关于智能水箱嵌入式软件ADC模块的说明

作者：薛荣坤

12位高精度ADC检测模块

总体上嵌入式开发，通过STM32F103ZETX和 制作嵌入式系统，实现基于定时器中断下嵌套DMA通道的3个ADC，基于FMC通信下的屏幕显示，基于ESP8266的WIFI模组，基于USART1的串口通信等功能，为后续和硬件的搭配提供基础。

本文档主要用于说明ADC各个模块的配置方法，为之后产品细节开发提供帮助

ADC实现思路：

我们首先需要了解STM32F103系列所拥有的三个高精度ADC，

12位ADC是一种逐次逼近型模拟数字转换器。它有多达18个通道，可测量16个外部和2个内部信号源。各通道的A/D转换可以单次、连续、扫描或间断模式执行。ADC的结果可以左对齐或右对齐方式存储在16位数据寄存器中。

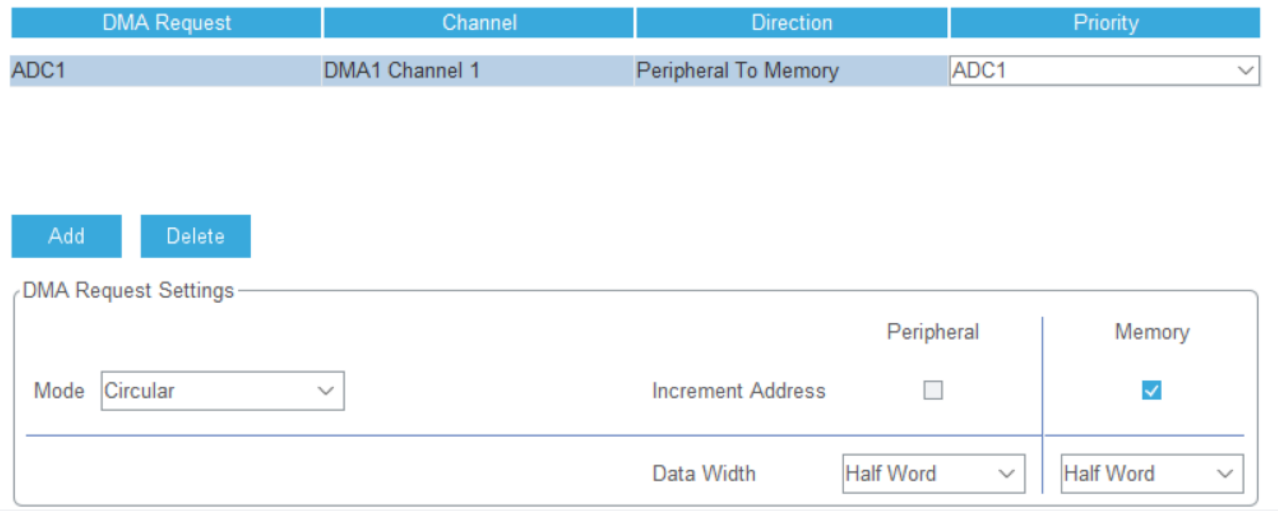
符号	参数	条件	最小值	典型值	最大值	单位
V_{DDA}	供电电压		2.4		3.6	V
V_{REF+}	正参考电压		2.4		V_{DDA}	V
I_{VREF}	在V_{REF}输入脚上的电压			160[1]	220[1]	μA
f_{ADC}	ADC时钟频率		0.6		14	MHz
f_S[2]	采样速率		0.05		1	MHz
f_{TRIG}[2]	外部触发频率	f_{ADC} = 14MHz			823	kHz
					17	$1/f_{ADC}$
V_{AIN}	转换电压范围[3]		0(V_{SSA}或V_{REF-}连接到地)		V_{REF+}	V
R_{AIN}[2]	外部输入阻抗		参见公式1和表59			kΩ
R_{ADC}[2]	采样开关电阻				1	kΩ
C_{ADC}[2]	内部采样和保持电容				12	pF
t_{CAL}[2]	校准时间	f_{ADC} = 14MHz	5.9			μs
			83			$1/f_{ADC}$
t_{lat}[2]	注入触发转换时延	f_{ADC} = 14MHz			0.214	μs
					3[4]	$1/f_{ADC}$
t_{latr}[2]	常规触发转换时延	f_{ADC} = 14MHz			0.143	μs
					2[4]	$1/f_{ADC}$

