# MCAL应用_CddDma模块配置及实例（一）

## 1. 版本

Config Tool Version：1.10.0

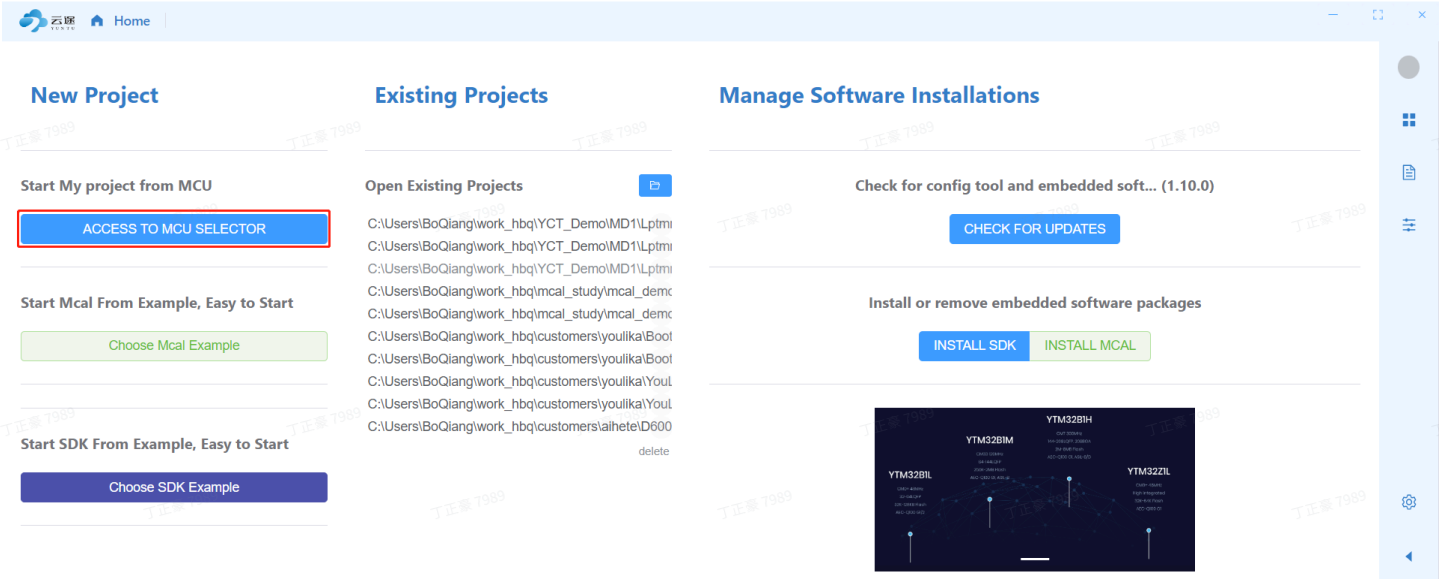
ME0 MCAL version：1.3.1

## 2. 前言

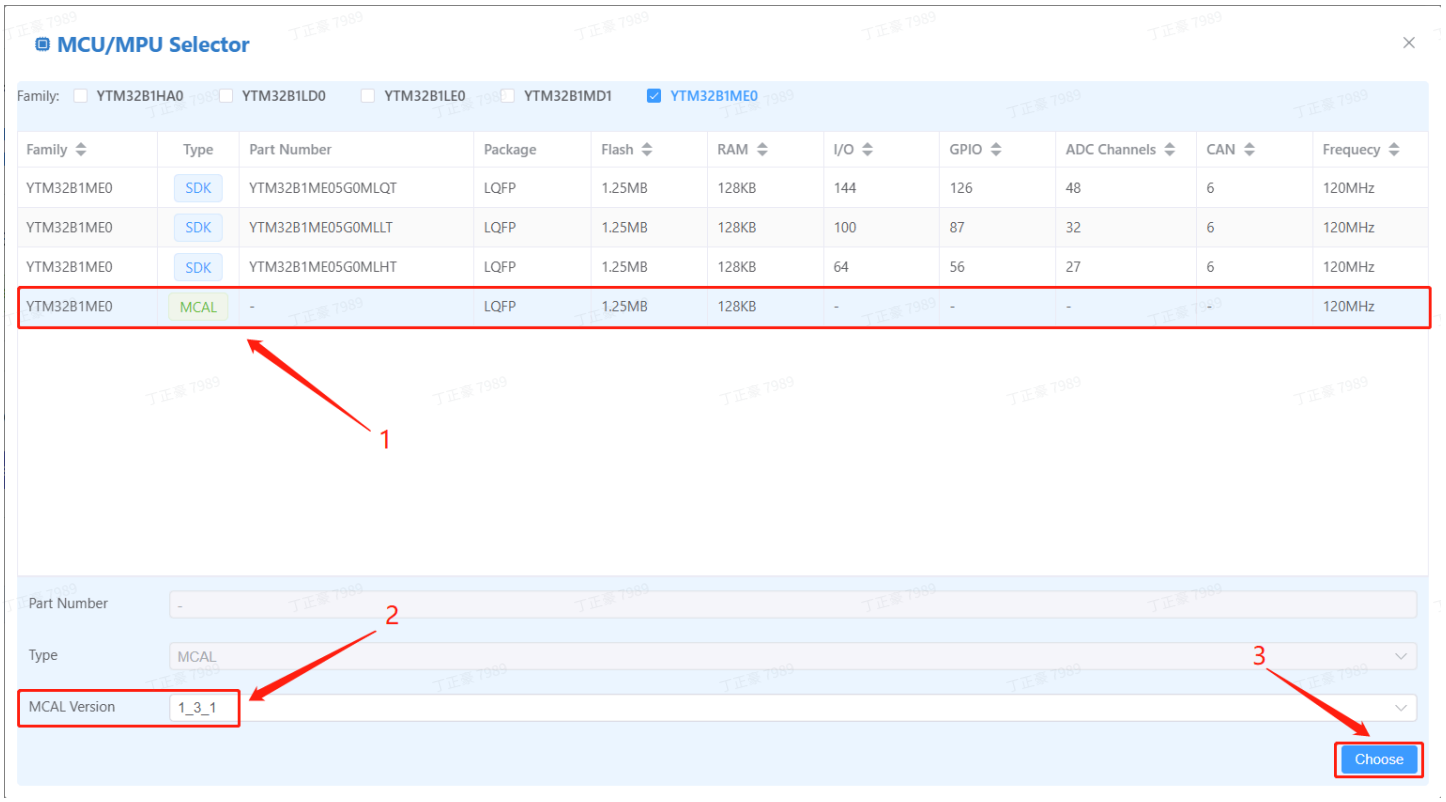
1. 本文基于YTM32B1ME0芯片，介绍了YCT（Yuntu Config Tool）中CddDma的配置步骤以及使用方式。
2. CddDma模块功能依赖于Mcu和Platform，需要在Mcu中配置系统时钟，选择核时钟，以供CddDma正常工作，当需要用到CddDma的半中断、搬运完成中断或者错误中断时，则需要在Platform中打开相应的中断开关。
3. 本实例配置了两个DMA通道进行memory to memory的数据搬运，两个通道的具体配置i如下：
   a. 通道0每次搬运一个字节，总共搬运16个字节，软件触发一次；
   b. 通道1每次搬运两个字节，总共搬运32个字节，软件触发两次。

