YTMicro MCAL应用 - Lin模块配置及实例

版本

Config Tool Version：2.7.8

ME05 MCAL version：2.3.0

MC03 MCAL version：2.3.0

前言

1. 本文介绍 LIN 模块原理、驱动配置与 API，以及 ME05 LINFlexD 主从通信、MC03 UART LIN 主机和从机三个配置实例。
2. 时钟、引脚、中断及 GPT 的基础配置分别参考 Mcu、Port、工程建立及 Gpt 模块文档。本文重点说明这些资源与 LIN 通信的配置关系。
3. 配置项包含 AUTOSAR 标准选项和云途扩展选项。工具树中粗体表示标准选项，细体表示扩展选项；标准选项可查阅项目所用版本的 AUTOSAR_CP_SWS_LinDriver.pdf。

1. LIN 模块概述

1.1 AUTOSAR 分层与驱动职责

下图为普通信号的典型通信路径。Lin 驱动位于 MCAL 层，向上提供通道控制和帧收发接口，向下操作 LINFlexD 或 UART 外设。



图 1-1 AUTOSAR LIN 通信分层

Com 负责信号打包与解析，PduR 负责 PDU 路由，LinIf 组织 LIN 通信并执行主节点调度，MCAL Lin 驱动负责底层帧收发。LinSM 协调网络通信状态，不属于逐帧数据传输路径。LIN 控制器是MCU LIN控制单元。LIN 收发器负责逻辑信号与总线电平转换。收发器的使能/休眠引脚仍需按板级电路配置。

1.2 LIN 网络和帧结构

LIN 网络：由一个主节点和一个或多个从节点组成。主节点按调度发送帧头，该帧的发布者发送响应；发布者既可以是主节点，也可以是从节点。从节点有数据待发送时，仍需等待主节点发送对应帧头。

LIN帧结构：由帧头和响应两部分组成。

帧头包括同步间隔段、同步段以及PID(Protected Identifier，受保护ID)段，应答包括数据段和校验和段，如图 3.2 所示，其中值“0”为显性电平(Dominant)，值“1”为隐性电平(Recessive)，总线上实行“线-与”：当总线上有大于等于一个节点发送显性电平时，总线呈显性电平；所有的节点都发送隐性电平或不发送信息(不发送任何信息时总线默认呈隐性电平)时，总线才呈现隐性电平，即显性电平起主导作用。图中帧间隔为帧之间的间隔；应答间隔为帧头和应答之间的间隔；字节间间隔包括同步段和受保护ID段之间的间隔、数据段各字节间之间的间隔以及数据段最后一个字节和校验和段之间的间隔。下面对帧头和应答的各部分进行详细说明。

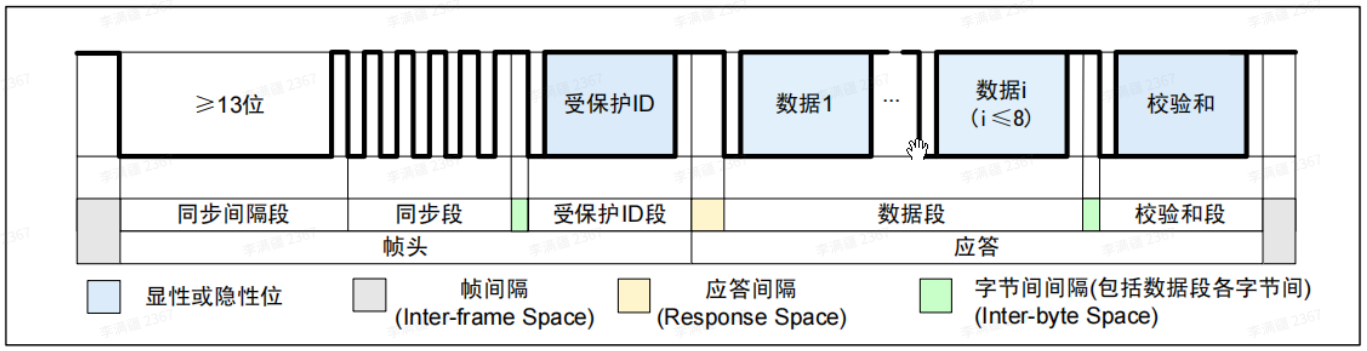


图 3.2 帧的结构

Break同步间隔段：帧起始标志，主节点发送至少 13 bit 显性电平和至少 1 bit 隐性间隔符。配置主节点发送长度时，应与从节点的 Break 检测门限区分。

Sync同步段：固定为 0x55，用于从节点测量位时间。是否启用自动同步以及标称波特率的配置方式，取决于外设和驱动实现。

PID受保护ID段：受保护标识符，由 6 bit 帧 ID 和 2 bit 奇偶校验位组成。

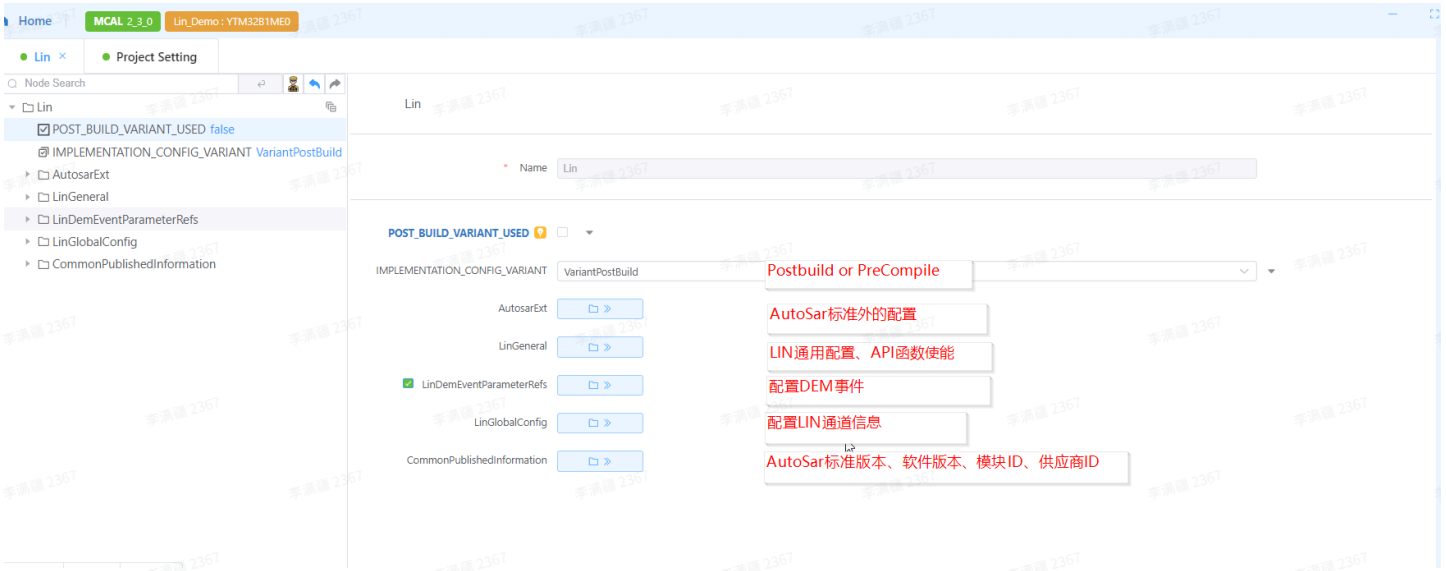
Data数据段、Checksum校验和段：响应包含 1～8 字节数据和 1 字节校验和。

2. Lin配置介绍

本章介绍如何配置通道和硬件资源，以及如何使用 MCAL LIN API。配置前应确定节点角色、网络速率及帧定义。LDF（LIN Description File）可用于描述节点、信号、帧和调度关系，但 Lin 驱动不直接读取 LDF。完整 AUTOSAR 系统将这些定义落实到 Com、LinIf 等模块；本文实例将帧定义落实到 PDU 数组、应用数据及回调函数。

2.1 通道与硬件配置

2.1.1 框架说明

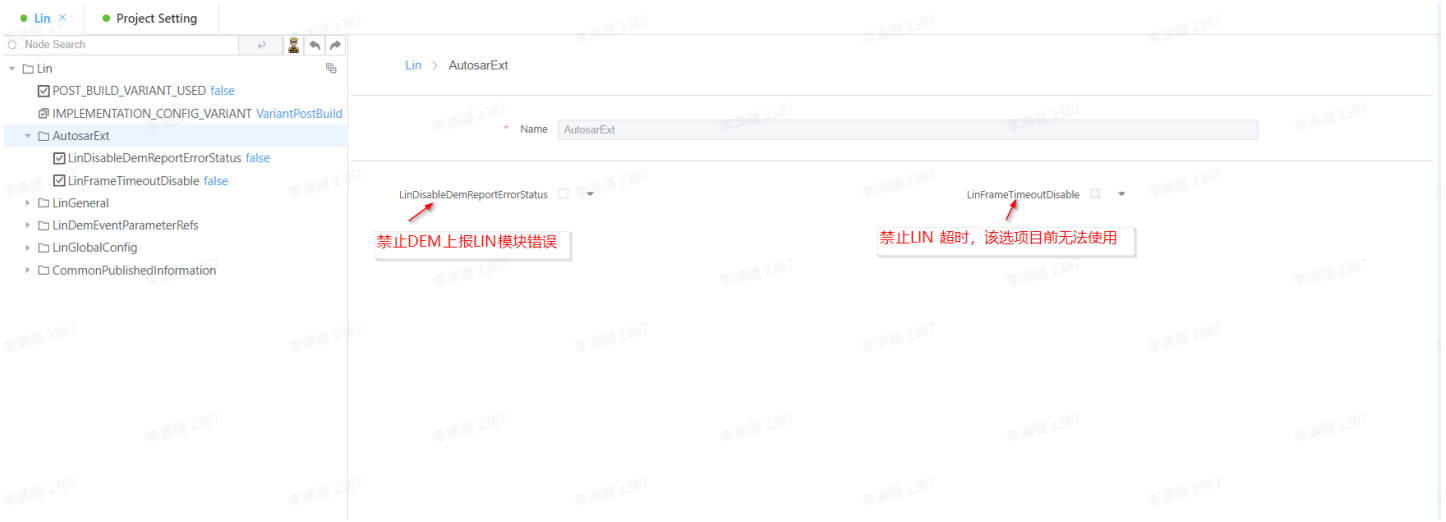


选项	含义
POST_BUILD_VARIANT_USED	声明是否使用构建后配置的“变体”机制，例如为不同产品提供不同配置变体
IMPLEMENTATION_CONFIG_VARIANT	选择模块采用哪种配置实现方式，如编译前配置、构建后配置

Tips:

- **Precompile：决定程序里“有没有这项功能”。**
  例如当前版本查询接口关闭。如果要启用，需要修改配置并重新编译相关代码，因为原程序可能没有编入该接口。
- **Post-build：为已经具备的功能提供“使用什么参数”。**
  当前工程将通道波特率等参数保存在 `Lin_Config` 配置集中，初始化时传给驱动。驱动根据配置数据初始化通道。

2.1.2 AutoSarExt



2.1.3 LinGeneral



- LinSchMEnable：是否启用驱动临界区保护。是否可以关闭，应依据中断、任务及 API 并发访问情况评估。

提示：

使用 OS 或存在任务、中断并发访问驱动资源时，应配置适当的临界区保护。

仅在确认相关驱动资源不存在并发访问，并完成任务、中断和 API 调用关系评估后，才可考虑关闭临界区保护。

- LinEnableUserModeSupport：是否启用用户模式访问支持。

提示：

- **用户模式和特权模式有什么区别?**

CPU 执行模式	与 LIN 驱动相关的区别
特权模式	在系统保护配置允许的范围内，可以访问外设寄存器、执行特权操作
用户模式	权限受限；直接访问受保护的寄存器或执行特权操作，可能触发保护异常

LIN 驱动需要操作时钟、控制、收发状态等硬件寄存器。如果调用路径处于用户模式，就需要通过系统提供的受控入口完成这些操作。

- **勾选这个选项后，驱动做了什么?**

配套驱动中确实存在两种调用路径。例如初始化通道时：

开启 `LIN_ENABLE_USER_MODE_SUPPORT` 后，会通过 `OsIf_Trusted_Call...` 包装底层操作；关闭时则直接调用。

- LinTimeoutDuration：驱动内部等待硬件状态变化的最大轮询次数，生成宏 LIN_TIMEOUT_DURATION。配置 1000 表示最多执行 1000 次轮询，例如等待发送中止完成。

2.1.4 LinDemEventParameterRefs



2.1.5 LinGlobalConfig



- LinChannelId： API 使用的逻辑通道号
- LinHardwareChannel： 指定实际外设
- LinNodeType： 指定主从角色。
- LinChannelBaudrate： 外设时钟配置波特率。
- LinResponseTimeout：响应超时配置（LINFlexD RTO）
- LinHeaderTimeout：帧头超时配置（LINFlexD HTO）

Tips:

- LINFlexD 超时参数对照（单位：bit time，1 bit time = Tbit）：

配置项 / 字段	计时含义	换算公式
LinResponseTimeout RTO[11:8]，0～15	每个响应字节的时间预算 包含数据字节及校验和	响应总预算 T响应 = RTO × (N + 1) × Tbit
LinHeaderTimeout HTO[6:0]，0～127	帧头计时预算 仅从节点模式可写	硬件计时区间 T帧头计时 = HTO × Tbit

Table 19.12: LINFlexD LINTOCR Register Description

Field	Function
11 - 8 RTO	**Response Timeout Value** The response timeout duration (in bit time) for 1 byte. The reset value is 0Eh = 14, corresponding to $T_{Response_Maximum} = 1.4 \times T_{Response_Nominal}$.
6 - 0 HTO	**Header Timeout Value** HTO specifies the header timeout duration (in bit time). HTO can be written only for slave mode. The header timeout should be programmed without considering 11 bits of break field.

