Contents

[DIO 1](#_Toc123109210)

[PORTS 2](#_Toc123109211)

[SPI 2](#_Toc123109212)

[SpiChannelBuffersAllowed 2](#_Toc123109213)

[SpiLevelDelivered 4](#_Toc123109214)

[Channel、job、sequence之间的关系（孙子，儿子，爸） 5](#_Toc123109215)

[ADC 6](#_Toc123109216)

[一、ADC模块介绍 6](#_Toc123109217)

[1、ADC模块的功能 6](#_Toc123109218)

[2、模块相关概念首字母缩略介绍： 6](#_Toc123109219)

[二、基于TC397的EB\_MCAL\_ADC模块介绍 7](#_Toc123109220)

[1.Config Variant与AdcConfigSet 7](#_Toc123109221)

[2. AdcGeneral 8](#_Toc123109222)

[3.AdcPublishedInformation 11](#_Toc123109223)

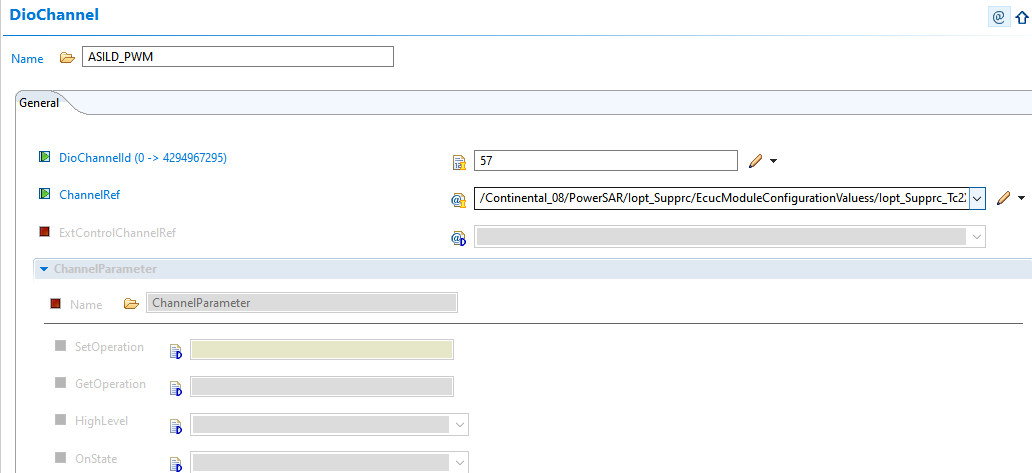
# DIO

1. DioChannelId

不需要配置，在预处理执行期间会自动配置。

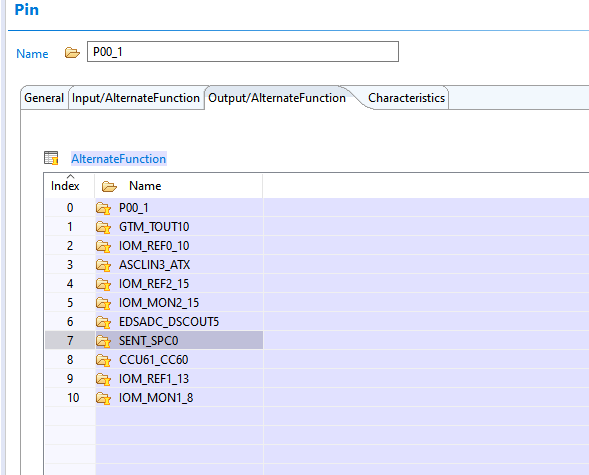
2.ChannelRef

配置好对应的引脚编号。

疑问：引脚编号根据什么进行选择？

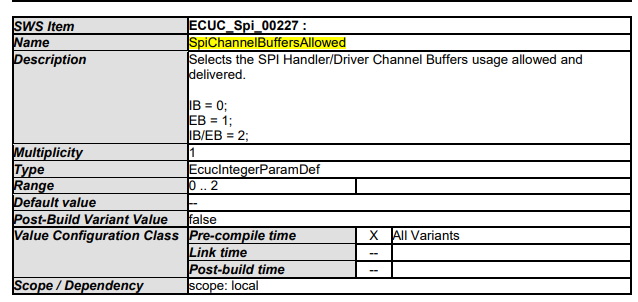
# PORTS

1.在PORT里，列举出了所有的端口以及内置的功能，并且端口号和功能进行对应。

疑问：PORTS配置的一堆功能是根据哪里？

# SPI

## SpiChannelBuffersAllowed





IB: 静态分配。 EB: 可以是静态分配，也可以是动态分配，取决于用户的使用环境。

