //显示DataB
    OLED_ShowHexNum(4,1,DataB[0],2);
    OLED_ShowHexNum(4,4,DataB[1],2);
    OLED_ShowHexNum(4,7,DataB[2],2);
    OLED_ShowHexNum(4,10,DataB[3],2);
    HAL_Delay(1000);
  }
  /* USER CODE END 3 */
}
```

21.扫描模式下的ADC和DMA

前面的ADC理论知识介绍到，在规则组中的多次(扫描或非扫描)转换ADC出来的结果只有一个寄存器存放，假如不及时使用DMA转运出来数据就会覆盖掉。现在就是利用DMA转运多次ADC的结果。下面代码是在单次扫

21.扫描模式下的ADC和DMA

前面的ADC理论知识介绍到，在规则组中的多次(扫描或非扫描)转换ADC出来的结果只有一个寄存器存放，假如不及时使用DMA转运出来数据就会覆盖掉。现在就是利用DMA转运多次ADC的结果。下面代码是在单次扫描ADC的基础上修改




```
Core
  Inc
    adc.h
```

```c
/* USER CODE BEGIN Private defines */
extern DMA_HandleTypeDef hdma_adc1;
/* USER CODE END Private defines */
```

```c
void StartAndGetResult(){
    //重新设置计数器，让它实现再次转运
    hhStatue=HAL_ADC_Start_DMA(&hadc1,(uint32_t*) AD_Value, 4);//开始ADC转换
    if(hhStatue!=HAL_OK){
        AD_Value[0]=0;
        AD_Value[1]=0;
        AD_Value[2]=0;
        AD_Value[3]=0;
    }
    while(hdma_adc1.State!=HAL_DMA_STATE_READY);//等待DMA转运数据完成
    HAL_ADC_Stop_DMA(&hadc1); //停止ADC转换
}
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_ADC1_Init();
  /* USER CODE BEGIN 2 */
   OLED_Init();
    HAL_ADCEx_Calibration_Start(&hadc1);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
      StartAndGetResult();
  }
  /* USER CODE END 3 */
}
```

```c
  OLED_Init();
   HAL_ADCEx_Calibration_Start(&hadc1);
      OLED_ShowString(1,1,"AD1:");
      OLED_ShowString(2,1,"AD2:");
      OLED_ShowString(3,1,"AD3:");
      OLED_ShowString(4,1,"AD4:");
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
    StartAndGetResult();
    StartAndGetResult();
      AD1= AD_Value[0];
      AD2= AD_Value[1];
      AD3= AD_Value[2];
      AD4= AD_Value[3];
  OLED_ShowNum(1,5,AD1,4);
```

1. 使用Normal模式，对应keil中的`DMA_InitStruct.DMA_Mode=DMA_Mode_Normal;`，转换一次完成后需要手动进行相关操作才能进行下次转运
2. 设置目标地址Memory存储结果为递增模式对应keil中的DMA_InitStruct.DMA_MemoryInc=DMA_MemoryInc_Enable;`，数据源地址已经自动设置为不可选择递增模式
3. 由于ADC的结果是16位的，所以使用半字。对应keil中的`DMA_InitStruct.DMA_PeripheralDataSize=DMA_PeripheralDataSize_HalfWord;`

22.USART 发送

```c
void hhSerialSendByte(uint8_t Byte){
    HAL_UART_Transmit(&huart1, &Byte, 1, HAL_MAX_DELAY);
}

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART1_UART_Init();
  /* USER CODE BEGIN 2 */
    OLED_Init();
    hhSerialSendByte('A');
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
```


```c
void hhSerialSendByte(uint8_t Byte){
    HAL_UART_Transmit(&huart1, &Byte, 1, HAL_MAX_DELAY);
}
void hhSerialSendArray(uint8_t *Array,uint16_t Length){
    for(uint16_t i=0;i<Length;i++){
        hhSerialSendByte(Array[i]);
    }
}
void hhSerialSendString(char * mString){
    for(uint16_t i=0;mString[i]!='\0';i++){
        hhSerialSendByte(mString[i]);
    }
}
```

重定向Serial_Printf

```c
#include <stdarg.h>
#include "stdio.h"

void Serial_Printf(char *format, ...)
{
    char String[100];
    va_list arg;
    va_start(arg, format);
    vsprintf(String, format, arg);
    va_end(arg);
```

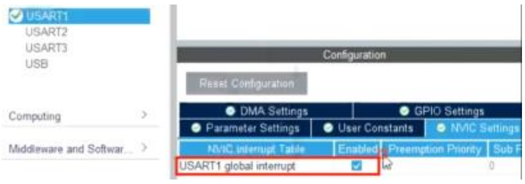
```c
    OLED_Init();
    hhSerialSendByte('A');
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

  }
}
```

```c
    va_list arg;
    va_start(arg, format);
    vsprintf(String, format, arg);
    va_end(arg);
    hhSerialSendString(String);
}
```

```c
Serial_Printf("\r\nNum=%f", 222.33);
Serial_Printf("\r\nNum=%d", 20231202);
```

23.USART接收

```c
__weak void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(huart);
  /* NOTE: This function should not be modified, when the callback is need
           the HAL_UART_RxCpltCallback could be implemented in the user fi
   */
}
```



重写回调函数

```c
//1.单字节发送
void hhSerialSendByte(uint8_t Byte){
    HAL_UART_Transmit(&huart1, &Byte, 1, HAL_MAX_DELAY);
}
uint8_t Serial_RxFlag;
uint8_t Serial_GetRxFlag(void)
{
    if (Serial_RxFlag == 1)
    {
        Serial_RxFlag = 0;
        return 1;
    }
    return 0;
}
uint8_t ByteRecv;
//接收中断函数
HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
    if (huart == &huart1){
        Serial_RxFlag=1;//已接收标志位，说明已经接收完一次
        HAL_UART_Receive_IT(&huart1, &ByteRecv, 1);//接收了一次后需要再次打开接收中断为
收做准备
    }
}
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART1_UART_Init();
   OLED_Init();
   OLED_Clear();
   HAL_UART_Receive_IT(&huart1, &ByteRecv, 1);//启动中断接收一个字节
    OLED_ShowString(1, 1, "RxData:");
  while (1)
  {
        if (Serial_GetRxFlag() == 1)
        {
            hhSerialSendByte(ByteRecv);//将接收到的数据重新发送返回给电脑串口
            OLED_ShowHexNum(1, 8, ByteRecv, 2);
        }
  }
}
```

23.USART收发数据包(中断和DMA)

1.发送数据包代码

1.发送数据包代码

