YTMicro MCAL应用 - Adc模块配置及实例

版本

Config Tool Version：2.7.7

ME0 MCAL version：2.3.0

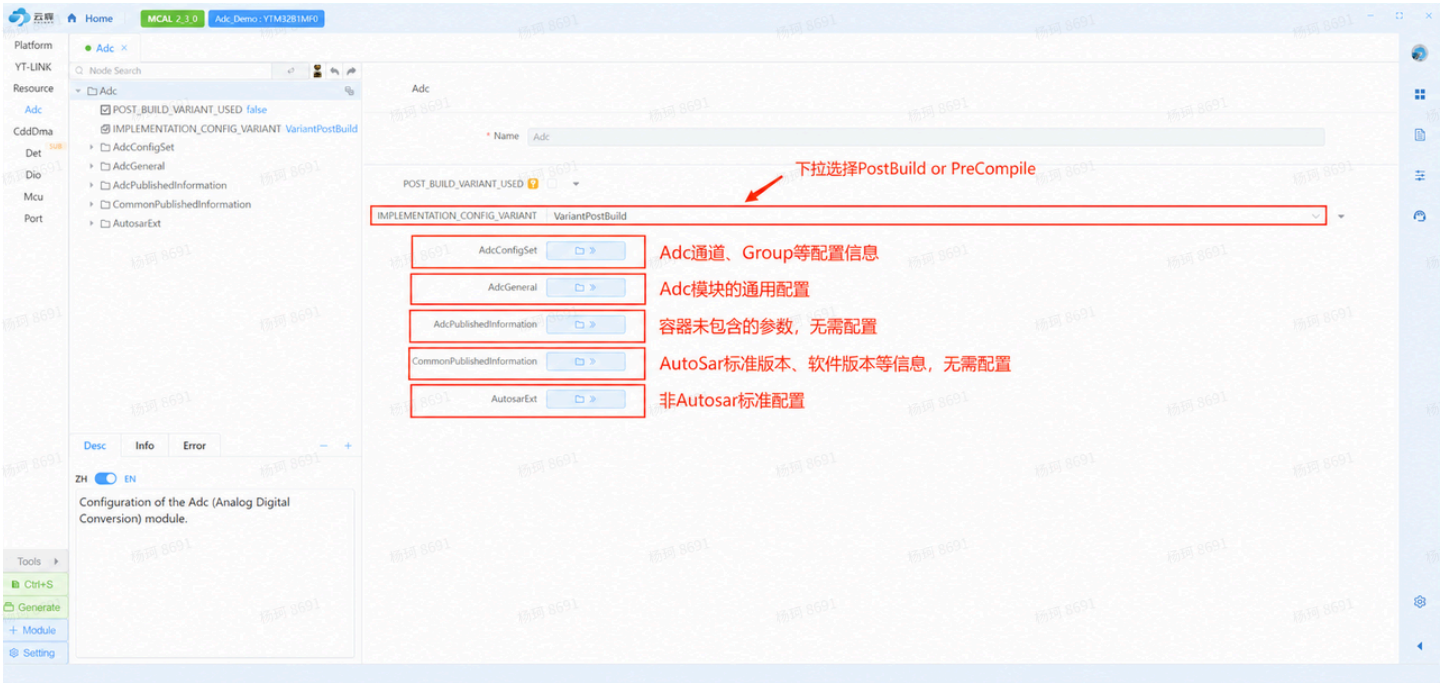
前言

1. Adc一般依赖Mcu和Platform，Mcu模块里面需要配置外设时钟，Platform里面配置外设中断。如果选择DMA方式则需要添加CddDMA模块。
2. Mcu和Platform的配置参考《MCAL应用 - 工程建立及工具通用设置》和《MCAL应用 - Mcu模块配置及实例》文档，CddDMA的配置参考《MCAL应用 - CddDMA模块配置及实例》
3. 配置中包含AutoSar规范标准选项和云途自定义选项，左边树选框中粗体字为AutoSar标准选项，细体字为云途自定义选项。

ADC配置介绍

1、如何配置ADC模块

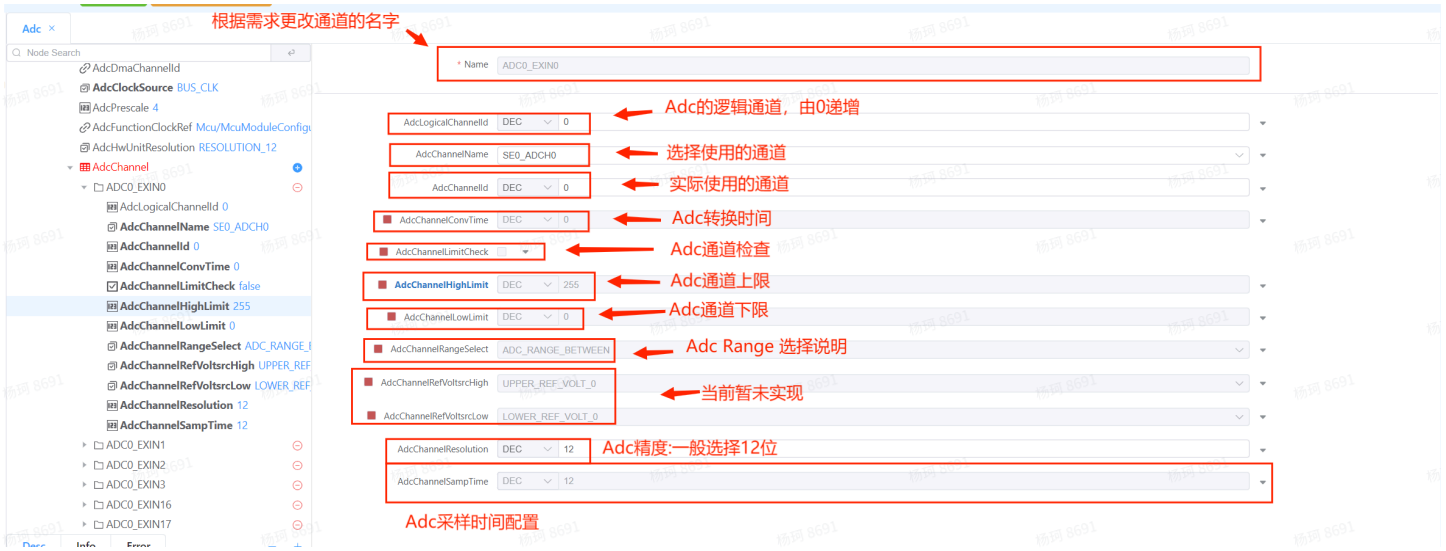
1.1 框架说明



1.2 AdcConfigSet



1.2.1 AdcChannel



Adc Channel Range select解释说明:

Channel Range Select

这个配置是在Autosar规范中定义的。类型如下:

选项	描述
ADC_RANGE_ALWAYS	完全范围，独立于通道限制设置
ADC_RANGE_BETWEEN	下限和上限之间的范围-包括高限制值
ADC_RANGE_NOT_BETWEEN	超出上限或低于下限的范围-包括下限值
ADC_RANGE_NOT_OVER_HIGH	范围低于高限值-包括高限值
ADC_RANGE_NOT_UNDER_LOW	范围高于下限值
ADC_RANGE_OVER_HIGH	范围高于上限值
ADC_RANGE_UNDER_LOW	范围低于下限制-包括下限值

Tips:

任何Channel都必须挂靠到Group下面，上层/API都是操作Group的，不能直接去操作Channel。

Tips:

每个Group下面可以挂一个或多个Channel，但至少有一个Channel。

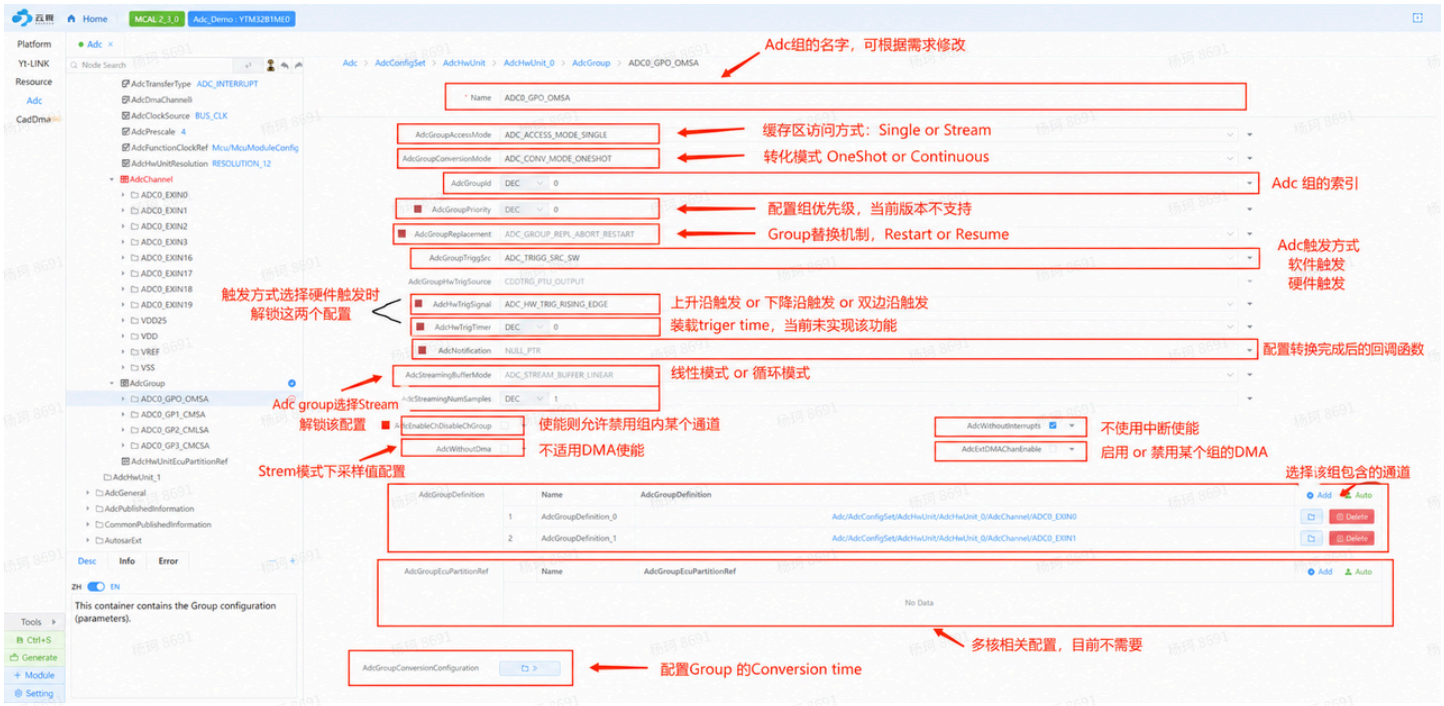
Tips:

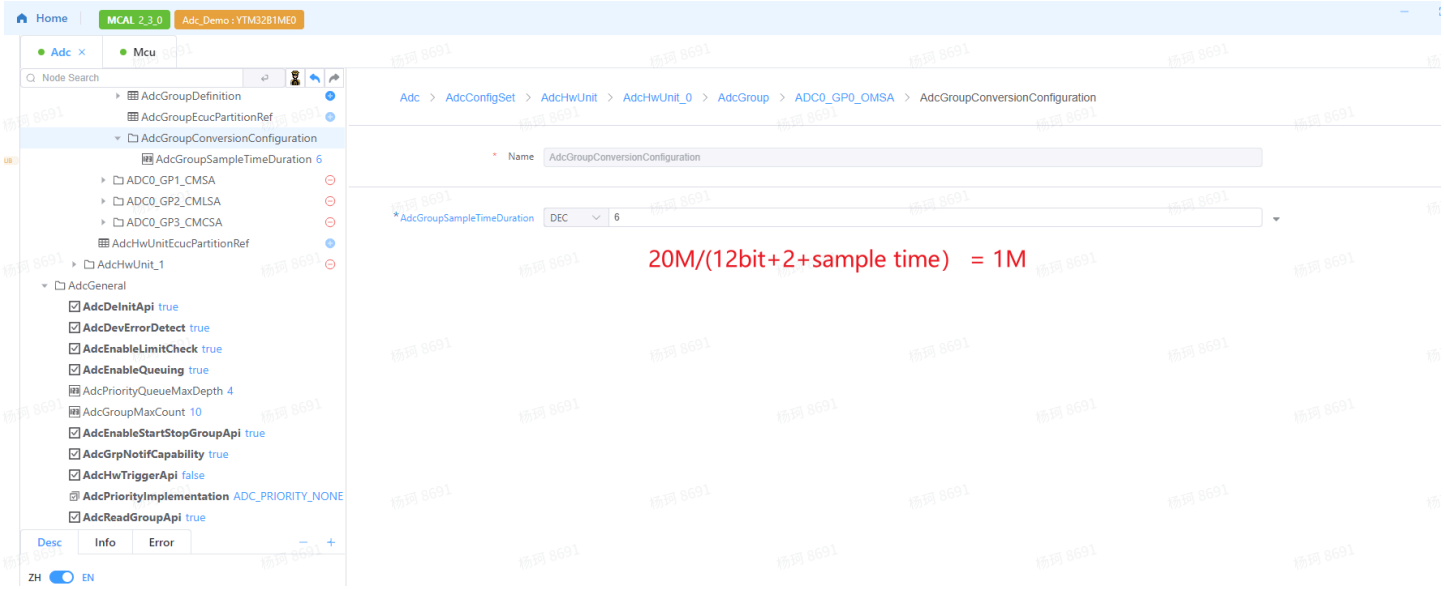
如果挂了多个Channel，则这些Channel必须属于同一硬件单元。不能跨多个物理单元。

