MCAL应用_Fls模块配置及实例

1. 版本

Config Tool Version：2.0.2

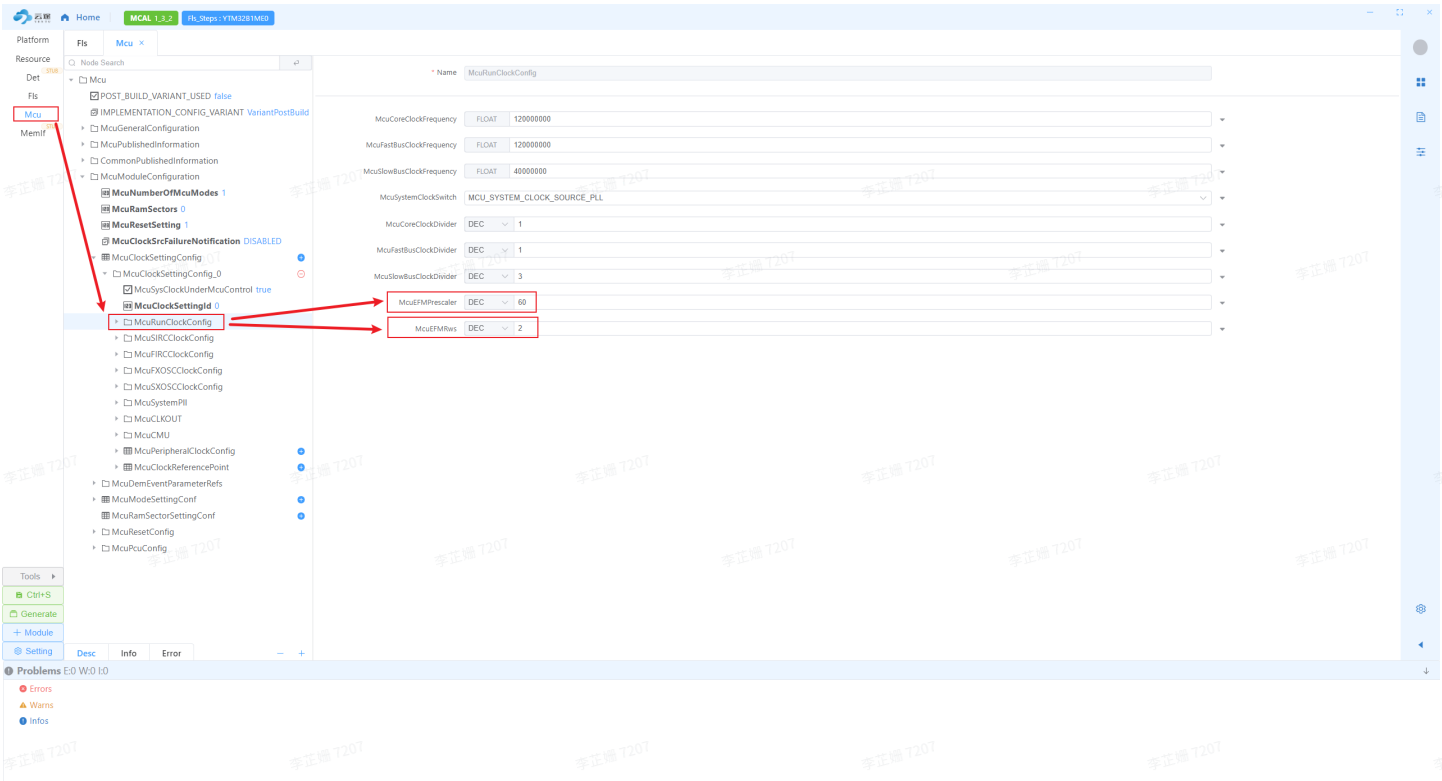
ME0 MCAL version：1.3.2

2. 前言

1. FLS一般依赖Mcu和Platform，Mcu模块里面需要配置FLS时钟，Platform里面可以配置FLS中断
2. Mcu和Platform的默认配置参考Setup篇和Mcu篇介绍

3. FLS配置介绍

3.1 时钟配置



如上图，跟FLS相关的时钟有两处需要配置：EcuEFMPrescaler、McuEFMRws。

1. EcuEFMPrescaler的配置原理

23 - 16 PRESCALE R	**Prescaler Configuration** These PRESCALER bits are used for EFM operation timing control. These bits need to be programmed to half of core frequency. Please use the following equation to calculate the PRESCALER value: PRESCALER = core frequency / 2. Please note that once the core frequency is changed, this field needs to be set to the appropriate value.