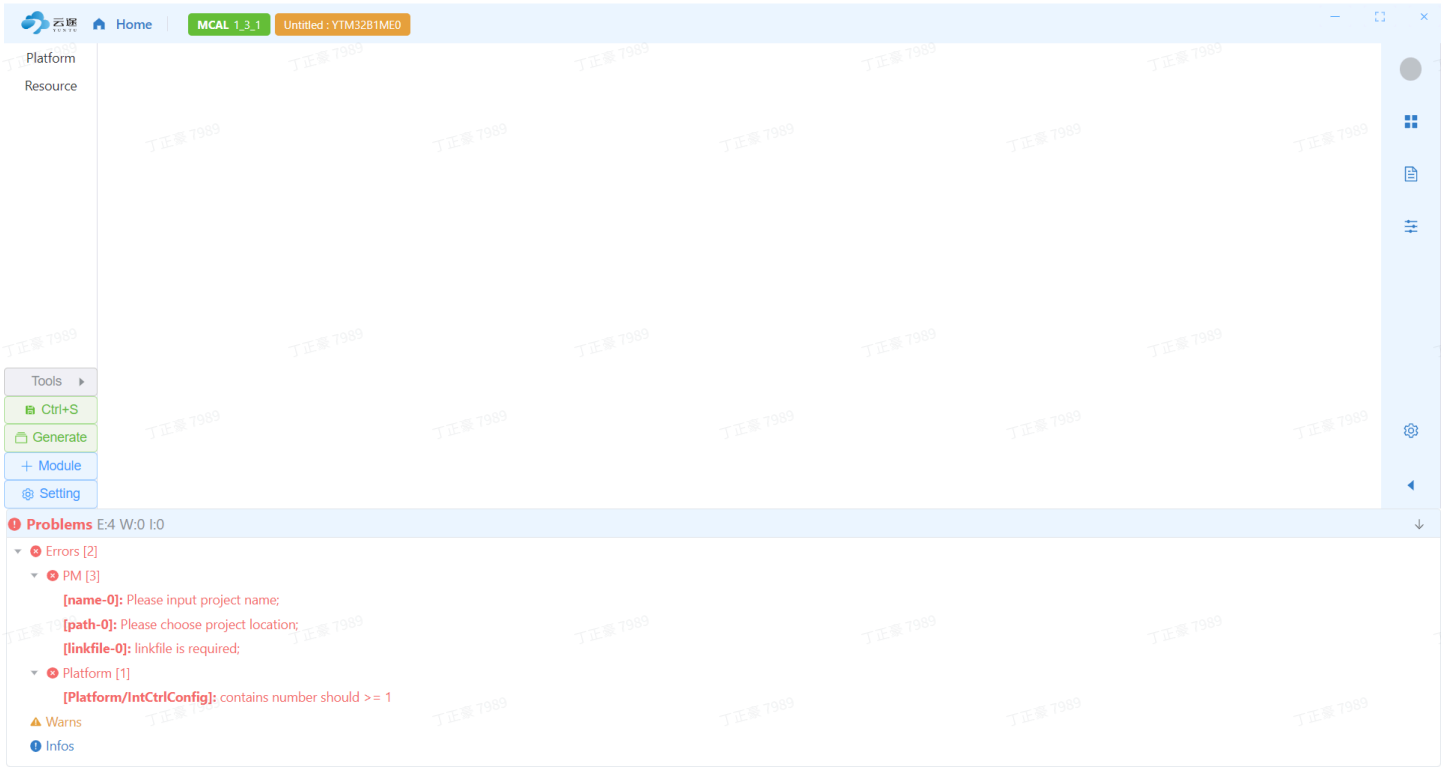
## 3. YCT配置步骤

1. 打开YT Config Tool，点击ACCESS TO MCU SELECTOR进入MCU选型界面。

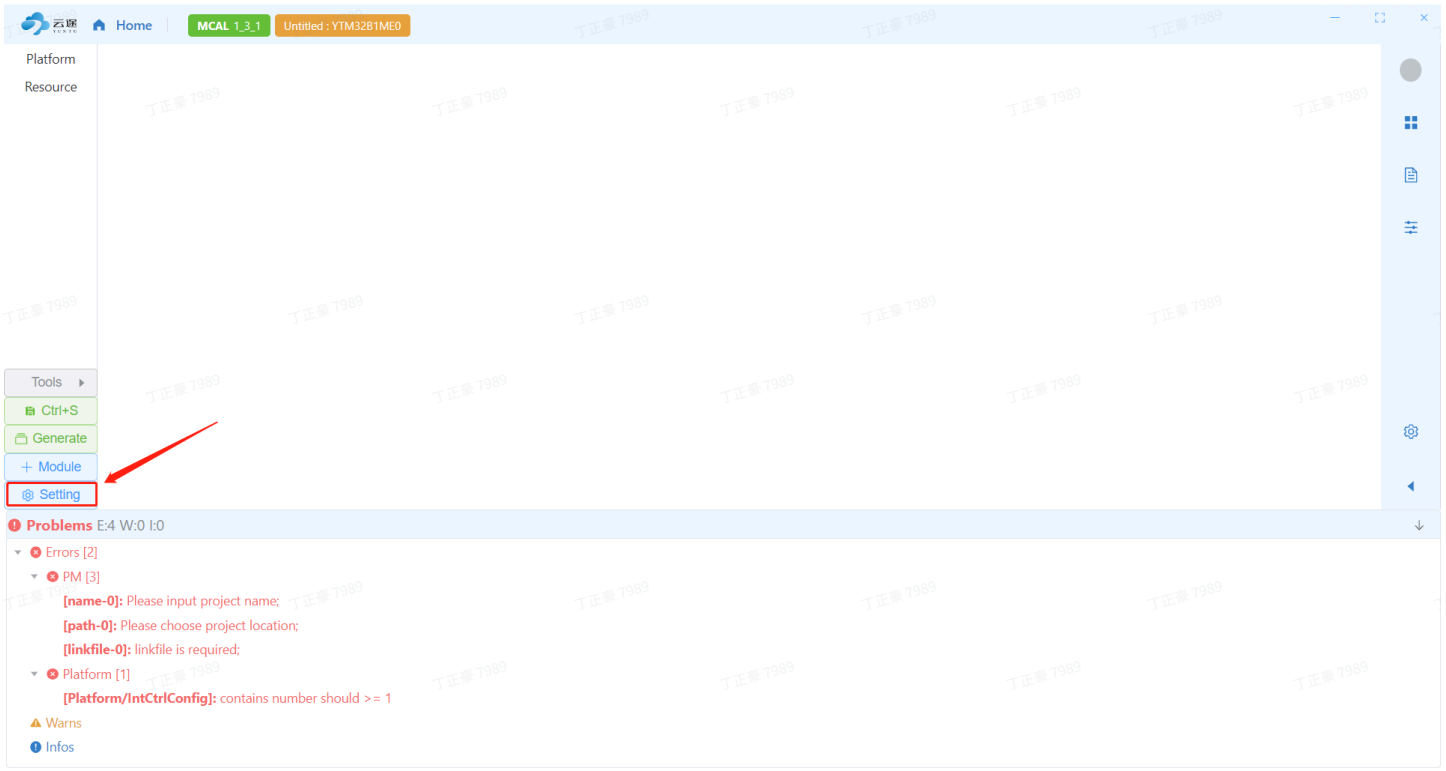


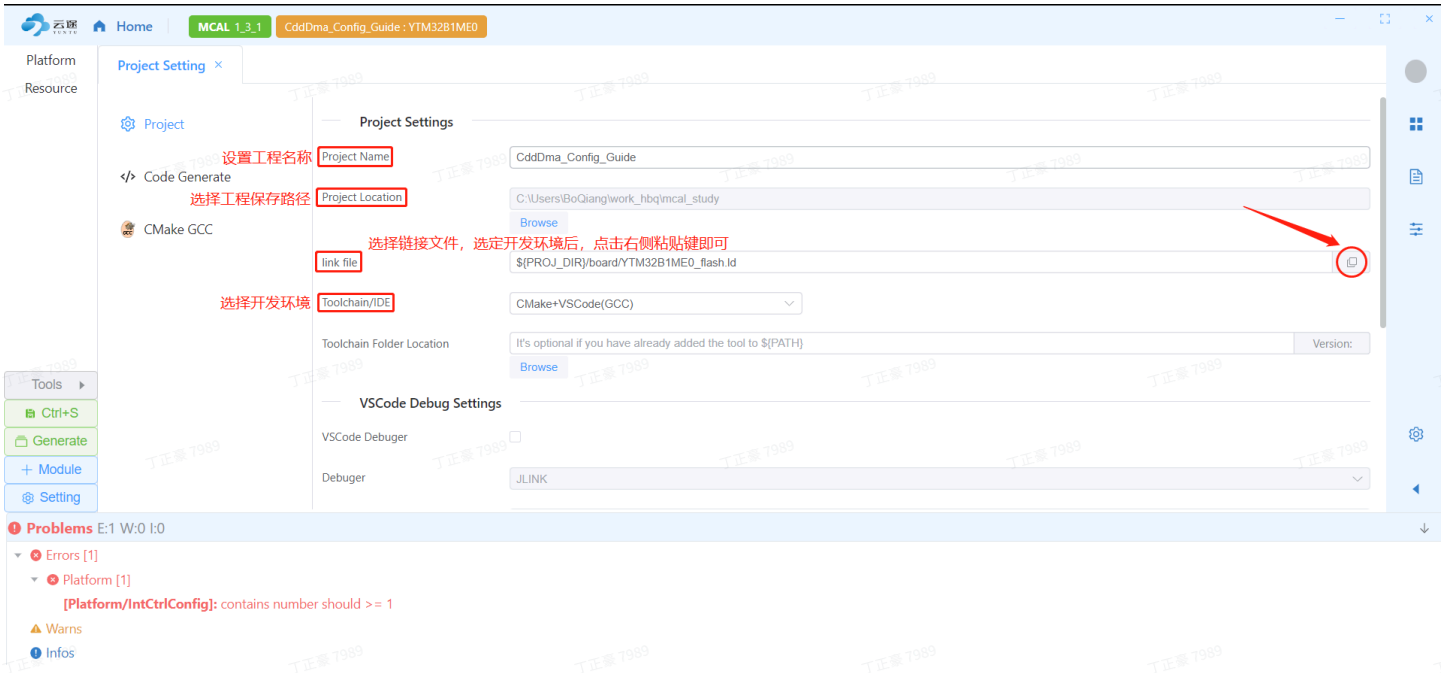

2. 选择MCU对应的MCAL开发包及版本，确认选择进入配置界面。本例中选用的是YTM32B1ME0芯片1.3.1版本的MCAL开发包。



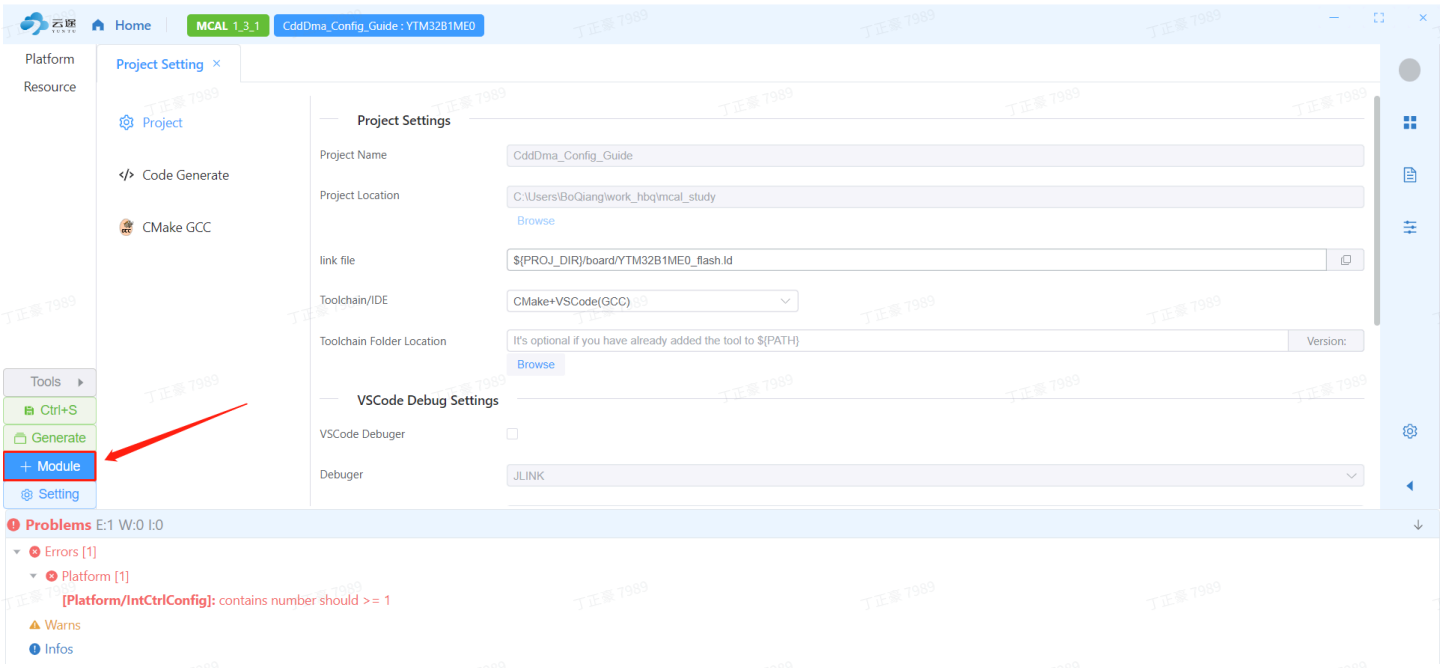




3. 点击Setting进入工程文件设置界面，设置工程名称、保存路径并选择集成开发环境，设置完成后保存配置。这里以CMake+VSCode(GCC)为例。



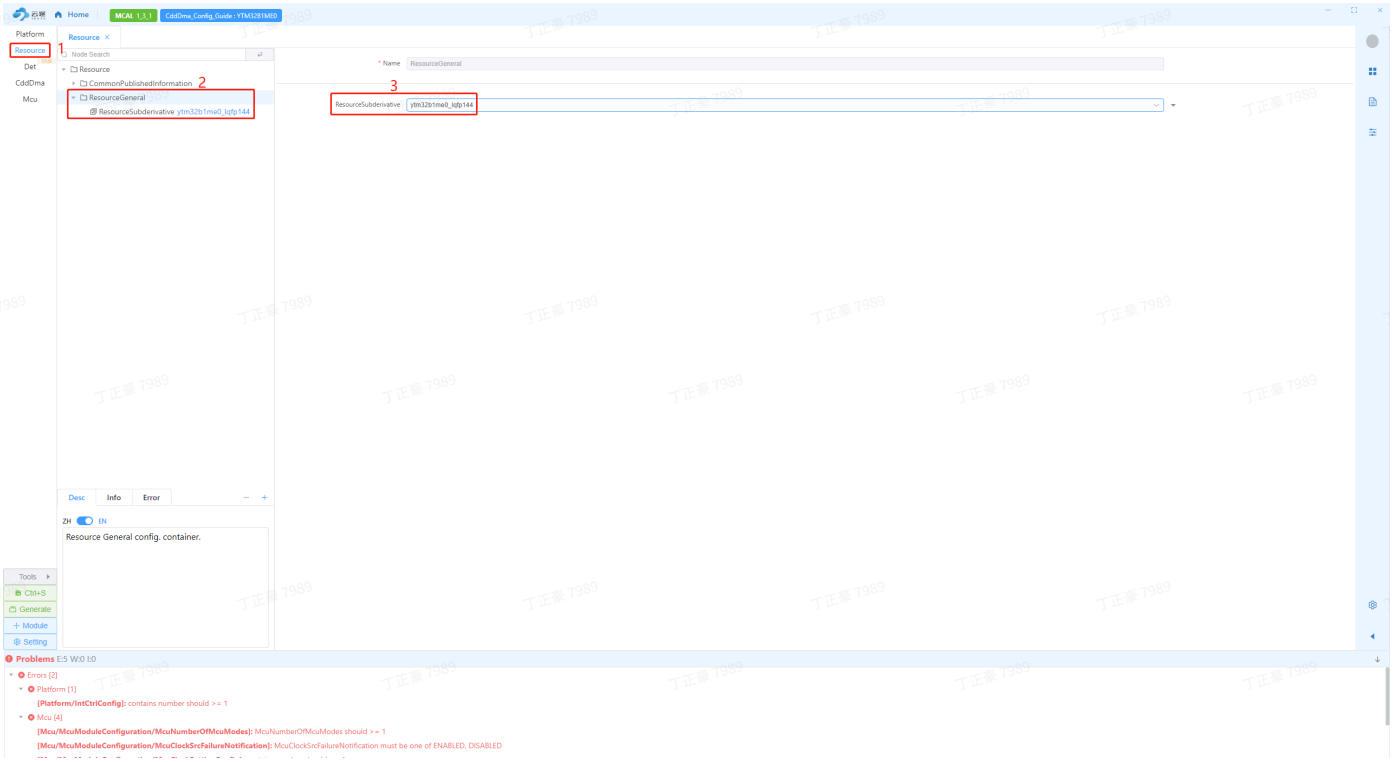