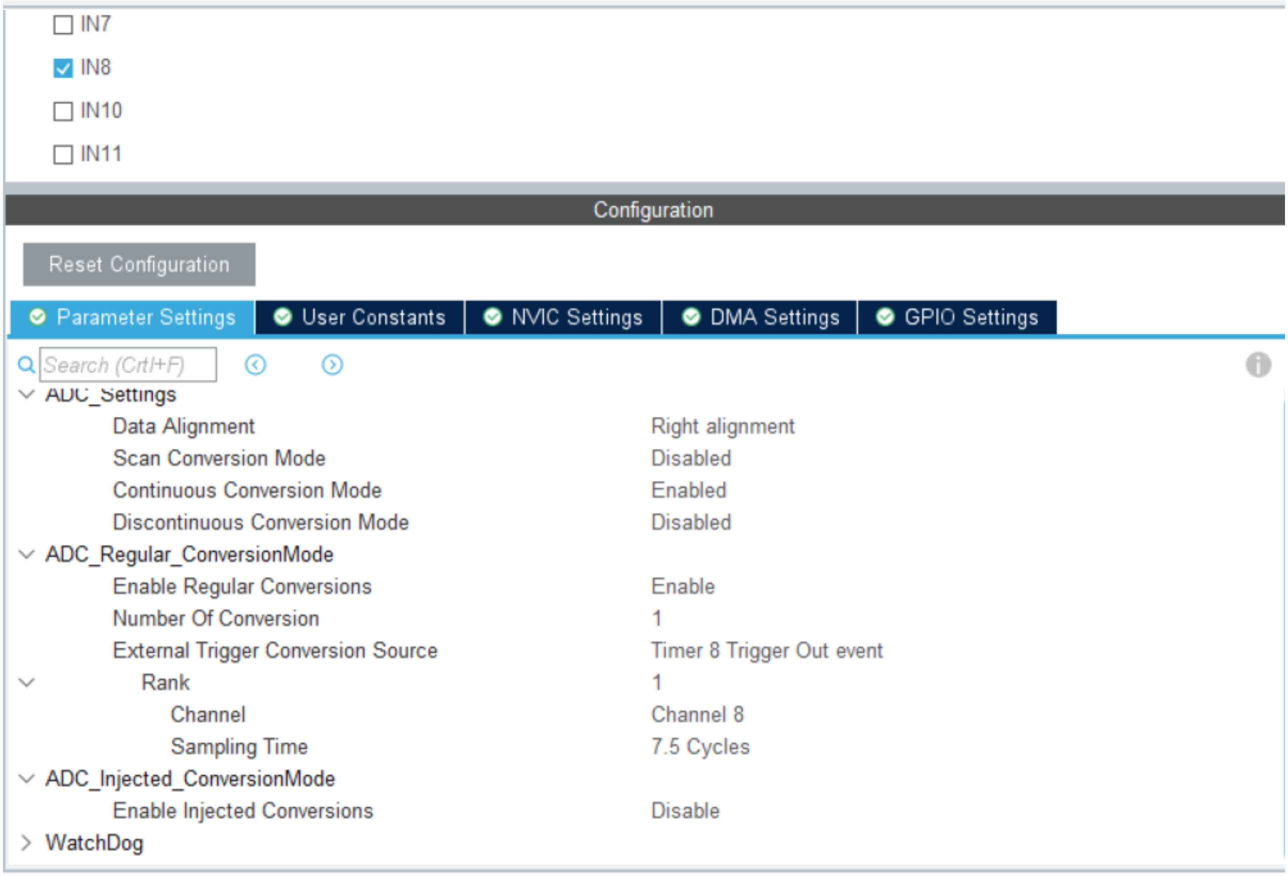
显然我们可以看到这里f_{adc}工作频率是14MHZ，而系统时钟是72MHZ，显然需要把分频因子设置为6。因为我们后续还希望CPU在处理ADC时执行更多操作，显然不应该使用轮询，由于我们希望由定时器中断触发ADC的采样，然后DMA源源不断的将代码从ADC的寄存器移动到指定数组。