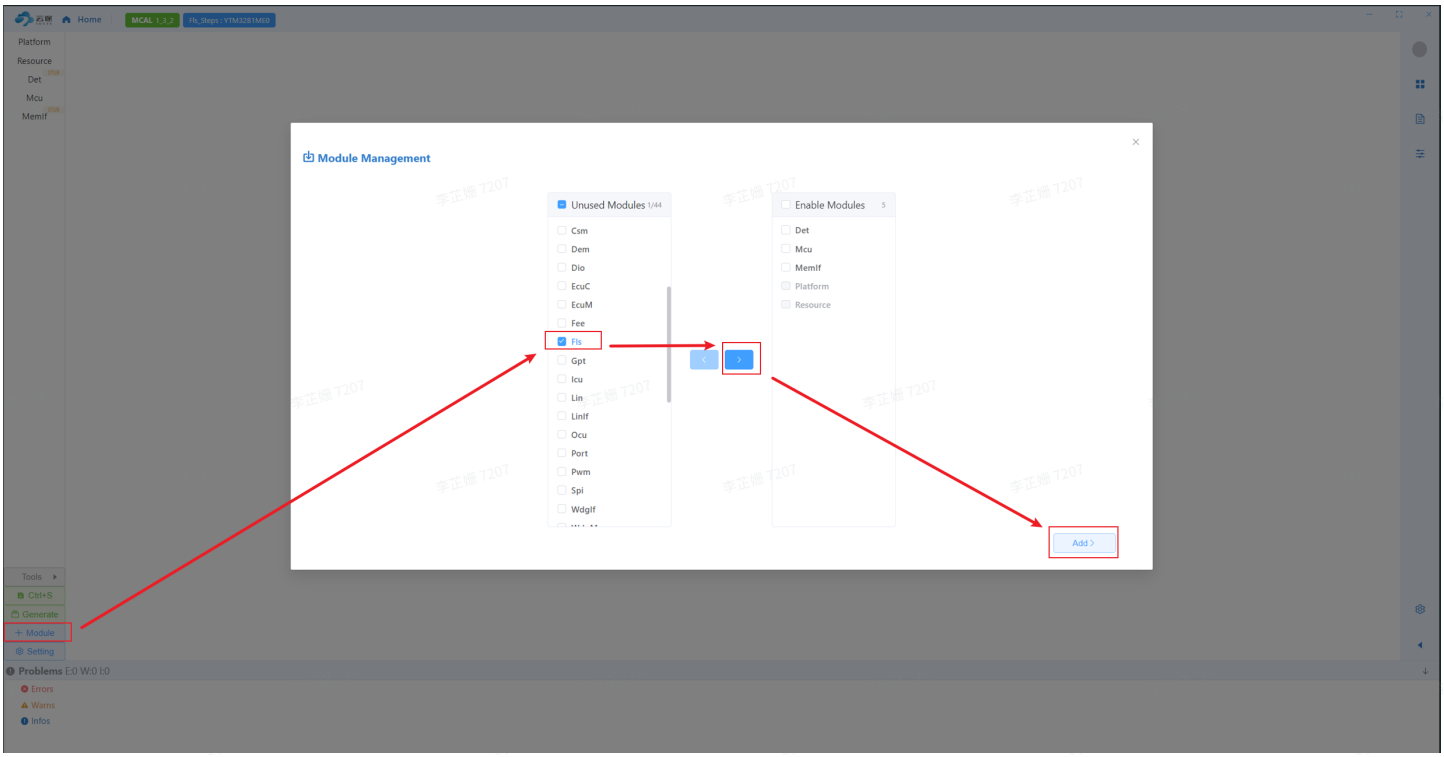
2. McuEFMRws的配置原理

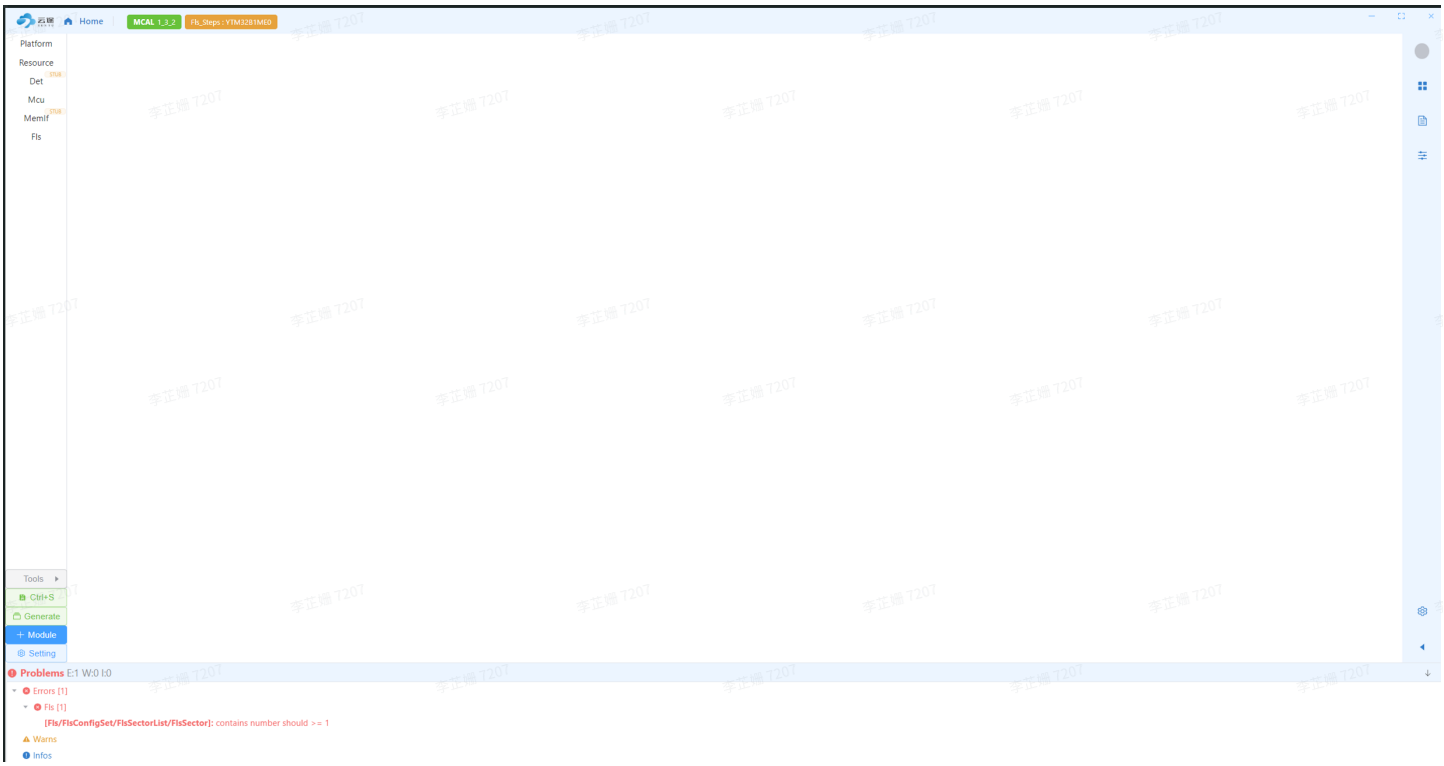
RWS	These RWS bits determine clock cycles number that needs to be waited to get the flash data out after flash received the read access signal. Please use the following equation to calculate the RWS value: (core frequency / (RWS + 1)) <= 40MHz If core frequency is less than 40MHz, RWS should be set to 0. Please note that once the core frequency is changed, this field needs to be set to the appropriate value. ***NOTE***. RWS should be reconfigured before switching to a higher core frequency.