计算条件：N 为数据字节数，N + 1 包含校验和；Tbit = 1 / 波特率。19200 bit/s 时，Tbit ≈ 52.083 µs。

每字节标称长度为 10 bit（起始位 1 + 数据位 8 + 停止位 1）。RTO=14 时，每字节预算为标称值的 1.4 倍。

响应算例：（19200 bit/s，RTO=14）：

数据长度 N	标称响应时间 10 × (N + 1) × Tbit	响应超时预算 14 × (N + 1) × Tbit
1 字节	1.042 ms	1.458 ms
2 字节	1.563 ms	2.188 ms
8 字节	4.688 ms	6.563 ms

帧头算例：HTO=44 → 计时区间约 2.292 ms。硬件不计 Break 前 11 bit；从 Break 起点折算为 (44 + 11) × Tbit ≈ 2.865 ms。HTO=44 是本例取值，配置时须按实际帧头时序核对。

- UART + GPT 分段计时：帧头阶段与响应阶段分别监控，不使用 LINFlexD 的 RTO/HTO 字段。下表对应当前 UART 驱动的三类计时路径。

计时阶段	启动与停止时机	预算系数 C
帧头：主节点 Break LIN_DRV_TO_BREAK	发送 Break 时启动；检测到本地 Break 后停止。随后发送 Sync、PID，该主节点路径未再启动 HEADER 计时。	48
帧头：从节点接收 LIN_DRV_TO_HEADER	检测到 Break 后启动，监控后续 Sync、PID；进入 PID 处理时停止。并非从 Break 起点计完整帧头。	34
响应：主／从节点接收 LIN_DRV_TO_M_RECEIVE LIN_DRV_TO_S_RECEIVE	帧头处理完成、确定本地接收响应后启动；接收完成后停止，超时则按对应错误路径处理。	14 × (N + 1)

统一公式：tbit_us = floor(1000000 / 波特率) + 1；K = floor(C × tbit_us / 500) + 2。K 为超时回调计数阈值，N 为响应数据字节数（不含校验和）。

算例（19200 bit/s → tbit_us=53 µs，GPT 回调周期 500 µs）：

阶段 / 条件	计数阈值 K	K × 500 µs
主节点 Break，C=48	floor(48 × 53 / 500) + 2 = 7	3.5 ms
从节点帧头，C=34	floor(34 × 53 / 500) + 2 = 5	2.5 ms
响应接收，N=8，C=126	floor(126 × 53 / 500) + 2 = 15	7.5 ms

说明：表中时间为计数阈值对应的节拍预算；实际触发时刻还受启动相位和中断延迟影响。三段按状态切换启停，不能直接相加作为统一整帧超时。以上软件超时路径以 LIN_FRAME_TIMEOUT_DISABLE=STD_OFF 为前提。

- LinChannelWakeupSupport：是否支持LIN总线外部唤醒。

LinChannelWakeupSupport	支持的唤醒类型
TRUE	**LIN 总线唤醒（外部）和内部本地唤醒**
flase	**不启用该通道的外部总线唤醒支持；**

Tips:

Lin唤醒的两种方式：

- 外部唤醒：其他节点在总线上发出显性脉冲，本节点检测并由上层完成唤醒处理。

脉冲信号时间参数：

发送方：250μs ~ 5ms 显性

接收方判定：>150μs ，接收检测条件按外设手册执行；有效唤醒的确认和上报按驱动及系统流程处理。

- 内部唤醒：内部应用请求唤醒，不发总线唤醒脉冲；

操作场景	API／处理入口	主节点行为	从节点行为
主动请求网络唤醒	Lin_Wakeup()	发送唤醒脉冲，恢复本地通道；由上层等待节点就绪后启动调度	发送唤醒脉冲，恢复本地通道；等待主节点帧头
检测外部总线唤醒	唤醒中断及上层处理；按驱动流程调用 Lin_CheckWakeup()	检查唤醒事件；通道恢复及调度由相应流程处理	检查唤醒事件；恢复后等待主节点帧头
仅恢复本地通道	Lin_WakeupInternal()	不发送唤醒脉冲；是否恢复调度由上层决定	不发送唤醒脉冲；等待主节点帧头

Tips:

- ID 过滤的平台差异：

LINFlexD 支持硬件 ID 过滤；MD24 当前 UART LIN 驱动未使用生成配置中的禁止 ID 列表字段。UART 从节点是否处理某帧，由 `LinIf_HeaderIndication()` 等上层逻辑决定。配置工具显示同名参数，不代表不同外设具有相同实现，迁移时应核对驱动路径

2.1.6 CommonPublishedInformation



2.2 帧定义与收发参数

在 MCAL 中，Lin_PduType 用于描述对应的 PID、响应方向、数据长度、校验方式及数据指针。以下按帧字段说明这些参数。

2.2.1 帧 ID 与 PID

Lin_PduType.Pid 使用 8 bit PID。ID 是 6 bit 帧标识符，取值 0x00～0x3F。普通信号帧使用 0x00～0x3B；0x3C/0x3D 为诊断帧 ID，0x3E/0x3F 保留。总线上发送的是加入奇偶校验后的 8 bit PID。

设 ID 的各位为 ID0～ID5，⊕ 表示异或，¬ 表示对单个校验位取反。

P0 = ID0 ⊕ ID1 ⊕ ID2 ⊕ ID4

P1 = ¬(ID1 ⊕ ID3 ⊕ ID4 ⊕ ID5)

PID = ID | (P0 << 6) | (P1 << 7)

ID = Pid & 0x3F 仅提取低 6 bit 帧 ID，并不验证 PID 的奇偶校验是否正确。

帧 ID	校验位 P1 / P0	总线 PID	对应说明
0x29	1 / 1	0xE9	代码 Pid=0xE9，与上位机 ID=0x29 是同一帧。
0x2E	0 / 0	0x2E	PID 恰好与 ID 数值相同，不能据此省略校验位检查。

2.2.2 响应方向

本次响应由谁发送	主机 PDU 的 Drc	从机的处理
主机发布响应	LIN_FRAMERESPONSE_TX	需要该帧的从机接收响应。
从机发布响应	LIN_FRAMERESPONSE_RX	指定从机发送响应，其他订阅者可接收。
主机只发帧头，不处理响应	LIN_FRAMERESPONSE_IGNORE	由指定从机发布，订阅者接收。

TX、RX、IGNORE 指本地通道对响应的处理，不改变由主节点发送帧头的规则。从节点回调中的 Drc=TX 表示本地发送响应，Drc=RX 表示本地接收响应。

2.2.3 数据长度与校验方式

Lin_PduType.Dl 指定响应数据长度，应与网络定义及通信对端一致。

SduPtr 指向数据缓冲区；信号的位布局、字节序、缩放和偏移由上层 Com 或应用处理。

经典校验和仅覆盖数据，增强校验和还包含 PID，均采用带回卷进位的字节求和并取反。诊断帧 ID=0x3C/0x3D 使用经典校验和；其他帧按网络约定选择，收发双方须一致。

2.3 MCAL LIN API

2.3.1 初始化与版本查询

API	介绍说明
Lin_Init	void Lin_Init(const Lin_ConfigType *Config); 初始化驱动和通道资源，无返回值。LIN_FIXED_PB_CONFIG 关闭时，Config 必须指向有效配置集；开启时使用固定配置。外设时钟及所需 GPT 应先就绪（UART 外设做LIN时有依赖），引脚和中断应在开始通信前配置完成。当前实现初始化后通道处于休眠状态，需按通信流程恢复通道。
Lin_DeInit	void Lin_DeInit(void); 反初始化驱动，无参数、无返回值。当前实现要求驱动已初始化，且所有通道均处于休眠状态；否则不执行反初始化。接口由 LIN_DEINIT_API_SUPPORT 控制，成功反初始化后再次通信需重新初始化。
Lin_GetVersionInfo	Lin_GetVersionInfo(Std_VersionInfoType *versioninfo); 将厂商 ID、模块 ID 和驱动软件版本写入 versioninfo 指向的结构体，无返回值。该接口由 LIN_VERSION_INFO_API 控制，在当前驱动中以宏实现。

2.3.2 主节点收发与状态查询

API	介绍说明
Lin_SendFrame	Std_ReturnType Lin_SendFrame(uint8 Channel, const Lin_PduType *PduInfoPtr); 由主节点发起一次帧传输，PduInfoPtr 指定 PID、响应方向、长度、校验方式及数据指针。TX 表示主节点发送响应，RX 表示接收响应，IGNORE 表示不处理响应。E_OK 不表示整帧已完成，完成结果需查询状态；
Lin_GetStatus	Lin_StatusType Lin_GetStatus(uint8 Channel, uint8 **Lin_SduPtr); 查询通道收发状态；接收成功时通过 Lin_SduPtr 返回驱动接收缓冲区指针。LIN_TX_OK / LIN_RX_OK 表示完成，LIN_TX_BUSY / LIN_RX_BUSY 表示进行中；LIN_RX_NO_RESPONSE、LIN_RX_ERROR 等表示无响应或错误。应在 LIN_RX_OK 时读取并及时复制接收数据。

Lin_SendFrame、Lin_GetStatus 及 Lin_GoToSleep 由 LIN_MASTER_API_SUPPORT 控制，并用于主节点。从节点的帧头处理和收发完成通过驱动向 LinIf 的回调接口交互。

2.2.3 休眠与唤醒

下列接口均以 Channel 指定本地通道，返回 Std_ReturnType：E_OK 表示对应请求被接受或检查可执行，E_NOT_OK 表示失败。通道休眠与 MCU 进入低功耗是不同层面的操作。

API	介绍说明
Lin_GoToSleep	Std_ReturnType Lin_GoToSleep(uint8 Channel); 由主节点发送总线休眠命令并进入休眠处理流程。当前实现将通道置为休眠待处理状态，后续 Lin_GetStatus() 将其转为休眠状态并返回 LIN_CH_SLEEP。E_OK 不代表远端节点或 MCU 已进入低功耗。
Lin_GoToSleepInternal	Std_ReturnType Lin_GoToSleepInternal(uint8 Channel); 使本地 LIN 通道进入休眠状态，不在总线上发送休眠命令。是否启用唤醒检测及硬件低功耗能力，取决于配置和外设支持；MCU 的低功耗进入由系统电源管理负责。
Lin_Wakeup	Std_ReturnType Lin_Wakeup(uint8 Channel); 在已初始化且通道处于休眠状态时，发送总线唤醒脉冲并恢复本地通道。主从节点均可请求总线唤醒；正常帧通信仍由主节点调度。E_OK 不保证其他节点已就绪。
Lin_WakeupInternal	Std_ReturnType Lin_WakeupInternal(uint8 Channel); 在已初始化且通道处于休眠状态时，仅恢复本地通道，不发送总线唤醒脉冲，也不自动启动主节点调度。适用于只需恢复本地通信状态的场景。
Lin_CheckWakeup	Std_ReturnType Lin_CheckWakeup(uint8 Channel); 检查通道的外部唤醒事件。在当前实现中，休眠通道检测到事件后通过 EcuM_SetWakeupEvent() 和 LinIf_WakeupConfirmation() 上报。有效调用也可能返回 E_OK 而未发现事件，不能仅凭返回值判定已检测到唤醒。

LIN 总线唤醒不等同于 MCU 低功耗唤醒，后者还需要收发器和芯片唤醒路径配合。

3. LIN模块应用实例1（LINFlexd 主机+从机）

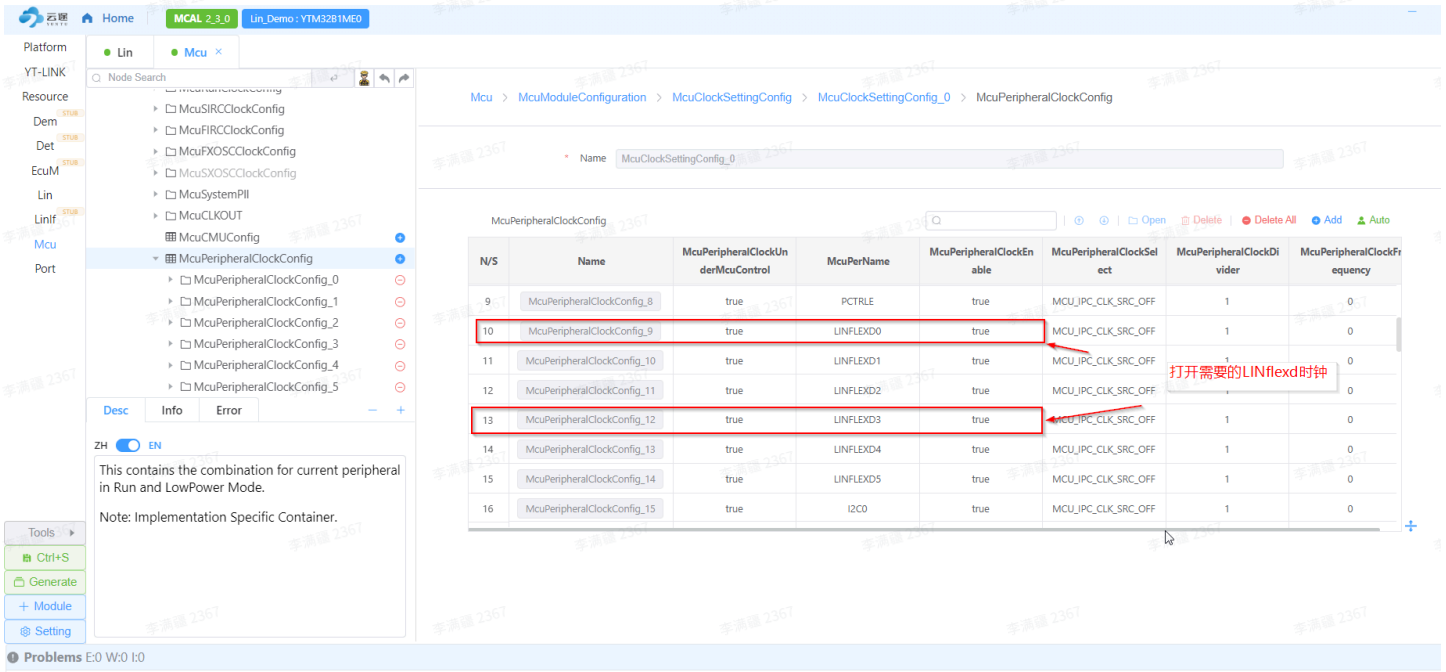
3.1 应用场景

- 基于ME05 EVB开发板，本工程配置了两个Lin通道(LINFlexD0, LINFlexD3)，Lin1(LINFlexD0)作为master节点，连续向Lin总线发送帧，LIN3(LINFlexD3)作为slave节点，从节点根据帧 ID 接收或发送响应。
- 配置波特率为19200
- 调用Lin_SendFrame发送了ID从0x1F到0x28的LIN_MASTER_RESPONSE类型的帧; 发送ID从0x29到0x32的LIN_SLAVE_RESPONSE类型的帧，其中ID为0x2E、0x30、0x32的帧slave节点不响应。
- LINFlexD3作为从机禁止以下ID通行:0x01,0x22,0x30,0x12,0x26,0x2E。