CUBEMX配置

ADC1和ADC3的配置

图形初始化

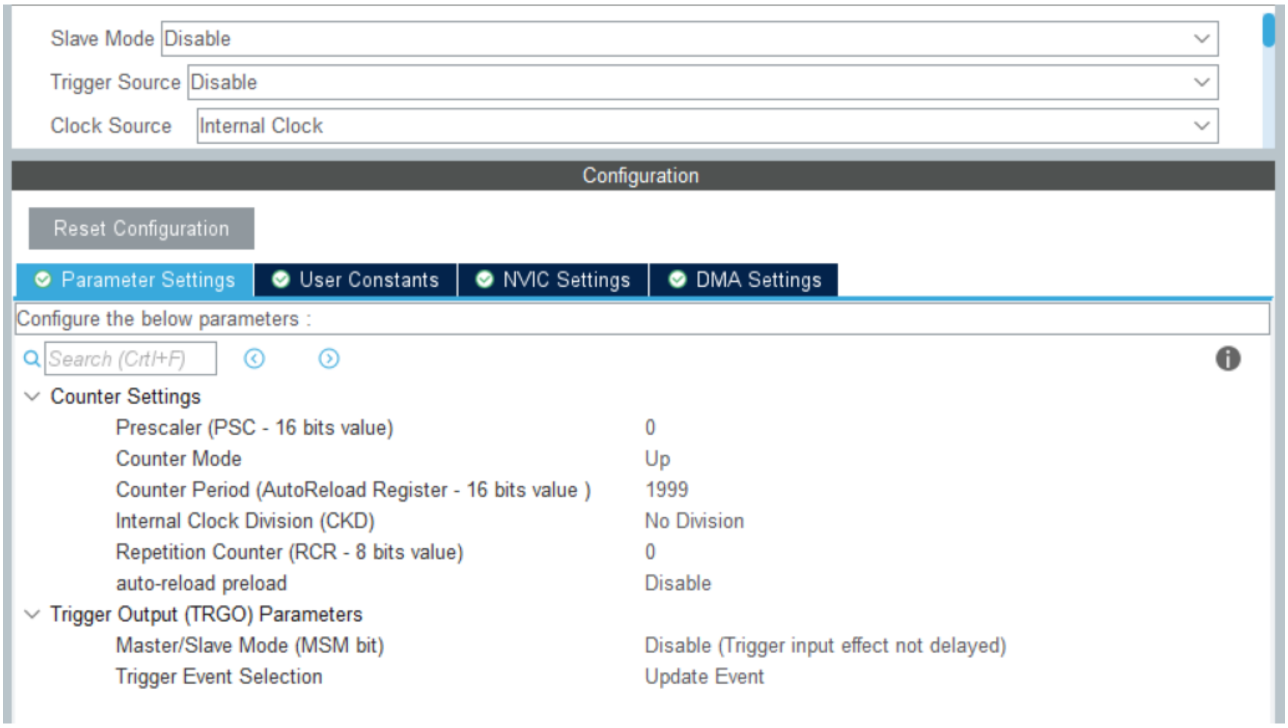




以ADC3的配置举例，（ADC3的通道8，对应PF10)

- 在ADC配置中需要修改 External Trigger Convision Source为定时器8触发，Sampling Time修改为7.5Cycles，Continuous Conversion Mode（连续转换）改为ENABLE。
- 在DMA配置中，Data Width(数据宽)改为Half World. *Mode* 改为循环，这至关重要，这使得*DMA*传送不会停止，笔者一开始默认了*NORMAL*，*DEBug*发现每次只传送一次，就结束了 ，方向改为Peripheral to Memory