```
uint8_t Serial_TxPacket[4];
void hhSerial_SendPacket(void)
{
    hhSerialSendByte(0xFF);
    hhSerialSendArray(Serial_TxPacket, 4);
    hhSerialSendByte(0xFE);
}
```

2.接收数据包代码

```
uint8_t Serial_TxPacket[4];
void hhSerial_SendPacket(void)
{
    hhSerialSendByte(0xFF);
    hhSerialSendArray(Serial_TxPacket, 4);
    hhSerialSendByte(0xFE);
}

uint8_t Serial_RxPacket[4];
void USART1_IRQHandler(void)
{
    static uint8_t RxState = 0;
    static uint8_t pRxPacket = 0;
    if (USART_GetITStatus(USART1, USART_IT_RXNE) == SET)
    {
       uint8_t RxData = USART_ReceiveData(USART1);

       if (RxState == 0)
       {
           if (RxData == 0xFF)
           {
               RxState = 1;
               pRxPacket = 0;
           }
       }
       else if (RxState == 1)
       {
           Serial_RxPacket[pRxPacket] = RxData;
           pRxPacket ++;
           if (pRxPacket >= 4)
           {
               RxState = 2;
           }
       }
        else if (RxState == 2)
        {
            if (RxData == 0xFE)
            {
                RxState = 0;
                Serial_RxFlag = 1;
            }
        }

        USART_ClearITPendingBit(USART1, USART_IT_RXNE);
    }
}
```

```
#include "stm32f10x.h"                  // Device header
#include "Delay.h"
#include "OLED.h"
#include "Serial.h"
#include <stdio.h>
uint8_t RxData;
int main(void)

    OLED_Init();
    InitAllUSART();
    OLED_ShowString(3, 1, "RxPacket");
    //发送
    Serial_TxPacket[0] = 0x01;
    Serial_TxPacket[1] = 0x02;
    Serial_TxPacket[2] = 0x03;
    Serial_TxPacket[3] = 0x04;
    Serial_SendPacket();
    while(1)
    {
        if (Serial_GetRxFlag() == 1)//接收数据包成功后显示
        {
            OLED_ShowHexNum(4, 1, Serial_RxPacket[0], 2);
            OLED_ShowHexNum(4, 4, Serial_RxPacket[1], 2);
            OLED_ShowHexNum(4, 7, Serial_RxPacket[2], 2);
            OLED_ShowHexNum(4, 10, Serial_RxPacket[3], 2);
        }
    }
}
```

> 需要前面的设置usart等相关参数和打开中断的基础上

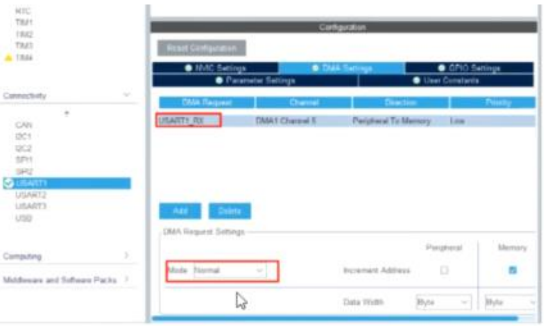
使用Hal库中的扩展函数`HAL_UARTEx_ReceiveToIdle_IT`，接收到串口空闲，然后会触发中断：`HAL_UARTEx_RxEventCallback`

```
char Serial_RxPacket[100];
void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart, uint16_t Size){
    if (huart == &huart1){
        Serial_RxPacket[Size]='\0';//由于C语言中的字符串都必须以'\0'为结束标志的，所以接收完后需
这行
        Serial_RxFlag=1;
            HAL_UARTEx_ReceiveToIdle_IT(&huart1,  Serial_RxPacket,
sizeof(Serial_RxPacket));
    }
}
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART1_UART_Init();
  OLED_Init();
  OLED_Clear();
  HAL_UARTEx_ReceiveToIdle_IT(&huart1,  Serial_RxPacket, sizeof(Serial_RxPacket));//
参数表示最大接收长度
   OLED_ShowString(3, 1, "RxPacket");
   //发送
   hhSerialSendString("你好世界！");
  while (1)
  {
      if (Serial_GetRxFlag() == 1){
          OLED_ShowString(4, 1, "                  "); //用于清空屏幕
          OLED_ShowString(4, 1, Serial_RxPacket);
      }
  }
}
```

以上是俩调用中断函数接收数据包的代码块 || 以下是使用"DMA转运"接收数据包的代码块

要把前面的`HAL_UARTEx_ReceiveToIdle_IT`替换成`HAL_UARTEx_ReceiveToIdle_DMA`即可。还需要特别注意的是，使用DMA时有时可能会触发两次`HAL_UART_RxCpltCallback`中断函数，第一次时接收到一半数据时，第二次是完整接收时。所以需要使用`__HAL_DMA_DISABLE_IT(&hdma_usart1_rx,DMA_IT_HT);`把接收到一半时的这种情况屏蔽掉。

