应用笔记

ACM32H5 系列芯片

ADC 应用笔记

版本：V1.1

日期：2024-12-11

上海航芯电子科技股份有限公司

1. 概述

本应用手册适用于 ACM32H5 系列芯片 ADC 模块。它描述了与 ADC 模块相关的设置和功能使用方法，以便在应用程序中进行优化设计。

本应用说明应与相关的用户手册、数据手册一同阅读。

2. ADC 模块初始化

2.1. 主要特性

- 多达 3 个 ADC，其中 ADC1 和 ADC2 可以在双重模式下运行
  - ADC1 连接 16 个外部通道+2 个内部通道
  - ADC2 连接 14 个外部通道+4 个内部通道
  - ADC3 连接 16 个外部通道+2 个内部通道
- 12 位分辨率，也可配置成 10 位、8 位或 6 位分辨率
- 可通过降低分辨率来缩短转换时间
- 转换速率最高可达 5Msps（12 位分辨率）
- 支持自校准
- ADC 转换时间可以与 AHB 总线时钟频率无关
- 每个 ADC 包含 19 个通道，其中通道 0 为校准通道
- 每个 ADC 的外部模拟输入通道
  - 源于 GPIO PAD 的 4 个快速通道（ADC12_INP1、ADC12_INP3、ADC3_INP2、ADC3_INP3）
- 包含 7 条内部通道
  - DAC2 的通道 1 和通道 2 连接到 ADC1
  - 内建 BGR 连接到 ADC2
  - DAC1 的通道 1 和通道 2 连接到 ADC2
  - 温度传感器连接到 ADC3
  - VBAT 连接到 ADC3
- 支持单端输入或差分输入（可按通道进行编程）
- 采样结束、转换结束、组转换结束、模拟看门狗事件或溢出事件时产生中断
- 支持单次、连续转换模式
- 支持间断模式
- 最多支持 16 个规则通道组和 4 个注入通道组
- 采样时间可以按通道分别编程
- 规则转换和注入转换均有外部触发选项
- 规则通道转换期间有 DMA 请求产生
- AHB 总线便于系统集成，同时实现高速的读写操作
- 数据对齐以保持内置数据一致性
- 支持双重模式（两个 ADC 设备）
- 过采样器
  - 16 位数据寄存器
  - 过采样率可以在 2 到 256 之间调整
  - 可编程数据移位高达 8 位

- 支持模拟看门狗
- ADC 供电要求：1.8V~3.6V
- ADC 输入范围：VREF- ≤ VIN ≤ VREF+

2.2. 结构体说明

2.2.1. ADC_HandleTypeDef 结构体

```
typedef struct __ADC_HandleTypeDef
{
    ADC_TypeDef *Instance;                                          /* 寄存器基地址指针 */
    ADC_InitTypeDef Init;                                           /* ADC 初始化参数结构体*/
    DMA_HandleTypeDef *DMA_Handle;                                  /* DMA 处理程序指针 */
    uint32_t ChannelNum;                                            /* 规则组通道总数 */
    uint32_t *AdcResults;                                           /* 转换结果地址指针 */
    void (*ConvCpltCallback)(struct __ADC_HandleTypeDef *hadc); /* ADC 规则转换完成回调程序指针 */
    void (*GroupCpltCallback)(struct __ADC_HandleTypeDef *hadc); /*ADC 规则组转换完成回调程序指*/
     void (*InjectedConvCpltCallback)(struct __ADC_HandleTypeDef *hadc); /* ADC 注入转换完成回调程序指针 */
     void (*InjectedGroupConvCpltCallback)(struct __ADC_HandleTypeDef *hadc); /* ADC 注入组转换完成回调程序指针 */
     void (*LevelOutOfWindowCallback)(struct __ADC_HandleTypeDef *hadc); /* ADC 模拟看门狗回调程序指针 */
}ADC_HandleTypeDef;
```

2.2.2. 初始化结构体

```
typedef struct
{
    uint32_t ClockSource;                   /* ADC 时钟源 */
    uint32_t ClockPrescaler;                /* ADC 时钟分频系数 */
    uint32_t Resolution;                    /* ADC 分辨率选择 */
    uint32_t DataAlign;                     /* 输出数据对齐方式 */
    FunctionalState ConConvMode;            /* 连续转换模式 */
    FunctionalState DiscontinuousConvMode;  /* 间断模式 */
    uint32_t NbrOfDiscConversion;           /* 间断模式通道计数 */
    uint32_t ExternalTrigConv;              /* 触发模式 */
    uint32_t ExternalTrigConvEdge;          /* 触发极性 */
    uint32_t ChannelEn;                     /* 通道使能 */
    uint32_t DMAMode;                       /* DMA 模式*/
```

```
    uint32_t OverMode;                          /* 溢出模式 */
    uint32_t OverSampMode;                      /* 过采样模式 */
    ADC_OversamplingTypeDef Oversampling;       /* 过采样参数结构体 */
    uint32_t AnalogWDGEn;                       /* 模拟看门狗使能 */
}ADC_InitTypeDef;
```

2.2.3. 规则通道配置结构体

```
typedef struct
{
    uint32_t Channel;           /* ADC 规则转换通道 */
    uint32_t Sq;                /* 转换顺序 */
    uint32_t Smp;               /* 采样周期 */
    uint32_t Diff;              /* 差分/单端模式 */
    uint32_t OffsetNumber;      /* 偏移组选择 */
    uint32_t Offset;            /* 偏移量 */
    uint32_t OffsetCalculate;   /* 偏移计算方式 */
    uint32_t Offsetsign;        /* 偏移结果格式 */
}ADC_ChannelConfTypeDef;
```

2.2.4. 注入通道配置结构体

```
typedef struct
{
    uint32_t InjectedChannel;                       /* ADC 注入转换通道 */
    uint32_t InjectedRank;                          /* 转换顺序 */
    uint32_t InjectedSamplingTime;                  /* 采样周期 */
    uint32_t InjectedDiff;                          /* 差分/单端模式 */
    uint32_t InjectedOffsetNumber;                  /* 偏移组选择 */
    uint32_t InjectedOffset;                        /* 偏移量 */
    uint32_t InjectedOffsetCalculate;               /* 偏移计算方式 */
    uint32_t InjectedOffsetSign;                    /* 偏移结果格式 */
    uint32_t InjectedNbrOfConversion;               /* 注入组序列长度 */
    FunctionalState InjectedDiscontinuousConvMode;  /* 间断模式 */
    FunctionalState AutoInjectedConv;               /* 自动转换模式 */
    uint32_t ExternalTrigInjecConv;                 /* 触发模式 */
    uint32_t ExternalTrigInjecConvEdge;             /* 触发极性 */
    FunctionalState InjecOversamplingMode;          /* 过采样 */
    ADC_InjOversamplingTypeDef  InjecOversampling;  /* 过采样参数结构体 */
```

```
}ADC_InjectionConfTypeDef;
```

2.2.5. 过采样配置结构体

```
typedef struct
{
    uint32_t Ratio;                 /* 过采样率 */
    uint32_t RightBitShift;         /* 过采样移位系数 */
    uint32_t TriggeredMode;         /* 过采样触发模式 */
}ADC_OversamplingTypeDef;
typedef struct
{
    uint32_t Ratio;                 /* 过采样率 */
    uint32_t RightBitShift;         /* 过采样移位系数 */
} ADC_InjOversamplingTypeDef;
```

2.2.6. 模拟看门狗配置结构体

```
typedef struct
{
    uint32_t WatchdogMode;          /* 选择在所有还是单一的注入或者规则通道上使用模拟看门狗 */
    uint32_t RegularChannel;        /* 模拟看门狗的规则通道 */
    uint32_t InjectChannel;         /* 模拟看门狗的注入通道 */
    uint32_t ITMode;                /* 模拟看门狗中断 *。
    uint32_t HighThreshold;         /* 模拟看门狗的高阈值 */
    uint32_t LowThreshold;          / * 模拟看门狗的低阈值 */
    uint32_t Diff;                  /* 单端或差分模式 */
}ADC_AnalogWDGConfTypeDef;
```

2.2.7. 双 ADC 配置结构体

```
typedef struct
{
    uint32_t Mode;                  /* 双 ADC 模式 */
    uint32_t DMAAccessMode;         /* 双 ADC 模式下的 DMA 模式 */
    uint32_t TwoSamplingDelay;      /* 2 个采样阶段之间的延迟 */
}ADC_MultiModeTypeDef;
```

2.3. 规则组初始化配置

我们首先需要定义使用的 ADC 通道数量，以及具体的 ADC 通道号。确定需要使用到的工作模式（独立模式、双 ADC 模式、单端、差分或者是否支持 DMA 等），设置 ADC CLK，根据需求进行初始化配置。下面将描述具