4. 点击左侧菜单栏“+Module”添加需要配置的模块。



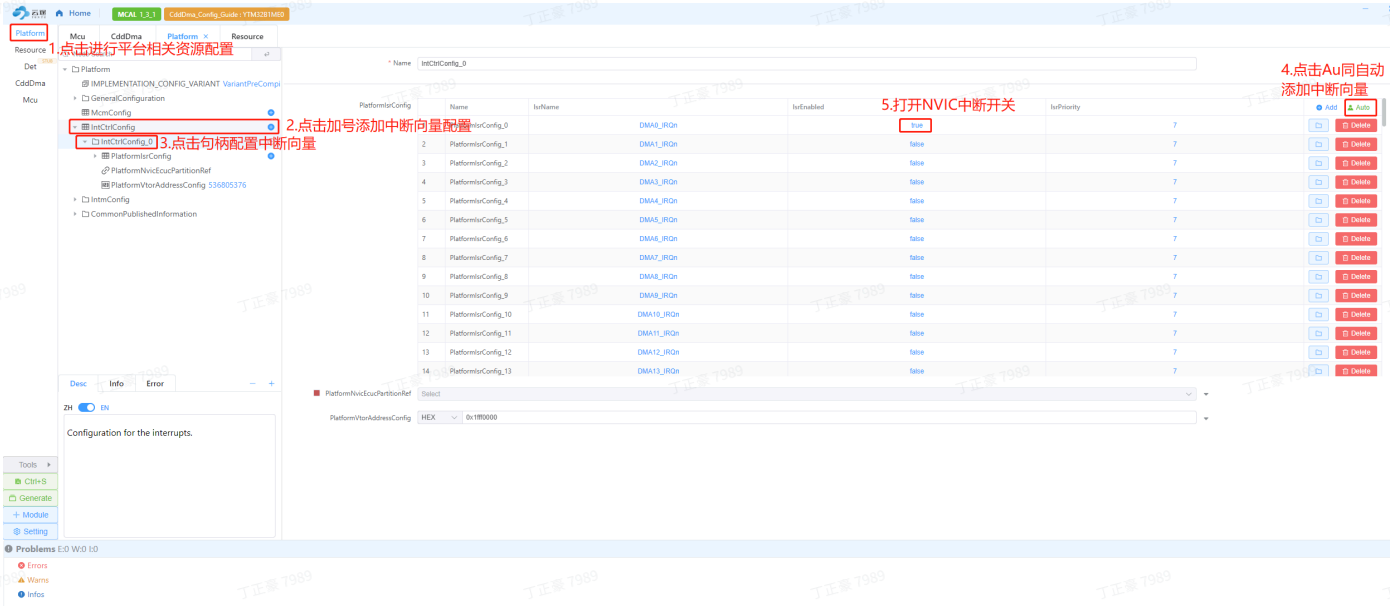


5. 添加模块后，分别进行各个模块的配置。Mcu和Platform的详细配置介绍请参考文档《MCAL应用 - 工程建立及工具通用设置》和《MCAL应用 - Mcu模块配置及实例》。

   a. 资源配置，选择目标芯片型号。这里以ytm32b1me0_lqfp144为例。

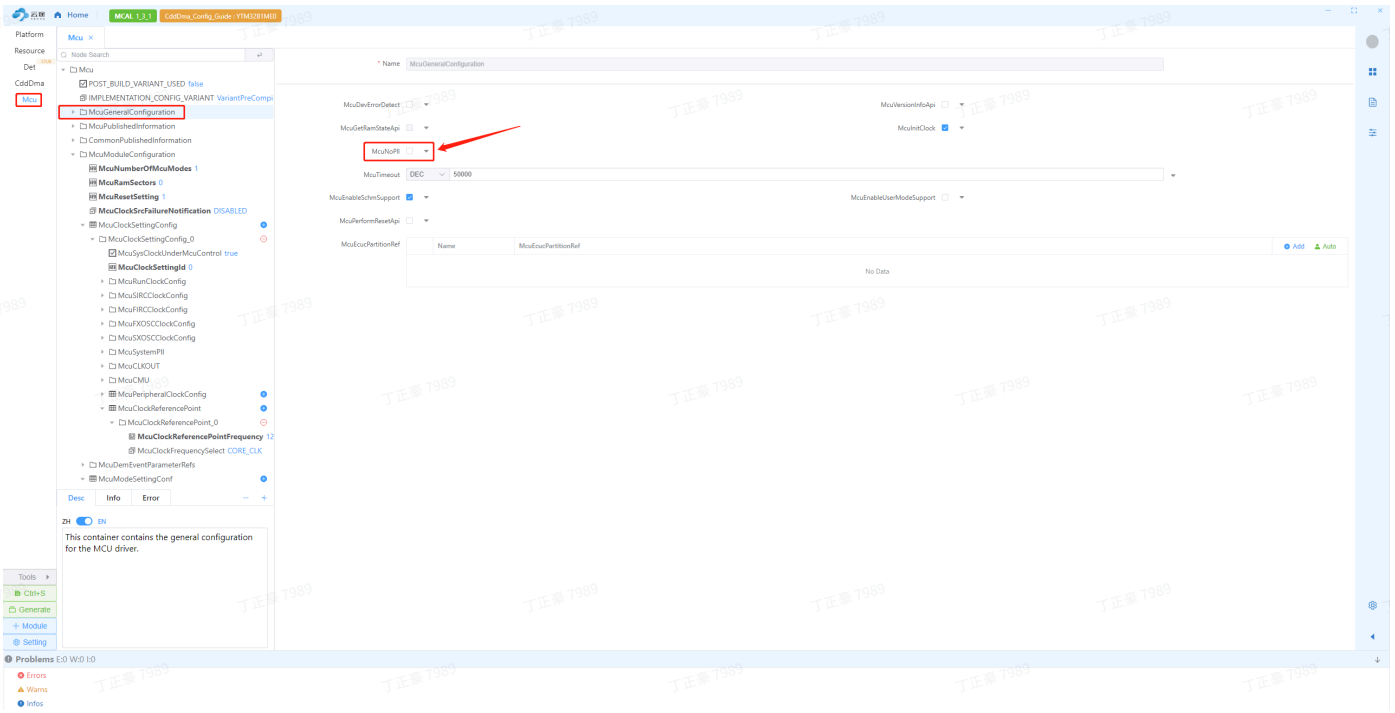



## b. 配置中断向量表



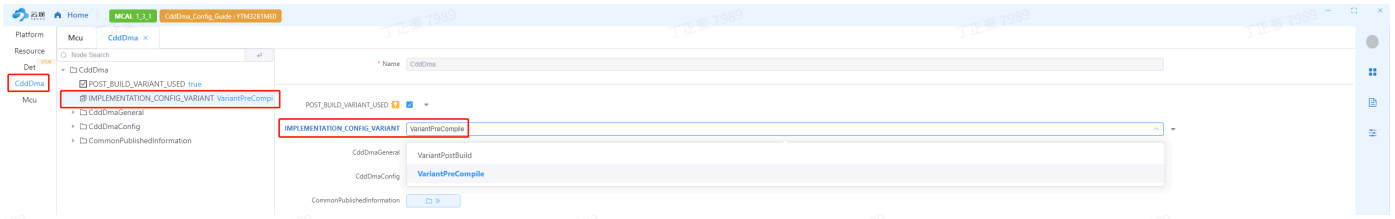

## c. 配置时钟

系统时钟和外设时钟的相关配置请参考Mcu模块的配置教程。这里系统时钟选择PLL（120MHz），core clock、fastbus clock和slowbus clock的分频分别为1、1、3。需要使用PLL时，必须去掉勾选McuNoPll（默认是勾选），如下。