Tips:

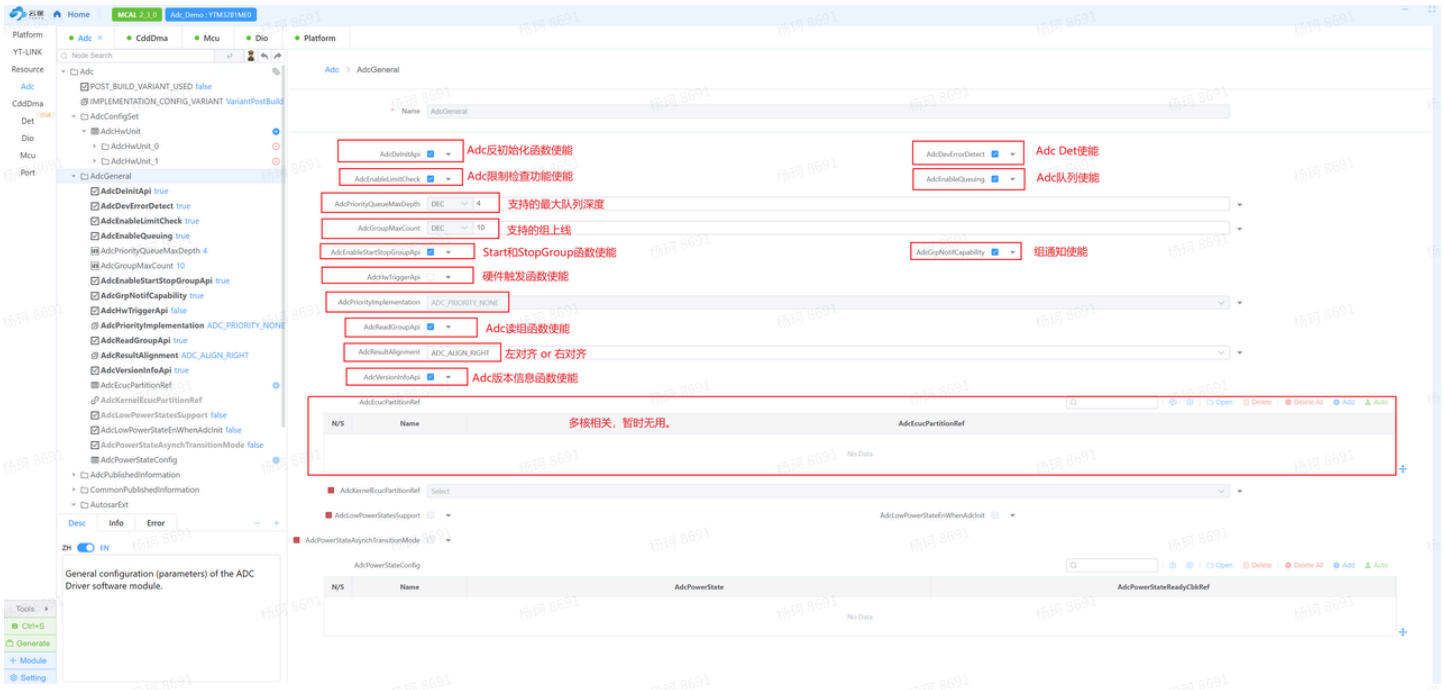
一个Channel可以同时属于多个Group，但这种情况需要用户自己合理管控转换顺序。否则可能会出现冲突。

