MCAL应用_CddUart模块配置及实例（一）

1. 版本

Config Tool Version：1.10.0

ME0 MCAL version：1.3.1

2. 前言

该文章主要描述CddUart模块的基础配置及使用

2.1 CddUart 一般依赖项

a. Mcu：包含所需配置的外设时钟

b. Platform：配置外设中断

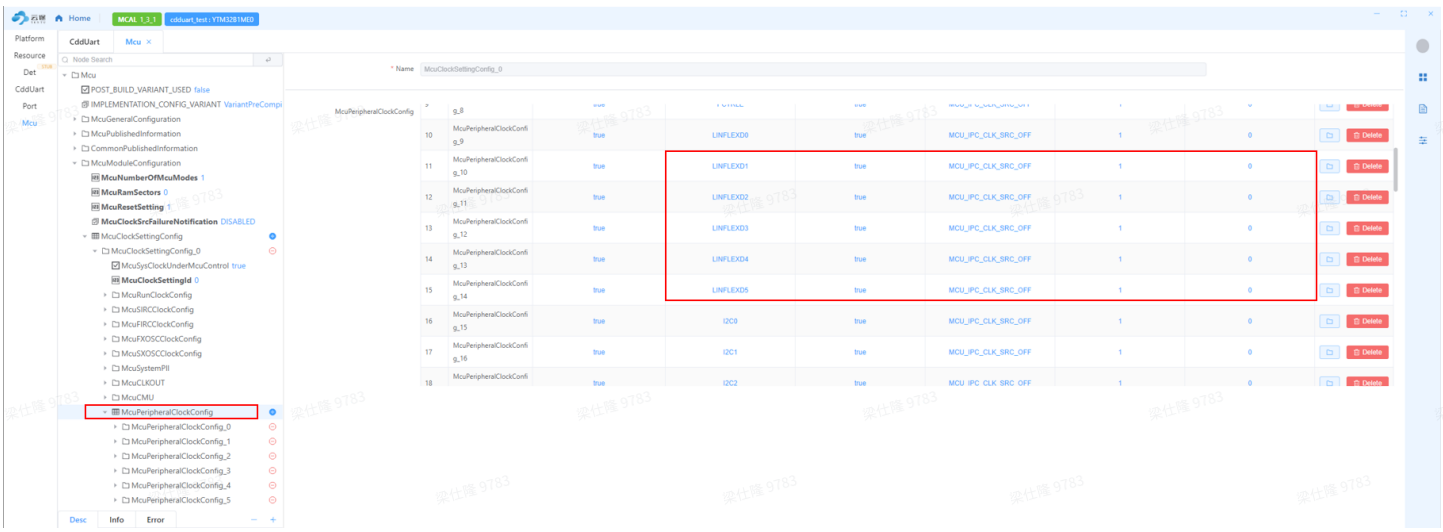
c. Port：配置相关模块所需引脚功能

Note: Mcu和Platform的默认配置参考Setup篇和Mcu篇介绍

3. 基础模块配置介绍

3.1 时钟配置

- 在MCU模块中，找到如图所示的LINFLEXD时钟配置项，在此处可以配置LINFLEXD模块的时钟源和分频系数



Tips:

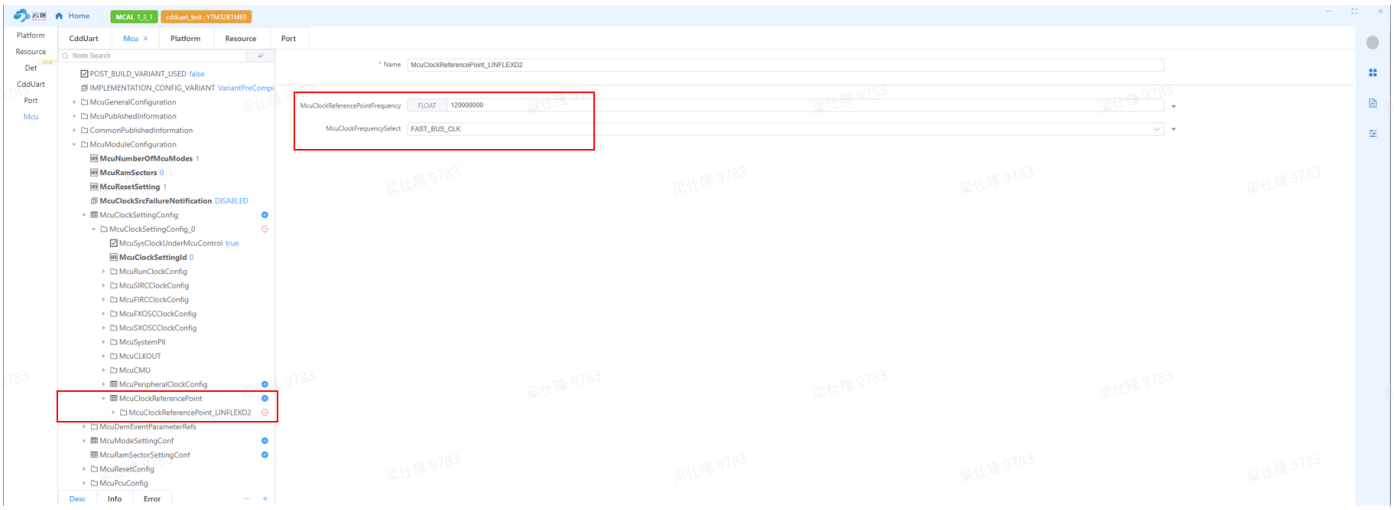
- ME0和MD1系列MCU采用LINFLEXD模块，LIN和UART功能均为该模块实现
- ME0系列LINFLEXD模块不支持配置外设时钟，默认选用Addition Clock中的FAST_BUS_CLK