```
extern DMA_HandleTypeDef hdma_usart1_rx;//由于hdma_usart1_rx对象在
usart.c文件中，假如main.c文件需要使用它就需要加这一行，即告诉编译器，加入main.c
文件中没有hdma_usart1_rx时，就去其他地方寻找

char Serial_RxPacket[100];
HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    if (huart == &huart1)
    {
            static uint8_t RxState = 0;
            static uint8_t pRxPacket = 0;
        if (RxState == 0)
        {
            if (ByteRecv == '@' && Serial_RxFlag == 0)//把数据读取走后
才接收下一个数据包防止粘包问题
            {
                RxState = 1;
                pRxPacket = 0;
            }
        }
        else if (RxState == 1)
        {
            if (ByteRecv == '\r')
            {
                RxState = 2;
            }
            else
            {
                Serial_RxPacket[pRxPacket] = ByteRecv;
                pRxPacket ++;
```



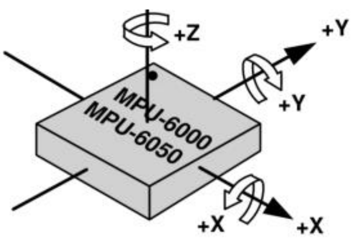
```
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_USART1_UART_Init();
  /* USER CODE BEGIN 2 */
   OLED_Init();
   OLED_Clear();
   HAL_UARTEx_ReceiveToIdle_DMA(&huart1,  Serial_RxPacket,
sizeof(Serial_RxPacket));
    __HAL_DMA_DISABLE_IT(&hdma_usart1_rx,DMA_IT_HT);
    OLED_ShowString(3, 1, "RxPacket");
    hhSerialSendString("你好世界");
  while (1)
  {
        if (Serial_GetRxFlag() == 1){
            OLED_ShowString(4, 1, "                  "); //用于清空
屏幕
            OLED_ShowString(4, 1, Serial_RxPacket);
        }
  }
}
```

```c
            }
        }
        else if (RxState == 1)
        {
            if (ByteRecv == '\r')
            {
                RxState = 2;
            }
            else
            {
                Serial_RxPacket[pRxPacket] = ByteRecv;
                pRxPacket ++;
            }
        }
        else if (RxState == 2)
        {
            if (ByteRecv == '\n')
            {
                RxState = 0;
                Serial_RxPacket[pRxPacket] = '\0';//字符串必须带'\0'作为结束标志
                Serial_RxFlag = 1;
            }
        }
        HAL_UART_Receive_IT(&huart1, &ByteRecv, 1);
    }
}

void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart, uint16_t Size){
    if (huart == &huart1){
        Serial_RxPacket[Size]='\0';
        Serial_RxFlag=1;
        HAL_UARTEx_ReceiveToIdle_DMA(&huart1,  Serial_RxPacket, sizeof(Serial_RxPacket));
        __HAL_DMA_DISABLE_IT(&hdma_usart1_rx,DMA_IT_HT);
    }
}
```

```c
    OLED_ShowString(3, 1, "RxPacket");
    hhSerialSendString("你好世界");
  while (1)
  {
      if (Serial_GetRxFlag() == 1){
          OLED_ShowString(4, 1, "                "); //用于清空屏幕
          OLED_ShowString(4, 1, Serial_RxPacket);
      }
  }
}
```

25.软件和硬件I2C操作MPU6050

```c
#include "gpio.h"
//对SCL写
void MyI2C_W_SCL(GPIO_PinState x)
{
    HAL_GPIO_WritePin(GPIOB, hhSCL_Pin, x);
//  Delay_us(10);
    //HAL_Delay(1);
}
//对SDA写
void MyI2C_W_SDA(GPIO_PinState x)
{
    HAL_GPIO_WritePin(GPIOB, hhSDA_Pin, x);
//  Delay_us(10);
    //HAL_Delay(1);
}
//对SDA读
uint8_t MyI2C_R_SDA(void)
{
    uint8_t BitValue;
//  Delay_us(10);
    if (HAL_GPIO_ReadPin(GPIOB, hhSDA_Pin)==GPIO_PIN_RESET){
        BitValue=0;
    }else{
        BitValue=1;
    }
//  HAL_Delay(1);
    return BitValue;
}
```



```c
void MyI2C_Init(void)
{

//  GPIO_SetBits(GPIOB, GPIO_Pin_10 | GPIO_Pin_11);
    HAL_GPIO_WritePin(GPIOB, hhSCL_Pin|hhSDA_Pin, GPIO_PIN_SET);

}
```

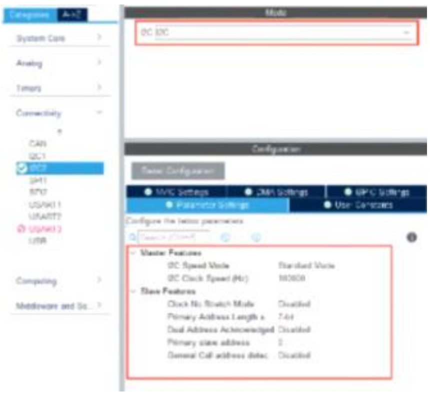
以上是软件I2C操作 || 下面是硬件I2C

```c
uint8_t MPU6050_ReadReg(uint8_t RegAddress)
{
    uint8_t Data;
    //1.******'λ
    MyI2C_Start();
    //2.Ψ.+ д
    MyI2C_SendByte(MPU6050_ADDRESS);
    MyI2C_ReceiveAck();
    //3.******д*|*.
    MyI2C_SendByte(RegAddress);
    MyI2C_ReceiveAck();

    //▩***'λ
    MyI2C_Start();
    //Ψ.**
    MyI2C_SendByte(MPU6050_ADDRESS | 0x01);//▩▩▩▩1λ*|*1********* 1101 0001
    MyI2C_ReceiveAck();
    Data = MyI2C_ReceiveByte();
    MyI2C_SendAck(1);//▩▩▩▩▩0λ**0*㐌ƒ**j***T************
    //*
    MyI2C_Stop();

    return Data;
}
```

```c
uint8_t MPU6050_ReadReg(uint8_t RegAddress)
{
    uint8_t Data;
//  //1.******'λ
//  MyI2C_Start();
//  //2.Ψ.+ д
//  MyI2C_SendByte(MPU6050_ADDRESS);
//  MyI2C_ReceiveAck();
//  //3.******д*|*.
//  MyI2C_SendByte(RegAddress);
//  MyI2C_ReceiveAck();
//
//  //▩***'λ
//  MyI2C_Start();
//  //Ψ.**
//  MyI2C_SendByte(MPU6050_ADDRESS | 0x01);//▩▩▩▩1λ*|*1********* 1101 0001
//  MyI2C_ReceiveAck();
//  Data = MyI2C_ReceiveByte();
//  MyI2C_SendAck(1);//▩▩▩▩▩0λ**0*㐌ƒ**j***T************
//  //*
//  MyI2C_Stop();
    HAL_I2C_Mem_Read(&hi2c2, MPU6050_ADDRESS, RegAddress, I2C_MEMADD_SIZE_8BIT, &Data, 1, HAL_MAX_
    return Data;
}
```