3.2 配置步骤

3.2.1 时钟配置

在MCU模块中配置LIN的reference clock。

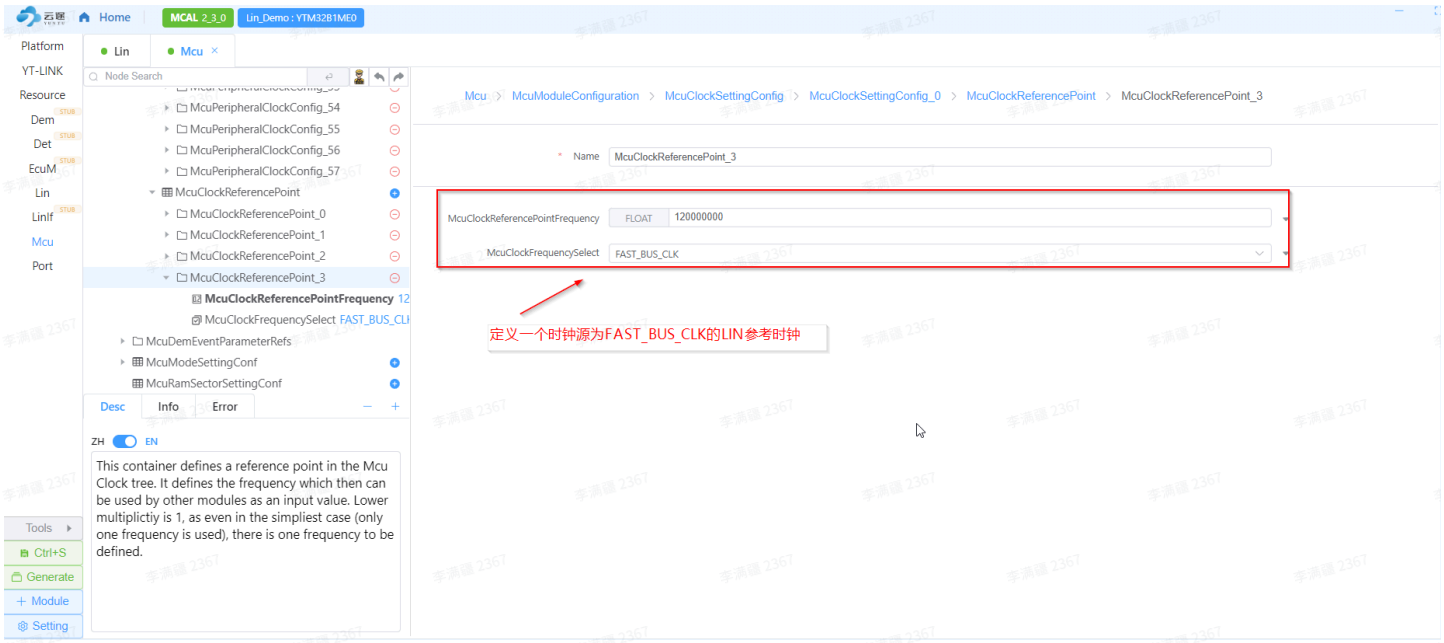


Tips:

ME05 Linflexd 时钟默认为SLOW_BUS_CLK

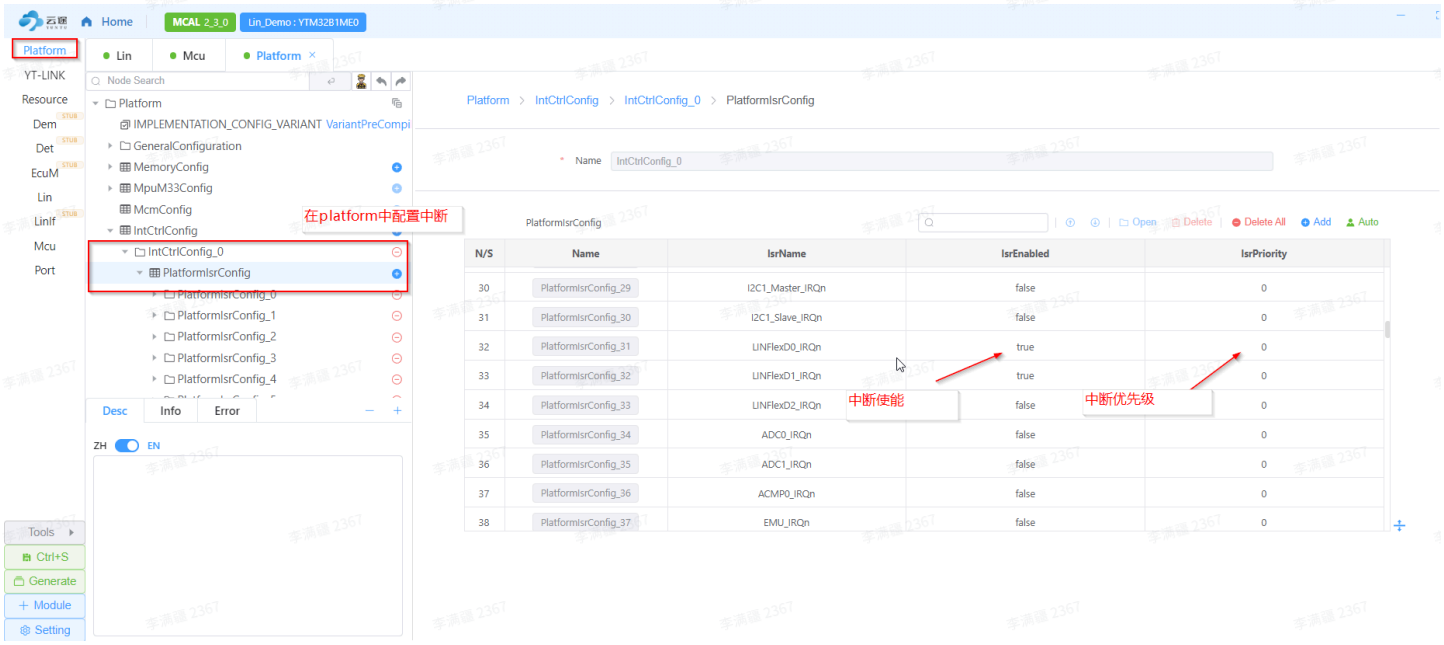
Table 13.1 continued from previous page

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



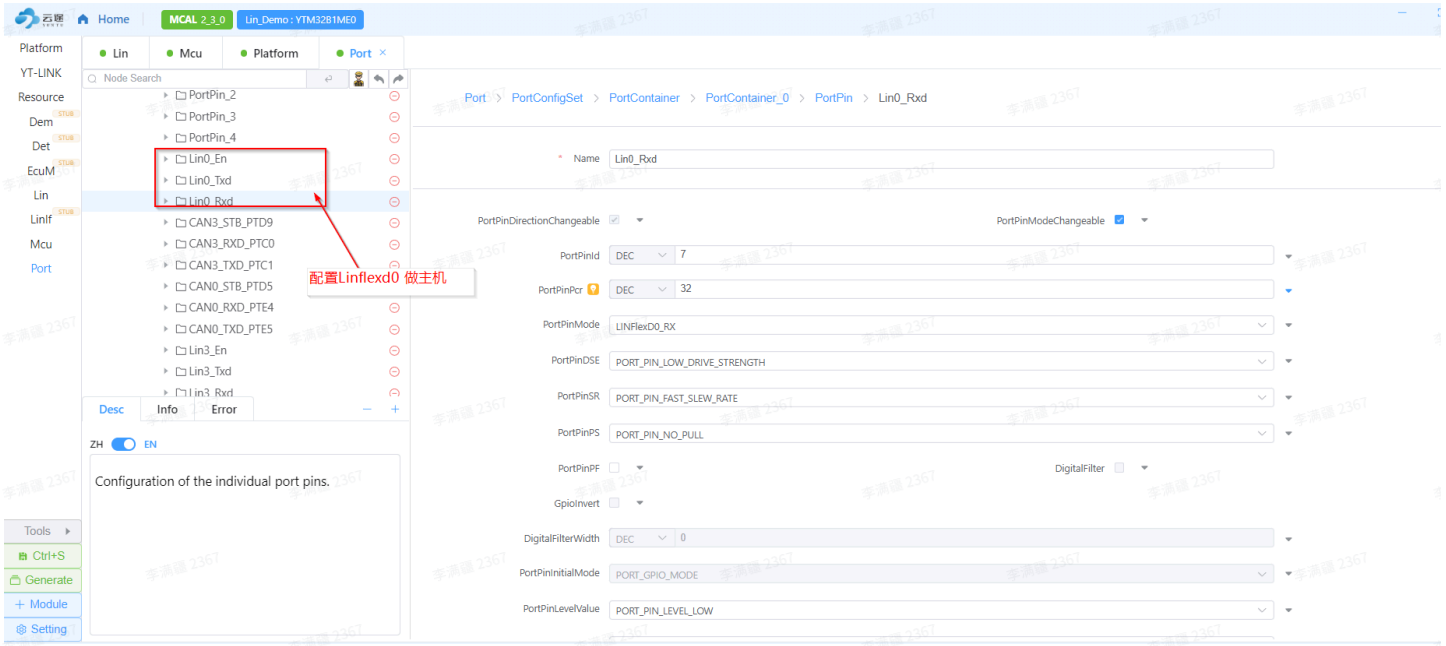
3.2.2 中断配置

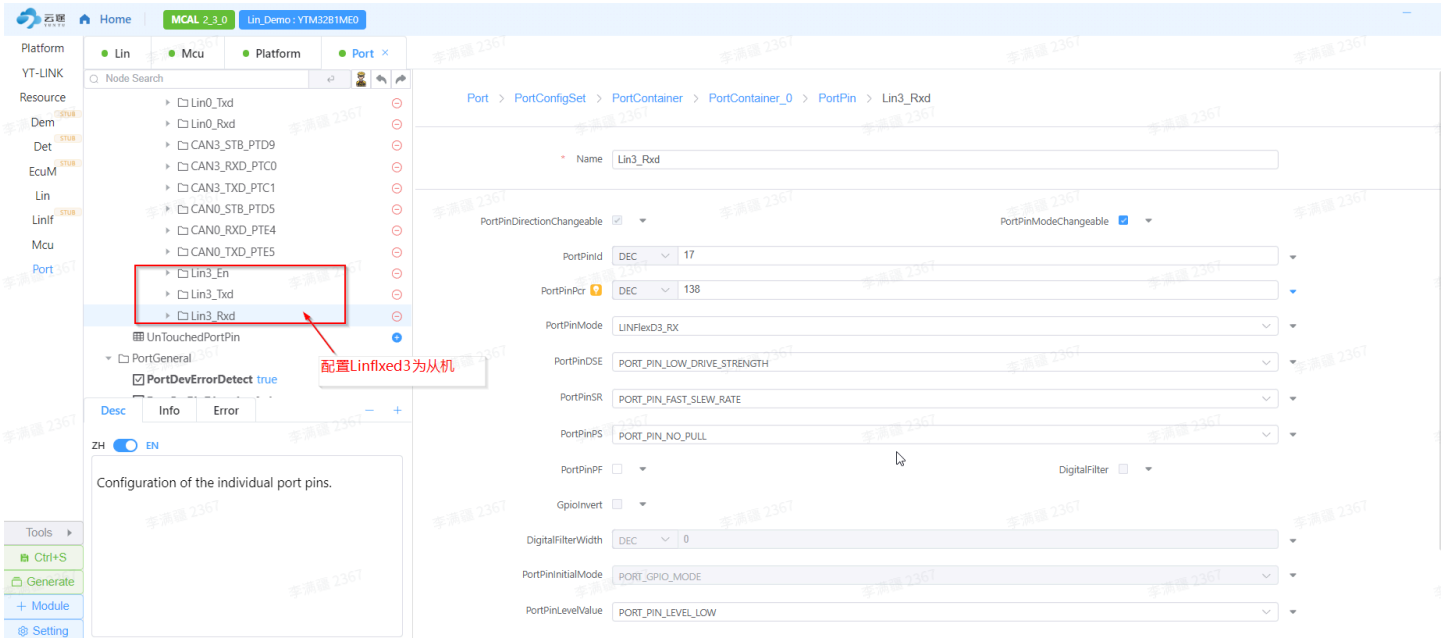
根据 LINFlexD 驱动的收发方式配置所需中断，在 Platform 中设置对应中断源和处理入口；使用 OS 时，还应完成 OS 中的中断配置。