No.	Module Name	Offset	Bus Clock	Bus Clock Enable	Peripheral Functional Clock	Software Reset	Additional Clock
24	-	0x060	-	-	-	-	-
25	-	0x064	-	-	-	-	-
26	-	0x068	-	-	-	-	-
27	LINFlexD0	0x06C	SLOW_BUS_CLK	OFF	-	YES	FAST_BUS_CLK
28	LINFlexD1	0x070	SLOW_BUS_CLK	OFF	-	YES	FAST_BUS_CLK
29	LINFlexD2	0x074	SLOW_BUS_CLK	OFF	-	YES	FAST_BUS_CLK
30	LINFlexD3	0x078	SLOW_BUS_CLK	OFF	-	YES	FAST_BUS_CLK
31	LINFlexD4	0x07C	SLOW_BUS_CLK	OFF	-	YES	FAST_BUS_CLK
32	LINFlexD5	0x080	SLOW_BUS_CLK	OFF	-	YES	FAST_BUS_CLK

- MD1 系列 LINFLEXD 模块支持外设时钟配置

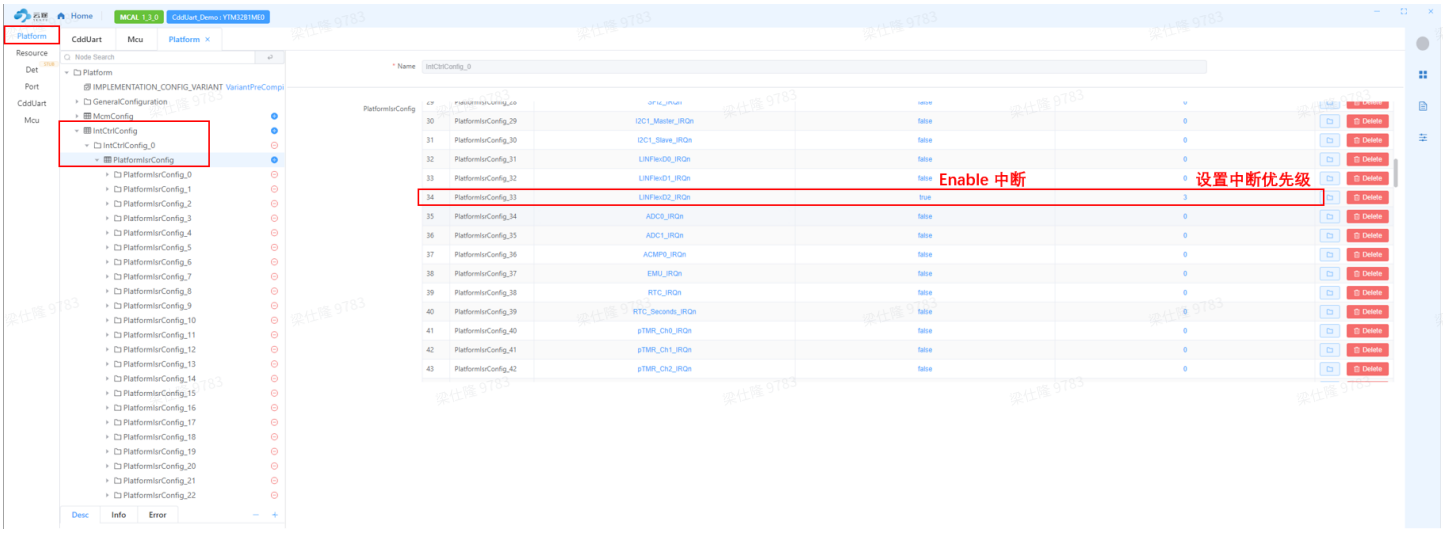
27	LINFlexD0	0x06C	SLOW_BUS_CLK	OFF	YES	YES	-
28	LINFlexD1	0x070	SLOW_BUS_CLK	OFF	YES	YES	-
29	LINFlexD2	0x074	SLOW_BUS_CLK	OFF	YES	YES	-

- 在McuClockReferencePoint中定义一个UART时钟参考频率，这个参考频率可以被其他模块作为输入值使用
  - ME0：不支持外设时钟配置，UART时钟参考频率只能设置为FAST_BUS_CLK
  - MD1：支持外设时钟配置，UART时钟参考频率有多种时钟源可以设置，但是必须和LINFLEXD模块时钟源一致