```c
void MPU6050_WriteReg(uint8_t RegAddress, uint8_t Data)
{
    MyI2C_Start();//1.******'λ
    MyI2C_SendByte(MPU6050_ADDRESS);//2.****Ψ**+д
    MyI2C_ReceiveAck();//3.****Ö**λ.*****?*******
    MyI2C_SendByte(RegAddress);//4.******Cд*|*.
    MyI2C_ReceiveAck();//5.****Ö**λ.*****?*******
    MyI2C_SendByte(Data);//6.******Cд******
    MyI2C_ReceiveAck();//7.****Ö**λ.*****?*******
    MyI2C_Stop();//8.**********
}
```

```c
void MPU6050_WriteReg(uint8_t RegAddress, uint8_t Data)
{
//  MyI2C_Start();//1.******'λ
//  MyI2C_SendByte(MPU6050_ADDRESS);//2.****Ψ**+д
//  MyI2C_ReceiveAck();//3.****Ö**λ.*****?*******
//  MyI2C_SendByte(RegAddress);//4.******Cд*|*.
//  MyI2C_ReceiveAck();//5.****Ö**λ.*****?*******
//  MyI2C_SendByte(Data);//6.******Cд******
//  MyI2C_ReceiveAck();//7.****Ö**λ.*****?*******
//  MyI2C_Stop();//8.**********
    HAL_I2C_Mem_Write(&hi2c2, MPU6050_ADDRESS, RegAddress, I2C_MEMADD_SIZE_8BIT, &Data, 1, HAL_MAX
}
```



```c
void MPU6050_Init(void)
{
//  MyI2C_Init();
    MPU6050_WriteReg(MPU6050_PWR_MGMT_1, 0x01);//解除睡眠模式，并且使用X轴
    MPU6050_WriteReg(MPU6050_PWR_MGMT_2, 0x00);//所有轴不需要待机
    MPU6050_WriteReg(MPU6050_SMPLRT_DIV, 0x09);//设置为10分频
    MPU6050_WriteReg(MPU6050_CONFIG, 0x06);//设置低通滤波值
    MPU6050_WriteReg(MPU6050_GYRO_CONFIG, 0x18);//陀螺仪设置量程为最大量程
    MPU6050_WriteReg(MPU6050_ACCEL_CONFIG, 0x18);//加速度计设置量程为最大
}
```

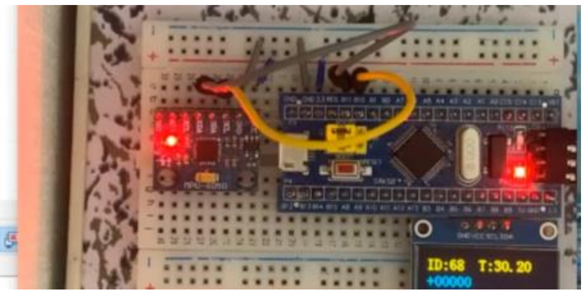
26.软件和硬件I2C操作MPU6050

MPU6050DMP_HardI2C (在28.MPU6050D
- Binaries
- Includes
- Core
- Drivers
- Mpu6050Dmp
- Debug
- LED Debug.launch
- MPU6050DMP_HardI2C.ioc
- MPU6050DMP_HardI2C Debug.launch
- Oled Debug.launch
- STM32F103C8TX_FLASH.ld

```c
#include "main.h"

/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

extern I2C_HandleTypeDef hi2c2;

/* USER CODE BEGIN Private defines */

/* USER CODE END Private defines */
```

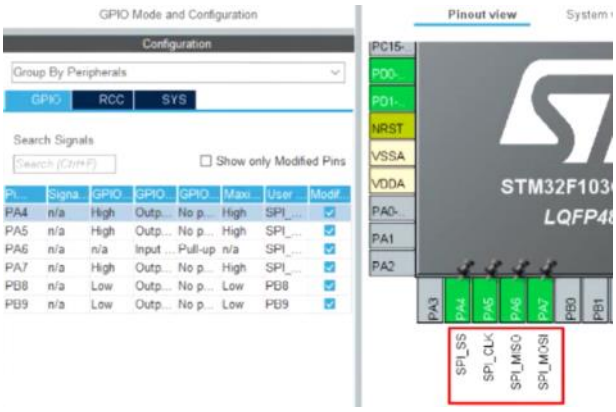
问题 任务 控制台 属性

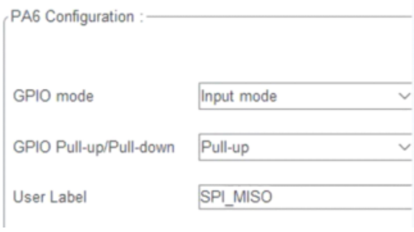
MPU6050DMP_HardI2C Debug [STM32 C/C++ Application] [pid: 144]



27.软件SPI操作W25Q64