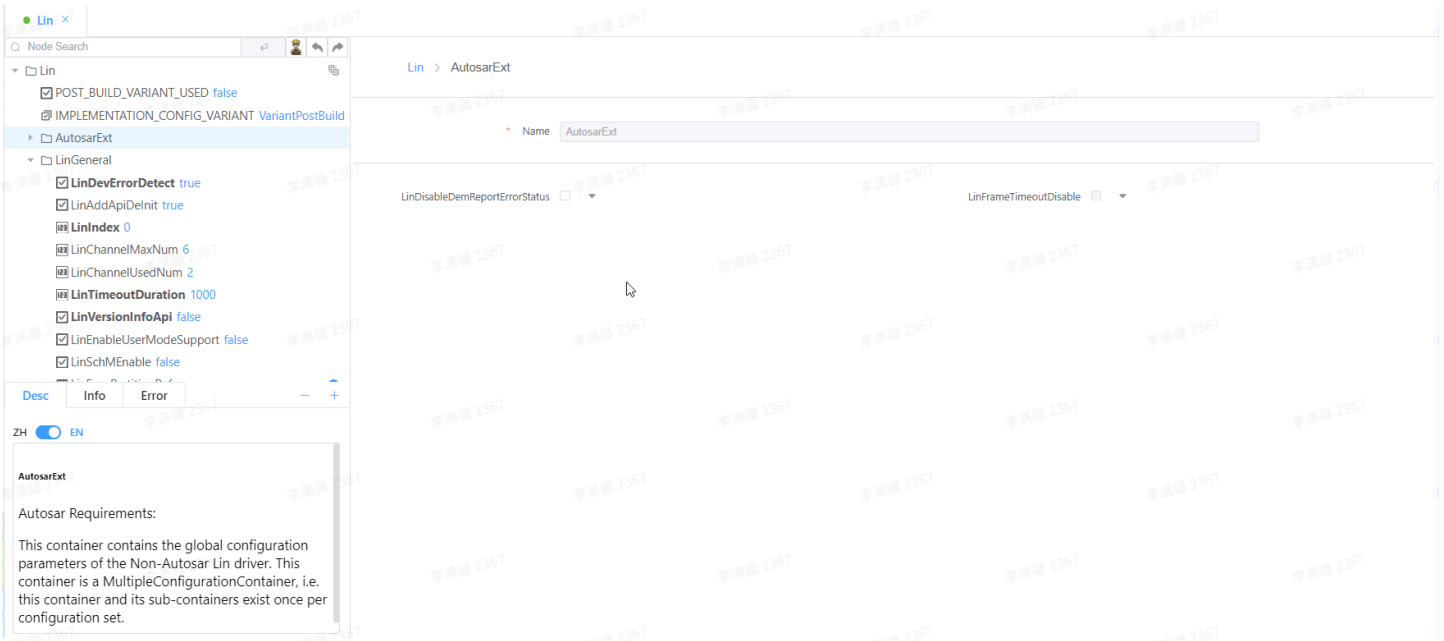
3.2.3 Port配置

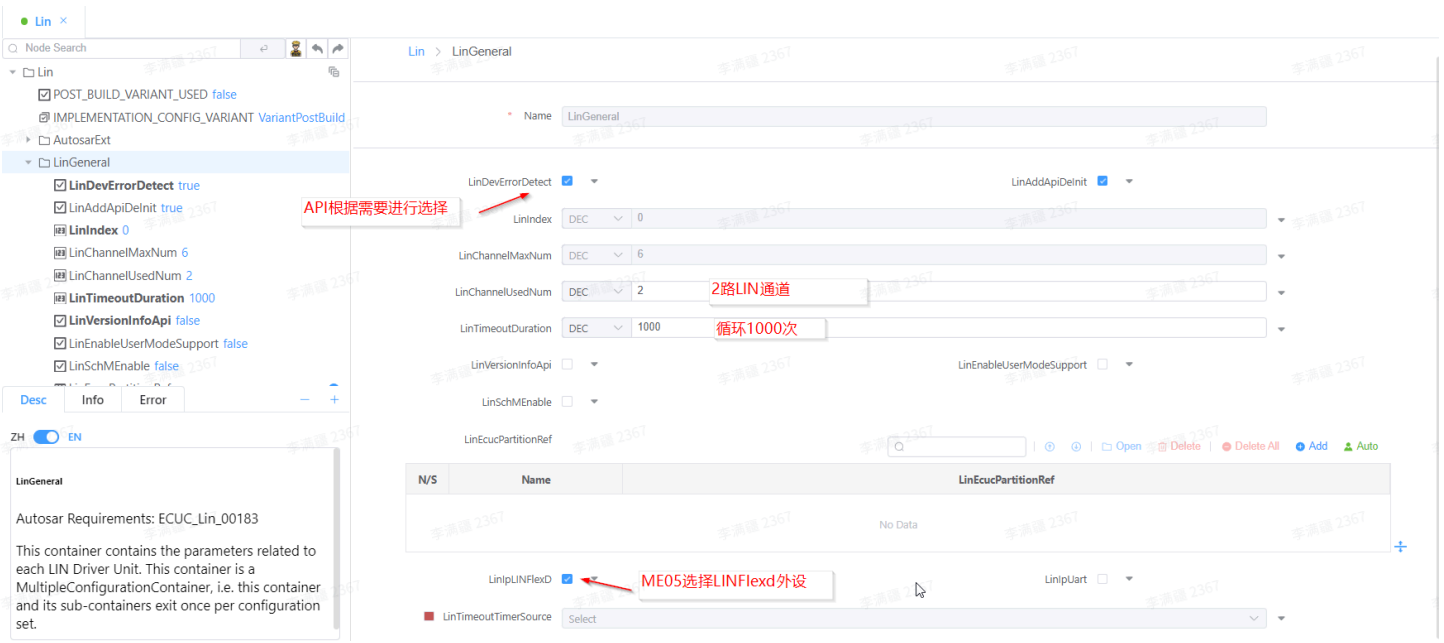
配置两路 LIN 的 TX/RX 复用及收发器控制引脚，见下图。

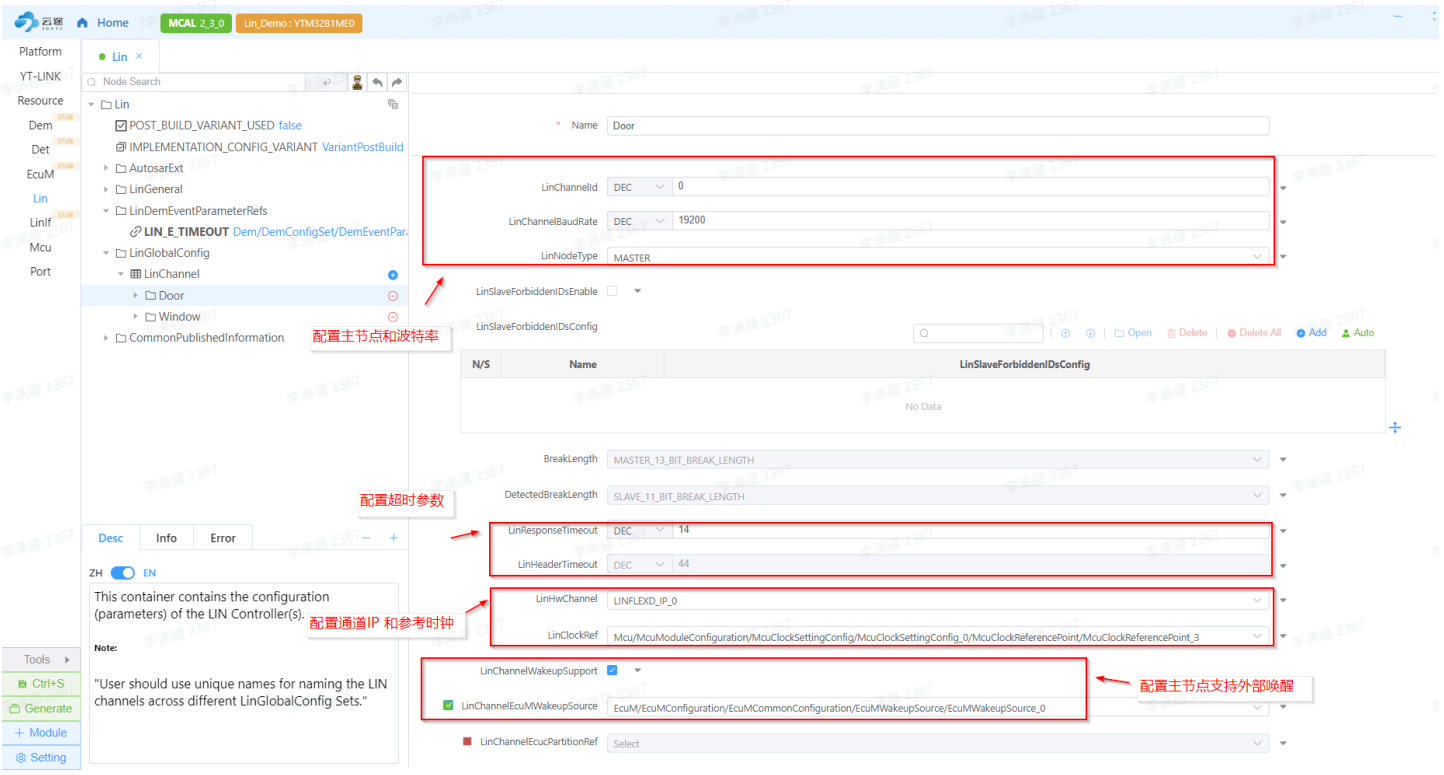

Home
MCAL 2_3_0
Lin_Demo : YTM32B1ME0
Platform
YT-LINK
Resource
Dem
Det
EcuM
Lin
LinIf
Mcu
Port
Lin
Mcu
Platform
Port
Node Search
PortPin_2
PortPin_3
PortPin_4
Lin0_En
Lin0_Txd
Lin0_Rxd
CAN3_STB_PTD9
CAN3_RXD_PTC0
CAN3_TXD_PTC1
CAN0_STB_PTD5
CAN0_RXD_PTE4
CAN0_TXD_PTE5
Lin3_En
Lin3_Txd
Lin3_Rxd
配置Linflexd0 做主机
Desc
Info
Error
ZH
EN
Configuration of the individual port pins.
Tools
Ctrl+S
Generate
Module
Setting
Port > PortConfigSet > PortContainer > PortContainer_0 > PortPin > Lin0_Rxd
Name
Lin0_Rxd
PortPinDirectionChangeable
PortPinModeChangeable
PortPinId
DEC
7
PortPinPcr
DEC
32
PortPinMode
LINFlexD0_RX
PortPinDSE
PORT_PIN_LOW_DRIVE_STRENGTH
PortPinSR
PORT_PIN_FAST_SLEW_RATE
PortPinPS
PORT_PIN_NO_PULL
PortPinPF
DigitalFilter
GpioInvert
DigitalFilterWidth
DEC
0
PortPinInitialMode
PORT_GPIO_MODE
PortPinLevelValue
PORT_PIN_LEVEL_LOW

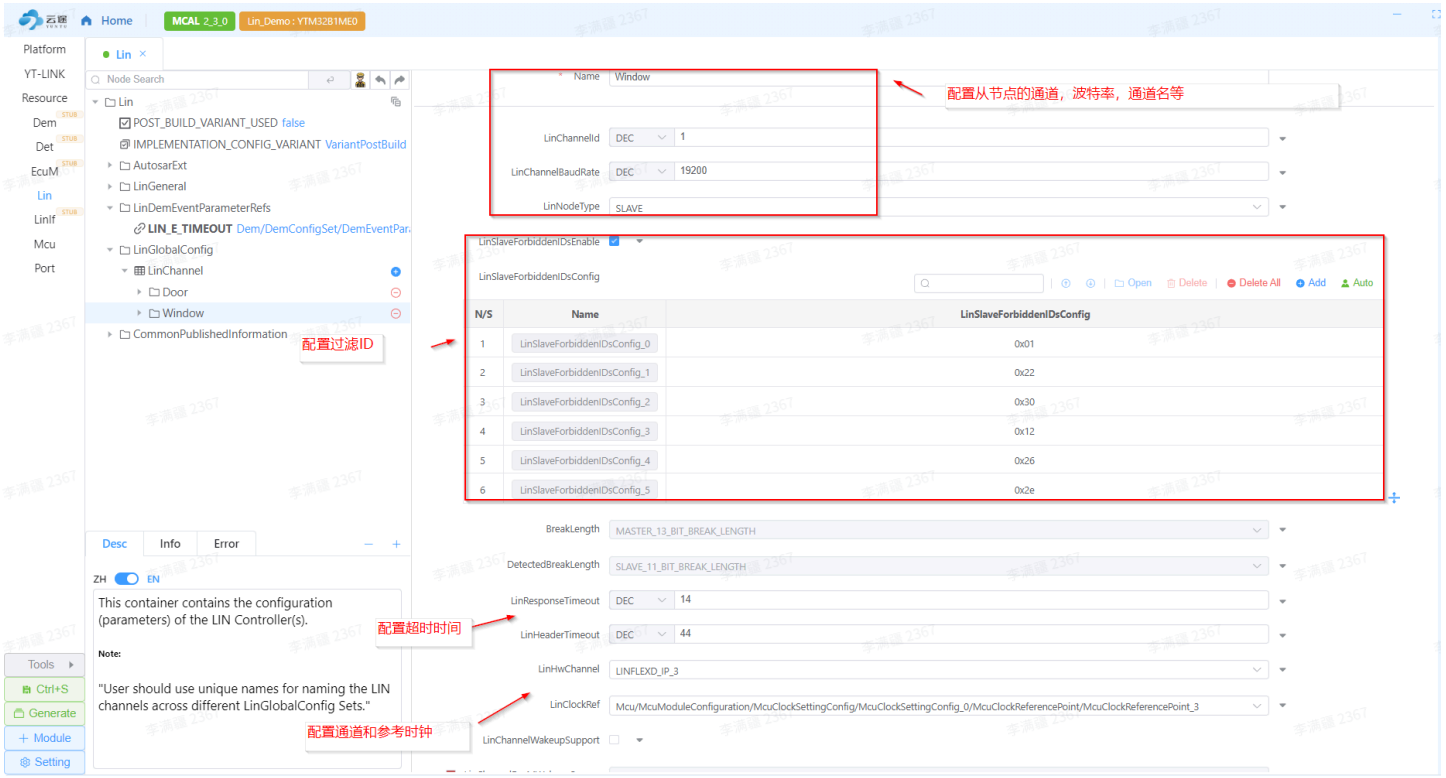

Home
MCAL 2_3_0
Lin_Demo : YTM32B1ME0
Platform
YT-LINK
Resource
Dem
Det
EcuM
Lin
LinIf
Mcu
Port
Lin
Mcu
Platform
Port
Node Search
Lin0_Txd
Lin0_Rxd
CAN3_STB_PTD9
CAN3_RXD_PTC0
CAN3_TXD_PTC1
CAN0_STB_PTD5
CAN0_RXD_PTE4
CAN0_TXD_PTE5
Lin3_En
Lin3_Txd
Lin3_Rxd
UnTouchedPortPin
PortGeneral
PortDevErrorDetect true
配置Linflxed3为从机
Desc
Info
Error
ZH
EN
Configuration of the individual port pins.
Tools
Ctrl+S
Generate
Module
Setting
Port > PortConfigSet > PortContainer > PortContainer_0 > PortPin > Lin3_Rxd
Name
Lin3_Rxd
PortPinDirectionChangeable
PortPinModeChangeable
PortPinId
DEC
17
PortPinPcr
DEC
138
PortPinMode
LINFlexD3_RX
PortPinDSE
PORT_PIN_LOW_DRIVE_STRENGTH
PortPinSR
PORT_PIN_FAST_SLEW_RATE
PortPinPS
PORT_PIN_NO_PULL
PortPinPF
DigitalFilter
GpioInvert
DigitalFilterWidth
DEC
0
PortPinInitialMode
PORT_GPIO_MODE
PortPinLevelValue
PORT_PIN_LEVEL_LOW