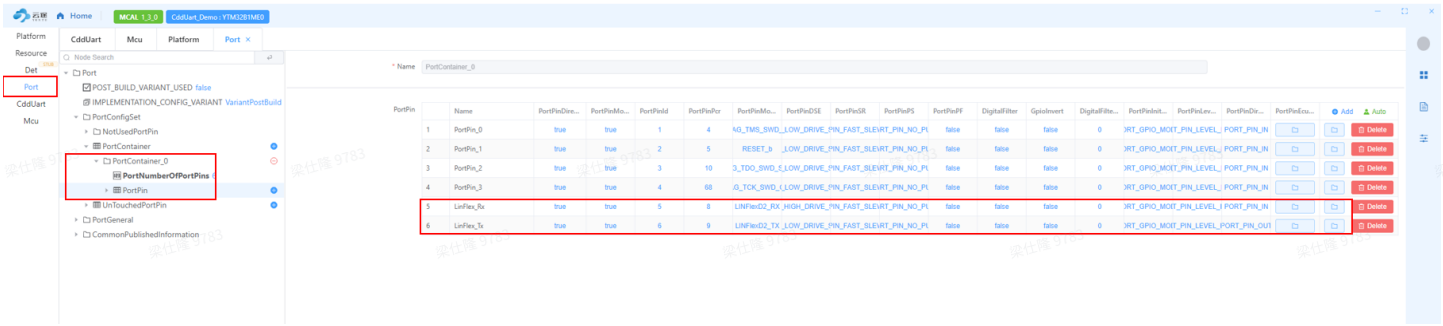
3.2 中断配置

- 在Platform中的中断控制配置项中配置，若使用OS，则中断管理将在OS中配置



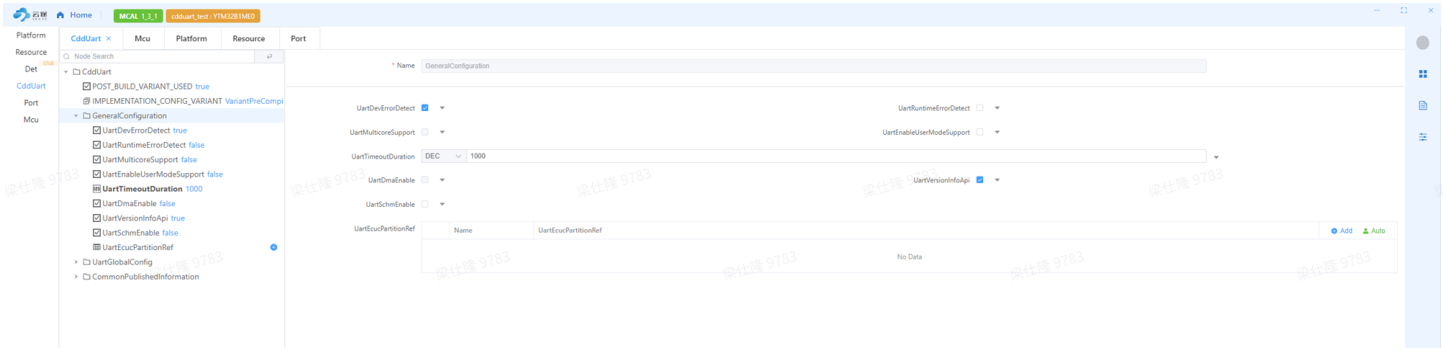
3.3 引脚配置

在Port模块中配置UART Rx和Tx引脚功能



4. CddUart 模块配置介绍

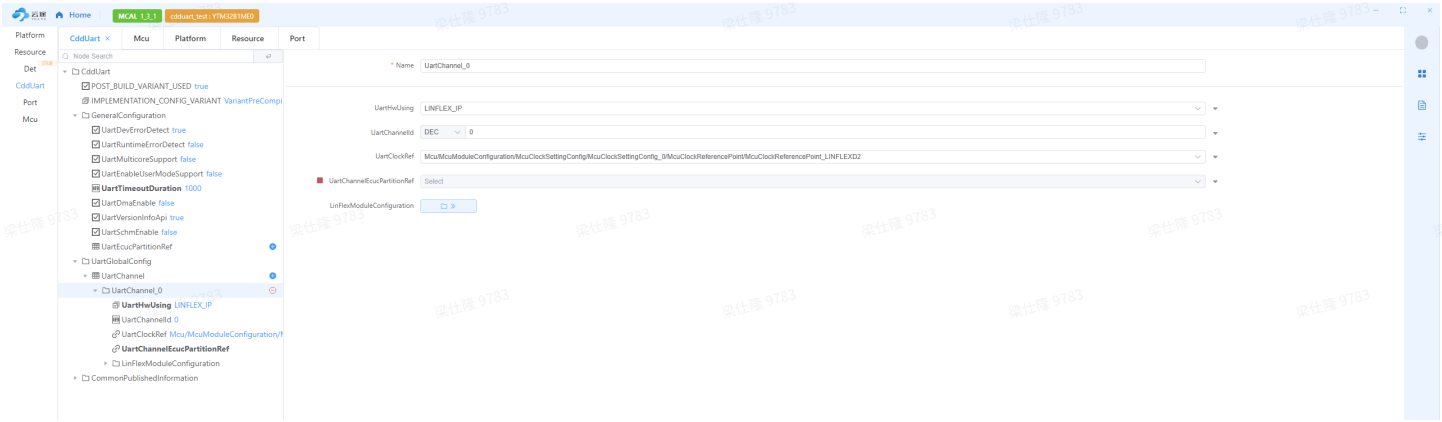
4.1 一般配置（GeneralConfiguration）



- **配置项介绍：**
  - **UartDevErrorDetect：**检测和报告UART模块中可能出现的错误，有助于开发人员更容易地诊断和解决与UART通信相关的问题。
  - **UartMulticoreSupport：**允许多核处理器系统中的不同核心并行使用UART通信，M系列不支持该功能，不可配置，默认disable。
  - **UartTimeoutDuration：**设置超时时间的一个宏，用户自行调用该宏使用。
  - **UartDmaEnable：**ME0和MD1暂不支持，默认disable。

