YTMicro MCAL应用 - CddI2c模块配置及实例

版本

Config Tool Version：2.7.8

ME0 MCAL version：2.3.0

前言

本文介绍云途 Mx 系列芯片 CddI2c 模块的基础配置与使用方法，并给出主机中断、主机 DMA 及从机通信示例。不同芯片的外设实例、引脚复用和功能支持可能不同，请结合目标芯片参考手册、对应 MCAL 版本说明及开发板原理图使用。

CDDI2c配置前提说明：

a. CddI2C一般依赖Mcu、Platform和Port模块，Mcu模块里面需要配置外设时钟，Platform里面可以配置外设中断，Port配置SCL、SDA 等引脚资源 ，使用 DMA 传输时，还需配置 CddDma 及对应的 DMA 请求源、通道和中断。

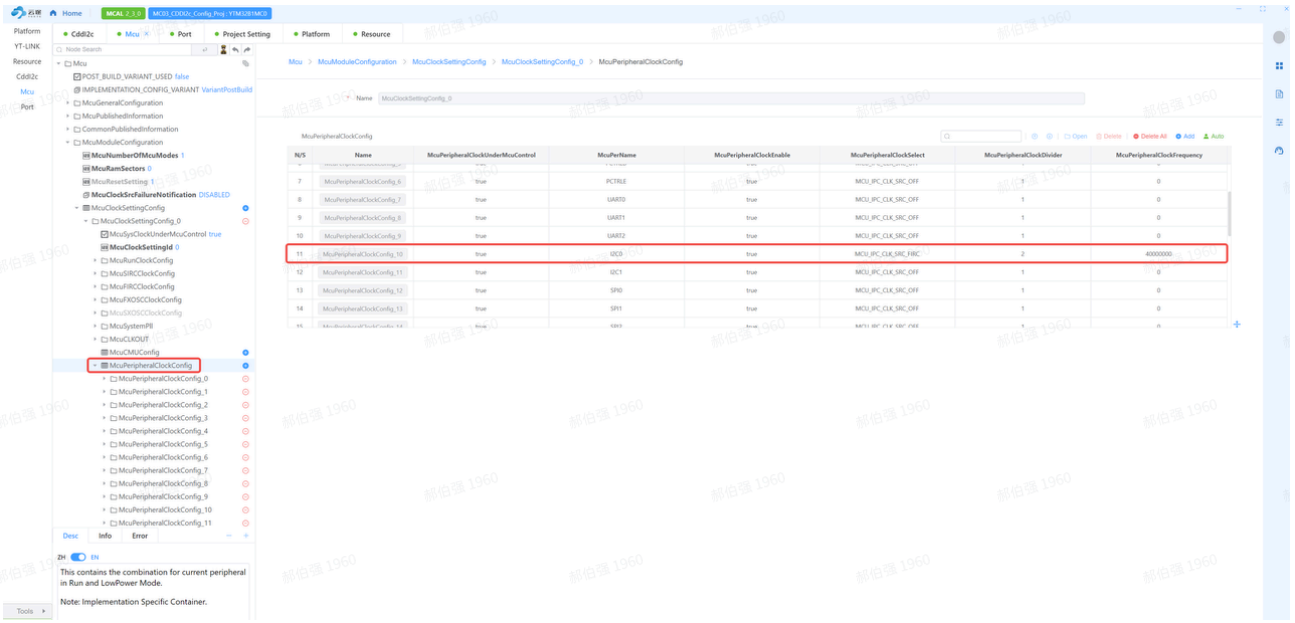
b. Port、Mcu、CddDMA和Platform的默认配置参考Port篇、Setup篇、CddDMA篇和Mcu篇介绍，建议先完成时钟、引脚和中断配置，再配置 CddI2c 通道。

c. 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。

基础模块配置介绍

1.1 时钟配置

i. MC03/MD14/MD24/ME05/HA01在MCU模块中，找到如图所示的I2C时钟配置项，在此处可以配置I2C模块的时钟源和分频系数，此处得到的是 I2C 外设输入时钟，不是 SCL 总线速率；



Tips:

- MC03和MD24 I2C 时钟频率最高限制40MHz，如下所示：

Table 13.1 continued from previous page

No.	Module Name	Offset	Bus Clock	Bus Clock Enable	Peripheral Functional Clock	Software Reset	Additional Clock	MAX Functional Clock Frequency
26	-	0x068	-	-	-	-	-	-
27	UART0	0x06C	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 40Mhz
28	UART1	0x070	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 40Mhz
29	UART2	0x074	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 40Mhz
30	-	0x078	-	-	-	-	-	-
31	-	0x07C	-	-	-	-	-	-
32	-	0x080	-	-	-	-	-	-
33	-	0x084	-	-	-	-	-	-
34	-	0x088	-	-	-	-	-	-
35	I2C0	0x08C	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 40Mhz
36	I2C1	0x090	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 40Mhz
37	-	0x094	-	-	-	-	-	-
38	-	0x098	-	-	-	-	-	-
39	-	0x09C	-	-	-	-	-	-
40	-	0x0A0	-	-	-	-	-	-
41	-	0x0A4	-	-	-	-	-	-
42	SPI0	0x0A8	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 40Mhz
43	SPI1	0x0AC	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 40Mhz
44	SPI2	0x0B0	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 40Mhz

- HA01 I2C时钟频率最高限制50MHz，如下所示：

Table 15.1 continued from previous page

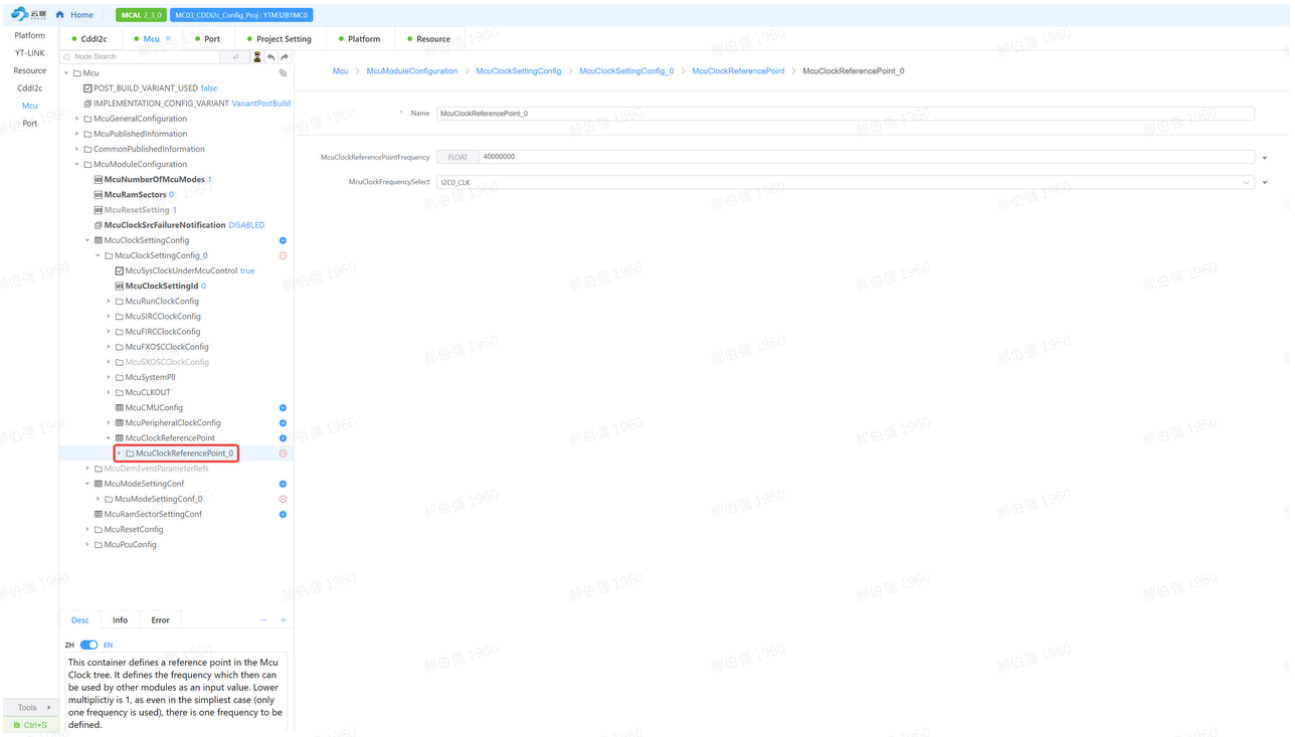
No.	Module Name	Offset	Bus Clock	Bus Clock Enable	Peripheral Functional Clock	Software Reset	Additional Clock	MAX Functional Clock Frequency
21	PCTRLD	0x054	SLOW_BUS_CLK	-	-	YES	-	-
22	PCTRLE	0x058	SLOW_BUS_CLK	-	-	YES	-	-
23	-	0x05C	-	-	-	-	-	-
24	-	0x060	-	-	-	-	-	-
25	-	0x064	-	-	-	-	-	-
26	-	0x068	-	-	-	-	-	-
27	LINFlexD0	0x06C	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 50MHz
28	LINFlexD1	0x070	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 50MHz
29	LINFlexD2	0x074	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 50MHz
30	LINFlexD3	0x078	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 50MHz
31	LINFlexD4	0x07C	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 50MHz
32	LINFlexD5	0x080	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 50MHz
33	LINFlexD6	0x084	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 50MHz
34	LINFlexD7	0x088	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 50MHz
35	LINFlexD8	0x08C	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 50MHz
36	LINFlexD9	0x090	SLOW_BUS_CLK	OFF	YES	YES	-	< 50MHz
37	I2C0	0x094	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 50MHz
38	I2C1	0x098	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 50MHz
39	I2C2	0x09C	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 50MHz
40	I2C3	0x0A0	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 50MHz
41	I2C4	0x0A4	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 50MHz
42	SPI0	0x0A8	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 50MHz
43	SPI1	0x0AC	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 50MHz
44	SPI2	0x0B0	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 50MHz
45	SPI3	0x0B4	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 50MHz
46	SPI4	0x0B8	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 50MHz
47	SPI5	0x0BC	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 50MHz
48	SPI6	0x0C0	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 50MHz
49	SPI7	0x0C4	SLOW_BUS_CLK	OFF	YES	YES	-	≤ 50MHz