1.2.2 Adc_Group

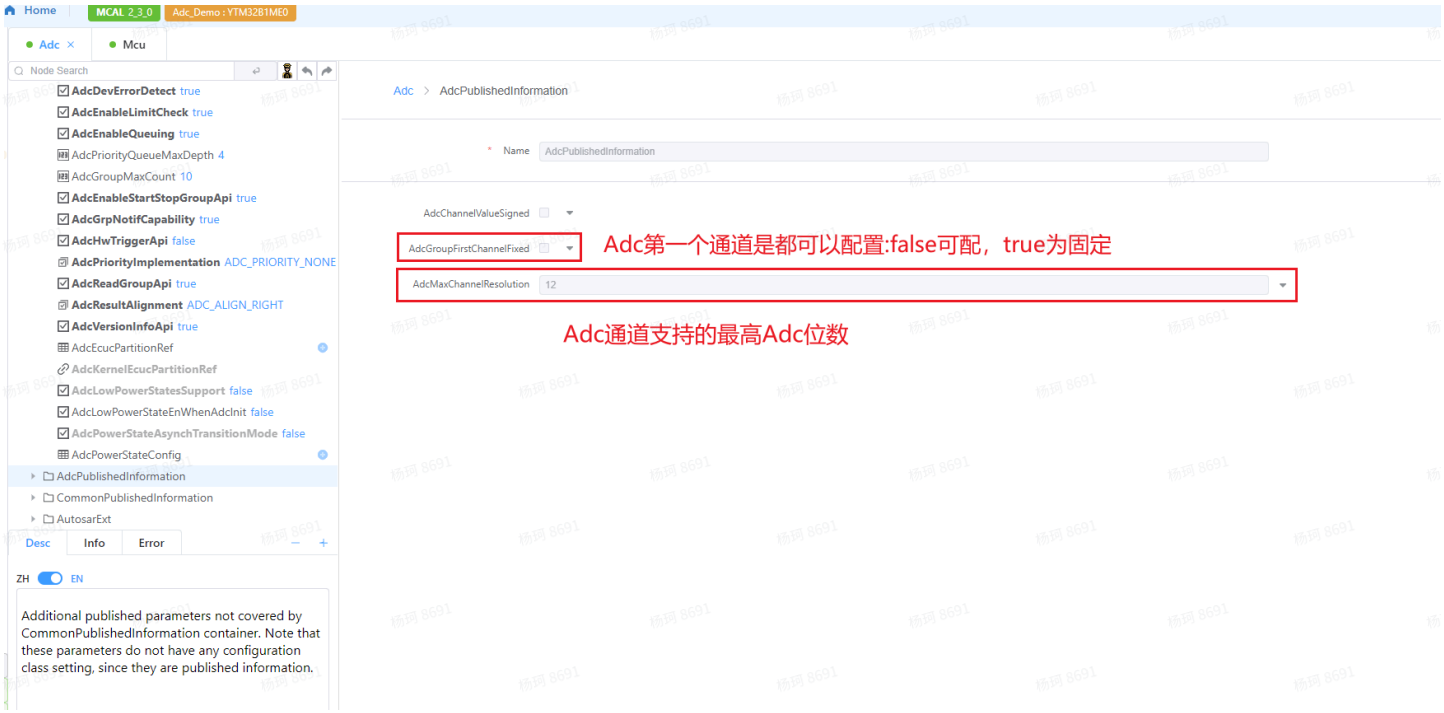




1.3 AdcGeneral

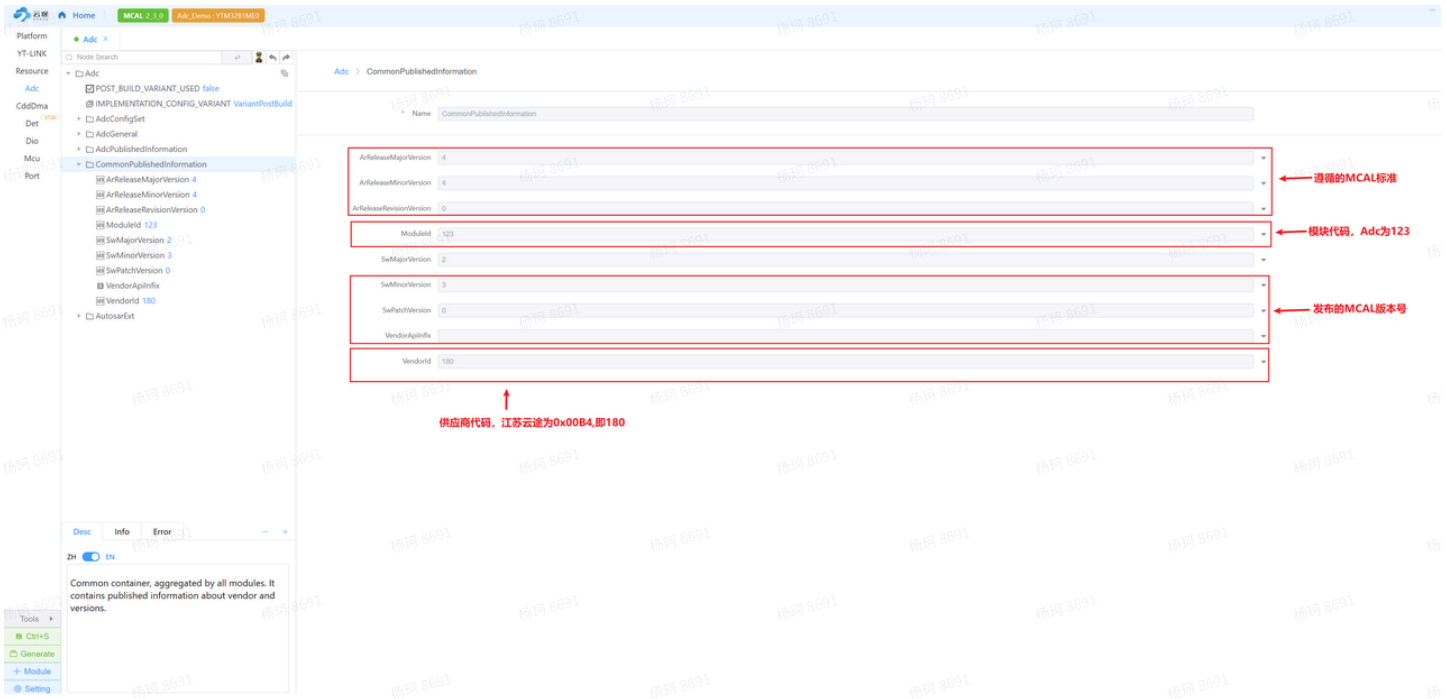


1. 4 AdcPublishInformation

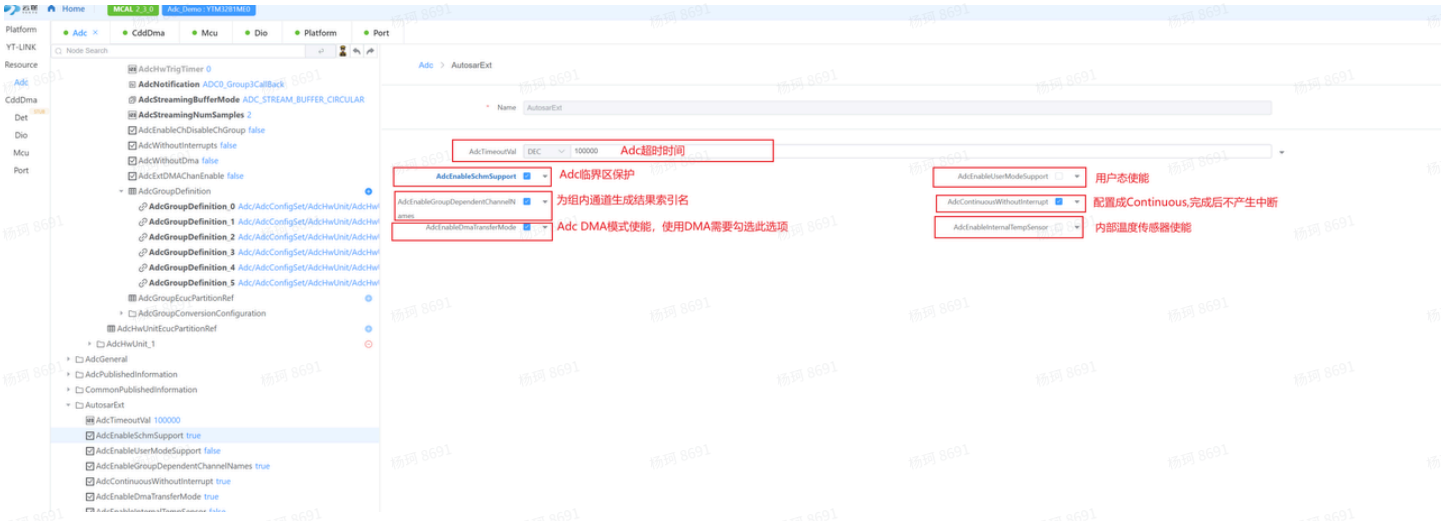


1.5 CommonPublishedInformation

此模块主要包含符合的AutoSar标准信息、模块ID、软件版本、以及供应商代码。



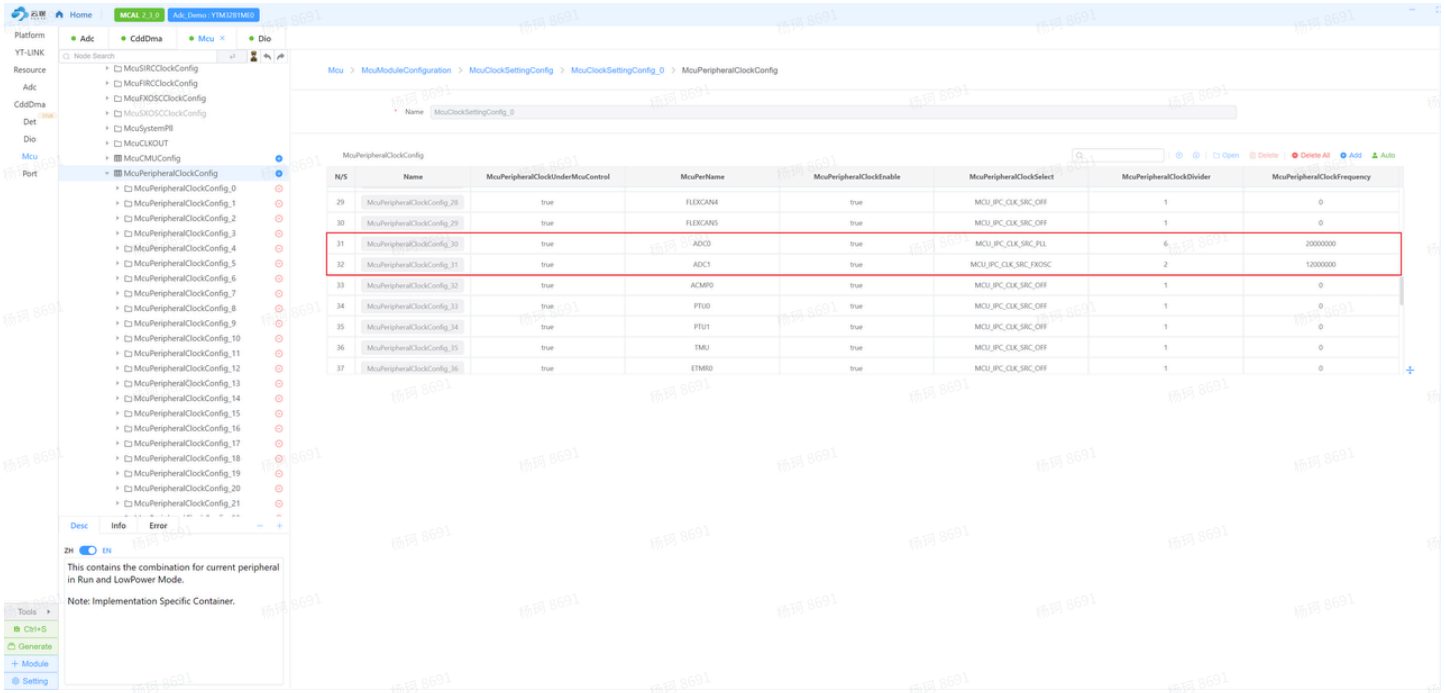
1.6 AutoSarExt



2、YT Config Tool配置示例

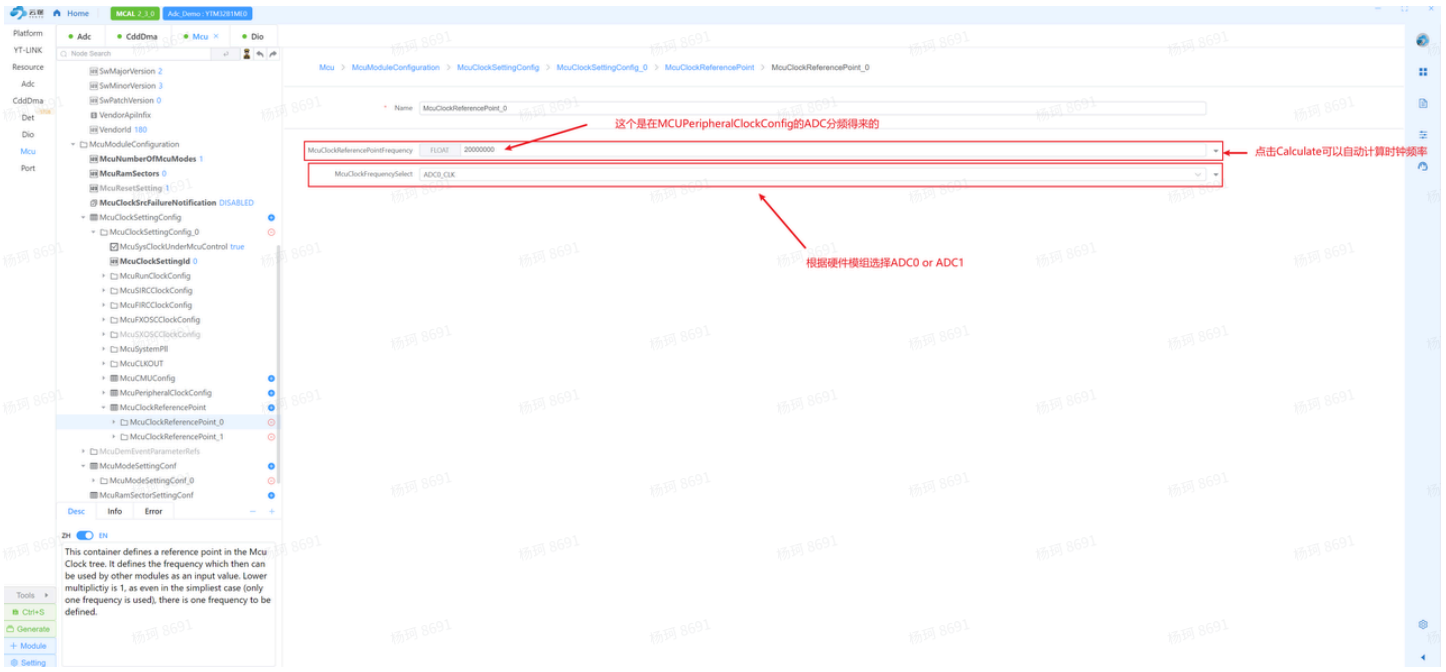
2.1时钟配置

2.1.1 在MCU模块中打开ADC时钟



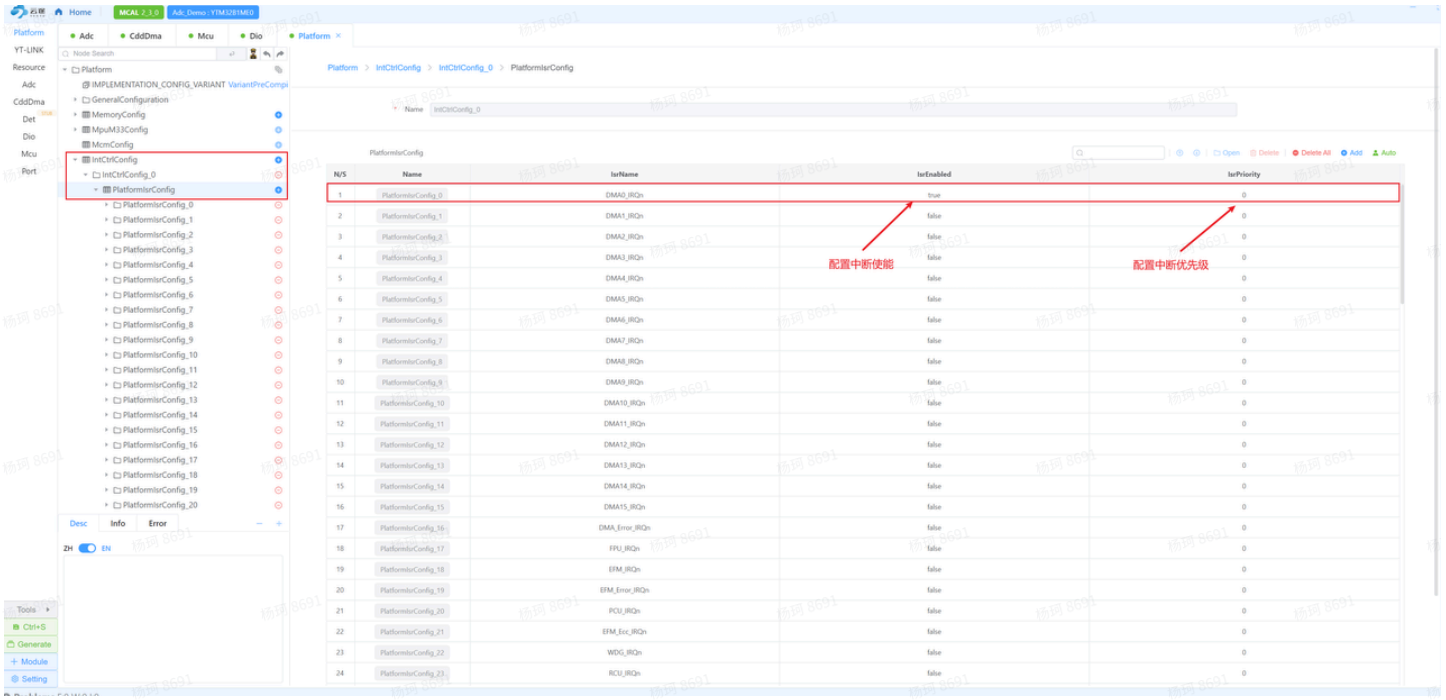
- adc0采用PLL作为时钟源，6分频后为20M
- adc1采用FXOSC作为时钟源，2分频后为12M