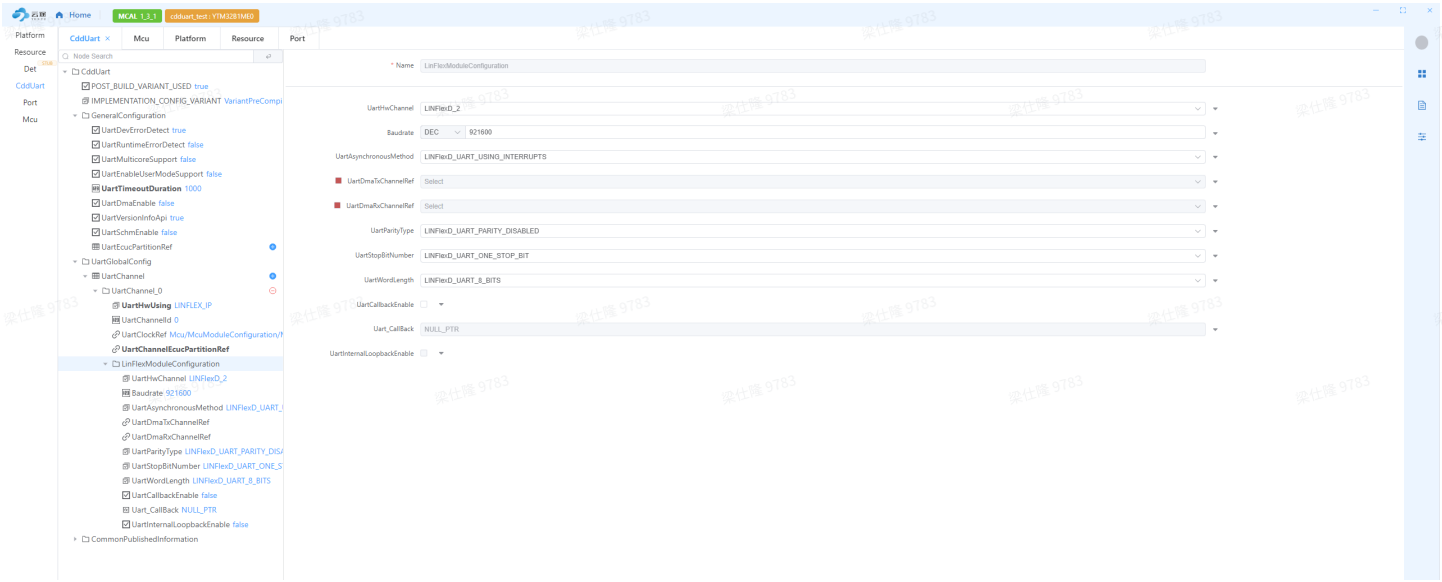
- **UartSchmEnable**：临界保护，进行原子操作。
- **UartRuntimeErrorDetect**：运行时（程序执行期间）检测并处理错误或异常情况的机制
- **UartEnableUserModeSupport**：用户模式是一种权限受限的操作模式，用于限制对系统资源的访问和操作，以提高系统的安全性和稳定性。如果该选项被设置为启用，即受到访问权限的限制。这意味着只有经过授权的应用程序或任务能够访问UART通信模块的某些功能，从而减少潜在的安全风险。用户模式通常是一种较低特权级别，只能执行有限的操作，以提高系统的安全性。
- **UartVersioninfoApi**：是否启用版本查询API。
- **UartEcucPartitionRef**：多核相关配置，M系列为单核，无需配置此项。

4.2 Uart全局配置（UartGlobalConfig）



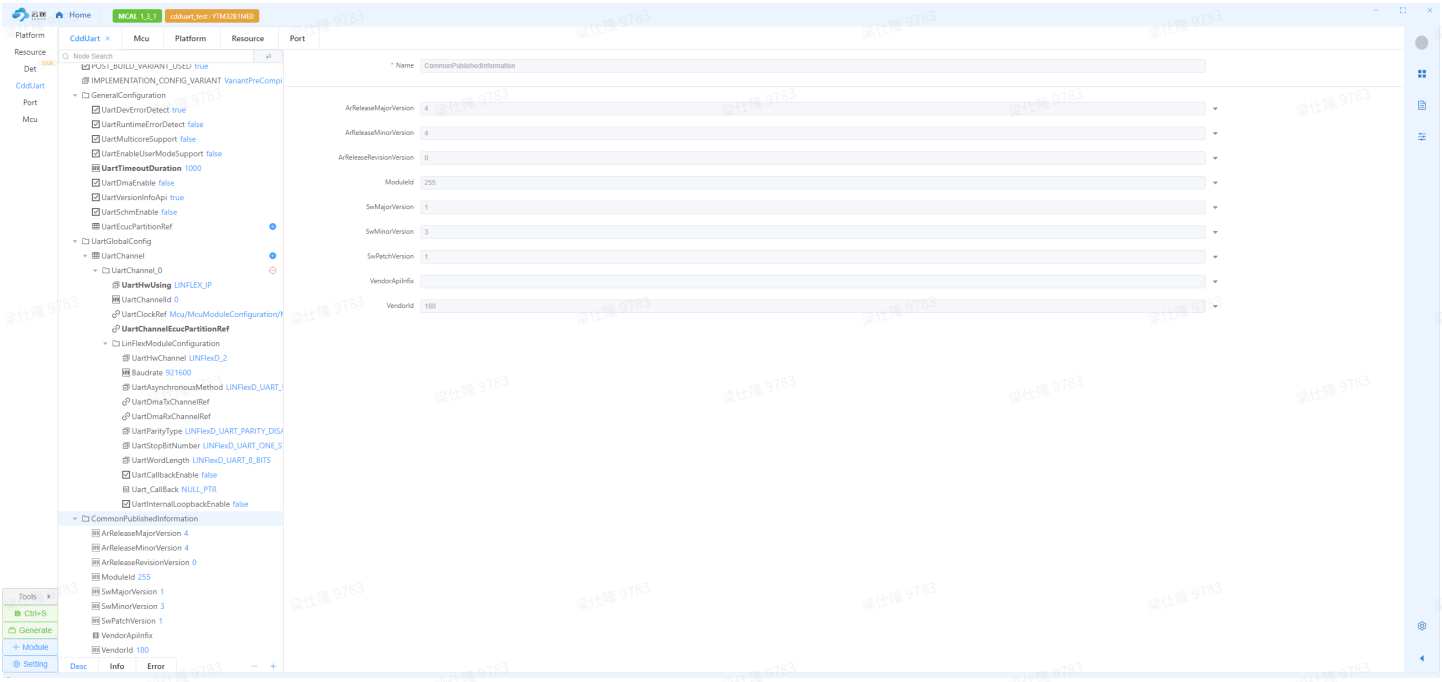
- **配置项介绍：**
  - **UartHwUsing**：依赖于MCU本身资源属性，M系列的UART功能为LINFLEXD模块实现。
  - **UartChannelId**：Channel索引号，当使用多个LINFLEXD模块时，通过此索引号去寻找相应的硬件通道。
  - **UartClockRef**：UART计算波特率所需时钟参数，可选项由前文3.1中McuClockReferencePoint配置。
  - **UartChannelEcucPartitionRef**：多核相关配置，M系列为单核，无需配置此项。