有些硬件层面提供了比较大的Buffer，这种情况IB类型可以充分发挥硬件特性，提高其性能（这是IB Buffer的设计初衷）（注意，如果有多个channel同时挂到一个device上面，则该功能使用有限制）。如果硬件没有Buffer，则需要软件来模拟实现。

IB类型的Buffer其大小是固定的。EB类型Buffer可以通过API进行设置。

SPI驱动不负责保证IB Buffer里数据的连续性。如果某个Channel被多个Job/Seq使用，SPI驱动也不负责维护该Buffer被多个Job/Seq重写这种场景。

但是发送和接收的Buffer是分开独立的。也就是发送Buffer不会被接收的数据覆盖。

EB Buffer的设计初衷是为了尽量重用外部（这里指用户）Buffer，因为很多情况下，用户已经有了一个Buffer，那么使用EB类型Buffer，只需要将用户Buffer的指针提供给SPI驱动以达到共用的目的。所以SPI驱动也是无法对该Buffer管理的，需要用户来保证其一致性。还有一种场景是，有时候我们的Buffer大小是变化的（比如多个使用者的需求可能不同，或者一个使用者数据长度是变化的），这种情况也需要使用EB类型Buffer来解决(因为IB类型Buffer是固定大小的)。当然EB类型Buffer也可以是固定大小。但是Buffer大小的最大值需要静态配置。总的来说EB类型Buffer使用更灵活。

内部缓冲通道（IB）：用于传输/接收数据的缓冲区由处理程序/驱动程序提供。

外部缓冲通道（EB）：用于传输/接收数据的缓冲区由用户提供（静态和/或动态）。

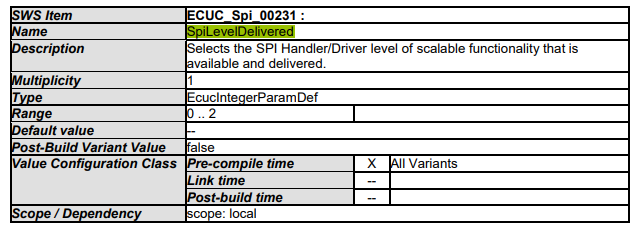
IB：静态分配；适用于硬件提供了较大的Buffer，可以提高性能，但在通信期间无法处理缓冲区中数据的一致性，接收和传输应该提供单独的缓冲区，以确保传输的数据不被接收数据覆盖；

EB：可以静态分配、也可以动态分配。适用于用户已经配置了一个Buffer，SPI可直接使用用户buffer的指针来达到共用的效果。

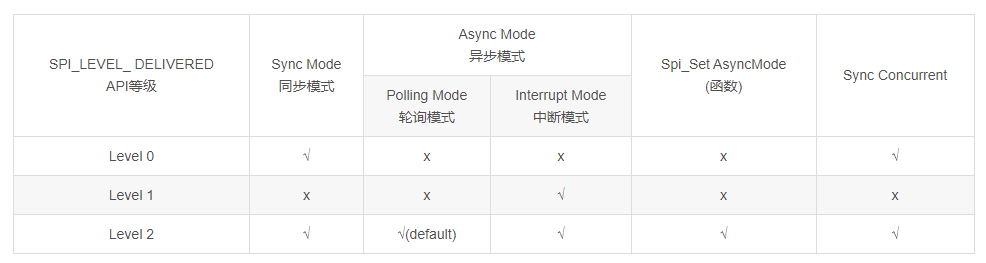
对于EB通道，应用程序应提供缓冲，并在传输过程中注意缓冲区中数据的一致性。不同点：IB类型的Buffer的大小是固定的，EB类型的Buffer的大小可以固定也可以不固定。