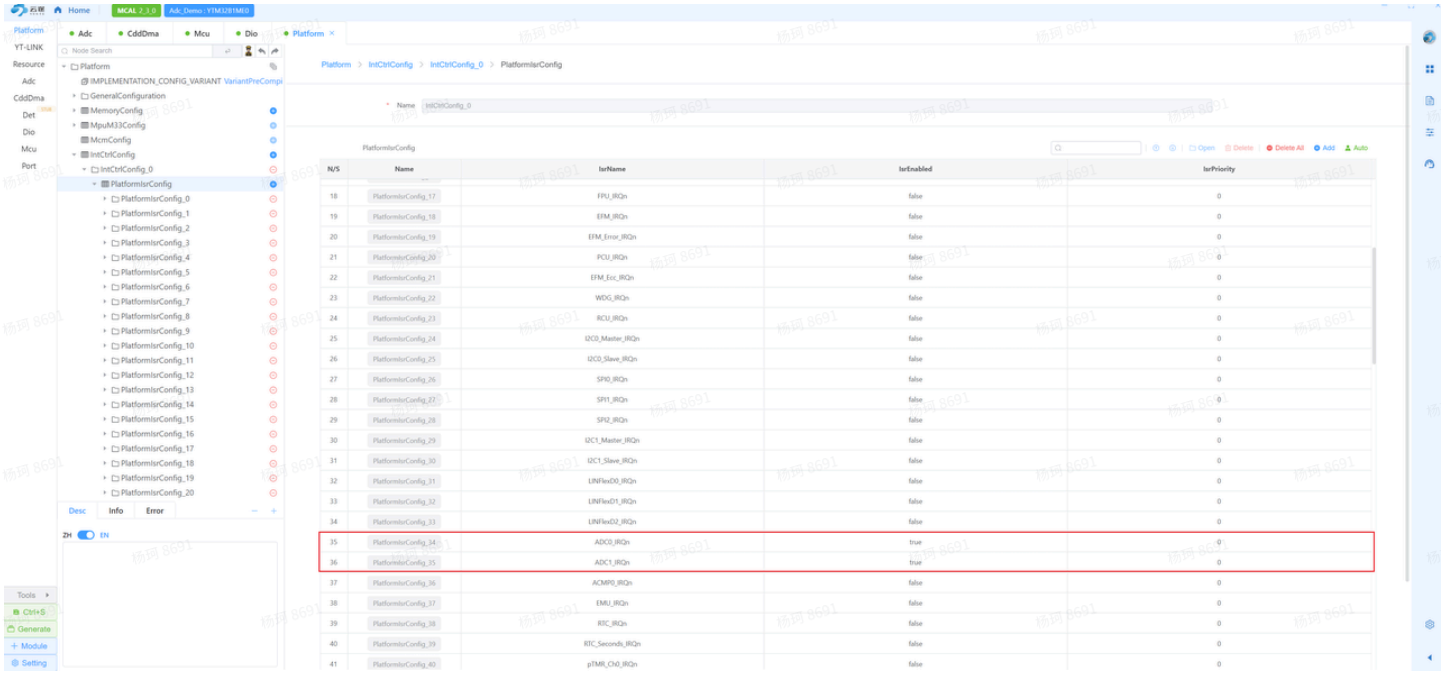
2.1.2在MCU模块中配置reference时钟



2.2中断配置

2.2.1 根据需求选择是否使能中断，如若需要中断，则在Platform中打开中断，若使用了OS，则需要在OS中配置Adc和DMA的外设中断。

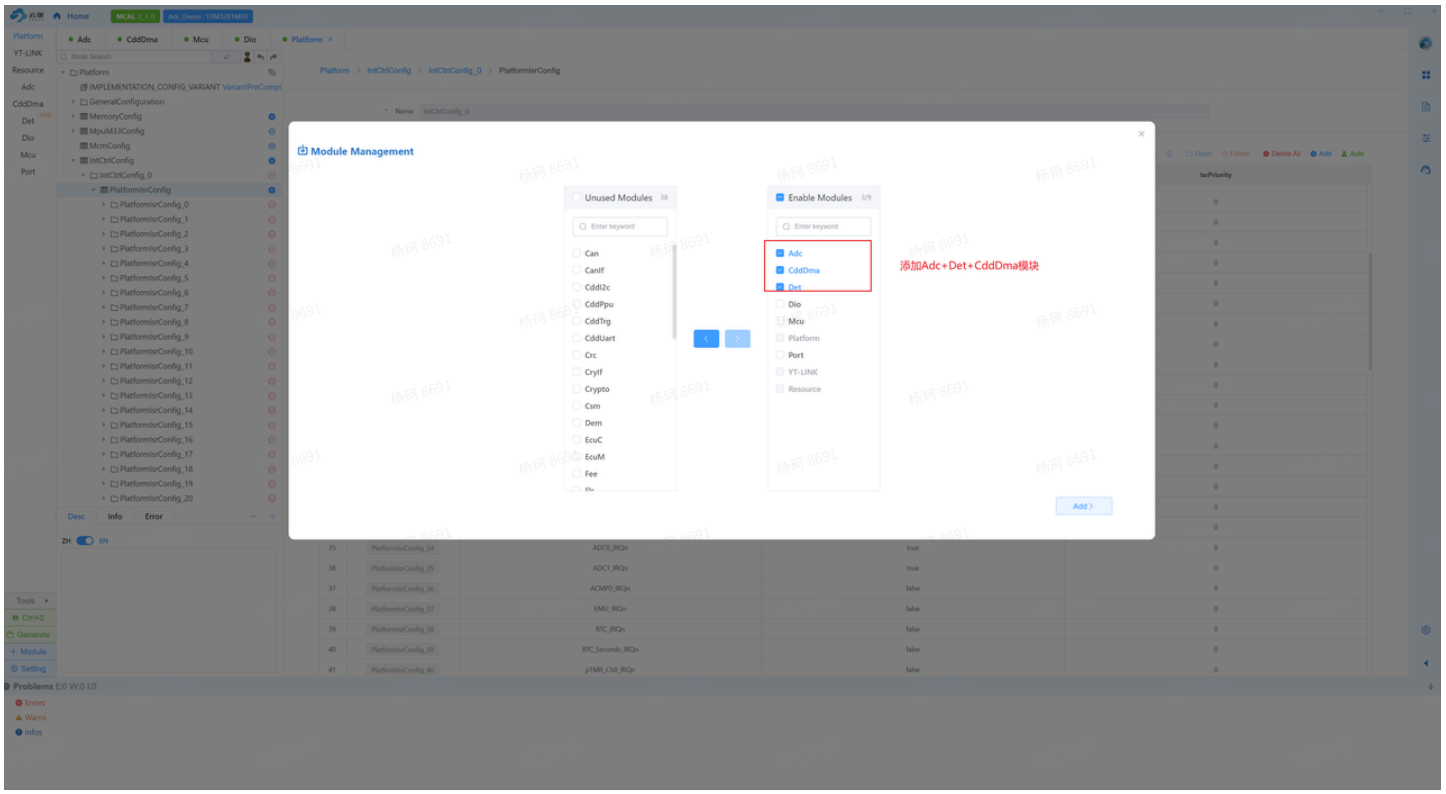




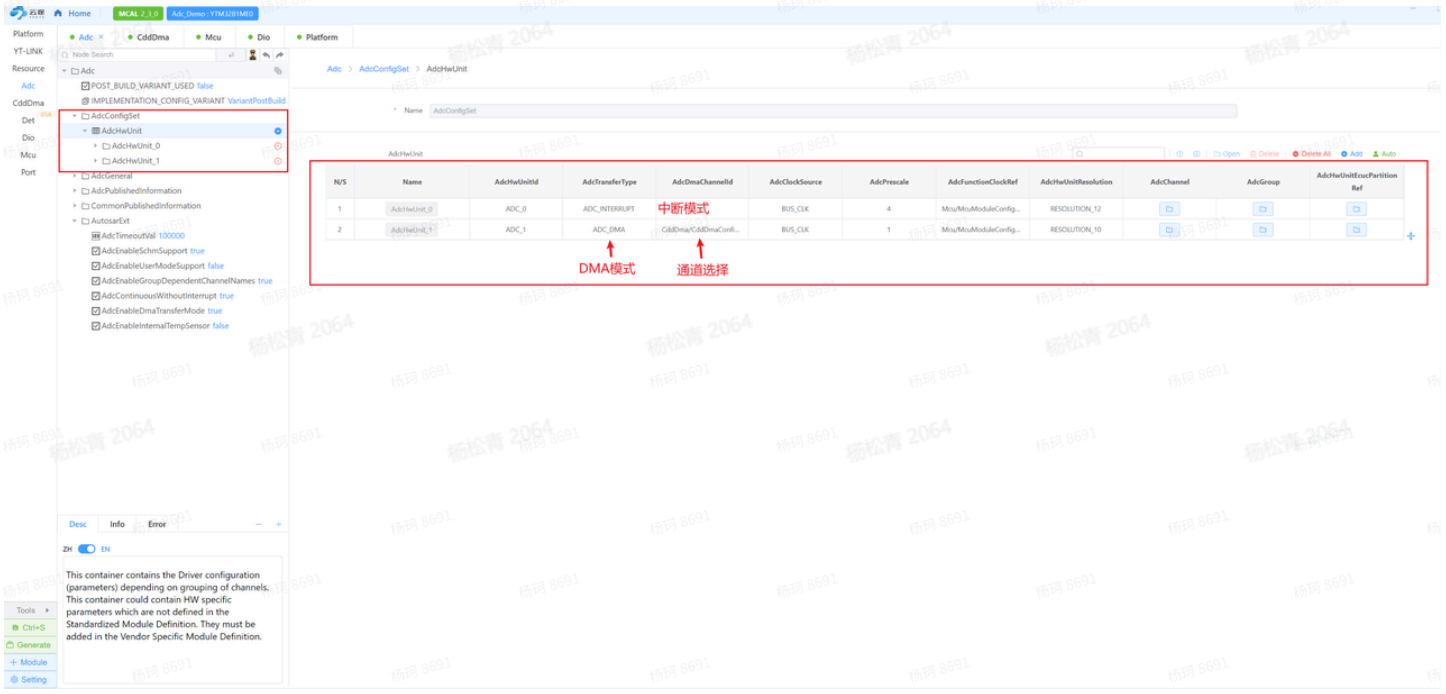
配置ADC0、ADC1、DMA0的外设中断

2.3配置Adc模块

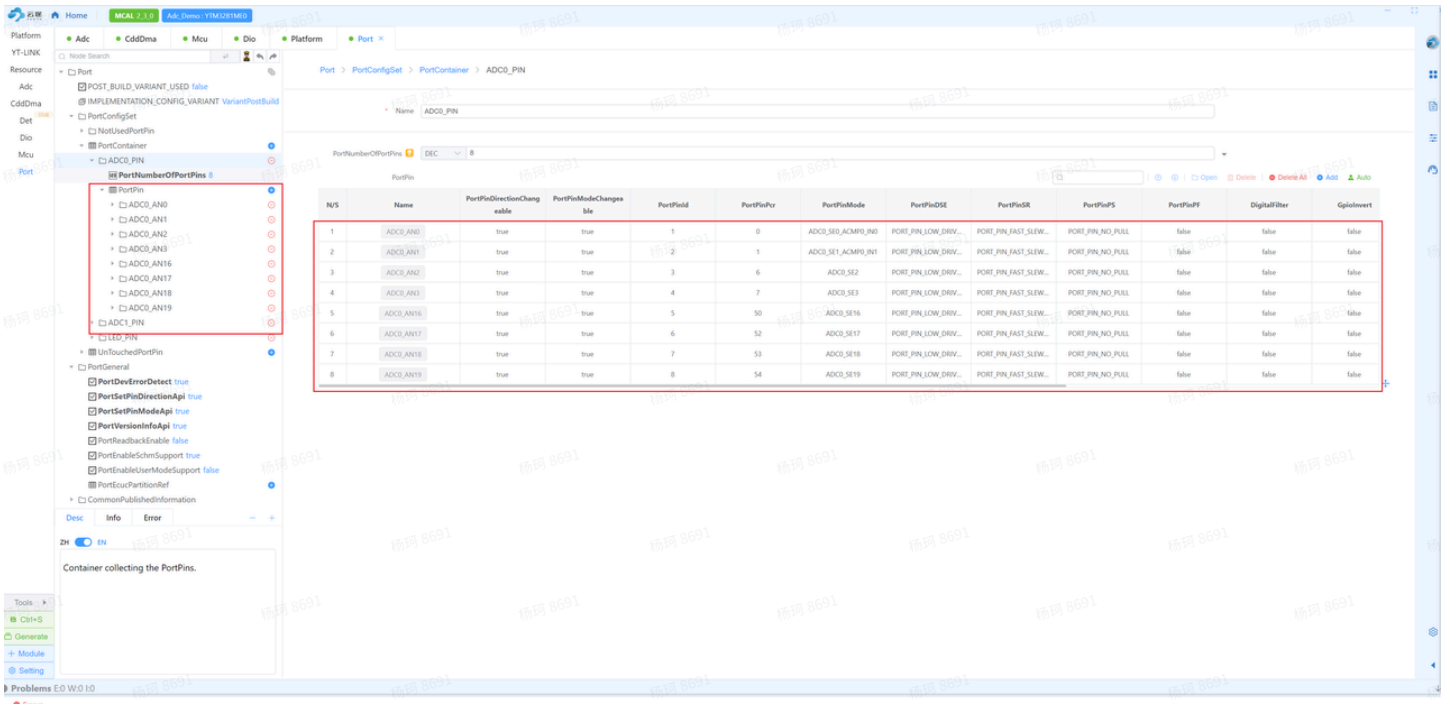
2.3.1添加Adc、Det、CddDma模块



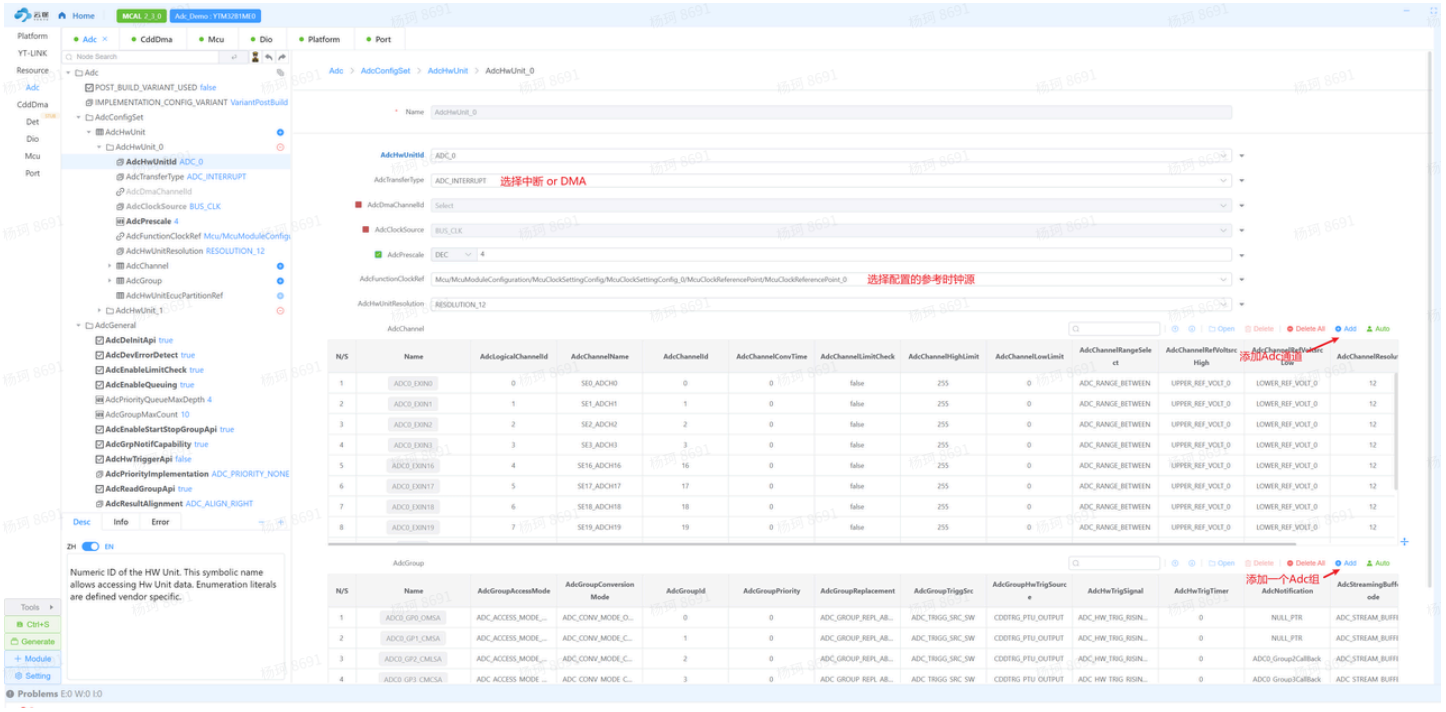
2.3.2添加两个AdcHwUnit



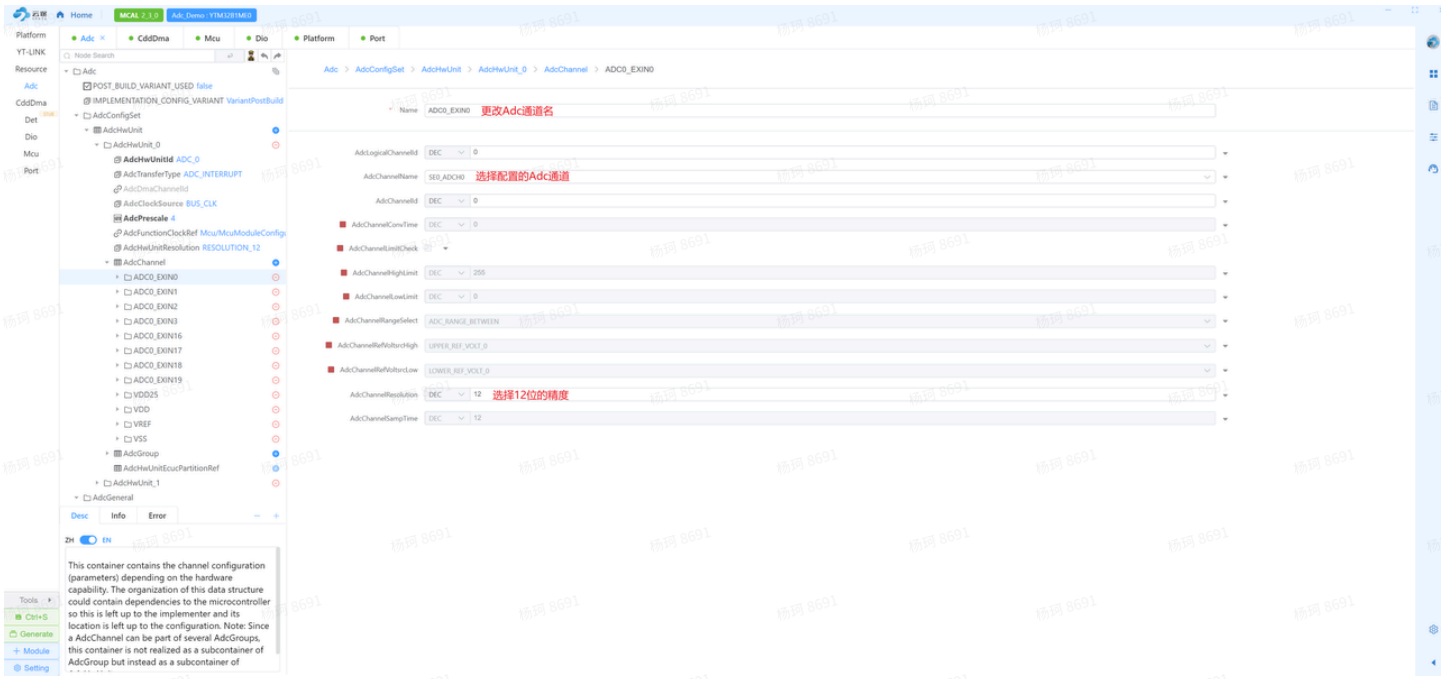
2.3.3在Port中配置好Adc0、Adc1通道



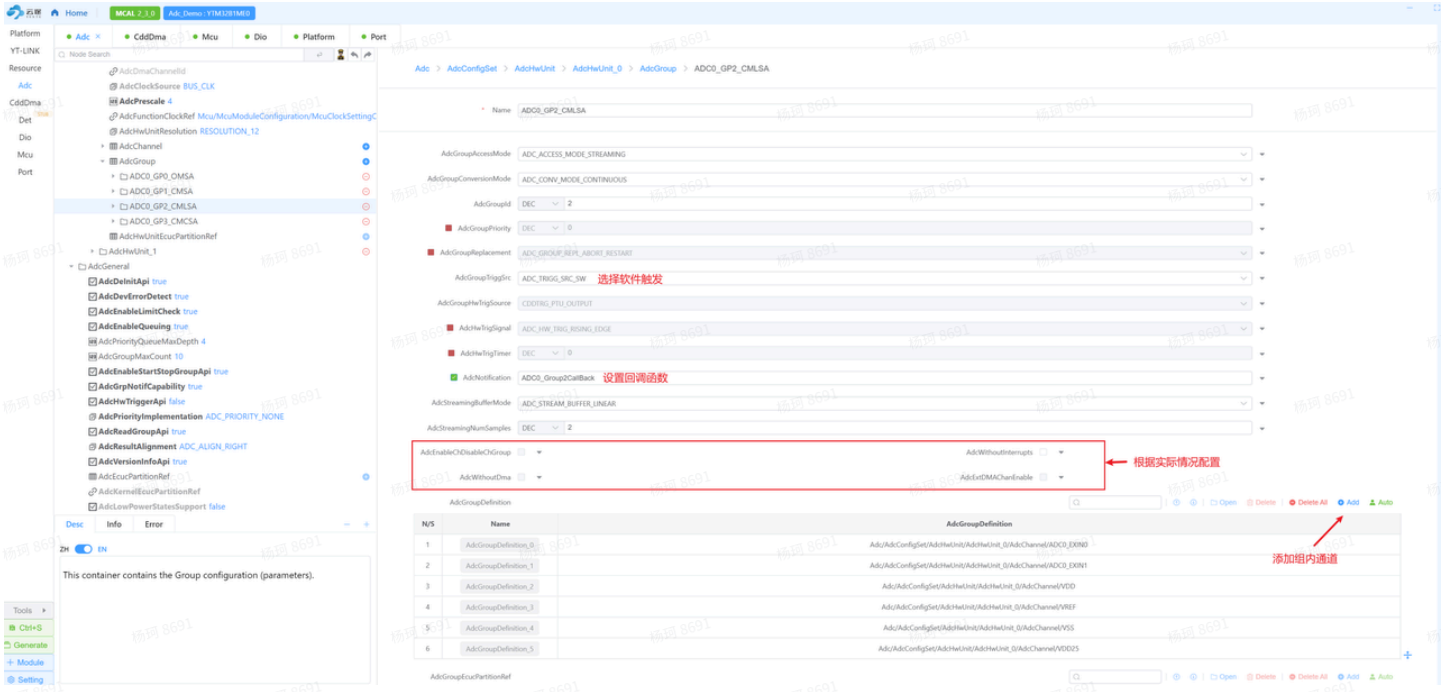
2.3.4添加通道和组（此处以ADC0中断模式为例）



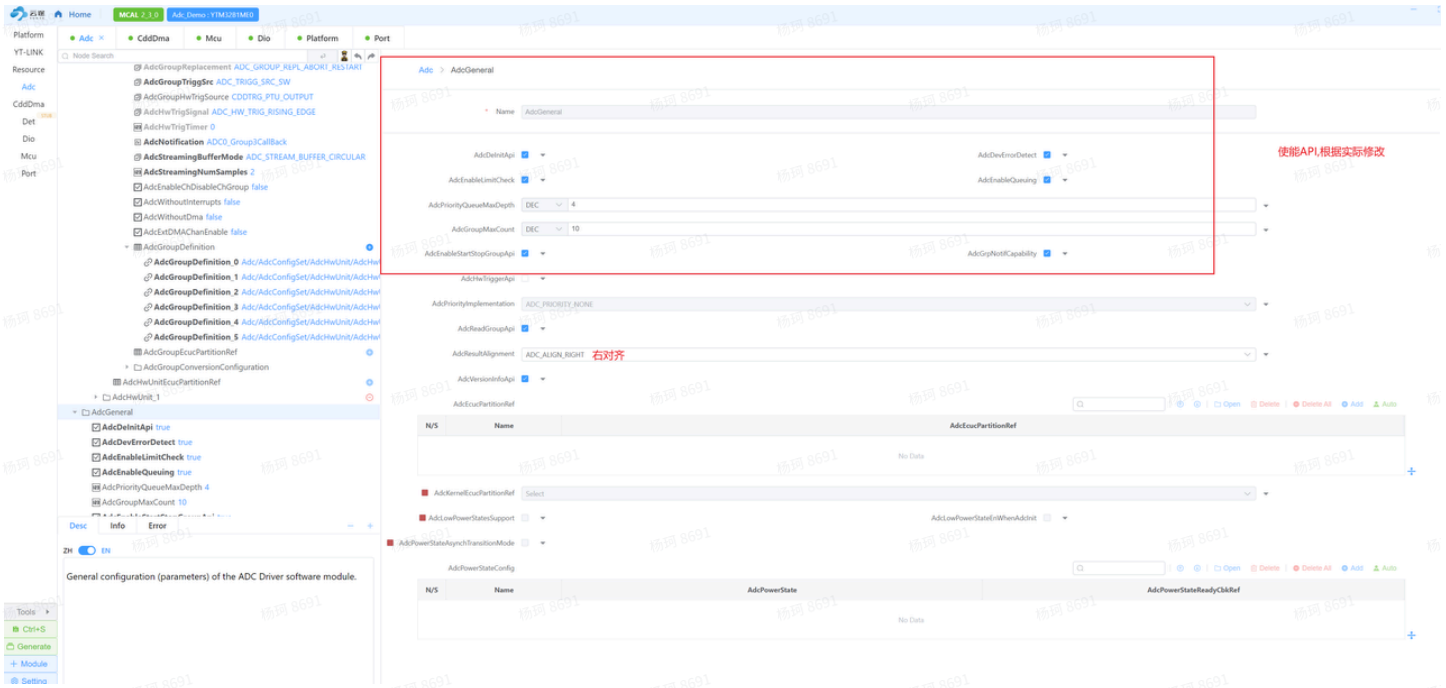
2.3.5根据实际需求更改参数



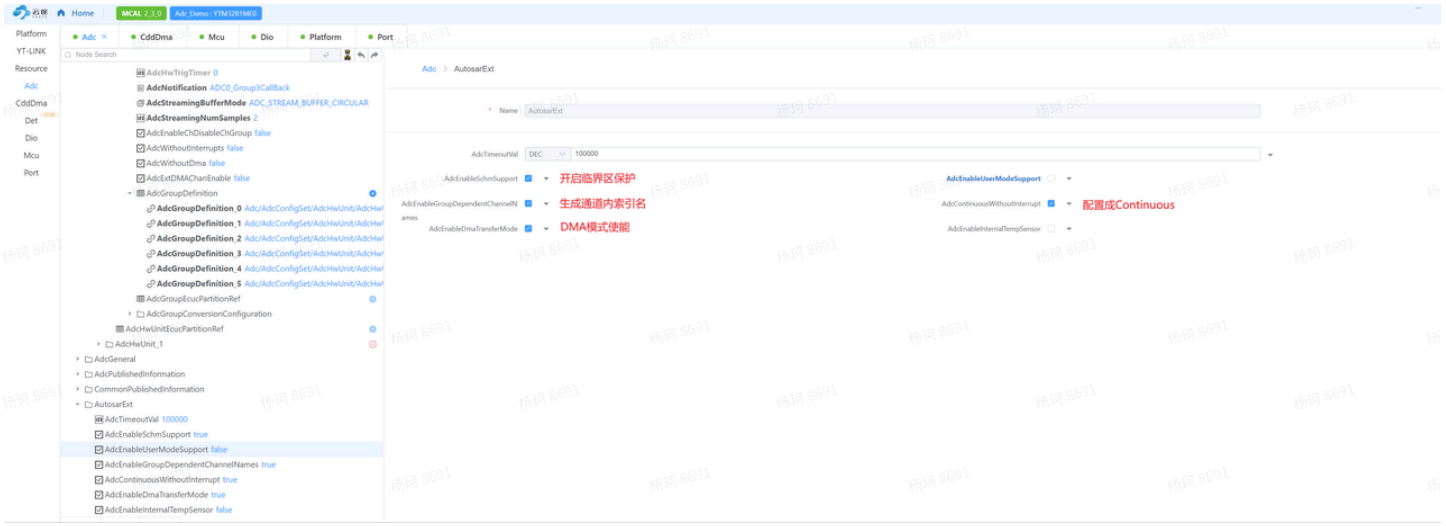
2.3.6根据实际需求更改组的参数。



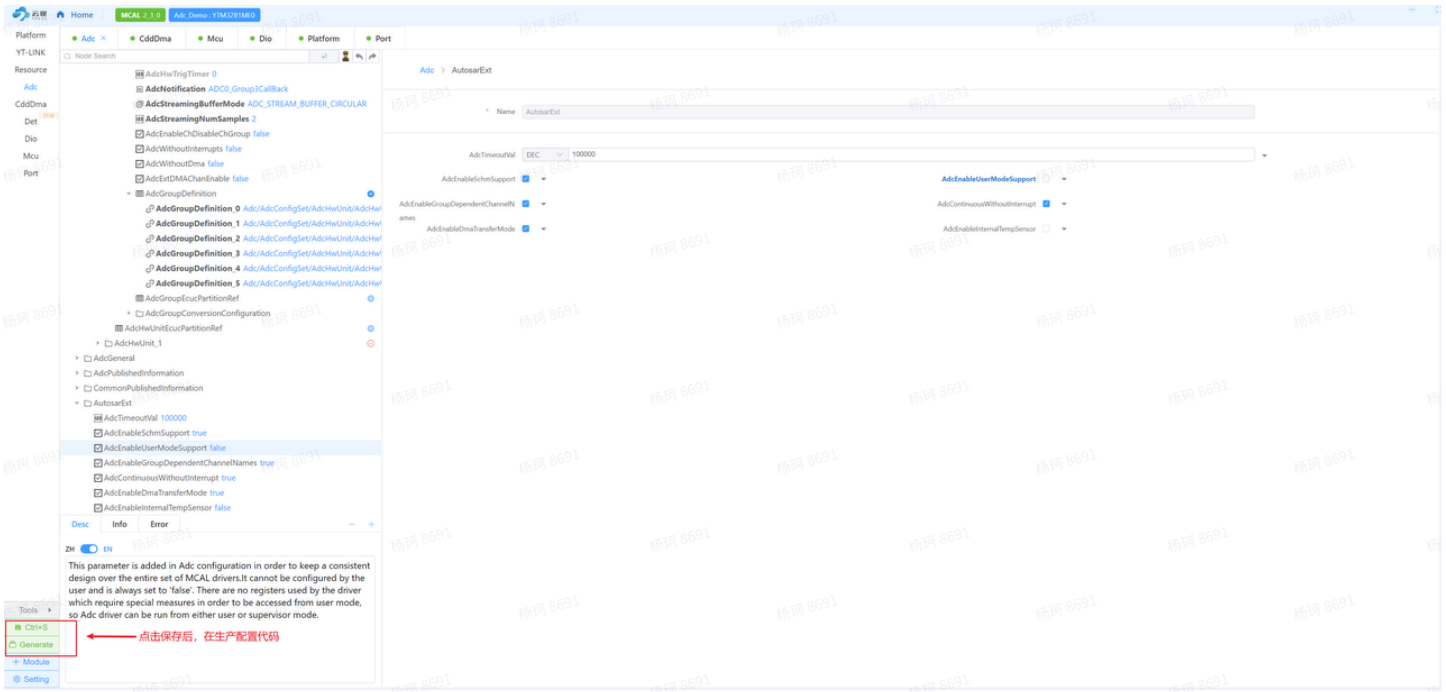
2.3.7配置Adc general。



2.3.8配置非标准的参数。

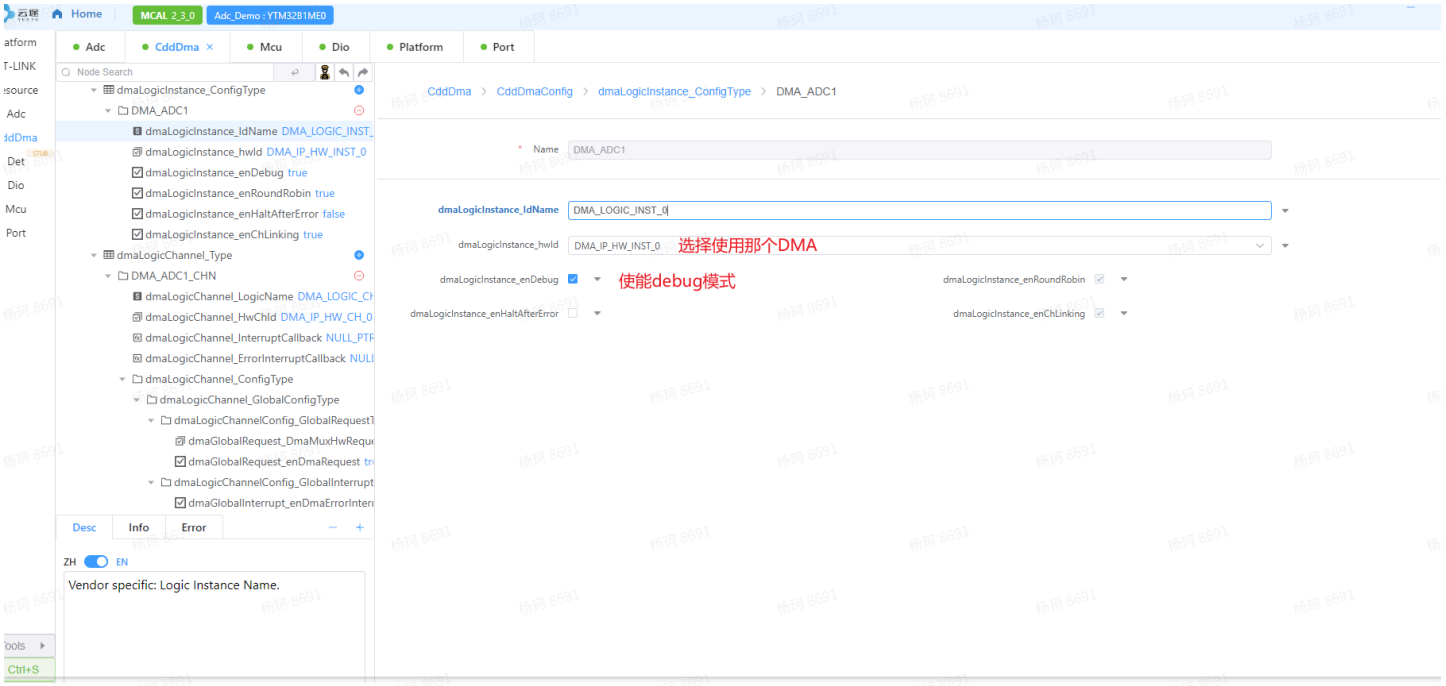


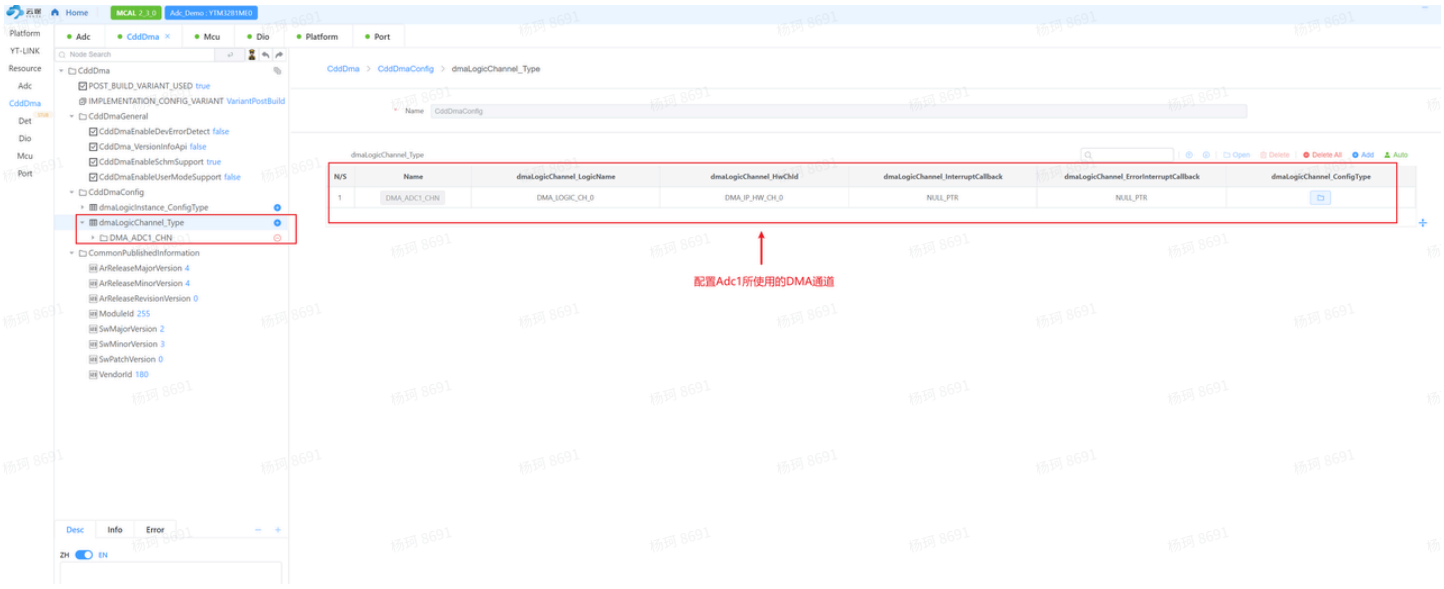
2.3.9生成配置工程，Adc配置完成。



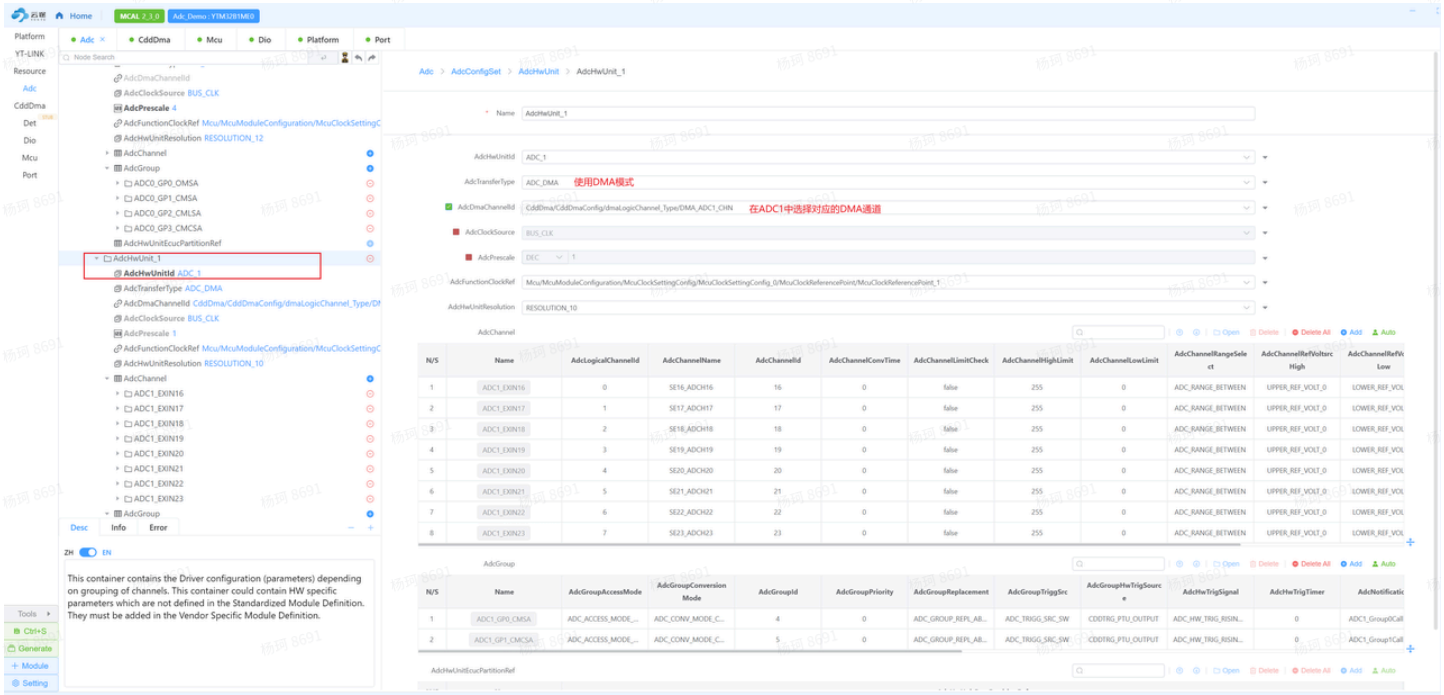
2.4DMA配置

2.4.1 配置DMA通道，DMA的详细配置介绍见DMA模块的介绍文档。





2.4.2 Adc模块中设置Adc1的DMA通道



2.5生成代码示例

3.1 ADC采样结果换算

ADC完成采样后，得到的是对应输入电压的数字量结果，通常需要将 ADC 原始采样值换算为实际电压值。

ADC 参考电压为 5V，分辨率为 12位，因此采样结果的换算公式如下：

```
Vin(mV) = ADC_Code × 5000 / 4096
```

其中：

- ADC_Code：ADC采样得到的原始数字量
- 5000：参考电压，单位为 mV
- 4096：12位 ADC 的分辨率，对应 2^12

根据上述公式，可将 ADC 原始结果换算为实际输入电压值，单位为毫伏（mV）。

3.2定义参数

```
/* USER CODE BEGIN PV */
/*The software result FIFO for all Adc group*/
Adc_ValueGroupType Adc0_Group0RstFIFO[ADC0_GP0_OMSA_CHANNEL_NUMBER];