Tips：不同的时钟源配置下的CoreClock都需要满足以上配置要求。

3.2 外设配置

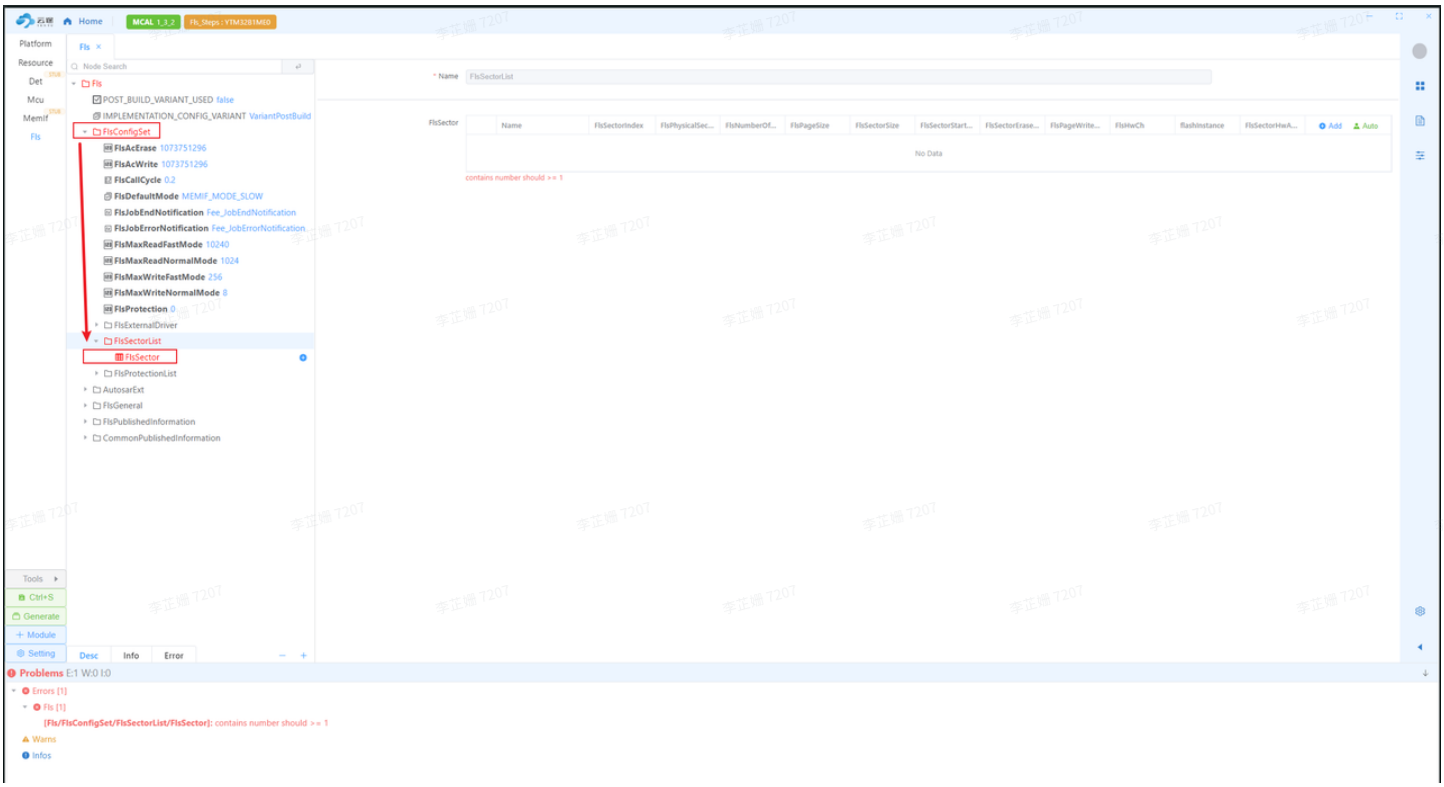
1. 添加Fls模块



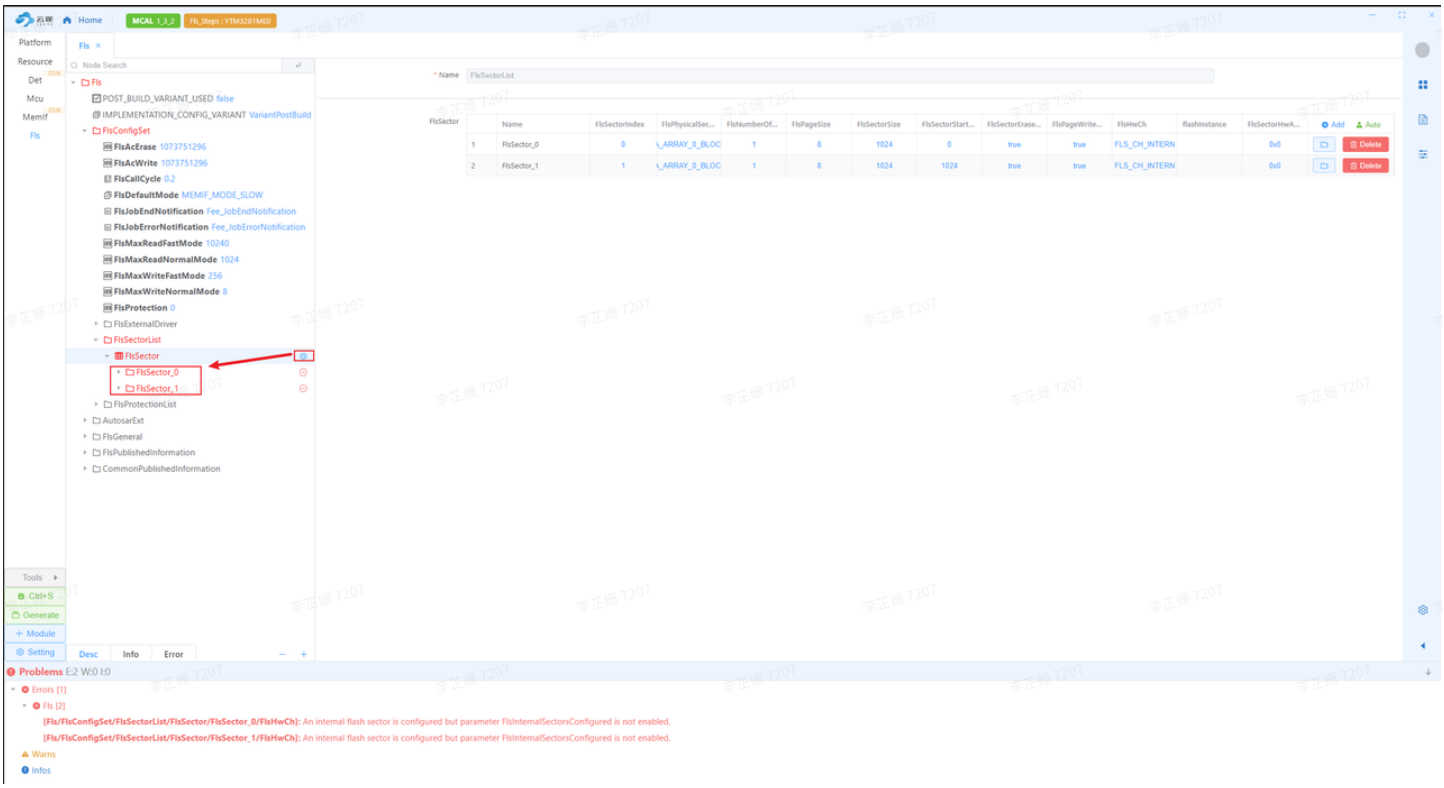


2. 找到缺失配置

根据红色错误提示，找到缺失的配置项

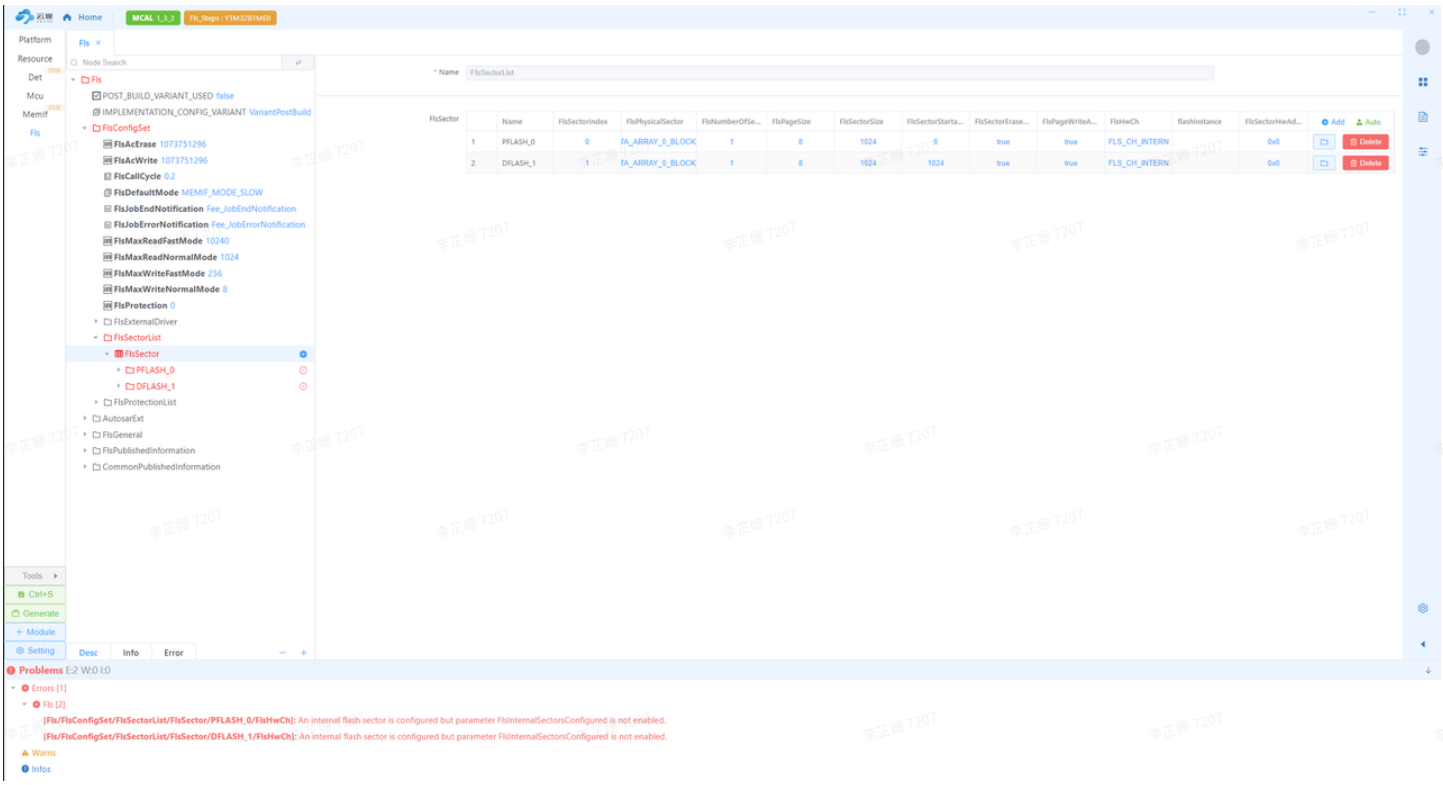


3. 新建FlsSector



4. 修改FlsSector名称

本章节以YTM32B1ME0芯片为例，新建两个SECTOR，一个选择PFLASH，一个选择DFLASH分别展示和应用说明。



5. 配置PFLASH SECTOR地址

Table 3.1: EFM Memory Map

Memory	Start Address	End Address	Size
PFlash0	0x0000_0000	0x0007_FFFF	512 KB
PFlash1	0x0008_0000	0x000F_FFFF	512 KB
DFlash	0x0010_0000	0x0013_FFFF	256 KB
HCU_NVR	0x1000_0000	0x1000_03FF	1 KB
OTP_NVR	0x1001_0000	0x1001_03FF	1 KB
CUS_NVR	0x1003_0000	0x1003_03FF	1 KB