```c
#include "gpio.h"
//输出片选-即当前与那个从机通信
void MySPI_W_SS(GPIO_PinState x)
{
//  GPIO_WriteBit(GPIOA, GPIO_Pin_4, (BitAction)BitValue);
    HAL_GPIO_WritePin(GPIOA, SPI_SS_Pin, x);
}
//SCK时钟信号设置
void MySPI_W_SCK(GPIO_PinState x)
{
//  GPIO_WriteBit(GPIOA, GPIO_Pin_5, (BitAction)BitValue);
    HAL_GPIO_WritePin(GPIOA, SPI_CLK_Pin, x);
}
//MOSI主机输出信号设置
void MySPI_W_MOSI(GPIO_PinState x)
{
//  GPIO_WriteBit(GPIOA, GPIO_Pin_7, (BitAction)BitValue);
    HAL_GPIO_WritePin(GPIOA, SPI_MOSI_Pin, x);
 }
 //MISO主机接收信号设置
 uint8_t MySPI_R_MISO(void)
 {
     uint8_t BitValue;
     if (HAL_GPIO_ReadPin(GPIOA, SPI_MISO_Pin)==GPIO_PIN_RESET
         BitValue=0;
     }else{
         BitValue=1;
     }
 }

 void MySPI_Init(void)
 {
//  //使用的是
//  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
//  //《42、SPI通讯协议理论知识.md》中硬件电路规定SPI输入的引脚为上拉、
//  //PA4/PA5/PA7分别为SPI的SS/CLK/MOSI这几个都是输出的引脚设置为推
//  GPIO_InitTypeDef GPIO_InitStructure;
//  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
//  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_5 |
//  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
//  GPIO_Init(GPIOA, &GPIO_InitStructure);
//  //PA6是MISO输入引脚，设置为上拉输入
//  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
//  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
//  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    MySPI_W_SS(1);//初始化时不与任何从机通信
    MySPI_W_SCK(0);//这里使用模式0，初始化后SCK时钟信号是低电平
 }
```



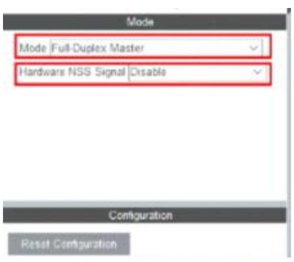


28.硬件SPI操作W25Q64

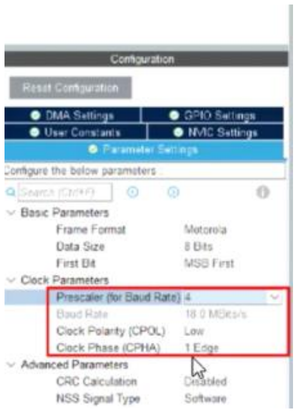
```c
void MySPI_Init(void)
{
//  //使用的是
//  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
//  //《42、SPI通讯协议理论知识.md》中硬件电路规定SPI输入的引脚为上拉、输出的引脚为推挽输出
//  //PA4/PA5/PA7分别为SPI的SS/CLK/MOSI这几个都是输出的引脚设置为推挽输出
//  GPIO_InitTypeDef GPIO_InitStructure;
//  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
//  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_5 | GPIO_Pin_7;
//  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
//  GPIO_Init(GPIOA, &GPIO_InitStructure);
//  //PA6是MISO输入引脚，设置为上拉输入
//  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
//  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
//  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
//  GPIO_Init(GPIOA, &GPIO_InitStructure);

    MySPI_W_SS(1);//初始化时不与任何从机通信

}
//起始信号
void MySPI_Start(void)
{
    MySPI_W_SS(0);
}
//终止信号
void MySPI_Stop(void)
{
```

- Mode：使用主机模式Full-Duplex Master
- 不使用硬件NSS，对应标准库中的`SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;//使用软件NSS`
- 预分频器系数选择4
- CPOL/CPHA设置：使用模式0，对应标准库中的代码

```c
    SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;//四种模式的模式选择
    SPI_InitStructure.SPI_CPHA = SPI_CPHA_1Edge;
```



```c
    MySPI_W_SS(0);
}
//终止信号
void MySPI_Stop(void)
{
    MySPI_W_SS(1);
}

//模式0 交换一个字节
uint8_t MySPI_SwapByteMode0(uint8_t ByteSend)
{
    //0x80、0x40等特殊数来精准提取某一位，详细可参考《39、I2C通信协议理论知识.md》中的 2、主机发送一个字节
//  uint8_t i, ByteReceive = 0x00;
//  for (i = 0; i < 8; i ++)
//  {
//      MySPI_W_MOSI(ByteSend & (0x80 >> i));//取出高位并发送
//      MySPI_W_SCK(1);//给上升沿，便于下面移入数据
//      if (MySPI_R_MISO() == 1){ByteReceive |= (0x80 >> i);}//把高位读读取到并存放到对应的ByteReceive中
//      MySPI_W_SCK(0);//给低电平便于下次移出数据
//  }
    uint8_t  ByteReceive = 0x00;
        HAL_SPI_TransmitReceive(&hspi1, &ByteSend, &ByteReceive, 1, HAL_MAX_DELAY);
    return ByteReceive;
}
```



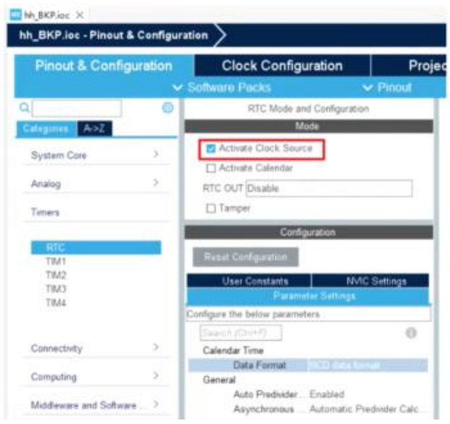
29.BKP备份寄存器

```c
> stm32f1xx_hal_rtc_ex.c
void HAL_RTCEx_BKUPWrite(RTC_HandleTypeDef *hrtc, uint32_t BackupRegister, uint32_t Data)
{
  uint32_t tmp = 0U;

  /* Prevent unused argument(s) compilation warning */
  UNUSED(hrtc);

  /* Check the parameters */
  assert_param(IS_RTC_BKP(BackupRegister));

  tmp = (uint32_t)BKP_BASE;
  tmp += (BackupRegister * 4U);