Adc_ValueGroupType Adc0_Group1RstFIFO[ADC0_GP1_CMSA_CHANNEL_NUMBER];
Adc_ValueGroupType Adc0_Group2RstFIFO[ADC0_GP2_CMLSA_CHANNEL_NUMBER * ADC0_GP2_CMLSA_STEAMING_NUMBER];
Adc_ValueGroupType Adc0_Group3RstFIFO[ADC0_GP3_CMCSA_CHANNEL_NUMBER * ADC0_GP3_CMCSA_STEAMING_NUMBER];
Adc_ValueGroupType Adc1_Group0RstFIFO[ADC1_GP0_CMSA_CHANNEL_NUMBER + 1];
Adc_ValueGroupType Adc1_Group1RstFIFO[(ADC1_GP1_CMCSA_CHANNEL_NUMBER + 1)*ADC1_GP1_CMCSA_STEAMING_NUMBER];
Adc_ValueGroupType *Adc_TestResultPtr = NULL_PTR;
Adc_ValueGroupType Adc0_GroupAppRst[8];
uint16 Adc0_Group0ResultPhys[ADC0_GP0_OMSA_CHANNEL_NUMBER];
uint16 Adc0_Group1ResultPhys[ADC0_GP1_CMSA_CHANNEL_NUMBER];
uint16 Adc0_Group2ResultPhys[ADC0_GP2_CMLSA_CHANNEL_NUMBER];
uint16 Adc0_Group3ResultPhys[ADC0_GP3_CMCSA_CHANNEL_NUMBER];
uint16 Adc1_Group0ResultPhys[ADC1_GP0_CMSA_CHANNEL_NUMBER];
uint16 Adc1_Group1ResultPhys[ADC1_GP1_CMCSA_CHANNEL_NUMBER];


volatile uint8 Adc0_Group0ConCompCnt = 0;
volatile uint8 Adc0_Group1ConCompCnt = 0;
volatile uint8 Adc0_Group2ConCompCnt = 0;
volatile uint8 Adc0_Group3ConCompCnt = 0;
volatile uint8 Adc1_Group0ConCompCnt = 0;
volatile uint8 Adc1_Group1ConCompCnt = 0;

uint8 Adc0_ResultNotMatchCnt = 0;
```

3.3定义回调函数

```
/*Call Back functions which are needed for integration test*/
void ADC1_Group0CallBack(void)
{
    Adc1_Group0ConCompCnt++;