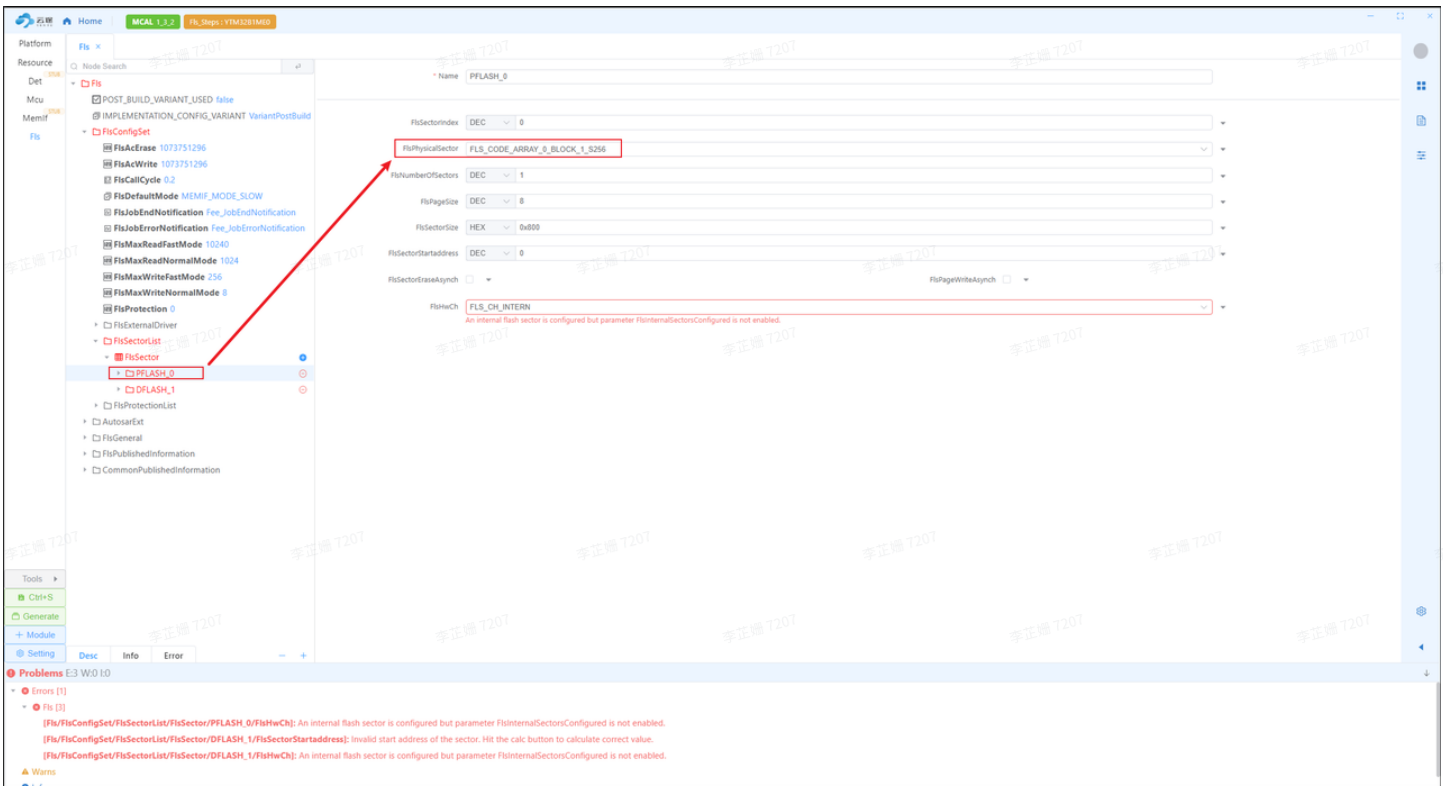
配置工具中sector名称与MCU存储器映射关系：

FLS_DATA_ARRAY_0_BLOCK_2_*：DFLASH

FLS_CODE_ARRAY_0_BLOCK_0_*：PFLASH0

FLS_CODE_ARRAY_0_BLOCK_1_*：PFLASH1

选择PFLASH1第一个SECTOR：



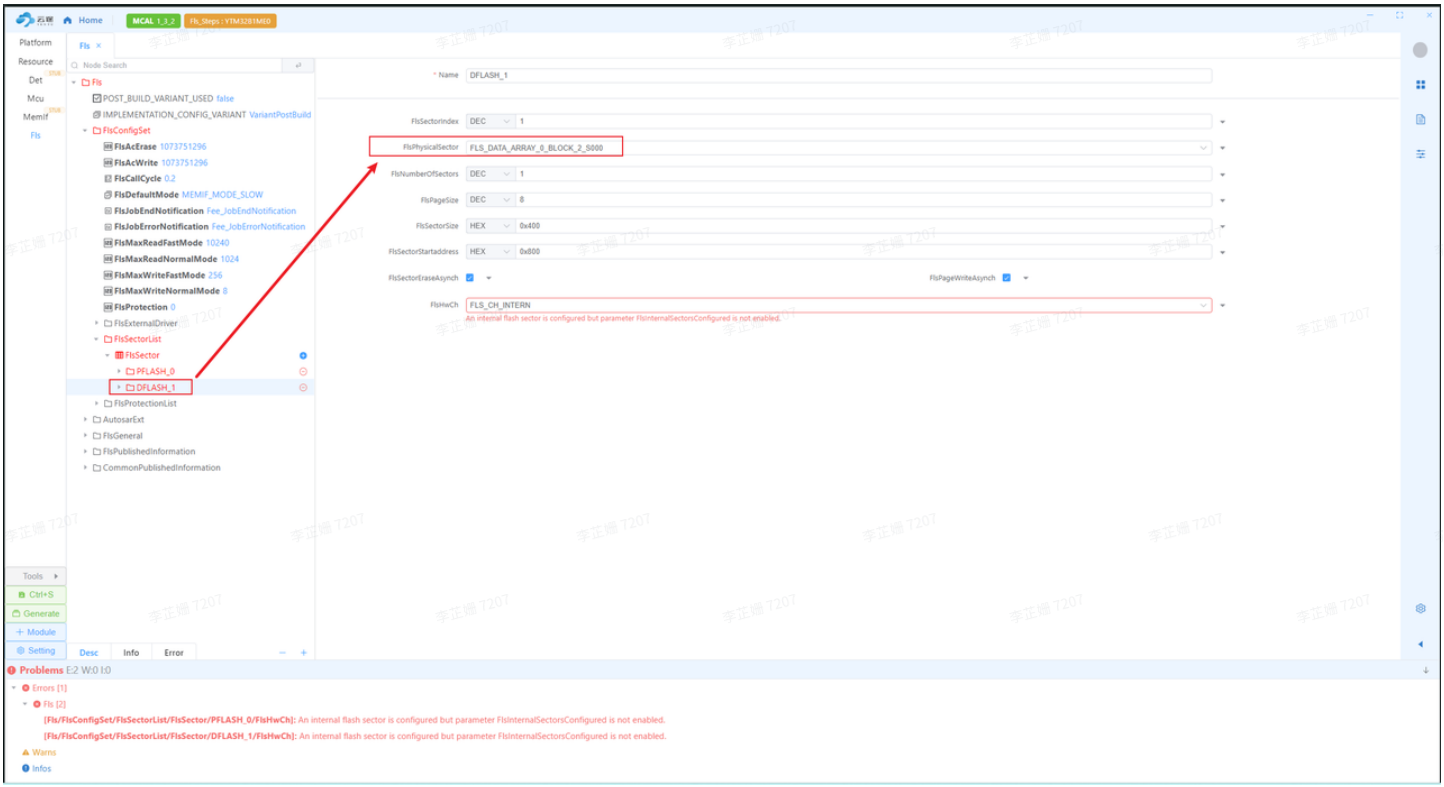
Tips：关于FlsPhysicalSector的映射关系，举例说明如下

FLS_CODE_ARRAY_0_BLOCK_1_S256，表示PFLASH1，S256表示第256个SECTOR（接连PFLASH0），起始地址256*2024=0x80000（生成的配置代码中sectorHwStartAddress地址对应），也即PFLASH1的第一个SECTOR。