  *(__IO uint32_t *) tmp = (Data & BKP_DR1_D);
}

uint32_t HAL_RTCEx_BKUPRead(RTC_HandleTypeDef *hrtc, uint32_t BackupRegister)
{
  uint32_t backupregister = 0U;
  uint32_t pvalue = 0U;

  /* Prevent unused argument(s) compilation warning */
  UNUSED(hrtc);

  /* Check the parameters */
  assert_param(IS_RTC_BKP(BackupRegister));

  backupregister = (uint32_t)BKP_BASE;
  backupregister += (BackupRegister * 4U);

  pvalue = (*(__IO uint32_t *)(backupregister)) & BKP_DR1_D;

  /* Read the specified register */
  return pvalue;
}
```

```c
  HAL_RTCEx_BKUPWrite(&hrtc,RTC_BKP_DR1,0x1234);
  uint32_t Ret=
HAL_RTCEx_BKUPRead(&hrtc,RTC_BKP_DR1);
   OLED_ShowHexNum(1, 1, Ret, 4);
```

BKP 寄存器通常与 RTC 一起使用，在CubeIDE中假如需要使用BKP寄存器，需要先打开RTC.

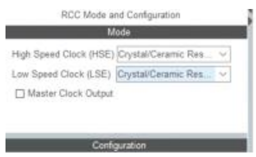
1、打开RTC



30.RTC实时时钟

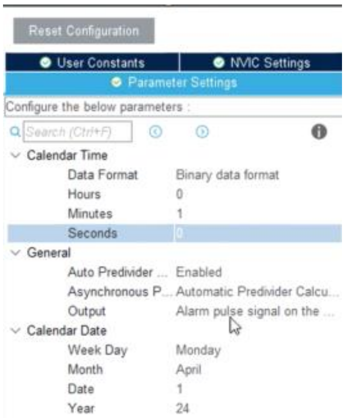
```c
rtc.c
void MX_RTC_Init(void)
{

  /* USER CODE BEGIN RTC_Init 0 */

  /* USER CODE END RTC_Init 0 */

  /* USER CODE BEGIN RTC_Init 1 */

  /* USER CODE END RTC_Init 1 */

  /** Initialize RTC Only
  */
```

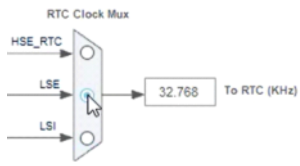
关闭RTC时钟的PC13侵入检测的亮灯功能



```c
/* USER CODE END RTC_Init 1 */

/** Initialize RTC Only
*/
hrtc.Instance = RTC;
hrtc.Init.AsynchPrediv = RTC_AUTO_1_SECOND;
hrtc.Init.OutPut = RTC_OUTPUTSOURCE_NONE;
if (HAL_RTC_Init(&hrtc) != HAL_OK)
{
  Error_Handler();
}
/* USER CODE BEGIN RTC_Init 2 */

/* USER CODE END RTC_Init 2 */

}
```

关闭RTC时钟的PC13侵入检测的亮灯功能






```c
RTC_TimeTypeDef RTC_Time ;
RTC_DateTypeDef RTC_Date;
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_RTC_Init();

   OLED_Init();
   OLED_Clear();
    OLED_ShowString(1, 1, "Date:20XX-XX-XX");
    OLED_ShowString(2, 1, "Time:XX:XX:XX");
    OLED_ShowString(3, 1, "CNT :");
    OLED_ShowString(4, 1, "DIV :");
  while (1)
  {
      HAL_RTC_GetDate(&hrtc, &RTC_Date, RTC_FORMAT_BIN);
      HAL_RTC_GetTime(&hrtc,&RTC_Time, RTC_FORMAT_BIN);
        OLED_ShowNum(1, 8, RTC_Date.Year, 2);
        OLED_ShowNum(1, 11, RTC_Date.Month, 2);
        OLED_ShowNum(1, 14, RTC_Date.Date, 2);
        OLED_ShowNum(2, 6, RTC_Time.Hours, 2);
        OLED_ShowNum(2, 9, RTC_Time.Minutes, 2);
        OLED_ShowNum(2, 12, RTC_Time.Seconds, 2);
        OLED_ShowNum(3, 6, RTC_ReadTimeCounter(&hrtc), 10);
  }
}
```

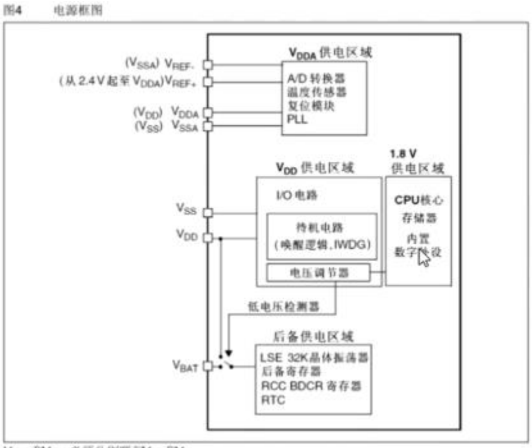
打开自动生成的函数**MX_RTC_Init**中，设置时间加上BKP特殊标志位0x1234重新写入BKP寄存器位0x1234，读取到则不设置。

```c
void MX_RTC_Init(void) {
    RTC_TimeTypeDef sTime = { 0 };
    RTC_DateTypeDef DateToUpdate = { 0 };
    hrtc.Instance = RTC;
    hrtc.Init.AsynchPrediv = RTC_AUTO_1_SECOND;
    hrtc.Init.OutPut = RTC_OUTPUTSOURCE_ALARM;
    if (HAL_RTC_Init(&hrtc) != HAL_OK) {
        Error_Handler();
    }
    if (HAL_RTCEx_BKUPRead(&hrtc, RTC_BKP_DR1) != 0x1234) {
        sTime.Hours = 0;
        sTime.Minutes = 1;
        sTime.Seconds = 0;

        if (HAL_RTC_SetTime(&hrtc, &sTime, RTC_FORMAT_BIN) !
            Error_Handler();
        }
        DateToUpdate.WeekDay = RTC_WEEKDAY_MONDAY;
        DateToUpdate.Month = RTC_MONTH_MAY;
       DateToUpdate.Date = 1;
       DateToUpdate.Year = 24;

        if (HAL_RTC_SetDate(&hrtc, &DateToUpdate, RTC_FORMAT
            Error_Handler();
        }
       HAL_RTCEx_BKUPWrite(&hrtc, RTC_BKP_DR1, 0x1234);
    }
}
```

31.PWR睡眠模式