}

void ADC1_Group1CallBack(void)
{
    Adc1_Group1ConCompCnt++;
}

void ADC0_Group2CallBack(void)
{
    Adc0_Group2ConCompCnt++;
    if (ADC_STREAM_COMPLETED == Adc_GetGroupStatus(Group: AdcConf_AdcConfigSet_ADC0_GP2_CMLSA))
    {
        Adc_ReadGroup(Group: AdcConf_AdcConfigSet_ADC0_GP2_CMLSA, DataBufferPtr: Adc0_Group2ResultPhys);
        uint8 i = 0;
        for (i = 0; i < ADC0_GP2_CMLSA_CHANNEL_NUMBER; i++)
        {
            Adc0_Group2ResultPhys[i] = (uint32)Adc0_Group2ResultPhys[i] * 5000 / 4096;
        }
    }
    else
    {
    }
}

void ADC0_Group3CallBack(void)
{
    Adc0_Group3ConCompCnt++;
}
```

- 注意：回调函数的名称需和在YT Config Tool中设置的名称一致。

3.4各模块初始化

```
static void Board_Init(void)
{
    Port_Init(ConfigPtr: &Port_Config);
    Platform_Init(Config: NULL_PTR);
    CddDma_Init(ConfigPtr: &CddDma_Config);
    Adc_Init(ConfigPtr: &Adc_Config);
}
```

```
    Mcu_Init(ConfigPtr: &Mcu_Config);
    Mcu_InitClock(ClockSetting: 0);
#if (MCU_NO_PLL == STD_OFF)
    while (MCU_PLL_LOCKED != Mcu_GetPllStatus())
    {
        /* Wait until PLL is locked */
    }
    Mcu_DistributePllClock();
#endif
```

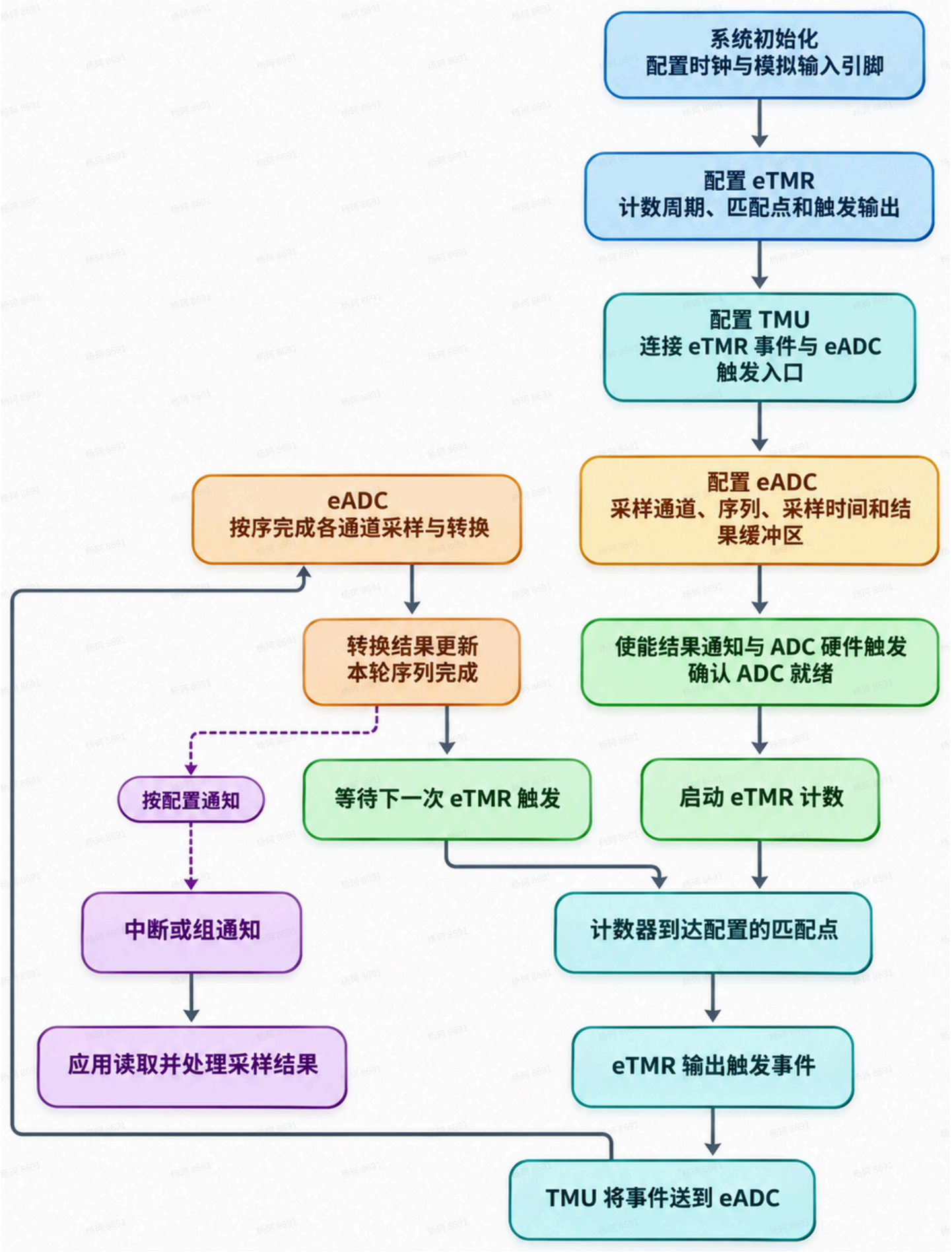
若配置有PLL,需等待PLL锁频完成

3.5 ADC 各通道组结果缓冲区配置、通知使能、转换启动。

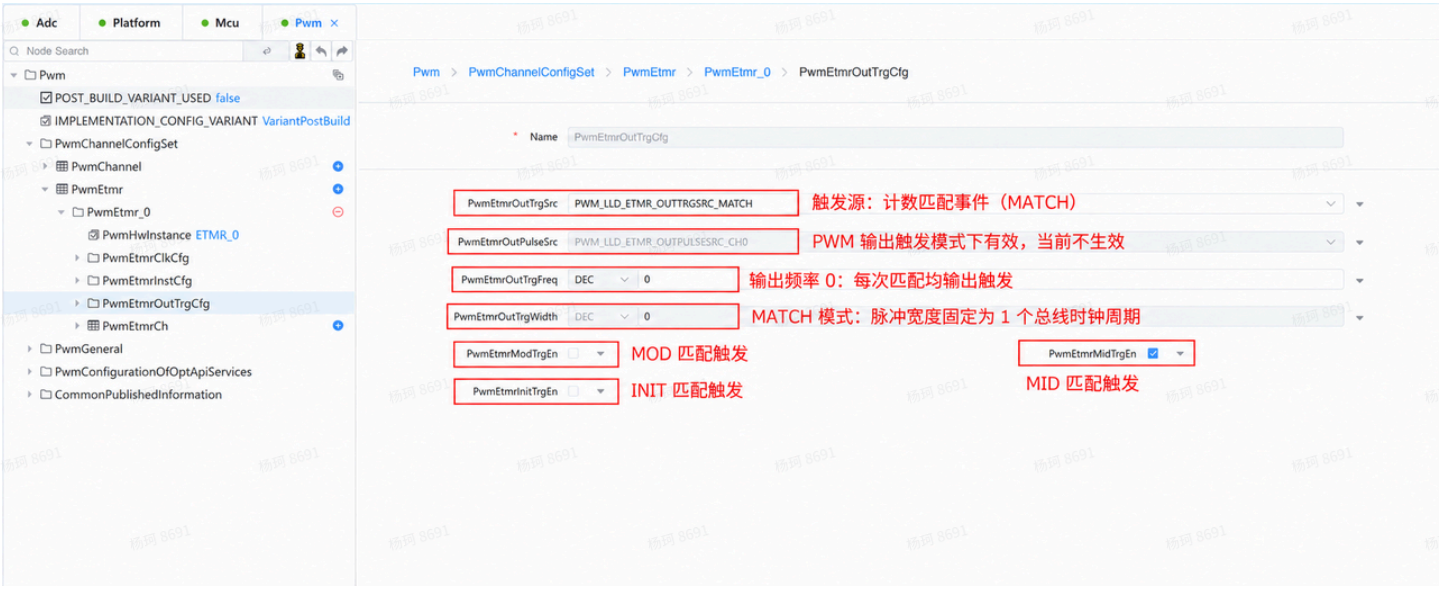
```
Adc_SetupResultBuffer(Group: AdcConf_AdcConfigSet_ADC0_GP0_OMSA, DataBufferPtr: Adc0_Group0RstFIFO);