Tips：FlsPageSize和FlsSectorSize需要caculate计算更新，其会根据FlsPhysicalSector范围自动计算对应

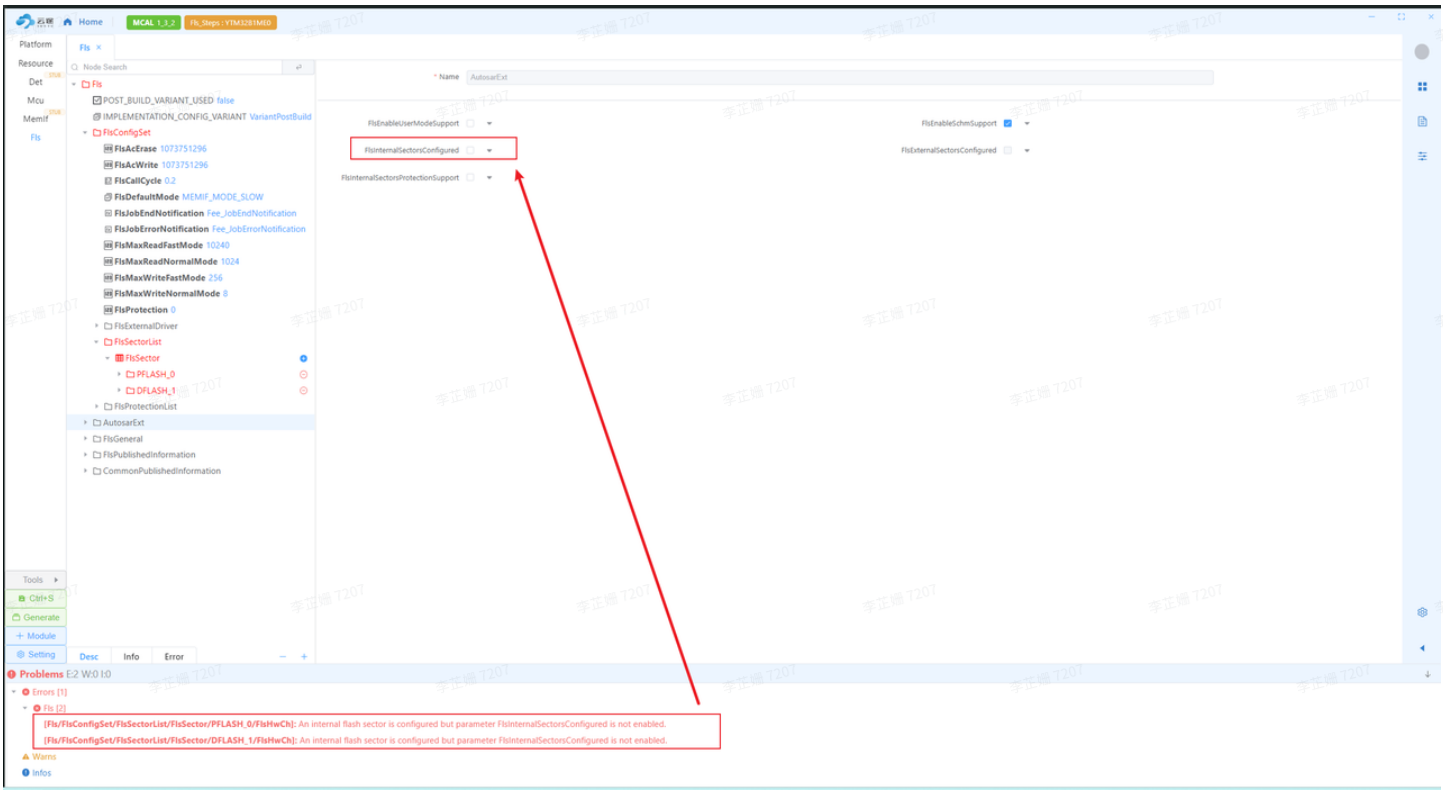
6. 配置DFLASH SECTOR地址

选择DFLASH第一个SECTOR：

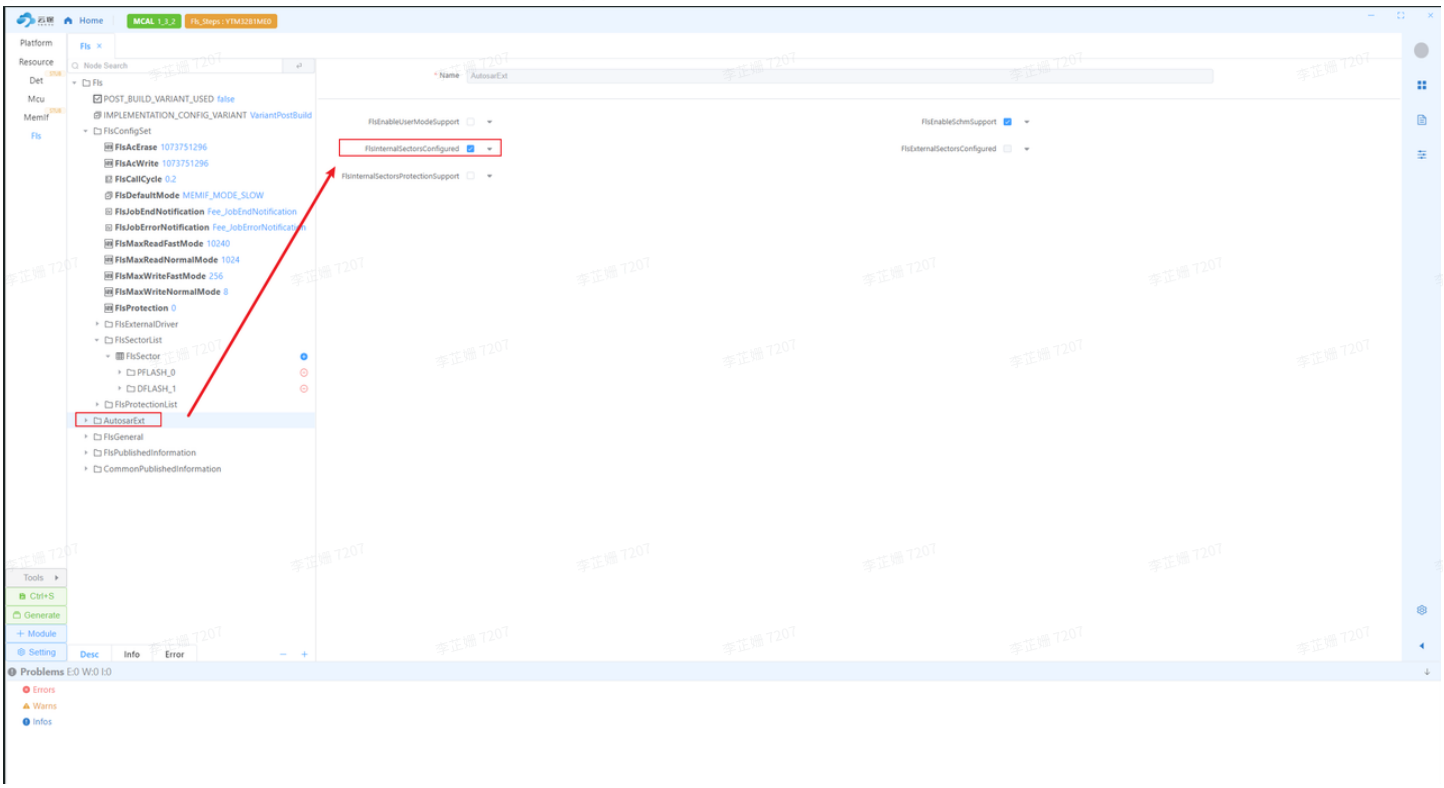


7. 错误消除

在配置完FlsSector后仍然提示错误，错误对应配置选项如下：

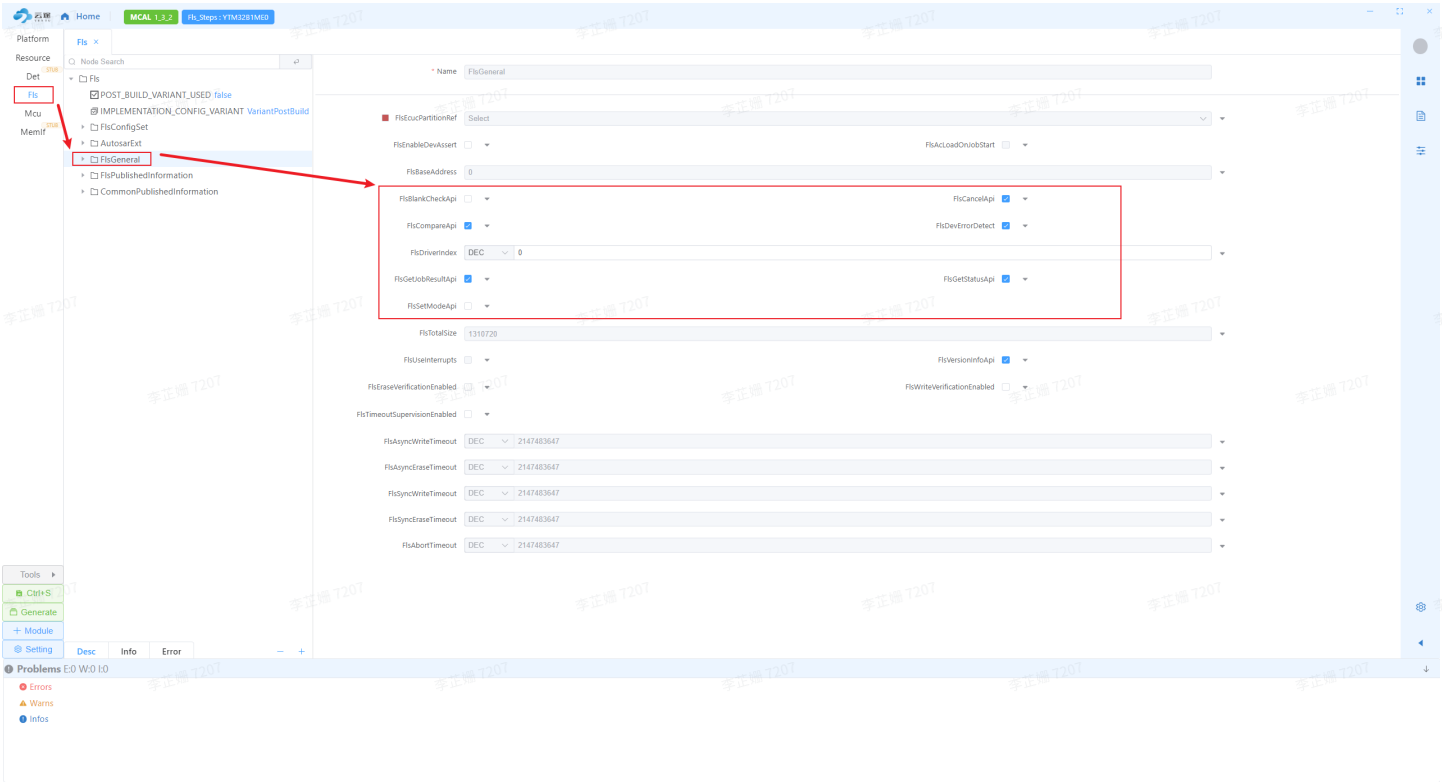


配置选项，消除错误



8. General配置

在FlsGeneral中可以配置API的生成，如下图红框中，可配置生成Fls使用的API，比如FlsCompareApi可以支持readback后数据比对校验（在下面FLS实例中展示此Api用法）。



4. FLS工程生成

配置完后参考《工程建立及通用设置》生成工程。

在board文件夹的Fls_PBconfig.c中有配置生成的软件如下：

Fls_PBcfg.c 6

Fls_Steps > board > Fls_PBcfg.c > ...

```c
/*==================================================================================================
*                                SOURCE FILE VERSION INFORMATION
==================================================================================================*/
#define FLS_VENDOR_ID_PBCFG_C                (180)
#define FLS_AR_REL_MAJOR_VER_PBCFG_C         (4)
#define FLS_AR_REL_MINOR_VER_PBCFG_C         (4)
#define FLS_AR_REL_REVISION_VER_PBCFG_C      (0)
#define FLS_SW_MAJOR_VER_PBCFG_C             (1)
#define FLS_SW_MINOR_VER_PBCFG_C             (3)
#define FLS_SW_PATCH_VER_PBCFG_C             (2)

/*==================================================================================================
*                                       Function Prototypes
