MCAL 总结

注：

芯片用户手册：Infineon-TC27x_D-step-UM-v02_02-EN.pdf
文中用 TC27X_UM

芯片数据手册： Infineon-TC27x_DS_Addendum-DataSheet-v01_20-EN.pd
文中用 TC27X_DS

MCAL 用户手册： Aurix_MC-ISAR_UM_MCUDriver.pdf
文中用 MCAL_MCU_UM

一、 MCU 配置

1、 功能简介

MCU 驱动程序直接访问微控制器硬件，位于微控制器抽象层（MCAL）。

MCU Driver – Overview

The MCU driver provides services for

- 初始化MCU的时钟及外设时钟等;
- 初始化RAM区;
- MCU的功耗控制;
- MCU的复位控制;
- 获取复位原因.

2、 时钟树：

时钟树讲了各模块如何计算以及来源

For Normal Mode $f_{pll} = \frac{N}{P \times K2} f_{osc}$

For Normal Mode $f_{pll2} = \frac{N}{P \times K3} f_{osc}$

MCAL_MCU_UM--P23

Where N = Ndiv + 1,
P = Pdiv + 1,
K1 = K1div + 1,
K2 = K2div + 1,
K3 = K3div + 1,

计算时注意这里的K1、K2、P

例如PLL1 与 PLL2 计算

上例：PLL1 与 PLL2 来源

TC27X_UM--P427

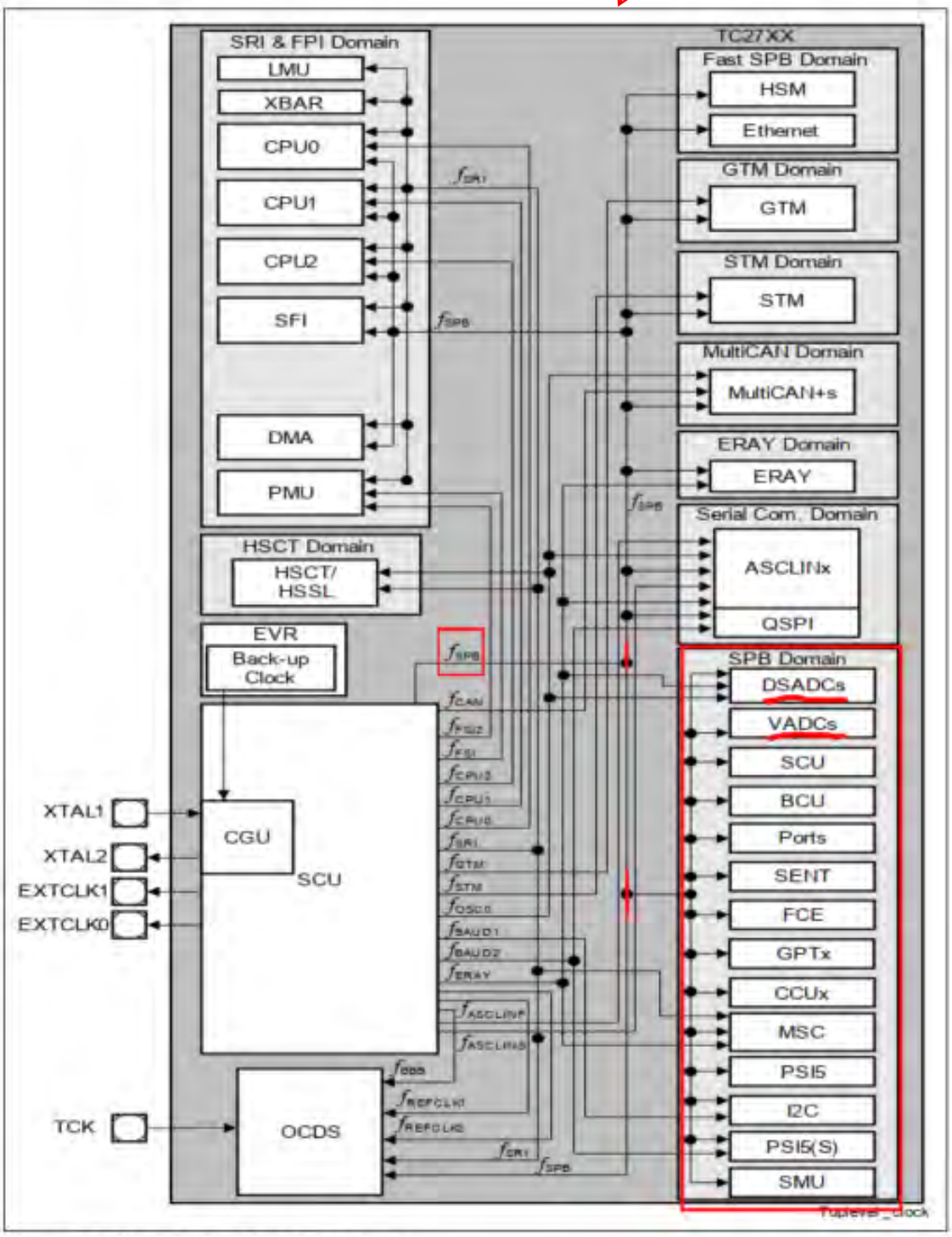
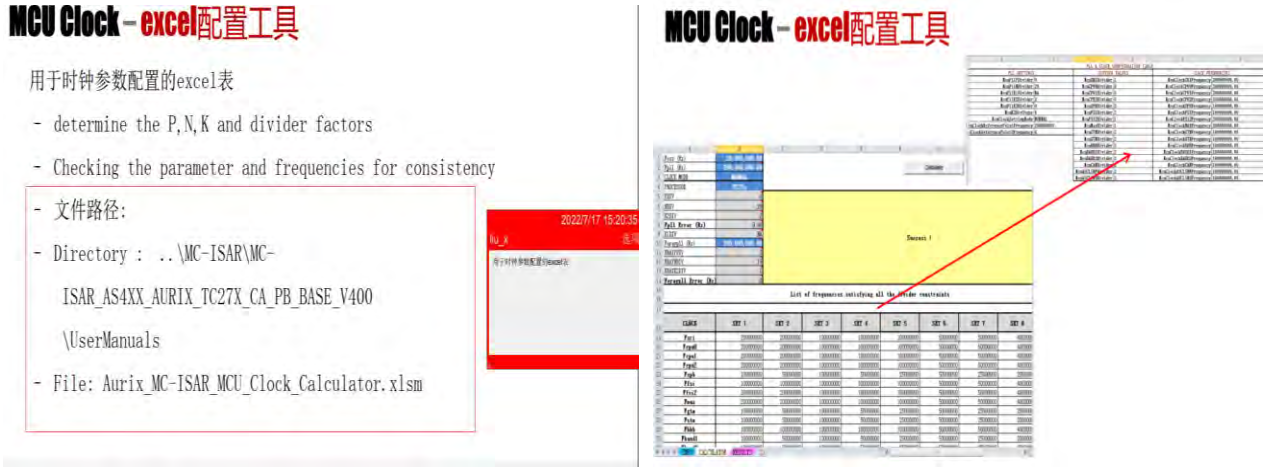
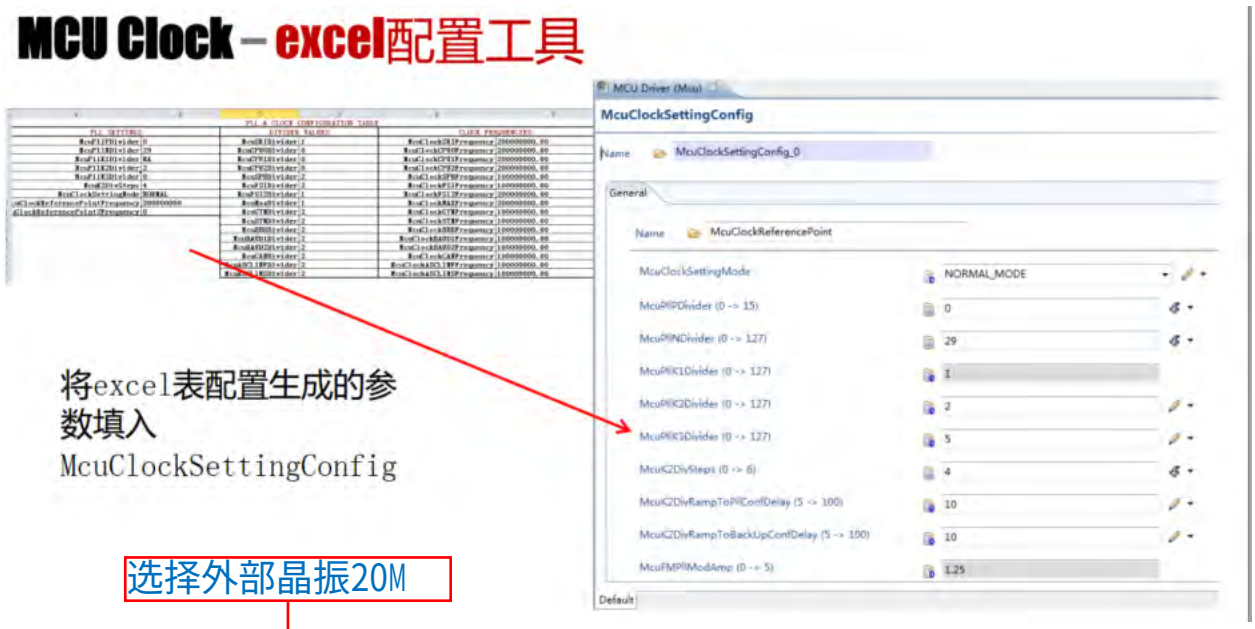


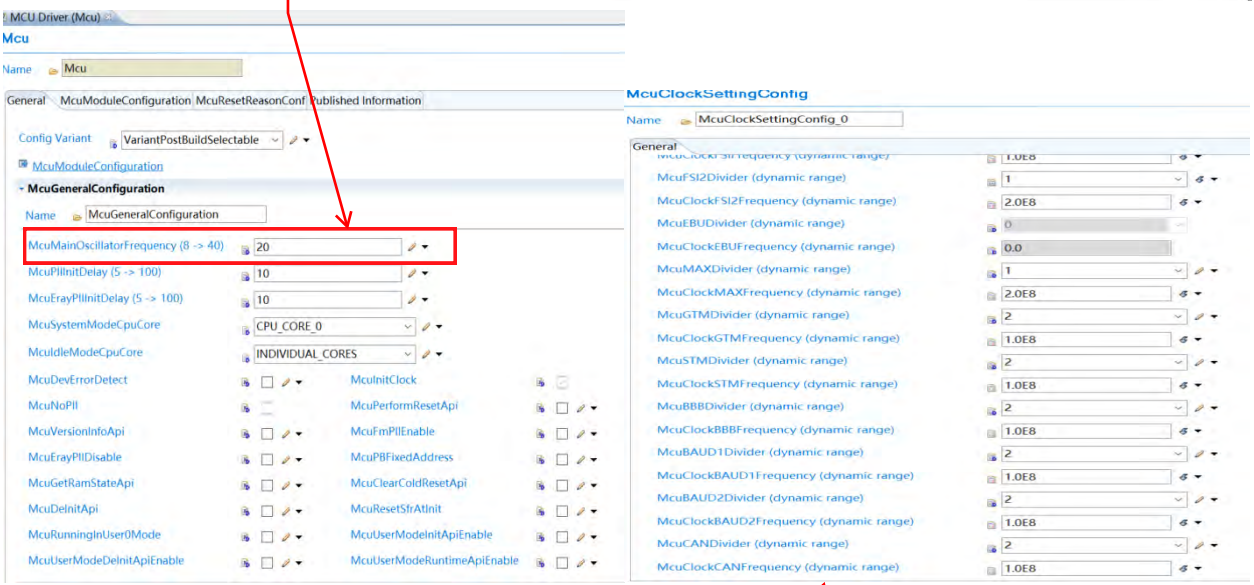
Figure 7-13 TC27x Clocking System

3、 Mcu 的 EB 配置

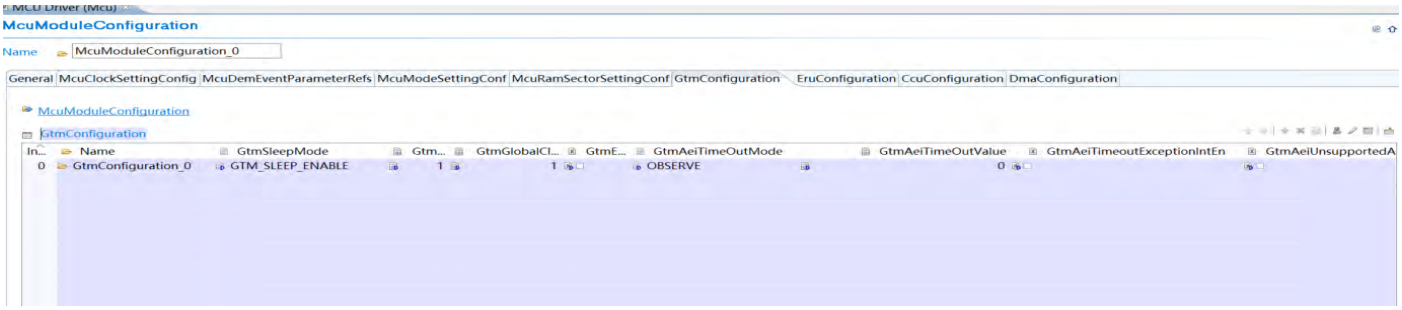




选择外部晶振20M



计算出来外设时钟



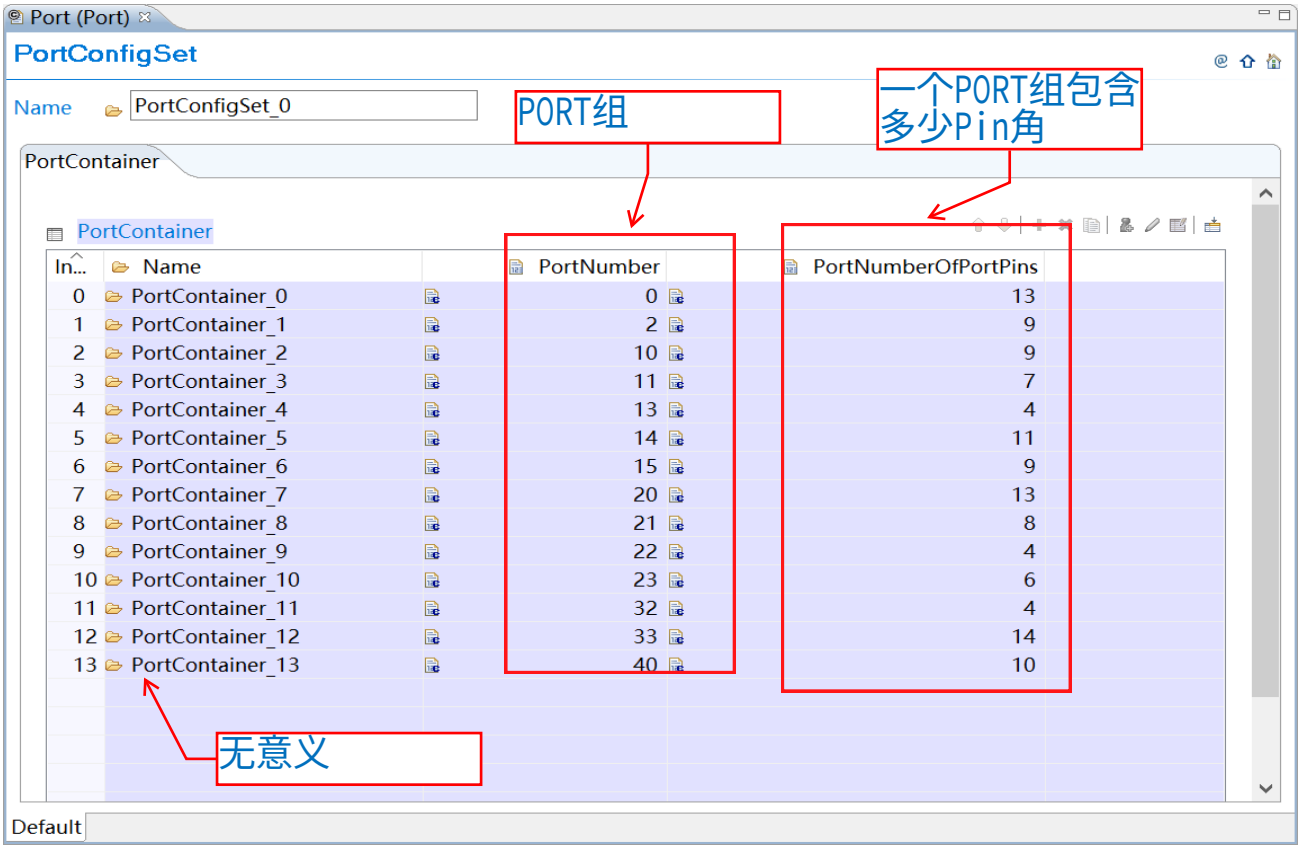
4、 Mcu 初始化

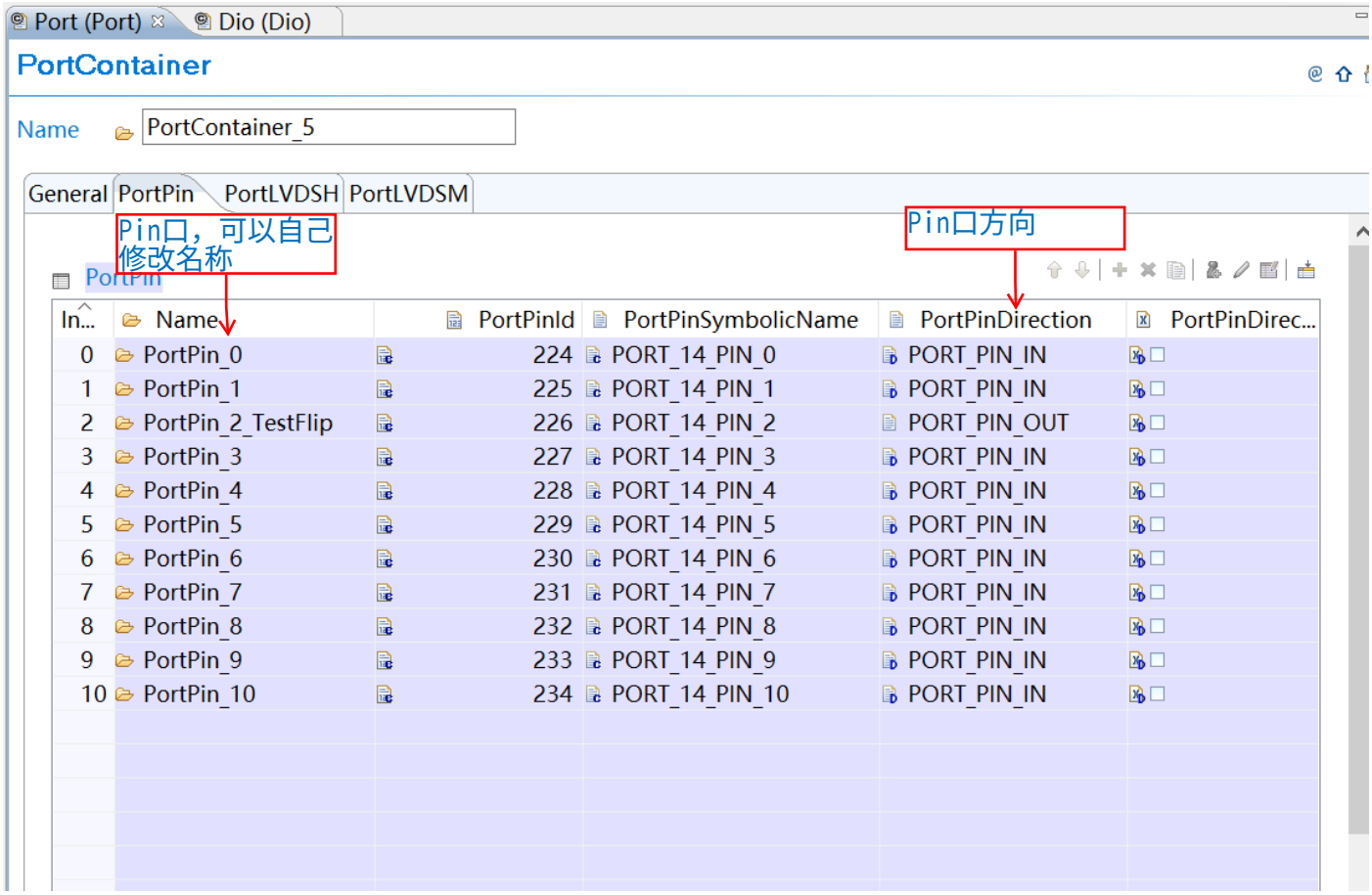
```
51 int main(void)
52 {
53     Mcu_Init(Mcu_ConfigRoot);
54     Mcu_InitClock(0);
55     while (MCU_PLL_UNLOCKED == Mcu_GetPllStatus())
56     {
57         /* wait for PLL locked */
58     }
59     Mcu_DistributePllClock();
60
```