Adc_SetupResultBuffer(Group: AdcConf_AdcConfigSet_ADC0_GP1_CMSA, DataBufferPtr: Adc0_Group1RstFIFO);
Adc_SetupResultBuffer(Group: AdcConf_AdcConfigSet_ADC0_GP2_CMLSA, DataBufferPtr: Adc0_Group2RstFIFO);
Adc_SetupResultBuffer(Group: AdcConf_AdcConfigSet_ADC0_GP3_CMCSA, DataBufferPtr: Adc0_Group3RstFIFO);
Adc_SetupResultBuffer(Group: AdcConf_AdcConfigSet_ADC1_GP0_CMSA, DataBufferPtr: Adc1_Group0RstFIFO);
Adc_SetupResultBuffer(Group: AdcConf_AdcConfigSet_ADC1_GP1_CMCSA, DataBufferPtr: Adc1_Group1RstFIFO);
Adc_EnableGroupNotification(Group: AdcConf_AdcConfigSet_ADC0_GP2_CMLSA);
Adc_EnableGroupNotification(Group: AdcConf_AdcConfigSet_ADC0_GP3_CMCSA);
Adc_EnableGroupNotification(Group: AdcConf_AdcConfigSet_ADC1_GP0_CMSA);
Adc_EnableGroupNotification(Group: AdcConf_AdcConfigSet_ADC1_GP1_CMCSA);
Adc0_Group0ConCompCnt = 0;
Adc0_Group1ConCompCnt = 0;
Adc0_Group2ConCompCnt = 0;
Adc0_Group3ConCompCnt = 0;
Adc1_Group0ConCompCnt = 0;
Adc1_Group1ConCompCnt = 0;
Adc_StartGroupConversion(Group: AdcConf_AdcConfigSet_ADC0_GP0_OMSA);
Adc_StartGroupConversion(Group: AdcConf_AdcConfigSet_ADC0_GP1_CMSA);
Adc_StartGroupConversion(Group: AdcConf_AdcConfigSet_ADC0_GP2_CMLSA);
Adc_StartGroupConversion(Group: AdcConf_AdcConfigSet_ADC0_GP3_CMCSA);
Adc_StartGroupConversion(Group: AdcConf_AdcConfigSet_ADC1_GP0_CMSA);
Adc_StartGroupConversion(Group: AdcConf_AdcConfigSet_ADC1_GP1_CMCSA);
```