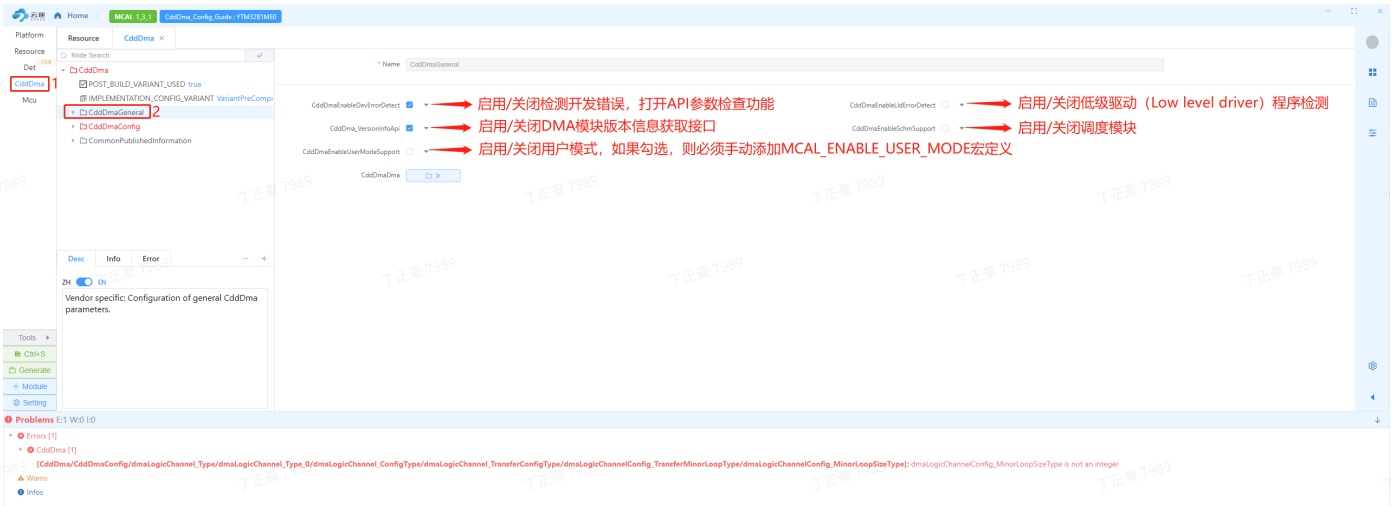


## d. 配置CddDma模块

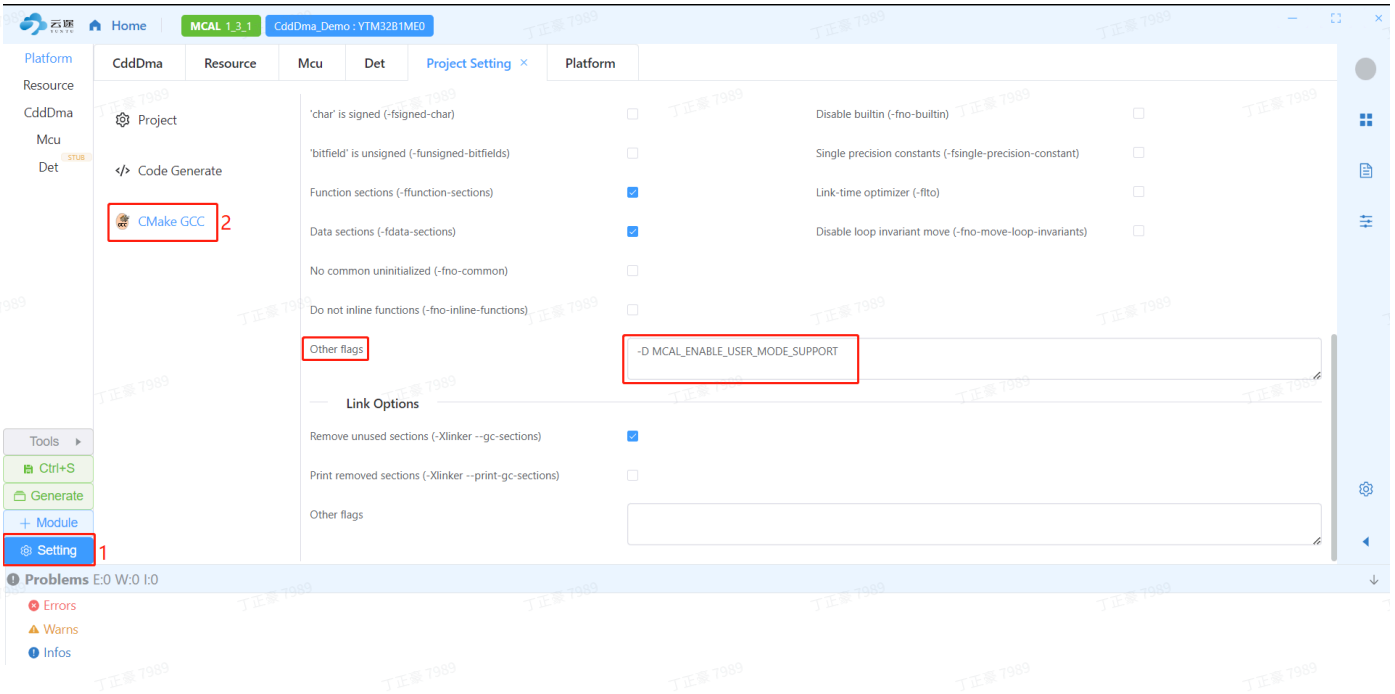
选择编译过程中的参数配置方式。VariantPreCompile和VariantPostBuild的区别可以参考博客https://blog.csdn.net/Xiaowestwind/article/details/107115826。



DMA通用设置。



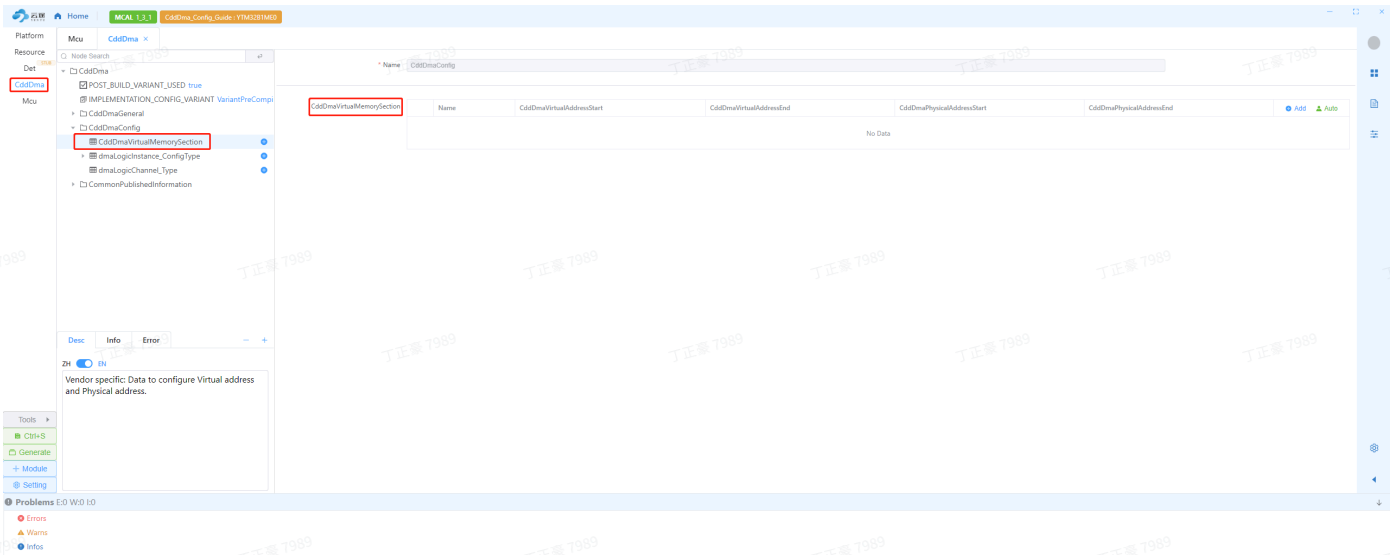

在上面设置中如果勾选了CddDmaEnableUserModeSupport配置，则必须要添加MCAL_ENABLE_USER_MODE_SUPPORT宏定义，添加方式如下：

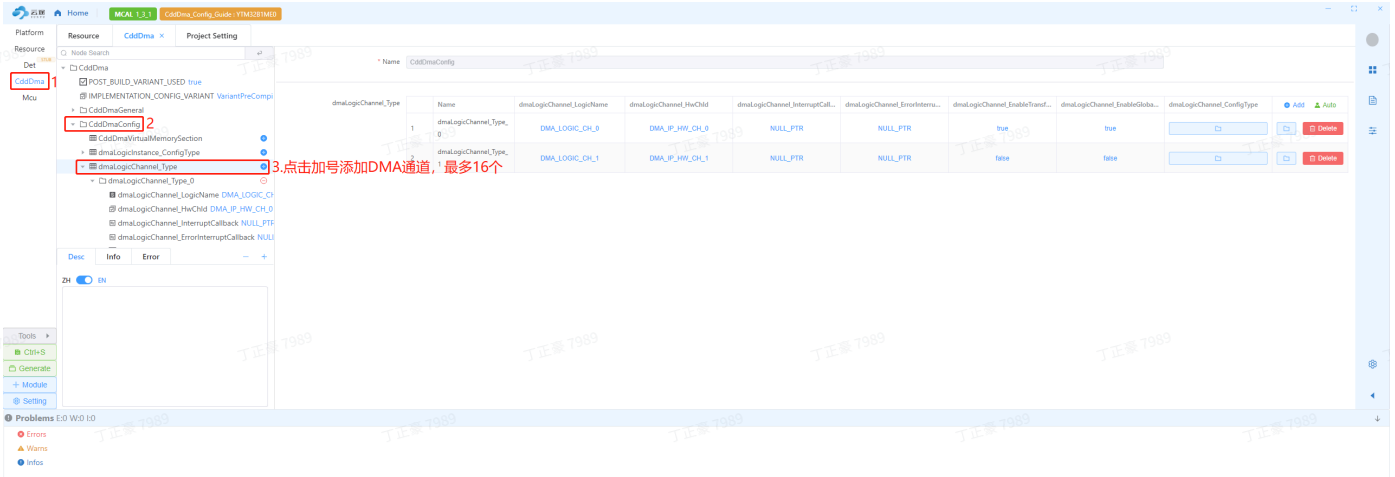




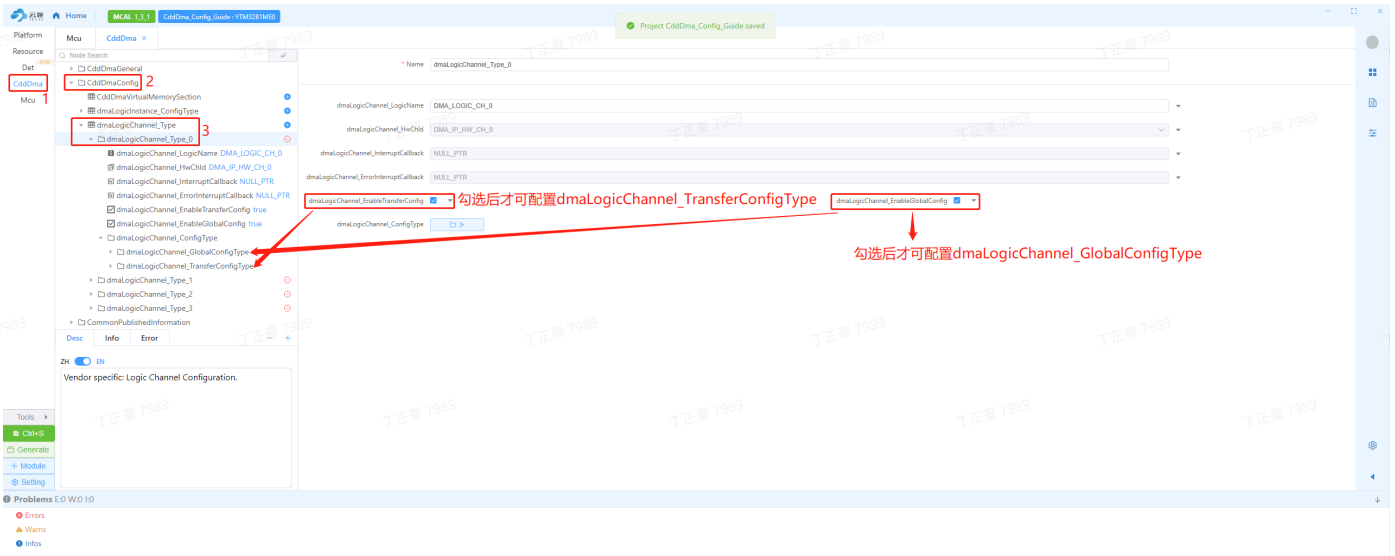
DMA模块功能配置。

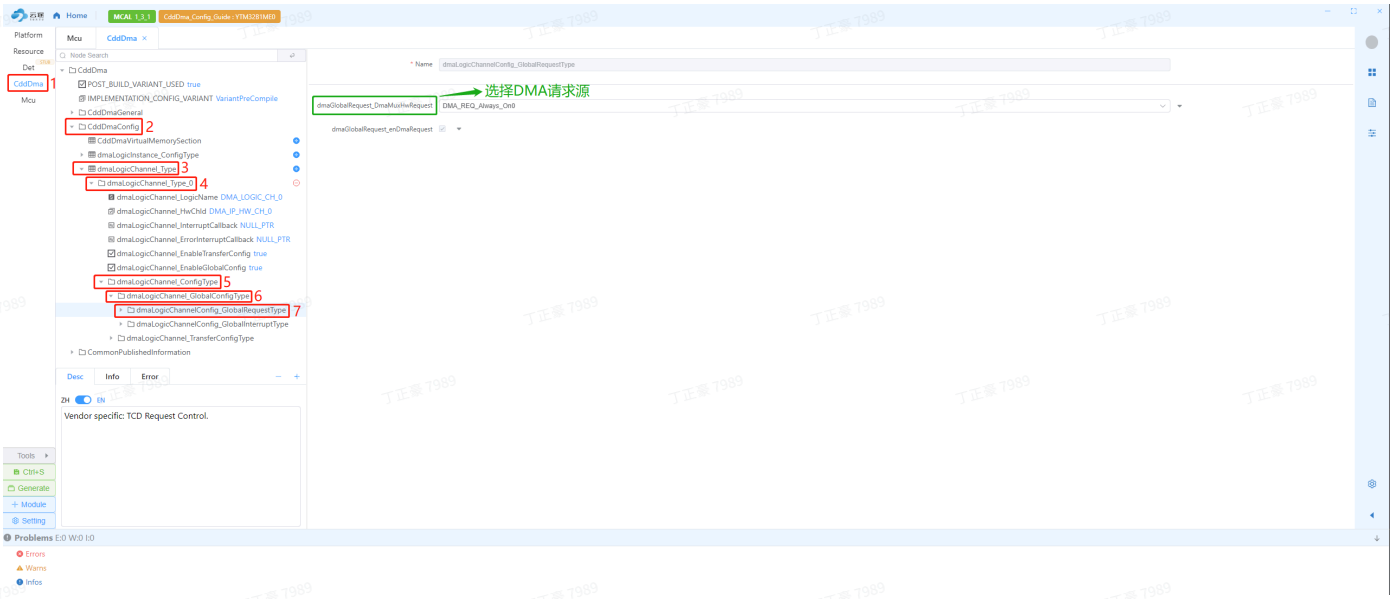
CddDmaConfig句柄中，CddDmaVirtualMemorySection配置是预留的，M系列不支持这个功能，会在后面的版本中删除，用户可以忽略这项配置。