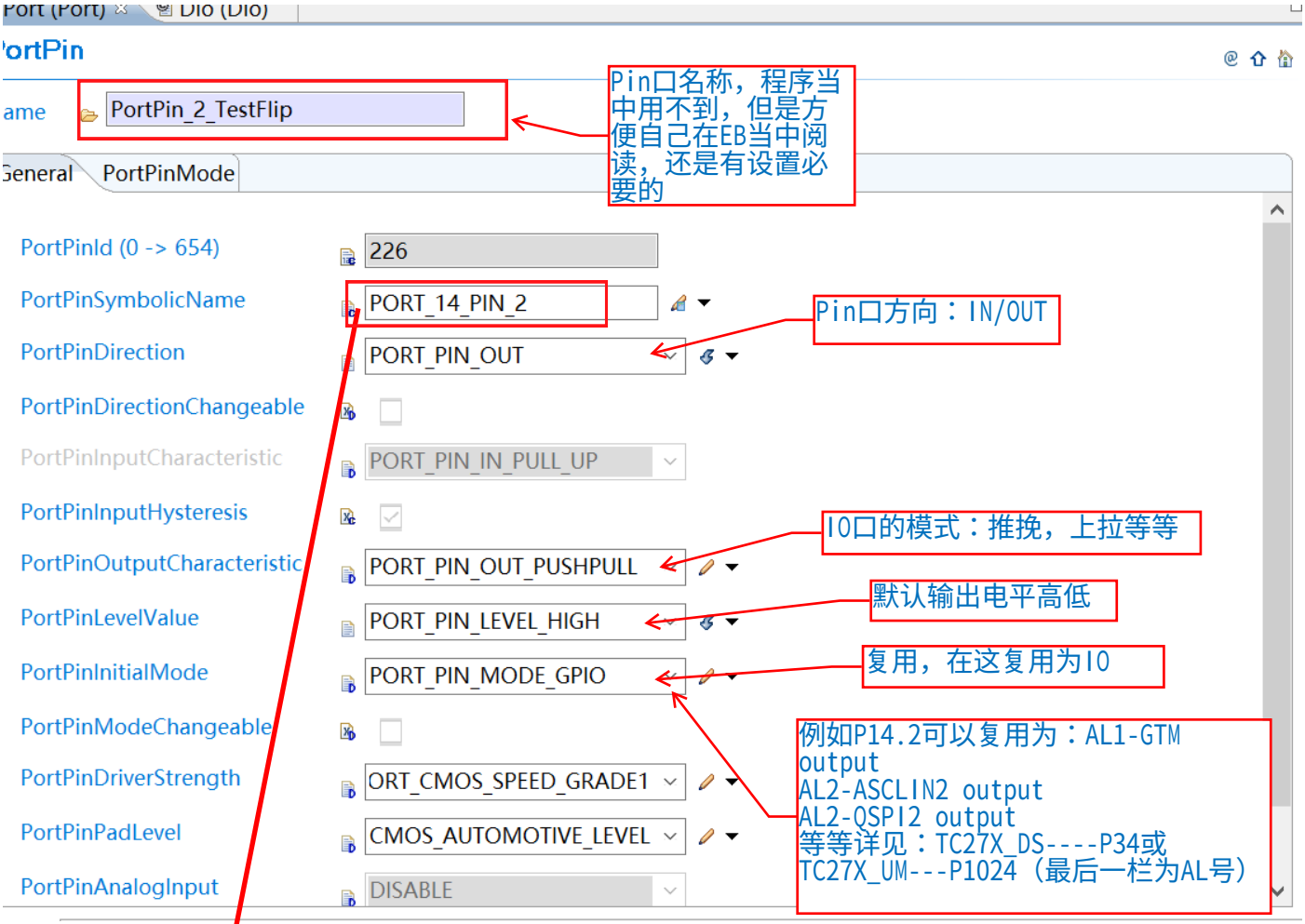
二、 普通 IO 的配置

在 MCAL 中，PORT 模块理解为 IO 复用（例如：普通 IO，ADC，SPI，IIC，CAN 等等），DIO 模块理解为 IO 口的输入输出功能。

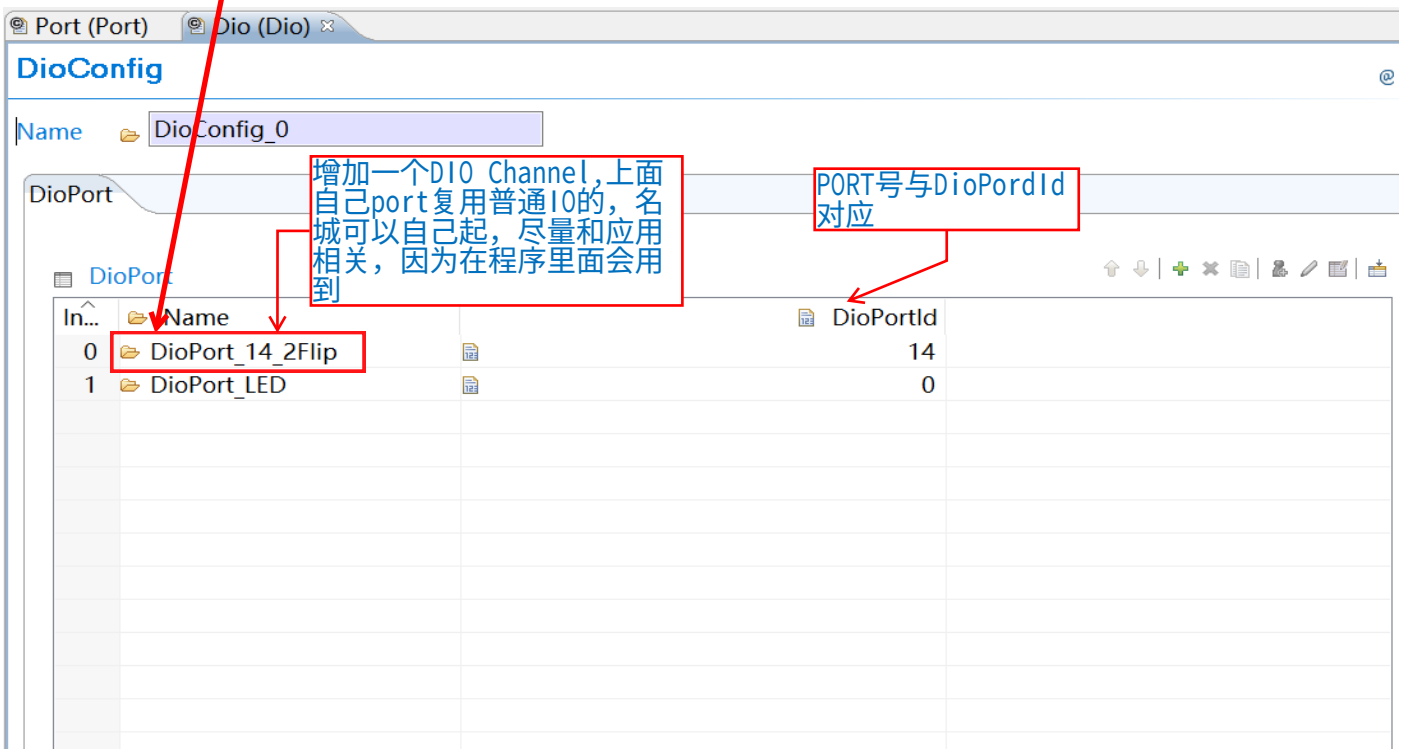
1、 PORT 模块

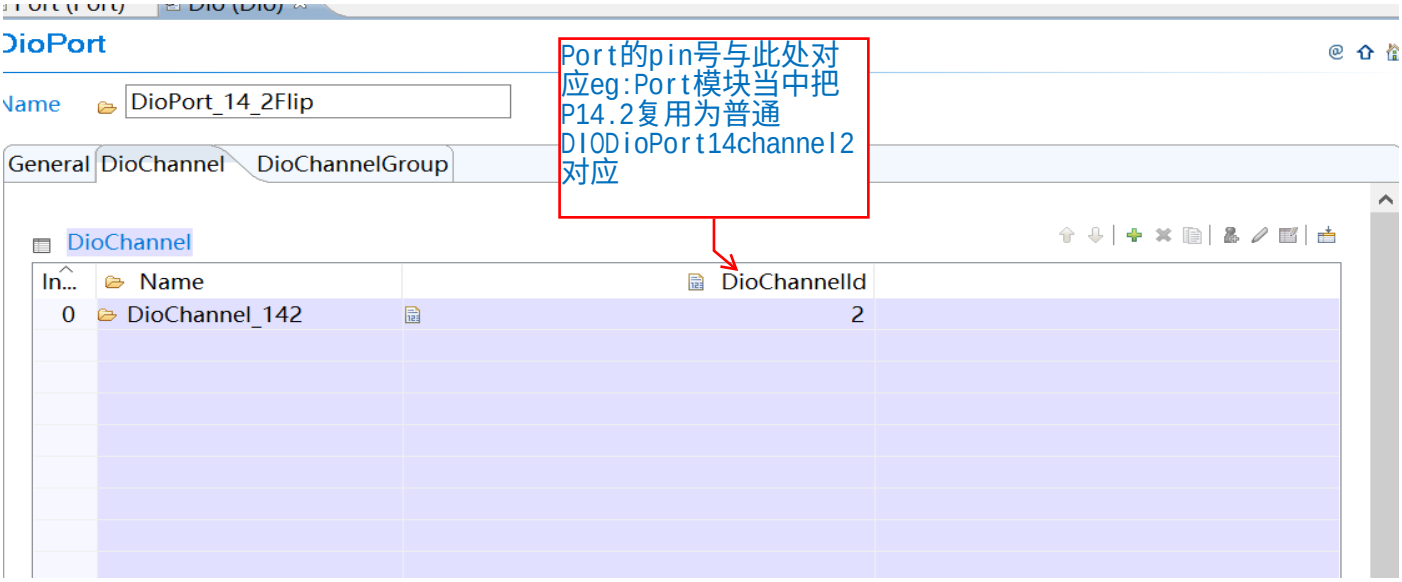





Port (Port)
Dio (Dio)
ortPin
ame
PortPin_2_TestFlip
Pin口名称，程序当中用不到，但是方便自己在EB当中阅读，还是有设置必要的
General
PortPinMode
PortPinId (0 -> 654)
226
PortPinSymbolicName
PORT_14_PIN_2
PortPinDirection
PORT_PIN_OUT
Pin口方向：IN/OUT
PortPinDirectionChangeable
PortPinInputCharacteristic
PORT_PIN_IN_PULL_UP
PortPinInputHysteresis
PortPinOutputCharacteristic
PORT_PIN_OUT_PUSHPULL
IO口的模式：推挽，上拉等等
PortPinLevelValue
PORT_PIN_LEVEL_HIGH
默认输出电平高低
PortPinInitialMode
PORT_PIN_MODE_GPIO
复用，在这复用为IO
PortPinModeChangeable
PortPinDriverStrength
ORT_CMOS_SPEED_GRADE1
PortPinPadLevel
CMOS_AUTOMOTIVE_LEVEL
PortPinAnalogInput
DISABLE
例如P14.2可以复用为：AL1-GTM output
AL2-ASCLIN2 output
AL2-QSPI2 output
等等详见：TC27X_DS----P34或TC27X_UM---P1024（最后一栏为AL号）

2、 DIO 模块


Port (Port)
Dio (Dio)
DioConfig
Name
DioConfig_0
DioPort
增加一个DIO Channel,上面自己port复用普通IO的，名城可以自己起，尽量和应用相关，因为在程序里面会用到
PORT号与DioPordId对应
In...
Name
DioPortId
0
DioPort_14_2Flip
14
1
DioPort_LED
0



3、 初始化及调用

```
/* Port Initialize  */
Port_Init(&Port_ConfigRoot[0]);


Dio_FlipChannel(DioConf_DioChannel_DioChannel_142);
```

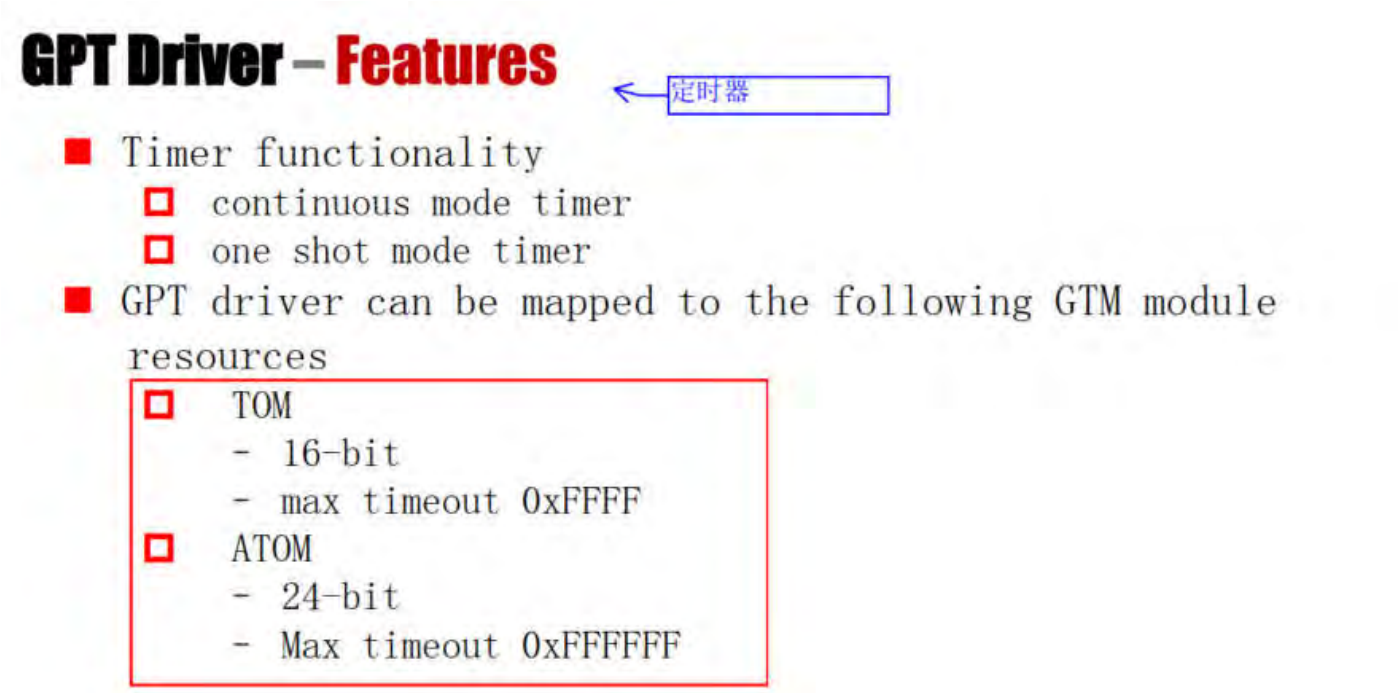
翻转P14.2

```
extern void Dio_WriteChannel
(
  Dio_ChannelType ChannelId,
  Dio_LevelType Level
);


extern Dio_LevelType Dio_ReadChannel
(
  Dio_ChannelType ChannelId
);
```

三. GTM 配置(例 TOMOCHO 定时 10ms)

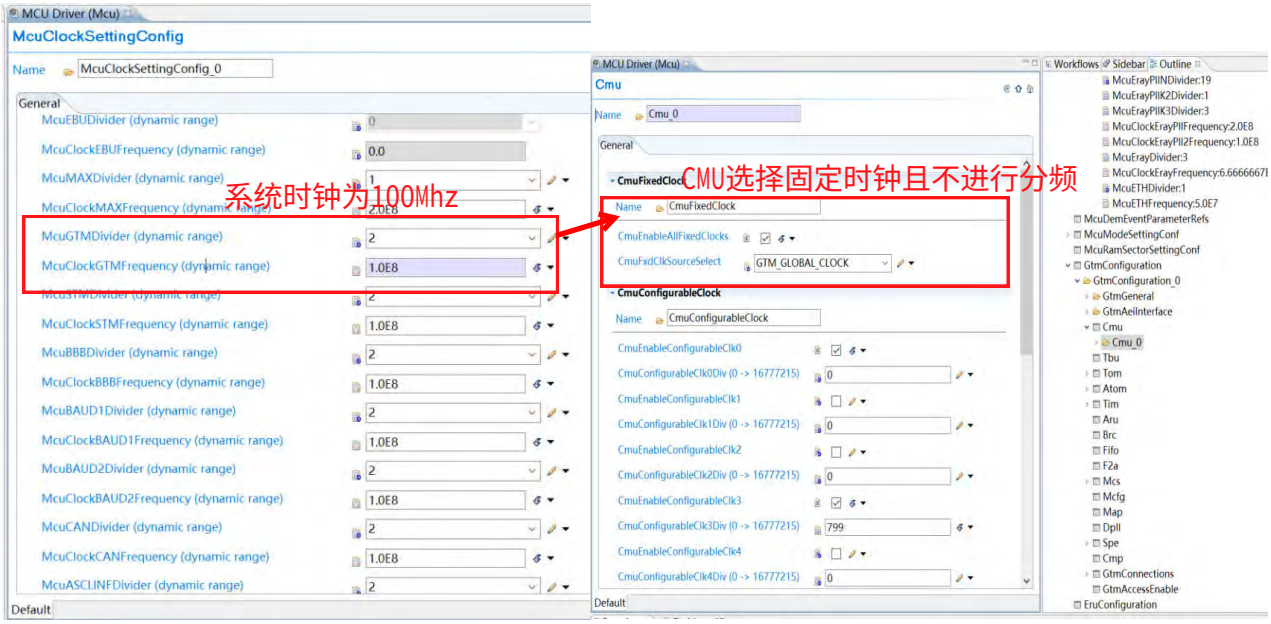
TC275 的定时时钟可以来源 tom,tim,Atom,CCU6 等等

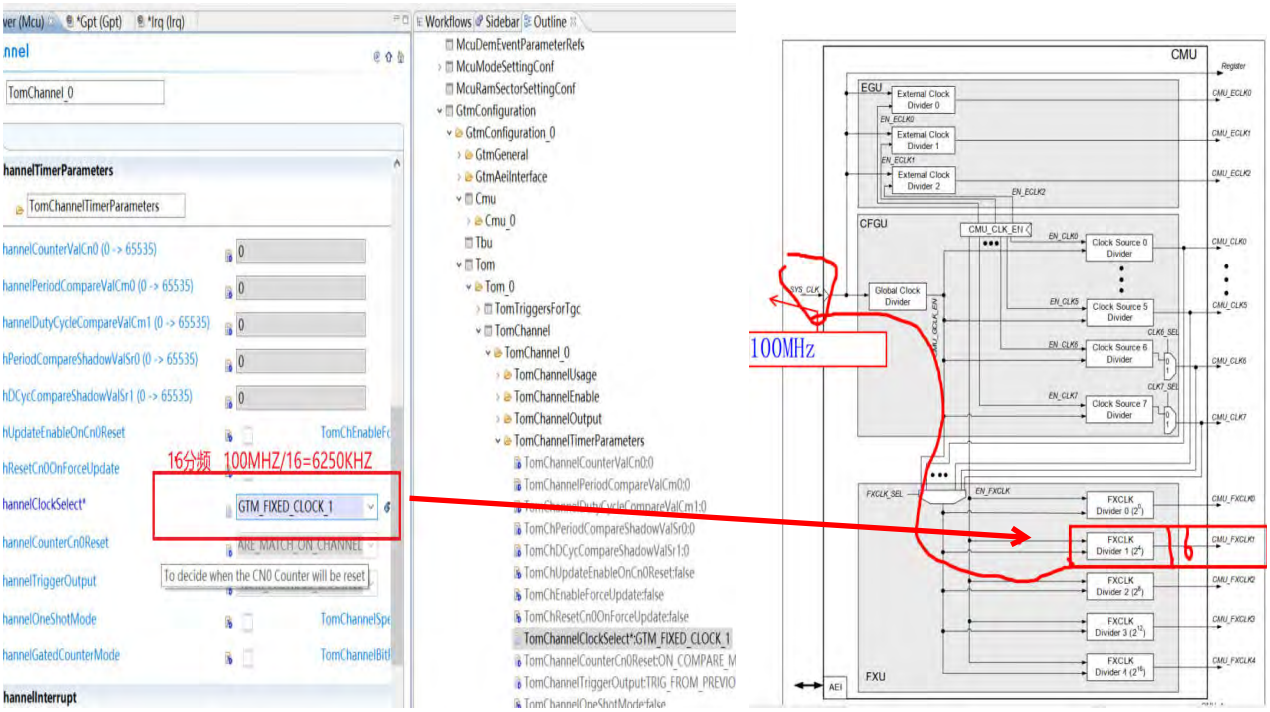
1、定时器简述



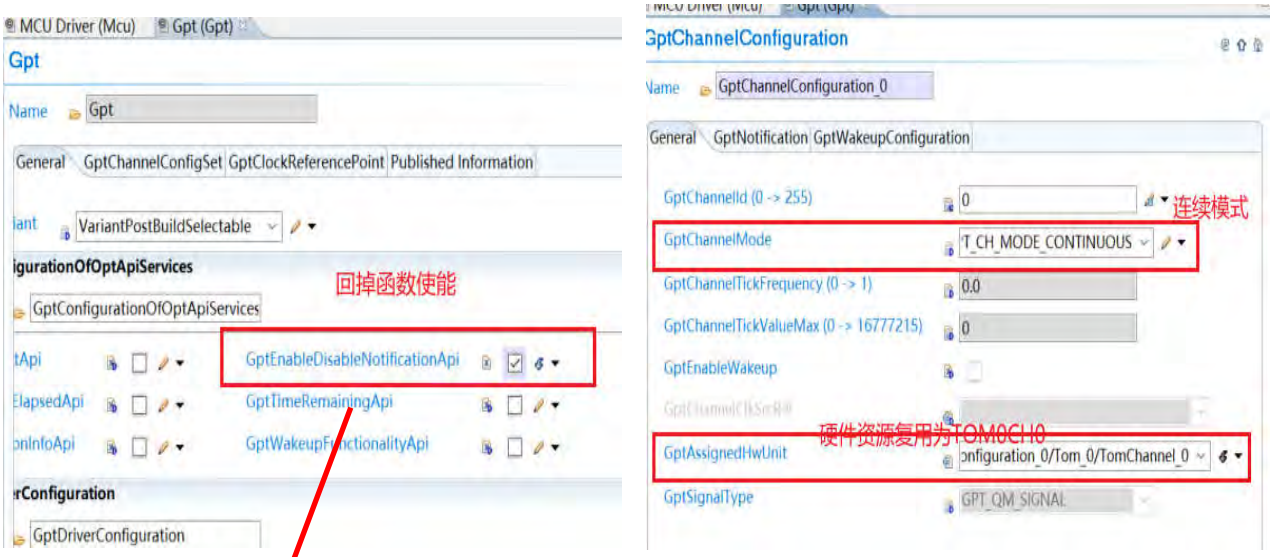
2.定时器时钟配置

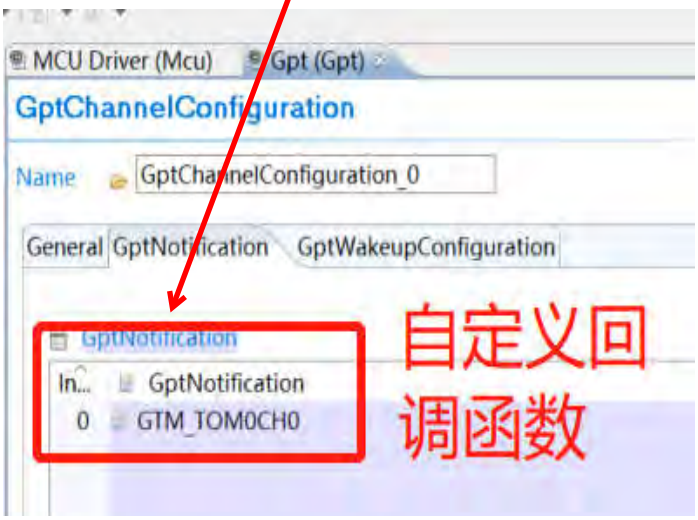
时钟经过系统，输入时钟管理单元 CMU,最后输入 TOM，ATOM 等外设资源



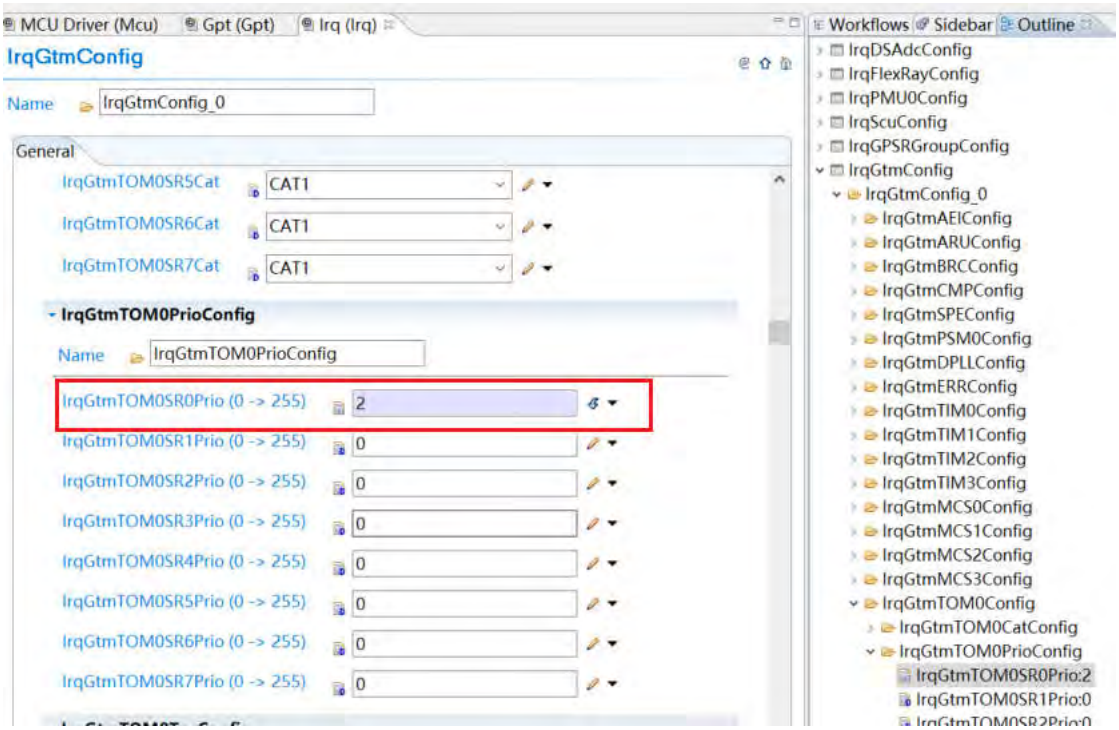

TomChannel_0
hannelTimerParameters
TomChannelTimerParameters
16分频 100MHZ/16=6250KHZ
GTM_FIXED_CLOCK_1
To decide when the CN0 Counter will be reset
Workflows
Sidebar
Outline
GtmConfiguration_0
Cmu_0
Tom_0
TomChannel_0
TomChannelTimerParameters
CMU
EGU
CFGU
FXU
100MHz
Global Clock Divider
FXCLK Divider 1 (2^4)