体的配置方法和过程。

首先定义一个 ADC 的总结构体变量，例如：

```
ADC_HandleTypeDef ADC_RegularHandle;
```

定义一个 ADC 的通道设置结构体变量，例如：

```
ADC_ChannelConfTypeDef ADC_ChannelConf;
```

以及定义一个 ADC 转换结果 buffer，例如：

```
uint32_t adcxValBuffer[21];
```

2.3.1. 基本配置

- ADC 时钟来源：

```
ADC_RegularHandle.Init. ClockSource = ADC_CLOCKSOURCE_HCLK;  //设置 ADC 时钟源，来自 HCLK 或 PLL3_P_CLK。
```

- ADC 时钟分频：

```
ADC_RegularHandle.Init.ClockPrescaler = ADC_CLOCK_DIV8;  //设置 ADC 时钟分频，ADC CLK 最大不得超过 75MHZ。
```

- 数据分辨率：

```
ADC_RegularHandle.Init.Resolution = ADC_RESOLUTION_12B;  //可配置 12 位、10 位、8 位或 6 位。
```

- 数据对齐：

```
ADC_RegularHandle.Init.DataAlign = ADC_DATAALIGN_RIGHT;  //左对齐或右对齐。
```

- 是否支持连续转换模式：

```
ADC_RegularHandle.Init.ConConvMode = DISABLE;
```

- 是否支持间断模式：

```
ADC_RegularHandle.Init.DiscontinuousConvMode = DISABLE;
```

- 间断模式通道计数：

```
ADC_RegularHandle.Init.NbrOfDiscConversion = 0;  //规则模式间断模式通道计数，最大为 8。
```

- 触发模式：

```
ADC_RegularHandle.Init.ExternalTrigConv = ADC_SOFTWARE_START;  // ExternalTrigConv 为 32bit 无符号数，其中 0~19 分别定义每个外部触发源，32 定义为软件触发，具体定义详情在例程的 hal_adc.h 中。
```

- 外部触发极性选择：

```
ADC_RegularHandle.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;  //外部触发边沿选择，为 32bit 无符号数，其中 0 表示禁止触发，1 表示上升沿触发，2 表示下降沿触发，3 表示上升沿和下降沿触发，具体定义详情在例程的 hal_adc.h 中
```

- 是否支持 DMA：

```
ADC_RegularHandle.Init.DMAMode = DISABLE;
```

- 溢出模式：

```
ADC_RegularHandle.Init.OverMode = DISABLE;  //溢出时是否保留上次采样数据
```

- 是否支持过采样：

```
ADC_RegularHandle.Init.OverSampMode = DISABLE;  //禁止过采样
```

- 过采样率：

```
ADC_RegularHandle.Init.Oversampling.Ratio = ADC_OVERSAMPLING_RATIO_8;  //过采样率，为 32bit 无符号数，其中 0~7 分别定义过采样倍数 2~256 倍，具体定义详情在例程的 hal_adc.h 中。
```

- 过采样移位系数：

```
ADC_RegularHandle.Init.Oversampling.RightBitShift = ADC_RIGHTBITSHIFT_3;  //过采样移位系数，为 32bit 无符号数，其中 0~8 分别定义右移位数，具体定义详情在例程的 hal_adc.h 中。
```

- 规则组过采样触发模式：

```
ADC_RegularHandle.Init.Oversampling.TriggeredMode = 0;  //一次触发进行 1 次或 N 次 ADC 转换
```

- 是否支持模拟看门狗：

```
ADC_RegularHandle.Init.AnalogWDGEn = DISABLE;
```

- 使能 ADC 通道：

```
ADC_RegularHandle.Init.ChannelEn = ADC_CHANNEL_1_EN;  //参数 ChannelEn 为 32bit 无符号数，其中 bit0~20 分别定义每个 ADC 通道使能，具体定义详情在例程的 hal_adc.h 中。ADC 对应的具体 GPIO 配置在底层函数 HAL_ADC_MspInit 中完成。
```

- 获取 ADC 寄存器基地址：

```
ADC_RegularHandle.Instance = ADC1;
```

- 获取 ADC 转换数据 buffer 指针：

```
ADC_RegularHandle.AdcResults = &adcxValBuffer[0];
```

- 其他参数设置按照例程默认设置
- 初始化 ADC 模块：

```
HAL_ADC_Init(&ADC_RegularHandle);
```

2.3.2. 添加 ADC 转换通道

- 设置规则组序列长度

```
ADC_RegularHandle.ChannelNum = 1;
```

设置的规则组序列长度，必须与添加的通道总数一致，必须在初始化过程中已经设置该通道对应的 GPIO 为模拟通道且已经使能，否则可能导致采样数据丢失或者出错。

- 添加 ADC 通道

```
ADC_ChannelConf.Channel = ADC_CHANNEL_1;  //参数 Channel 为 32bit 无符号数，其中 0~19 分别定义每个 ADC 通道，具体定义详情在例程的 hal_adc.h 中。
```

- 转换顺序

```
ADC_ChannelConf.Sq = ADC_SEQUENCE_SQ1;  //参数 Sq 为 32bit 无符号数，其中 1~16 分别定义在规则序列中的转换顺序，具体定义详情在例程的 hal_adc.h 中。
```

- 采样时间

```
ADC_ChannelConf.Smp = ADC_SMP_CLOCK_320;   //参数 Smp 为 32bit 无符号数，其中 0~15 分别定义通道采样时间，具体定义详情在例程的 hal_adc.h 中。
```

- 差分/单端模式

```
ADC_ChannelConf.Diff = DISABLE;  //单端或差分模式
```

- 偏移组选择

```
ADC_ChannelConf.OffsetNumber = ADC_OFFSET_NONE; //对应偏移寄存器 ADC_OFRx
```

- 偏移量

```
ADC_ChannelConf.Offset = 0;  //偏移量，0~4095
```

- 偏移计算方式

```
ADC_ChannelConf.OffsetSign = ADC_OFFSET_MINUS;  //转换结果加上或减去偏移量
```

- 偏移结果格式

```
ADC_ChannelConf.OffsetSaturation = ADC_OFFSET_SIGN_UNSIGNED;  //计算结果为有符号数或无符号数
```

```
HAL_ADC_ConfigChannel(&ADC_RegularHandle, &ADC_ChannelConf);
```

每调用一次 HAL_ADC_ConfigChannel 函数，添加一个 ADC 通道，添加多个 ADC 通道需多次设置。其中参数 Channel 是 ADC 通道号。参数 Sq 为规则序列的序列号，规则序列号必须从 1 开始连续设置，每个序列设置的 ADC 通道号可以是任意 ADC 通道号。

2.4. 注入组初始化配置

我们首先需要定义使用的 ADC 通道数量，以及具体的 ADC 通道号。确定需要使用到的工作模式（独立模式、双 ADC 模式、单端、差分或者是否支持 DMA 等），设置 ADC CLK，根据需求进行初始化配置。下面将描述具体的配置方法和过程。

首先定义一个 ADC 的总结构体变量，例如：

```
ADC_HandleTypeDef ADC_InjectHandle;
```

定义一个 ADC 的通道设置结构体变量，例如：

```
ADC_InjectionConfTypeDef sConfigInjected;
```

以及定义一个 ADC 转换结果变量，例如：

```
uint32_t adcxConvertedValue;
```

2.4.1. 基本配置

- ADC 时钟来源：

```
ADC_InjectHandle.Init. ClockSource = ADC_CLOCKSOURCE_HCLK;  //设置 ADC 时钟源，来自 HCLK 或 PLL3_P_CLK。
```

- ADC 时钟分频：

```
ADC_InjectHandle.Init.ClockPrescaler = ADC_CLOCK_DIV8;  //设置 ADC 时钟分频，ADC CLK 最大不得超过 75MHZ。
```

- 数据分辨率：

```
ADC_InjectHandle.Init.Resolution = ADC_RESOLUTION_12B;  //可配置 12 位、10 位、8 位或 6 位。
```

- 数据对齐：

```
ADC_InjectHandle.Init.DataAlign = ADC_DATAALIGN_RIGHT;  //左对齐或右对齐。
```

- 是否支持连续转换模式：

```
ADC_InjectHandle.Init.ConConvMode = DISABLE;
```

- 是否支持间断模式：

```
ADC_InjectHandle.Init.DiscontinuousConvMode = DISABLE;
```

- 触发模式：

```
ADC_InjectHandle.Init.ExternalTrigConv = ADC_SOFTWARE_START;  // ExternalTrigConv 为 32bit 无符号数，其中 0~19 分别定义每个外部触发源，32 定义为软件触发，具体定义详情在例程的 hal_adc.h 中。
```

- 外部触发极性选择：

```
ADC_Handle.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;  //外部触发边沿选择，为 32bit 无符号数，其中 0 表示禁止触发，1 表示上升沿触发，2 表示下降沿触发，3 表示上升沿和下降沿触发，具体定义详情在例程的 hal_adc.h 中
```