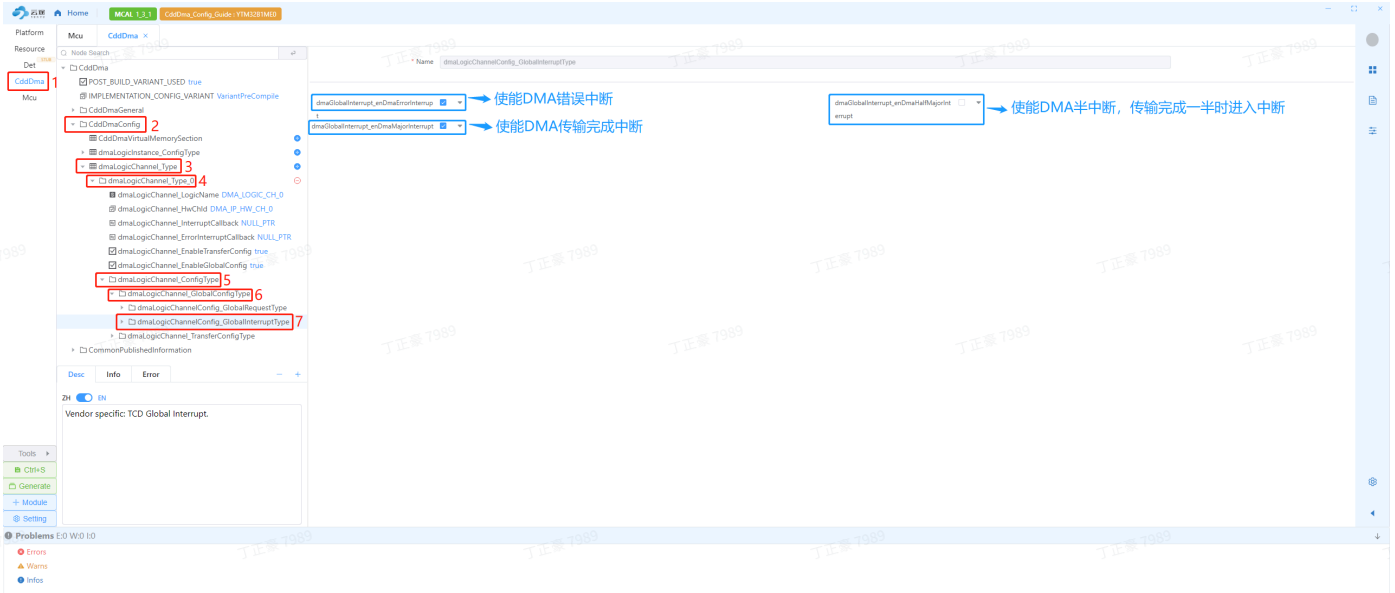


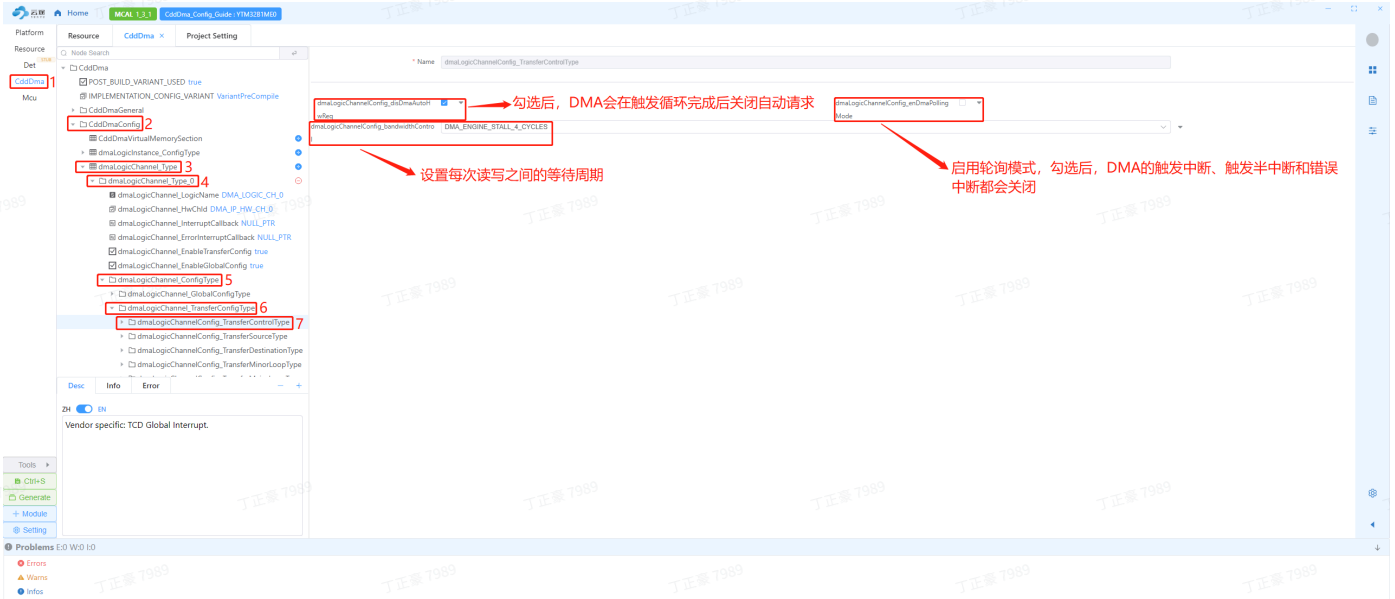



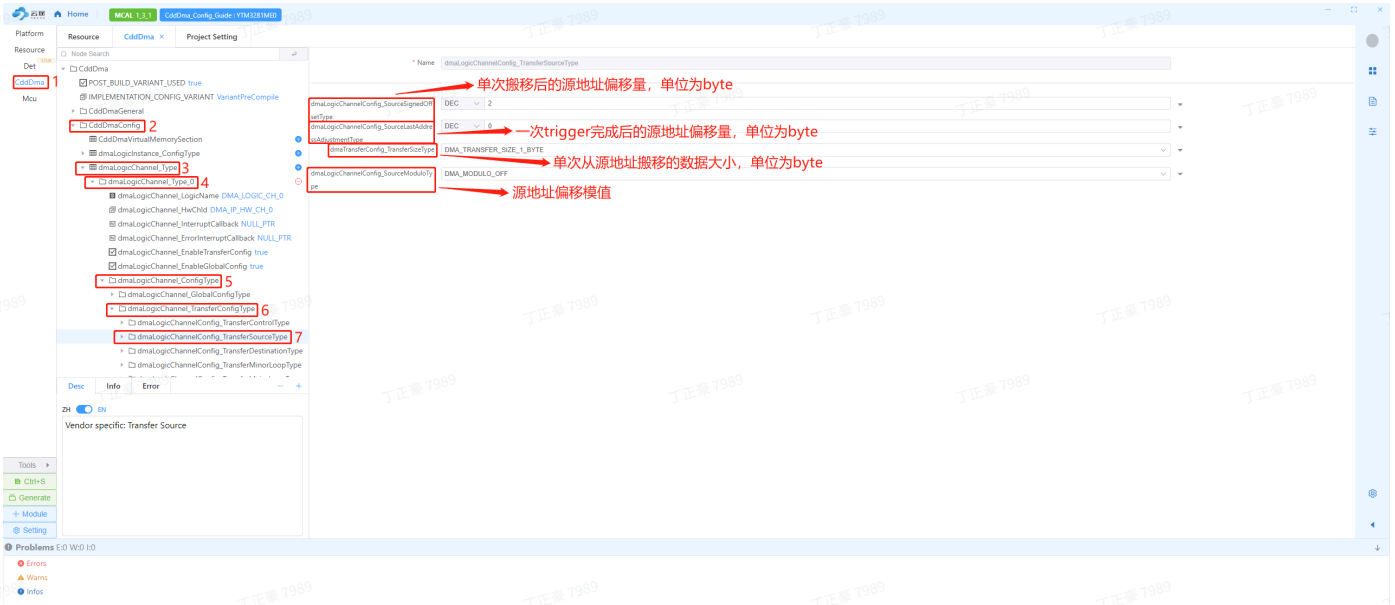

然后进行单个通道的功能配置。



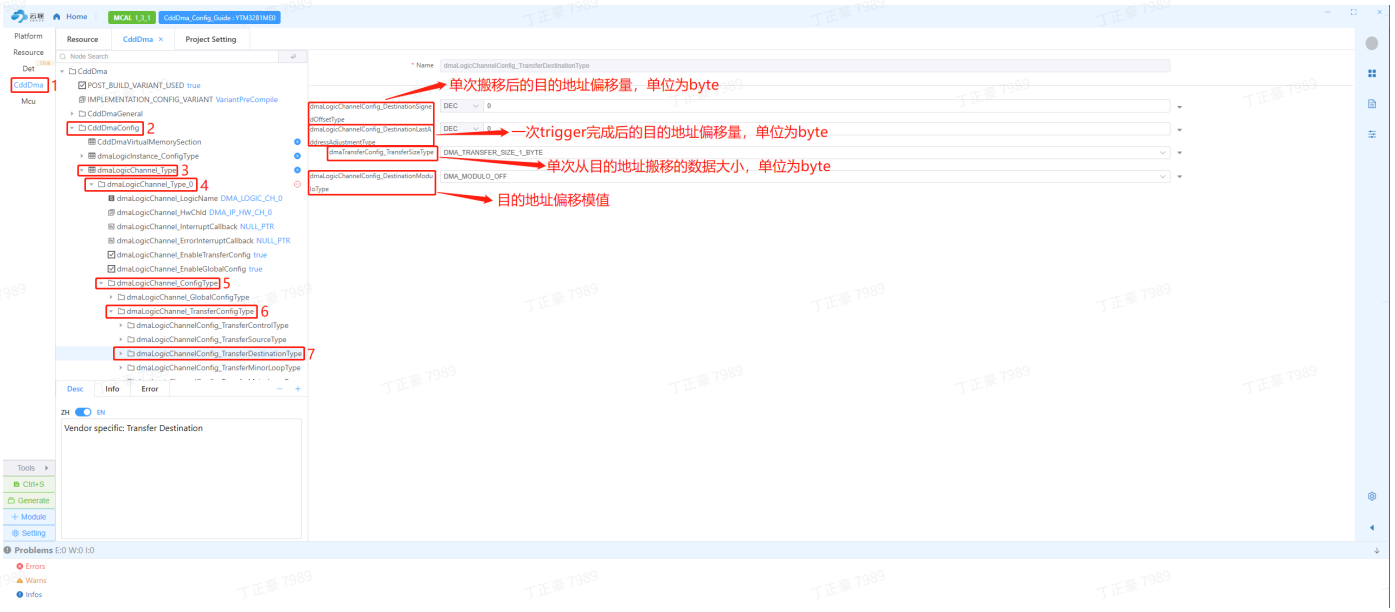

选择DMA请求源
Vendor specific: TCD Request Control.