2. GPT 配置

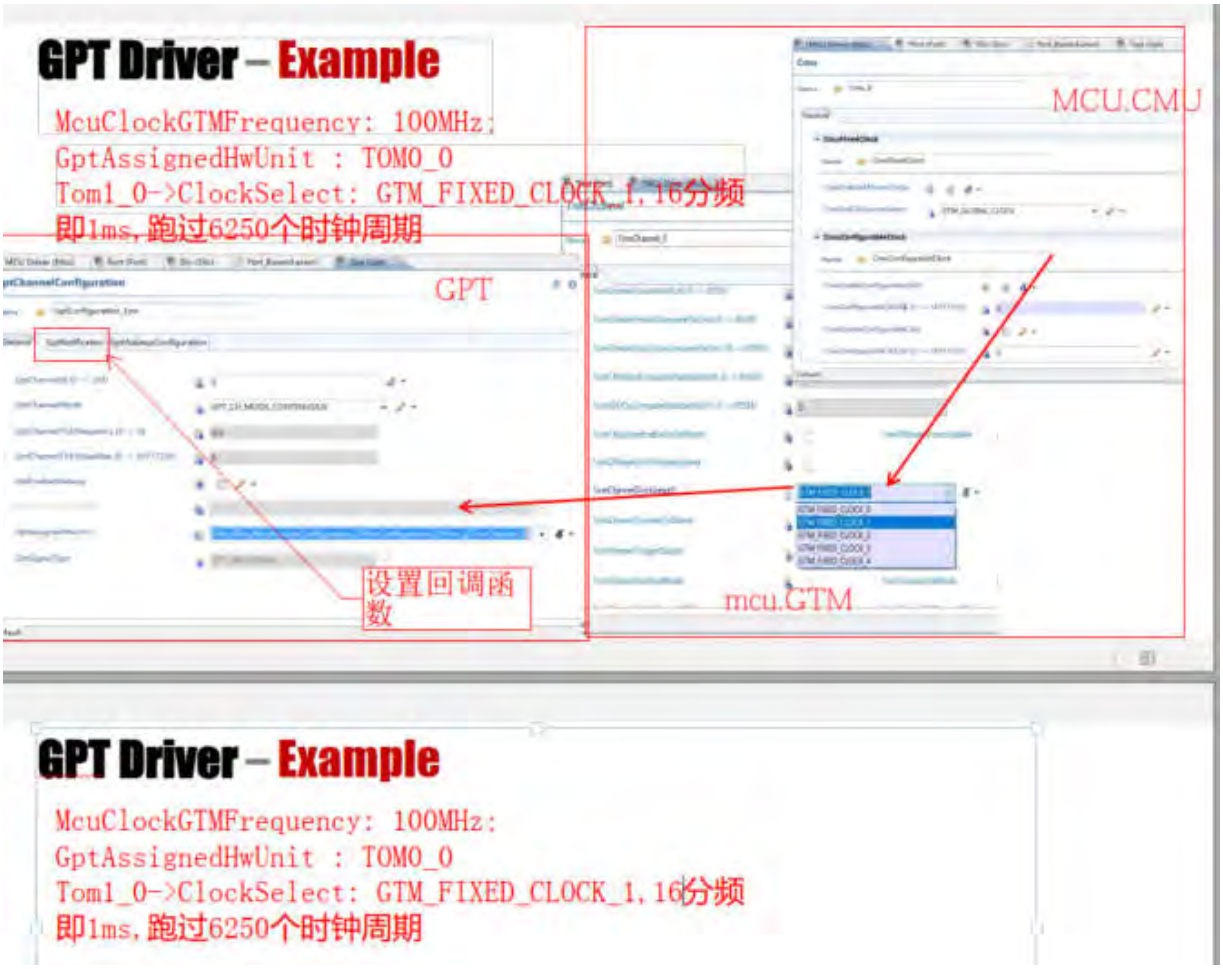

MCU Driver (Mcu)
Gpt (Gpt)
Gpt
Name
Gpt
General
GptChannelConfigSet
GptClockReferencePoint
Published Information
VariantPostBuildSelectable
GptConfigurationOfOptApiServices
回掉函数使能
GptEnableDisableNotificationApi
GptTimeRemainingApi
GptDriverConfiguration
GptChannelConfiguration
GptChannelConfiguration_0
GptNotification
GptWakeupConfiguration
GptChannelId (0 -> 255)
GptChannelMode
连续模式
GptChannelTickFrequency (0 -> 1)
GptChannelTickValueMax (0 -> 16777215)
GptEnableWakeup
硬件资源复用为TOM0CH0
GptAssignedHwUnit
GptSignalType
GPT_QM_SIGNAL


MCU Driver (Mcu)
Gpt (Gpt)
GptChannelConfiguration
Name
GptChannelConfiguration_0
General
GptNotification
GptWakeupConfiguration
GptNotification
In...
GptNotification
0
GTM_TOM0CH0
自定义回调函数

3. 中断优先级配置



4. Example 使用



TC275_HighTec_GTM_TOM0CH0 ["TriCore 4.9.3.0" - TC27X_Debug]

回掉函数，直接调用

```c
#include "Can_17_MCanP.h"
//#include "Bsw_Test.h"
#include "Icu_17_GtmCcu6.h"
#include "Pwm_17_Gtm.h"
#include "Spi.h"


void GTM_TOM0CH0(void)
{
    Dio_FlipChannel(DioConf_DioPort_DioPort2_LED1);
    Dio_FlipChannel(DioConf_DioPort_DioPort0_LED2);
}

int main(void)
{
    Mcu_Init(Mcu_ConfigRoot);
    Mcu_InitClock(0);
    while (MCU_PLL_UNLOCKED == Mcu_GetPllStatus())
    {
        /* wait for PLL locked */
    }
    Mcu_DistributePllClock();



    /* Port Initialize  */
    Port_Init(&Port_ConfigRoot[0]);

    /* GPT Initialize */
    Gpt_Init(Gpt_ConfigRoot);

    /* IrqGtm_Init */
    IrqGtm_Init();
    /* Gpt enable 1ms notification,and start */
    Gpt_EnableNotification(GptConf_GptChannel_GptChannelConfiguration_0);
    Gpt_StartTimer(GptConf_GptChannel_GptChannelConfiguration_0, 6250);

    Mcal_EnableAllInterrupts();//开全局中断

    while(1)
    {

    }


}
```

四. PWM 配置（TOM1CH0 作为参考 PWM，TOM1CH1—6 生成 10KHzPWM，参考 PWM 下降沿触发中断）

1. 功能简介

1. 支持四种类型的PWM通道
- 固定周期
- 固定周期偏移
- 可变周期
- 固定周期中心对称

2. 周期，占空比，极性可配置

3. 占空比或周期更新时刻点设置：
- 立即生效
- 当前周期结束生效

4. 允许占空比或周期更新的通道设置：
- 各个通道单独生效
- 所有PWM通道都生效

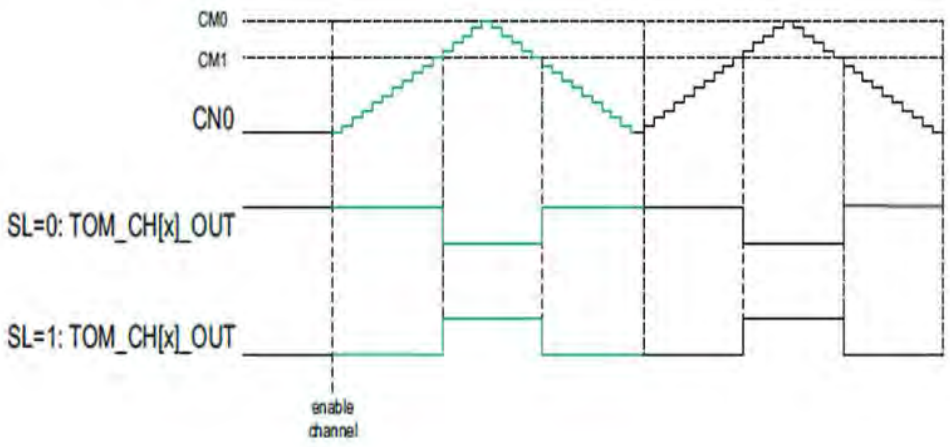
5. PWM通道允许设定一个空闲态电平，提供API用于使输出信号改变为定义的空闲态电平

6. 支持硬件源为GTM或CCU6

TOM模块工作简介
- 连续模式——输出连续PWM信号
- 单次模式——每次触发后生成单个周期的PWM信号

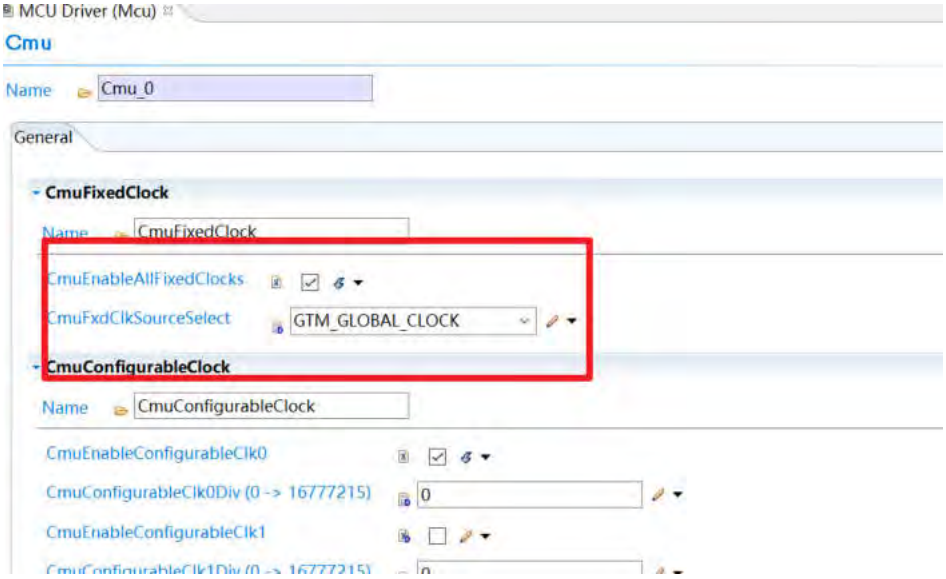
以连续模式为例：

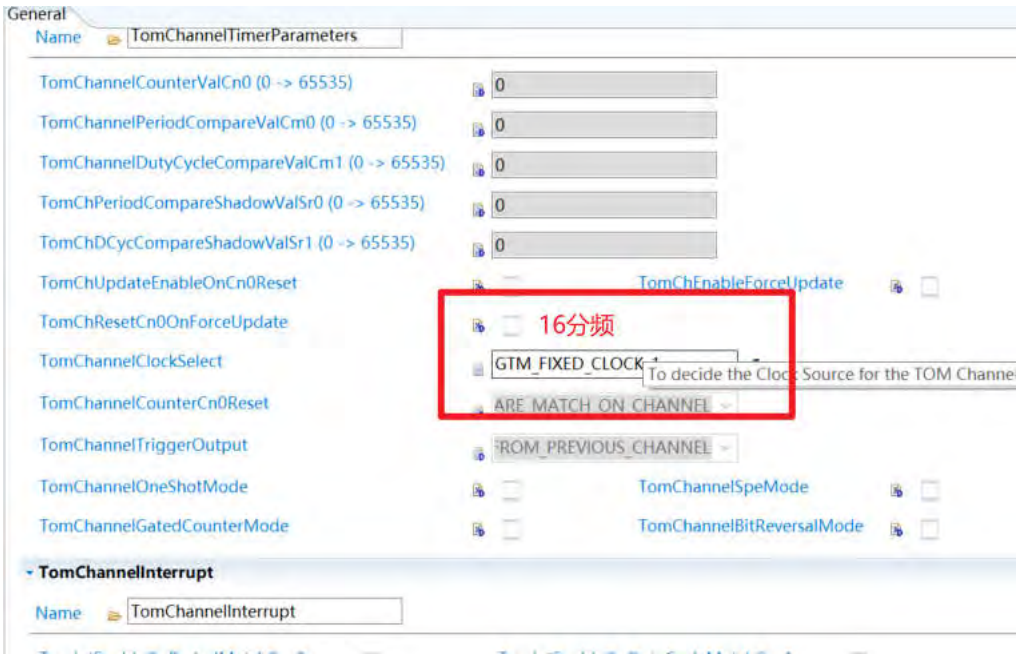
2. Continuous Counting Up-Down Mode
- CN0初始化值可设置
- PWM信号极性由SL位决定
- CM0参数决定了周期长度，CM0和CM1共同决定了占空比长度