3.2.4 配置LIN模块


Lin
Node Search
Lin
POST_BUILD_VARIANT_USED false
IMPLEMENTATION_CONFIG_VARIANT VariantPostBuild
AutosarExt
LinGeneral
LinDevErrorDetect true
LinAddApiDeInit true
LinIndex 0
LinChannelMaxNum 6
LinChannelUsedNum 2
LinTimeoutDuration 1000
LinVersionInfoApi false
LinEnableUserModeSupport false
LinSchMEnable false
Desc
Info
Error
ZH
EN
AutosarExt
Autosar Requirements:
This container contains the global configuration parameters of the Non-Autosar Lin driver. This container is a MultipleConfigurationContainer, i.e. this container and its sub-containers exist once per configuration set.
Lin > AutosarExt
Name AutosarExt
LinDisableDemReportErrorStatus
LinFrameTimeoutDisable


Lin
Node Search
Lin
POST_BUILD_VARIANT_USED false
IMPLEMENTATION_CONFIG_VARIANT VariantPostBuild
AutosarExt
LinGeneral
LinDevErrorDetect true
LinAddApiDeInit true
LinIndex 0
LinChannelMaxNum 6
LinChannelUsedNum 2
LinTimeoutDuration 1000
LinVersionInfoApi false
LinEnableUserModeSupport false
LinSchMEnable false
Desc
Info
Error
ZH
EN
LinGeneral
Autosar Requirements: ECUC_Lin_00183
This container contains the parameters related to each LIN Driver Unit. This container is a MultipleConfigurationContainer, i.e. this container and its sub-containers exit once per configuration set.
Lin > LinGeneral
Name LinGeneral
LinDevErrorDetect
LinAddApiDeInit
API根据需要进行选择
LinIndex DEC 0
LinChannelMaxNum DEC 6
LinChannelUsedNum DEC 2
2路LIN通道
LinTimeoutDuration DEC 1000
循环1000次
LinVersionInfoApi
LinEnableUserModeSupport
LinSchMEnable
LinEcucPartitionRef
Open
Delete
Delete All
Add
Auto
N/S
Name
LinEcucPartitionRef
No Data
LinIpLINFlexD
ME05选择LINFlexd外设
LinIpUart
LinTimeoutTimerSource
Select


Home
MCAL 2_3_0
Lin_Demo : YTM32B1ME0
Platform
YT-LINK
Resource
Dem
Det
EcuM
Lin
LinIf
Mcu
Port
Lin
Node Search
Lin
POST_BUILD_VARIANT_USED false
IMPLEMENTATION_CONFIG_VARIANT VariantPostBuild
AutosarExt
LinGeneral
LinDemEventParameterRefs
LIN_E_TIMEOUT Dem/DemConfigSet/DemEventPar
LinGlobalConfig
LinChannel
Door
Window
CommonPublishedInformation
配置主节点和波特率
Desc
Info
Error
ZH
EN
This container contains the configuration (parameters) of the LIN Controller(s).
Note:
"User should use unique names for naming the LIN channels across different LinGlobalConfig Sets."
Name Door
LinChannelId DEC 0
LinChannelBaudRate DEC 19200
LinNodeType MASTER
LinSlaveForbiddenIDsEnable
LinSlaveForbiddenIDsConfig
Open
Delete
Delete All
Add
Auto
N/S
Name
LinSlaveForbiddenIDsConfig
No Data
BreakLength MASTER_13_BIT_BREAK_LENGTH
DetectedBreakLength SLAVE_11_BIT_BREAK_LENGTH
配置超时参数
LinResponseTimeout DEC 14
LinHeaderTimeout DEC 44
LinHwChannel LINFLEXD_IP_0
配置通道IP 和参考时钟
LinClockRef Mcu/McuModuleConfiguration/McuClockSettingConfig/McuClockSettingConfig_0/McuClockReferencePoint/McuClockReferencePoint_3
LinChannelWakeupSupport
LinChannelEcuMWakeupSource EcuM/EcuMConfiguration/EcuMCommonConfiguration/EcuMWakeupSource/EcuMWakeupSource_0
配置主节点支持外部唤醒
LinChannelEcucPartitionRef Select
Tools
Ctrl+S
Generate
Module
Setting



3.3 API说明

- **主从收发流程**

本例 Door 为逻辑通道 0，对应 LINFlexD0；Window 为逻辑通道 1，对应 LINFlexD3。以下两种响应方向均由 Door 发送帧头。

- **主节点发布：以ID=0x1F为例**

1. 主机构造 PDU：Pid=0x1F、Drc=TX、Dl=8、Cs=增强校验，数据指针指向主机发送矩阵。
2. 应用调用 Lin_SendFrame(Door, ...)，驱动发送 Break、Sync、PID，再发送数据与校验和。
3. 从机识别 ID=0x1F，在帧头回调中选择 RX。接收完成后，LinIf_RxIndication() 按示例逻辑更新应用数据。

- **从节点发布：以ID=0x29为例**

1. 主机构造 PDU：Pid=0xE9、Drc=RX、Dl=8、Cs=增强校验，调用 Lin_SendFrame(Door, ...)。
2. 主机只发送帧头并等待响应。从机回调提取 ID=0x29，准备数据并设置 Drc=TX，随后发送响应。
3. 主机查询 Lin_GetStatus()；只有响应接收成功后才能按接口要求使用接收数据。发送请求被接受，不等于整帧已完成。

- **接口调用与回调实现**

以下代码展示 PDU 数组、初始化及回调处理。Lin_SendFrame() 返回请求处理结果，整帧状态通过 Lin_GetStatus() 等接口确认。

main.c 9+

Lin_Demo › app › main.c › main

```c
/* Private variables -----------------------------------------------------------*/
/* USER CODE BEGIN PV */
uint8 *lin_RxData;

uint8 lin_TxDataMatrix[10][8] =
{
    {0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x01},
    {0x31, 0x42, 0x53, 0x64, 0x78, 0x69, 0x37, 0x02},
    {0x21, 0x92, 0x03, 0x13, 0x22, 0x06, 0x07, 0x04},
    {0x31, 0x32, 0x04, 0x04, 0x33, 0xF6, 0x07, 0x08},
    {0x41, 0xB2, 0x05, 0x33, 0x05, 0xd6, 0x57, 0x10},
    {0x77, 0xA2, 0x06, 0x04, 0x7D, 0xE6, 0x07, 0x18},
    {0x99, 0xF2, 0x07, 0x58, 0x05, 0x06, 0x67, 0x20},
    {0xBB, 0xE2, 0x08, 0x32, 0x8F, 0xD6, 0x07, 0x28},
    {0xDD, 0xD2, 0x09, 0xA4, 0x05, 0x06, 0xB7, 0x30},
    {0xFF, 0xC2, 0xAA, 0x24, 0x66, 0xA6, 0x07, 0x40},
};

uint8 lin_Slave_TxDataMatrix[10][8] =
{
    {0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x11, 0x10},
    {0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x22, 0x10},
    {0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x33, 0x10},
    {0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x44, 0x10},
    {0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x55, 0x10},
    {0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x66, 0x10},
    {0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x77, 0x10},
    {0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x88, 0x10},
    {0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x99, 0x10},
    {0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0xAA, 0x10},
};
```

LIN MASTER 节点发送数据矩阵

LIN SLAVE 节点发送数据矩阵

main.c 9+

Lin_Demo › app › main.c › main

```c
const Lin_PduType linTxPduMatrix[10] =
{
    {
        .Pid = 0x1F,
        .Cs = LIN_ENHANCED_CS,
        .Drc = LIN_FRAMERESPONSE_TX,
        .Dl = 8U,
        .SduPtr = lin_TxDataMatrix[0],
    },
    {
        .Pid = 0x20,
        .Cs = LIN_ENHANCED_CS,
        .Drc = LIN_FRAMERESPONSE_TX,
        .Dl = 8U,
        .SduPtr = lin_TxDataMatrix[1],
    },
    {
        .Pid = 0x61,
        .Cs = LIN_ENHANCED_CS,
        .Drc = LIN_FRAMERESPONSE_TX,
        .Dl = 8U,
        .SduPtr = lin_TxDataMatrix[2],
    },
    {
        .Pid = 0xE2,
        .Cs = LIN_ENHANCED_CS,
```

Master发送slave接收 Pdu定义

main.c 9+

Lin_Demo › app › main.c › main

```c
const Lin_PduType linRxPduMatrix[10] =
{
    {
        .Pid = 0xE9,
        .Cs = LIN_ENHANCED_CS,
        .Drc = LIN_FRAMERESPONSE_RX,
        .Dl = 8U,
        .SduPtr = NULL_PTR,
    },
    {
        .Pid = 0x6A,
        .Cs = LIN_ENHANCED_CS,
        .Drc = LIN_FRAMERESPONSE_RX,
        .Dl = 8U,
        .SduPtr = NULL_PTR,
    },
    {
        .Pid = 0x2B,
        .Cs = LIN_ENHANCED_CS,
        .Drc = LIN_FRAMERESPONSE_RX,
        .Dl = 8U,
        .SduPtr = NULL_PTR,
    },
    {
        .Pid = 0xEC,
        .Cs = LIN_ENHANCED_CS,
```

Master接收Slave 发送 Pdu 定义

```c
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
Std_ReturnType LinIf_HeaderIndication(NetworkHandleType Channel, Lin_PduType *PduPtr)
{
    Std_ReturnType RetVal = E_NOT_OK;
    uint8 loopVar;
    if (LinConf_LinChannel_Window == Channel)
    {
        if ((0x29U <= (PduPtr->Pid & 0x3fU)) && (0x31U >= (PduPtr->Pid & 0x3fU)))
        {
            PduPtr->Cs = LIN_ENHANCED_CS;
            PduPtr->Dl = 8U;
            for (loopVar = 0u; loopVar < PduPtr->Dl; loopVar ++)
            {
                PduPtr->SduPtr[loopVar] = lin_Slave_TxDataMatrix[(PduPtr->Pid & 0x3fU) - 0x29U][loopVar];
            }
            PduPtr->Drc = LIN_FRAMERESPONSE_TX;
            RetVal = E_OK;
        }
        if ((0x1fU <= (PduPtr->Pid & 0x3fU)) && (0x28U >= (PduPtr->Pid & 0x3fU)))
        {
            PduPtr->Cs = LIN_ENHANCED_CS;
            PduPtr->Drc = LIN_FRAMERESPONSE_RX;
            Current_ID = PduPtr->Pid & 0x3fU;
            RetVal = E_OK;
        }
    }
    return RetVal;
}

void LinIf_RxIndication(NetworkHandleType Channel, uint8 *Lin_SduPtr)
{
    if (LinConf_LinChannel_Window == (uint8)Channel)
    {
        lin_Slave_TxDataMatrix[Current_ID - 0x1fU][0] += Lin_SduPtr[7];
    }
}
/* USER CODE END 0 */


/**
```

从节点帧头回调函数

ID 在0x29 和0x31 之间为slave 发送

ID在0x1F和0x28之间为slave接收

从节点接收完成后，对发送矩阵每行第一个字节自加1后填充slave发送数据，

```c
static void Board_Init(void)
{
    Lin_Init(&Lin_Config);
    Platform_Init(NULL_PTR);
    Port_Init(&Port_Config);
}

/* USER CODE BEGIN 4 */
/* USER CODE END 4 */

```

LIN初始化

main.c 9+

Lin_Demo › app › main.c › main

```c
int main(void)
{
    /* USER CODE BEGIN 1 */
    Mcu_Init(&Mcu_Config);
    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 */
    Lin_Wakeup(LinConf_LinChannel_Door);
    Lin_WakeupInternal(LinConf_LinChannel_Window);
    /* USER CODE END 2 */

    /* Infinite loop */
```

master 主节点唤醒网络；
slave 从节点唤醒本节点；

```c
    while (1)
    {
#if (LIN_MASTER_API_SUPPORT == STD_ON)
        Cnt++;
        if (Cnt > 150000U)
        {
            Cnt = 0;
            Lin_CheckWakeup(Channel: LinConf_LinChannel_Door);
            Lin_SendFrame(Channel: LinConf_LinChannel_Door, PduInfoPtr: linTxPduMatrix + DataLoop);
        }
        if ((Cnt % 1000U) == 0U)
        {
            LinStatus = Lin_GetStatus(Channel: LinConf_LinChannel_Door, Lin_SduPtr: &lin_RxData);
            if ((LinStatus == LIN_TX_OK) || (LinStatus == LIN_TX_HEADER_ERROR) || (LinStatus == LIN_TX_ERROR))
            {
                Lin_SendFrame(Channel: LinConf_LinChannel_Door, PduInfoPtr: linRxPduMatrix + DataLoop);
                DataLoop++;
                if (DataLoop >= 10)
                {
                    DataLoop = 0;
                }
            }
            else if (LinStatus == LIN_RX_OK)
            {
                for (uint8 Index = 0; Index < 8; Index ++)
                {
                    Buffer[Index] = *(lin_RxData + Index);
                }
                /* here process the data */
            }
        }
        /* USER CODE END WHILE */
        /* USER CODE BEGIN 3 */
#endif
    }
    /* USER CODE END 3 */
}
```