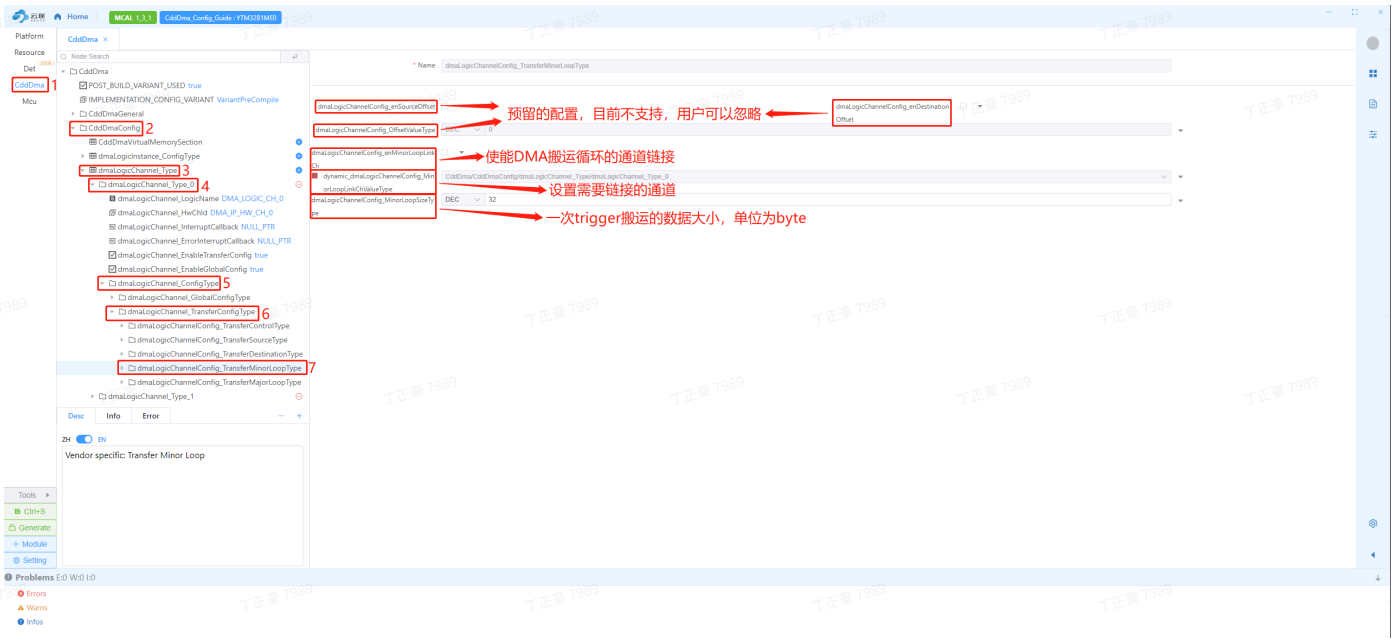

使能DMA错误中断
使能DMA传输完成中断
使能DMA半中断，传输完成一半时进入中断
Vendor specific: TCD Global Interrupt.

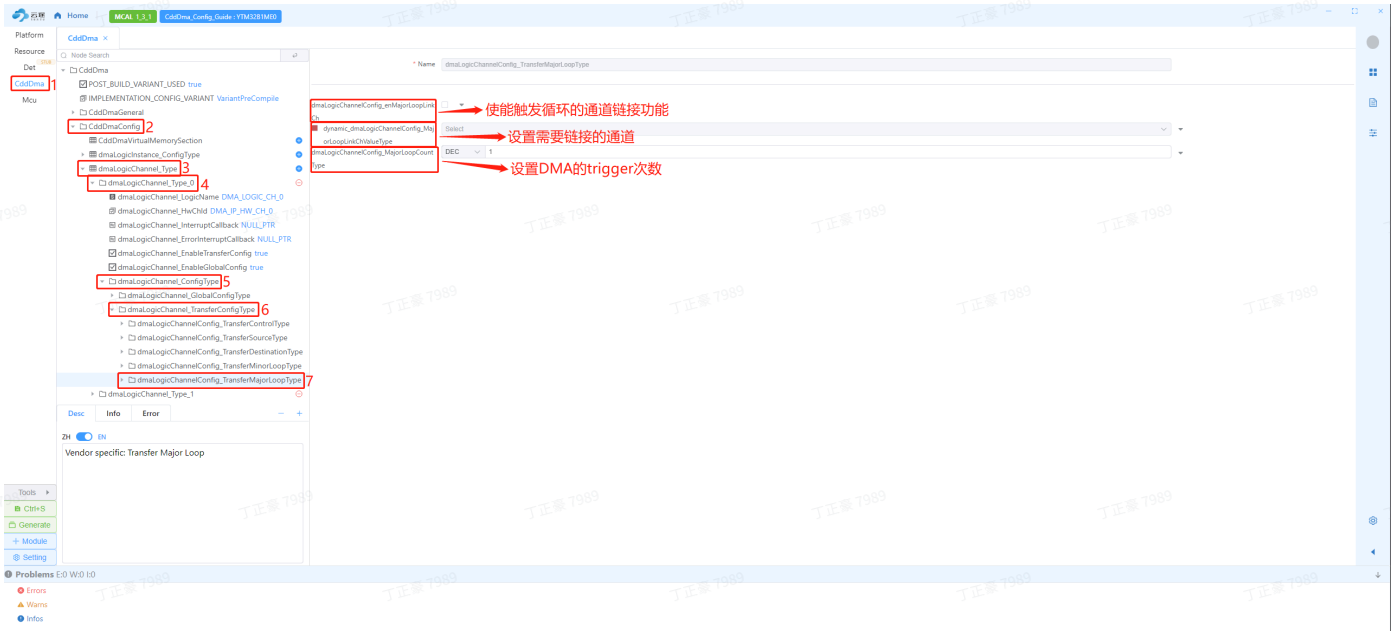

勾选后，DMA会在触发循环完成后关闭自动请求
设置每次读写之间的等待周期
启用轮询模式，勾选后，DMA的触发中断、触发半中断和错误中断都会关闭
Vendor specific: TCD Global Interrupt.




上面配置中的modulo是针对源地址偏移的上限设置。若modulo设置为0（DMA_MODULO_OFF），则地址偏移不受影响；若modulo设为1（DMA_DODULO_2B），则源地址总共最多可偏移2^1bytes；若modulo设为2（DMA_MODULO_4B），则源地址总共最多可偏移2^2bytes，后面的目的地址模值同理。

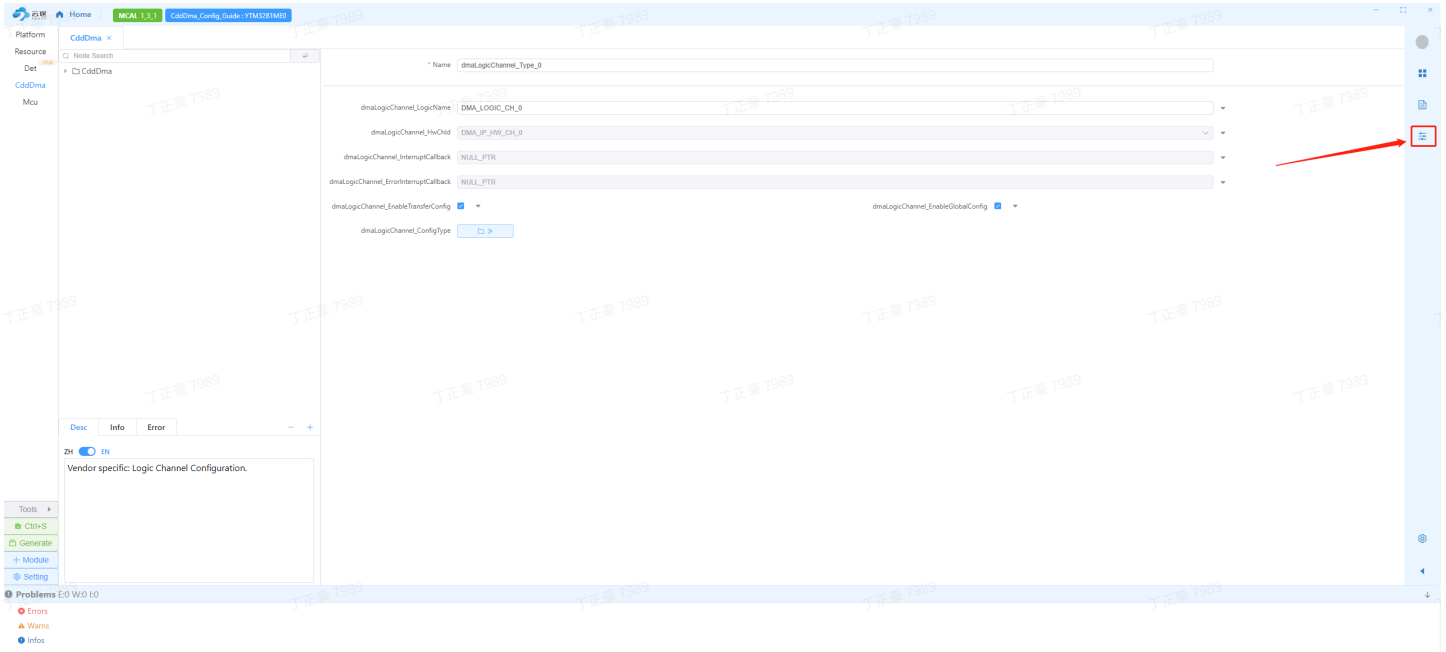




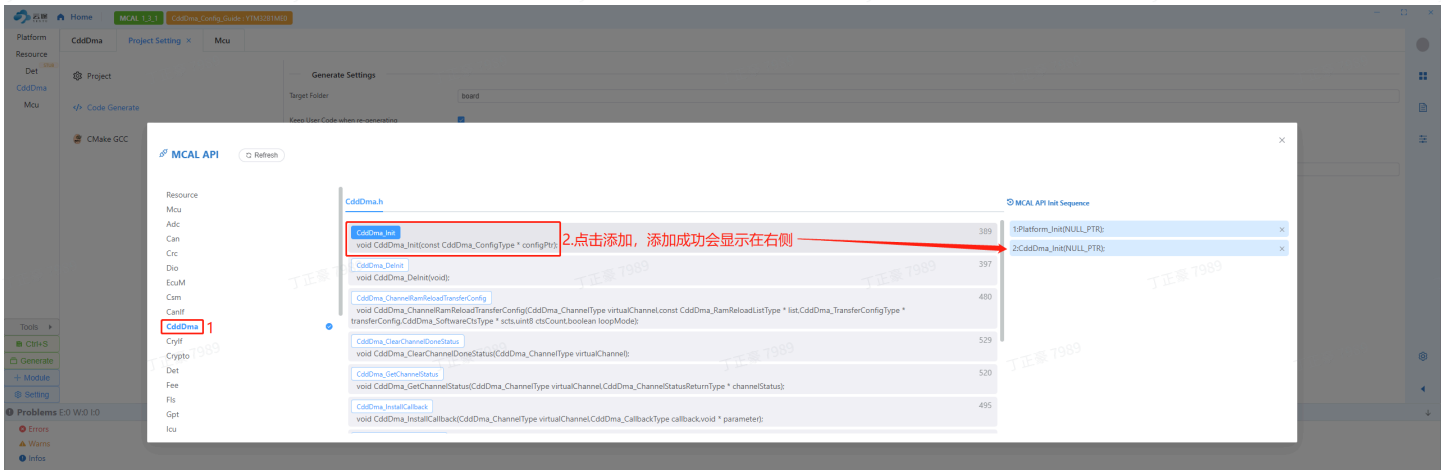




搬运循环和trigger循环请用户参考DMA的应用笔记，https://cloud.ytm32.cn/s/vBECk?path=%2F04_Application%20Note中的AN0030。其他通道的配置同上。