MD14/ME05 I2C时钟频率无最大限制。

Table 13.1 continued from previous page

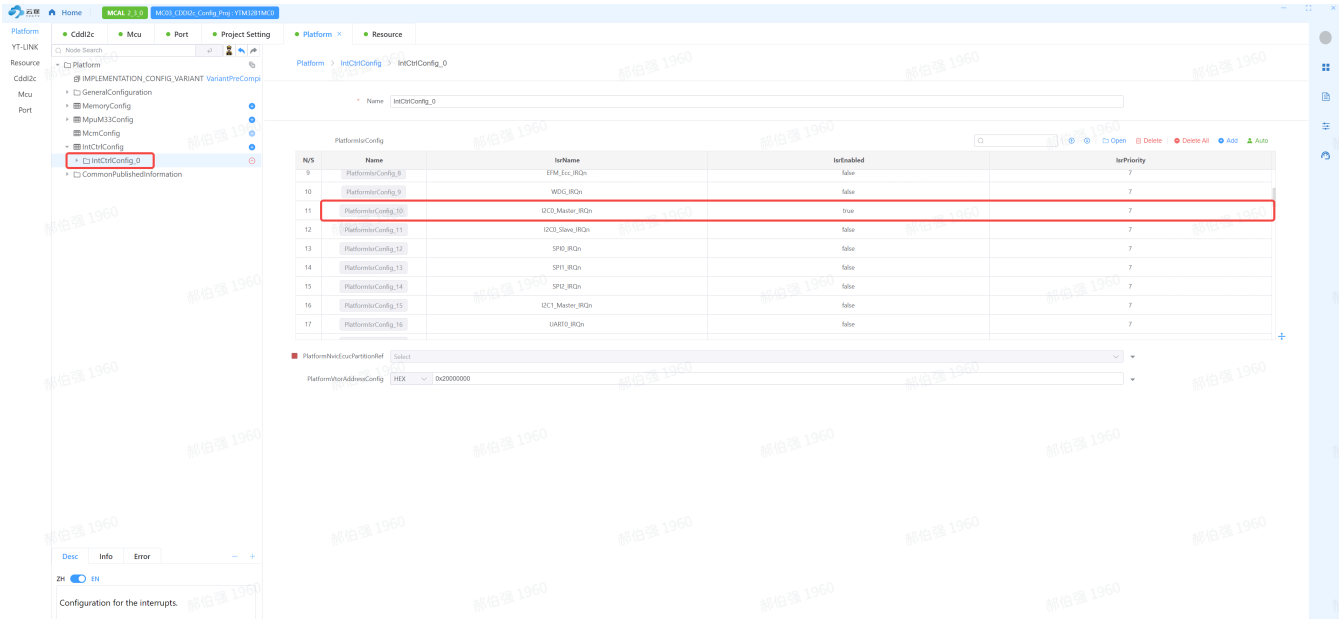
No.	Module Name	Offset	Bus Clock	Bus Clock Enable	Peripheral Functional Clock	Software Reset	Additional Clock
10	-	0x028	-	-	-	-	-
11	-	0x02C	-	-	-	-	-
12	-	0x030	-	-	-	-	-
13	-	0x034	-	-	-	-	-
14	-	0x038	-	-	-	-	-
15	TRACE	0x03C	FAST_BUS_CLK	-	YES	-	-
16	EFM	0x040	SLOW_BUS_CLK	ON	-	YES	-
17	GPIO	0x044	CORE_CLK	OFF	YES	YES	-
18	PCTRLA	0x048	SLOW_BUS_CLK	-	-	YES	-
19	PCTRLB	0x04C	SLOW_BUS_CLK	-	-	YES	-
20	PCTRLC	0x050	SLOW_BUS_CLK	-	-	YES	-
21	PCTRLD	0x054	SLOW_BUS_CLK	-	-	YES	-
22	PCTRLE	0x058	SLOW_BUS_CLK	-	-	YES	-
23	-	0x05C	-	-	-	-	-
24	-	0x060	-	-	-	-	-
25	-	0x064	-	-	-	-	-
26	-	0x068	-	-	-	-	-
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	-
30	-	0x078	-	-	-	-	-
31	-	0x07C	-	-	-	-	-
32	-	0x080	-	-	-	-	-
33	SENT0	0x084	SLOW_BUS_CLK	OFF	YES	YES	-
34	-	0x088	-	-	-	-	-
35	I2C0	0x08C	SLOW_BUS_CLK	OFF	YES	YES	-
36	I2C1	0x090	SLOW_BUS_CLK	OFF	YES	YES	-
37	-	0x094	-	-	-	-	-
38	-	0x098	-	-	-	-	-
39	-	0x09C	-	-	-	-	-
40	-	0x0A0	-	-	-	-	-
41	-	0x0A4	-	-	-	-	-

ii. 在McuClockReferencePoint中定义一个I2C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；