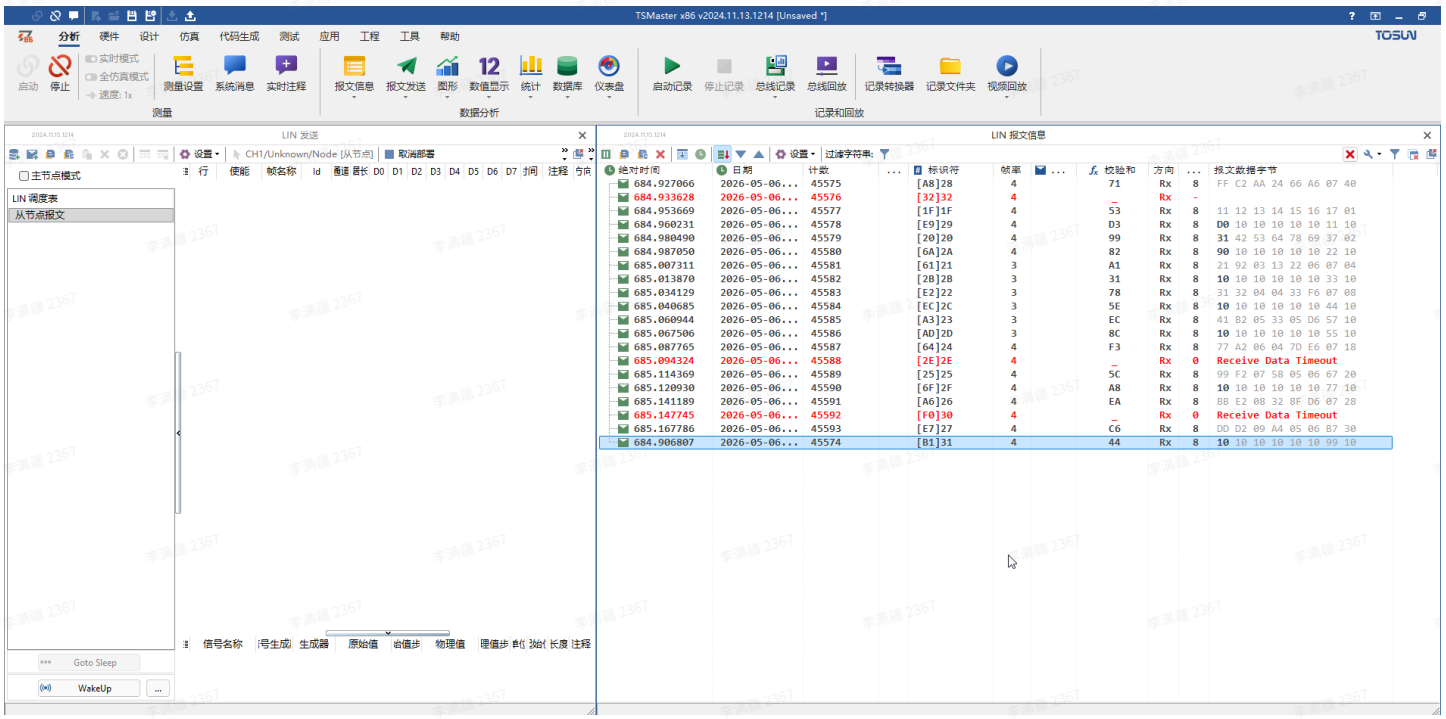
检查是否存在外部唤醒事件

发送数据，该函数只针对主节点

获取LIN总线节点状态信息

3.4 现象

使用EVB DEMO板LIN1（LINFlexD0）做主机，LIN3（LINFlexD3）做从机，使用TSmaster 上位机从机功能做数据监控，使用OZONE 观测slave发送的全局变量。

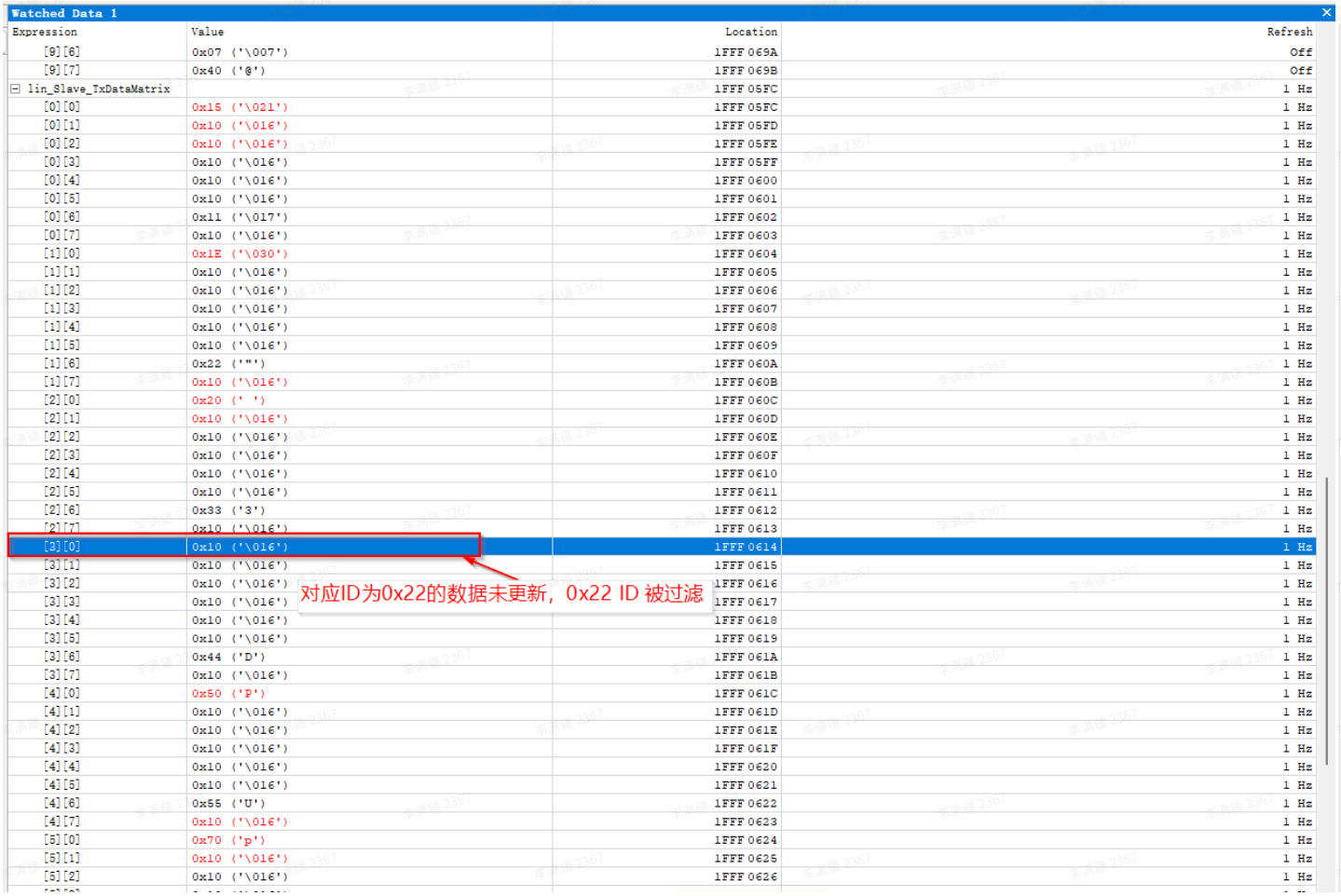


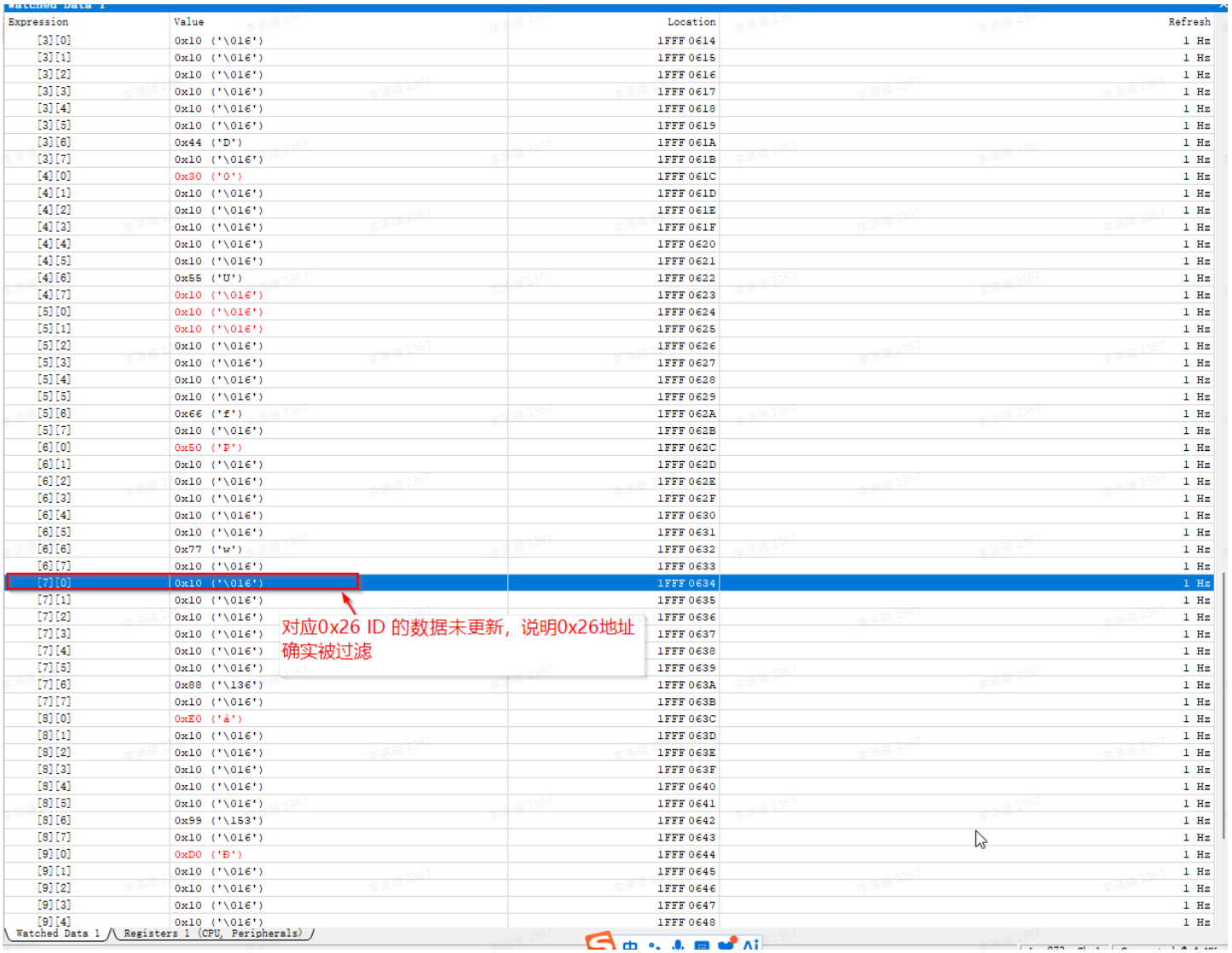
ID	响应方向	配置 / 原因	期望现象与原记录

0x01、0x12	未发送	禁止 ID 表包含；主机本例未发送	未覆盖
0x32	从机 → 主机	不在禁止表；LinIf_HeaderIndication() 仅允许 0x29～0x31 发送	回调拒绝响应，主机/监控端表现为响应超时；属于应用回调过滤。
0x1F～0x28，除 0x22、0x26	主机 → 从机	回调允许接收，未被禁止表过滤	应接收并按 LinIf_RxIndication() 更新矩阵；
0x22、0x26	主机 → 从机	禁止 ID 表过滤接收	主机仍可发送；从机不进入对应接收更新。
0x29～0x31，除 0x2E、0x30	从机 → 主机	回调允许发送，未被禁止表过滤	应返回 8 字节响应；
0x2E、0x30	从机 → 主机	禁止 ID 表阻止从机响应	无从机响应，主机/监控端表现为响应超时。根因是配置过滤，超时是观察结果。