**一般配置为EB，也就是2.**

## SpiLevelDelivered



根据配置，相关参数和功能会在预编译或其他阶段进行自配置。



**LEVEL 0**： 提供一组简化的服务，只处理简单的同步传输。对于包含简单SPI网络的ECU来说，这种情况经常发生，即便对于使用高速外部设备的ECU来说也是如此。

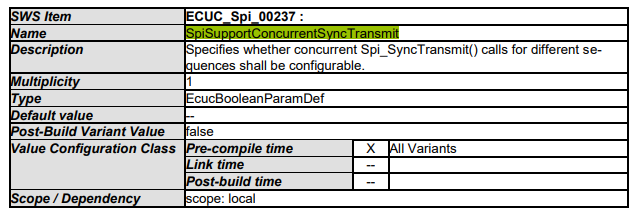
同步传输意味着一旦调用了传输服务函数，在该传输请求完成之前，程序将会一直被阻塞（一个好的代码实现，通常也会有超时监控机制，超时后也会释放CPU）。

**LEVEL 1**： 提供一组简化的服务，只处理异步传输。对于那些使用SPI且定义了不同优先级的功能来说，异步传输经常被用到。还有一些低速外设也同样适合该异步模式。

异步传输意味着当传输正在进行时，调用传输服务的用户不会被阻塞。驱动程序可以在传输结束时通知用户（可由用户进行配置）。异步传输模式又可以分为通过轮询或中断来实现。

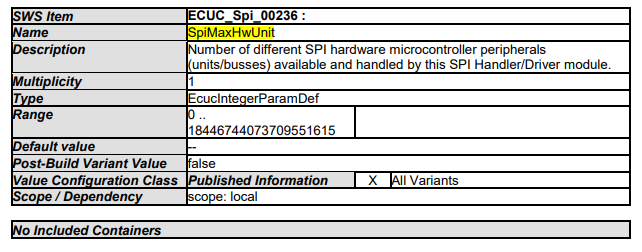
**LEVEL 2**： 前面两种的组合。包含全部功能。可满足不同速率，不同优先级等复杂需求场景。是为那些需要单独提供至少两路SPI总线的芯片而指定的。 否则，使用这种级别的功能没有意义。

SpiSupportConcurrentSyncTransmit



指定对不同序列的并发Spi\_SyncTransmit()调用是否可配置。

SpiMaxHwUnit



可用的不同SPI硬件微控制器外设(单元/总线)的数量，并由此SPI处理程序/驱动程序模块处理。（根据SPIDrive中的VendorDevice数量选择）

## Channel、job、sequence之间的关系（孙子，儿子，爸）

Channel是使用相同条件定义的数据的软件交换介质：配置参数、具有相同大小的数据元素数以及数据指针（源和目标）或位置。

Job由具有相同芯片选择的一个或多个Channel组成（在处理Job期间不会释放），Job具有分配的优先级。

Sequence是连续Job的数量，但可以使用优先级机制在Job之间重新调度。Sequence通信是可中断的（通过另一个Sequence通信）或不依赖于静态配置。

# ADC

## 一、ADC模块介绍

### 1、ADC模块的功能

初始化并控制微控制器的内部模拟数字转换器单元(s)。它提供启动和停止转换的服务，以启用和禁用转换的触发源。此外，它还提供服务来启用和禁用通知机制和例程，以查询转换的状态和结果。