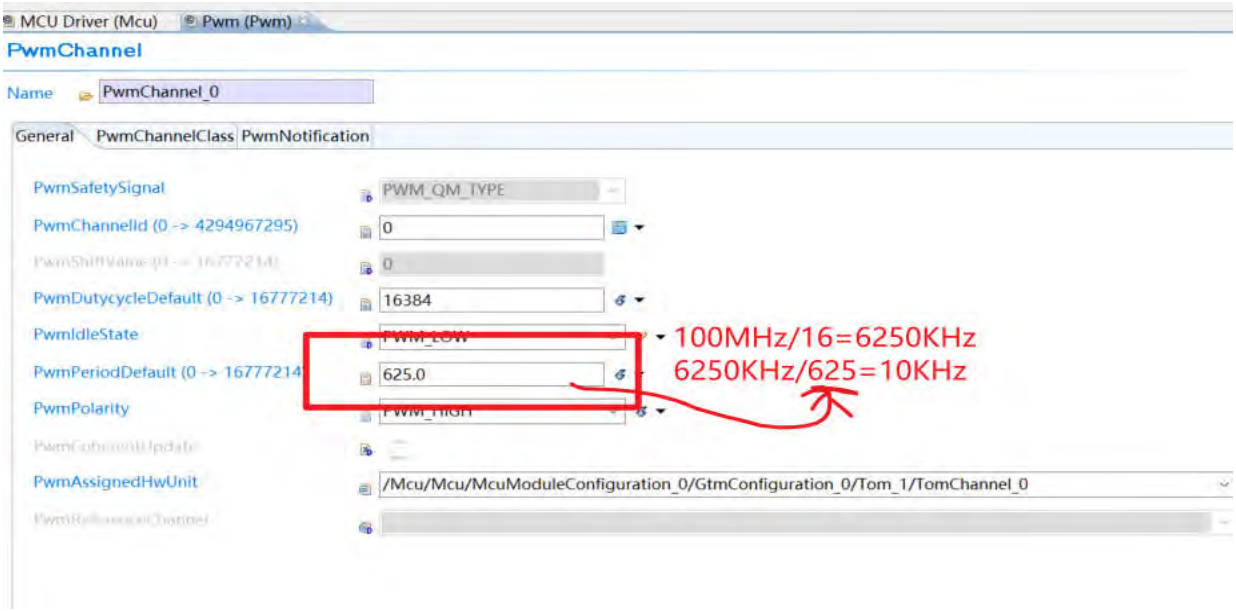


2.时钟配置

中心对齐模式下，仅需要配置参考的 PWM 周期，配置方式与 GTM 配置相同

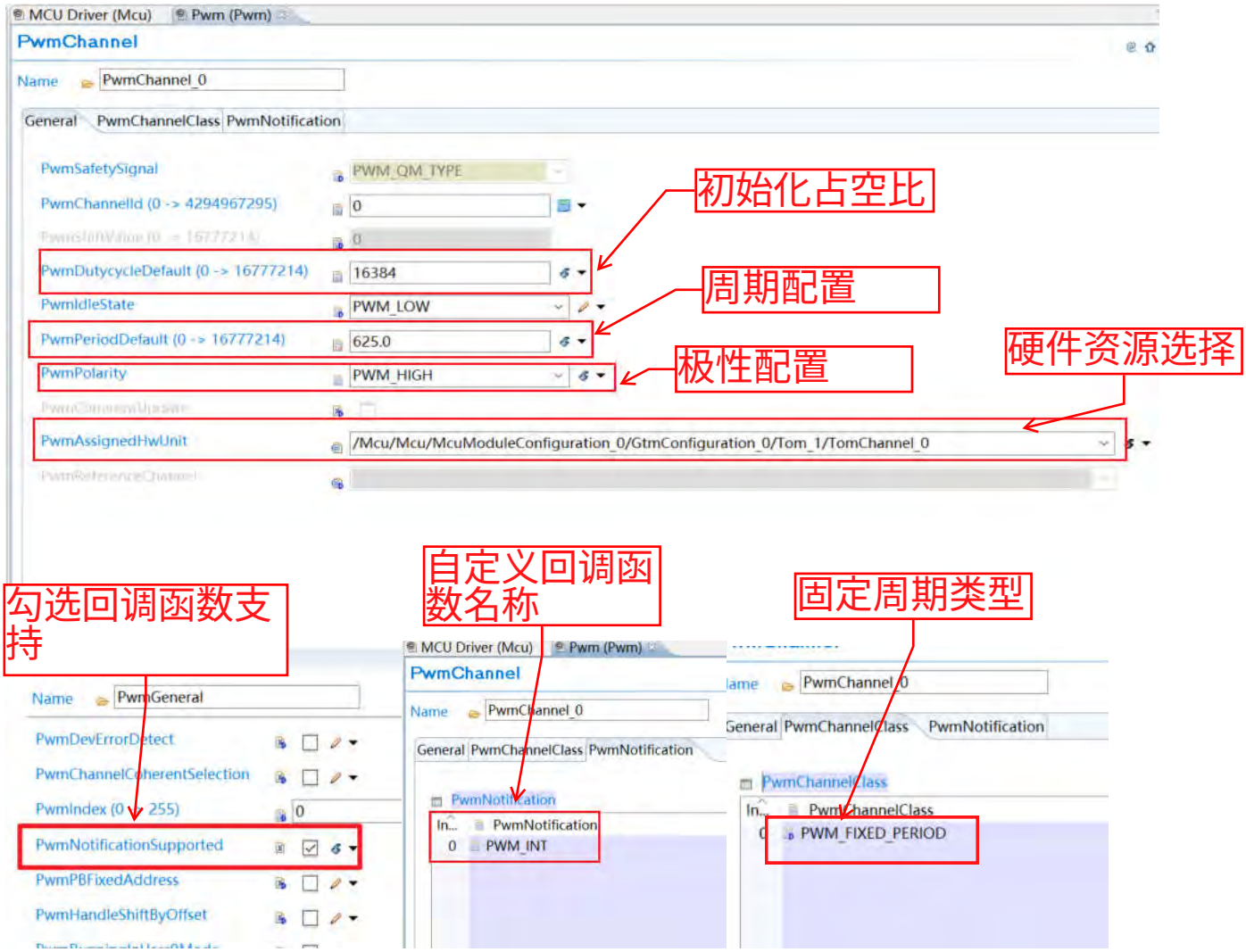


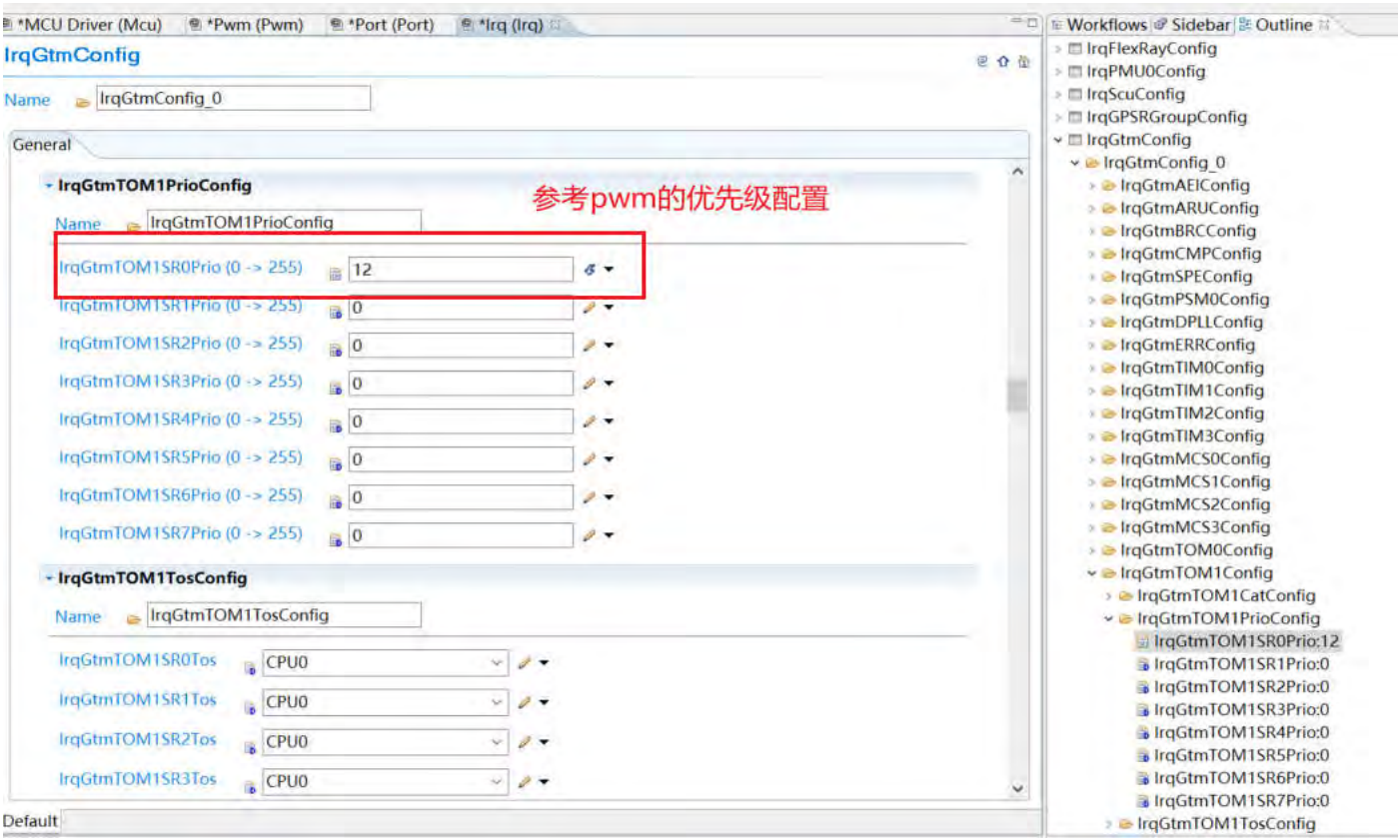




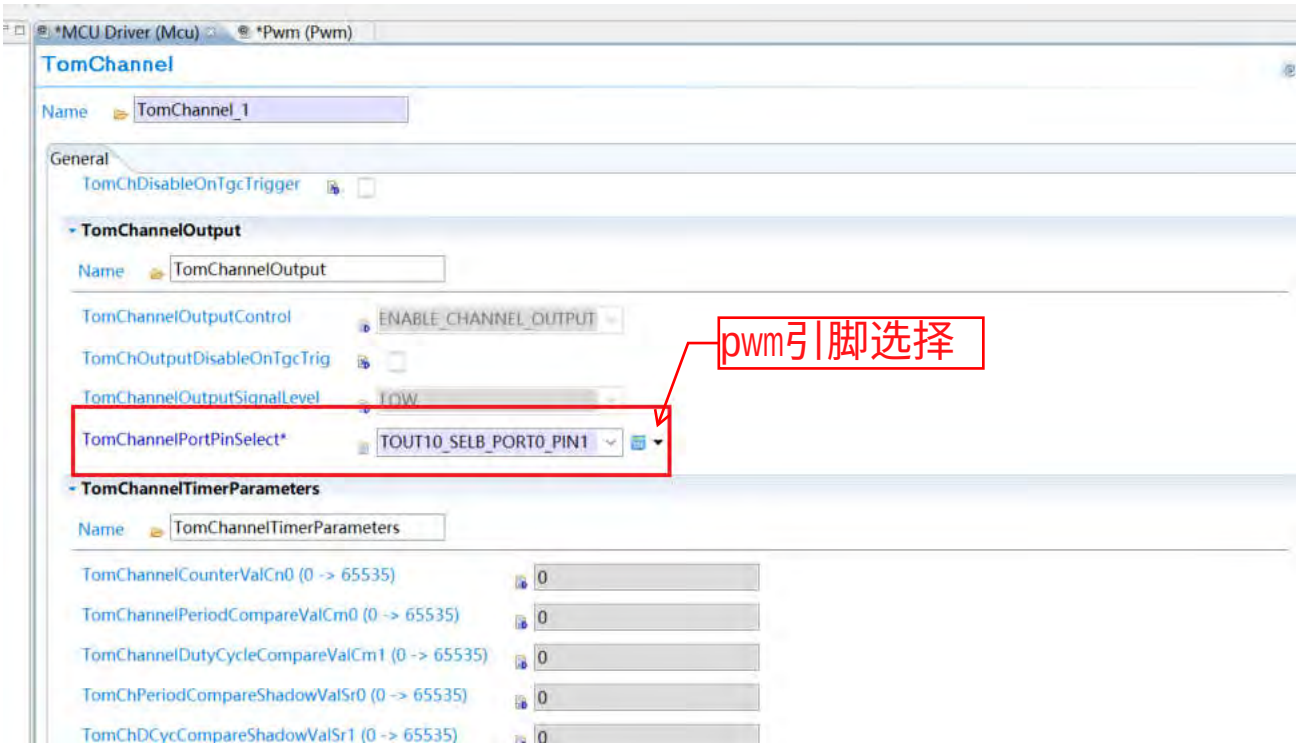
2. PWM 配置

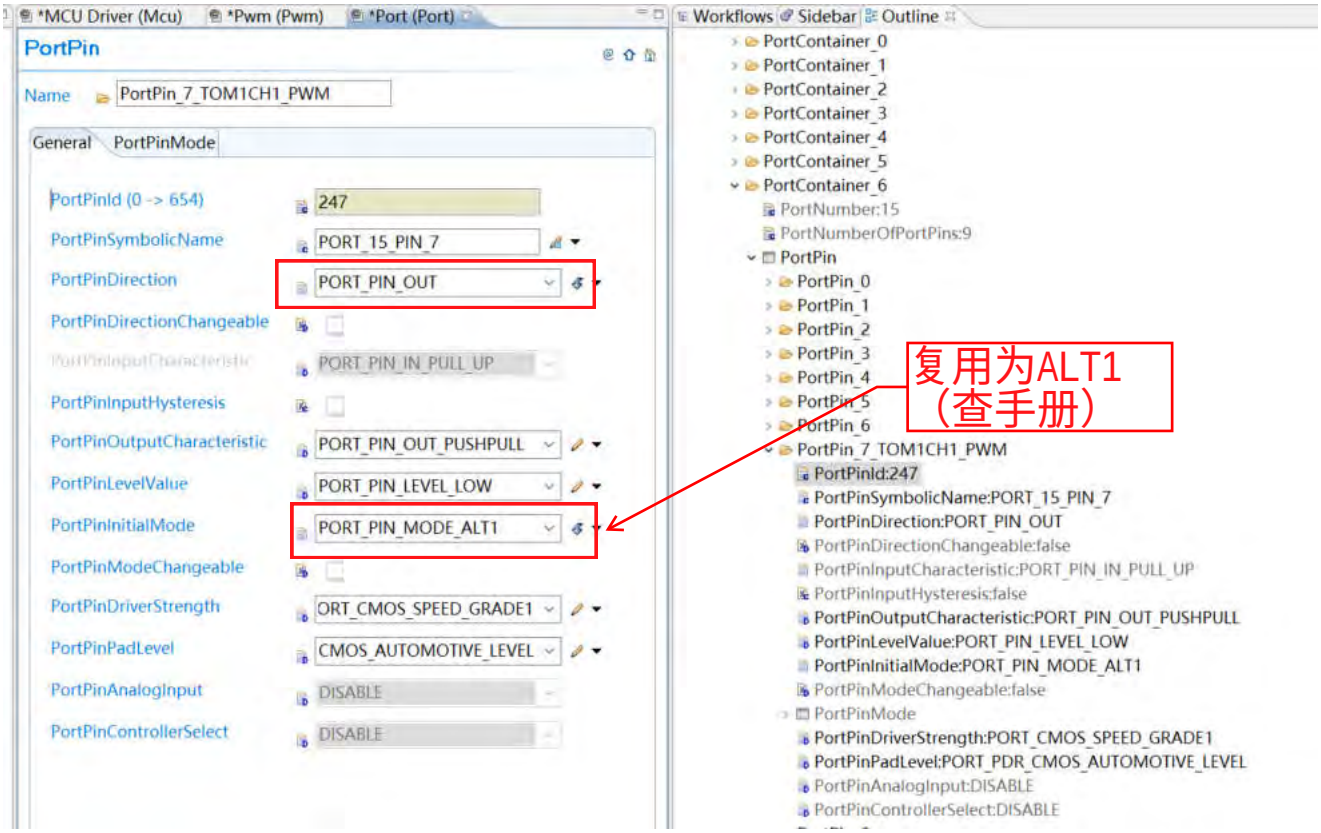
（1）参考 pwm 配置-TOM1CH0

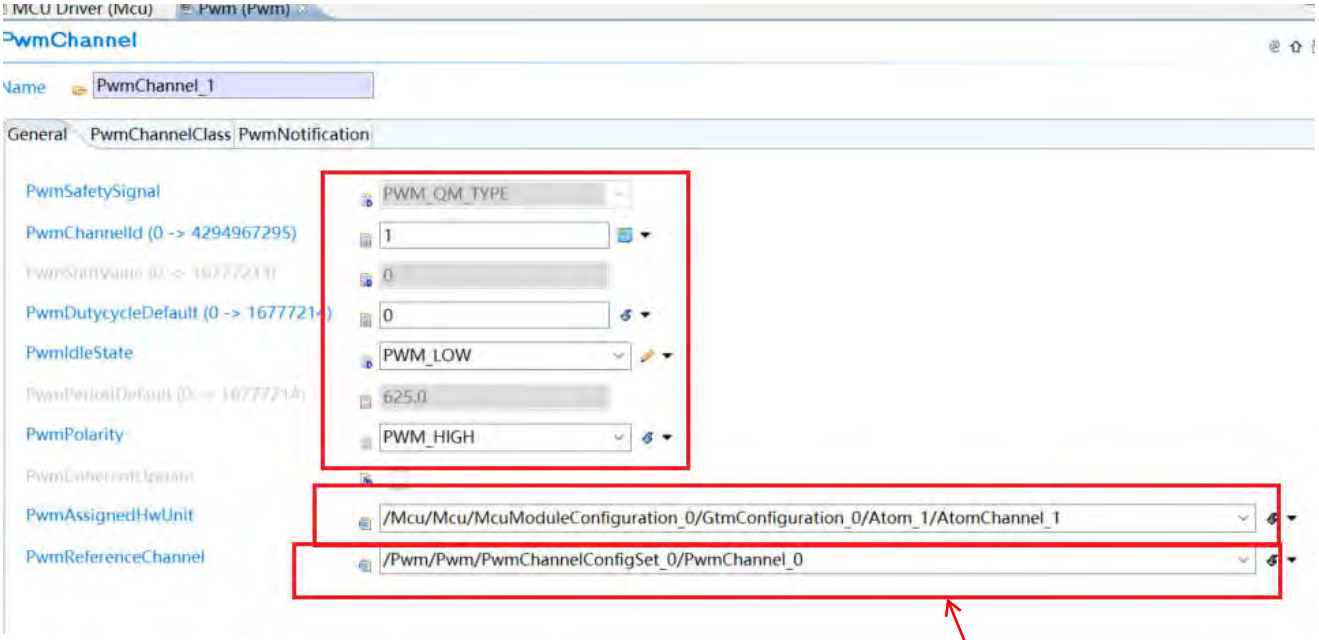




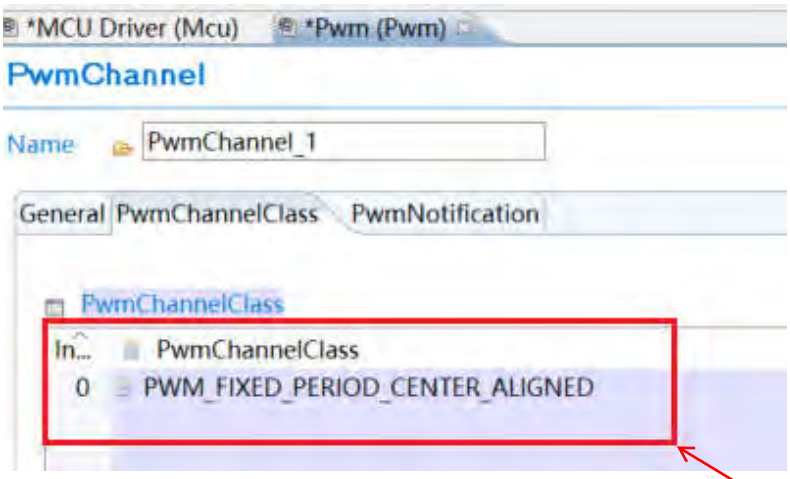
（2）中心对齐 pwm 配置-TOM1CH1







参考pwm通道选择



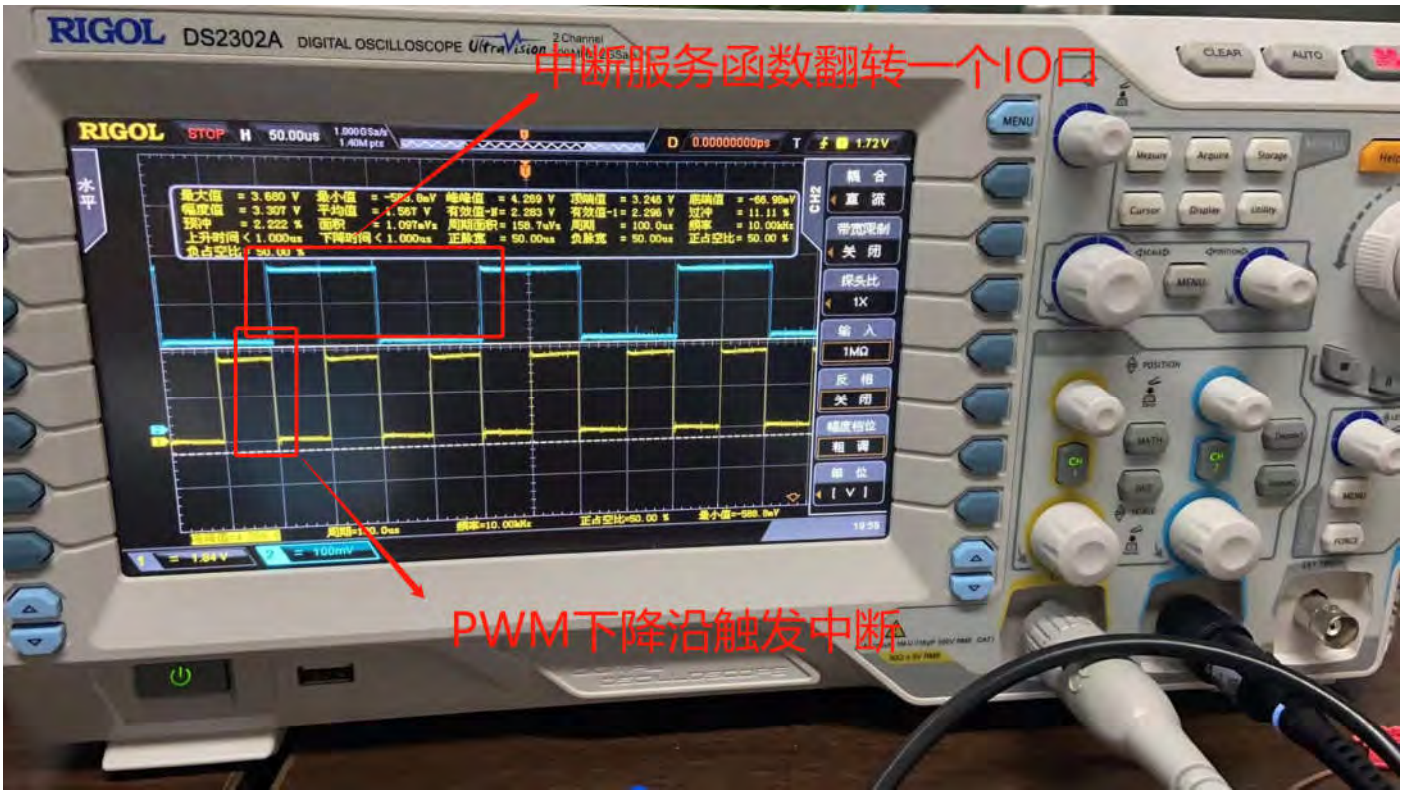
中心对其模式

（2）Hightec 调用

```
uint16 PWM_00_9_Duty=0;

void PWM_INT(void)
{

    Dio_FlipChannel(DioConf_DioPort_DioPort2_LED1);
    Dio_FlipChannel(DioConf_DioPort_DioPort0_LED2);
    Pwm_17_Gtm_SetDutyCycle(Pwm_17_GtmConf_PwmChannel_PwmChannel_1,PWM_00_9_Duty);
}

void GTM_TOM0CH0(void)
{
//  Pwm_17_Gtm_SetPeriodAndDuty(Pwm_17_GtmConf_PwmChannel_PwmChannel_1,625, PWM_00_9_Duty  );

//  Dio_FlipChannel(DioConf_DioPort_DioPort2_LED1);
//  Dio_FlipChannel(DioConf_DioPort_DioPort0_LED2);
//  Pwm_17_Gtm_SetDutyCycle(Pwm_17_GtmConf_PwmChannel_PwmChannel_1,PWM_00_9_Duty);
}

int main(void)
{
    Mcu_Init(Mcu_ConfigRoot);
    Mcu_InitClock(0);
    while (MCU_PLL_UNLOCKED == Mcu_GetPllStatus())
    {
        /* wait for PLL locked */
    }
    Mcu_DistributePllClock();

    /* IrqGtm_Init */
    IrqGtm_Init();

    /* Port Initialize  */
    Port_Init(&Port_ConfigRoot[0]);

    /* GPT Initialize */
    Gpt_Init(Gpt_ConfigRoot);


    /* Gpt enable 1ms notification,and start */
    Gpt_EnableNotification(GptConf_GptChannel_GptChannelConfiguration_0);
    Gpt_StartTimer(GptConf_GptChannel_GptChannelConfiguration_0, 6250);

    /* Pwm Initialize */
    Pwm_17_Gtm_Init(&Pwm_ConfigRoot[0]);

    /* Enable Notification*/
    Pwm_17_Gtm_EnableNotification(Pwm_17_GtmConf_PwmChannel_PwmChannel_0,PWM_FALLING_EDGE);

    Mcal_EnableAllInterrupts();//开全局中断
```

eb里面定义的回调函数

初始化以及使能下降沿触发回调函数



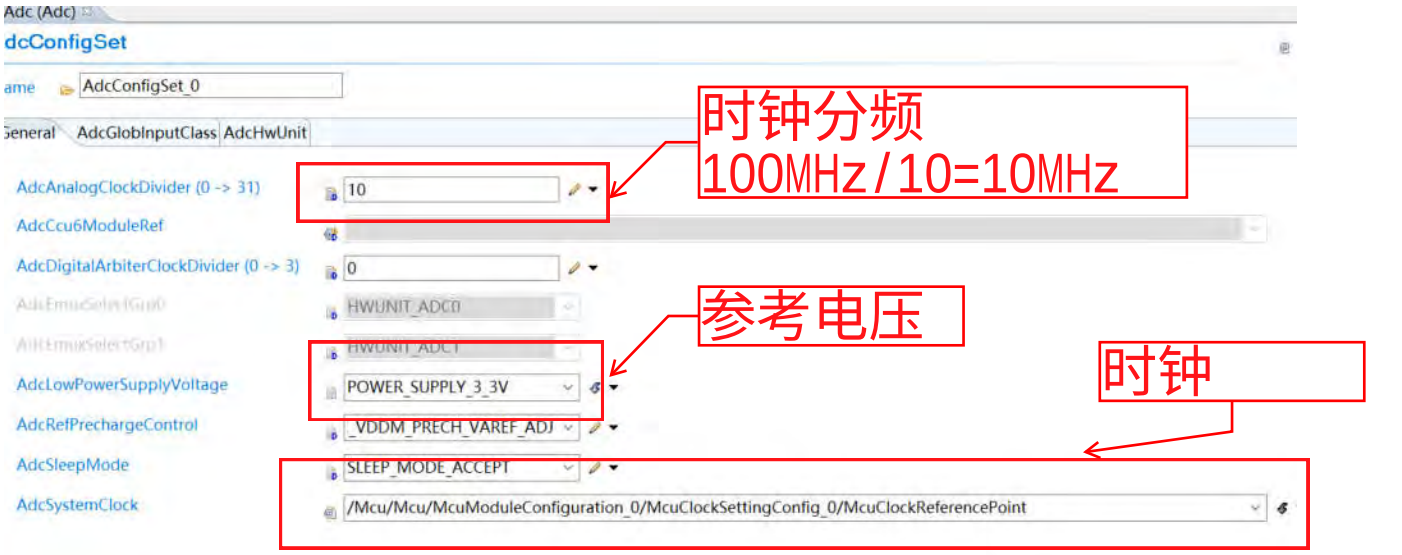
五． ADC 配置

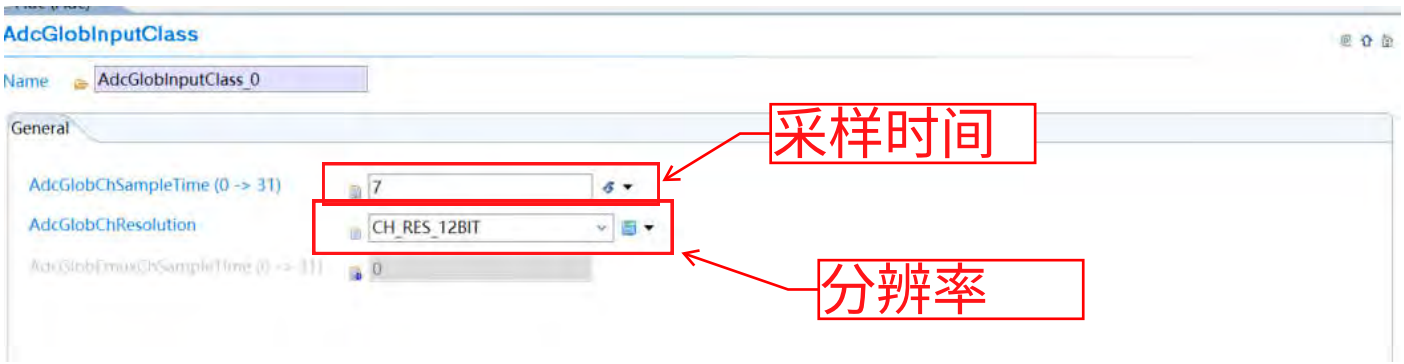
1, adc 简介

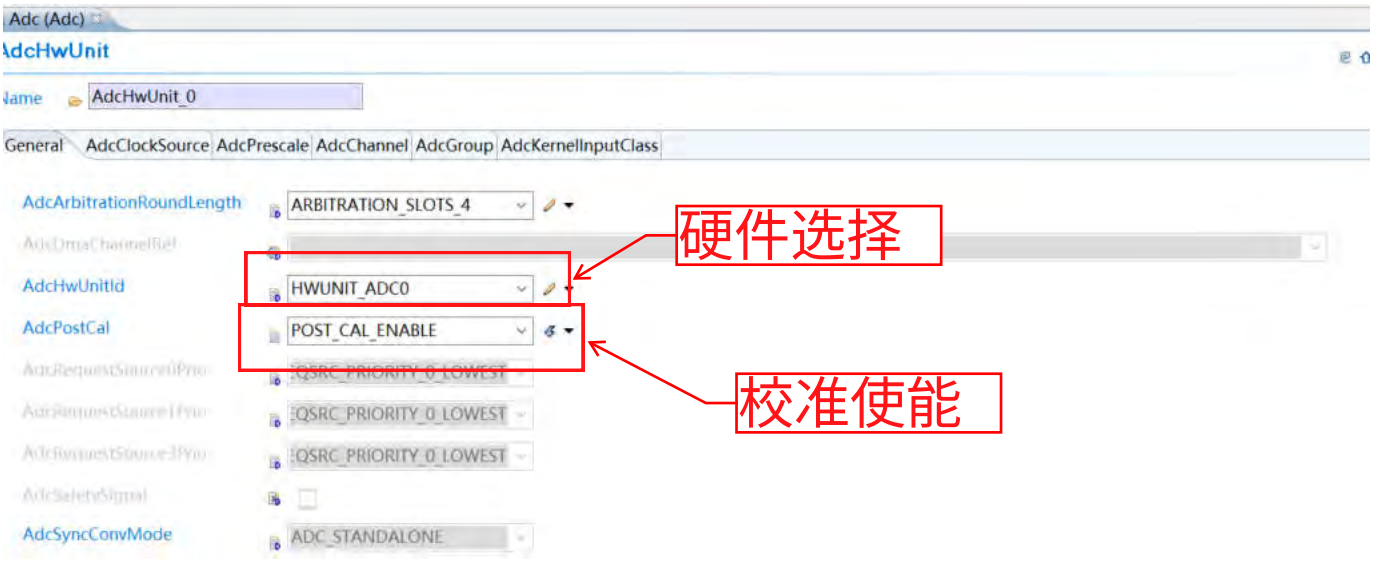
1. 可单通道或多通道进行单次或连续转换
2. 支持多通道间的同步采样、同步转换以及相位同步
3. 支持软件和硬件信号触发转换
   - 硬件触发信号有GTM内部周期信号和ERU外部引脚触发信号等
4. 转换速度和采样时间可配置
5. 转换结果分辨率为 12 位，可选的结果对齐模式
6. 支持CH0和CH1的替换特性
7. 支持通道边界极限检测
8. 支持多种中断服务请求
9. 安全等级: ASIL B
10. 多核资源分配：按照ADC内核（AdcHwUnit）分配