排查结论：0x2E/0x30 与 0x32 的共同结果是“没有从机响应”，区别在于前者由配置禁止 ID，后者由回调范围拒绝。

0x22、0x26 属于主机发送场景：过滤影响从机的接收与应用更新，从下图OZONE数据观测中可以看到确实过滤成功。





4. LIN模块应用实例2(UART LIN主机)

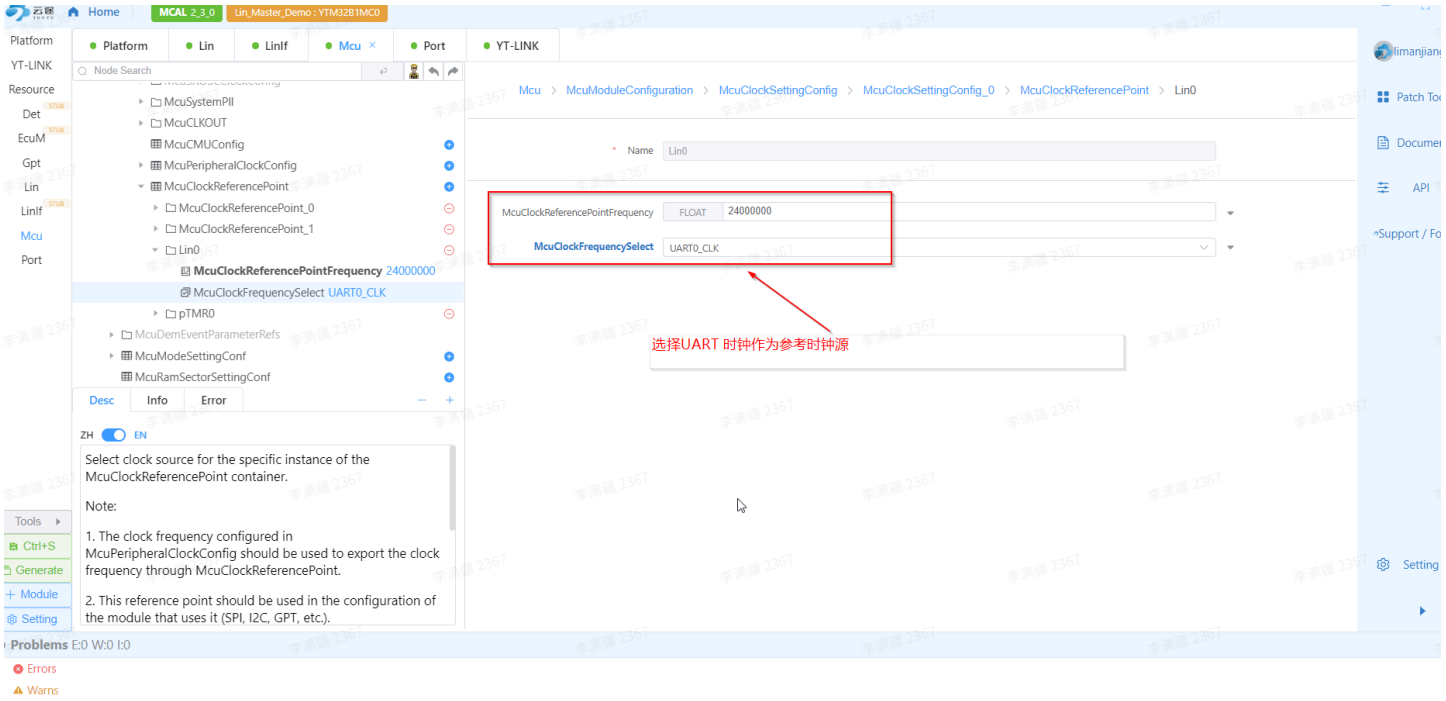
4.1 应用场景

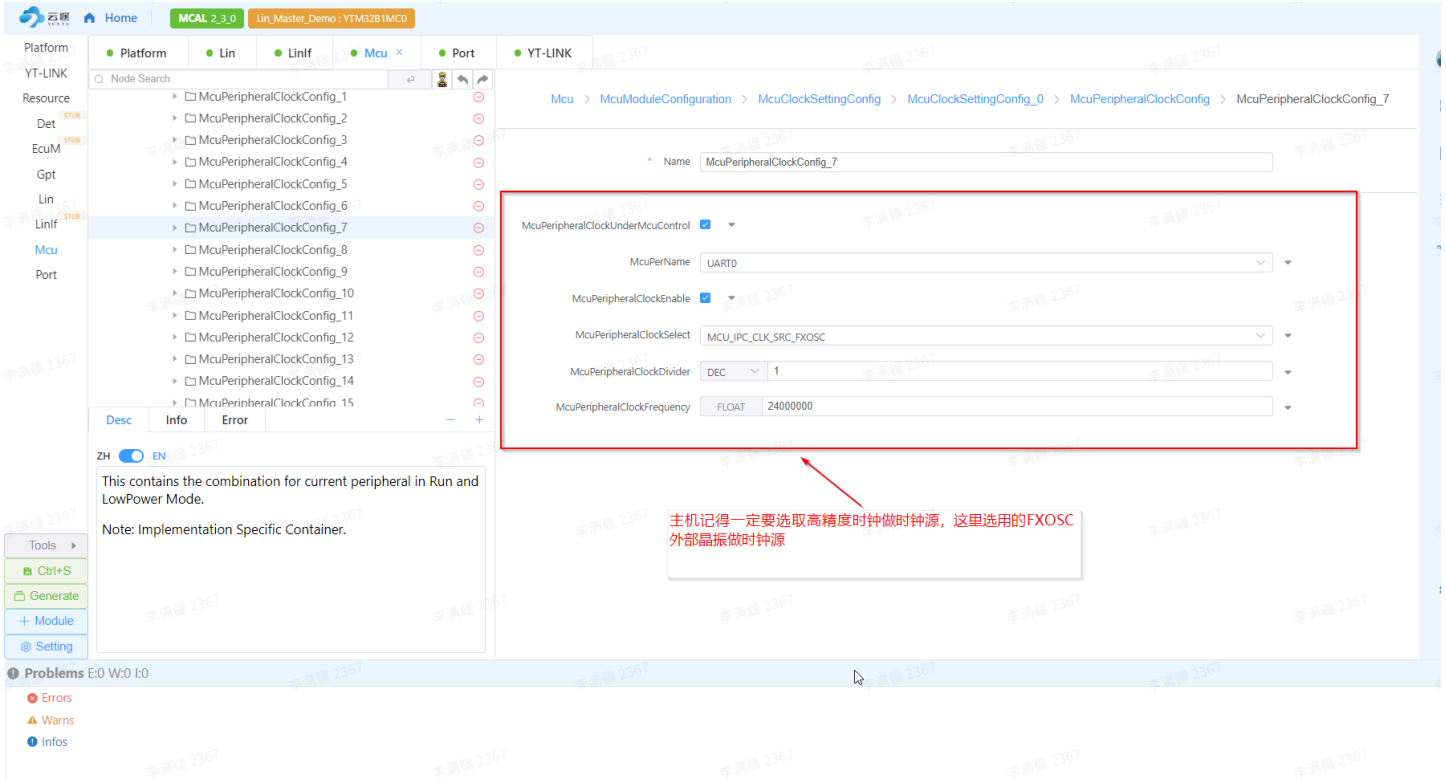
- 基于MC0 Demo板，MC03 使用UART 外设来实现LIN master功能，上位机作为从节点。
- 配置波特率为19200
- ID=0x1F～0x28 时，开发板发送帧头和响应；ID=0x29～0x32 时，开发板发送帧头并接收上位机的响应。