TIMEer定时器界面

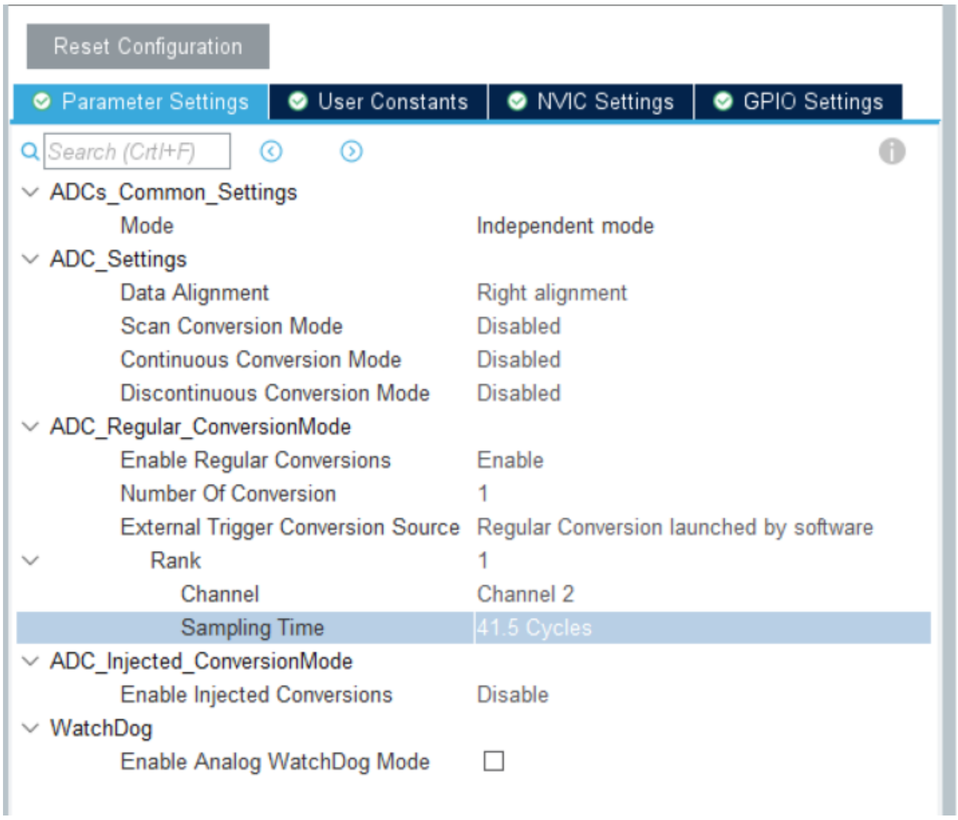


要想实现每0.x秒的精准采样，这一点是必不可少的，因为我们如果只使用DMA技术，ADC将不断以高频率采样，笔者发现这种速度太快且不可控，尤其在实际生产中，我们还需要协调3个ADC的关系，如果3个ADC都在同一个时刻采样，显然是可以很好的解决我们的问题。

- 分频系数改为1999，Trigger Event Selection 改为自动更新，每次中断结束后，自己重新开始计数。

ADC2的配置

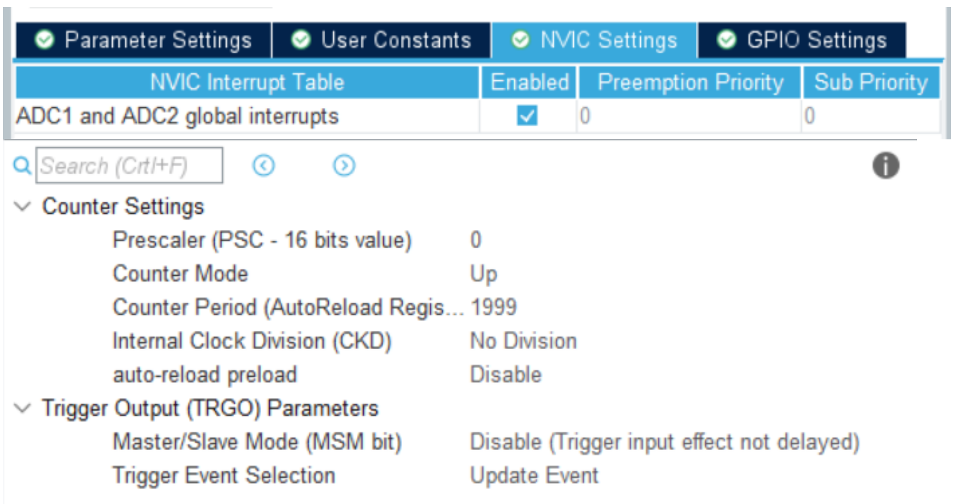
图形化初始化



因为ADC2没有DMA系统，所以我们需要使用单独的Timer3来产生中断，这里我们用一个小技巧。我们如果直接设置ADC2由TIMER3触发，发生很奇怪的现象，始终没有中断写入。不如直接使用软件中断触发，然后写两个中断回调函数来判断，这么做本质原因就是调用HAL库之后，程序员部分失去了底层的控制。

- Continuous Conversion Mode :改为Disabled
- Samping Time (因为没有DMA，我们丧失了快速反应能力，不如直接增加Samping Time) :41.5 Cycles

Timer定时器界面



- TIMER3的初始设置和TIMER8一样

中断初始化

ADC1 and ADC2 global interrupts	☑	0	0
TIM3 global interrupt	☑	0	0

- 中断初始化只需要将值（0，0）赋值给（Preemption Priority, Sub Pority）
- 确实没有必要给ADC太高的优先级，因为我们ADC采样是不断进行的，我们应该注意串口的优先级，串口再发送数据时，发送WIFI串口优先级>>发送Debug优先级>>ADC. 个人觉得打断串口的中断是一种疯狂的行为，曾经这个错误把我折腾了一下午，甚至无法靠调试得到结果