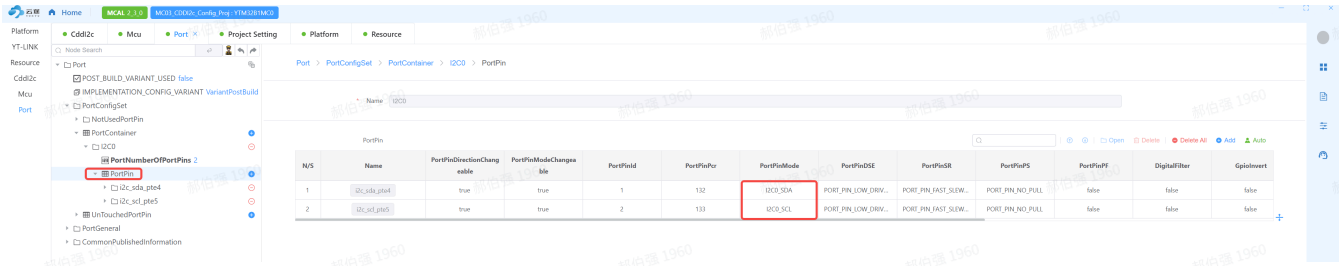
1.2 中断配置

在Platform中的中断控制配置项中配置I2C的中断，若使用OS，则中断管理将在OS中配置，此处使能了 I2C0 master 的中断，优先级为7；



1.3 引脚配置

在Port模块中配置I2C SCL和SDA引脚功能：I2C0_SDA, I2C0_SCL；



2. CddI2C配置介绍

2.1 如何配置IIC模块

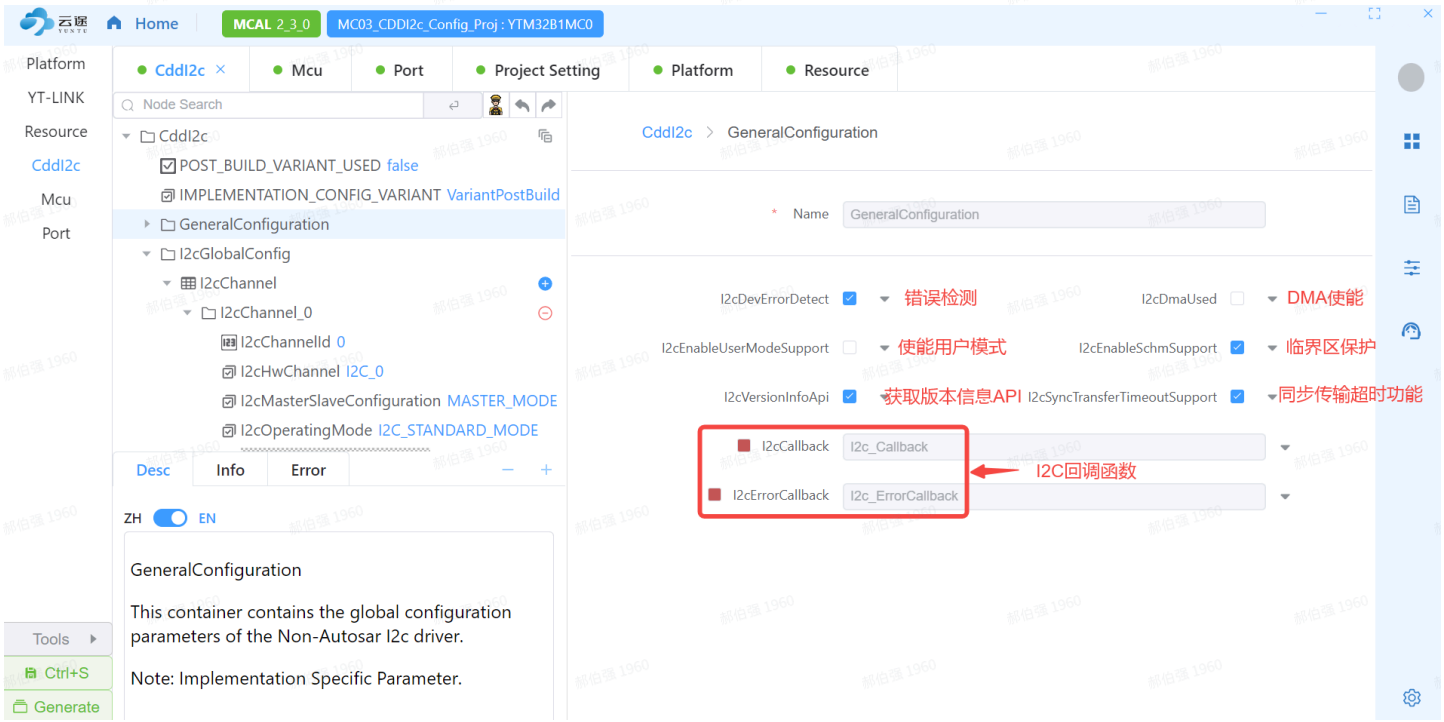
2.1.1 框架说明

此处主要介绍了整个I2C模块的框架，需要我们着重注意的是I2cGlobalConfig。



2.1.2 GeneralConfiguration

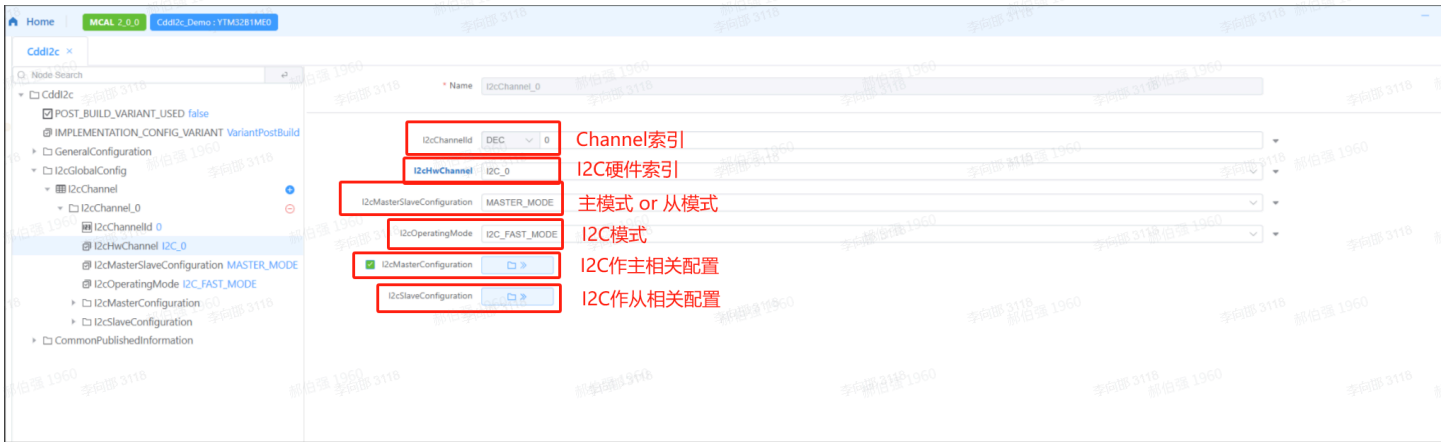
此处是通用的一些配置，按照需求配置即可。



2.2 I2cGlobalConfig

2.2.1 I2cChannel

此处是设置I2C模块的具体参数，如果使用多个I2C模块，则建多个IIC即可

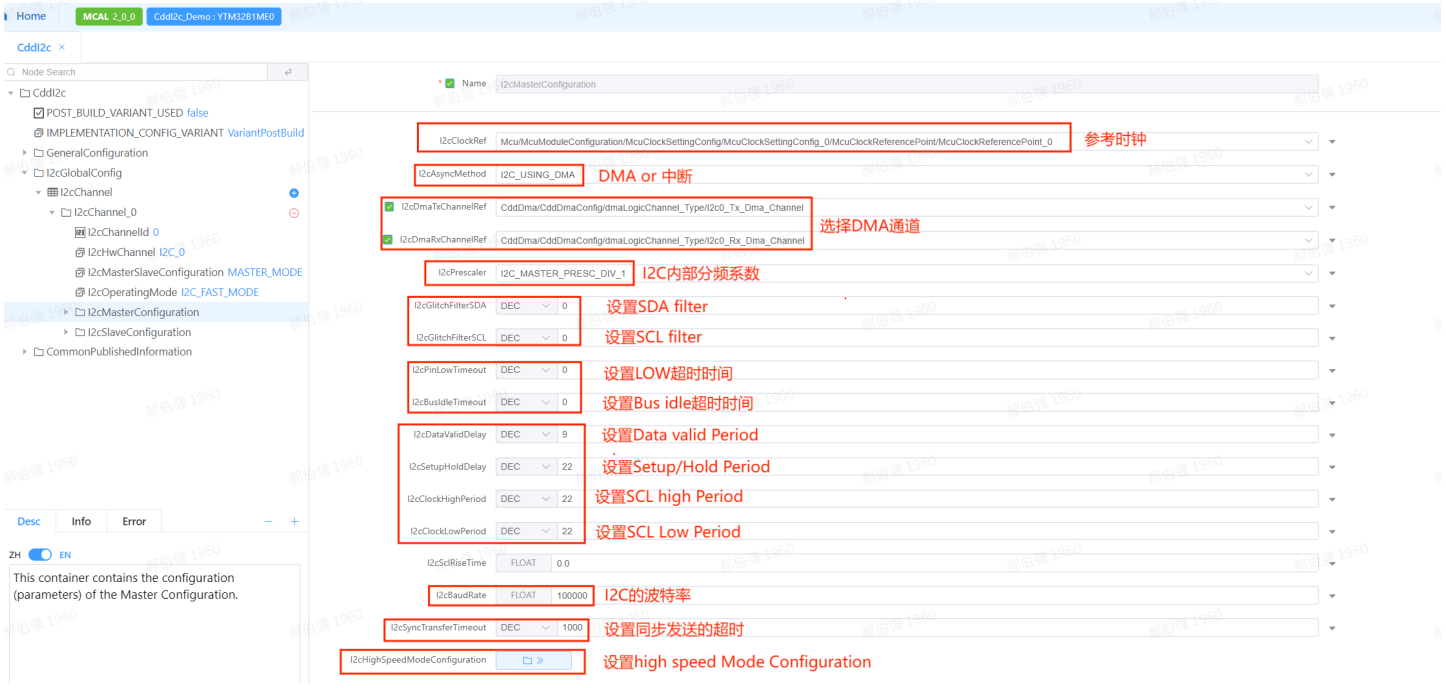


Tips:

I2C Operate Mode： I2C_Standard_Mode、I2C_Fast_Mode、I2C_FASTPLUS_MODE、I2C_HighSpeed_Mode

2.2.2 I2cMasterConfiguration

此处配置I2C作为主节点时候的参数（需要在I2Cchannel界面选择主模式）。



Tips:

SDA Filter、SCLFilter对应下图寄存器，小于配置值*模块时钟的信号会被过滤掉

Table 21.8: I2C MFLTCFG Register Description

Field	Function
27 - 24 FLTSCL	**SCL Filter Control** Control digital filter on SCL line SCL digital filter use undivided I2C function clock Noise less or equal FLTSCL * function period will be filtered
19 - 16 FLTSDA	**SDA Filter Control** Control digital filter on SDA line SDA digital filter use undivided I2C function clock Noise less or equal FLTSDA * function period will be filtered