- 是否支持 DMA：

```
ADC_InjectHandle.Init.DMAMode = DISABLE;
```

- 溢出模式：

```
ADC_InjectHandle.Init.OverMode = DISABLE;  //溢出时是否保留上次采样数据
```

- 是否支持过采样：

```
ADC_InjectHandle.Init.OverSampMode = DISABLE;  //禁止过采样
```

- 规则组过采样触发模式：

```
ADC_InjectHandle.Init.Oversampling.TriggeredMode = 0;  //一次触发进行 1 次或 N 次 ADC 转换
```

- 是否支持模拟看门狗：

```
ADC_InjectHandle.Init.AnalogWDGEn = DISABLE;
```

- 使能 ADC 通道：

```
ADC_InjectHandle.Init.ChannelEn = ADC_CHANNEL_1_EN;  //参数 ChannelEn 为 32bit 无符号数，其中 bit0~20 分别定义每个 ADC 通道使能，具体定义详情在例程的 hal_adc.h 中。ADC 对应的具体 GPIO 配置在底层函数 HAL_ADC_MspInit 中完成。
```

- 获取 ADC 寄存器基地址：

```
ADC_InjectHandle.Instance = ADC2;
```

- 其他参数设置按照例程默认设置
- 初始化 ADC 模块：

```
HAL_ADC_Init(&ADC_InjectHandle);
```

2.4.2. 添加 ADC 转换通道

- 设置注入组序列长度

```
sConfigInjected.InjectedNbrOfConversion = 1;
```

设置的注入组序列长度，必须与添加的通道总数一致，必须在初始化过程中已经设置该通道对应的 GPIO 为模拟通道且已经使能，否则可能导致采样数据丢失或者出错。

- 添加 ADC 通道

```
sConfigInjected.InjectedChannel = ADC_CHANNEL_1;  //参数 Channel 为 32bit 无符号数，其中 0~19 分别定义每个 ADC 通道，具体定义详情在例程的 hal_adc.h 中。
```

- 转换顺序

```
sConfigInjected.InjectedRank = ADC_INJECTED_RANK_1;  //参数 Sq 为 32bit 无符号数，其中 1~16 分别定义在规则序列中的转换顺序，具体定义详情在例程的 hal_adc.h 中。
```

- 采样时间

```
sConfigInjected.InjectedSamplingTime = ADC_SMP_CLOCK_320;   //参数 Smp 为 32bit 无符号数，其中 0~15 分别定义通道采样时间，具体定义详情在例程的 hal_adc.h 中。
```

- 差分/单端模式

```
sConfigInjected.InjectedDiff = DISABLE;  //单端或差分模式
```

- 偏移组选择

```
sConfigInjected.InjectedOffsetNumber = ADC_OFFSET_NONE; //对应偏移寄存器 ADC_OFRx
```

- 偏移量

```
sConfigInjected.InjectedOffset = 0;  //偏移量，0~4095
```

- 偏移计算方式

```
sConfigInjected.InjectedOffsetCalculate = ADC_OFFSET_MINUS;  //转换结果加上或减去偏移量
```

- 偏移结果格式

```
sConfigInjected.InjectedOffsetSign = ADC_OFFSET_SIGN_UNSIGNED;  //计算结果为有符号数或无符号数
```

- 触发模式：

```
sConfigInjected.ExternalTrigInjecConv = ADC_SOFTWARE_START;  // ExternalTrigConv 为 32bit 无符号数，其中 0~19 分别定义每个外部触发源，32 定义为软件触发，具体定义详情在例程的 hal_adc.h 中。
```

- 外部触发极性选择：

```
sConfigInjected.ExternalTrigInjecConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;  //外部触发边沿选择，为 32bit 无符号数，其中 0 表示禁止触发，1 表示上升沿触发，2 表示下降沿触发，3 表示上升沿和下降沿触发，具体定义详情在例程的 hal_adc.h 中
```

- 是否自动转换

```
sConfigInjected.AutoInjectedConv = DISABLE;
```

- 是否支持间断模式

```
sConfigInjected.InjectedDiscontinuousConvMode = DISABLE;
```

- 是否支持过采样：

```
sConfigInjected.InjecOversamplingMode = DISABLE;  //禁止过采样
```

- 过采样率：

```
sConfigInjected.InjecOversampling.Ratio = ADC_OVERSAMPLING_RATIO_8;  //过采样率，为 32bit 无符号数，其中 0~7 分别定义过采样倍数 2~256 倍，具体定义详情在例程的 hal_adc.h 中。
```

- 过采样移位系数：

```
sConfigInjected.InjecOversampling.RightBitShift = ADC_RIGHTBITSHIFT_3;  //过采样移位系数，为 32bit 无符号数，其中 0~8 分别定义右移位数，具体定义详情在例程的 hal_adc.h 中。
```

```
HAL_ADCEx_InjectedConfigChannel (&ADC_InjectHandle, & sConfigInjected);
```

每调用一次 HAL_ADCEx_InjectedConfigChannel 函数，添加一个 ADC 通道，添加多个 ADC 通道需多次设置。其中参数 Channel 是 ADC 通道号。参数 InjectedRank 为注入序列的序列号，注入序列号必须从 1 开始连续设置，每个序列设置的 ADC 通道号可以是任意 ADC 通道号。

2.5. 双 ADC 初始化配置

我们首先需要定义需要使用的 ADC 通道数量，以及具体的 ADC 通道号。确定需要使用到的工作模式（独立模式、双 ADC 模式、单端、差分或者是否支持 DMA 等），设置 ADC CLK，根据需求进行初始化配置。下面将描述具体的配置方法和过程。

首先定义两个 ADC 的总结构体变量，例如：

```
ADC_HandleTypeDef DualADC1_Handle;
ADC_HandleTypeDef DualADC2_Handle;
```

定义两个 ADC 的通道设置结构体变量，例如：

```
ADC_ChannelConfTypeDef ADC_RegChannelConf;
ADC_InjectionConfTypeDef sConfigInjected;
```

定义一个 ADC 的多模式设置结构体变量，例如：

```
ADC_MultiModeTypeDef Multimode;
```

以及定义一个 ADC 转换结果 buffer 和转换结果变量，例如：

```
uint32_t gadcxValBuffer[21];
uint32_t gadcxConvertedValue;
```

2.5.1. 主 ADC1 基本配置

- ADC 时钟来源：

```
DualADC1_Handle.Init. ClockSource = ADC_CLOCKSOURCE_HCLK;  //设置 ADC 时钟源，来自 HCLK 或 PLL3_P_CLK。
```

- ADC 时钟分频：

```
DualADC1_Handle.Init.ClockPrescaler = ADC_CLOCK_DIV8;  //设置 ADC 时钟分频，ADC CLK 最大不得超过 75MHZ。
```

- 数据分辨率：

```
DualADC1_Handle.Init.Resolution = ADC_RESOLUTION_12B;  //可配置 12 位、10 位、8 位或 6 位。
```

- 数据对齐：

```
DualADC1_Handle.Init.DataAlign = ADC_DATAALIGN_RIGHT;  //左对齐或右对齐。
```

- 是否支持连续转换模式：

```
DualADC1_Handle.Init.ConConvMode = DISABLE;
```

- 是否支持间断模式：

```
DualADC1_Handle.Init.DiscontinuousConvMode = DISABLE;
```

- 间断模式通道计数：

```
DualADC1_Handle.Init.NbrOfDiscConversion = 0;  //规则模式间断模式通道计数，最大为 8。
```

- 触发模式：

```
DualADC1_Handle.Init.ExternalTrigConv = ADC_SOFTWARE_START;  // ExternalTrigConv 为 32bit 无符号数，其中 0~19 分别定义每个外部触发源，32 定义为软件触发，具体定义详情在例程的 hal_adc.h 中。
```

- 外部触发极性选择：

```
DualADC1_Handle.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;  //外部触发边沿选择，为 32bit 无符号数，其中 0 表示禁止触发，1 表示上升沿触发，2 表示下降沿触发，3 表示上升沿和下降沿触发，具体定义详情在例程的 hal_adc.h 中
```

- 是否支持 DMA：

```
DualADC1_Handle.Init.DMAMode = ENABLE;
```

- 溢出模式：

```
DualADC1_Handle.Init.OverMode = DISABLE;  //溢出时是否保留上次采样数据
```

- 是否支持过采样：

```
DualADC1_Handle.Init.OverSampMode = DISABLE;  //禁止过采样
```

- 过采样率：

```
DualADC1_Handle.Init.Oversampling.Ratio = ADC_OVERSAMPLING_RATIO_8;  //过采样率，为 32bit 无符号数，其中 0~7 分别定义过采样倍数 2~256 倍，具体定义详情在例程的 hal_adc.h 中。
```