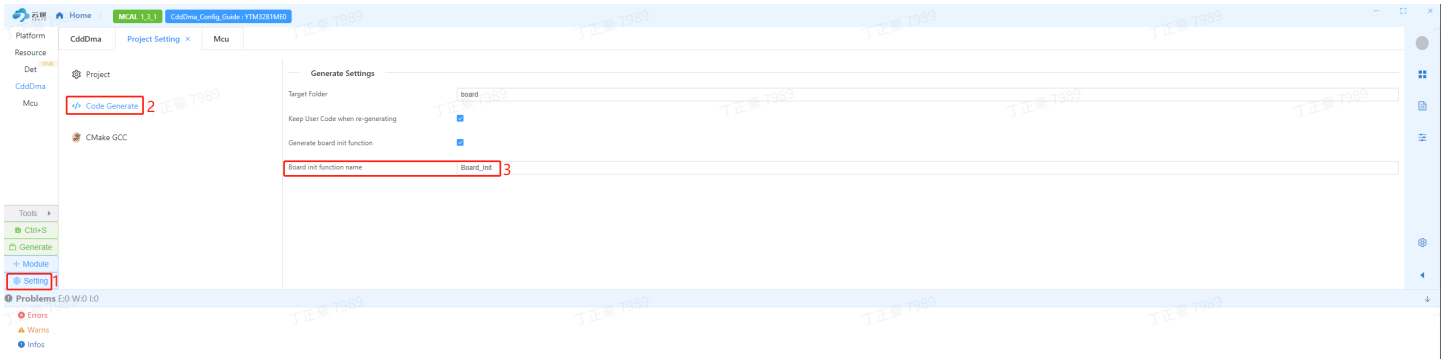
6. 所有模块配置完成后，点击右侧菜单栏的API选项添加初始化函数。





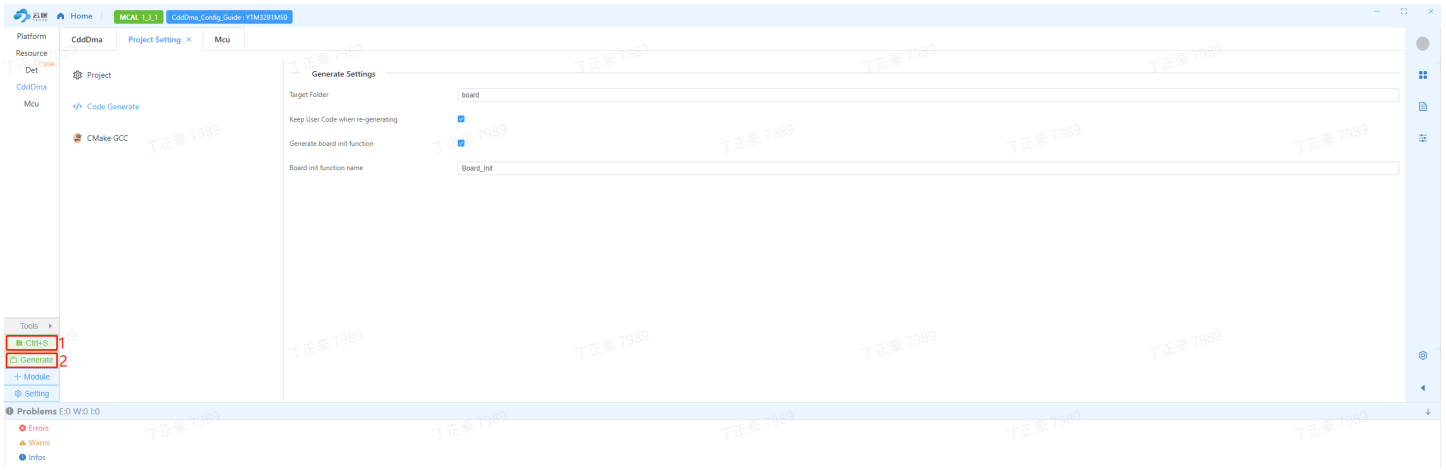



上面添加的接口函数会生成在Board_Init()里面，如下所示：




其中的Board init function name用户可以根据需要自行更改。

7. API添加完成后，保存并生成工程。




点击generate生成代码。


# 4. 生成工程介绍

1. 时钟初始化。

```c
int main(void)
{
    /* USER CODE BEGIN 1 */
    uint8 bufferloop = 0;
    mem2memTransferConfig = *DmaChannelTransferConfigArry[CddDmaConf_CddDmaConfig_dmaLogicChannel_Type_0];

    Mcu_Init(NULL_PTR);
    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 */
```

初始化系统时钟和外设时钟，并等待PLL lock

2. 配置DMA通道传输结构（CTS）。对应的API为，CddDma_SetLogicChannelTransfer()，四个参数分别为DMA通道编号、源地址、目的地址和通道传输配置结构体（CddDma_TransferConfigType）。这里配置两个通道进行数据搬运。

```c
int main(void)
{
    /* USER CODE BEGIN 1 */
    uint8 bufferloop = 0;
    mem2memTransferConfig0 = *DmaChannelTransferConfigArry[CddDmaConf_CddDmaConfig_dmaLogicChannel_Type_0];
    mem2memTransferConfig1 = *DmaChannelTransferConfigArry[CddDmaConf_CddDmaConfig_dmaLogicChannel_Type_1];

    Mcu_Init(NULL_PTR);
    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 */
    CddDma_SetLogicChannelTransfer(CddDmaConf_CddDmaConfig_dmaLogicChannel_Type_0, (uint32)SourceData1, (uint32)DestData1,  &mem2memTransferConfig0);
    CddDma_SetLogicChannelTransfer(CddDmaConf_CddDmaConfig_dmaLogicChannel_Type_1, (uint32)SourceData2, (uint32)DestData2,  &mem2memTransferConfig1);
    CddDma_InstallCallback(CddDmaConf_CddDmaConfig_dmaLogicChannel_Type_0, DmaChannel0Callback, NULL_PTR);
    CddDma_InstallCallback(CddDmaConf_CddDmaConfig_dmaLogicChannel_Type_1, DmaChannel1Callback, NULL_PTR);
    CddDma_InstallErrorCallback(CddDmaConf_CddDmaConfig_dmaLogicChannel_Type_0, DmaChannel0ErrorCallback, NULL_PTR);
    CddDma_InstallErrorCallback(CddDmaConf_CddDmaConfig_dmaLogicChannel_Type_1, DmaChannel1ErrorCallback, NULL_PTR);
```

3. 注册DMA中断回调函数和错误中断回调函数。

```c
int main(void)
{
    /* USER CODE BEGIN 1 */
    uint8 bufferloop = 0;
    mem2memTransferConfig0 = *DmaChannelTransferConfigArry[CddDmaConf_CddDmaConfig_dmaLogicChannel_Type_0];
    mem2memTransferConfig1 = *DmaChannelTransferConfigArry[CddDmaConf_CddDmaConfig_dmaLogicChannel_Type_1];

    Mcu_Init(NULL_PTR);
    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 */
    CddDma_SetLogicChannelTransfer(CddDmaConf_CddDmaConfig_dmaLogicChannel_Type_0, (uint32)SourceData1, (uint32)DestData1,  &mem2memTransferConfig0);
    CddDma_SetLogicChannelTransfer(CddDmaConf_CddDmaConfig_dmaLogicChannel_Type_1, (uint32)SourceData2, (uint32)DestData2,  &mem2memTransferConfig1);
    CddDma_InstallCallback(CddDmaConf_CddDmaConfig_dmaLogicChannel_Type_0, DmaChannel0Callback, NULL_PTR);
    CddDma_InstallCallback(CddDmaConf_CddDmaConfig_dmaLogicChannel_Type_1, DmaChannel1Callback, NULL_PTR);
    CddDma_InstallErrorCallback(CddDmaConf_CddDmaConfig_dmaLogicChannel_Type_0, DmaChannel0ErrorCallback, NULL_PTR);
    CddDma_InstallErrorCallback(CddDmaConf_CddDmaConfig_dmaLogicChannel_Type_1, DmaChannel1ErrorCallback, NULL_PTR);
```