3、硬件触发ADC采样

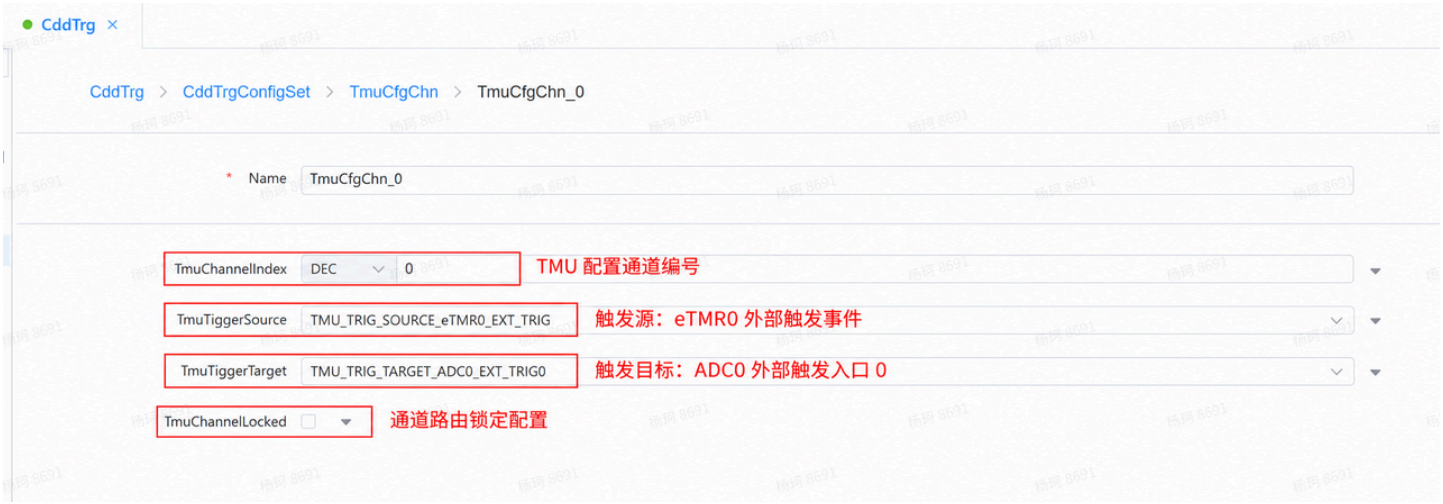
3.1eTMR触发ADC采样流程图

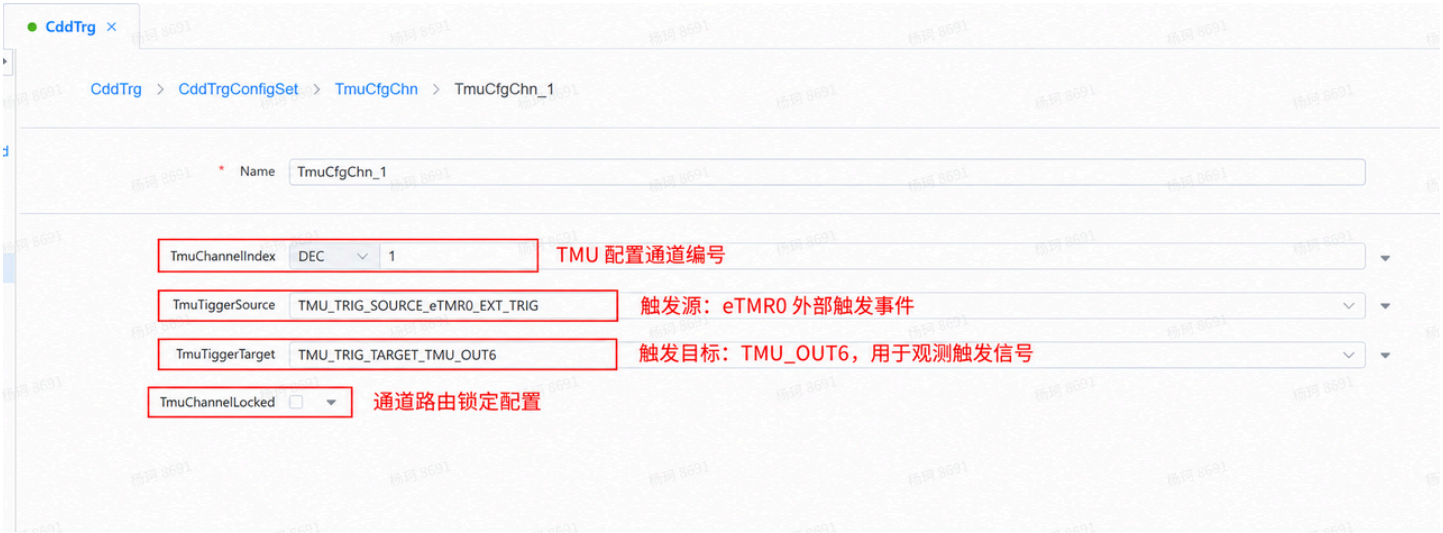

系统初始化
配置时钟与模拟输入引脚
配置 eTMR
计数周期、匹配点和触发输出
配置 TMU
连接 eTMR 事件与 eADC
触发入口
配置 eADC
采样通道、序列、采样时间和结果缓冲区
eADC
按序完成各通道采样与转换
转换结果更新
本轮序列完成
使能结果通知与 ADC 硬件触发
确认 ADC 就绪
按配置通知
等待下一次 eTMR 触发
启动 eTMR 计数
中断或组通知
计数器到达配置的匹配点
应用读取并处理采样结果
eTMR 输出触发事件
TMU 将事件送到 eADC

3.2eTMR触发配置

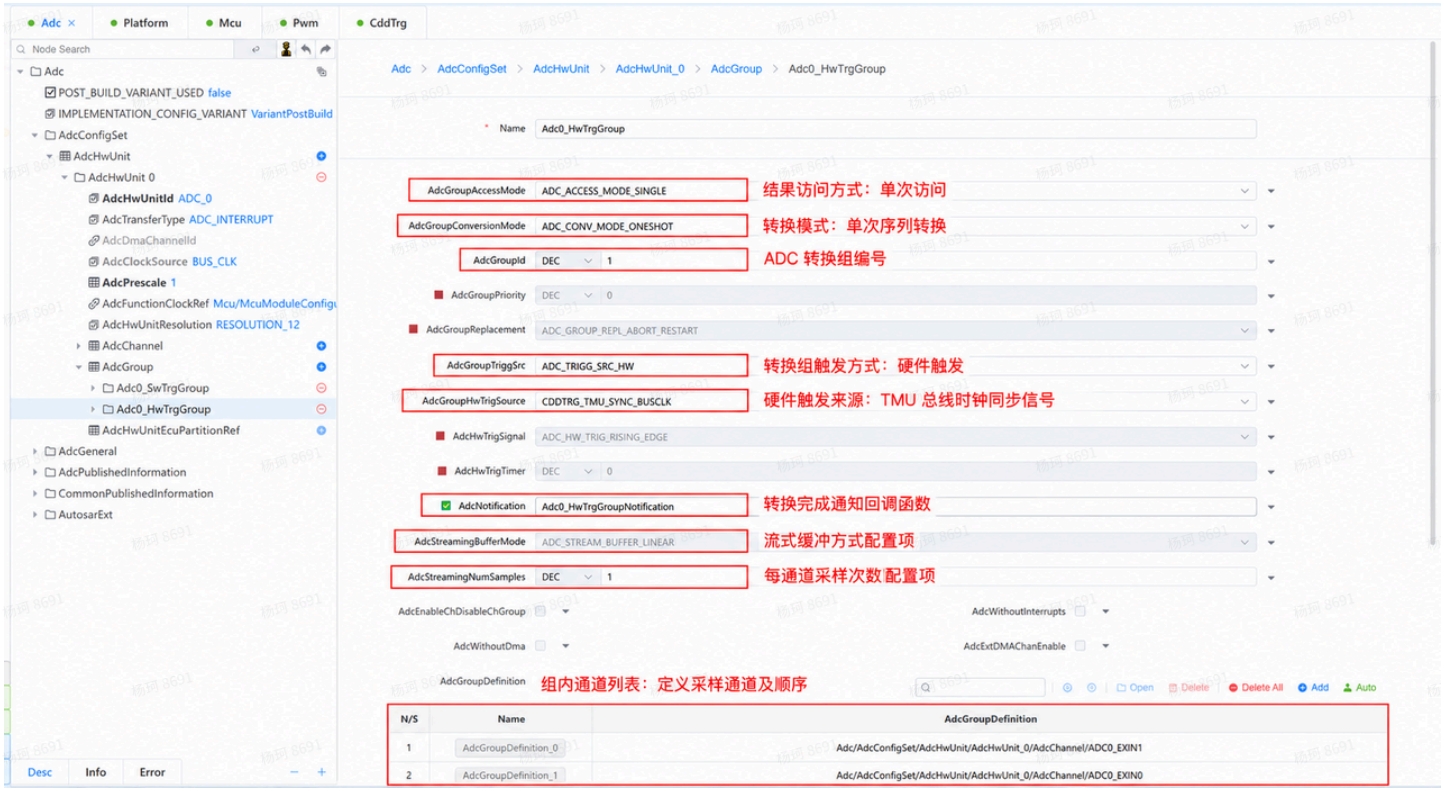


3.3TMU触发配置





3.4ADC触发配置



3.5中断配置



3.6生成代码示例

- 各个模块初始化调用

```
145    static void Board_Init(void)
146    {
147        Port_Init(&Port_Config);
148        Platform_Init(NULL_PTR);
149        CddTrg_Init(&CddTrg_Config);
150        Adc_Init(&Adc_Config);
151        Pwm_Init(&Pwm_Config);
152    }
```

- 启动ADC采样

```
    /* USER CODE BEGIN 2 */
    Adc_SetupResultBuffer(AdcConf_AdcConfigSet_Adc0_SwTrgGroup, Adc0_SwTrigGroupRstFIFO);
    Adc_SetupResultBuffer(AdcConf_AdcConfigSet_Adc0_HwTrgGroup, Adc0_HwTrigGroupRstFIFO);
    Adc_EnableGroupNotification(AdcConf_AdcConfigSet_Adc0_SwTrgGroup);
    Adc_EnableGroupNotification(AdcConf_AdcConfigSet_Adc0_HwTrgGroup);
    Adc0_SwTrigGroupConCompCnt = 0;
    Adc0_HwTrigGroupConCompCnt = 0;
    Adc_EnableHardwareTrigger(AdcConf_AdcConfigSet_Adc0_HwTrgGroup);
    Adc_StartGroupConversion(AdcConf_AdcConfigSet_Adc0_SwTrgGroup);
```

- ADC回调函数处理

```
/*Call Back functions which are needed for integration test*/
void Adc0_HwTrgGroupNotification(void)
{
    Adc0_HwTrigGroupConCompCnt++;
    if (Adc_GetGroupStatus(AdcConf_AdcConfigSet_Adc0_HwTrgGroup) == ADC_STREAM_COMPLETED)
    {
        Adc_ReadGroup(AdcConf_AdcConfigSet_Adc0_HwTrgGroup, Adc0_HwTrigGroupResultPhys);
        for (uint8 LoopCnt = 0; LoopCnt < Adc0_HwTrgGroup_CHANNEL_NUMBER; LoopCnt++)
        {
            Adc0_HwTrigGroupResultPhys[LoopCnt] = (uint32)Adc0_HwTrigGroupResultPhys[LoopCnt] * 5000 / 4095;
        }
        Adc_DisableHardwareTrigger(AdcConf_AdcConfigSet_Adc0_HwTrgGroup);
    }
}
```

- eTMR触发ADC信号与TMU_OUT信号



文档历史

版本号	日期	修订记录
V1.0	2023.11.20	1. 初始版本

V2.0	2026.05.10	1. 更新配置工具YCT版本为2.7.7及MCAL版本为2.3.0； 2. Fix一些新的功能，例如ADC_DMA模式； 3. 新增代码示例；
V3.0	2026.11.10	1. 增加eTMR触发ADC采样