代码源码

ADC.c 与ADC.h

代码主体由*CUBEMX*组成，我们需要理解后，对指定部分进行更改，我们在后面进行了详细注释，尤其是修改部分

（ADC1 ADC3）（ ADC2）分别有两种配置方式

```
/* USER CODE BEGIN Header */
/**
  *****************************************************************
*************
  * @file    adc.c
  * @brief   This file provides code for the configuration
  *          of the ADC instances.
  *****************************************************************
*************
  * @attention
  *
  * Copyright (c) 2022 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the
LICENSE file
  * in the root directory of this software component.
```

```
  * If no LICENSE file comes with this software, it is provided AS-
IS.
  *
  ******************************************************************
************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------
-----------*/
#include "adc.h"

/* USER CODE BEGIN 0 */
__IO uint16_t ADC_ConvertedValue1;
__IO uint16_t ADC_ConvertedValue2;
__IO uint16_t ADC_ConvertedValue3;
#include "tim.h"
/* USER CODE END 0 */

ADC_HandleTypeDef hadc1;
ADC_HandleTypeDef hadc2;
ADC_HandleTypeDef hadc3;
DMA_HandleTypeDef hdma_adc1;
DMA_HandleTypeDef hdma_adc3;

/* ADC1 init function */
void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */
    __HAL_RCC_DMA1_CLK_ENABLE();
  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */
  /** Common config
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.ContinuousConvMode = ENABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
```

```c
  hadc1.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T8_TRGO;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }
  /** Enable or disable the remapping of ADC1_ETRGREG:
  * ADC1 External Event regular conversion is connected to TIM8
TRGO
  */
  __HAL_AFIO_REMAP_ADC1_ETRGREG_ENABLE();
  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_4;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_7CYCLES_5;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */
    //HAL_ADC_Start_DMA(&hadc1, (uint32_t*)&ADC_ConvertedValue1,
1);
  /* USER CODE END ADC1_Init 2 */
}
/* ADC2 init function */
void MX_ADC2_Init(void)
{

  /* USER CODE BEGIN ADC2_Init 0 */
  /* USER CODE END ADC2_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC2_Init 1 */

  /* USER CODE END ADC2_Init 1 */
  /** Common config
  */
  hadc2.Instance = ADC2;
  hadc2.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc2.Init.ContinuousConvMode = DISABLE;
```

```
  hadc2.Init.DiscontinuousConvMode = DISABLE;
  hadc2.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc2.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc2.Init.NbrOfConversion = 1;
  if (HAL_ADC_Init(&hadc2) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_2;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_41CYCLES_5;
  if (HAL_ADC_ConfigChannel(&hadc2, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC2_Init 2 */

  /* USER CODE END ADC2_Init 2 */

}
/* ADC3 init function */
void MX_ADC3_Init(void)
{

  /* USER CODE BEGIN ADC3_Init 0 */
    __HAL_RCC_DMA2_CLK_ENABLE();
  /* USER CODE END ADC3_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC3_Init 1 */

  /* USER CODE END ADC3_Init 1 */
  /** Common config
  */
  hadc3.Instance = ADC3;
  hadc3.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc3.Init.ContinuousConvMode = ENABLE;
  hadc3.Init.DiscontinuousConvMode = DISABLE;
  hadc3.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T8_TRGO;
  hadc3.Init.DataAlign = ADC_DATAALIGN_RIGHT;
```

```
  hadc3.Init.NbrOfConversion = 1;
  if (HAL_ADC_Init(&hadc3) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_8;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_7CYCLES_5;
  if (HAL_ADC_ConfigChannel(&hadc3, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC3_Init 2 */
    //HAL_ADC_Start_DMA(&hadc3, (uint32_t*)&ADC_ConvertedValue3,
1);
  /* USER CODE END ADC3_Init 2 */

}

void HAL_ADC_MspInit(ADC_HandleTypeDef* adcHandle)
{

  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(adcHandle->Instance==ADC1)
  {
  /* USER CODE BEGIN ADC1_MspInit 0 */

  /* USER CODE END ADC1_MspInit 0 */
    /* ADC1 clock enable */
    __HAL_RCC_ADC1_CLK_ENABLE();

    __HAL_RCC_GPIOA_CLK_ENABLE();
    /**ADC1 GPIO Configuration
    PA4     ------> ADC1_IN4
    */
    GPIO_InitStruct.Pin = GPIO_PIN_4;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    /* ADC1 DMA Init */
    /* ADC1 Init */
```

```
    hdma_adc1.Instance = DMA1_Channel1;
    hdma_adc1.Init.Direction = DMA_PERIPH_TO_MEMORY;
    hdma_adc1.Init.PeriphInc = DMA_PINC_DISABLE;
    hdma_adc1.Init.MemInc = DMA_MINC_ENABLE;
    hdma_adc1.Init.PeriphDataAlignment = DMA_PDATAALIGN_HALFWORD;
    hdma_adc1.Init.MemDataAlignment = DMA_MDATAALIGN_HALFWORD;
    hdma_adc1.Init.Mode = DMA_CIRCULAR;
    hdma_adc1.Init.Priority = DMA_PRIORITY_MEDIUM;
    if (HAL_DMA_Init(&hdma_adc1) != HAL_OK)
    {
      Error_Handler();
    }

    __HAL_LINKDMA(adcHandle,DMA_Handle,hdma_adc1);

    /* ADC1 interrupt Init */
    HAL_NVIC_SetPriority(ADC1_2_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(ADC1_2_IRQn);
  /* USER CODE BEGIN ADC1_MspInit 1 */

  /* USER CODE END ADC1_MspInit 1 */
  }
  else if(adcHandle->Instance==ADC2)
  {
  /* USER CODE BEGIN ADC2_MspInit 0 */

  /* USER CODE END ADC2_MspInit 0 */
    /* ADC2 clock enable */
    __HAL_RCC_ADC2_CLK_ENABLE();

    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    /**ADC2 GPIO Configuration
    PC2     ------> ADC2_IN12
    PA2     ------> ADC2_IN2
    */
    GPIO_InitStruct.Pin = GPIO_PIN_2;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = GPIO_PIN_2;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