4.2.1 LinFlex模块配置（LinFlexModuleConfiguration）



- **配置项介绍：**
  - **UartHwChannel：**选择硬件instance。
  - **Baudrate：**设置波特率。
  - **UartAsynchronousMethod：**选择Uart传输模式，ME和MD只能使用中断模式。
  - **UartDmaTxChannelRef：**ME和MD只能使用中断模式，该选项不可选。
  - **UartDmaRxChannelRef：**ME和MD只能使用中断模式，该选项不可选。
  - **UartParityType：**奇偶校验类型。
  - **UartStopBitNumber：**设置Uart停止位长度。
  - **UartWordLength：**数据字长度表示每个UART通信帧中数据部分的位数。
  - **UartCallbackEnable：**是否启用Callback。
  - **Uart_CallBack：**只有Enable Callback才可配置此项，设置CallBack函数名称。
  - **UartInternalLoopbackEnable：**当启用 UART 内部环回功能时，UART 接收和发送的数据会在通信模块内部回送，而不会发送到外部设备或线路。这意味着发送到 UART 的数据会被模块接收，并立即回送回 UART 的接收缓冲区，而不需要外部设备的参与。

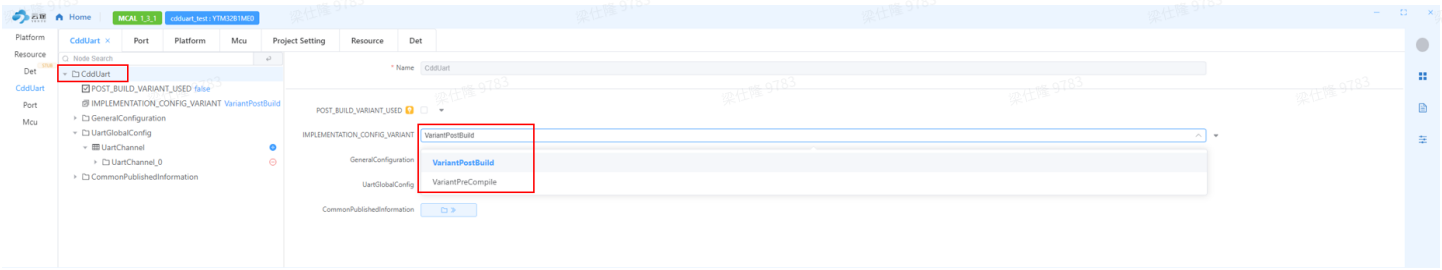
4.3 公共发布信息（CommonPublishedInformation）



- **ModuleId：**每个MCAL模块通常都有一个唯一的Module ID（模块标识符），用于标识特定模块。Module ID 是 DET 模块用于标识和记录错误信息的一部分。DET中需要这个值去定位出问题的模块，由于Uart是非标准的，ID是我们自己定义的。
- 其余配置项为版本信息，全部保持默认，用户无法修改。