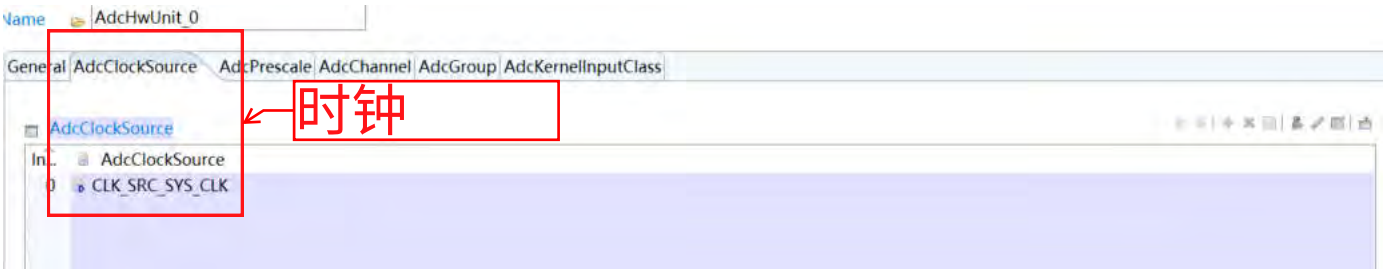
5.1 、 软件触发

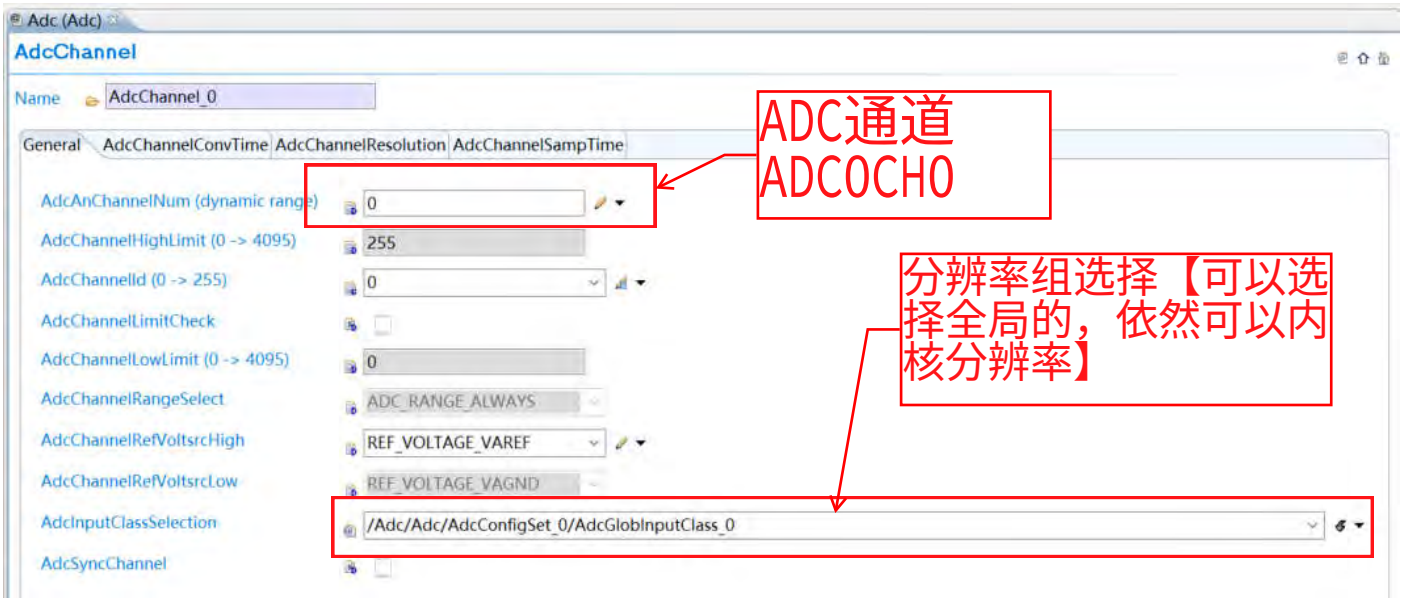
第二页可以知道 ADC 时钟来源系统时钟 Fspb=100MHz

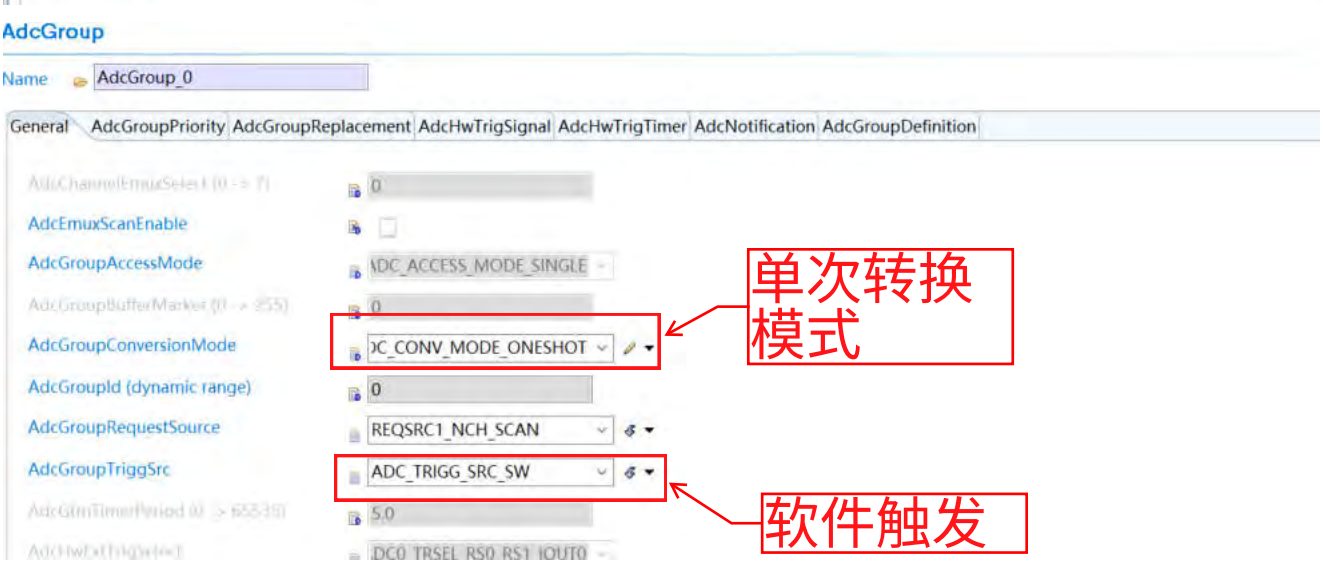


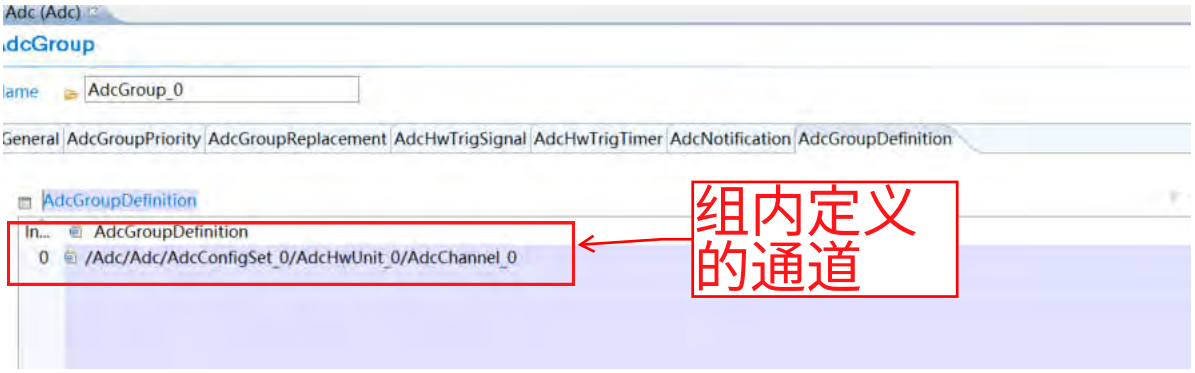


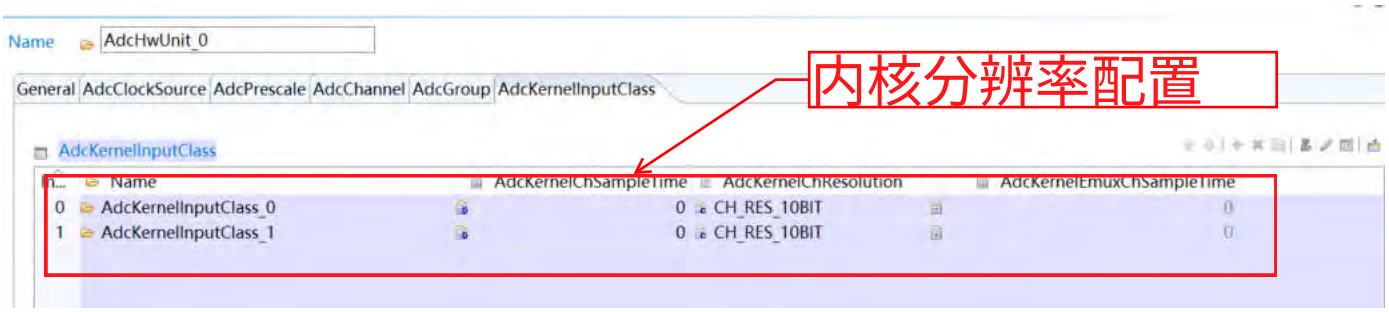

Adc (Adc)
AdcHwUnit
Name
AdcHwUnit_0
General
AdcClockSource
AdcPrescale
AdcChannel
AdcGroup
AdcKernelInputClass
AdcArbitrationRoundLength
ARBITRATION_SLOTS_4
AdcHwUnitId
HWUNIT_ADC0
AdcPostCal
POST_CAL_ENABLE
AdcSyncConvMode
ADC_STANDALONE
硬件选择
校准使能


Name
AdcHwUnit_0
General
AdcClockSource
AdcPrescale
AdcChannel
AdcGroup
AdcKernelInputClass
AdcClockSource
AdcClockSource
CLK_SRC_SYS_CLK
时钟


Adc (Adc)
AdcChannel
Name
AdcChannel_0
General
AdcChannelConvTime
AdcChannelResolution
AdcChannelSampTime
AdcAnChannelNum (dynamic range)
0
AdcChannelHighLimit (0 -> 4095)
255
AdcChannelId (0 -> 255)
0
AdcChannelLimitCheck
AdcChannelLowLimit (0 -> 4095)
0
AdcChannelRangeSelect
ADC_RANGE_ALWAYS
AdcChannelRefVoltsrcHigh
REF_VOLTAGE_VAREF
AdcChannelRefVoltsrcLow
REF_VOLTAGE_VAGND
AdcInputClassSelection
/Adc/Adc/AdcConfigSet_0/AdcGlobInputClass_0
AdcSyncChannel
ADC通道
ADC0CH0
分辨率组选择【可以选择全局的，依然可以内核分辨率】


AdcGroup
Name
AdcGroup_0
General
AdcGroupPriority
AdcGroupReplacement
AdcHwTrigSignal
AdcHwTrigTimer
AdcNotification
AdcGroupDefinition
AdcEmuxScanEnable
AdcGroupAccessMode
AdcGroupConversionMode
AdcGroupId (dynamic range)
0
AdcGroupRequestSource
REQSRC1_NCH_SCAN
AdcGroupTriggSrc
ADC_TRIGG_SRC_SW
5.0
单次转换模式
软件触发





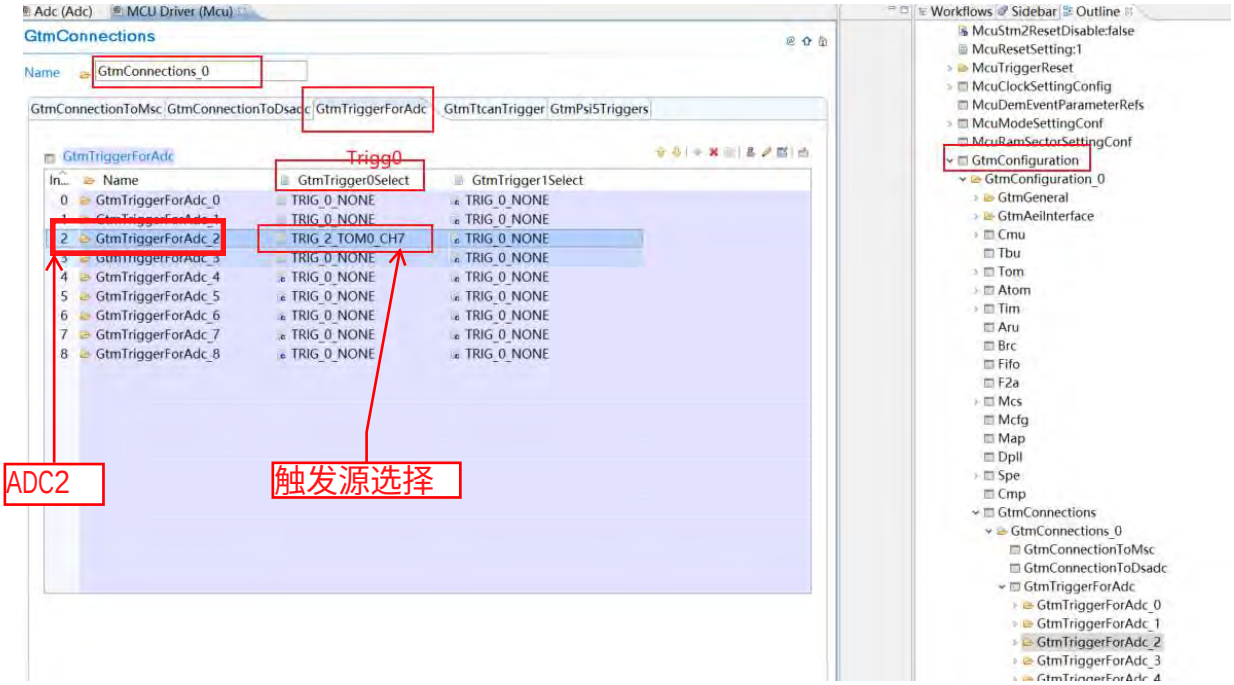

```c
#include "Spi.h"

uint16 PWM_00_9_DutyCycle = 2000;
uint16 PWM_00_9_Duty=0,
Adc_ValueGroupType SoftWareADCResult[2];


void ADC_TestSoftWare(void)
{
    Adc_StartGroupConversion(AdcConf_AdcGroup_AdcGroup_0);
    Adc_17_GetGroupResult(AdcConf_AdcGroup_AdcGroup_0,SoftWareADCResult);
}


void PWM_INT(void)
{
    Dio_WriteChannel(DioConf_DioChannel_LED1,0);

//  Dio_FlipChannel(DioConf_DioPort_DioPort2_LED1);
    Dio_FlipChannel(DioConf_DioPort_DioPort0_LED2);


    Pwm_17_Gtm_SetDutyCycle(Pwm_17_GtmConf_PwmChannel_PwmChannel_1_CH1,((PWM_00_9_DutyCycle-100)*((0x8000u)/5000.0)));
    Pwm_17_Gtm_SetDutyCycle(Pwm_17_GtmConf_PwmChannel_PwmChannel_2_CH1N,((PWM_00_9_DutyCycle+100)*((0x8000u)/5000.0)));


    ADC_TestSoftWare();
//  PWM_00_9_DutyCycle++;
//  if(PWM_00_9_DutyCycle>4500)
//  {
//      PWM_00_9_DutyCycle=500;
//  }
    Dio_WriteChannel(DioConf_DioChannel_LED1,1);

}


void GTM_TOM0CH0(void)
{
//  Dio_FlipChannel(DioConf_DioPort_DioPort2_LED1);
//  Dio_FlipChannel(DioConf_DioPort_DioPort0_LED2);
//  Pwm_17_Gtm_SetDutyCycle(Pwm_17_GtmConf_PwmChannel_PwmChannel_1,PWM_00_9_Duty);
}

int main(void)
{
```

10KHz的中断

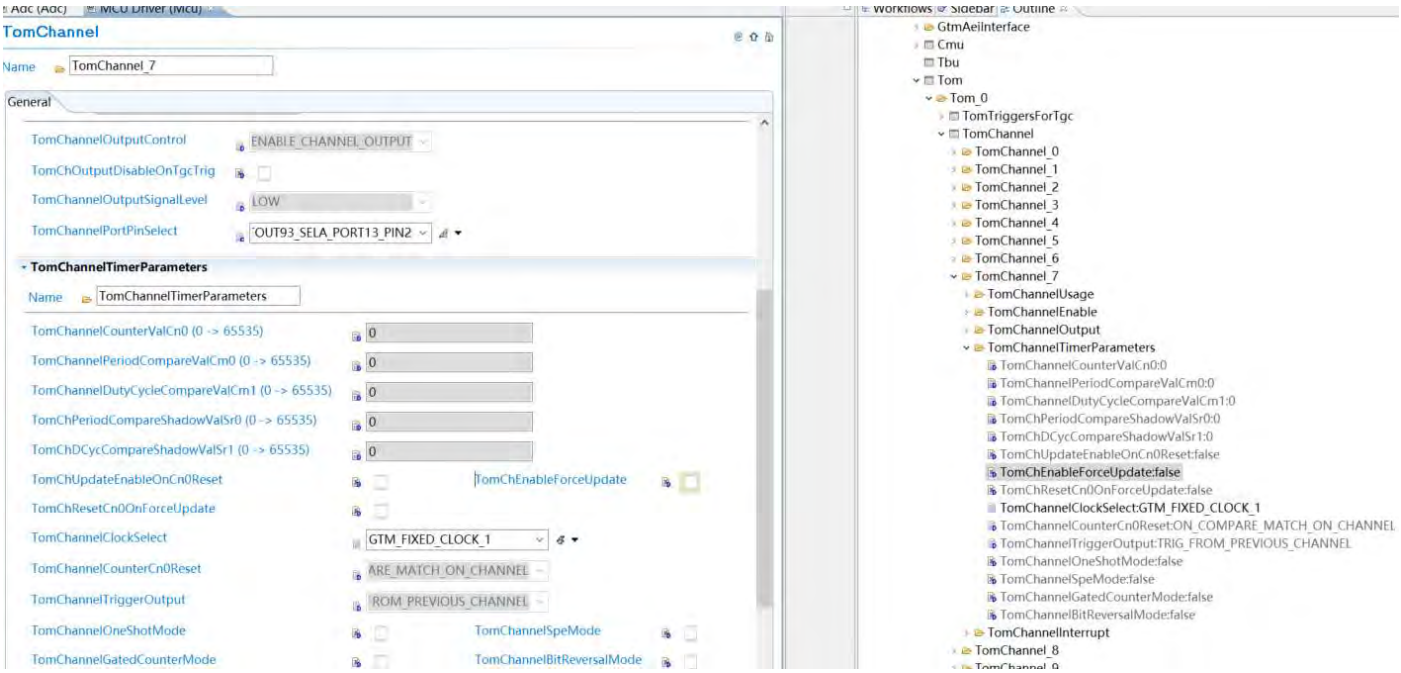
```c
Pwm_17_Gtm_Init(&Pwm_ConfigRoot[0]);
/* Enable Notification*/
Pwm_17_Gtm_EnableNotification(Pwm_17_GtmConf_PwmChannel_PwmChannel_0_Reference,PWM_FALLING_EDGE);

/* Adc Initialize */
Adc_Init(&Adc_ConfigRoot[0]);
```

5.2 硬件触发 ADC（PWM：TOMOCH7）

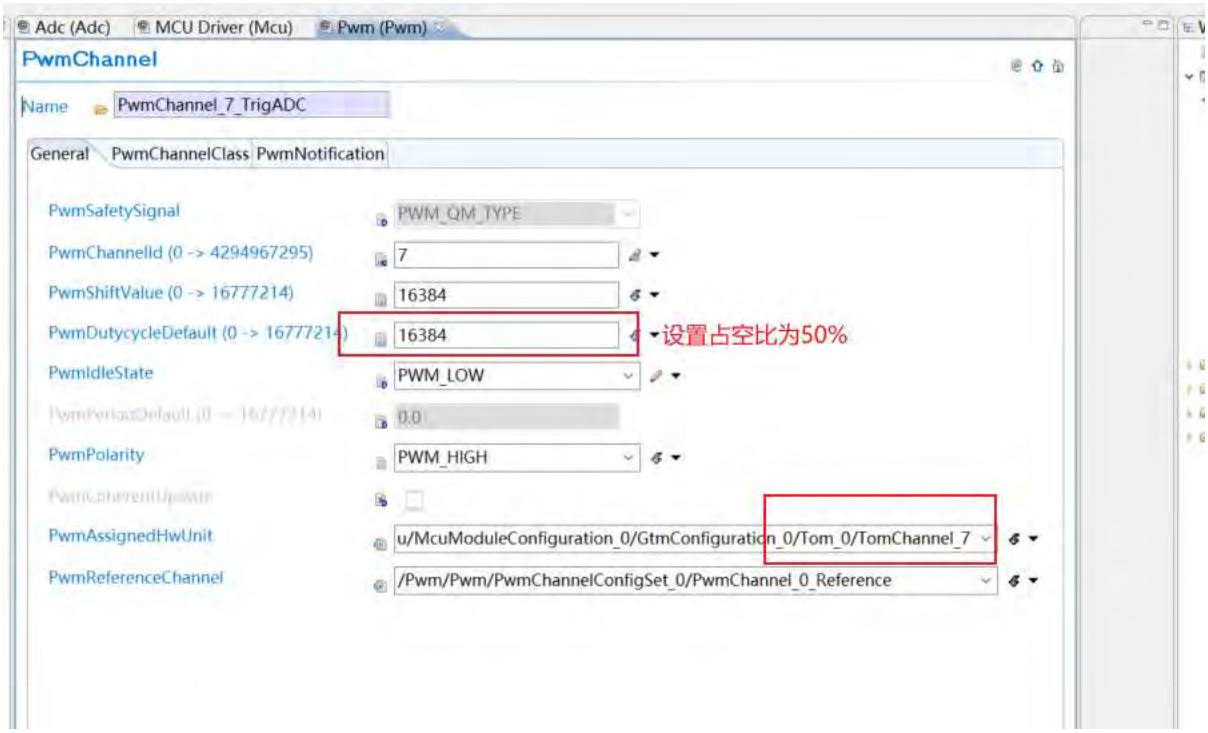
Mcu 配置：触发源配置



TOMOCH7 的配置与 PWM 配置同

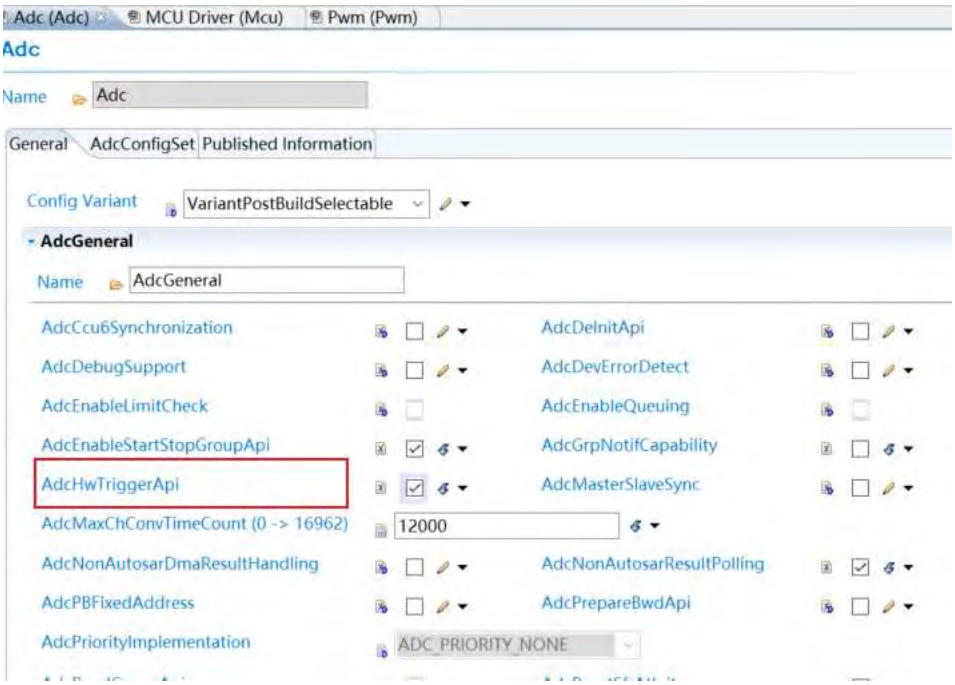


PWM 的配置

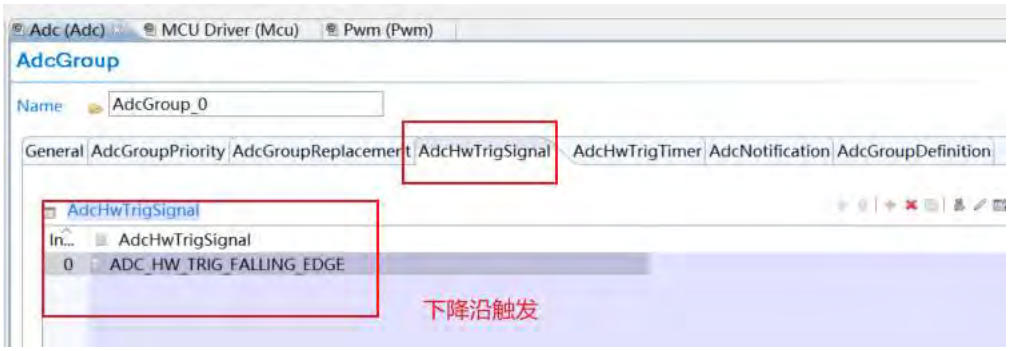


ADC 配置

Api 勾选



只讲和软件触发不一样的，相同的地方：略



Hightec 调用：

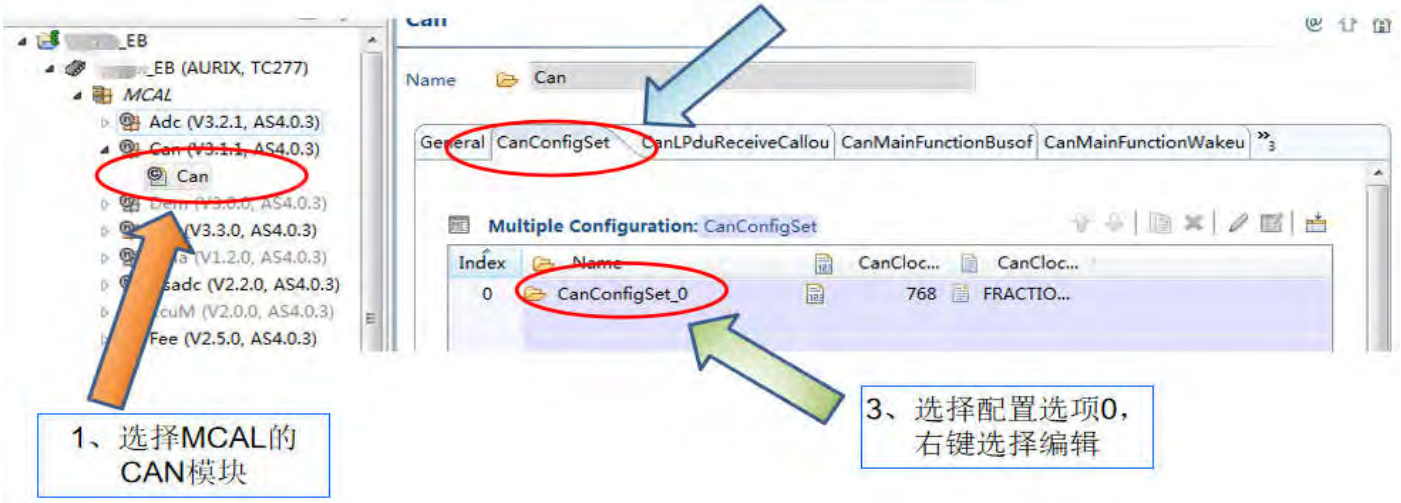
初始化

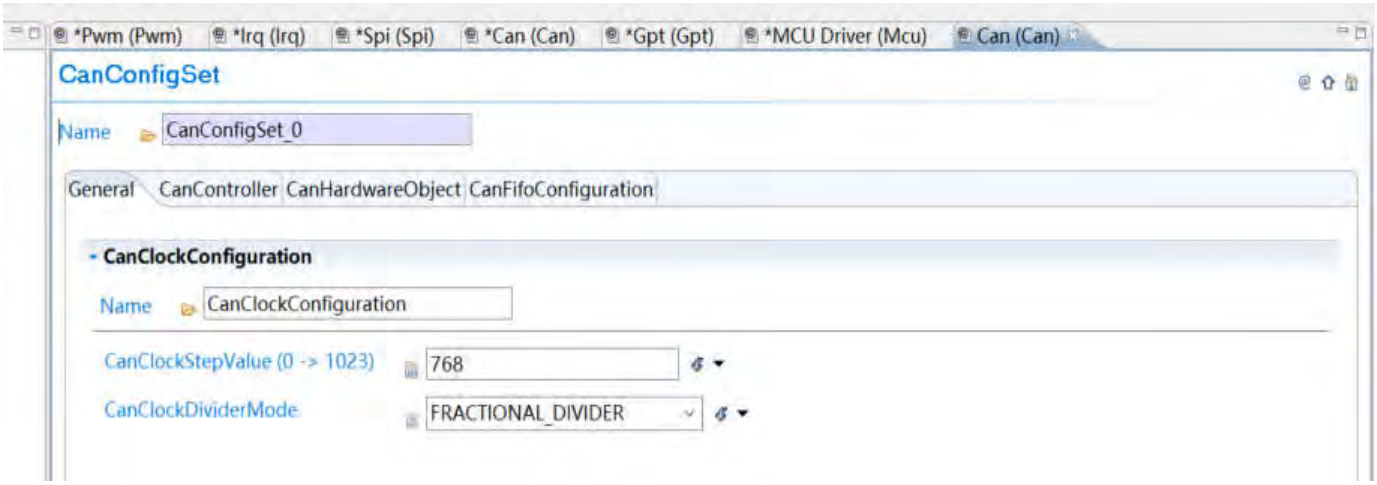
```
/* Pwm Initialize */
Pwm_17_Gtm_Init(&Pwm_ConfigRoot[0]);
/* Enable Notification*/
Pwm_17_Gtm_EnableNotification(Pwm_17_GtmConf_PwmChannel_PwmChannel_0_Reference,PWM_FALLING_EDGE);

/* Adc Initialize */
Adc_Init(&Adc_ConfigRoot[0]);

/* CAN Initialize */
Can_17_MCanP_Init(&Can_17_MCanP_ConfigRoot[0]);
IrqCan_Init();
Can_17_MCanP_EnableControllerInterrupts(Can_17_MCanPConf_CanController_CanController_0);
SRC_CAN_CAN0_INT6.B.SRE = 1;
```

```
void ADC_TestHardWare(void)
{
    Adc_EnableHardwareTrigger(AdcConf_AdcGroup_AdcGroup_0);
    Adc_17_GetGroupResult(AdcConf_AdcGroup_AdcGroup_0,HardWareADCResult);
}
```