Tips: 如何计算IIC的波特率，按照下图进行计算

```
Calculated as Frequency/((((CLKLO+CLKHI+2)*2^PRESCALER)+ROUNDDOWN((2+FILTSCL)/2^PRESCALER))
```

Tips:

IDLE Timeout、PINLow Timeout对应下图寄存器，计算公式如图所示

Table 21.15: I2C TOCFG Register Description

Field	Function
27 - 16 IDLE	**Line Idle Timeout Control** Line idle flag set when SDA and SCL line high for (IDLE) * divided function clock, after line idle, master can send out START.
15 SDA	**SDA Line Low Detect** Set this bit to enable SDA line low detection 0b - Line low only detect SCL 1b - Line low detect SDA and SCL
11 - 0 LOW	**Line Low Timeout Control** Line low period is (LOW * 256) * divided function clock

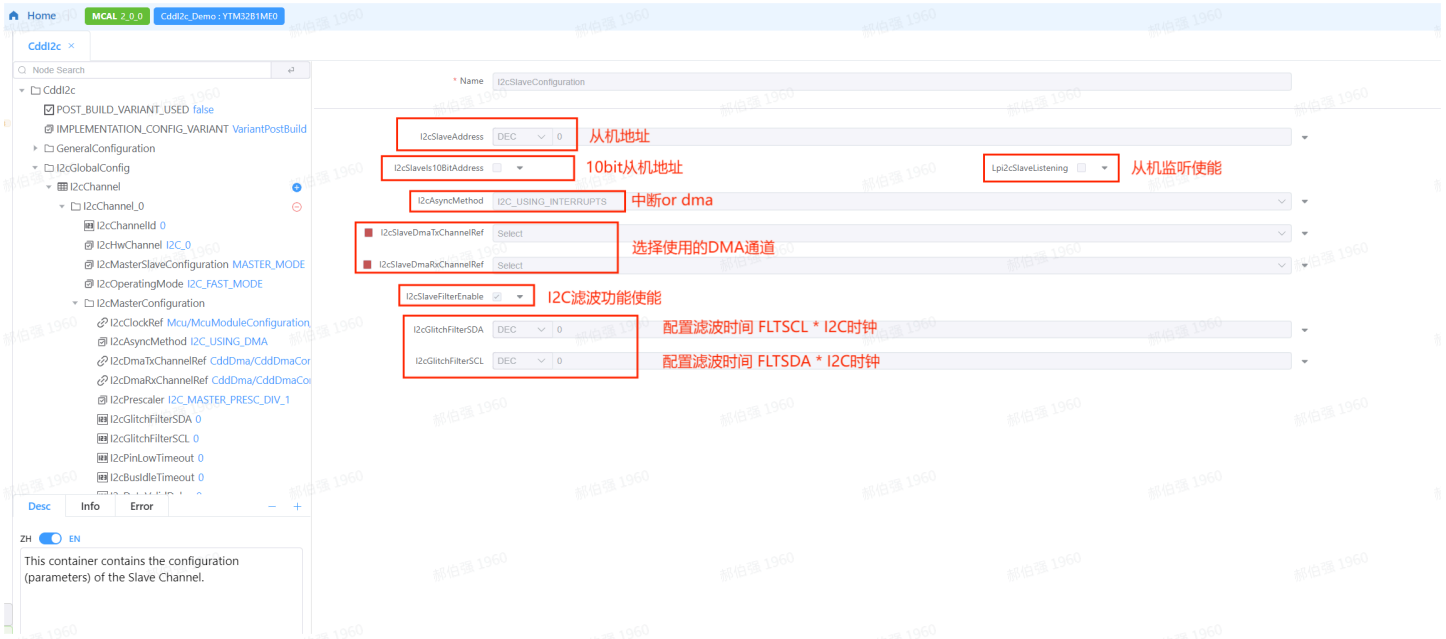
2.2.2.1 I2cHighSpeedModeConfiguration

此处配置HIghSpeedMode时的波特率，需要在I2cChannel界面选择HIghSpeedMode。



2.2.3 I2cSlaveConfiguration

此处配置I2C作为从节点时候的参数（需要在I2Cchannel界面选择从模式）。



2.3 CommonPublishedInformation

此处主要是版本信息、AutoSar标准、模块ID、供应商ID信息。需要注意的是，I2C并不是标准的MCAL模块，属于

复杂驱动，这里ModuleID为255。

3. I2C模块应用实例

3.1 中断方式实例

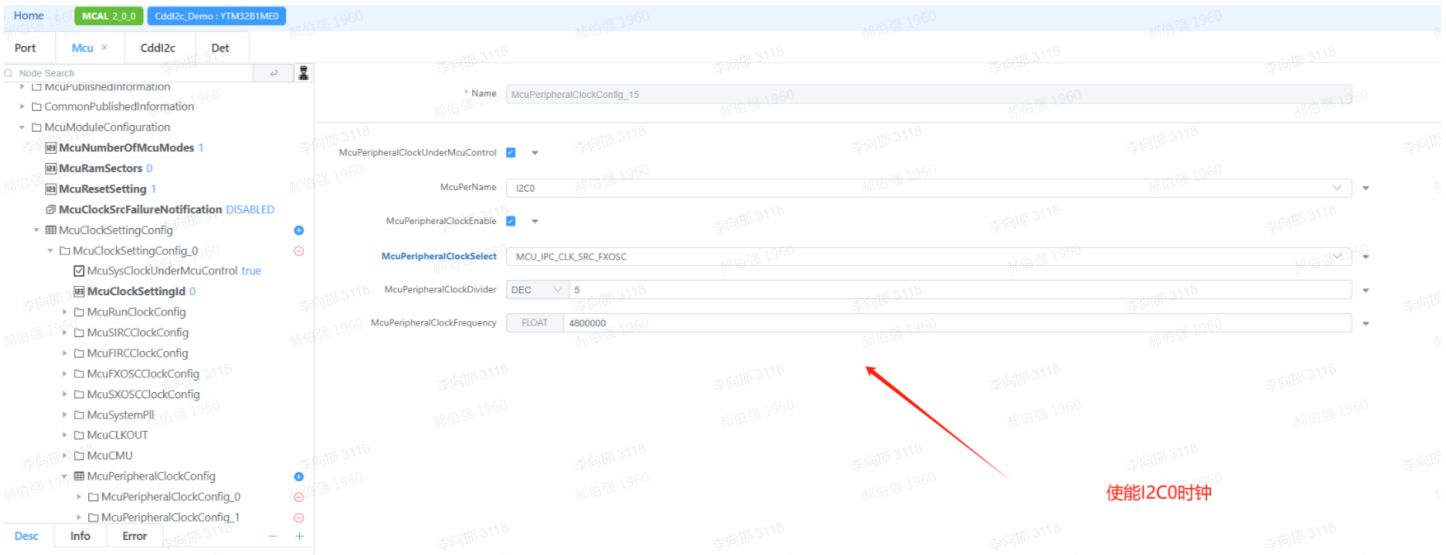
- 基于ME0 Demo板，使用I2C0做主机和Demo板上的Eeprom进行通信。
- 设置通信波特率为100K。从地址0x50，写入数据为0x18, 0x20, 0x31, 0x47, 0x5, 0x6, 0x7
- Port上选择