5. 工程示例

5.1 YT Config Tool配置示例

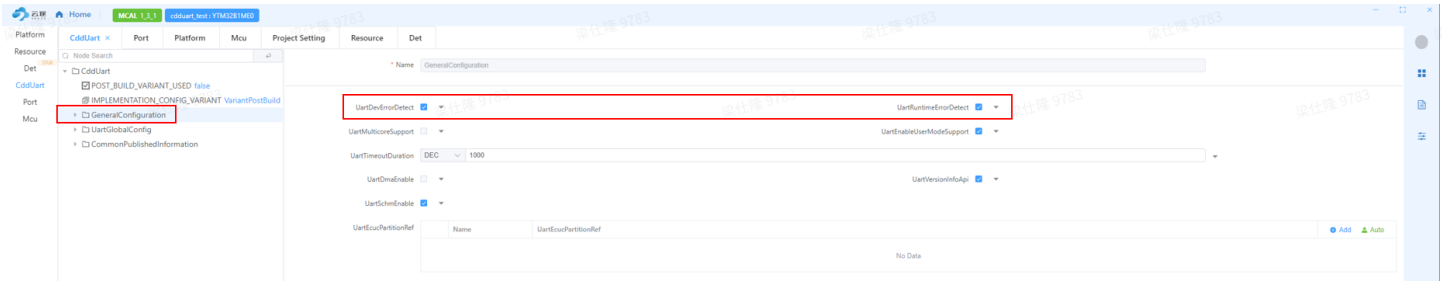


- 选择VariantPostBuild还是VariantPreCompile根据需求选择，具体区别详见：

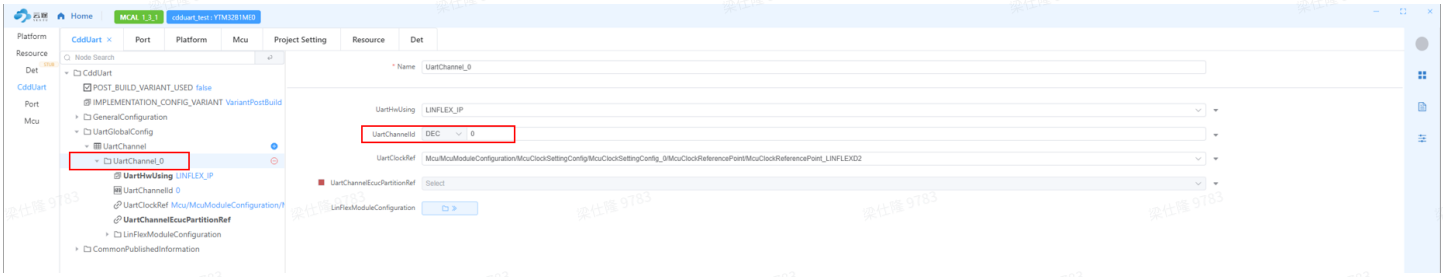
  https://blog.csdn.net/Xiaowestwind/article/details/107115826

- **Tips：**选择VariantPostBuild和VariantPreCompile，初始化传参不一样，如下图：

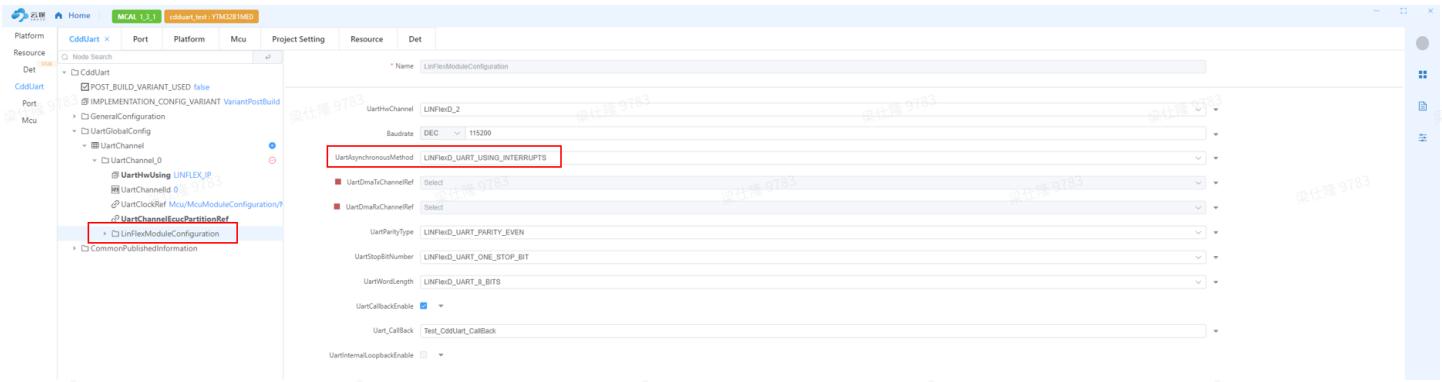
```
/* VariantPostBuild */
CddUart_Init(CddUart_PBCfgVariant[CddUart_GetCoreID()]);
/* VariantPreCompile */
CddUart_Init(0);
```