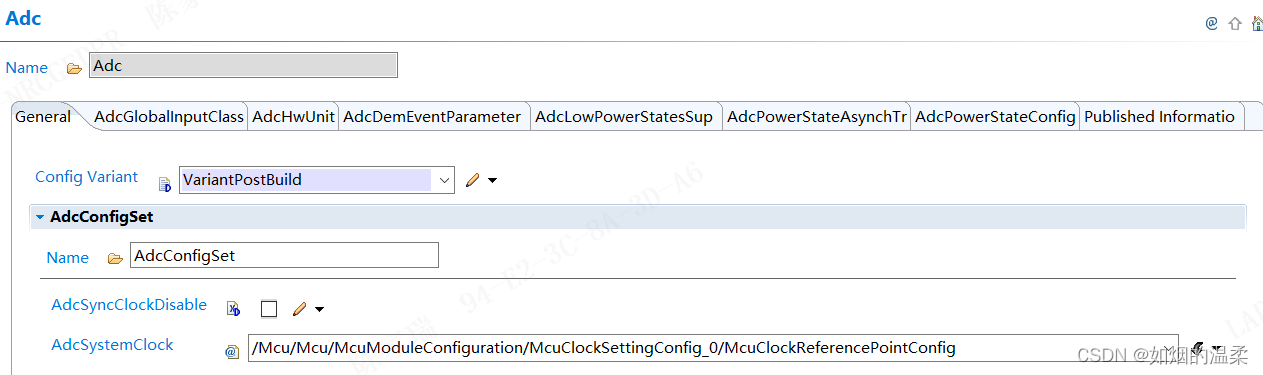
ADC模块在的ADC通道组上工作，这是由的ADC通道构建的。ADC通道组将模拟输入引脚（ADC Channel）、所需的ADC电路本身和转换结果寄存器组合成一个实体，它可以通过ADC模块进行单独控制和访问。

### 2、模块相关概念首字母缩略介绍：

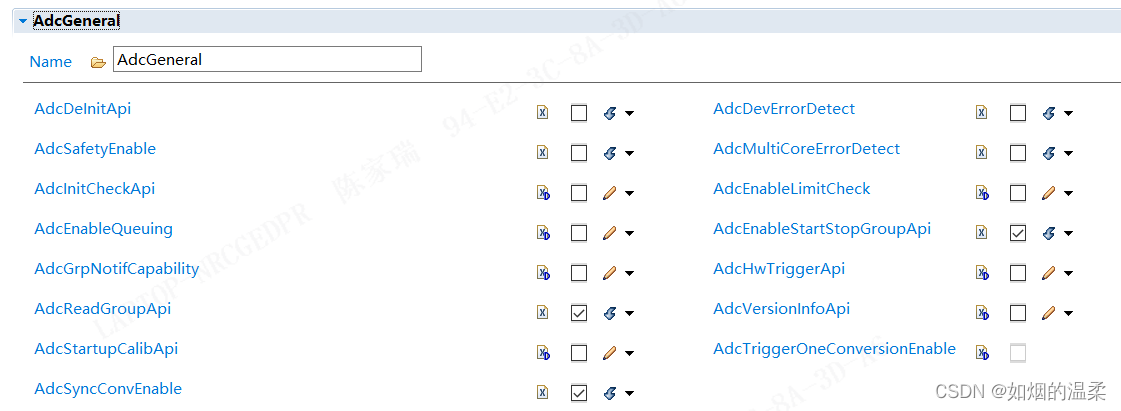
**DEM** ：Diagnostic Event Manager 诊断事件管理  
**DET** ：Default Error Tracer 默认错误跟踪器  
**ADC** :Analogue Digital Converter 模数转换器  
**MCU** :Microcontroller Unit 微控制器管理单元  
**API** :Application Programming Interface 应用程序接口  
**HW** :Hardware 硬件  
**SW** :Software 软件  
**ADC HW Unit** : ADC硬件模块，表示一种微控制器输入电子设备，它包括执行“模拟到数字转换”所需的所有部件。  
**ADC Module** ：ADC基本软件模块ADC驱动程序，也缩写为ADC驱动程序  
**ADC Channel** ：表示绑定到一个端口引脚的逻辑ADC实体。多个ADC实体可以映射到同一个端口引脚。  
**ADC Channel Group** ：连接到同一ADC硬件单元的一组ADC通道（例如，一个采样和保持转换器和一个A/D转换器）。整个组的转换是由一个触发源触发。  
**ADC Result Buffer (ADC Streaming Buffer, ADC Stream Buffer)** ：ADC驱动程序的用户必须为每个组提供一个缓冲区。如果选择了流媒体访问模式（streaming access mode）该缓冲区可以保存同一组通道的多个样本。如果选择了单一访问模式，缓冲组中保留每个通道的一个样本。  
**Software Trigger** ：启动一个ADC通道组或连续一系列ADC通道组转换的软件API调用  
**Hardware Trigger** ：启动ADC通道组的一次转换的ADC内部触发信号。ADC硬件触发器在ADC硬件内部生成，例如基于ADC计时器或触发器边缘信号。触发器硬件是紧密耦合的或集成在ADC硬件中的。在检测到硬件触发器后，无需使用任何软件即可启动ADC通道组转换。注意：如果ADC硬件不支持硬件触发器，则将软件触发器与GPT/ICU驱动程序结合使用，可以实现类似的行为。例如，在GPT计时器通知功能中，可以启动软件触发的ADC通道组转换。  
**Conversion Mode** ：  
**One-Shot**: ADC通道组的转换在触发后执行一次，并将结果写入分配的结果缓冲区。触发器可以是软件API调用或硬件事件。  
**Continuous:** ADC通道组的转换在软件API调用（开始）后连续执行，并将结果写入分配的结果缓冲区。转换本身正在自动运行（受硬件/中断控制）。连续转换可以通过软件API调用（停止）来停止。  
**Sampling Time, Sample Time** ：模拟值采样的时间（例如加载电容器，  
**Conversion Time** ：将采样的模拟值转换为数字表示的时间。  
**Acquisition Time** ： Sample Time + Conversion Time（样本时间+转换时间）