```
    /* ADC2 interrupt Init */
    HAL_NVIC_SetPriority(ADC1_2_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(ADC1_2_IRQn);
  /* USER CODE BEGIN ADC2_MspInit 1 */

  /* USER CODE END ADC2_MspInit 1 */
  }
  else if(adcHandle->Instance==ADC3)
  {
  /* USER CODE BEGIN ADC3_MspInit 0 */

  /* USER CODE END ADC3_MspInit 0 */
    /* ADC3 clock enable */
    __HAL_RCC_ADC3_CLK_ENABLE();

    __HAL_RCC_GPIOF_CLK_ENABLE();
    /**ADC3 GPIO Configuration
    PF10     ------> ADC3_IN8
    */
    GPIO_InitStruct.Pin = GPIO_PIN_10;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    HAL_GPIO_Init(GPIOF, &GPIO_InitStruct);

    /* ADC3 DMA Init */
    /* ADC3 Init */
    hdma_adc3.Instance = DMA2_Channel5;
    hdma_adc3.Init.Direction = DMA_PERIPH_TO_MEMORY;
    hdma_adc3.Init.PeriphInc = DMA_PINC_DISABLE;
    hdma_adc3.Init.MemInc = DMA_MINC_ENABLE;
    hdma_adc3.Init.PeriphDataAlignment = DMA_PDATAALIGN_HALFWORD;
    hdma_adc3.Init.MemDataAlignment = DMA_MDATAALIGN_HALFWORD;
    hdma_adc3.Init.Mode = DMA_CIRCULAR;
    hdma_adc3.Init.Priority = DMA_PRIORITY_MEDIUM;
    if (HAL_DMA_Init(&hdma_adc3) != HAL_OK)
    {
      Error_Handler();
    }

    __HAL_LINKDMA(adcHandle,DMA_Handle,hdma_adc3);

  /* USER CODE BEGIN ADC3_MspInit 1 */
```

```
  /* USER CODE END ADC3_MspInit 1 */
  }
}

void HAL_ADC_MspDeInit(ADC_HandleTypeDef* adcHandle)
{

  if(adcHandle->Instance==ADC1)
  {
  /* USER CODE BEGIN ADC1_MspDeInit 0 */

  /* USER CODE END ADC1_MspDeInit 0 */
    /* Peripheral clock disable */
    __HAL_RCC_ADC1_CLK_DISABLE();

    /**ADC1 GPIO Configuration
    PA4     ------> ADC1_IN4
    */
    HAL_GPIO_DeInit(GPIOA, GPIO_PIN_4);

    /* ADC1 DMA DeInit */
    HAL_DMA_DeInit(adcHandle->DMA_Handle);

    /* ADC1 interrupt Deinit */
  /* USER CODE BEGIN ADC1:ADC1_2_IRQn disable */
    /**
    * Uncomment the line below to disable the "ADC1_2_IRQn"
interrupt
    * Be aware, disabling shared interrupt may affect other IPs
    */
    /* HAL_NVIC_DisableIRQ(ADC1_2_IRQn); */
  /* USER CODE END ADC1:ADC1_2_IRQn disable */

  /* USER CODE BEGIN ADC1_MspDeInit 1 */

  /* USER CODE END ADC1_MspDeInit 1 */
  }
  else if(adcHandle->Instance==ADC2)
  {
  /* USER CODE BEGIN ADC2_MspDeInit 0 */

  /* USER CODE END ADC2_MspDeInit 0 */
    /* Peripheral clock disable */
```

```
    __HAL_RCC_ADC2_CLK_DISABLE();

    /**ADC2 GPIO Configuration
    PC2     ------> ADC2_IN12
    PA2     ------> ADC2_IN2
    */
    HAL_GPIO_DeInit(GPIOC, GPIO_PIN_2);

    HAL_GPIO_DeInit(GPIOA, GPIO_PIN_2);