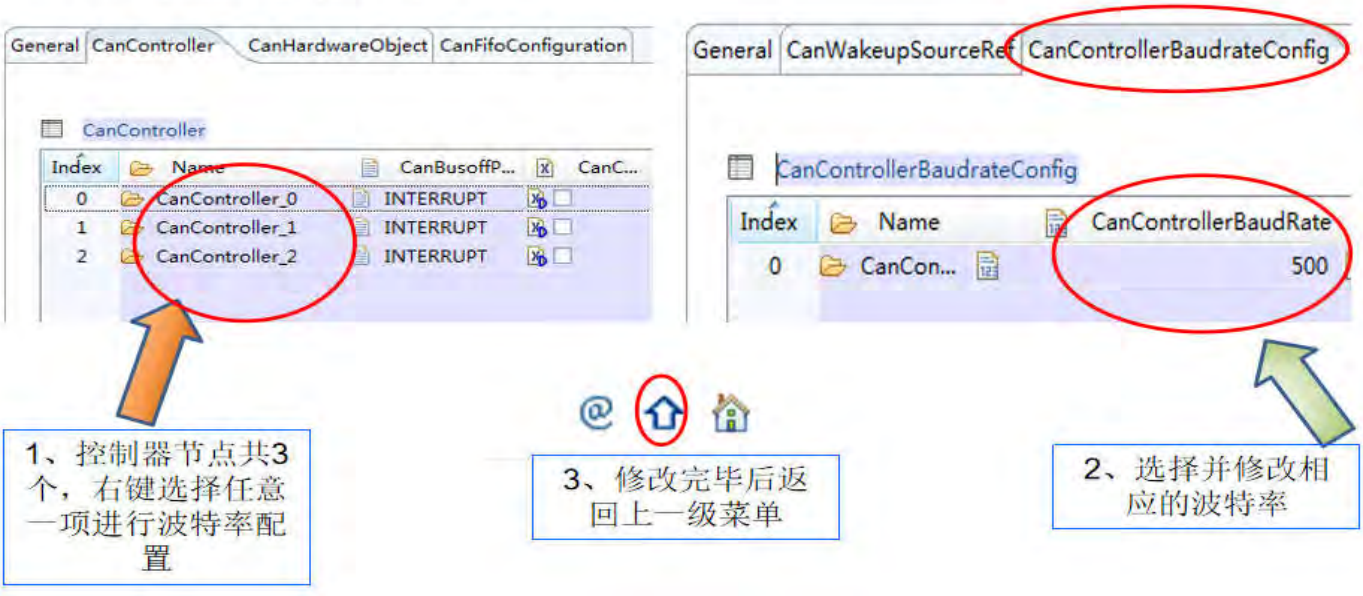
六. CAN 的配置（TX 轮询、RX 中断）

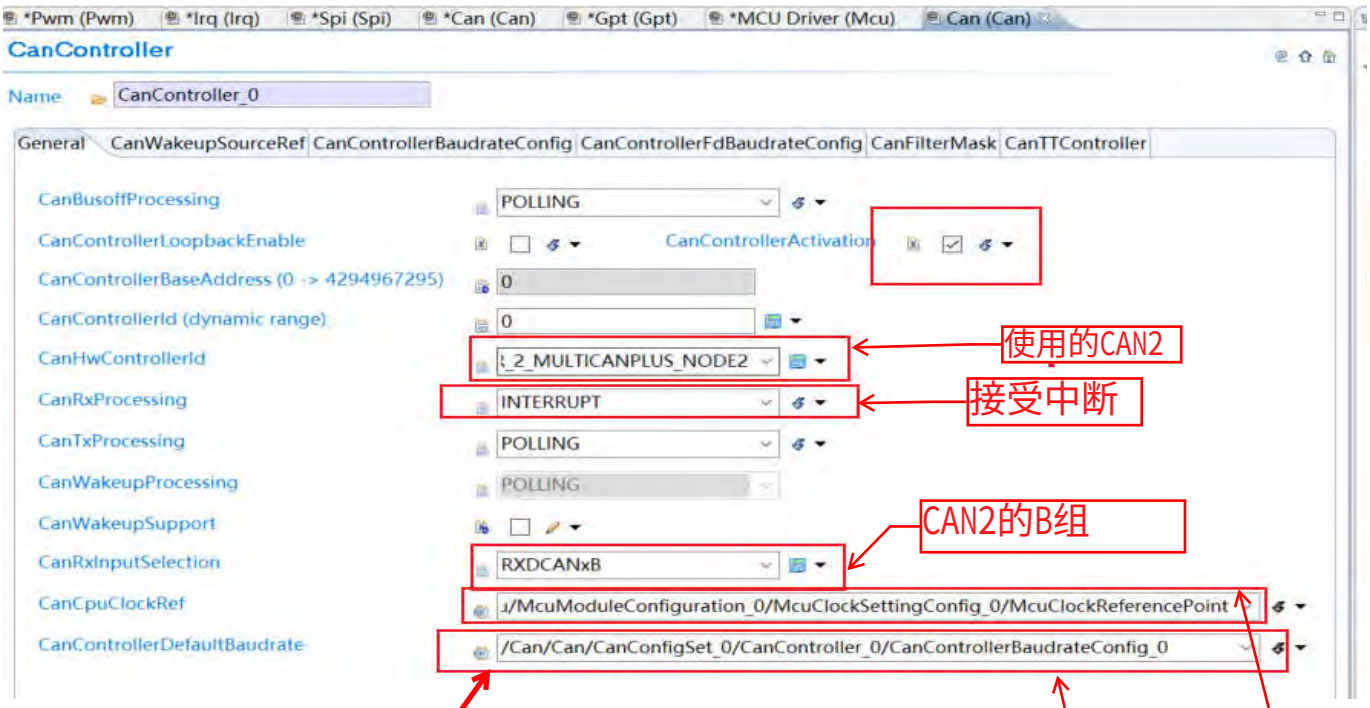
Tresos配置

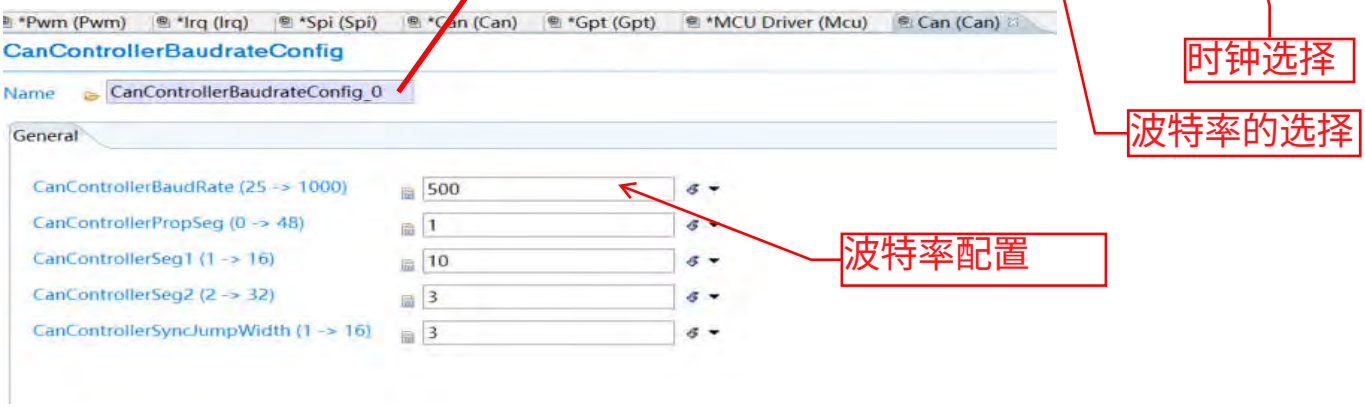


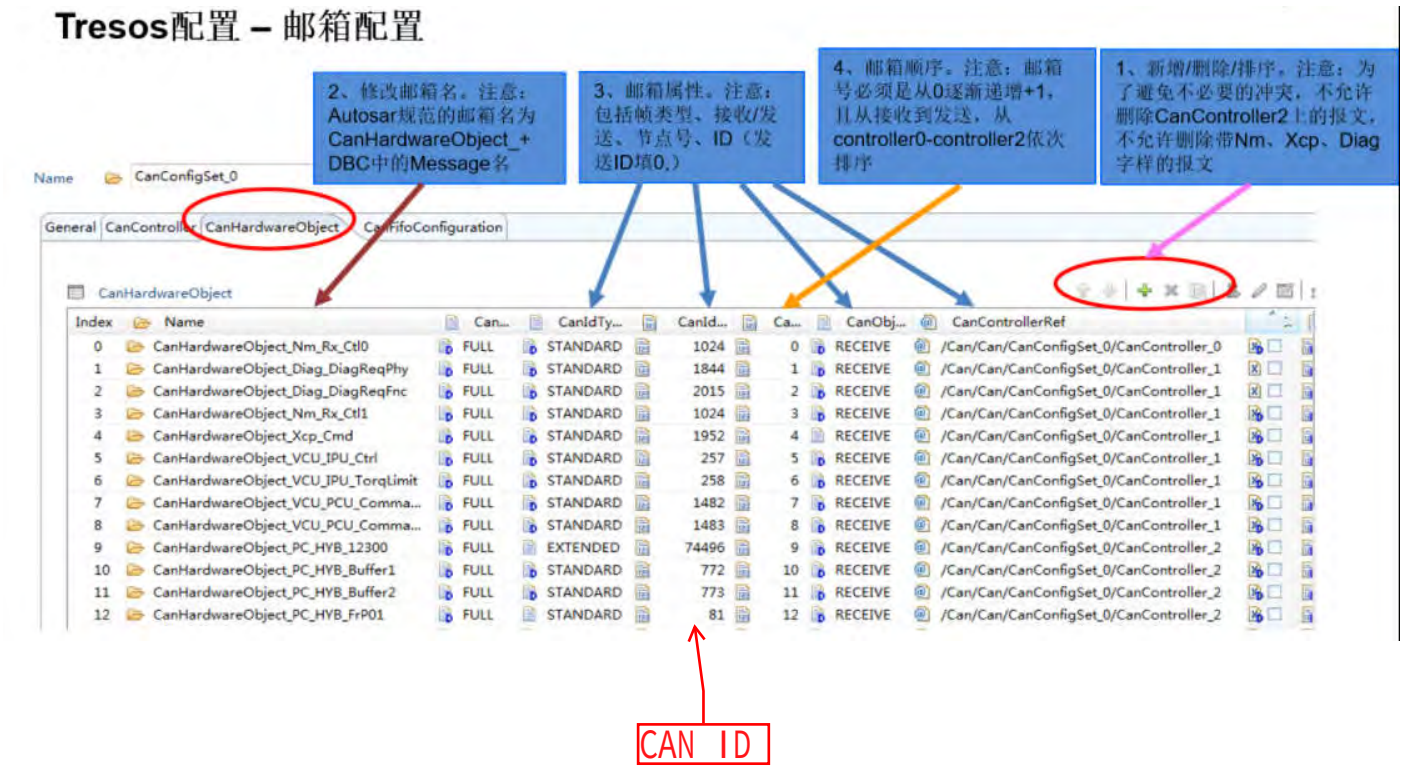


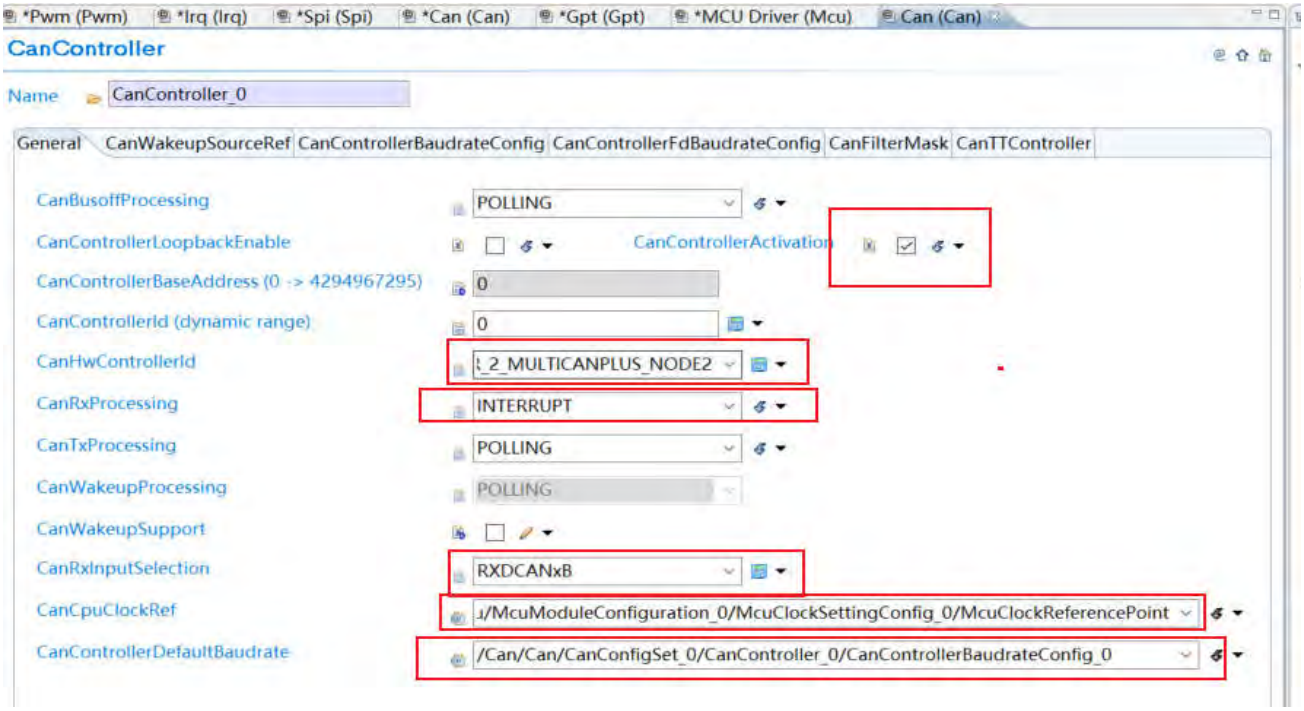
Tresos配置 – 波特率设置



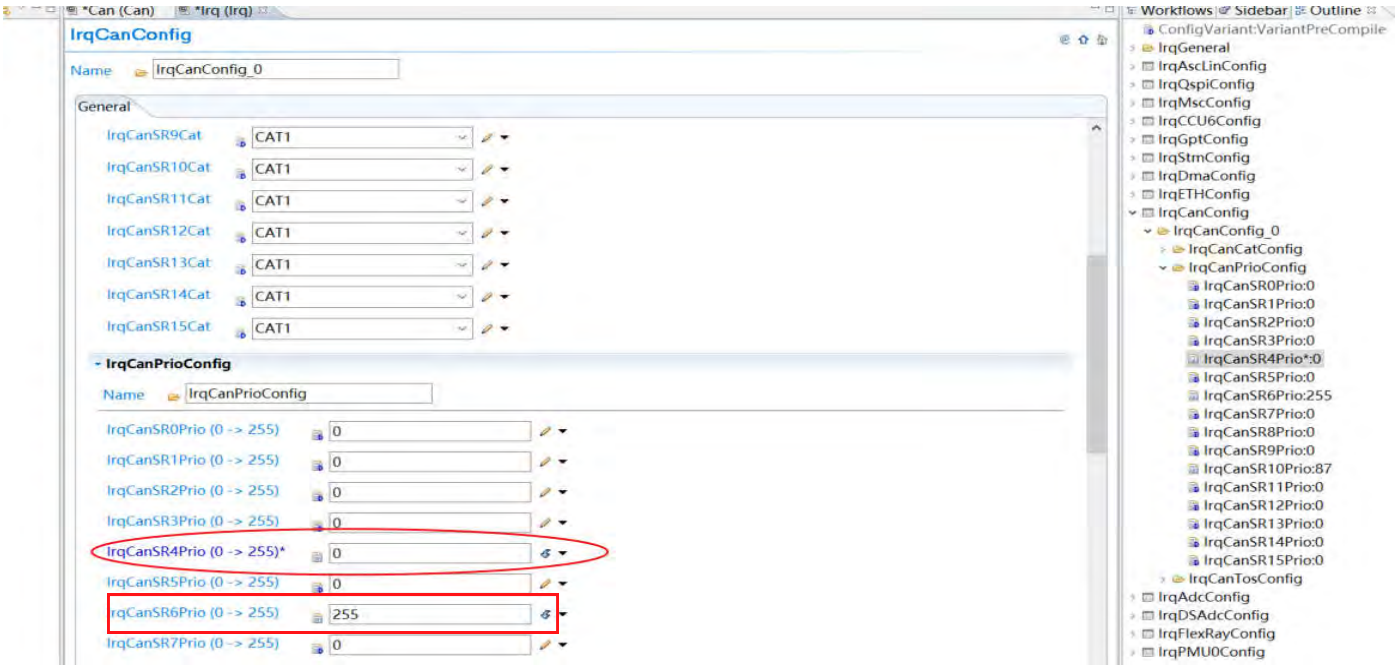

*Pwm (Pwm)
*Irq (Irq)
*Spi (Spi)
*Can (Can)
*Gpt (Gpt)
*MCU Driver (Mcu)
Can (Can)
CanController
Name
CanController_0
General
CanWakeupSourceRef
CanControllerBaudrateConfig
CanControllerFdBaudrateConfig
CanFilterMask
CanTTController
CanBusoffProcessing
POLLING
CanControllerLoopbackEnable
CanControllerActivation
CanControllerBaseAddress (0 -> 4294967295)
0
CanControllerId (dynamic range)
0
CanHwControllerId
_2_MULTICANPLUS_NODE2
CanRxProcessing
INTERRUPT
CanTxProcessing
POLLING
CanWakeupProcessing
POLLING
CanWakeupSupport
CanRxInputSelection
RXDCANxB
CanCpuClockRef
/McuModuleConfiguration_0/McuClockSettingConfig_0/McuClockReferencePoint
CanControllerDefaultBaudrate
/Can/Can/CanConfigSet_0/CanController_0/CanControllerBaudrateConfig_0
使用的CAN2
接受中断
CAN2的B组
时钟选择
波特率的选择


*Pwm (Pwm)
*Irq (Irq)
*Spi (Spi)
*Can (Can)
*Gpt (Gpt)
*MCU Driver (Mcu)
Can (Can)
CanControllerBaudrateConfig
Name
CanControllerBaudrateConfig_0
General
CanControllerBaudRate (25 -> 1000)
500
CanControllerPropSeg (0 -> 48)
1
CanControllerSeg1 (1 -> 16)
10
CanControllerSeg2 (2 -> 32)
3
CanControllerSyncJumpWidth (1 -> 16)
3
波特率配置


Tresos配置 – 邮箱配置
2、修改邮箱名。注意：Autosar规范的邮箱名为CanHardwareObject_+DBC中的Message名
3、邮箱属性。注意：包括帧类型、接收/发送、节点号、ID（发送ID填0.）
4、邮箱顺序。注意：邮箱号必须是从0逐渐递增+1，且从接收到发送，从controller0-controller2依次排序
1、新增/删除/排序。注意：为了避免不必要的冲突，不允许删除CanController2上的报文，不允许删除带Nm、Xcp、Diag字样的报文
Name
CanConfigSet_0
General
CanController
CanHardwareObject
CanFifoConfiguration
CanHardwareObject
Index
Name
Can...
CanIdTy...
CanId...
Ca...
CanObj...
CanControllerRef
0 CanHardwareObject_Nm_Rx_Ctl0 FULL STANDARD 1024 0 RECEIVE /Can/Can/CanConfigSet_0/CanController_0
1 CanHardwareObject_Diag_DiagReqPhy FULL STANDARD 1844 1 RECEIVE /Can/Can/CanConfigSet_0/CanController_1
2 CanHardwareObject_Diag_DiagReqFnc FULL STANDARD 2015 2 RECEIVE /Can/Can/CanConfigSet_0/CanController_1
3 CanHardwareObject_Nm_Rx_Ctl1 FULL STANDARD 1024 3 RECEIVE /Can/Can/CanConfigSet_0/CanController_1
4 CanHardwareObject_Xcp_Cmd FULL STANDARD 1952 4 RECEIVE /Can/Can/CanConfigSet_0/CanController_1
5 CanHardwareObject_VCU_IPU_Ctrl FULL STANDARD 257 5 RECEIVE /Can/Can/CanConfigSet_0/CanController_1
6 CanHardwareObject_VCU_IPU_TorqLimit FULL STANDARD 258 6 RECEIVE /Can/Can/CanConfigSet_0/CanController_1
7 CanHardwareObject_VCU_PCU_Comma... FULL STANDARD 1482 7 RECEIVE /Can/Can/CanConfigSet_0/CanController_1
8 CanHardwareObject_VCU_PCU_Comma... FULL STANDARD 1483 8 RECEIVE /Can/Can/CanConfigSet_0/CanController_1
9 CanHardwareObject_PC_HYB_12300 FULL EXTENDED 74496 9 RECEIVE /Can/Can/CanConfigSet_0/CanController_2
10 CanHardwareObject_PC_HYB_Buffer1 FULL STANDARD 772 10 RECEIVE /Can/Can/CanConfigSet_0/CanController_2
11 CanHardwareObject_PC_HYB_Buffer2 FULL STANDARD 773 11 RECEIVE /Can/Can/CanConfigSet_0/CanController_2
12 CanHardwareObject_PC_HYB_FrP01 FULL STANDARD 81 12 RECEIVE /Can/Can/CanConfigSet_0/CanController_2
CAN ID