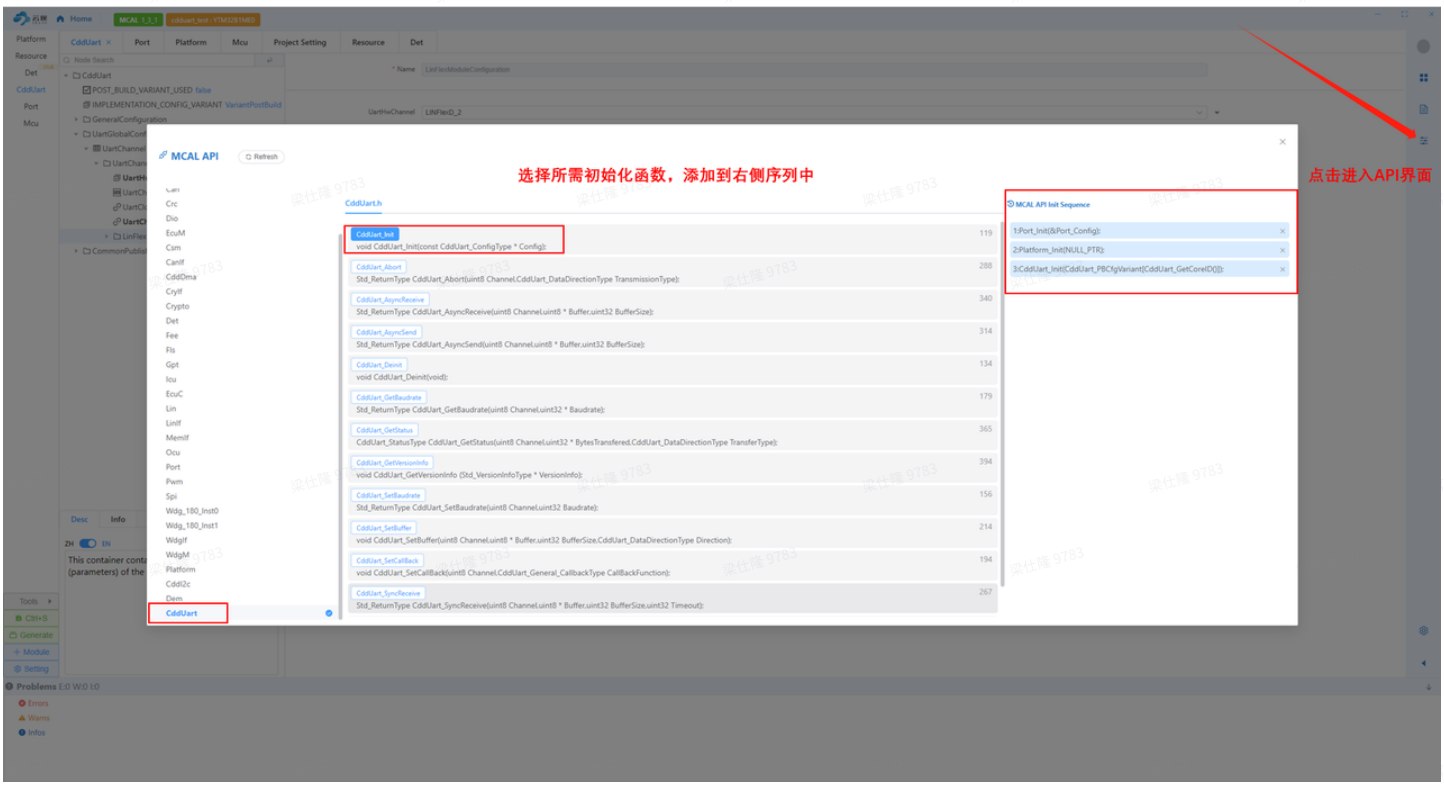
- 在调试时建议打开ErrorDetect便于开发人员定位问题

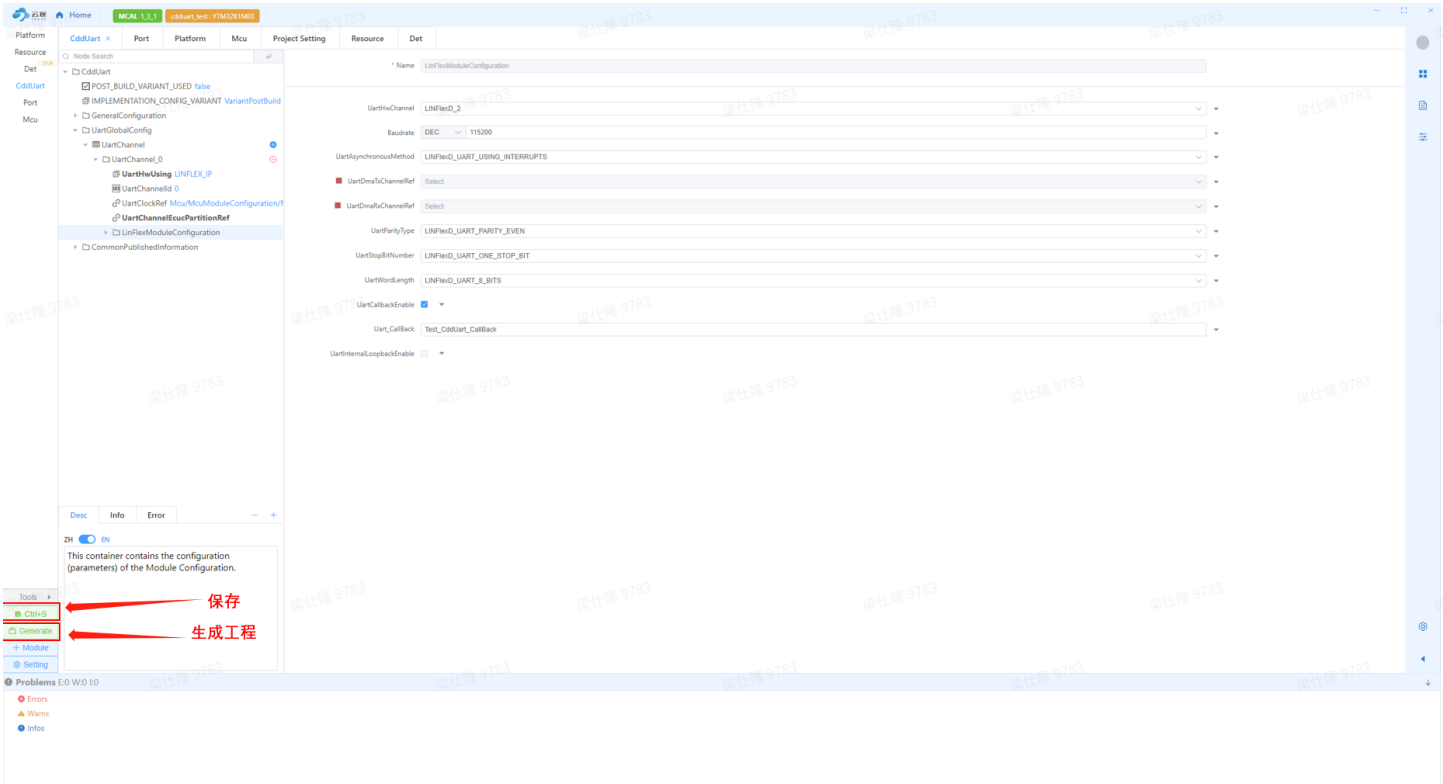


- 注意UartChannelId和UartHwChannel的区别，UartChannelId只是一个索引号



- 注意ME和MD系列MCU只能选择中断模式





5.2 生成代码示例

使用中断方式进行收发测试

5.2.1 头文件

```
/* USER CODE END Header */
#include "Mcal.h"
/* Includes ------------------------------------------------------------------*/

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "CddUart_Define_Types.h"
#include "CddUart_Cfg.h"
#include "CddUart.h"
/* USER CODE END Includes */
```