- 过采样移位系数：

```
DualADC1_Handle.Init.Oversampling.RightBitShift = ADC_RIGHTBITSHIFT_3;  //过采样移位系数，为 32bit 无符号数，其中 0~8 分别定义右移位数，具体定义详情在例程的 hal_adc.h 中。
```

- 规则组过采样触发模式：

```
DualADC1_Handle.Init.Oversampling.TriggeredMode = 0;  //一次触发进行 1 次或 N 次 ADC 转换
```

- 是否支持模拟看门狗：

```
DualADC1_Handle.Init.AnalogWDGEn = DISABLE;
```

- 使能 ADC 通道：

```
ADC_RegularHandle.Init.ChannelEn = ADC_CHANNEL_4_EN;  //参数 ChannelEn 为 32bit 无符号数，其中 bit0~20 分别定义每个 ADC 通道使能，具体定义详情在例程的 hal_adc.h 中。ADC 对应的具体 GPIO 配置在底层函数 HAL_ADC_MspInit 中完成。
```

- 获取 ADC 寄存器基地址：

```
ADC_RegularHandle.Instance = ADC1;
```

- 其他参数设置按照例程默认设置
- 初始化 ADC 模块：

```
HAL_ADC_Init(&DualADC1_Handle);
```

2.5.2. 添加 ADC 转换通道

- 设置规则组序列长度

```
DualADC1_Handle.ChannelNum = 1;
```

设置的规则组序列长度，必须与添加的通道总数一致，必须在初始化过程中已经设置该通道对应的 GPIO 为模拟通道且已经使能，否则可能导致采样数据丢失或者出错。

- 添加 ADC 通道

```
ADC_RegChannelConf.Channel = ADC_CHANNEL_4;  //参数 Channel 为 32bit 无符号数，其中 0~19 分别定义每个 ADC 通道，具体定义详情在例程的 hal_adc.h 中。
```

- 转换顺序

```
ADC_RegChannelConf.Sq = ADC_SEQUENCE_SQ1;  //参数 Sq 为 32bit 无符号数，其中 1~16 分别定义在规则序列中的转换顺序，具体定义详情在例程的 hal_adc.h 中。
```

- 采样时间

```
ADC_RegChannelConf.Smp = ADC_SMP_CLOCK_320;   //参数 Smp 为 32bit 无符号数，其中 0~15 分别定义通道采样时间，具体定义详情在例程的 hal_adc.h 中。
```

- 差分/单端模式

```
ADC_RegChannelConf.Diff = DISABLE;  //单端或差分模式
```

- 偏移组选择

```
ADC_RegChannelConf.OffsetNumber = ADC_OFFSET_NONE; //对应偏移寄存器 ADC_OFRx
```

- 偏移量

```
ADC_RegChannelConf.Offset = 0;  //偏移量，0~4095
```

- 偏移计算方式

```
ADC_RegChannelConf.OffsetSign = ADC_OFFSET_MINUS;  //转换结果加上或减去偏移量
```

- 偏移结果格式

```
ADC_RegChannelConf.OffsetSaturation = ADC_OFFSET_SIGN_UNSIGNED;  //计算结果为有符号数或无符号数
```

```
HAL_ADC_ConfigChannel(&DualADC1_Handle, & ADC_RegChannelConf);
```

每调用一次 HAL_ADC_ConfigChannel 函数，添加一个 ADC 通道，添加多个 ADC 通道需多次设置。其中参

数 Channel 是 ADC 通道号。参数 Sq 为规则序列的序列号，规则序列号必须从 1 开始连续设置，每个序列设置的 ADC 通道号可以是任意 ADC 通道号。

- 设置注入组序列长度

```
sConfigInjected.InjectedNbrOfConversion = 1;
```

设置的注入组序列长度，必须与添加的通道总数一致，必须在初始化过程中已经设置该通道对应的 GPIO 为模拟通道且已经使能，否则可能导致采样数据丢失或者出错。

- 添加 ADC 通道

```
sConfigInjected.InjectedChannel = ADC_CHANNEL_1;  //参数 Channel 为 32bit 无符号数，其中 0~19 分别定义每个 ADC 通道，具体定义详情在例程的 hal_adc.h 中。
```

- 转换顺序

```
sConfigInjected.InjectedRank = ADC_INJECTED_RANK_1;  //参数 Sq 为 32bit 无符号数，其中 1~16 分别定义在规则序列中的转换顺序，具体定义详情在例程的 hal_adc.h 中。
```

- 采样时间

```
sConfigInjected.InjectedSamplingTime = ADC_SMP_CLOCK_320;   //参数 Smp 为 32bit 无符号数，其中 0~15 分别定义通道采样时间，具体定义详情在例程的 hal_adc.h 中。
```

- 差分/单端模式

```
sConfigInjected.InjectedDiff = DISABLE;  //单端或差分模式
```

- 偏移组选择

```
sConfigInjected.InjectedOffsetNumber = ADC_OFFSET_NONE; //对应偏移寄存器 ADC_OFRx
```

- 偏移量

```
sConfigInjected.InjectedOffset = 0;  //偏移量，0~4095
```

- 偏移计算方式

```
sConfigInjected.InjectedOffsetCalculate = ADC_OFFSET_MINUS;  //转换结果加上或减去偏移量
```

- 偏移结果格式

```
sConfigInjected.InjectedOffsetSign = ADC_OFFSET_SIGN_UNSIGNED;  //计算结果为有符号数或无符号数
```

- 触发模式：

```
sConfigInjected.ExternalTrigInjecConv = ADC_EXTERNAL_TIG6;  // ExternalTrigConv 为 32bit 无符号数，其中 0~19 分别定义每个外部触发源，32 定义为软件触发，具体定义详情在例程的 hal_adc.h 中。
```

- 外部触发极性选择：

```
sConfigInjected.ExternalTrigInjecConvEdge = ADC_EXTERNALTRIGCONVEDGE_RISING;  //外部触发边沿选择，为 32bit 无符号数，其中 0 表示禁止触发，1 表示上升沿触发，2 表示下降沿触发，3 表示上升沿和下降沿触发，具体定义详情在例程的 hal_adc.h 中
```

- 是否自动转换

```
sConfigInjected.AutoInjectedConv = DISABLE;
```

- 是否支持间断模式

```
sConfigInjected.InjectedDiscontinuousConvMode = DISABLE;
```

- 是否支持过采样：

```
sConfigInjected.InjecOversamplingMode = DISABLE;  //禁止过采样
```

- 过采样率：

```
sConfigInjected.InjecOversampling.Ratio = ADC_OVERSAMPLING_RATIO_8;  //过采样率，为 32bit 无符号数，其中 0~7 分别定义过采样倍数 2~256 倍，具体定义详情在例程的 hal_adc.h 中。
```

- 过采样移位系数：

```
sConfigInjected.InjecOversampling.RightBitShift = ADC_RIGHTBITSHIFT_3;  //过采样移位系数，为 32bit 无符号数，其中 0~8 分别定义右移位数，具体定义详情在例程的 hal_adc.h 中。
```

```
HAL_ADCEx_InjectedConfigChannel (&DualADC1_Handle, & sConfigInjected);
```

每调用一次 HAL_ADCEx_InjectedConfigChannel 函数，添加一个 ADC 通道，添加多个 ADC 通道需多次设置。其中参数 Channel 是 ADC 通道号。参数 InjectedRank 为注入序列的序列号，注入序列号必须从 1 开始连续设置，每个序列设置的 ADC 通道号可以是任意 ADC 通道号。

2.5.3. 从 ADC2 基本配置

- ADC 时钟来源：

```
DualADC2_Handle.Init. ClockSource = ADC_CLOCKSOURCE_HCLK;  //设置 ADC 时钟源，来自 HCLK 或 PLL3_P_CLK。
```

- ADC 时钟分频：

```
DualADC2_Handle.Init.ClockPrescaler = ADC_CLOCK_DIV8;  //设置 ADC 时钟分频，ADC CLK 最大不得超过 75MHZ。
```

- 数据分辨率：

```
DualADC2_Handle.Init.Resolution = ADC_RESOLUTION_12B;  //可配置 12 位、10 位、8 位或 6 位。
```

- 数据对齐：

```
DualADC2_Handle.Init.DataAlign = ADC_DATAALIGN_RIGHT;  //左对齐或右对齐。
```

- 是否支持连续转换模式：

```
DualADC2_Handle.Init.ConConvMode = DISABLE;
```

- 是否支持间断模式：

```
DualADC2_Handle.Init.DiscontinuousConvMode = DISABLE;
```

- 间断模式通道计数：

```
DualADC2_Handle.Init.NbrOfDiscConversion = 0;  //规则模式间断模式通道计数，最大为 8。
```