  PTA3 -- I2C0_SCL

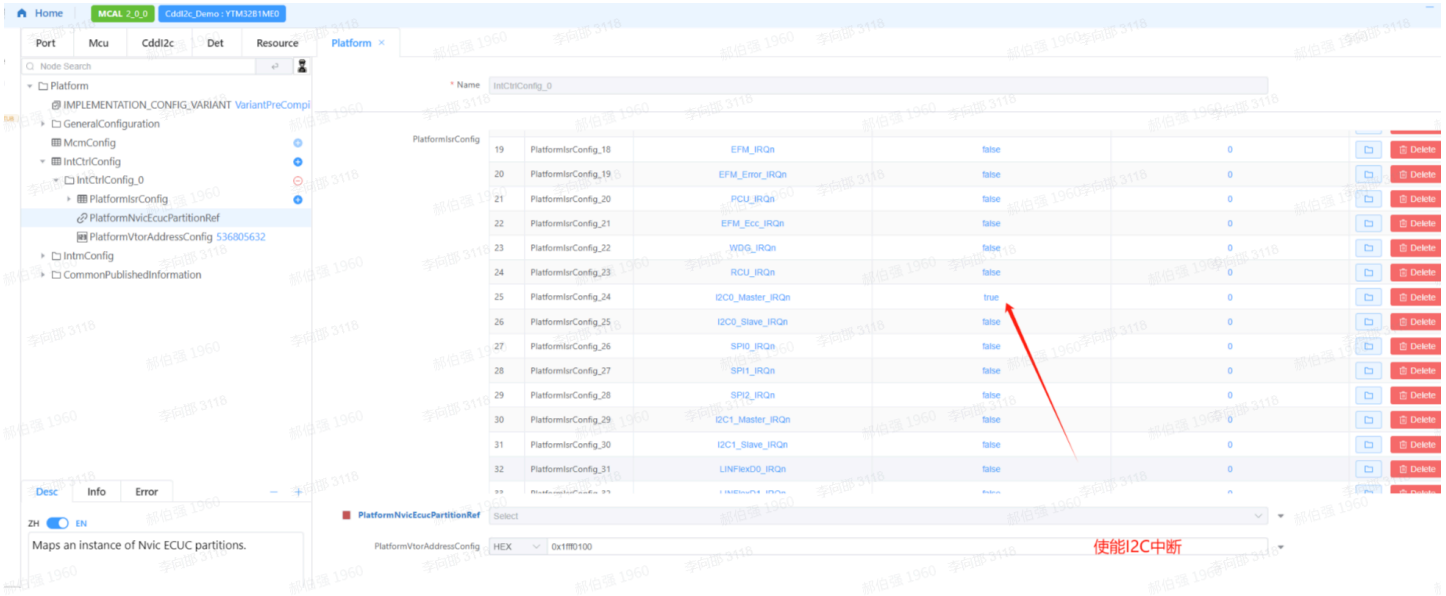
  PTA2 -- I2C0_SDA

  PTB11 -- EEP_WP Eeprom芯片唤醒脚
- 逻辑分析仪抓取PTA3 、PTA2上的波形，软件验证读写buffer一致。

3.1.1 时钟配置

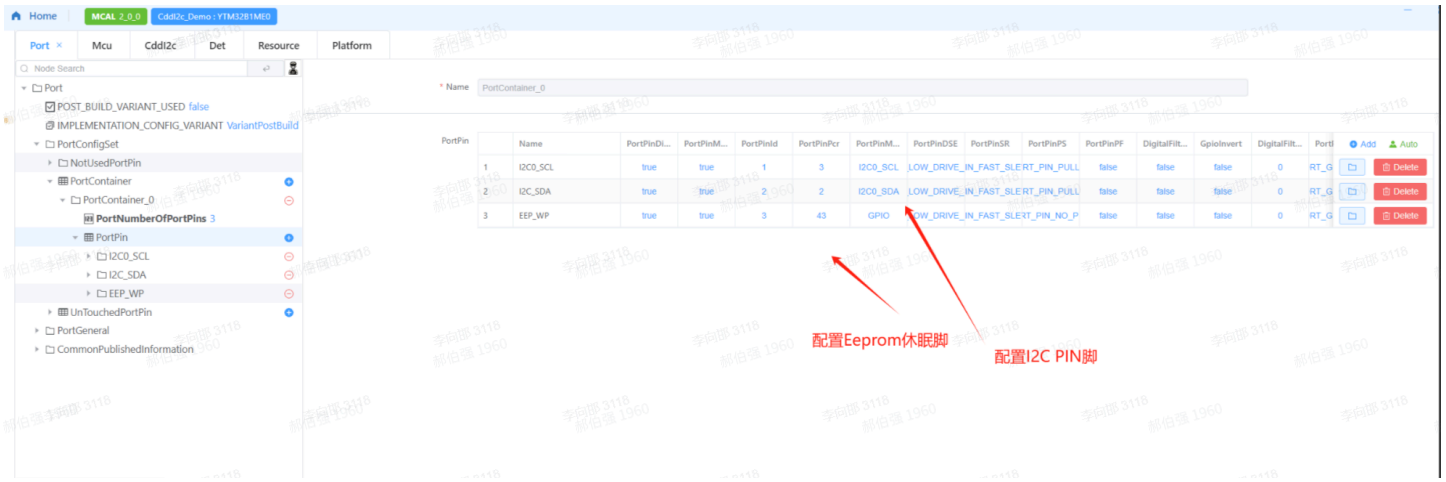


3.1.2 中断配置

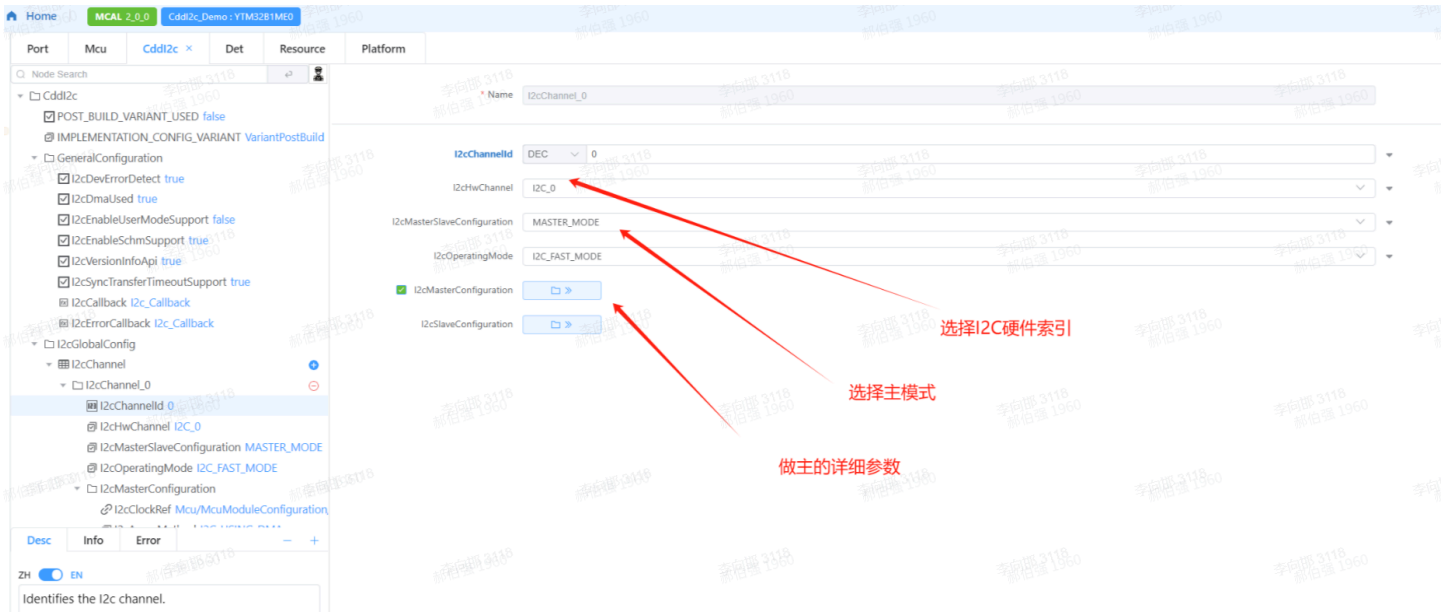


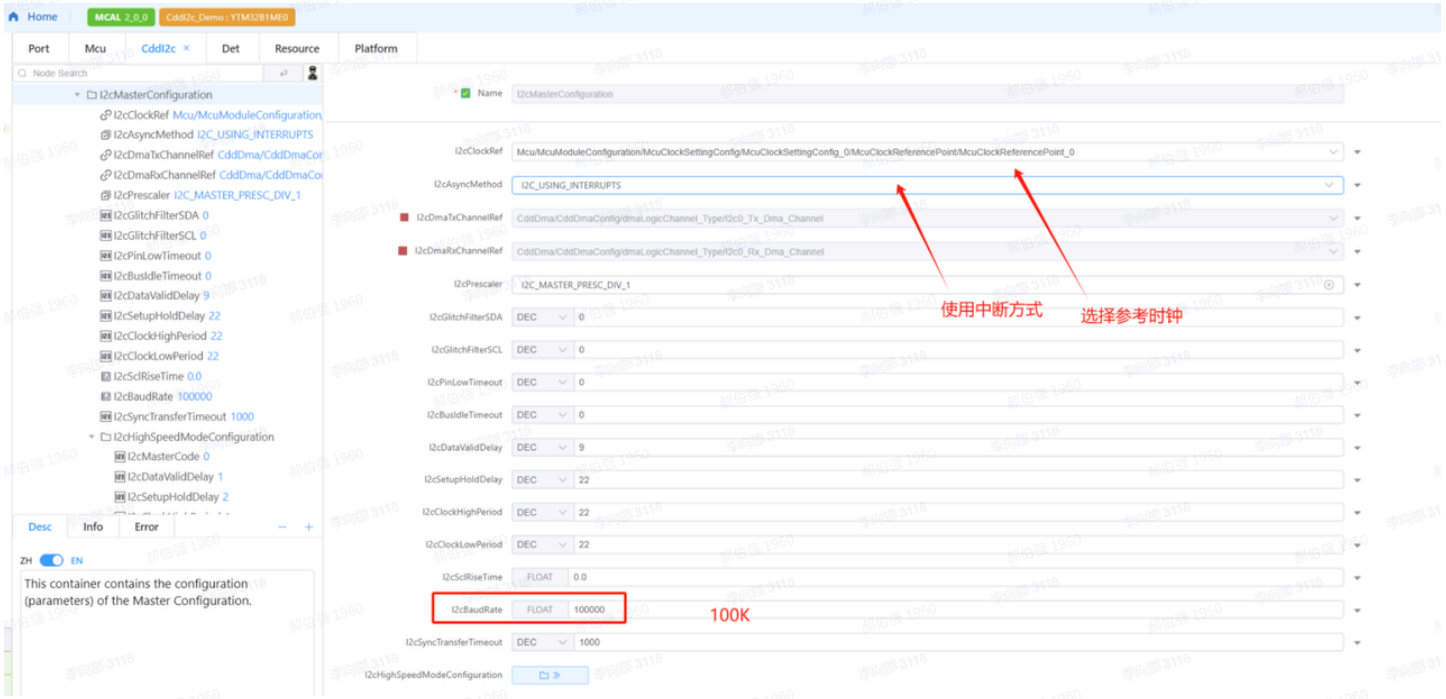
3.1.3 Port配置

这里配置PTA3、PTA2、PTB11。



3.1.4 配置I2C模块





3.1.5 API说明

这里我们用同步接口往Eeprom写数据，用异步接口读取之前写到的数据。

```
#endif
    /* USER CODE END 1 */
    Board_Init();
    /* USER CODE BEGIN 2 */
    /* USER CODE END 2 */

    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    /*write data to EEPROM in EVB with the I2C0 */
#if (I2C_TX_MODE == I2C_TRANS_IN_SYNCMODE)
    CddI2c_SyncModeTransfer(CddI2cConf_I2cChannel_I2cChannel_0, &CDDI2cRequestTx);

    TempRav = CddI2c_GetTransferStatus(CddI2cConf_I2cChannel_I2cChannel_0);
#else
    CddI2c_AsyncModeTransfer(CddI2cConf_I2cChannel_I2cChannel_0, &CDDI2cRequestTx);
    do
    {
        TempRav = CddI2c_GetTransferStatus(CddI2cConf_I2cChannel_I2cChannel_0);
    } while (I2C_CHN_COMPLETED != TempRav);
#endif

    /*delay 10ms*/
    i = 50000;
    while (i--);

    /*Read data from EEPROM in EVB with the I2C0*/
#if (I2C_RX_MODE == I2C_TRANS_IN_ASYNCMODE)
    CDDI2cRequestTx.BufferSize = 1;
    CddI2c_AsyncModeTransfer(CddI2cConf_I2cChannel_I2cChannel_0, &CDDI2cRequestTx);
    do
    {
        TempRav = CddI2c_GetTransferStatus(CddI2cConf_I2cChannel_I2cChannel_0);
    } while (I2C_CHN_COMPLETED != TempRav);

    CddI2c_AsyncModeTransfer(CddI2cConf_I2cChannel_I2cChannel_0, &CDDI2cRequestRx);
    do
    {
        TempRav = CddI2c_GetTransferStatus(CddI2cConf_I2cChannel_I2cChannel_0);
    } while (I2C_CHN_COMPLETED != TempRav);
#else
    CDDI2cRequestTx.BufferSize = 1;
    CddI2c_SyncModeTransfer(CddI2cConf_I2cChannel_I2cChannel_0, &CDDI2cRequestTx);
    CddI2c_SyncModeTransfer(CddI2cConf_I2cChannel_I2cChannel_0, &CDDI2cRequestRx);

    TempRav = CddI2c_GetTransferStatus(CddI2cConf_I2cChannel_I2cChannel_0);

#endif
    /* Verify the recieve data is right or not*/
    while (BufferCompare(&TestTxBuffer[1], &TestRxBuffer[0], 7) == FALSE);

    CddI2c_DeInit();
```

delay一段时间

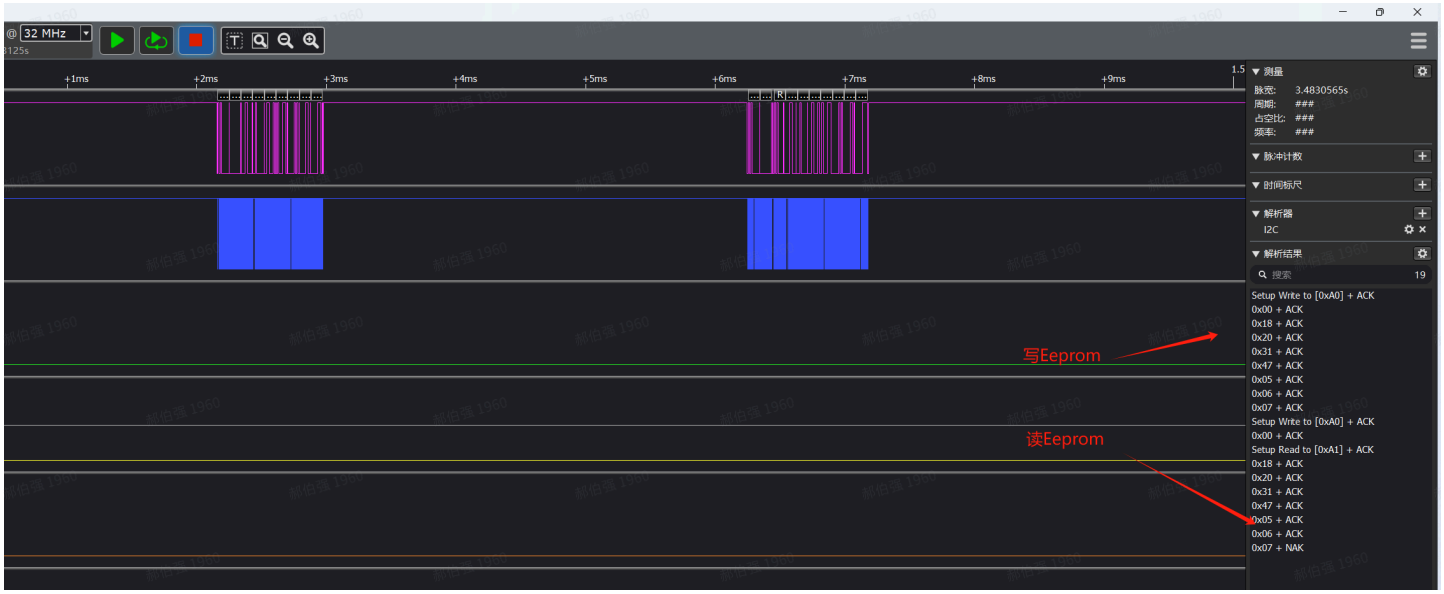
使用同步接口写数据

使用异步接口读数据

校验读写buffer一致

3.1.6 现象

线上抓到的数据，与预期一致



3.2 DMA方式实例

- 基于MC0 Demo板，使用I2C1做主机和Demo板上的Eeprom进行通信。
- 设置通信波特率为100K。从地址0x50，写入数据为0x19, 0x28, 0x37, 0x46, 0x55, 0x64, 0x73
- Port上选择