优先级配置



HighTec

回调函数是 AUTOSAR 自定的，在 CanIf_Cbk.c 文件当中

```
Adc_Init(&Adc_ConfigRoot[0]);

 /* CAN Initialize  */
 Can_17_MCanP_Init(&Can_17_MCanP_ConfigRoot[0]);
 IrqCan_Init();
 Can_17_MCanP_EnableControllerInterrupts(Can_17_MCanPConf_CanController_CanController_0);
 SRC_CAN_CAN0_INT6.B.SRE = 1;
/*Enable CAN*/
Can_17_MCanP_SetControllerMode(Can_17_MCanPConf_CanController_CanController_0, CAN_T_START);

Mcal_EnableAllInterrupts();//开全局中断

PWM_00_9_Duty=((PWM_00_9_DutyCycle*((0x8000u)/5000.0)));
```

函数初始化

中断初始化

使能cancontroller0中断

can开启

```
#include "Pwm_17_Gtm.h"
#include "Spi.h"

uint16 PWM_00_9_DutyCycle = 2000;
uint16 PWM_00_9_Duty=0;
Adc_ValueGroupType SoftWareADCResult[2];
uint8 Test_RxConfirmCount;
uint8 Test_TxConfirmCount;

uint8 Buffer_TX0[8],Buffer_TX1[8];
uint8 Buffer_RX0[8] ;
Can_PduType TestPduInfo[ ] =
{
    {8,8, 0x520, Buffer_RX0 },
    {8,8, 0x400, Buffer_TX0 },
    {8,8, 0x600, Buffer_TX1 },

},

void CAN_Test(void)
{
    uint8 i;
    /*-----------------test the can   ---------------------*/
    for(i = 0; i < 8; i++)
          {
        Buffer_TX0[i]= i+2;
        Buffer_TX1[i]= i+3;
           }
    Can_17_MCanP_Write(Can_17_MCanPConf_CanHardwareObject_CanHardwareObject_PDU_370h, &TestPduInfo[1]);
    Can_17_MCanP_Write(Can_17_MCanPConf_CanHardwareObject_CanHardwareObject_PDU_258h, &TestPduInfo[2]);

}

void ADC_TestSoftWare(void)
{
    Adc_StartGroupConversion(AdcConf_AdcGroup_AdcGroup_0);
    Adc_17_GetGroupResult(AdcConf_AdcGroup_AdcGroup_0,SoftWareADCResult);
}

void GTM_TOM0CH0(void)
{
    Dio_FlipChannel(DioConf_DioPort_DioPort0_LED2);

    j++;
    if(j>500)
    {
        CAN_Test();
        Can_17_MCanP_MainFunction_Write();
        j=0;
    }
```

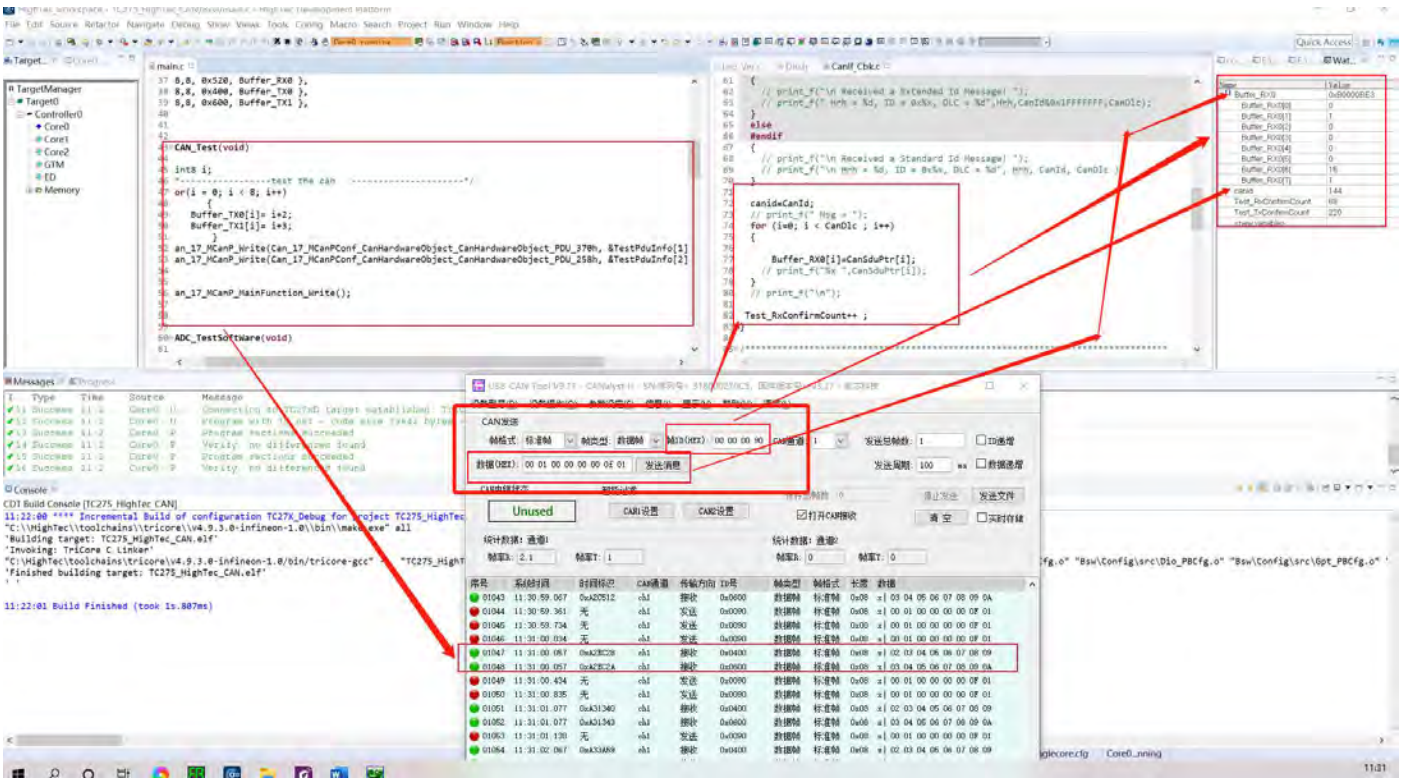
发送

中断线判定

1. 从程序中找，Can_Irq.c 当中：例如 can2 接收中断在 321 行处，可以判定为 SR6
2. 从 Aurix_MC-ISAR_UM_CANDriver.pdf 找，在 87 页，同样可以判定 SR6

回调函数是 AUTOSAR 自定的，在 CanIf_Cbk.c 文件当中

特别注意：当报文ID为扩展帧的时候，接收与发送的首位为 1，
发送时：与上0X80000000；例如： 0x1DFFFF60|0x80000000,
接收时：或上0X1fffffff; eg:canid=CanId&0x1fffffff;

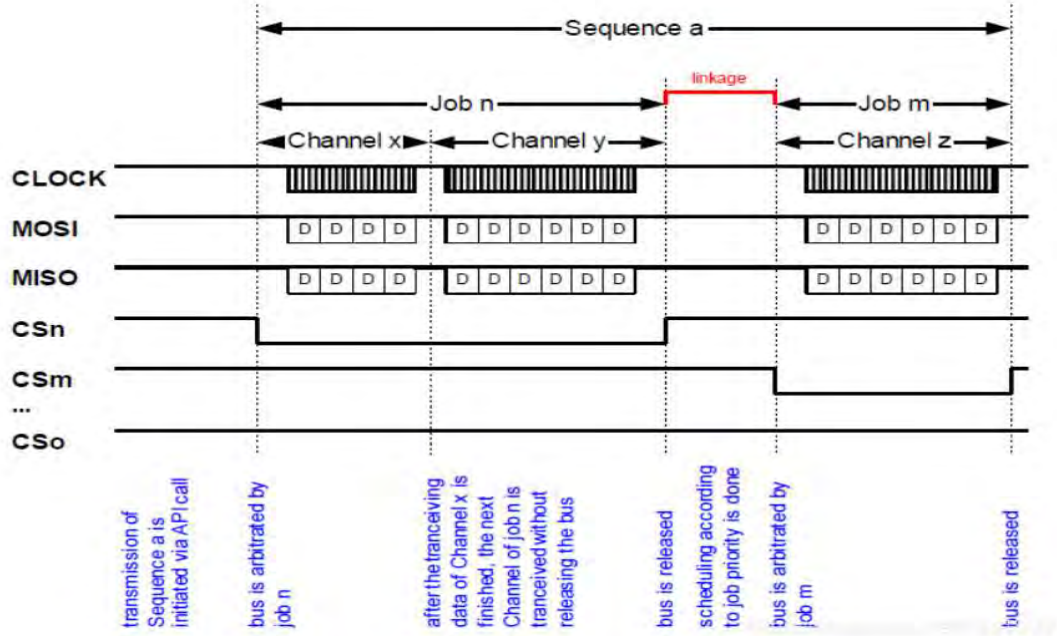


七. SPI 配置

Autosar SPI 概述

- **基础知识介绍**
  在AutoSar标准中，与SPI通讯相关的三个术语：Channel、Job和Sequence。
  1个Channel对应1个发送缓冲区和1接收缓冲区；1个Job对应着1次SPI通讯发送的内容（既SPI 一次片选过程所传输的内容）。1个Sequence 对应着1个SPI通讯序列（job序列）。多个Job可以分配给一个Sequence。
  关于每个术语的详细解释，参考AutoSar标准。SPI通讯是基于Sequence触发的，即使发送1个Job也要将该Job分配给1个队列，然后通过触发Sequence来实现Job的传输。

- **两种使用方法**
  方法1： 对应1个SPI外设芯片，分配1个Job、1个Sequence。使用此种方法，触发一次Sequence，只能传输一个Job，当对外设发送多个Job时，需要多次触发Sequence，且在下一次触发Sequence时，必须确保上一次Sequence已经传输完毕，否则下一次Sequence传输会因为上一次Sequence传输占用SPI总线而失败。
  方法2： 对应1个SPI外设芯片，结合对外设芯片的控制方法，分配多个Job和多个Sequence，有目的分配Job到相应的Sequence中。在不同控制逻辑中，触发不同的Sequence，传输不同个数的Job序列。当对1个Sequence分配多Job时，触发此Sequence就可以完成多个Job的传输，SPI 驱动本身来保证Job序列的传输，不会产生下一个Job传输因为上一个Job占用SPI总线而失败的情况。对外设传输多个Job时，使用方法2可以降低CPU使用率。



- **Channel、Job、Sequence**
  AUTOSAR 提供了三种SPI的抽象。Channel Job Sequence。
  一张图可以说明三者的关系。

- **同步调用与异步调用**
  AUTOSAR 为SPI 提供了两种方式通信的接口。
  同步调用 Spi_SyncTransmit
  异步调用 Spi_AsyncTransmit
  同步调用，是需要等待返回调用结果，而异步调用这是发起任务，一般可以通过回调函数来告知调用结果。

数据类型定义

- Channel配置包含：
  1. EB/IB buffer的使用情况
  2. 传输位宽 (1~32bits)
  3. 传输数据个数
  4. 传输字节序LSB/MSB
  5. 传输的默认值

- Job配置包含：
  1. 使用哪个SPI硬件实例
  2. 使用该实例的哪个片选引脚cs
  3. 片选功能是否启用
  4. 片选高/低有效
  5. 波特率
  6. clock 和 chip select之间的时间
  7. 时钟在空闲时是高/低
  8. 数据传输在上升沿/下降沿
  9. 优先级（低0~3高）
  10. Job完成的通知函数
  11. MCU相关的属性（可选）
  12. Channels（至少1个）
- sequence配置包含：
  1. Jobs (至少1个)
  2. 在每个Job完成后是否产生中断
  3. Sequence完成的通知函数

部分API函数

void Spi_Init(const Spi_ConfigType* ConfigPtr)

1. 初始化所有ConfigPtr相关寄存器
2. 定义ConfigPtr相关的默认值
3. 设置SPI状态为“IDLE”，job、sequence的状态设为“ok”
4. 对于Level2，异步模式设为polling，禁用spi相关中断

Std_ReturnType Spi_DeInit(void)

1. 如果驱动状态不是BUSY，将spi外设置为复位上电后的状态
2. 如果驱动状态是BUSY，请求被拒绝
3. 去初始化后，模块的状态是UINIT

Std_ReturnType Spi_WriteIB(Spi_ChannelType Channel, const Spi_DataBufferType* DataBufferPtr)

1. 返回值：E_OK：该命令被接受，E_NOT_OK：该命令被拒绝。
2. 将入参数据写入对应channel的内部缓存
3. 如果入参DataBufferPtr为NULL，将传输默认值

Std_ReturnType Spi_ReadIB(Spi_ChannelType Channel, Spi_DataBufferType* DataBufferPointer)

1. 返回值：E_OK：该命令被接受，E_NOT_OK：该命令被拒绝。
2. 同步读取一条或多条数据
3. 应该在Transmit后调用，以获取相关数据

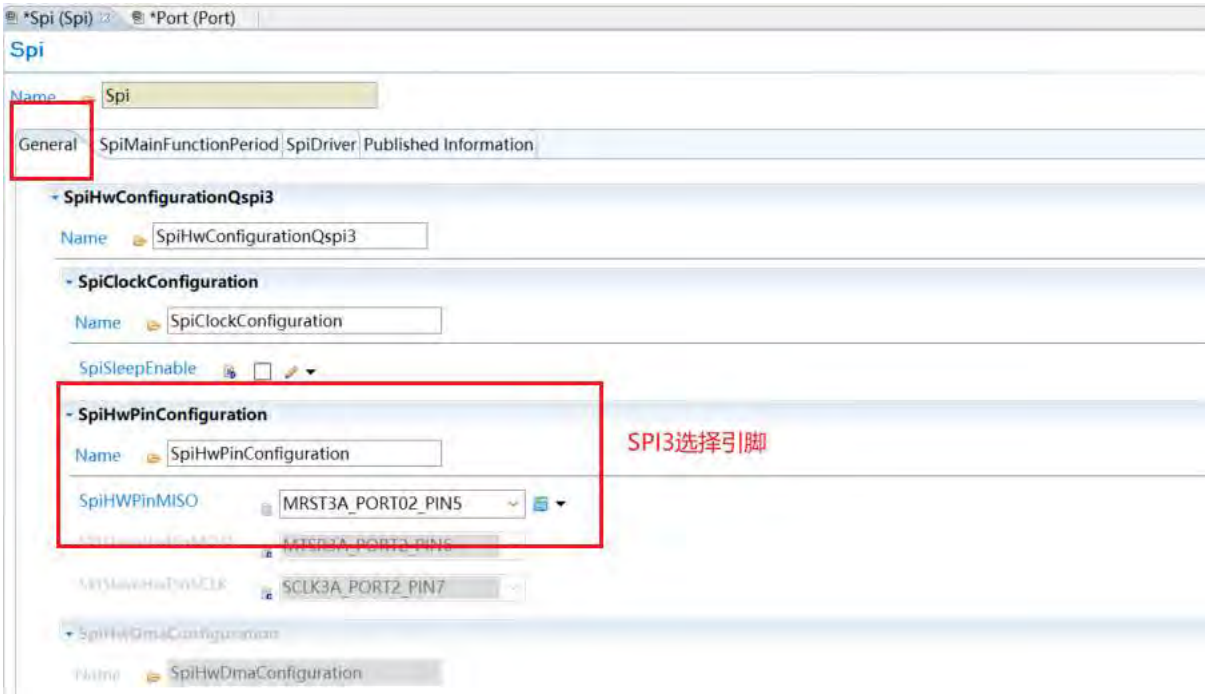
Std_ReturnType Spi_SyncTransmit(Spi_SequenceType Sequence)

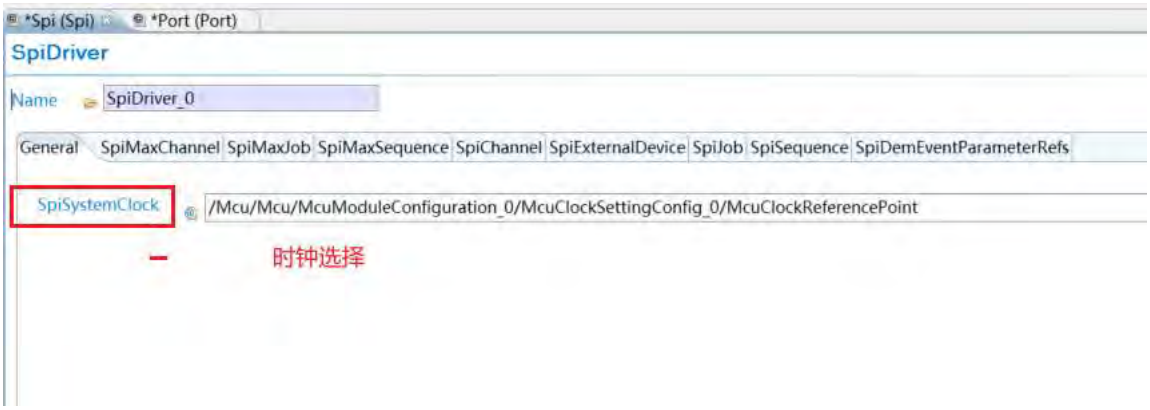
1. 返回值：E_OK：该命令被接受，E_NOT_OK：该命令被拒绝。
2. 在SPI总线上传输数据
3. 驱动状态设为“BUSY”
4. Sequence、Job状态设为“PENDING”
5. 此时如果有其他Sequence正在传输，应该返回 E_NOT_OK，并上报错误。

Spi_StatusType Spi_GetStatus(void)

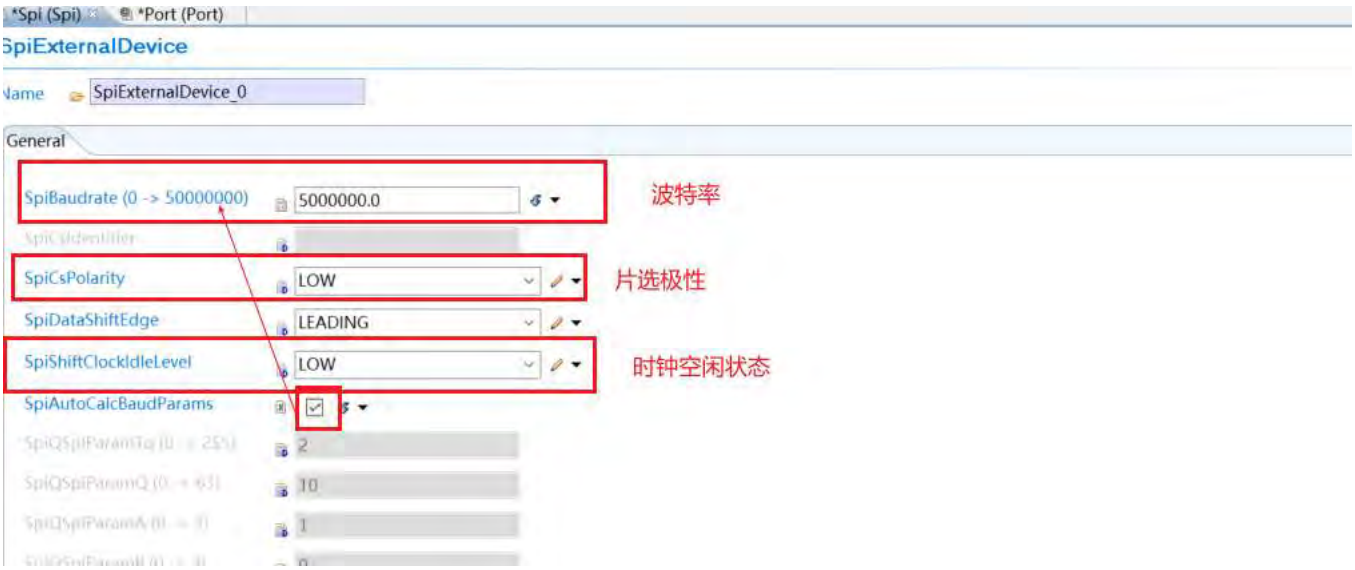
1. 获取驱动状态

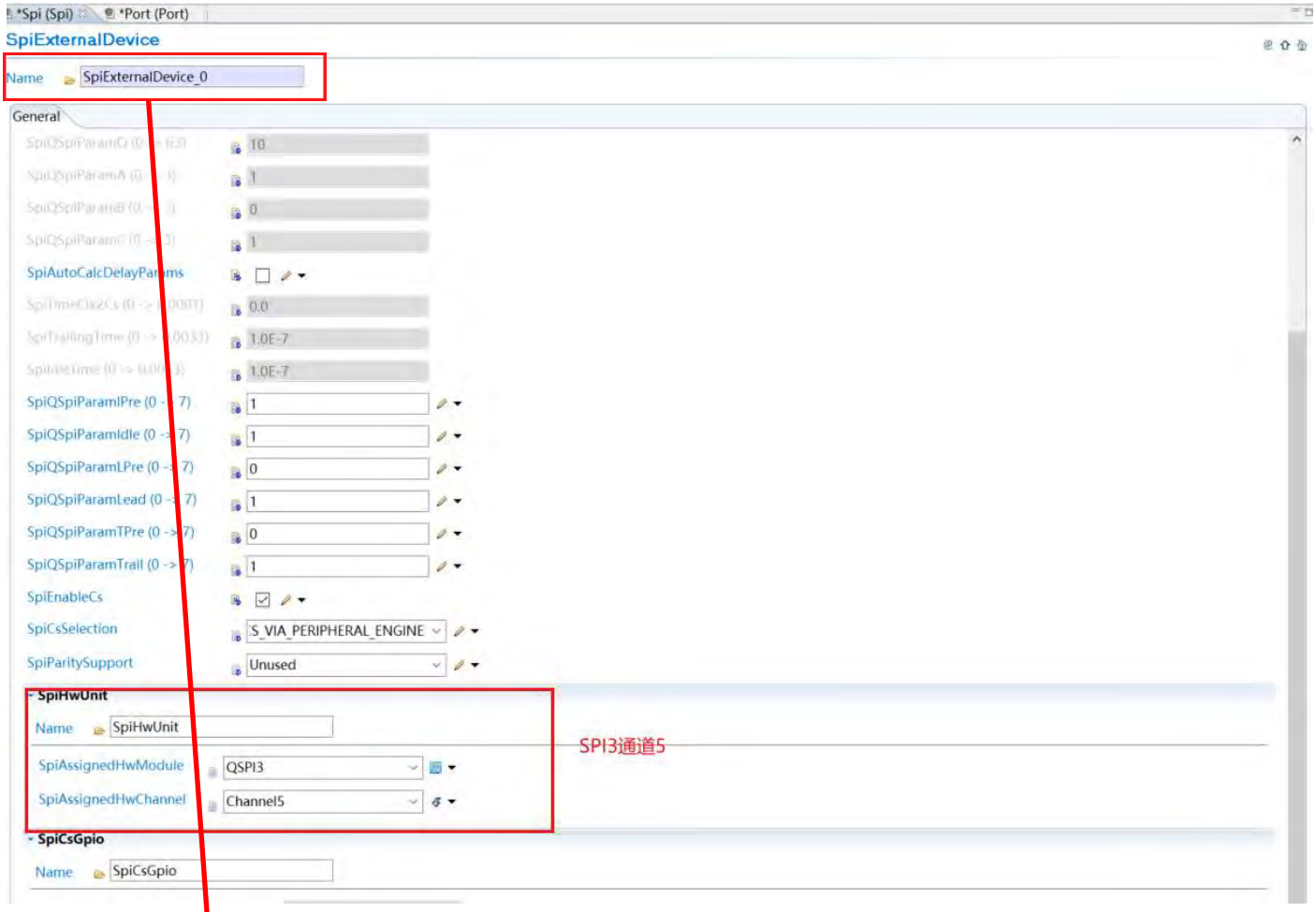
2、EB 的配置

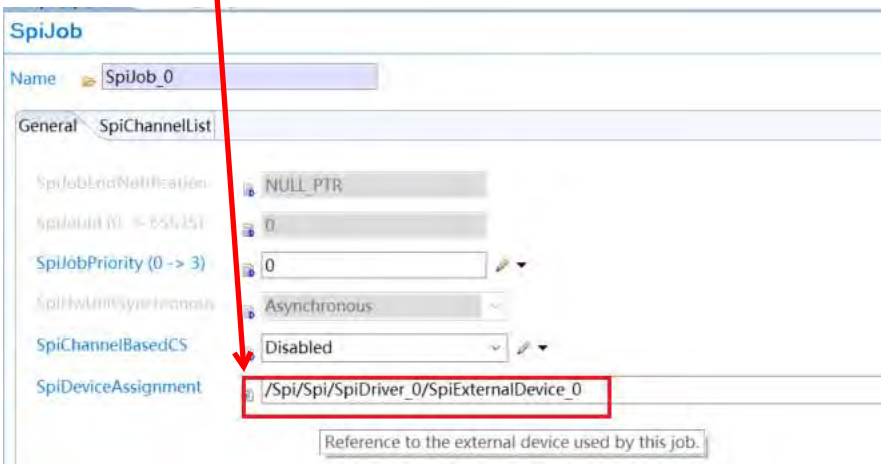

*Spi (Spi)
*Port (Port)
Spi
Name
Spi
General
SpiMainFunctionPeriod
SpiDriver
Published Information
SpiHwConfigurationQspi3
Name
SpiHwConfigurationQspi3
SpiClockConfiguration
Name
SpiClockConfiguration
SpiSleepEnable
SpiHwPinConfiguration
Name
SpiHwPinConfiguration
SpiHWPinMISO
MRST3A_PORT02_PIN5
SCLK3A_PORT2_PIN7
SpiHwDmaConfiguration
SPI3选择引脚