- 触发模式：

```
DualADC2_Handle.Init.ExternalTrigConv = ADC_SOFTWARE_START;  // ExternalTrigConv 为 32bit 无符号数，其中 0~19 分别定义每个外部触发源，32 定义为软件触发，具体定义详情在例程的 hal_adc.h 中。
```

- 外部触发极性选择：

```
DualADC2_Handle.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;  //外部触发边沿选择，为 32bit 无符号数，其中 0 表示禁止触发，1 表示上升沿触发，2 表示下降沿触发，3 表示上升沿和下降沿触发，具体定义详情在例程的 hal_adc.h 中
```

- 是否支持 DMA：

```
DualADC2_Handle.Init.DMAMode = ENABLE;
```

- 溢出模式：

```
DualADC2_Handle.Init.OverMode = DISABLE;  //溢出时是否保留上次采样数据
```

- 是否支持过采样：

```
DualADC2_Handle.Init.OverSampMode = DISABLE;  //禁止过采样
```

- 过采样率：

```
DualADC2_Handle.Init.Oversampling.Ratio = ADC_OVERSAMPLING_RATIO_8;  //过采样率，为 32bit 无符号数，其中 0~7 分别定义过采样倍数 2~256 倍，具体定义详情在例程的 hal_adc.h 中。
```

- 过采样移位系数：

```
DualADC2_Handle.Init.Oversampling.RightBitShift = ADC_RIGHTBITSHIFT_3;  //过采样移位系数，为 32bit 无符号数，其中 0~8 分别定义右移位数，具体定义详情在例程的 hal_adc.h 中。
```

- 规则组过采样触发模式：

```
DualADC2_Handle.Init.Oversampling.TriggeredMode = 0;  //一次触发进行 1 次或 N 次 ADC 转换
```

- 是否支持模拟看门狗：

```
DualADC2_Handle.Init.AnalogWDGEn = DISABLE;
```

- 使能 ADC 通道：

```
ADC_RegularHandle.Init.ChannelEn = ADC_CHANNEL_5_EN;  //参数 ChannelEn 为 32bit 无符号数，其中 bit0~20 分别定义每个 ADC 通道使能，具体定义详情在例程的 hal_adc.h 中。ADC 对应的具体 GPIO 配置在底层函数 HAL_ADC_MspInit 中完成。
```

- 获取 ADC 寄存器基地址：

```
ADC_RegularHandle.Instance = ADC2;
```

- 其他参数设置按照例程默认设置
- 初始化 ADC 模块：

```
HAL_ADC_Init(&DualADC1_Handle);
```

2.5.4. 添加 ADC 转换通道

- 设置规则组序列长度

```
DualADC2_Handle.ChannelNum = 1;
```

设置的规则组序列长度，必须与添加的通道总数一致，必须在初始化过程中已经设置该通道对应的 GPIO 为模拟通道且已经使能，否则可能导致采样数据丢失或者出错。

- 添加 ADC 通道

```
ADC_RegChannelConf.Channel = ADC_CHANNEL_5;  //参数 Channel 为 32bit 无符号数，其中 0~19 分别定义每个 ADC 通道，具体定义详情在例程的 hal_adc.h 中。
```

- 转换顺序

```
ADC_RegChannelConf.Sq = ADC_SEQUENCE_SQ1;  //参数 Sq 为 32bit 无符号数，其中 1~16 分别定义在规则序列中的转换顺序，具体定义详情在例程的 hal_adc.h 中。
```

- 采样时间

```
ADC_RegChannelConf.Smp = ADC_SMP_CLOCK_320;   //参数 Smp 为 32bit 无符号数，其中 0~15 分别定义通道采样时间，具体定义详情在例程的 hal_adc.h 中。
```

- 差分/单端模式

```
ADC_RegChannelConf.Diff = DISABLE;  //单端或差分模式
```

- 偏移组选择

```
ADC_RegChannelConf.OffsetNumber = ADC_OFFSET_NONE; //对应偏移寄存器 ADC_OFRx
```

- 偏移量

```
ADC_RegChannelConf.Offset = 0;  //偏移量，0~4095
```

- 偏移计算方式

```
ADC_RegChannelConf.OffsetSign = ADC_OFFSET_MINUS;  //转换结果加上或减去偏移量
```

● 偏移结果格式

```
ADC_RegChannelConf.OffsetSaturation = ADC_OFFSET_SIGN_UNSIGNED;  //计算结果为有符号数或无符号数
```

```
HAL_ADC_ConfigChannel(&DualADC2_Handle, & ADC_RegChannelConf);
```

每调用一次 HAL_ADC_ConfigChannel 函数，添加一个 ADC 通道，添加多个 ADC 通道需多次设置。其中参数 Channel 是 ADC 通道号。参数 Sq 为规则序列的序列号，规则序列号必须从 1 开始连续设置，每个序列设置的 ADC 通道号可以是任意 ADC 通道号。

● 设置注入组序列长度

```
sConfigInjected.InjectedNbrOfConversion = 1;
```

设置的注入组序列长度，必须与添加的通道总数一致，必须在初始化过程中已经设置该通道对应的 GPIO 为模拟通道且已经使能，否则可能导致采样数据丢失或者出错。

● 添加 ADC 通道

```
sConfigInjected.InjectedChannel = ADC_CHANNEL_11;  //参数 Channel 为 32bit 无符号数，其中 0~19 分别定义每个 ADC 通道，具体定义详情在例程的 hal_adc.h 中。
```

● 转换顺序

```
sConfigInjected.InjectedRank = ADC_INJECTED_RANK_1;  //参数 Sq 为 32bit 无符号数，其中 1~16 分别定义在规则序列中的转换顺序，具体定义详情在例程的 hal_adc.h 中。
```

● 采样时间

```
sConfigInjected.InjectedSamplingTime = ADC_SMP_CLOCK_320;   //参数 Smp 为 32bit 无符号数，其中 0~15 分别定义通道采样时间，具体定义详情在例程的 hal_adc.h 中。
```

● 差分/单端模式

```
sConfigInjected.InjectedDiff = DISABLE;  //单端或差分模式
```

● 偏移组选择

```
sConfigInjected.InjectedOffsetNumber = ADC_OFFSET_NONE; //对应偏移寄存器 ADC_OFRx
```

● 偏移量

```
sConfigInjected.InjectedOffset = 0;  //偏移量，0~4095
```

● 偏移计算方式

```
sConfigInjected.InjectedOffsetCalculate = ADC_OFFSET_MINUS;  //转换结果加上或减去偏移量
```

● 偏移结果格式

```
sConfigInjected.InjectedOffsetSign = ADC_OFFSET_SIGN_UNSIGNED;  //计算结果为有符号数或无符号数
```

● 触发模式：

```
sConfigInjected.ExternalTrigInjecConv = ADC_EXTERNAL_TIG6;  // ExternalTrigConv 为 32bit 无符号数，其中 0~19 分别定义每个外部触发源，32 定义为软件触发，具体定义详情在例程的 hal_adc.h 中。
```

● 外部触发极性选择：

```
sConfigInjected.ExternalTrigInjecConvEdge = ADC_EXTERNALTRIGCONVEDGE_RISING;  //外部触发边沿选择，为 32bit 无符号数，其中 0 表示禁止触发，1 表示上升沿触发，2 表示下降沿触发，3 表示上升沿和下降沿触发，具体定义详情在例程的 hal_adc.h 中
```

● 是否自动转换

```
sConfigInjected.AutoInjectedConv = DISABLE;
```

● 是否支持间断模式

```
sConfigInjected.InjectedDiscontinuousConvMode = DISABLE;
```

● 是否支持过采样：

```
sConfigInjected.InjecOversamplingMode = DISABLE;  //禁止过采样
```

● 过采样率：

```
sConfigInjected.InjecOversampling.Ratio = ADC_OVERSAMPLING_RATIO_8;  //过采样率，为 32bit 无符号数，其中 0~7 分别定义过采样倍数 2~256 倍，具体定义详情在例程的 hal_adc.h 中。
```

● 过采样移位系数：

```
sConfigInjected.InjecOversampling.RightBitShift = ADC_RIGHTBITSHIFT_3;  //过采样移位系数，为 32bit 无符号数，其中 0~8 分别定义右移位数，具体定义详情在例程的 hal_adc.h 中。
```

```
HAL_ADCEx_InjectedConfigChannel (&DualADC2_Handle, & sConfigInjected);
```

每调用一次 HAL_ADCEx_InjectedConfigChannel 函数，添加一个 ADC 通道，添加多个 ADC 通道需多次设置。其中参数 Channel 是 ADC 通道号。参数 InjectedRank 为注入序列的序列号，注入序列号必须从 1 开始连续设置，每个序列设置的 ADC 通道号可以是任意 ADC 通道号。

2.5.5. 双 ADC 模式配置

● 双 ADC 模式选择

```
Multimode.Mode = ADC_DUALMODE_REGSIMULT_INJECSIMULT; //规则同步 + 注入同步混合模式，具体定义详情在例程的 hal_adc.h 中。
```