## 二、基于TC397的EB\_MCAL\_ADC模块介绍

### 1.Config Variant与AdcConfigSet

  
**Config Variant** ：变体配置，默认选择VariantPostBuild就好了；  
**AdcSyncClockDisable** ：此参数确定模拟时钟是否在同步/非同步模式下生成。转换器控制（CONVCTRL）时钟的配置由MCU驱动器完成；  
**AdcSystemClock** ：这个参数是指MCU驱动程序配置的系统时钟。使用这个系统时钟，给ADC组的计时器触发器重新加载值（触发器也是一个计时器，相当于ADC模块的计时器需要一个标准时间，就像现实里会用伦敦时间，北京时间作为标准，系统时钟就是起到这个作用）；

### 2. AdcGeneral



**AdcDeInitApi** ：从代码中添加/删除服务Adc\_DeInit()，若勾选，则可使用去初始化功能；

**AdcDevErrorDetect** ：打开或关闭默认错误跟踪器(Det)的检测和通知。

**AdcSafetyEnable**：启用/禁用安全检查和所有相关通知；启用ADC驱动程序安全特性的预处理器开关。通过打开此开关，隐式地启用了这些安全功能：对API参数进行了范围检查；ADC配置结构有一个唯一的标记手段，该手段将由驱动程序验证；发布ADC驱动程序的安全使用情况，供用户设置相应的ADC设置；（AUTOSAR标准文档中未规定，而是TC\_397芯片自带的功能）

**AdcMultiCoreErrorDetect** ：从代码中添加/删除多核错误检测功能

**AdcInitCheckApi** ：从代码中添加/删除服务Adc\_InitCheck()，该服务用于验证ADC驱动程序完成的初始化（AUTOSAR标准文档中未规定，而是TC\_397芯片自带的功能）；

**AdcEnableLimitCheck**：启用或禁用ADC驱动程序中的限制检查功能；确定在优先级机制禁用的情况下，队列机制是否处于活动状态；（全局启用限制检查，若未启用，则AdcChannelHighLimit和AdcChannelLowLimit、AdcChannelRangeSelect不可用）（AUTOSAR标准文档中未规定，而是TC\_397芯片自带的功能）