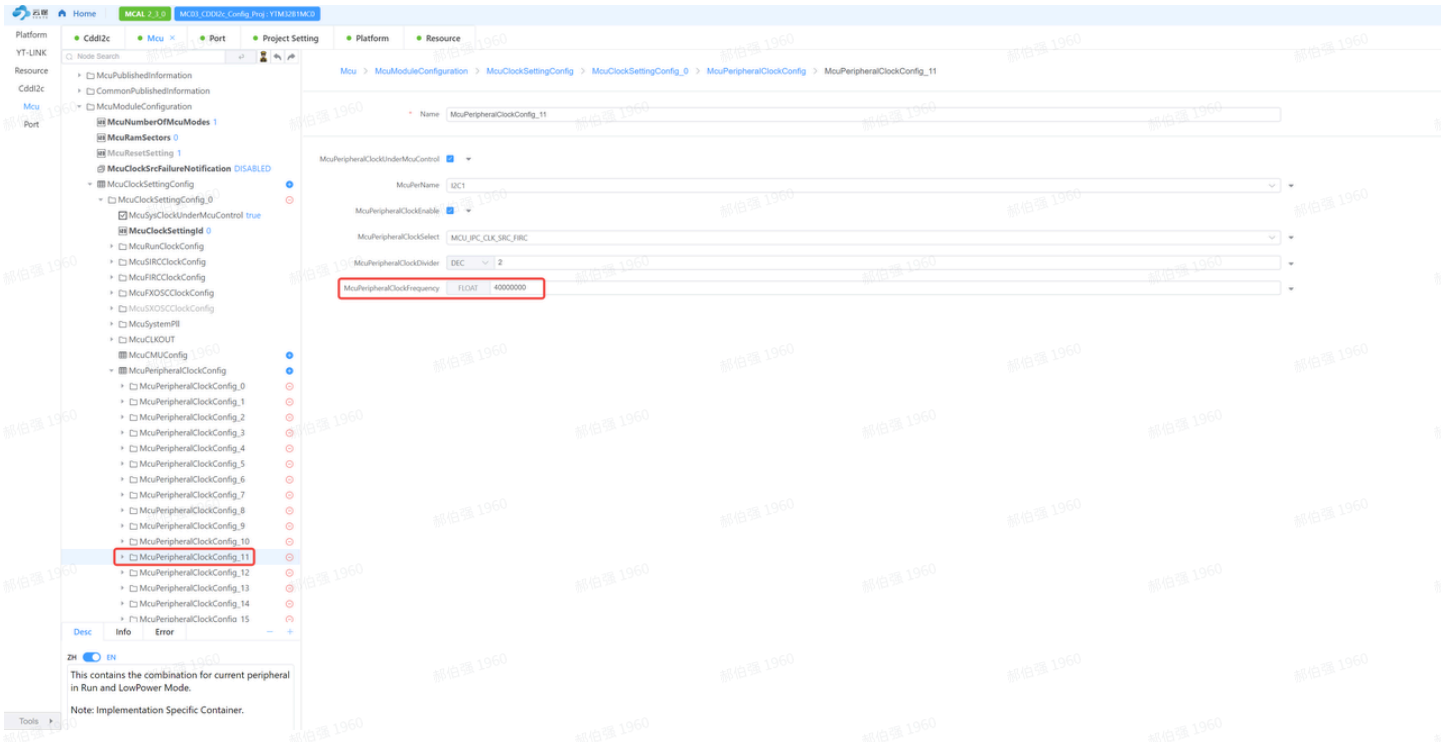
PTD1 -- I2C1_SCL

PTD0 -- I2C1_SDA

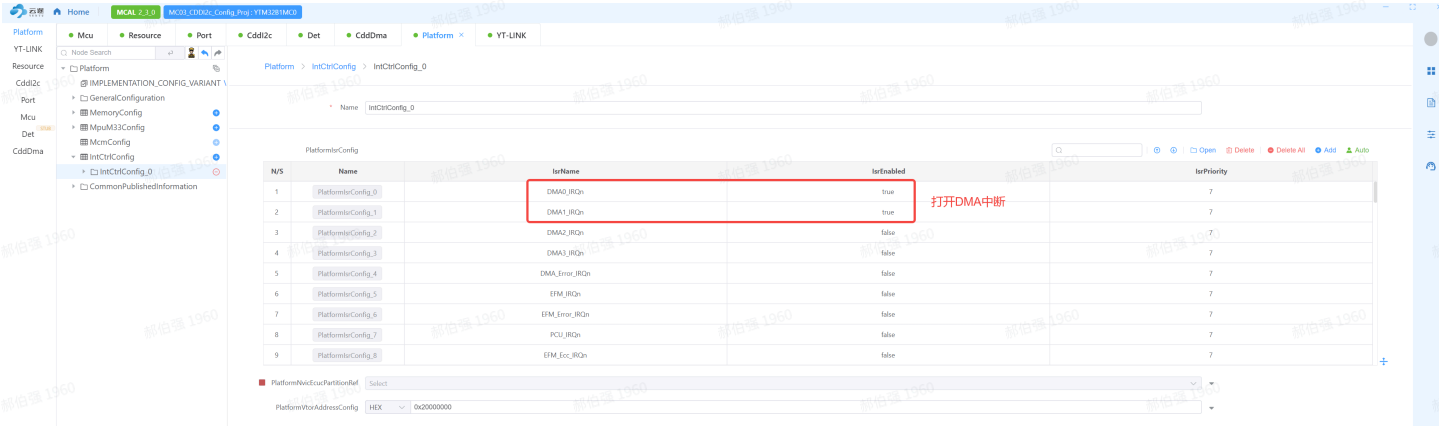
PTB11 -- EEP_WP Eeprom芯片唤醒脚

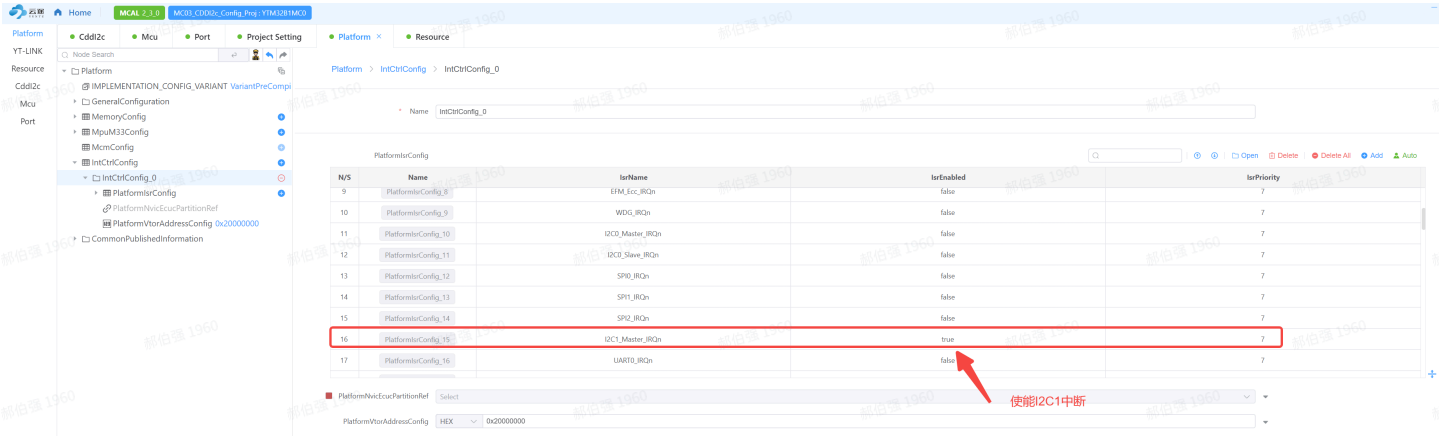
- 逻辑分析仪抓取PTD1 、PTD0上的波形，软件验证读写buffer一致。

3.2.1 时钟配置



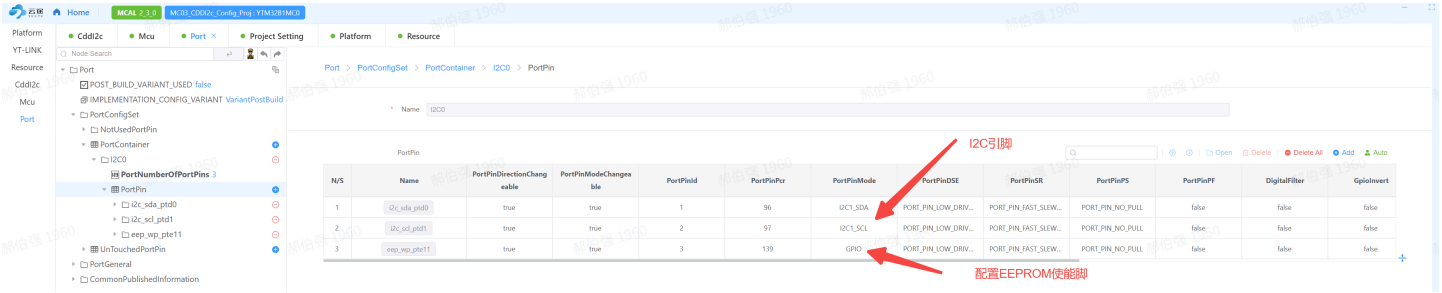
3.2.2 中断配置



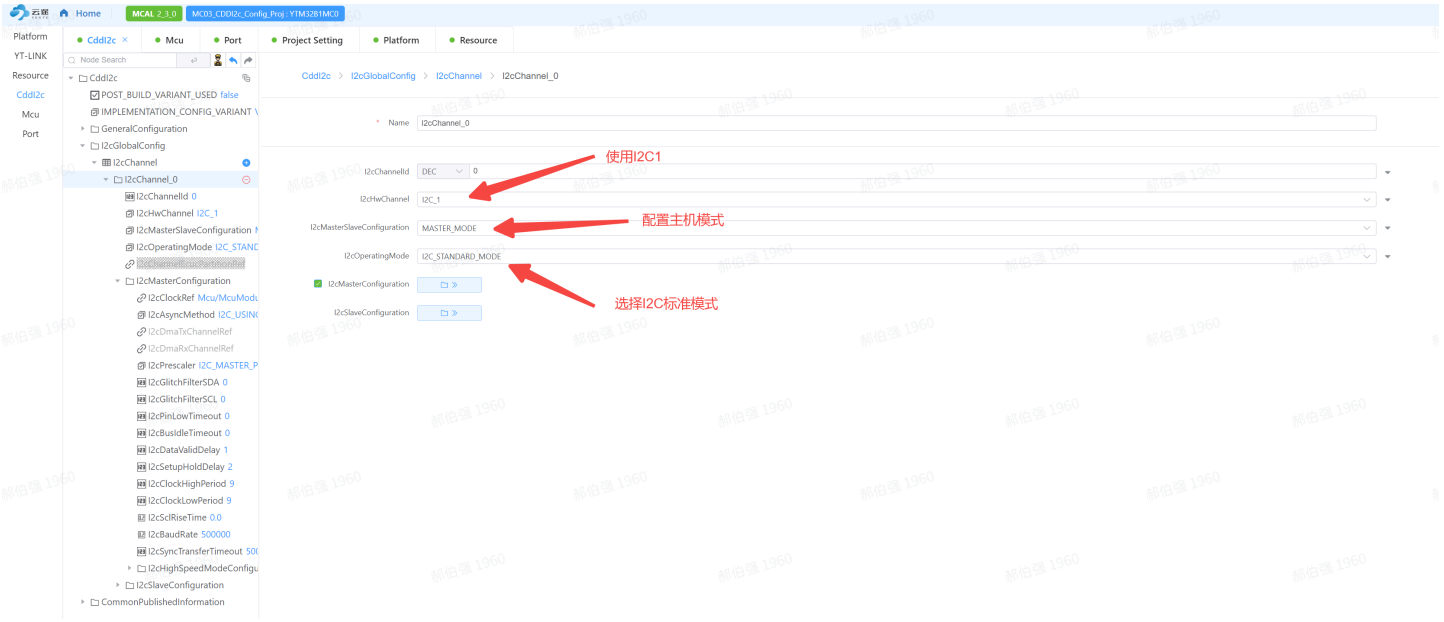


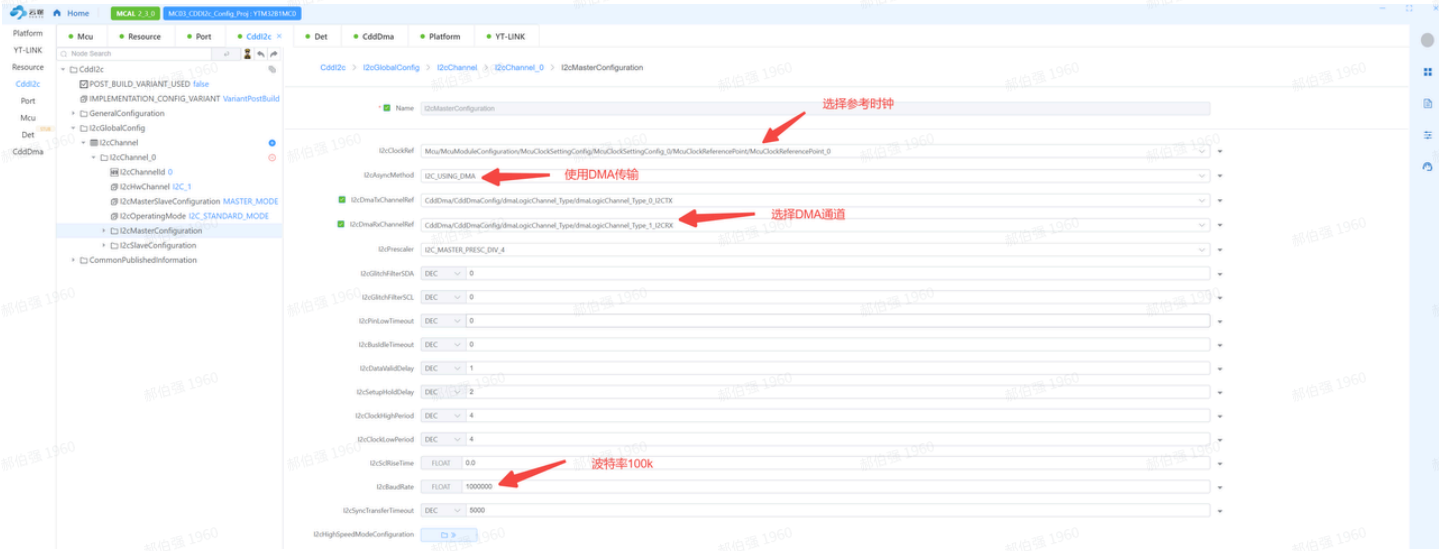
3.2.3 Port配置

这里配置PTD1、PTD0、PTE11。



3.2.4 配置I2C模块





3.2.5 API说明

这里我们用同步接口往Eeprom写数据，用异步接口读取之前写到的数据。

```
#if (I2C_TX_MODE == I2C_TRANS_IN_SYNCMODE)
    CddI2c_SyncModeTransfer(CddI2cConf_I2cChannel_I2cChannel_0, &CDDI2cRequestTx);
    TempRav = CddI2c_GetTransferStatus(CddI2cConf_I2cChannel_I2cChannel_0);
    do
    {
        TempRav = CddI2c_GetTransferStatus(CddI2cConf_I2cChannel_I2cChannel_0);
    } while (I2C_CHN_COMPLETED != TempRav);
#else
    CddI2c_AsyncModeTransfer(CddI2cConf_I2cChannel_I2cChannel_0, &CDDI2cRequestTx);
#endif
    /*delay 10ms*/
    i = 50000;
    while (i--)
    {
        __ASM("NOP");
    }
    /*Read data from EEPROM in EVB with the I2C1*/
#if (I2C_RX_MODE == I2C_TRANS_IN_ASYNCMODE)
    CDDI2cRequestTx.BufferSize = 1;
    CddI2c_AsyncModeTransfer(CddI2cConf_I2cChannel_I2cChannel_0, &CDDI2cRequestTx);
    do
    {
        TempRav = CddI2c_GetTransferStatus(CddI2cConf_I2cChannel_I2cChannel_0);
    } while (I2C_CHN_COMPLETED != TempRav);
    CddI2c_AsyncModeTransfer(CddI2cConf_I2cChannel_I2cChannel_0, &CDDI2cRequestRx);
    do
    {
        TempRav = CddI2c_GetTransferStatus(CddI2cConf_I2cChannel_I2cChannel_0);
    } while (I2C_CHN_COMPLETED != TempRav);
#else
    CDDI2cRequestTx.BufferSize = 1;
    CddI2c_SyncModeTransfer(CddI2cConf_I2cChannel_I2cChannel_0, &CDDI2cRequestTx);
    CddI2c_SyncModeTransfer(CddI2cConf_I2cChannel_I2cChannel_0, &CDDI2cRequestRx);
    TempRav = CddI2c_GetTransferStatus(CddI2cConf_I2cChannel_I2cChannel_0);
#endif
    /* Verify the recieve data is right or not*/
    while (BufferCompare(&TestTxBuffer[1], &TestRxBuffer[0], 7) == FALSE);
    CddI2c_DeInit();
    while (1)
```