● 双 ADC 下 DMA 功能选择

```
Multimode.DMAAccessMode = ADC_DMAACCESSMODE_12_10_BITS;  //主 ADC1 产生 DMA 请求，ADC_CDR 寄存读取数据，分辨率为支持 12 和 10 比特；具体定义详情在例程的 hal_adc.h 中。
```

● 2 个采样阶段之间的延迟

```
Multimode.TwoSamplingDelay = ADC_TWOSAMPLINGDELAY_5CYCLES;  //仅在双重交叉模式下使用
HAL_ADCEx_MultiModeConfigChannel(&DualADC1_Handle, &Multimode);
```

调用 HAL_ADCEx_MultiModeConfigChannel 函数，进行 ADC 多模式配置。

2.6. 模拟看门狗初始化配置

我们首先需要定义使用的 ADC 通道数量，以及具体的 ADC 通道号。确定需要使用到的工作模式（独立模式、双 ADC 模式、单端、差分或者是否支持 DMA 等），设置 ADC CLK，根据需求进行初始化配置。下面将描述具体的配置方法和过程。

首先定义一个 ADC 的总结构体变量，例如：

```
ADC_HandleTypeDef ADC_RegularHandle;
```

定义一个 ADC 的通道设置结构体变量，例如：

```
ADC_AnalogWDGConfTypeDef ADC_AnalogWDGConf;
```

2.6.1. 基本配置

● ADC 时钟来源：

```
ADC_RegularHandle.Init. ClockSource = ADC_CLOCKSOURCE_HCLK;  //设置 ADC 时钟源，来自 HCLK 或 PLL3_P_CLK。
```

- ADC 时钟分频：

```
ADC_RegularHandle.Init.ClockPrescaler = ADC_CLOCK_DIV8;  //设置 ADC 时钟分频，ADC CLK 最大不得超过 75MHZ。
```

- 数据分辨率：

```
ADC_RegularHandle.Init.Resolution = ADC_RESOLUTION_12B;  //可配置 12 位、10 位、8 位或 6 位。
```

- 数据对齐：

```
ADC_RegularHandle.Init.DataAlign = ADC_DATAALIGN_RIGHT;  //左对齐或右对齐。
```

- 是否支持连续转换模式：

```
ADC_RegularHandle.Init.ConConvMode = DISABLE;
```

- 是否支持间断模式：

```
ADC_RegularHandle.Init.DiscontinuousConvMode = DISABLE;
```

- 间断模式通道计数：

```
ADC_RegularHandle.Init.NbrOfDiscConversion = 0;  //规则模式间断模式通道计数，最大为 8。
```

- 触发模式：

```
ADC_RegularHandle.Init.ExternalTrigConv = ADC_SOFTWARE_START;  // ExternalTrigConv 为 32bit 无符号数，其中 0~19 分别定义每个外部触发源，32 定义为软件触发，具体定义详情在例程的 hal_adc.h 中。
```

- 外部触发极性选择：

```
ADC_RegularHandle.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;  //外部触发边沿选择，为 32bit 无符号数，其中 0 表示禁止触发，1 表示上升沿触发，2 表示下降沿触发，3 表示上升沿和下降沿触发，具体定义详情在例程的 hal_adc.h 中
```

- 是否支持 DMA：

```
ADC_RegularHandle.Init.DMAMode = DISABLE;
```

- 溢出模式：

```
ADC_RegularHandle.Init.OverMode = DISABLE;  //溢出时是否保留上次采样数据
```

- 是否支持过采样：

```
ADC_RegularHandle.Init.OverSampMode = DISABLE;  //禁止过采样
```

- 过采样率：

```
ADC_RegularHandle.Init.Oversampling.Ratio = ADC_OVERSAMPLING_RATIO_8;  //过采样率，为 32bit 无符号数，其中 0~7 分别定义过采样倍数 2~256 倍，具体定义详情在例程的 hal_adc.h 中。
```

- 过采样移位系数：

```
ADC_RegularHandle.Init.Oversampling.RightBitShift = ADC_RIGHTBITSHIFT_3;  //过采样移位系数，为 32bit 无符号数，其中 0~8 分别定义右移位数，具体定义详情在例程的 hal_adc.h 中。
```

- 规则组过采样触发模式：

```
ADC_RegularHandle.Init.Oversampling.TriggeredMode = 0;  //一次触发进行 1 次或 N 次 ADC 转换
```

- 是否支持模拟看门狗：

```
ADC_RegularHandle.Init.AnalogWDGEn = DISABLE;
```

- 使能 ADC 通道：

```
ADC_RegularHandle.Init.ChannelEn = ADC_CHANNEL_1_EN;  //参数 ChannelEn 为 32bit 无符号数，其中 bit0~20 分别定义每个 ADC 通道使能，具体定义详情在例程的 hal_adc.h 中。ADC 对应的具体 GPIO 配置在底层函数 HAL_ADC_MspInit 中完成。
```

- 获取 ADC 寄存器基地址：

```
ADC_RegularHandle.Instance = ADC1;
```

- 其他参数设置按照例程默认设置
- 初始化 ADC 模块：

```
HAL_ADC_Init(&ADC_RegularHandle);
```

2.6.2. 添加 ADC 转换通道

- 设置规则组序列长度

```
ADC_RegularHandle.ChannelNum = 1;
```

设置的规则组序列长度，必须与添加的通道总数一致，必须在初始化过程中已经设置该通道对应的 GPIO 为模拟通道且已经使能，否则可能导致采样数据丢失或者出错。

- 添加 ADC 通道

```
ADC_ChannelConf.Channel = ADC_CHANNEL_1;  //参数 Channel 为 32bit 无符号数，其中 0~19 分别定义每个 ADC 通道，具体定义详情在例程的 hal_adc.h 中。
```

- 转换顺序

```
ADC_ChannelConf.Sq = ADC_SEQUENCE_SQ1;  //参数 Sq 为 32bit 无符号数，其中 1~16 分别定义在规则序列中的转换顺序，具体定义详情在例程的 hal_adc.h 中。
```

- 采样时间

```
ADC_ChannelConf.Smp = ADC_SMP_CLOCK_320;   //参数 Smp 为 32bit 无符号数，其中 0~15 分别定义通道采样时间，具体定义详情在例程的 hal_adc.h 中。
```

- 差分/单端模式

```
ADC_ChannelConf.Diff = DISABLE;  //单端或差分模式
```

- 偏移组选择

```
ADC_ChannelConf.OffsetNumber = ADC_OFFSET_NONE; //对应偏移寄存器 ADC_OFRx
```

- 偏移量

```
ADC_ChannelConf.Offset = 0;  //偏移量，0~4095
```

- 偏移计算方式

```
ADC_ChannelConf.OffsetSign = ADC_OFFSET_MINUS;  //转换结果加上或减去偏移量
```

- 偏移结果格式

```
ADC_ChannelConf.OffsetSaturation = ADC_OFFSET_SIGN_UNSIGNED;  //计算结果为有符号数或无符号数
```

```
HAL_ADC_ConfigChannel(&ADC_RegularHandle, &ADC_ChannelConf);
```

每调用一次 HAL_ADC_ConfigChannel 函数，添加一个 ADC 通道，添加多个 ADC 通道需多次设置。其中参数 Channel 是 ADC 通道号。参数 Sq 为规则序列的序列号，规则序列号必须从 1 开始连续设置，每个序列设置的 ADC 通道号可以是任意 ADC 通道号。

2.6.3. 添加模拟看门狗通道

- 添加模拟看门狗的规则通道

```
ADC_AnalogWDGConf.RegularChannel = ADC_CHANNEL_1; //参数 Channel 为 32bit 无符号数，其中 0~19 分别定义每个 ADC 通道，具体定义详情在例程的 hal_adc.h 中。
```

设置的模拟看门狗的规则通道，必须与 2.5.2 中 ADC 转换通道一致，必须在初始化过程中已经设置该通道对应

的 GPIO 为模拟通道且已经使能，否则可能导致模拟看门狗出错。

- 添加模拟看门狗的注入通道

```
ADC_AnalogWDGConf.InjectChannel = ADC_CHANNEL_1;  //参数 Channel 为 32bit 无符号数，其中 0~19 分别定义每个 ADC 通道，具体定义详情在例程的 hal_adc.h 中。
```

- 选择在所有还是单一的注入或者规则通道上使用模拟看门狗

```
ADC_AnalogWDGConf.WatchdogMode = ADC_ANALOGWATCHDOG_RCH_SINGLE;  //选择在所有还是单一的注入或者规则通道上使用模拟看门狗，具体定义详情在例程的 hal_adc.h 中。
```