    /* ADC2 interrupt Deinit */
  /* USER CODE BEGIN ADC2:ADC1_2_IRQn disable */
    /**
    * Uncomment the line below to disable the "ADC1_2_IRQn"
interrupt
    * Be aware, disabling shared interrupt may affect other IPs
    */
    /* HAL_NVIC_DisableIRQ(ADC1_2_IRQn); */
  /* USER CODE END ADC2:ADC1_2_IRQn disable */

  /* USER CODE BEGIN ADC2_MspDeInit 1 */

  /* USER CODE END ADC2_MspDeInit 1 */
  }
  else if(adcHandle->Instance==ADC3)
  {
  /* USER CODE BEGIN ADC3_MspDeInit 0 */

  /* USER CODE END ADC3_MspDeInit 0 */
    /* Peripheral clock disable */
    __HAL_RCC_ADC3_CLK_DISABLE();

    /**ADC3 GPIO Configuration
    PF10     ------> ADC3_IN8
    */
    HAL_GPIO_DeInit(GPIOF, GPIO_PIN_10);

    /* ADC3 DMA DeInit */
    HAL_DMA_DeInit(adcHandle->DMA_Handle);
  /* USER CODE BEGIN ADC3_MspDeInit 1 */

  /* USER CODE END ADC3_MspDeInit 1 */
  }
```

```
}

/* USER CODE BEGIN 1 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)    //定时器中断回调
{
    HAL_ADC_Start_IT(&hadc2);
}

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* AdcHandle){
        HAL_ADC_Stop_IT(&hadc2);
    HAL_TIM_Base_Stop_IT(&htim3);
        ADC_ConvertedValue2=HAL_ADC_GetValue(&hadc2);
    HAL_TIM_Base_Start_IT(&htim3);
}



/* USER CODE END 1 */
```

```
/* USER CODE BEGIN Header */
/**

*******************************************************************
***********
  * @file    adc.h
  * @brief   This file contains all the function prototypes for
  *          the adc.c file

*******************************************************************
***********
  * @attention
  *
  * Copyright (c) 2022 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the
LICENSE file
  * in the root directory of this software component.
```

```
  * If no LICENSE file comes with this software, it is provided AS-
IS.
  *

*******************************************************************
***********
  */
/* USER CODE END Header */
/* Define to prevent recursive inclusion --------------------------
-----------*/
#ifndef __ADC_H__
#define __ADC_H__

#ifdef __cplusplus
extern "C" {
#endif

/* Includes -------------------------------------------------------
-----------*/
#include "main.h"

/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

extern ADC_HandleTypeDef hadc1;
extern ADC_HandleTypeDef hadc2;
extern ADC_HandleTypeDef hadc3;

/* USER CODE BEGIN Private defines */
// ADC1转换的电压值通过DMA方式传到SRAM
extern __IO uint16_t ADC_ConvertedValue1;
extern __IO uint16_t ADC_ConvertedValue2;
extern __IO uint16_t ADC_ConvertedValue3;
/* USER CODE END Private defines */

void MX_ADC1_Init(void);
void MX_ADC2_Init(void);
void MX_ADC3_Init(void);

/* USER CODE BEGIN Prototypes */

/* USER CODE END Prototypes */
```

```
#ifdef __cplusplus
}
#endif

#endif /* __ADC_H__ */
```

ADC3中断函数的说明

这里注意，一定要在中断里面关闭时钟，然后处理完再打开，这是一个好习惯，因为谁都不知道会在里面处理多久，如果不关闭，下一次中断来临，可能会造成程序异常。血的教训

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)    //定时器中断回调
{
    HAL_ADC_Start_IT(&hadc2); //定时器中断里面开启ADC中断转换，1ms开启一次采集
}

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* AdcHandle){
        HAL_ADC_Stop_IT(&hadc2);     //关闭adc2
    HAL_TIM_Base_Stop_IT(&htim3);    //关闭时钟Timer2
        ADC_ConvertedValue2=HAL_ADC_GetValue(&hadc2);//传递值
    HAL_TIM_Base_Start_IT(&htim3);   //打开时钟Timer2
}
```

ADC1,2初始化时的说明

为了使用DMA传输，我们需要在第一句使能DMA时钟，但是注意我们不用HAL库生成的DMA_init,其存在一定的时间逻辑混乱。

```
void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */
```

```c
    __HAL_RCC_DMA1_CLK_ENABLE();
  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};
  ...
  }



  void MX_ADC3_Init(void)
{

  /* USER CODE BEGIN ADC3_Init 0 */
    __HAL_RCC_DMA2_CLK_ENABLE();
  /* USER CODE END ADC3_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC3_Init 1 */
  ...}
```