**AdcEnableQueuing**：确定在优先级机制禁用的情况下，队列机制是否处于活动状态；如果启用了优先级机制，则队列机制始终处于活动状态，而参数ADC\_ENABLE\_QUEUING不会被求值；

**AdcEnableStartStopGroupApi** ：从代码中添加/删除服务Adc\_StartGroupConversion()和Adc\_StopGroupConversion()；

**AdcGrpNotifCapability** ：确定组通知机制(启用和禁用通知的功能)在运行时是否可用。AdcNotification函数必须在该API启用时才可用；

**AdcHwTriggerApi**：从代码中添加/删除服务Adc\_EnableHardwareTrigger()和Adc\_DisableHardwareTrigger()

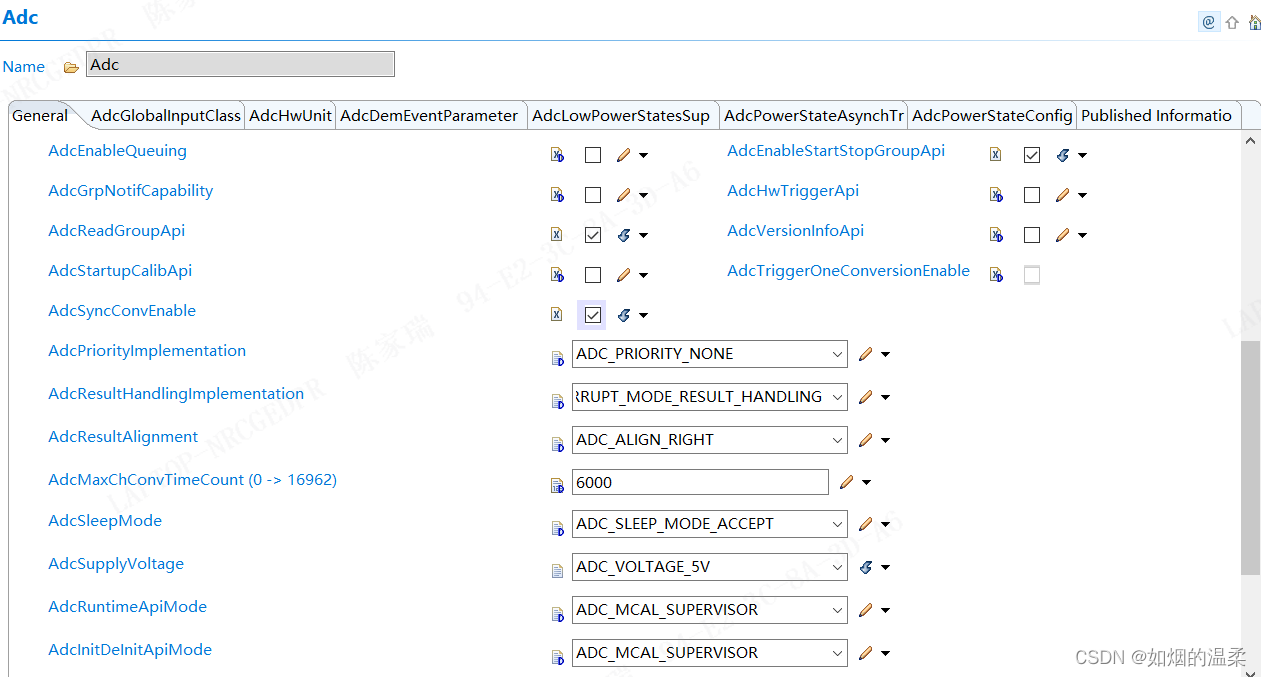
**AdcReadGroupApi** ：从代码中添加/删除服务Adc\_ReadGroup()。