```c
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART1_UART_Init();
   OLED_Init();
   OLED_Clear();
   HAL_UART_Receive_IT(&huart1, &ByteRecv, 1);//启动中断接收一个字节
    OLED_ShowString(1, 1, "RxData:");
  while (1)
  {
        if (Serial_GetRxFlag() == 1)
        {
            hhSerialSendByte(ByteRecv);//将接收到的数据重新发送返回给电脑串口
            OLED_ShowHexNum(1, 8, ByteRecv, 2);
        }
        OLED_ShowString(2, 1, "Running");
        HAL_Delay(100);
        OLED_ShowString(2, 1, "       ");
        HAL_Delay(100);
        HAL_SuspendTick();//关闭SysTick定时器
        HAL_PWR_EnterSLEEPMode(PWR_MAINREGULATOR_ON,PWR_SLEEPENTRY_WFI);
        HAL_ResumeTick();//恢复SysTick定时器
  }
```

图4 电源框图



V_{DDA}和V_{SSA}必须分别联到V_{DD}和V_{SS}。

```
        HAL_SuspendTick();//关闭SysTick定时器
        HAL_PWR_EnterSLEEPMode(PWR_MAINREGULATOR_ON,PWR_SLEEPENTRY_WFI);
        HAL_ResumeTick();//恢复SysTick定时器
    }
}
```

32.PWR停止模式

```
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  OLED_Init();
  OLED_Clear();
  OLED_ShowString(1, 1, "count:");
  OLED_ShowString(2, 1, "Clk:");
  while (1)
  {
      OLED_ShowNum(1, 7, GetCountRet(),5);
      OLED_ShowNum(2, 5,
HAL_RCC_GetSysClockFreq(),8);//显示时钟
        OLED_ShowString(3, 1, "Running");
        HAL_Delay(500);
        OLED_ShowString(3, 1, "       ");
        HAL_Delay(500);
 HAL_PWR_EnterSTOPMode(PWR_MAINREGULATOR_ON,PWR_ST
OPENTRY_WFI);//进入Stop模式
      SystemClock_Config();//恢复时钟
  }
}
```

33.PWR待机模式

```
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_RTC_Init();
  if( HAL_RTCEx_BKUPRead(&hrtc,RTC_BKP_DR1)!=0x1234){
      MyRTC_SetTime();
      HAL_RTCEx_BKUPWrite(&hrtc,RTC_BKP_DR1,0x1234);
  }
  OLED_Init();
  OLED_Clear();
   OLED_ShowString(1, 1, "Date:XXXX-XX-XX");
   OLED_ShowString(2, 1, "Time:XX:XX:XX");
   OLED_ShowString(3, 1, "CNT :");
   OLED_ShowString(4, 1, "DIV :");
           HAL_PWR_EnableWakeUpPin(PWR_WAKEUP_PIN1);
           __HAL_PWR_CLEAR_FLAG(PWR_FLAG_WU);
  while (1)
  {
        MyRTC_ReadTime();
        OLED_ShowNum(1, 6, MyRTC_Time[0], 4);
            OLED_ShowNum(1, 11, MyRTC_Time[1], 2);
            OLED_ShowNum(1, 14, MyRTC_Time[2], 2);
            OLED_ShowNum(2, 6, MyRTC_Time[3], 2);
            OLED_ShowNum(2, 9, MyRTC_Time[4], 2);
            OLED_ShowNum(2, 12, MyRTC_Time[5], 2);
            OLED_ShowNum(3, 6, RTC_ReadTimeCounter(&hrtc), 10);
            __HAL_RCC_PWR_CLK_ENABLE();
            HAL_PWR_EnterSTANDBYMode();
  }
}
```

1.设置办法

PDDS设置成0，LPDS用于设置电压调节器是否开启，设置成0表示开启，设置成1表示进入低功耗模式。设置完后也需要执行WFI或WFE才能进入停机模式

2.唤醒办法

需要特定的中断WKUP引脚上升沿(PA0引脚)、RTC闹钟事件、外部NRST引脚复位(开发板上的复位按键)、IWDG复位才能唤醒。

3.被关闭的电路

关闭1.8V区域时钟，电压调节器关闭。也就是不仅仅是CPU、外设关了还吧PLL/HSI/HSE也关了。由于断电，其内部寄存器的数据也全部丢失。

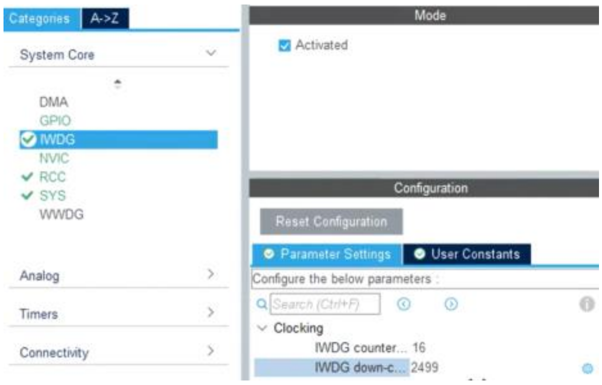
4.功能限制

- 进入待机模式再唤醒后程序从头开始运行。而不是从暂停地方开始运行
- 备份寄存器仍然有备份电源供电
- 在待机模式下，所以IO引脚变成高阻态(浮空输入)