- 添加CddUart模块所需头文件。
- **注意：**main.c中的所有用户代码必须写在BEGIN和END段之间，否则重新生成工程时用户代码会被清除。

5.2.2 定义参数

```c
/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
#define CDDUART_MSG_1 "Hi, Welcome to use this demo to test uart!\r\n"      //宏定义两条发送信息
#define CDDUART_MSG_2 "Please Input 5 datas,I will them send back!\r\n"
uint8 CddUart_TestedRxBuffer[5] = {0};                                      //定义接收buffer
uint8 CddUart_TestedTxBuffer[5] = {'n','i','h','a','o'};                    //定义发送buffer
/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
Std_ReturnType T_Uart_Status;           //返回状态
#define  CddUart_TestedChannel   2      //Uart的硬件通道号，demo示例采用Uart2
CddUart_General_EventType CddUart_TestedEvent = UART_EVENT_END_TX;  //Uart传输状态赋初值
/* USER CODE END PD */
```

5.2.3 声明回调函数

```c
/* Private function declare ---------------------------------------------------*/
/* USER CODE BEGIN PFDC */
void Test_CddUart_CallBack(uint8 HwChannel,CddUart_General_EventType Event);
/* USER CODE END PFDC */
static void Board_Init(void);

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
void Test_CddUart_CallBack(uint8 HwChannel,CddUart_General_EventType Event)
{
  if(CddUart_TestedChannel == HwChannel)
  {
    CddUart_TestedEvent = Event;
  }
}
/* USER CODE END 0 */
```

- 回调函数的名称需和在YT Config Tool中设置的名称一致。

5.2.4 各模块初始化

```c
static void Board_Init(void)
{
    Port_Init(&Port_Config);
    Platform_Init(NULL_PTR);
    CddUart_Init(CddUart_PBCfgVariant[CddUart_GetCoreID()]);
}
```

- 在配置工具中选择的初始化函数，会生成到main.c中的Board_Init()函数中。

```c
int main(void)
{
    /* USER CODE BEGIN 1 */
    Mcu_Init(0);
    Mcu_InitClock(0);
#if (MCU_NO_PLL == STD_OFF)
    while (MCU_PLL_LOCKED != Mcu_GetPllStatus())
    {
        /*Busy wait until the System Pll is locked*/
    }
    Mcu_DistributePllClock();
#endif
    /* USER CODE END 1 */
    Board_Init();
    /* USER CODE BEGIN 2 */
    /* USER CODE END 2 */
```

若启用PLL，则需在MCU初始化后等待PLL锁频完成

- 添加Mcu初始化函数。

5.2.5 CddUart收发函数调用

```
95      /* Infinite loop */
96      /* USER CODE BEGIN WHILE */
97      T_Uart_Status = CddUart_SyncSend(CDDUART_Channel_0, (uint8 *)CDDUART_MSG_1, sizeof(CDDUART_MSG_1),0xFFFFFFFF);
98      T_Uart_Status = CddUart_SyncSend(CDDUART_Channel_0, (uint8 *)CDDUART_MSG_2, sizeof(CDDUART_MSG_2),0xFFFFFFFF);
99      while (1)
100     {
101         /* USER CODE END WHILE */
102         /* USER CODE BEGIN 3 */
103         if(CddUart_TestedEvent == UART_EVENT_END_RX)
104         {
105         CddUart_TestedEvent = UART_EVENT_RX_FULL;
106         T_Uart_Status = CddUart_AsyncSend(CDDUART_Channel_0, CddUart_TestedTxBuffer, 5);
107         }
108         else if(CddUart_TestedEvent == UART_EVENT_END_TX)
109         {
110         CddUart_TestedEvent = UART_EVENT_TX_EMPTY;
111         T_Uart_Status = CddUart_AsyncReceive(CDDUART_Channel_0,CddUart_TestedRxBuffer,5);
112         }
113         else
114         {
115             ;
116         }
117     }
118     /* USER CODE END 3 */
119 }
```

- CddUart_SyncSend(uint8 Channel,uint8* Buffer, uint32 BufferSize, uint32 Timeout)

同步发送函数，执行上图中97和98行会发送宏定义CDDUART_MSG_1和CDDUART_MSG_2的信息，上位机将收　到如下信息：

- CddUart_AsyncReceive(uint8 Channel, uint8* Buffer, uint32 BufferSize)

  异步接收函数，在此示例中，注意示例中发送和接收使用的不是一个buffer。

- CddUart_AsyncSend(uint8 Channel,uint8* Buffer, uint32 BufferSize)

  异步发送函数，在示例中，当上位机向mcu发送“hello”时，mcu会将“nihao”返回给上位机。

- 如下图所示，上位机向mcu发送hello，mcu将返回hello。

文档历史

版本号	日期	修订记录
V1.0	2023.11.20	初始版本