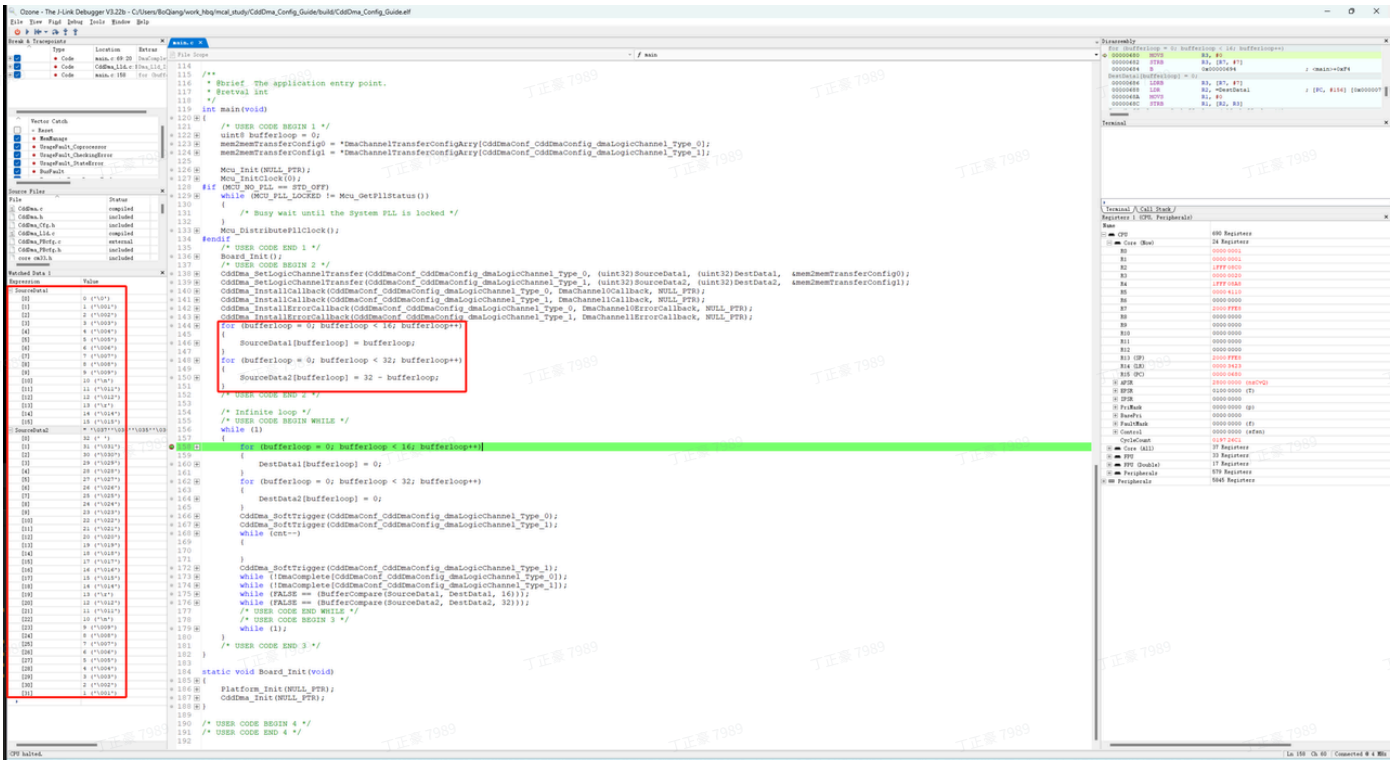
4. 运行结果展示。

本实例中使用的软件trigger，此时只需要调用如下函数使能DMA通道，使能后该通道会自动开启搬运。

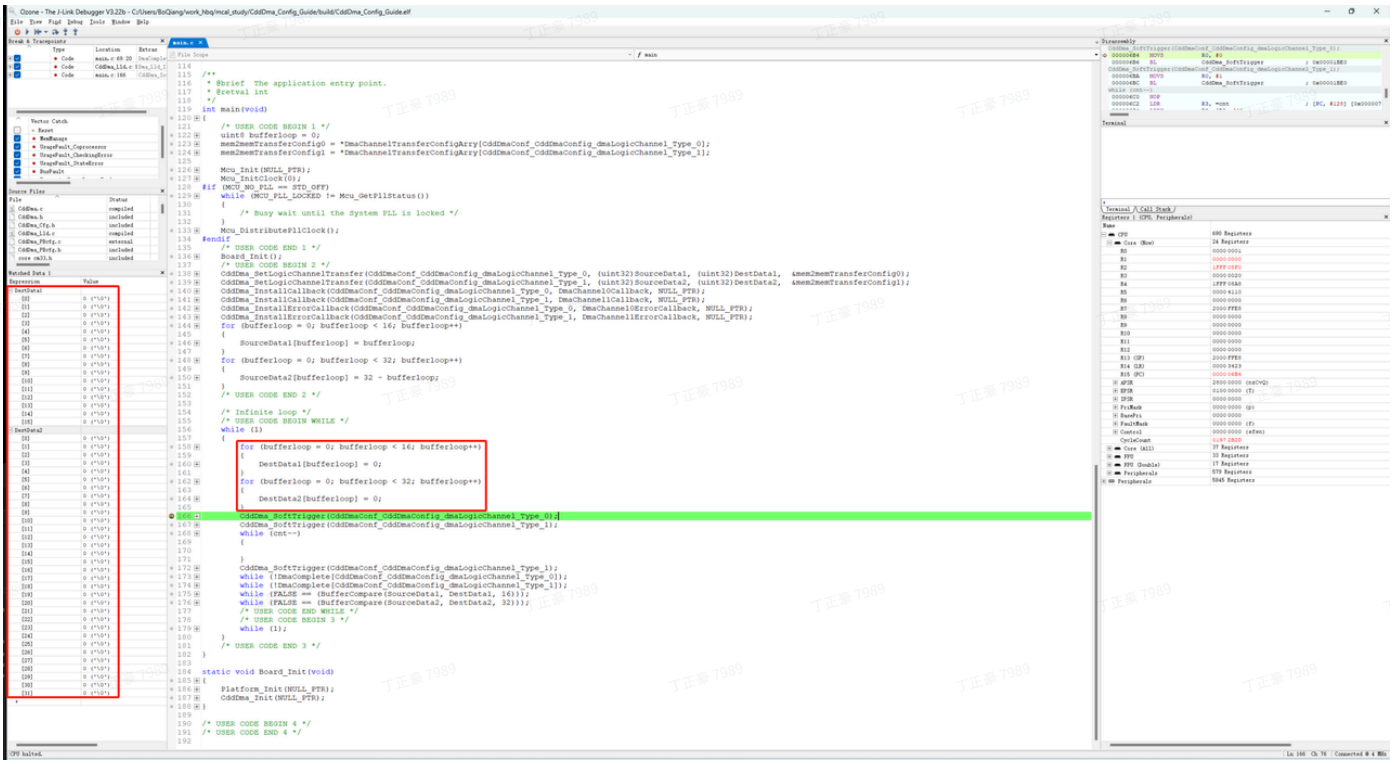
代码块

```
CddDma_SoftTrigger(CddDma_ChannelType virtualChannel)
```

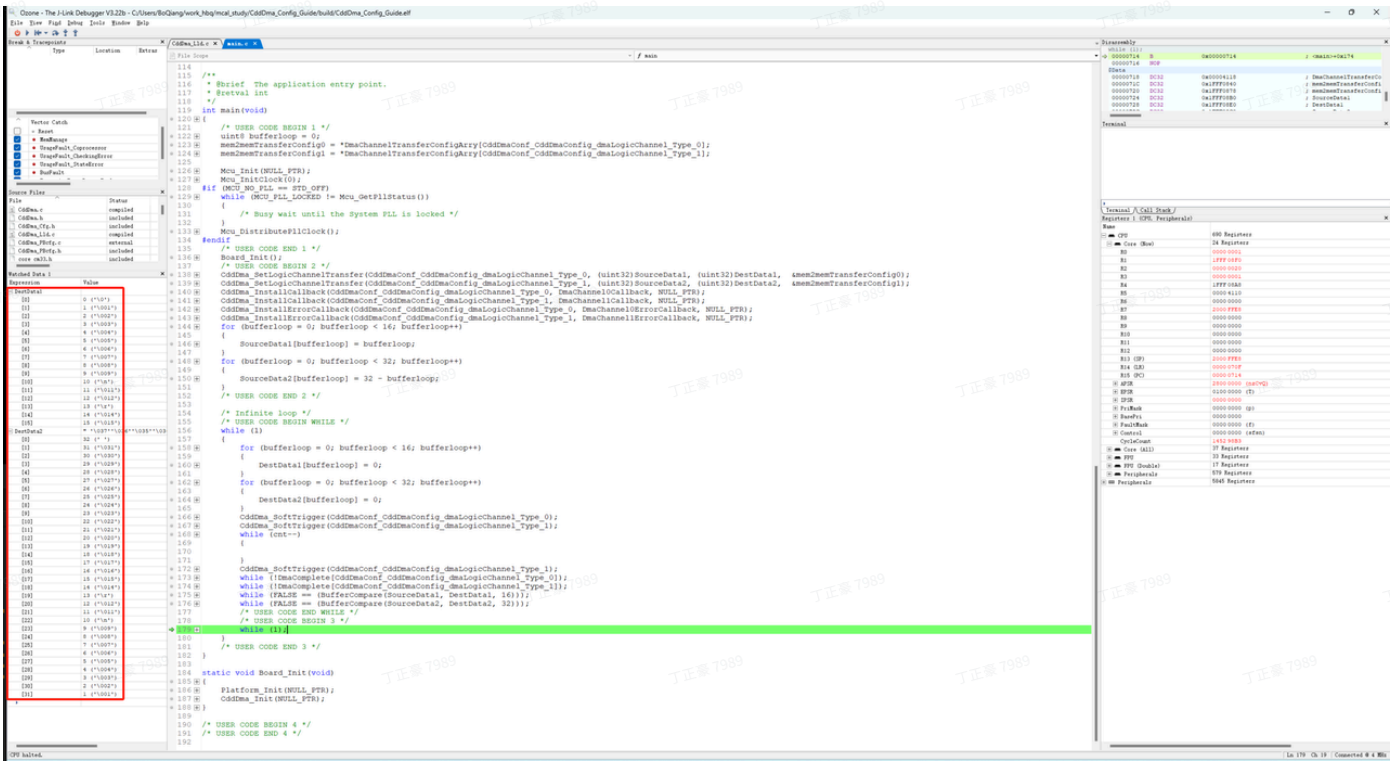
a. Memory数据源SourceData1和SourceData2赋值。



b. 目的地址Memory清零。



c. Memory to memory搬运结果



💡 因为通道0设置的trigger次数为1，所以只需要调用一次软件trigger即可触发trigger loop中断，而通道2设置的trigger次数为2，所以此时需要调用两次软件trigger才能触发该中断。

CddDma_PBcfg.c | main.c

board > CddDma_PBcfg.c > DmaChannelTransferConfig_0

```c
/**< Dma transfer configuration 0*/
CDDDMA_CONST static const CddDma_TransferConfigType DmaChannelTransferConfig_0 =
{
    /**< Memory address pointing to the source data. */
    .srcAddr = 0,
    /**< Memory address pointing to the destination data. */
    .destAddr = 0,
    /**< Source data transfer size. */
    .srcTransferSize = DMA_TRANSFER_SIZE_1_BYTE,
    /**< Destination data transfer size. */
    .destTransferSize = DMA_TRANSFER_SIZE_1_BYTE,
    /**< Sign-extended offset Bytes applied to the current source address to form the next-state value as each source read/write is completed. */
    .srcOffset = 1,
    /**< Sign-extended offset Bytes applied to the current destination address to form the next-state value as each source read/write is completed. */
    .destOffset = 1,
    /**< Last source address adjustment. */
    .srcLastAddrAdjust = 0,
    /**< Last destination address adjustment.
     *Note here it is only valid when ram reload feature is not enabled. */
    .destLastAddrAdjust = 0,
    /**< Source address modulo. */
    .srcModulo = DMA_MODULO_OFF,
    /**< Destination address modulo. */
    .destModulo = DMA_MODULO_OFF,
    /**< Number of bytes to be transferred in each service request of the channel. */
    .transferLoopByteCount = 16,
    /**< Number of major interation count
     * Note: This value is not used when channel link(loop/tirgger link) is enabled*/
    .TriggerCount = 1,
    /**< Disables the DMA channle automatic request after the trigger loop completes for the CTS*/
    .disableReqOnCompletion = TRUE,
    /**< If TRUE the intrrupt of Error, Major and HalfMajor will be disabled */
    .channelPollingMode = FALSE,
    /**< Used for ram reload feature, it shoule be configured by the API CddDma_ChannelRamReloadTransferConfig */
    .ramReloadEnable = FALSE,
    /**< The address of the next descriptor to be used, when ram reload feature is enabled.
     * Note: this value is not used when ram reload feature is disabled. It shoule be configured by the API CddDma_ChannelRamReloadTransferConfig */
    .ramReloadNextDescAddr = 0U,
    /**< Used to configures the interrupt request for RamReload configuration
     * Note: It shoule be configured by the API CddDma_ChannelRamReloadTransferConfig */
```

CddDma_PBcfg.c | main.c

board > CddDma_PBcfg.c > DmaChannelTransferConfig_0

```c
};
/**< Dma transfer configuration 1*/
CDDDMA_CONST static const CddDma_TransferConfigType DmaChannelTransferConfig_1 =
{
    /**< Memory address pointing to the source data. */
    .srcAddr = 0,
    /**< Memory address pointing to the destination data. */
    .destAddr = 0,
    /**< Source data transfer size. */
    .srcTransferSize = DMA_TRANSFER_SIZE_2_BYTE,
    /**< Destination data transfer size. */
    .destTransferSize = DMA_TRANSFER_SIZE_2_BYTE,
    /**< Sign-extended offset Bytes applied to the current source address to form the next-state value as each source read/write is completed. */
    .srcOffset = 2,
    /**< Sign-extended offset Bytes applied to the current destination address to form the next-state value as each source read/write is completed. */
    .destOffset = 2,
    /**< Last source address adjustment. */
    .srcLastAddrAdjust = 0,
    /**< Last destination address adjustment.
     *Note here it is only valid when ram reload feature is not enabled. */
    .destLastAddrAdjust = 0,
    /**< Source address modulo. */
    .srcModulo = DMA_MODULO_OFF,
    /**< Destination address modulo. */
    .destModulo = DMA_MODULO_OFF,
    /**< Number of bytes to be transferred in each service request of the channel. */
    .transferLoopByteCount = 32,
    /**< Number of major interation count
     * Note: This value is not used when channel link(loop/tirgger link) is enabled*/
    .TriggerCount = 2,
    /**< Disables the DMA channle automatic request after the trigger loop completes for the CTS*/
    .disableReqOnCompletion = TRUE,
    /**< If TRUE the intrrupt of Error, Major and HalfMajor will be disabled */
    .channelPollingMode = FALSE,
    /**< Used for ram reload feature, it shoule be configured by the API CddDma_ChannelRamReloadTransferConfig */
    .ramReloadEnable = FALSE,
    /**< The address of the next descriptor to be used, when ram reload feature is enabled.
     * Note: this value is not used when ram reload feature is disabled. It shoule be configured by the API CddDma_ChannelRamReloadTransferConfig */
    .ramReloadNextDescAddr = 0U,
    /**< Used to configures the interrupt request for RamReload configuration
```

这里设置多少就需要多少次trigger

```c
/* USER CODE BEGIN WHILE */
while (1)
{
    for (bufferloop = 0; bufferloop < 16; bufferloop++)
    {
        DestData1[bufferloop] = 0;
    }
    for (bufferloop = 0; bufferloop < 32; bufferloop++)
    {
        DestData2[bufferloop] = 0;
    }
    CddDma_SoftTrigger(CddDmaConf_CddDmaConfig_dmaLogicChannel_Type_0);
    CddDma_SoftTrigger(CddDmaConf_CddDmaConfig_dmaLogicChannel_Type_1);
    while (cnt--)
    {

    }
    CddDma_SoftTrigger(CddDmaConf_CddDmaConfig_dmaLogicChannel_Type_1);
    while (!DmaComplete[CddDmaConf_CddDmaConfig_dmaLogicChannel_Type_0]);
    while (!DmaComplete[CddDmaConf_CddDmaConfig_dmaLogicChannel_Type_1]);
    while (FALSE == (BufferCompare(SourceData1, DestData1, 16)));
    while (FALSE == (BufferCompare(SourceData2, DestData2, 32)));
    /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */
    while (1);
}
/* USER CODE END 3 */
```

通道1调用了一次软件trigger

通道2需要调用两次软件trigger

## 5. 文档历史

| 版本号 | 日期 | 修订记录 |
| --- | --- | --- |
| V1.0 | 2024.01.04 | 初始版本 |
| | | |