```
        __HAL_RCC_PWR_CLK_ENABLE();
        HAL_PWR_EnterSTANDBYMode();
    }
  }
```

34.独立看门狗

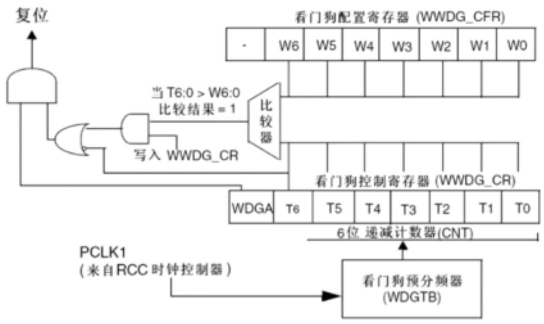
```
    if (__HAL_RCC_GET_FLAG(RCC_FLAG_IWDGRST) != RESET) {
        // IWDG reset flag is set
        OLED_ShowString(2, 1, "IWDGRST");          //OLED闪烁IWDGRST字符串
        HAL_Delay(500);
        OLED_ShowString(2, 1, "       ");
        HAL_Delay(100);
         __HAL_RCC_CLEAR_RESET_FLAGS();
    }
    else{
        OLED_ShowString(3, 1, "RST");              //OLED闪烁RST字符串
        HAL_Delay(500);
        OLED_ShowString(3, 1, "   ");
        HAL_Delay(100);
    }
/* USER CODE END 2 */
```

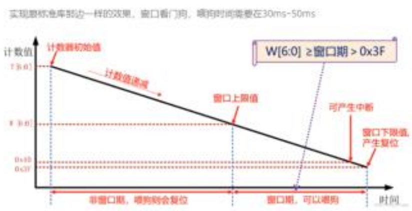


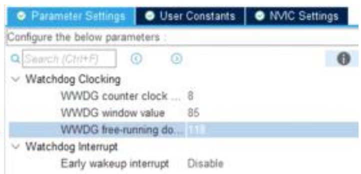

```
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
   HAL_IWDG_Refresh(&hiwdg);//喂狗
   HAL_Delay(1100);
}
```

35.窗口看门狗

```
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  OLED_Init();
  OLED_ShowString(1, 1, "WWDG TEST");
    //获取当前的复位是WWDG造成的复位还是按Rst键复位，这里代码有大耗时，需
MX_WWDG_Init(); 之前
    if (__HAL_RCC_GET_FLAG(RCC_FLAG_WWDGRST) != RESET) {
        // IWDG reset flag is set
        OLED_ShowString(2, 1, "WWDGRST");          //OLED闪烁I
        HAL_Delay(500);
        OLED_ShowString(2, 1, "       ");
        HAL_Delay(100);
         __HAL_RCC_CLEAR_RESET_FLAGS();
    }
    else{
        OLED_ShowString(3, 1, "RST");              //OLED闪烁R
        HAL_Delay(500);
        OLED_ShowString(3, 1, "   ");
        HAL_Delay(100);
    }
  MX_WWDG_Init();
  while (1)
  {
      HAL_Delay(40);
      HAL_WWDG_Refresh(&hwwdg);//喂狗
      HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PI
  }
}
```







1. 窗口看门狗的最晚时间也类似：

TWWDG = TPCLK1 × 4096 × WDGTB预分频系数 × (T[6:0] - 63)

50 = 1/36000 * 4096 * 8 * (T[6:0] - 63) **T[6:0] 为117.93**

2. 窗口时间是 TWIN = TPCLK1 × 4096 × WDGTB预分频系数 × (T[6:0] - W[6:0])

30 = 1/36000 * 4096 * 8 * (T[6:0] - W[6:0]) **W[6:0]为84.97**

36.读写内部Flash闪存

```c
uint32_t MyFLASH_ReadWord(uint32_t Address)
{
    return *((__IO uint32_t *)(Address));   //使用指针访问指定地址下的数据并返回
}

  //FLASH读取一个16位的字
uint16_t MyFLASH_ReadHalfWord(uint32_t Address)
{
    return *((__IO uint16_t *)(Address));   //使用指针访问指定地址下的数据并返回
}

  //FLASH读取一个8位的字
uint8_t MyFLASH_ReadByte(uint32_t Address)
{
    return *((__IO uint8_t *)(Address));    //使用指针访问指定地址下的数据并返回
}
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  /* USER CODE BEGIN 2 */
    OLED_Init();
    OLED_ShowHexNum(1,1,MyFLASH_ReadWord(0x8000000),8);//32位，16进制的长度为8
    OLED_ShowHexNum(2,1,MyFLASH_ReadHalfWord(0x8000002),4);//16位，16进制的长度
    OLED_ShowHexNum(3,1,MyFLASH_ReadByte(0x8000000),2);//8位，16进制的长度为2
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

  }
  /* USER CODE END 3 */
}
```

```c
void hhMyFLASH_ErasePage(uint32_t ErasePageBaseAddr,uint32_t
ErasePageNbPageCount)
{
        HAL_FLASH_Unlock();
        FLASH_EraseInitTypeDef EraseInitStruct = {
            .TypeErase = FLASH_TYPEERASE_PAGES,        //页擦除
            .PageAddress = ErasePageBaseAddr,               //擦除地址
            .NbPages = ErasePageNbPageCount
  //擦除页数
        };
        uint32_t PageError = 0;
        __disable_irq();                                //擦除前关闭中断
        if (HAL_FLASHEx_Erase(&EraseInitStruct,&PageError) == HAL_OK)
        {
            printf("擦除 成功\r\n");
        }
        __enable_irq();
        HAL_FLASH_Lock();                      //加锁
}


     hhMyFLASH_ErasePage(0x0800FC00,1);//擦除最后一页
     HAL_FLASH_Unlock();

HAL_FLASH_Program(FLASH_TYPEPROGRAM_WORD,0x0800FC00,0x12346666);//往最
后一页写入字数据
//   HAL_FLASH_Lock();
//   HAL_FLASH_Unlock();

HAL_FLASH_Program(FLASH_TYPEPROGRAM_HALFWORD,0x0800FC10,0xABCE);//往最
后一页写入半字数据
     HAL_FLASH_Lock();
```