MAIN.C （ADC代码使用部分）

```c
//main函数变量
// ADC1转换的电压值通过DMA方式传到SRAM
//这部分代码，后面将移动到ADC.H
extern __IO uint16_t ADC_ConvertedValue1;
extern __IO uint16_t ADC_ConvertedValue2;
extern __IO uint16_t ADC_ConvertedValue3;

// 局部变量，用于保存转换计算后的电压值
float ADC_Vol;
float ADC_Vol2;
float ADC_Vol3;

//时钟出花
HAL_ADC_Start_IT(&hadc2);     //使能hadc2的中断
HAL_TIM_Base_Start(&htim3);   //使能TIM3的时钟
HAL_TIM_Base_Start(&htim8);      //使能TIM8的时钟
```

```
HAL_TIM_Base_Stop_IT(&htim3);     //关闭ADC2采样
HAL_TIM_Base_Stop_IT(&htim8);    //关闭ADC3采样
HAL_ADC_Start_DMA(&hadc1, (uint32_t*)&ADC_ConvertedValue1, 1);
HAL_ADC_Start_DMA(&hadc3, (uint32_t*)&ADC_ConvertedValue3, 1);

while(1){
    // 读取转换的AD值
    HAL_TIM_Base_Start_IT(&htim8);    //打开ADC2采样
    HAL_TIM_Base_Start_IT(&htim3);    //打开ADC1,ADC3采样
    ADC_Vol =(float) ADC_ConvertedValue1/4096*(float)3.3;// 读取转换的AD1值
    ADC_Vol2 =(float) ADC_ConvertedValue2/4096*(float)3.3;// 读取转换的AD2值
    ADC_Vol3 =(float) ADC_ConvertedValue3/4096*(float)3.3;// 读取转换的AD3值
    printf("\r\n The current AD value = 0x%04X \r\n", ADC_ConvertedValue2);
    printf("\r\n The current AD value = %f V \r\n",ADC_Vol2);
}
```

测试结果

ADC1

ADC2

```
[21:26:46.678]收←◆
 The current AD2 value = 0x0FB9

 The current AD2 value = 3.242798 V

[21:26:46.786]收←◆
 The current AD2 value = 0x0FCC

 The current AD2 value = 3.258106 V

[21:26:46.895]收←◆
 The current AD2 value = 0x0FC5

 The current AD2 value = 3.252466 V

[21:26:47.005]收←◆
 The current AD2 value = 0x0FC4

 The current AD2 value = 3.251660 V

[21:26:47.110]收←◆
 The current AD2 value = 0x0FC5

 The current AD2 value = 3.252466 V

[21:26:47.219]收←◆
 The current AD2 value = 0x0FC0

 The current AD2 value = 3.248437 V
```

ADC3

```
[21:29:27.857]收←◆
 The current AD2 value = 0x0FC8

 The current AD2 value = 3.254883 V

[21:29:27.965]收←◆
 The current AD2 value = 0x0FC8

 The current AD2 value = 3.254883 V

[21:29:28.073]收←◆
 The current AD2 value = 0x0FC8

 The current AD2 value = 3.254883 V

[21:29:28.179]收←◆
 The current AD2 value = 0x0FC9

 The current AD2 value = 3.252466 V

[21:29:28.288]收←◆
 The current AD2 value = 0x0FC8

 The current AD2 value = 3.254883 V

[21:29:28.397]收←◆
 The current AD2 value = 0x0FC8

 The current AD2 value = 3.254883 V
```

测试结论

精度12位，均为有效数字，ADC1，2，3均工作正常，精度符合项目要求，但是还需要极限测试。

硬件说明（ADC1,2,3通用）

ADC对应端口

- ADC1 ->PA4
- ADC2->PA2
- ADC3->PF10

ADC硬件连接

由于*ADC*采样只能采集*0-3.3v*之间的电压，所以需要硬件设计中考虑对传感器的分压，建议采用*2*个贴片电阻将*5V*分压至*3.3.v,*请务必考虑*ADC*的输入电阻R_{AIN}

公式1：最大R_{AIN}公式

$$R_{AIN} < \frac{T_S}{f_{ADC} \times C_{ADC} \times \ln(2^{N+2})} - R_{ADC}$$

上述公式(公式1)用于决定最大的外部阻抗，使得误差可以小于1/4 LSB。其中N=12(表示12位分辨率)。

表59 f_{ADC}=14MHz[1]时的最大R_{AIN}

T_S(周期)	t_S(μs)	最大R_{AIN}(kΩ)
1.5	0.11	1.2
7.5	0.54	10
13.5	0.96	19
28.5	2.04	41
41.5	2.96	60
55.5	3.96	80
71.5	5.11	104
239.5	17.1	350

1. 由设计保证，不在生产中测试。

本嵌入式，均采用T_S=*55.5*