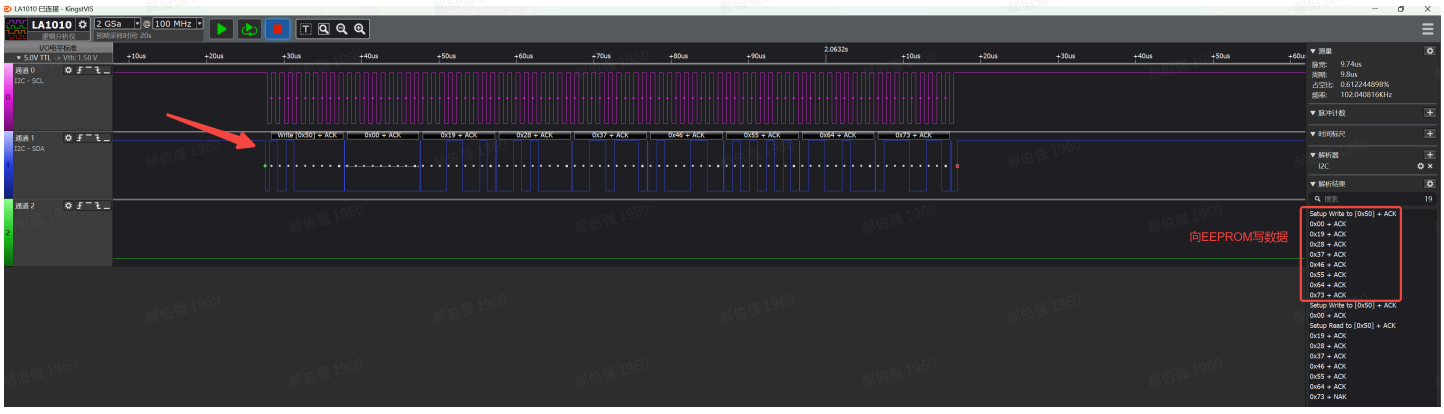
使用同步方式发送数据并等待发送完成

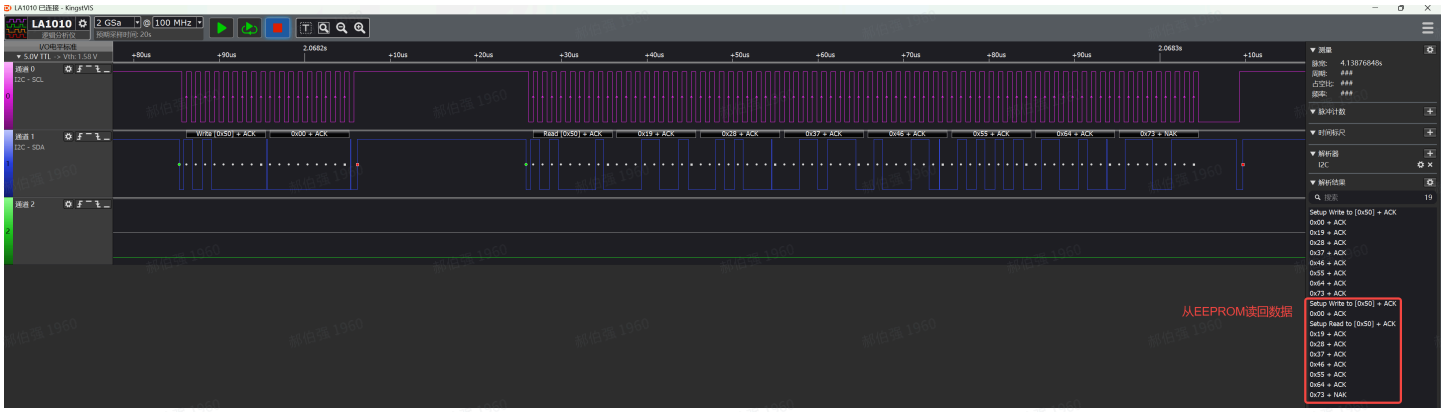
使用异步方式接收数据并等待接收完成

比较发送数据和收到的数据

3.2.6 现象

线上抓到的数据，与预期一致





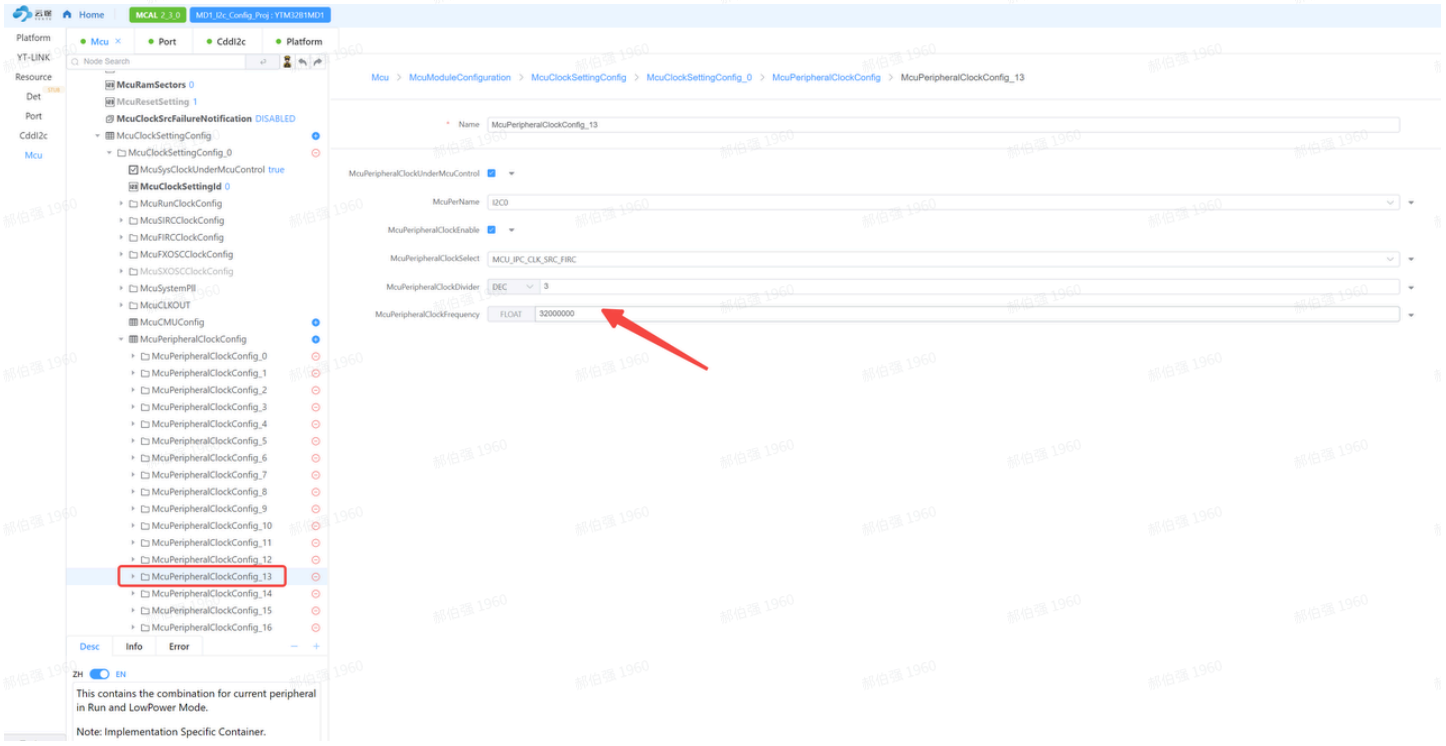
3.3 Slave方式实例

- 基于MC03和MD1 Demo板，使用MC03做主机，和作为从机的MD14demo板进行通信。
- 设置通信波特率为100K。从地址0x50，写入数据为0x19, 0x28, 0x37, 0x46, 0x55, 0x64, 0x73
- Port上选择

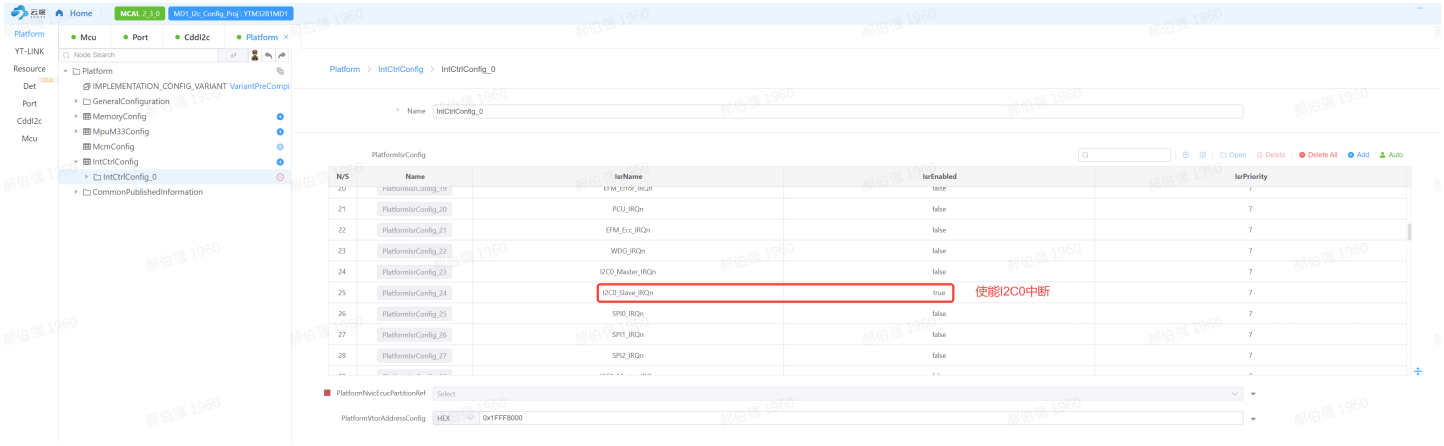
  MC03: PTD1 -- I2C1_SCL PTD0 -- I2C1_SDA