- 差分/单端模式

```
ADC_AnalogWDGConf.Diff = DISABLE;  //单端或差分模式
```

- 模拟看门狗中断设置

```
ADC_AnalogWDGConf.ITMode = ENABLE; //使能中断
```

- 设置模拟看门狗的高阈值

```
ADC_AnalogWDGConf.HighThreshold = 0xA00;  // 0~4095
```

- 设置模拟看门狗的低阈值

```
ADC_AnalogWDGConf.LowThreshold = 0x200;  //0~4095
```

- 模拟看门狗的中断回调函数

```
ADC_RegularHandle.LevelOutOfWindowCallback = ADCx_AnalogWDGCallback;
```

```
HAL_ADC_AnalogWDGConfig(&ADC_RegularHandle, &ADC_AnalogWDGConf);
```

调用 HAL_ADC_AnalogWDGConfig 函数，进行模拟看门狗配置。

3. ADC 数据采集

3.1. 轮询方式

运行 HAL_ADC_Polling 获取 ADC 通道数据。轮询方式受限于 MCU 运行速度，无法支持太高采样频率，实际采样频率和 MCU 速度和应用有关，在应用中测试。

3.2. 中断方式

中断方式可以方便的处理注入采样，看门狗事件等。

初始化完通道，配置好中断回调函数。通过执行 HAL_ADC_Start_IT 函数启动规则通道采样，通过执行 HAL_ADC_InjectedStart_IT 函数启动注入通道采样（如果开启注入通道）

3.3. 外部触发方式

初始化完通道，配置好外部触发事件。启动通道采样，等待外部触发。

3.4. DMA 方式

初始化完通道，配置好 DMA 通道，运行 HAL_ADC_Start_DMA 获取 ADC 通道数据。

3.5. 数据格式

获取到的数据中包含通道号和采样结果，bit16~20 为通道号，bit0~15 为采样结果。设置为单端模式时，采样结果为 12bit 无符号数。设置为差分模式时，采样结果为 16bit 补码形式有符号数，符号位为最高位 bit15，在未使用过采样功能的情况下，仅低 11 位数据有效。

3.6. 结果计算

采样结果计算，根据采样结果 outdata 和 ADC 参考电压 VREF 计算采样结果电压 Vadc。

以 12bit 分辨率为例：

单端模式，outdata 和 Vadc 为无符号数：Vadc = outdata * VREF/ 4095;

差分模式，outdata 和 Vadc 为有符号数：Vadc = outdata * VREF/ 2047-VREF;

4. 应用例程说明

提供规则通道、注入通道、模拟看门狗、双 ADC 模式应用例程。可以在 APP_Test 函数中通过修改 ADC_Test 函数的输入参数来切换不同 demo。包含轮询、中断、外部触发、DMA 方式，以及模拟看门狗、差分模式、规则同步+注入同步模式、温度传感器示例。

4.1. 设置 VREFP 来源

在 VREF-接地的情况下，VREF+的值即是用于计算最终 ADC 采样结果电压时所用的参考电压值。由于 VREFP 可能来自外部输入，也可以选择来自内部电压参考缓冲器 VREFBUF。可以通过设置 ADC 共用电压参考缓冲器寄存器 CVRB 的[1:0]位来选择。当选择来自内部电压参考缓冲器 VREFBUF 时，可以通过设置 ADC 共用电压参考缓冲器寄存器 CVRB 的[3:2]位来选择参考电压值（00：1.5V，01：1.8V，10：2.0V，11：2.5V）。具体见例程中函数 ADC_VREFP_Config ()。

4.2. ADC 自校准

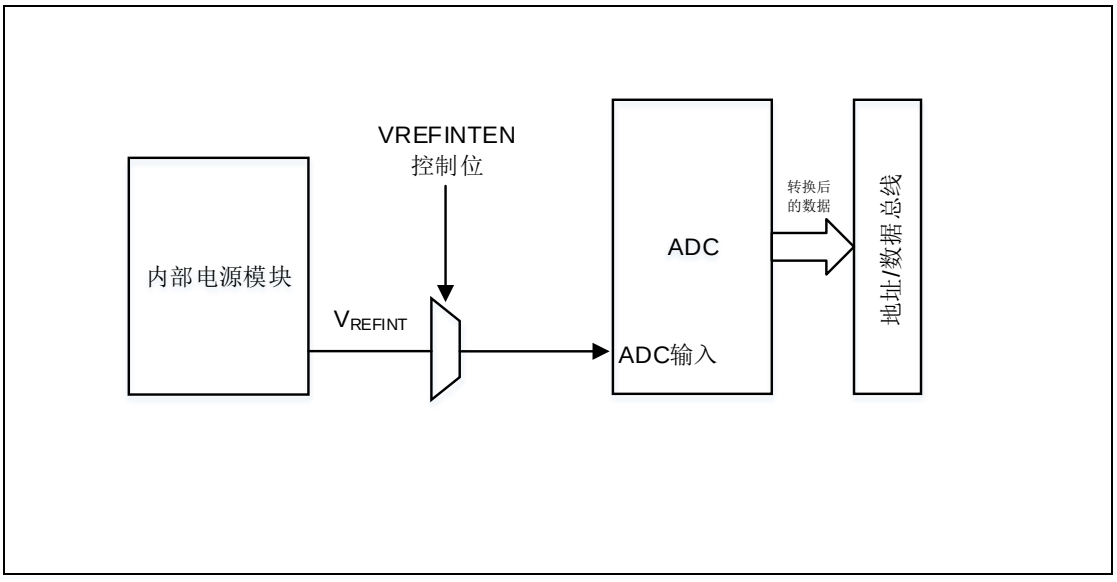
ADC1、ADC2、ADC3 的通道 0（ADC_INP/INN0）为校准通道。为了提高 ADC 采样精度，需要先进行自校准。可以选择软件校准，也可以选择硬件校准。具体见例程中函数 ADC_Calibration ()。

4.3. ADC 获取 VREFP

在 VREF-接地的情况下，VREF+的值即是用于计算最终 ADC 采样结果电压时所用的参考电压值。由于 VREFP 可能来自外部输入，因此不同的项目 VREFP 接的外部参考电压不一致；有时候对 ADC 的参考电压精度有更高的要求，而板级之间 VDDA 存在着差异。在这些情况下，可以通过监测内部参考电压来获得用于评估 ADC VREF+电压的参考值，内部参考电压 V_{REFINT} 在内部连接到输入通道 ADC2_INP[13]（BGR 通道），利用出厂时 VREFP 为 3V 采集的带隙电压校准值（存储在 EFUSE 区，见例程）， 来计算出当前的 VREFP。具体见例程中函数 ADC_GetVrefP()。

注意：

1、如果用户有高精度的已知基准电压源或者对 ADC 采样精度要求不是特别高（电机应用通常要求较高），可以省略该步骤，直接使用已知电压值进行计算。



5. 4.常见问题汇总

5.1. VREFP 来源与用法

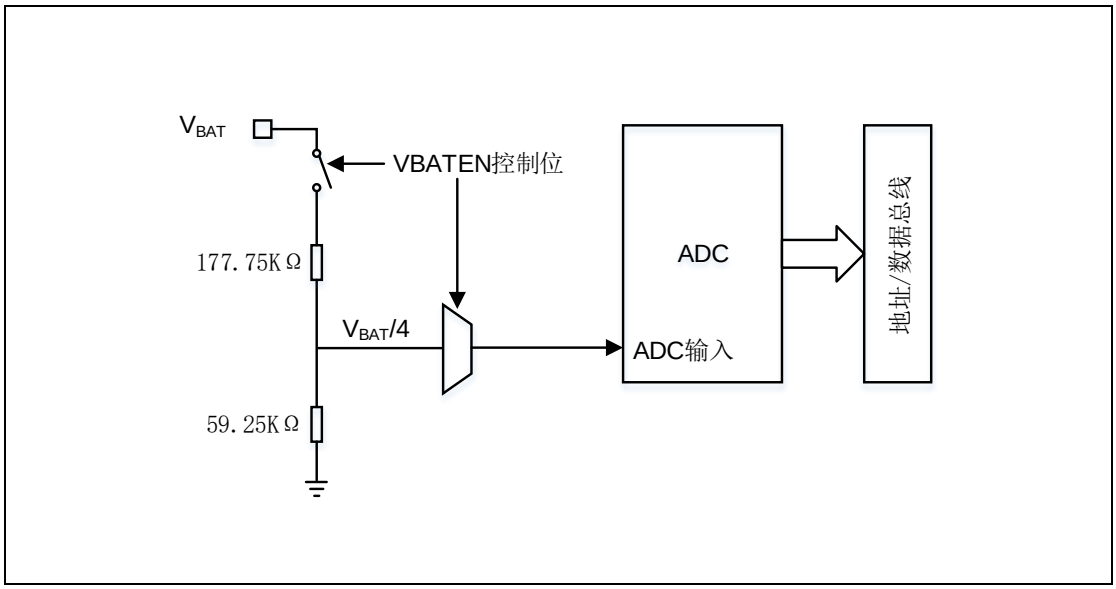
- 对于 ACM32H5 系列，部分封装芯片的 VREFP 引脚有引出；部分封装芯片的 VREFP 和 VDDA 在芯片内部进行了短接。具体封装信息请见数据手册。
- VREFP 和 VDDA 内部短接的封装，VREFP 的电压等于 VDDA 的电压，ADC 的正向参考电压等于 VDDA 的电压；VREFP 和 VDDA 内部未短接的封装，ADC 的参考电压默认使用 VREFP 引脚所接的外部电压，也可以使用 ADC 模块内嵌 VREF 电压，详见 ADC_CVRB 寄存器。