==================================================================================================*/

#define FLS_START_SEC_CONFIG_DATA_UNSPECIFIED
#include "Fls_MemMap.h"
/================================================================================================== */
FLS_CONST const Fls_SectorType Fls_SectorConfig[2] ={
    {

        .sectorId = FlsConf_FlsConfigSet_PFLASH_0,
        .sectorStartAddress = 0x0U,
        .sectorSize = 0x800U,
        .pageSize = 0x8U,
        .sectorHwStartAddress = 0x80000U,
        .asyncAccess = FALSE,
    },
    {

        .sectorId = FlsConf_FlsConfigSet_DFLASH_1,
        .sectorStartAddress = 0x800U,
        .sectorSize = 0x400U,
        .pageSize = 0x8U,
        .sectorHwStartAddress = 0x100000U,
        .asyncAccess = TRUE,
    },
};


FLS_CONST const Fls_ConfigType Fls_Config = {
    .acEraseFunPtr = NULL_PTR,
    .acWriteFunPtr = NULL_PTR,
    .jobEndNotificationFunPtr = NULL_PTR,
    .jobErrorNotificationFunPtr = NULL_PTR,
    .eDefaultMode = MEMIF_MODE_SLOW,
    .maxReadFastMode = 10240U,
    .maxReadNormalMode = 1024U,
    .maxWriteFastMode = 256U,
    .maxWriteNormalMode = 8U,
    .configuredSectorNumber = 2U,
    .sectorList = Fls_SectorConfig,
};

#define FLS_STOP_SEC_CONFIG_DATA_UNSPECIFIED
#include "Fls_MemMap.h"