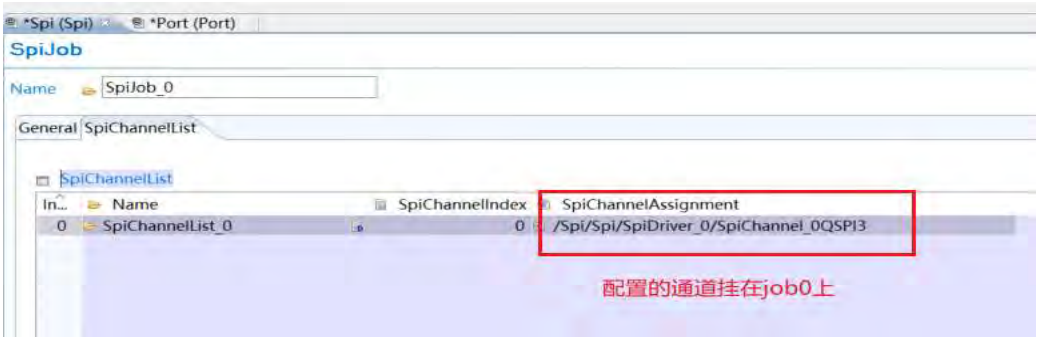

*Spi (Spi)
*Port (Port)
SpiDriver
Name
SpiDriver_0
General
SpiMaxChannel
SpiMaxJob
SpiMaxSequence
SpiChannel
SpiExternalDevice
SpiJob
SpiSequence
SpiDemEventParameterRefs
SpiSystemClock
/Mcu/Mcu/McuModuleConfiguration_0/McuClockSettingConfig_0/McuClockReferencePoint
时钟选择

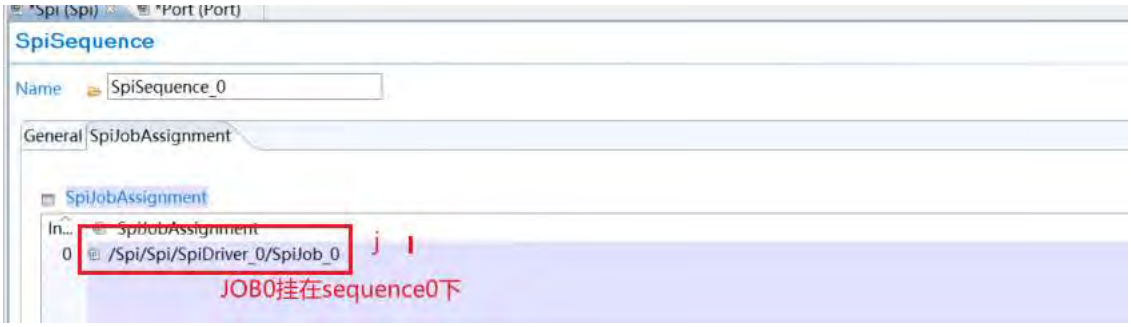

*Spi (Spi)
*Port (Port)
SpiChannel
Name*
SpiChannel_0QSPI3
General
SpiChannelType
EB
SpiDataWidth (2 -> 32)
16
SpiDefaultData (0 -> 4294967295)
0
SpiEbMaxLength (1 -> 65535)
2
10
SpiTransferStart
MSB
0xa5
外部缓冲还是内部缓冲
数据长度：注意程序当中注意默认为8位，
要改数据位类型
数据为方式


*Spi (Spi)
*Port (Port)
SpiExternalDevice
Name
SpiExternalDevice_0
General
SpiBaudrate (0 -> 50000000)
5000000.0
波特率
SpiCsPolarity
LOW
片选极性
SpiDataShiftEdge
LEADING
SpiShiftClockIdleLevel
LOW
时钟空闲状态
SpiAutoCalcBaudParams
2
10
1


*Spi (Spi)
*Port (Port)
SpiExternalDevice
Name
SpiExternalDevice_0
General
10
1
0
1
SpiAutoCalcDelayParams
0.0
1.0E-7
1.0E-7
SpiQSpiParamIPre (0 -
1
SpiQSpiParamIdle (0 -
1
SpiQSpiParamLPre (0 -
0
SpiQSpiParamLead (0 -
1
SpiQSpiParamTPre (0 -
0
SpiQSpiParamTrail (0 -
1
SpiEnableCs
SpiCsSelection
S_VIA_PERIPHERAL_ENGINE
SpiParitySupport
Unused
SpiHwUnit
Name
SpiHwUnit
SpiAssignedHwModule
QSPI3
SpiAssignedHwChannel
Channel5
SPI3通道5
SpiCsGpio
Name
SpiCsGpio


SpiJob
Name
SpiJob_0
General
SpiChannelList
NULL_PTR
0
SpiJobPriority (0 -> 3)
0
Asynchronous
SpiChannelBasedCS
Disabled
SpiDeviceAssignment
/Spi/Spi/SpiDriver_0/SpiExternalDevice_0
Reference to the external device used by this job.





3、 PORT 引脚复用

QSPI Module	MRST(MISO) Master Input	MTSR (MOSI) Slave Input	SCLK Slave Clock	ChipSelect
				SLSO2-9: P20.3
QSPI3	MRST3A: P02.5	MTSR3: P01.6	SCLK3: P01.7	SLSO3-0: P02.4
	MRST3B: P10.7	MTSR3: P02.6	SCLK3: P02.7	SLSO3-1: P02.0
	MRST3C: P01.5	MTSR3: P10.6	SCLK3: P10.8	SLSO3-10: P01.4
				SLSO3-2: P02.1
				SLSO3-3: P00.5
				SLSO3-3: P02.2
				SLSO3-4: P00.2
				SLSO3-4: P02.3
				SLSO3-5: P02.12
				SLSO3-5: P02.8
				SLSO3-6: P00.8
				SLSO3-6: P02.15
				SLSO3-7: P00.9
				SLSO3-7: P02.13
				SLSO3-8: P10.5
				SLSO3-9: P01.3
QSPI4	MRST4A: P33.13	MTSR4: P33.12	SCLK4: P33.3	SLSO4-0: P33.10

SPI3 CH5

P02.5	I	General-purpose input	P02_IN.P5	P02_IOCR4. PC5	$0XXXX_B$
		GTM input	TIN5		
		QSPI3 input	MRST3A		
		PSI5 input	PSIRX1B		

P02.6	I	General-purpose input	P02_IN.P6	P02_IOCR4.PC6	$0XXXX_B$
		GTM input	TIN6		
		QSPI3 input	MTSR3A		
		SENT input	SENT2C		
		CCU60 input	CC60INC		
		CCU60 input	CCPOS0A		
		CCU61 input	T12HRB		
		GPT120 input	T3INA		
		DSADC input	DSDIN4B		
		CIF input	CIFD6		
		DSADC input	DSITR5E		
	O	General-purpose output	P02_OUT.P6		$1X000_B$
		GTM output	TOUT6		$1X001_B$
		PSI5-S output	PSISTX		$1X010_B$
		QSPI3 output	MTSR3		$1X011_B$
		PSI5 output	PSITX1		$1X100_B$
P02.7	I	General-purpose input	P02_IN.P7	P02_IOCR4.PC7	$0XXXX_B$
		GTM input	TIN7		
		QSPI3 input	SCLK3A		
		PSI5 input	PSIRX2B		
		SENT input	SENT1C		
		CCU60 input	CC61INC		
		CCU60 input	CCPOS1A		
		CCU61 input	T13HRB		
		GPT120 input	T3EUDA		
		DSADC input	DSCIN3B		
		CIF input	CIFD7		
		DSADC input	DSITR4E		
	O	General-purpose output	P02_OUT.P7		$1X000_B$
		GTM output	TOUT7		$1X001_B$
		Reserved	–		$1X010_B$
		QSPI3 output	SCLK3		$1X011_B$
		DSADC output	DSCOUT3		$1X100_B$

P02.8	I	General-purpose input	P02_IN.P8	P02_IOCR8.PC8	$0XXXX_B$
		GTM input	TIN8		
		SENT input	SENT0C		
		CCU60 input	CC62INC		
		CCU60 input	CCPOS2A		
		CCU61 input	T12HRC		
		CCU61 input	T13HRC		
		GPT120 input	T4INA		
		DSADC input	DSDIN3B		
		CIF input	CIFD8		
		DSADC input	DSITR3E		
	O	General-purpose output	P02_OUT.P8		$1X000_B$
		GTM output	TOUT8		$1X001_B$
		QSPI3 output	SLSO35		$1X010_B$
		Reserved	–		$1X011_B$

4、 hightec 应用

```
SRC_CAN_CAN0_INT6.B.SRE = 1;
/*Enable CAN*/
Can_17_MCanP_SetControllerMode(Can_17_MCanPConf_CanController_CanC

/* Spi Initialize  */
Spi_Init(&Spi_ConfigRoot[0]);

Mcal_EnableAllInterrupts();//开全局中断

PWM_00_9_Duty=((PWM_00_9_DutyCycle*((0x8000u)/5000.0)));
```

初始化

```
};
Spi_DataType readdata1[2];
Spi_DataType writeCommand1[2] = {0x55,0xAA};

void SPI_Test(void)
{
    Spi_SetupEB(SpiConf_SpiJob_SpiJob_0, writeCommand1, readdata1, 2);
    Spi_SyncTransmit(SpiConf_SpiJob_SpiJob_0);
}
```

默认位uint8类型

写的数据

读的数据

```
/* [cover parentID=DS_AS_SPI149_SPI289_SPI290_SPI_PR4518_SPI164_SPI355] */
/*
Type : Spi_DataType
Type of application data buffer elements.
In order to always get the same user interface, the SPI Handler/Driver
shall handle these differences using the following proposed rules:
A) Spi_DataType 8/16/32 bits, data width 8/16/32 bits. Straightforward send and
receive.
B) Spi_DataType superior to data width. Send the lower part, ignore the upper
part. Receive the lower part, extend with zero.
C) Spi_DataType inferior to data width. According to the memory alignement use
prior both rules.
*/
typedef uint16 Spi_DataType;
/* [/cover] */

/* [cover parentID=DS_AS_SPI165_SPI_PR4522] */
/*
```

八. ICU 配置

ICU模块功能简介

1. 支持四种类型的ICU通道
- 边沿计数
- 边沿检测
- 信号测量
- 时间戳捕获

2. 可以选择启动测量的边沿

3. 通道类型，过滤器可配置

4. 用户定义相关中断函数

5. 支持硬件源为GTM/CCU6/ERU/GTP12

ICU通道类型特点

1. 边沿计数
- 测量指定边沿的个数

2. 边沿检测
- 检测到指定边沿后产生中断

3. 信号测量
- 输入信号的周期和占空比
- 输入信号高电平时间
- 输入信号低电平时间
- 输入信号周期

4. 时间戳捕获
- 检测到指定边沿后捕获时间戳信号

TIM模块功能简介

1. 有多种工作模式，可以实现对输入信号周期，占空比，边沿个数，边沿类型的检测

2. 有多种滤波模式，可以实现对输入信号的滤波处理

3. 可以在检测到指定边沿或周期数后产生中断

4. 测量后的数据保存在指定寄存器中供用户读取和使用

5. 时钟信号来源于GTM的可配置时钟

TIM模块滤波模式特点

上升沿和下降沿可分别配置滤波模式

以上升沿滤波为例：

1. 即时边沿传播模式

- 检测到上升沿后立即输出，并在指定时间内保持高电平状态

2. 独立抗尖峰脉冲时间模式（上/下计数）

- 检测到上升沿后计数器开始加数计数，检测到下降沿后计数器减数计数，直到计数器达到指定值，则判定该边沿有效并作为滤波后的输出

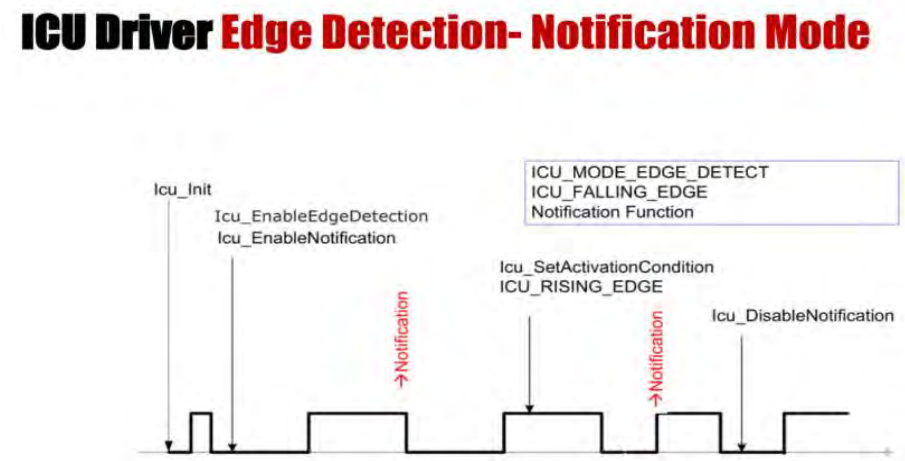
3. 独立抗尖峰脉冲时间模式（计数保持）

- 检测到上升沿后计数器开始加数计数，检测到下降沿后保持当前计数值不变，直到计数器达到指定值，则判定该边沿有效并作为滤波后的输出

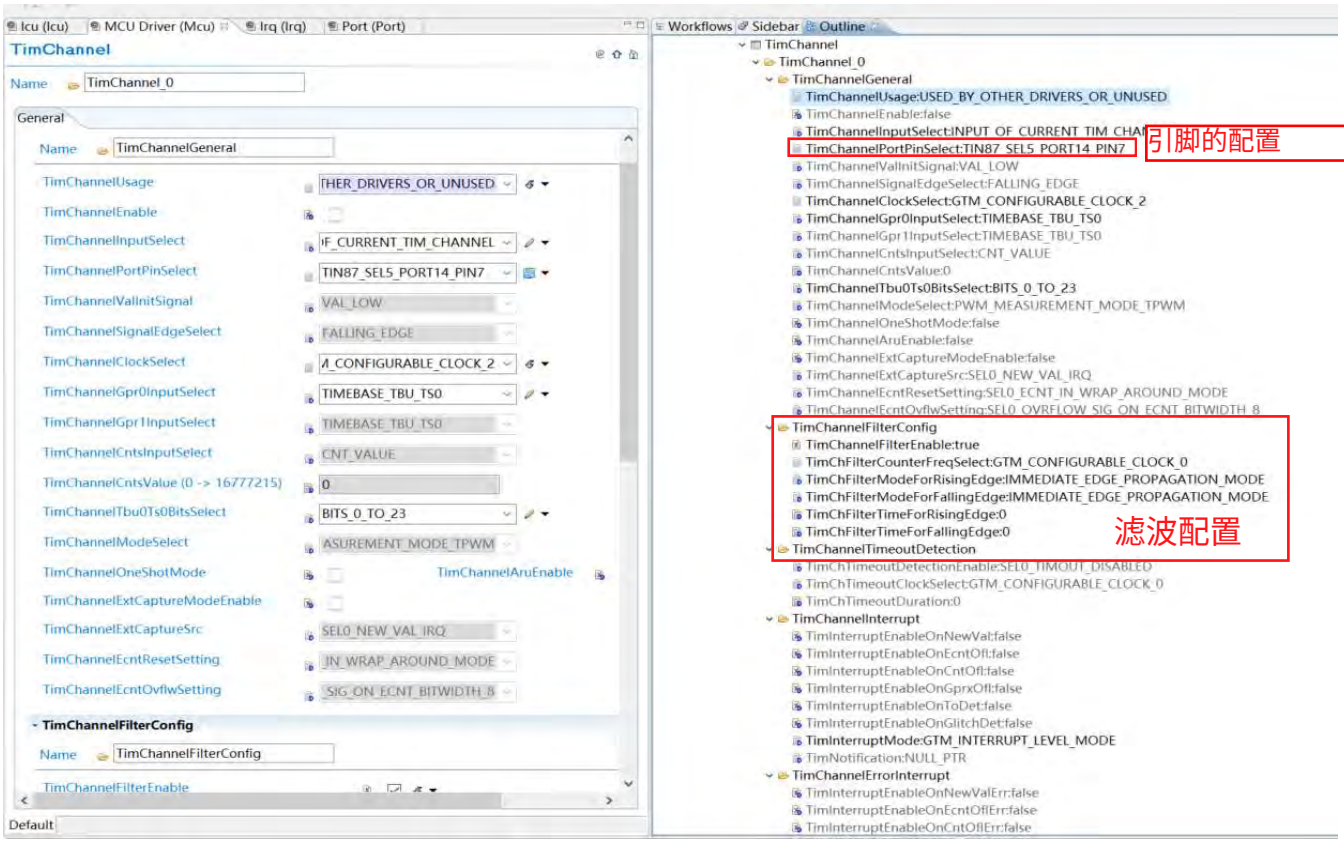
4. 独立抗尖峰脉冲时间模式（计数复位）

- 检测到上升沿后计数器开始加数计数，检测到下降沿后计数器复位为0，直到计数器达到指定值，则判定该边沿有效并作为滤波后的输出

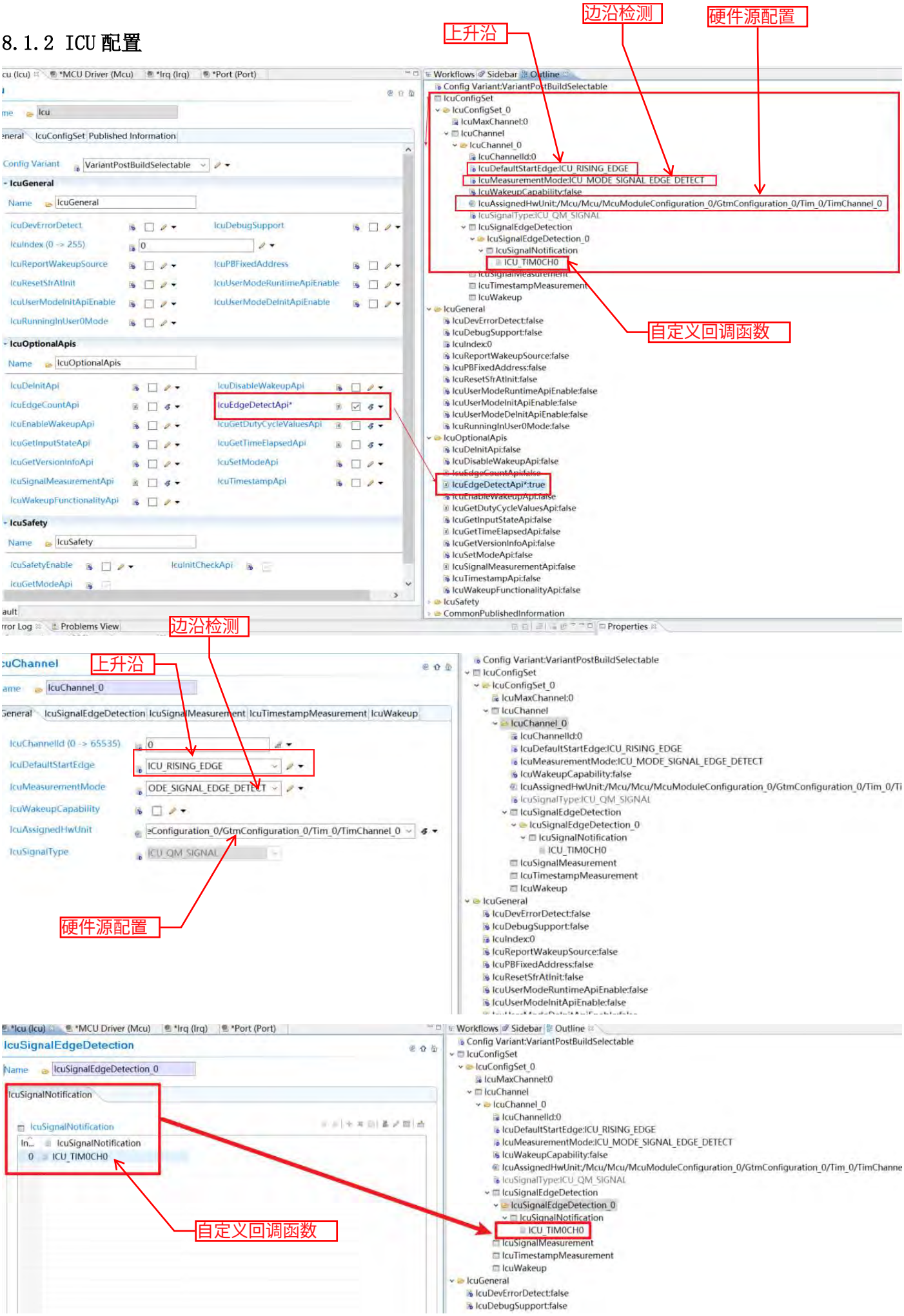
8.1 边沿检测



8.1.1 MCU 时钟配置



8.1.2 ICU 配置


上升沿
边沿检测
硬件源配置
自定义回调函数
IcuDefaultStartEdge:ICU_RISING_EDGE
IcuMeasurementMode:ICU_MODE_SIGNAL_EDGE_DETECT
IcuAssignedHwUnit:/Mcu/Mcu/McuModuleConfiguration_0/GtmConfiguration_0/Tim_0/TimChannel_0
IcuSignalNotification
ICU_TIM0CH0
IcuEdgeDetectApi*:true
ICU_RISING_EDGE
ICU_QM_SIGNAL

8.1.3 IRQ 配置：略

8.1.4：HighTec 调用

```
    Pwm_17_Gtm_EnableNotification(Pwm_17_GtmConf_PwmChannel_PwmChannel_0_Reference,PWM_FALLING_EDGE);

    /* Adc Initialize */
    Adc_Init(&Adc_ConfigRoot[0]);

     /* CAN Initialize  */
     Can_17_MCanP_Init(&Can_17_MCanP_ConfigRoot[0]);
     IrqCan_Init();
     Can_17_MCanP_EnableControllerInterrupts(Can_17_MCanPConf_CanController_CanController_0);
     SRC_CAN_CAN0_INT6.B.SRE = 1;
    /*Enable CAN*/
    Can_17_MCanP_SetControllerMode(Can_17_MCanPConf_CanController_CanController_0, CAN_T_START);

    /* Spi Initialize  */
    Spi_Init(&Spi_ConfigRoot[0]);

    /* Icu Initialization */
    Icu_17_GtmCcu6_Init(&Icu_ConfigRoot[0]);
    Icu_17_GtmCcu6_EnableEdgeDetection(IcuConf_IcuChannel_IcuChannel_0);/* Icu Enable the Edge Detection */
    Icu_17_GtmCcu6_SetActivationCondition(IcuConf_IcuChannel_IcuChannel_0, ICU_FALLING_EDGE);//设置下降沿模式
    Icu_17_GtmCcu6_EnableNotification(IcuConf_IcuChannel_IcuChannel_0);//Enable 回调函数

    Mcal_EnableAllInterrupts();//开全局中断

    while(1)
    {

    }


void ICU_TIM0CH0(void)    回调函数
{
    Dio_FlipChannel(DioConf_DioPort_DioPort0_LED2);
}
```

8.1 信号检测