**AdcVersionInfoApi** ：从代码中添加/删除服务Adc\_GetVersionInfo()。

**AdcStartupCalibApi**：从代码中添加/删除服务Adc\_GetStartupCalStatus()和Adc\_TriggerStartupCal()

**AdcTriggerOneConversionEnable** ：该参数允许在API—Adc\_TriggerStartupCal()中触发启动校准之前，为每个配置的HW单元执行一个虚拟转换。

**AdcSyncConvEnable** ：支持ADC HW组之间的同步转换。

  
**AdcPriorityImplementation** ：  
确定转换请求的优先级机制是否可用，如果可用，则确定优先级机制的类型。此选择适用于具有软件触发源和硬件触发源的组；  
有三种优先级机制可以选择分别是：  
λAdcPriorityHw（硬件优先级机制）:只有硬件优先级机制可用  
λ(AdcPriorityHwSw)：硬件和软件优先级都可用；  
λ ADC\_PRIORITY\_NONE ：优先级机制不可用；  
软件触发组的组优先级通常配置为低于硬件触发组的组优先级级别；

**AdcResultHandlingImplementation**：确定ADC驱动程序的结果处理模式；  
ADC\_INTERRUPT\_MODE\_RESULT\_HANDLING:转换结果在请求源ISR (AUTOSAR模式)中传输。  
ADC\_DMA\_MODE\_RESULT\_HANDLING:使用DMA通道传输转换结果。ADC\_POLLING\_MODE\_RESULT\_HANDLING:使用同步api传输转换结果。

**AdcResultAlignment** ：ADC结果缓冲区中的ADC原始结果对齐方式(左/右对齐)。

**AdcMaxChConvTimeCount** (0 -> 16962)  
在转换请求停止时，A/D转换器中正在进行的信道转换不能立即停止。相反，软件需要等待信道转换完成。为了确保转换器处于空闲状态，需要对转换器状态轮询进行等待，其最大等待时间由AdcMaxChConvTimeCount中输入的值定义

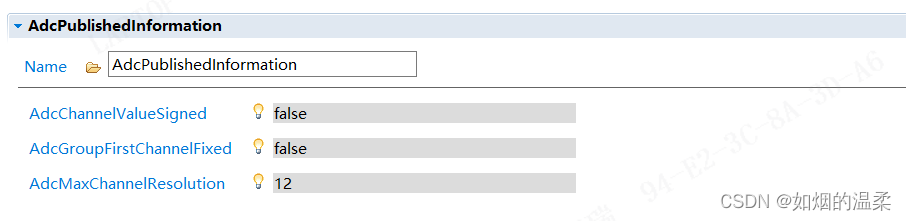
**AdcSleepMode** ：  
该参数决定ADC驱动程序是否接受或拒绝来自SCU的休眠模式请求。  
ADC\_SLEEP\_MODE\_ACCEPT: 接受SCU的睡眠模式请求 ；  
ADC\_SLEEP\_MODE\_REJECT: 拒绝SCU的睡眠模式请求；

**AdcSupplyVoltage**  
该参数将模拟电路调整到所选的电源电压，该参数的默认值为对应SFR的重置值  
ADC\_VOLTAGE\_3P3V: 3.3V固定电源接通；  
ADC\_VOLTAGE\_5V: 固定5V电源接通；  
ADC\_VOLTAGE\_CONTROLLED\_BY\_SUPPLY: 电压范围由电源控制；

**AdcRuntimeApiMode**  
参数定义运行时api将在其中操作的特权模式。  
由于ADC驱动程序访问SFRs，在监控器模式下操作ADC驱动程序更加高效。因此，默认的操作模式是supervisor。  
当AdcInitDeInitApiMode配置为User-1模式时，AdcRuntimeApiMode必须配置为User-1模式

**AdcInitDeInitApiMode**  
配置参数定义初始化和反初始化api将在其中操作的特权模式。  
由于ADC驱动程序访问SFRs，在监控器模式下操作ADC驱动程序更加高效。因此，默认的操作模式是supervisor。

### 3.AdcPublishedInformation

  
**AdcPublishedInformation**：  
关于PublishedInformation(发布信息)  
这个模块包含了由BSW模块的实现者定义的数据，当模块适应(即配置)到实际的硬件和软件环境时，这些数据不会改变。它包含版本和制造商信息。这对于为每个BSW提供明确的版本标识是必要的模块。这个模块一般不用配置，保持默认即可，因为主要与硬件相关的部分有关。