```

其中：

Line41~Line49，是界面配置的PFLASH_SECTOR的配置，起始地址是0x00080000，SECTOR SIZE 0x800；

Line50~Line58，是界面配置的DFLASH_SECTOR的配置，起始地址是0x00100000，SECTOR SIZE 0x400；

Line48：配置PFLASH_SECTOR区间执行Fls同步操作；

Line57：配置DFLASH_SECTOR区间执行Fls异步操作；

> 💡 Tips：关于Fls同步操作和异步操作的原理
>
> 在不同的物理BLOCK上的FLASH程序和数据可支持异步操作不产生阻塞；
>
> 在相同的物理BLOCK上的FLASH程序和数据只支持同步操作会产生阻塞；

在本次配置案例中，FLASH软件代码量不大，CODE SIZE小于256K，位于PFLASH0区间；

配置的PFLASH_SECTOR位于PFLASH1区间；

配置的DFLASH_SECTOR位于DFLASH区间；

因此本次配置案例中的PFLASH_SECTOR和DFLASH_SECTOR均可支持异步操作，然后人为配置一个同步一个异步。

5. FLS实例

1. FLASH操作的主动驱动为读、写、擦。
2. YTM32B1ME0芯片的PFLASH SECTOR SIZE是0x800，DFLASH SECTOR SIZE是0x400，定义如下的全局数组：

```
42  /* Private variables ---------------------------------------------------------*/
43  /* USER CODE BEGIN PV */
44
45  uint8 PFLS_WriteData[0x800] = {0x00};
46  uint8 PFLS_ReadData[0x800] = {0x00};
47
48  uint8 DFLS_WriteData[0x400] = {0x00};
49  uint8 DFLS_ReadData[0x400] = {0x00};
50
51  /* USER CODE END PV */
```

3. FLS初始化

```
Fls_Init(ConfigPtr: &Fls_Config);
```

5.1 PFLASH操作

1. FLS擦数据
2. FLS写数据

3. FLS读数据
4. FLS校验数据

```
/*********************************************************************************************************************************************************************/
  Fls_Erase(TargetAddress: Fls_Config.sectorList[FlsConf_FlsConfigSet_PFLASH_0].sectorStartAddress, Length: Fls_Config.sectorList[FlsConf_FlsConfigSet_PFLASH_0].sectorSize);
  do
  {
    Fls_MainFunction();
  } while (Fls_GetStatus() != MEMIF_IDLE);

  for(int i = 0; i < 0x800; i++)
  {
    PFLS_ReadData[i] = 0;
  }
  Fls_Read(SourceAddress: Fls_Config.sectorList[FlsConf_FlsConfigSet_PFLASH_0].sectorStartAddress, TargetAddressPtr: PFLS_ReadData, Length: 0x800);
  do
  {
    Fls_MainFunction();
  } while (Fls_GetStatus() != MEMIF_IDLE);

  for(int i = 0; i < 0x800; i++)
  {
    if(PFLS_ReadData[i] != 0xFF)
    {
      while(1);
    }
  }

  for(int i= 0; i< 0x800; i++)
  {
    PFLS_WriteData[i] = i;
  }
  Fls_Write(TargetAddress: Fls_Config.sectorList[FlsConf_FlsConfigSet_PFLASH_0].sectorStartAddress, SourceAddressPtr: PFLS_WriteData, Length: 0x800U);
  do
  {
    Fls_MainFunction();
  } while (Fls_GetStatus() != MEMIF_IDLE);

  Fls_Read(SourceAddress: Fls_Config.sectorList[FlsConf_FlsConfigSet_PFLASH_0].sectorStartAddress, TargetAddressPtr: PFLS_ReadData, Length: 0x800);
  do
  {
    Fls_MainFunction();
  } while (Fls_GetStatus() != MEMIF_IDLE);

  for(int i = 0; i < 0x800; i++)
  {
    if(PFLS_ReadData[i] != PFLS_WriteData[i])
    {
      while(1);
    }
  }

  Fls_Compare(SourceAddress: Fls_Config.sectorList[FlsConf_FlsConfigSet_PFLASH_0].sectorStartAddress, TargetAddressPtr: PFLS_WriteData, Length: 0x800);
  do
  {
    Fls_MainFunction();
  } while (Fls_GetStatus() != MEMIF_IDLE);

  while((Fls_GetJobResult() != MEMIF_JOB_OK))
  {
  }
/*********************************************************************************************************************************************************************/
```

5.2 DFLASH操作

1. FLS擦数据
2. FLS写数据
3. FLS读数据
4. FLS校验数据

```
/*****************************************************************************************************************************/
  Fls_Erase(TargetAddress: Fls_Config.sectorList[FlsConf_FlsConfigSet_DFLASH_1].sectorStartAddress, Length: Fls_Config.sectorList[FlsConf_FlsConfigSet_DFLASH_1].sectorSize);
  do
  {
    Fls_MainFunction();
  } while (Fls_GetStatus() != MEMIF_IDLE);

  for(int i = 0; i < 0x400; i++)
  {
    DFLS_ReadData[i] = 0;
  }
  Fls_Read(SourceAddress: Fls_Config.sectorList[FlsConf_FlsConfigSet_DFLASH_1].sectorStartAddress, TargetAddressPtr: DFLS_ReadData, Length: 0x400);
  do
  {
    Fls_MainFunction();
  } while (Fls_GetStatus() != MEMIF_IDLE);

  for(int i = 0; i < 0x400; i++)
  {
    if(DFLS_ReadData[i] != 0xFF)
    {
      while(1);
    }
  }

  for(int i= 0; i< 0x400; i++)
  {
    DFLS_WriteData[i] = i;
  }
  Fls_Write(TargetAddress: Fls_Config.sectorList[FlsConf_FlsConfigSet_DFLASH_1].sectorStartAddress, SourceAddressPtr: DFLS_WriteData, Length: 0x400);
  do
  {
    Fls_MainFunction();
  } while (Fls_GetStatus() != MEMIF_IDLE);

  Fls_Read(SourceAddress: Fls_Config.sectorList[FlsConf_FlsConfigSet_DFLASH_1].sectorStartAddress, TargetAddressPtr: DFLS_ReadData, Length: 0x400);
  do
  {
    Fls_MainFunction();
  } while (Fls_GetStatus() != MEMIF_IDLE);

  for(int i = 0; i < 0x400; i++)
  {
    if(DFLS_ReadData[i] != PFLS_WriteData[i])
    {
      while(1);
    }
  }

  Fls_Compare(SourceAddress: Fls_Config.sectorList[FlsConf_FlsConfigSet_DFLASH_1].sectorStartAddress, TargetAddressPtr: DFLS_WriteData, Length: 0x400);
  do
  {
    Fls_MainFunction();
  } while (Fls_GetStatus() != MEMIF_IDLE);

  while((Fls_GetJobResult() != MEMIF_JOB_OK))
  {
  }
/*****************************************************************************************************************************/
```

6. 文档历史

版本号	日期	修订记录
V1.0	2023.11.20	初始版本