5.2. 输入电压采用外部电阻分压方式对采样时间的影响

当采样输入是经由外部电阻分压输送到 ADC 引脚时，相当于有电阻串联到 ADC 内部的充电电路，充电时间会延长，相对应的采样时间也会延长，建议此时软件增大采样周期。

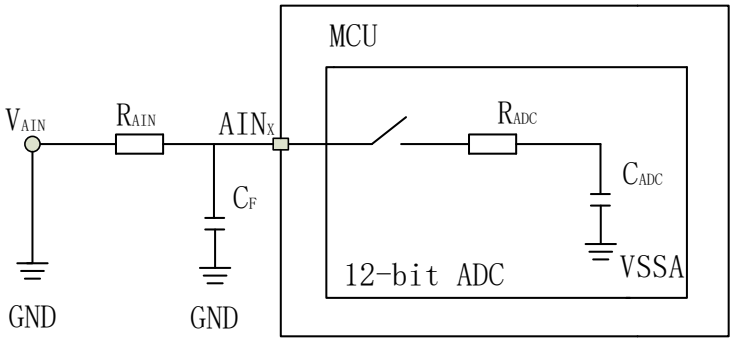
5.3. VBAT 分压电路

ADC_CCR 寄存器中的 VBATEN 位用于切换到 VBAT 电池监测，由于 VBAT 电压可能高于 VDDA，因此 VBAT 引脚需要从内部连接到桥接分配器（除以 4），以确保 ADC 正确运行。VBATEN 位置 1 时，会自动使能此桥，以将 VBAT/4 连接到 ADC3_INP[17]输入通道。因此，转换出的数字值为 VBAT 电压的 1/4。为防止电池出现意外的电能消耗，建议仅在必须要时为 ADC 转换使能桥接分配器。



6. ADC RC 电路充放电时间

ADC 采样等效电路如下图所示：



注意：为了提高稳定性，降低噪声干扰，可外部添加电容 CF。此时 RAIN 与 CF 构成 RC 滤波电路。

6.1. ADC RC 电路充电放电时间公式

$$Vt = V0 + (V1 - V0) * (1 - e^{-t/RC})$$

$$T = RC \ln(\frac{V1 - V0}{V1 - Vt})$$

- RC 电路的时间常数：τ=RC（R=RAIN+RADC，C=CADC，RADC=50Ω，CADC=5pF）；
- V0 为电容初始电压值；
- V1 为电容充满电压值；
- Vt 为任意时刻的电压值；
- $t = \frac{T_S}{f_{sADC}}$，fADC 为 ADC 时钟，Ts 为采样周期数；
- E 为 ADC 管脚输入电压（采集电压）；
  - 当 t= 1RC 时，电容电压=0.63*V1；
  - 当 t= 2RC 时，电容电压=0.86*V1；
  - 当 t= 3RC 时，电容电压=0.90*V1；
  - 当 t= 4RC 时，电容电压=0.98*V1；
  - 当 t= 5RC 时，电容电压=0.99*V1；
  - 经过 3~5 个 RC 后，充电过程基本结束。

6.2. ADC RC 电路充电时间计算

电压为 E 的电源通过 R 向初值为 0 的电容 C 充电, V0=0，V1=E，故充到 t 时刻电容上的电压为

$$Vt = V1 * (1 - e^{-t/RC})$$

$$T = RC \ln(\frac{V1}{V1 - Vt})$$

例如当 RAIN 为 1KΩ ，ADC 管脚输入电压 E 为 2V，ADC 采样时钟为 40MHZ，采样周期数为 3，Vt 能充到多少：

$$RC = (1000 + 50) * 5 * 10^{-12} = 5.25 * 10^{-9}$$

$$t = \frac{3}{40} * 10^{-6} = 7.5 * 10^{-8}$$

$$Vt = 2 * (1 - e^{-14.285}) = 1.999\,V$$

例如当 RAIN 为 1KΩ ，ADC 管脚输入电压 E 为 2V，充到 Vt 为 1.999V 需要的时间：

$$RC = (1000 + 50) * 5 * 10^{-12} = 5.25 * 10^{-9}$$

$$T = 5.25 * 10^{-9} * \ln\left(\frac{2}{2 - 1.999}\right) = 39.9 * 10^{-9} = 39.9\,ns$$

6.3. ADC RC 电路放电时间计算

初始电压为 E 的电容 C 通过 R 放电 ，V0=E，V1=0，故放到 t 时刻电容上的电压为

$$Vt = \mathrm{V0} - V0 * (e^{-t/RC})$$

$$T = RC \ln(\frac{V0}{Vt})$$

例如 RAIN 为 1KΩ，初始电压 E 为 2V，ADC 采样时钟为 40MHZ，采样周期数为 3，Vt 能放到多少：

$$RC = (1000 + 50) * 5 * 10^{-12} = 5.25 * 10^{-9}$$

$$t = \frac{3}{40} * 10^{-6} = 7.5 * 10^{-8}$$

$$Vt = 2 - 2 * (1 - e^{-14.285}) = 0.001\,V$$

例如当 RAIN 为 1KΩ ，ADC 管脚输入电压 E 为 2V，放到 Vt 为 1.999V 需要的时间：

$$RC = (1000 + 50) * 5 * 10^{-12} = 5.25 * 10^{-9}$$

$$T = 5.25 * 10^{-9} * \ln\left(\frac{2}{0.001}\right) = 39.9 * 10^{-9} = 39.9\,ns$$

7. ADC 外部输入阻抗计算

$$R_{AIN} = \frac{T_s}{f_{ADC} * C_{ADC} * \ln(2^{N+Y})} - R_{ADC}$$

- RAIN 为外部输入阻抗；
- Ts 为 ADC_SMPR 寄存器中定义的采样周期数；
- fADC 为 ADC 时钟，ADC_CCR 寄存器中的 ADCDIV 定义了 fADC 相对于 HCLK 或 PLL3_P_CLK 的分频数，fADC 最大不能超过 75MHZ；
- CADC 为内部采样和保持电容，CADC=5pF；
- RADC 为内部开关电阻，RADC=50Ω；
- N 为分辨率，N=12（12bits 分辨率）；
- Y 为精度（采集误差），Y 为正数时，精度为 1/2Y LSB；Y 为负数时，精度为 1*2Y LSB。（$1\ LSB = \frac{V_{refp}}{2^N}$）
  - Y=2，表示 1/4 LSB；
  - Y=1，表示 1/2 LSB；
  - Y=0，表示 1 LSB；
  - Y=-1，表示 2 LSB；
  - Y=-2，表示 4LSB。

例如 ADC 采样时钟为 75MHZ，采样周期数为 3，误差为 1 LSB，RAIN 外部输入阻抗为：

$$R_{AIN} = \frac{3}{75 * 10^6 * 5 * 10^{-12} * \ln(2^{12})} - 50 = 961.8\Omega$$

8. ADC 采样率计算

$$f_{SAMPLE} = \frac{f_{ADC}}{T_s + bit}$$

- fSAMPLE 为 ADC 采样率；
- fADC 为 ADC 时钟，ADC_CCR 寄存器中的 ADCDIV 定义了 fADC 相对于 HCLK 或 PLL3_P_CLK 的分频数，fADC 最大不能超过 75MHZ；
- Ts 为 ADC_SMPR 寄存器中定义的采样周期数;
- bit 为 12/10/8/6，由 ADC_CR2 寄存器中的 RES 设置；

例如 12bit 分辨率，采样周期为 3，采样率：

$$f_{SAMPLE} = \frac{75}{3 + 12} = 5\ Msps$$

9. 版本历史

版本	日期	作者	描述
V1.0	2024-11-19	Aisinochip	初始版
V1.1	2024-12-11	Aisinochip	新增结构体说明，以及 RC 放电计算示例

联系我们

公司：上海航芯电子科技股份有限公司

地址：上海市闵行区合川路 2570 号科技绿洲三期 2 号楼 702 室

邮编：200241

电话：+86-21-6125 9080

传真：+86-21-6125 9080-830

Email： service@AisinoChip.com

Website： www.AisinoChip.com