  MD14: PTA3 -- I2C0_SCL PTA2 -- I2C0_SDA
- 逻辑分析仪抓取PTA3 、PTA2 上的波形，软件验证读写buffer一致。

3.3.1 时钟配置

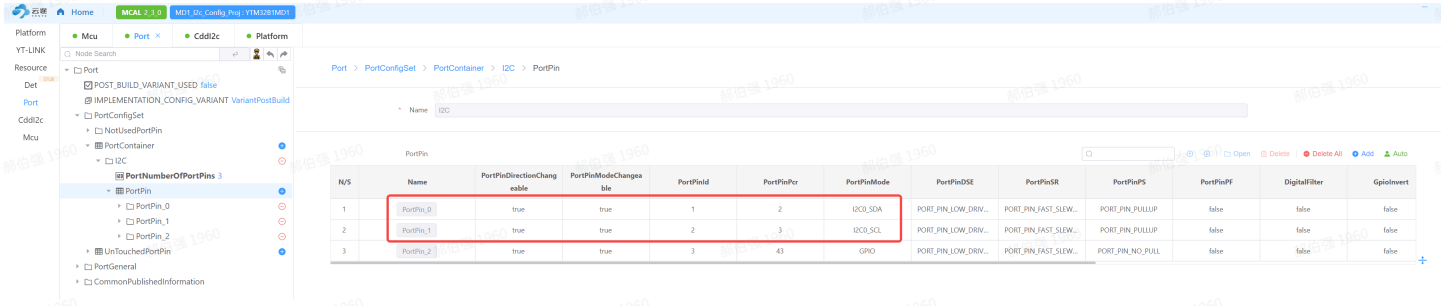


3.3.2 中断配置

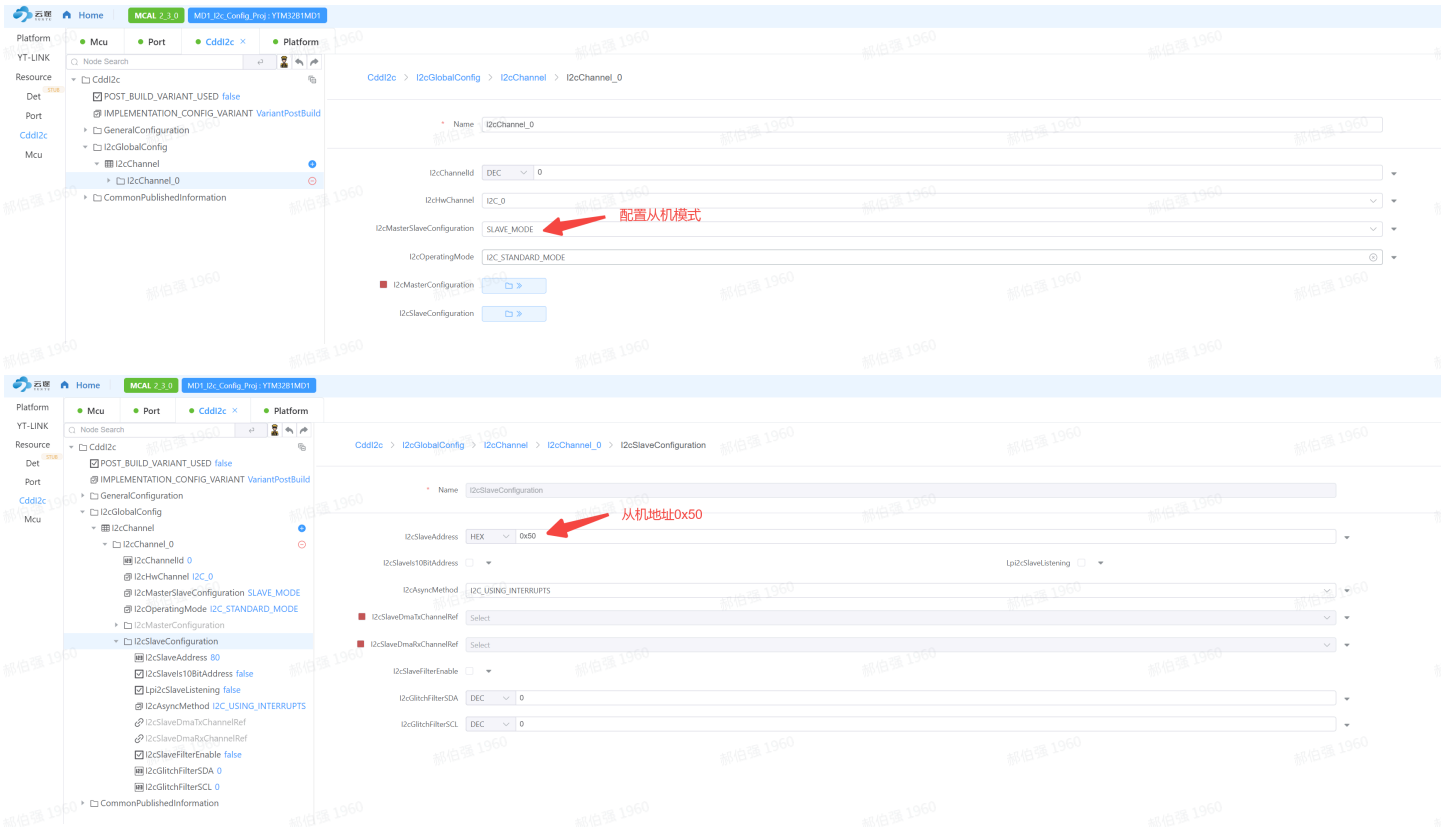


3.3.3 Port配置

这里配置PTD1、PTD0、PTE11。



3.3.4 配置I2C模块



3.3.5 API说明

这里我们用同步接口往Eeprom写数据，用异步接口读取之前写到的数据。

```
    while (1)
    {
        /*master send and slave receive data*/
        CddI2c_StartListening(CddI2cConf_I2cChannel_I2cChannel_0);
        while (CddI2c_GetTransferStatus(CddI2cConf_I2cChannel_I2cChannel_0) != I2C_CHN_COMPLETED);
        while (BufferCompare(&TestTxBuffer[1], &SlaveRxBuffer[0], MASTER_SEND_SIZE) == FALSE);
        /*master read data from slave*/
        CddI2c_StartListening(CddI2cConf_I2cChannel_I2cChannel_0);
        while (CddI2c_GetTransferStatus(CddI2cConf_I2cChannel_I2cChannel_0) != I2C_CHN_COMPLETED);
        while (BufferCompare(&MasterRxBuffer[0], &SlaveTxBuffer[0], MASTER_SEND_SIZE) == FALSE);
```

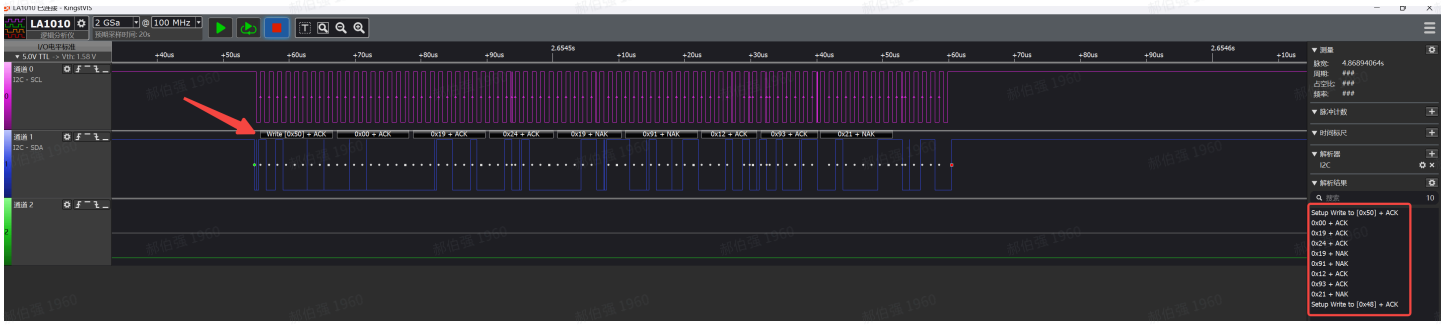
I2C从机等待接收主机数据

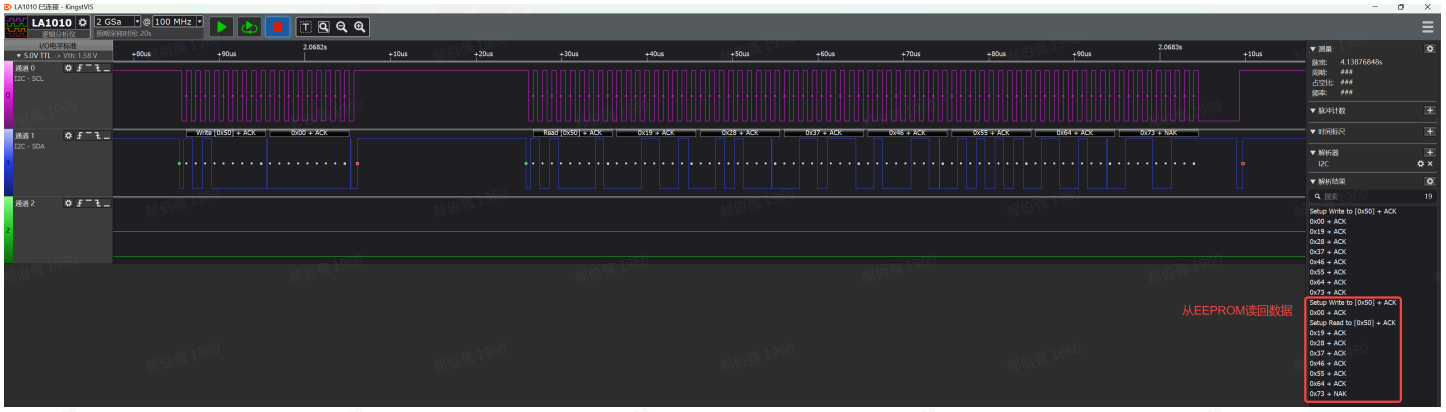
I2C从机等到主机接收命令

```
void I2c_Callback(uint8 Event, uint8 Channel)
{
    switch (Event)
    {
        case I2C_EVENT_TX_REQ_SLAVE:
            (void)CddI2c_PrepareSlaveBuffer(Channel, SlaveTxBuffer, 7, TRUE);
            break;
        case I2C_EVENT_RX_REQ_SLAVE:
            (void)CddI2c_PrepareSlaveBuffer(Channel, SlaveRxBuffer, 8, FALSE);
            break;
    }
}
```

3.3.6 现象

线上抓到的数据，与预期一致





文档历史

版本号	日期	修订记录
V1.0	2024.01.04	初始版本
V2.0	2026.04.30	1. 更新配置工具版本为2.7.8及MCAL版本为2.3.0； 2. 更新除ME05外其他型号部分介绍； 3. 新增时钟部分介绍； 4. 新增DMA模式代码示例； 5. 新增Slave模式代码示例