4.2 配置步骤

4.2.1 时钟配置

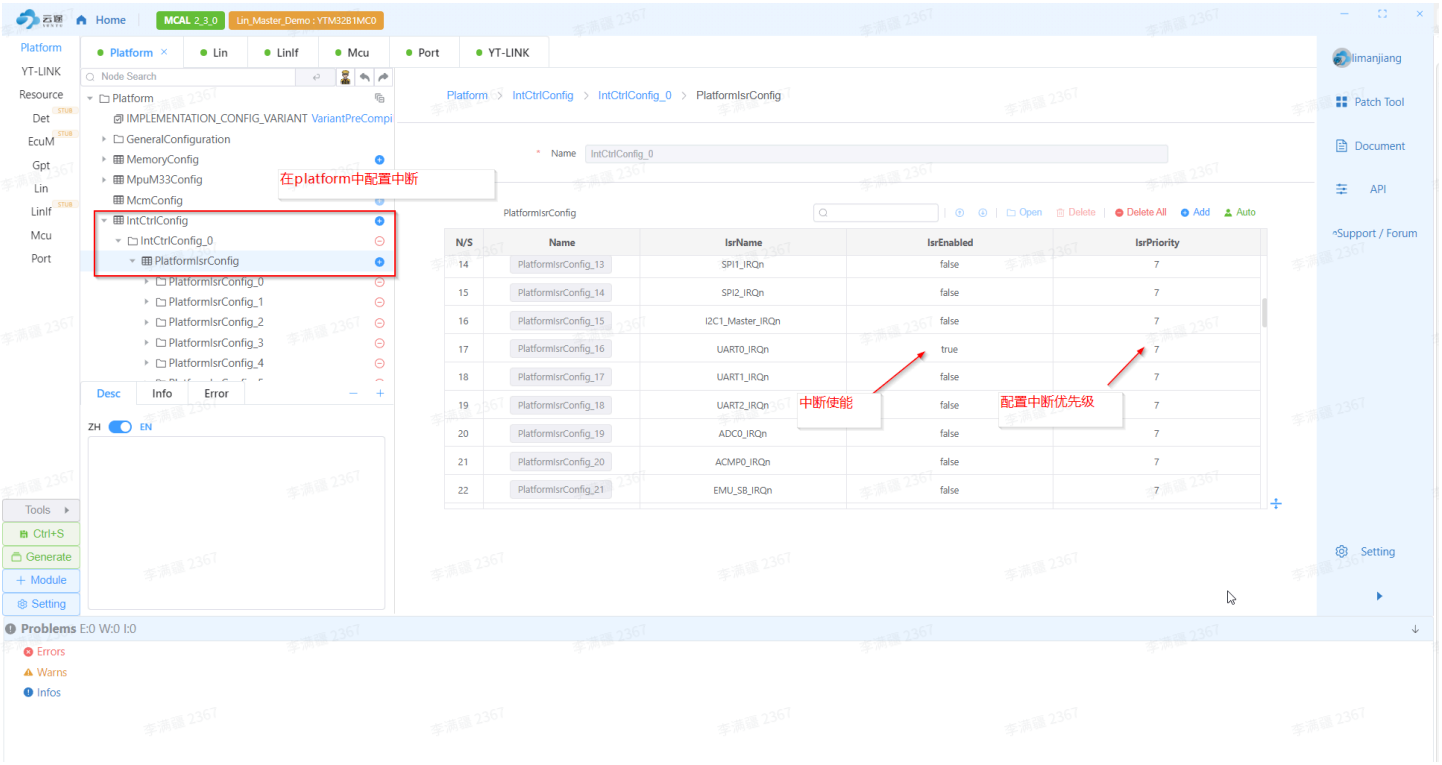
在MCU模块中配置LIN的reference clock。





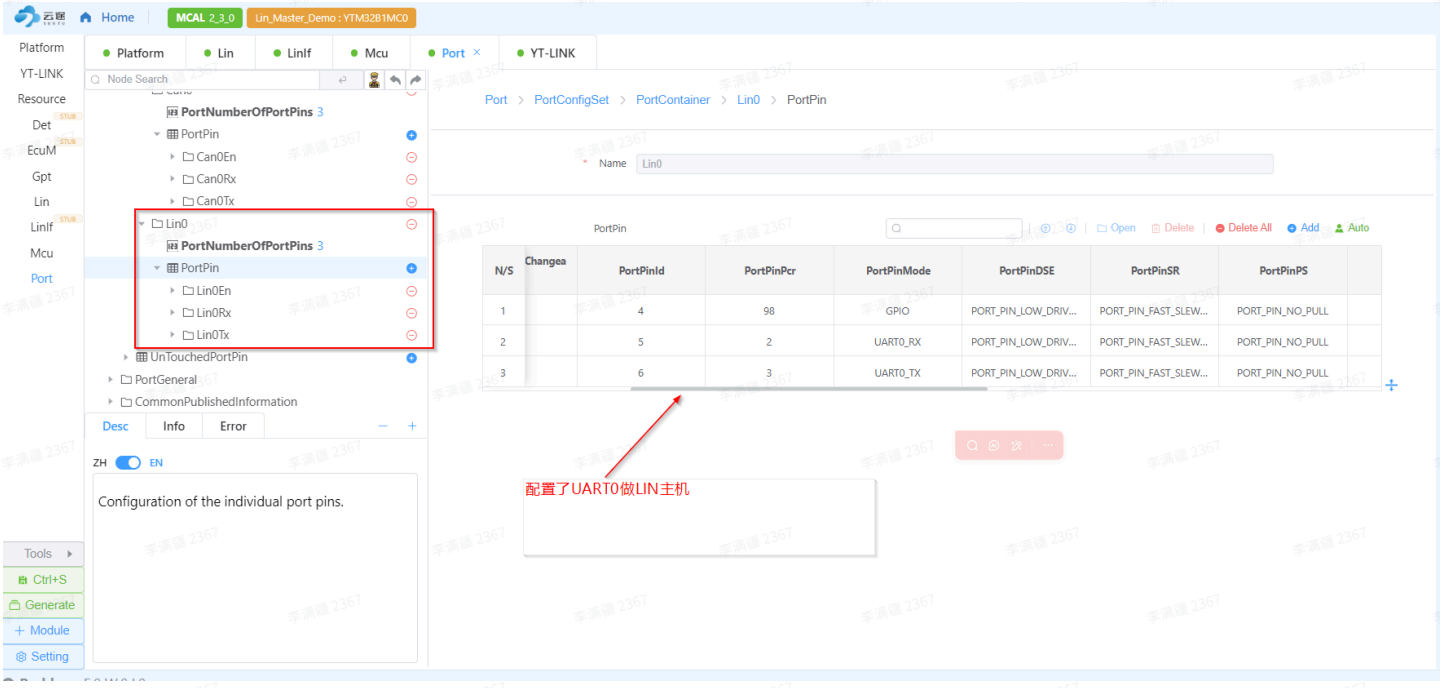
4.2.2 中断配置

根据需求选择是否使能中断，如若需要中断，则在Platform中打开中断，若使用了OS，则需要在OS中配置LIN的外设中断。

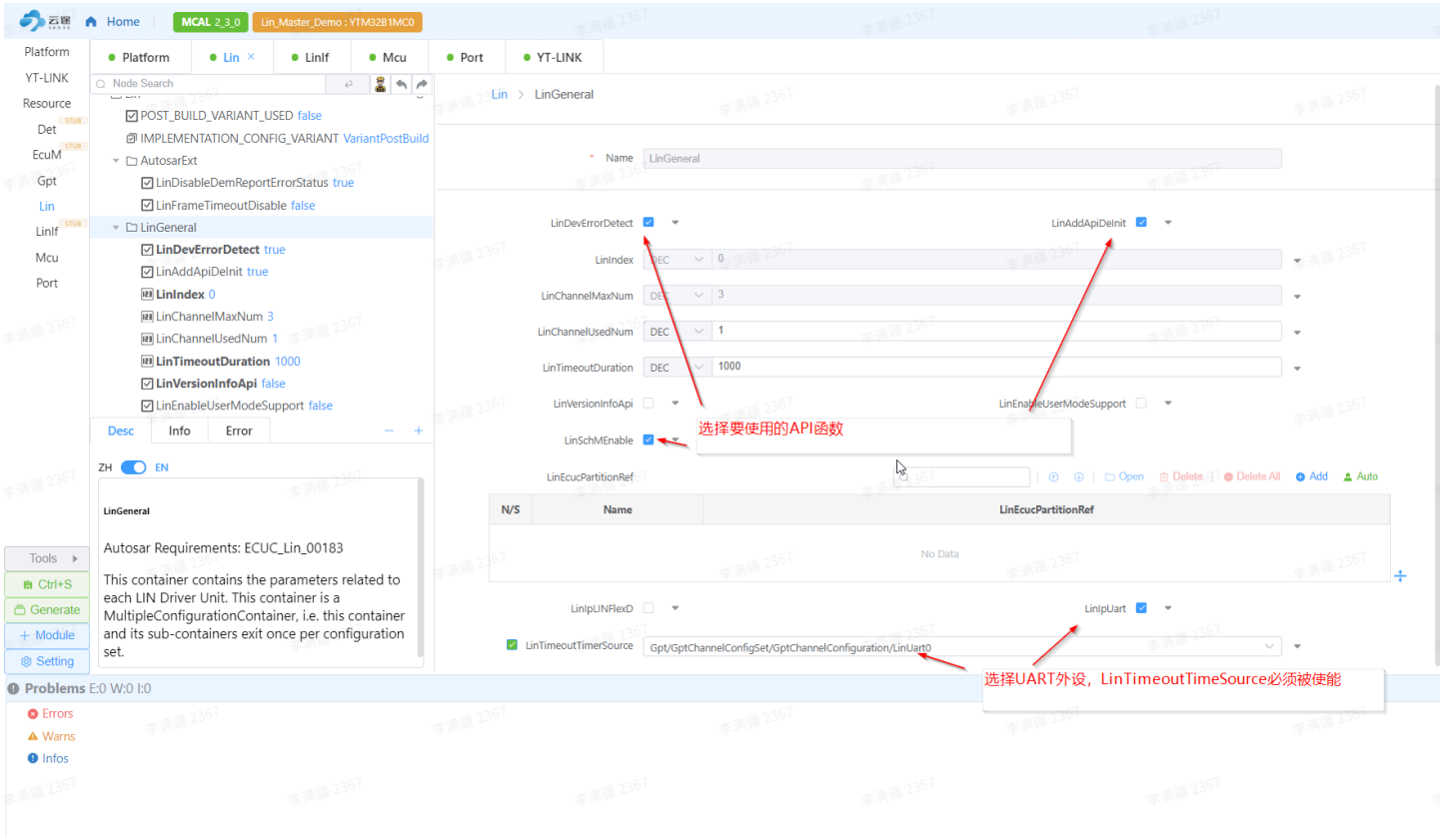


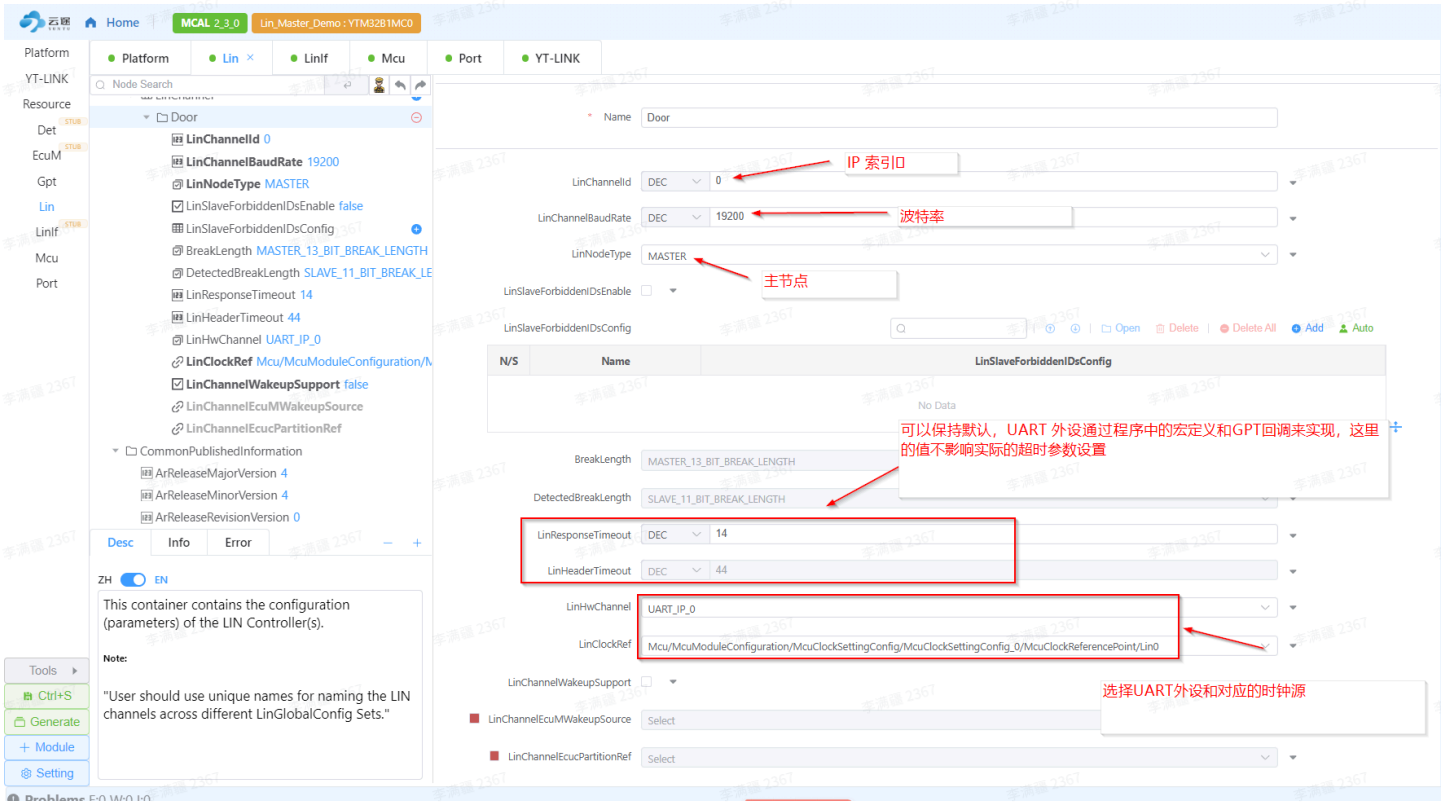
4.2.3 Port配置

这里配置了一路LIN



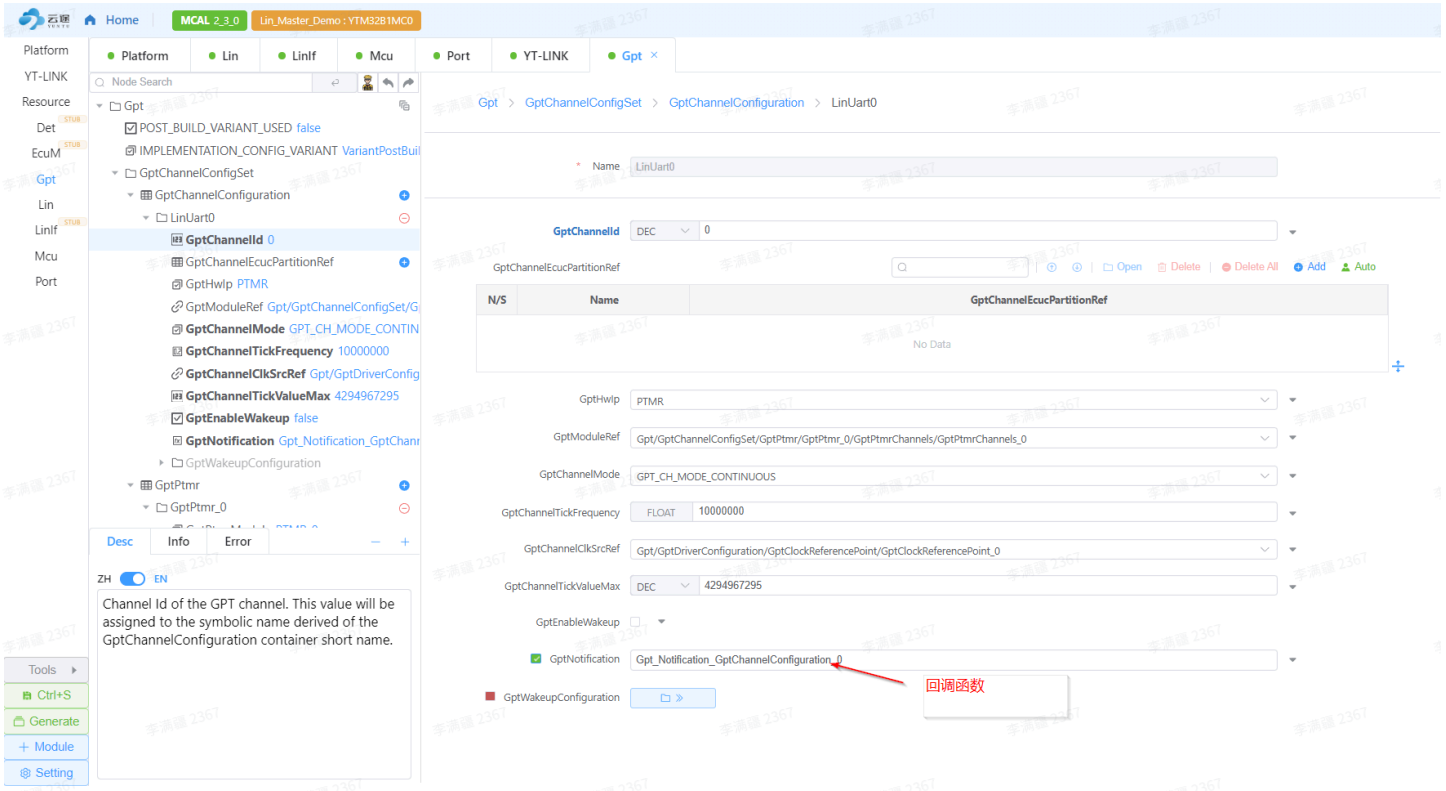
4.2.4 配置LIN模块





4.2.5 配置GPT模块

GPT模块的具体配置可以参考GPT模块的配置说明，这里仅做引用。



4.3 API说明

```
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
/* USER CODE BEGIN PV */
uint8 *lin_RxData; /* Pointer to the received data buffer */
uint8 lin_RxDataBuffer[8]; /* Buffer for received data */

uint8 lin_TxDataMatrix[10][8] =
{
    [0]={[0]=0x00, [1]=0x01, [2]=0x02, [3]=0x03, [4]=0x04, [5]=0x05, [6]=0x06, [7]=0x07},
    [1]={[0]=0x08, [1]=0x09, [2]=0x0A, [3]=0x0B, [4]=0x0C, [5]=0x0D, [6]=0x0E, [7]=0x0F},
    [2]={[0]=0x10, [1]=0x11, [2]=0x12, [3]=0x13, [4]=0x14, [5]=0x15, [6]=0x16, [7]=0x17},
    [3]={[0]=0x18, [1]=0x19, [2]=0x1A, [3]=0x1B, [4]=0x1C, [5]=0x1D, [6]=0x1E, [7]=0x1F},
    [4]={[0]=0x20, [1]=0x21, [2]=0x22, [3]=0x23, [4]=0x24, [5]=0x25, [6]=0x26, [7]=0x27},
    [5]={[0]=0x28, [1]=0x29, [2]=0x2A, [3]=0x2B, [4]=0x2C, [5]=0x2D, [6]=0x2E, [7]=0x2F},
    [6]={[0]=0x30, [1]=0x31, [2]=0x32, [3]=0x33, [4]=0x34, [5]=0x35, [6]=0x36, [7]=0x37},
    [7]={[0]=0x38, [1]=0x39, [2]=0x3A, [3]=0x3B, [4]=0x3C, [5]=0x3D, [6]=0x3E, [7]=0x3F},
    [8]={[0]=0x40, [1]=0x41, [2]=0x42, [3]=0x43, [4]=0x44, [5]=0x45, [6]=0x46, [7]=0x47},
    [9]={[0]=0x48, [1]=0x49, [2]=0x4A, [3]=0x4B, [4]=0x4C, [5]=0x4D, [6]=0x4E, [7]=0x4F}
};
```

Pdu 矩阵

```c
const Lin_PduType linTxPduMatrix[10] =
{
    [0]={
        .Pid = 0x1F,
        .Cs = LIN_ENHANCED_CS,
        .Drc = LIN_FRAMERESPONSE_TX,
        .Dl = 8U,
        .SduPtr = lin_TxDataMatrix[0],
    },
    [1]={
        .Pid = 0x20,
        .Cs = LIN_ENHANCED_CS,
        .Drc = LIN_FRAMERESPONSE_TX,
        .Dl = 8U,
        .SduPtr = lin_TxDataMatrix[1],
    },
    [2]={
        .Pid = 0x61,
        .Cs = LIN_ENHANCED_CS,
        .Drc = LIN_FRAMERESPONSE_TX,
        .Dl = 8U,
        .SduPtr = lin_TxDataMatrix[2],
    },
    [3]={
        .Pid = 0xE2,
        .Cs = LIN_ENHANCED_CS,
        .Drc = LIN_FRAMERESPONSE_TX,
        .Dl = 8U,
        .SduPtr = lin_TxDataMatrix[3],
    },
    [4]={
        .Pid = 0xA3,
        .Cs = LIN_ENHANCED_CS,
        .Drc = LIN_FRAMERESPONSE_TX,
        .Dl = 8U,
        .SduPtr = lin_TxDataMatrix[4],
    },
    [5]={
        .Pid = 0x64,
        .Cs = LIN_ENHANCED_CS,
```

发送矩阵

```c
/* Lin Rx Pdu matrix, Slave response, Master rece
const Lin_PduType linRxPduMatrix[10] =
{
    [0]={
        .Pid = 0xE9,
        .Cs = LIN_ENHANCED_CS,
        .Drc = LIN_FRAMERESPONSE_RX,
        .Dl = 8U,
        .SduPtr = NULL_PTR,
    },
    [1]={
        .Pid = 0x6A,
        .Cs = LIN_ENHANCED_CS,
        .Drc = LIN_FRAMERESPONSE_RX,
        .Dl = 8U,
        .SduPtr = NULL_PTR,
    },
    [2]={
        .Pid = 0x2B,
        .Cs = LIN_ENHANCED_CS,
        .Drc = LIN_FRAMERESPONSE_RX,
        .Dl = 8U,
        .SduPtr = NULL_PTR,
    },
    [3]={
        .Pid = 0xEC,
        .Cs = LIN_ENHANCED_CS,
        .Drc = LIN_FRAMERESPONSE_RX,
        .Dl = 8U,
        .SduPtr = NULL_PTR,
    },
    [4]={
        .Pid = 0xAD,
        .Cs = LIN_ENHANCED_CS,
        .Drc = LIN_FRAMERESPONSE_RX,
        .Dl = 8U,
        .SduPtr = NULL_PTR,
    },
    [5]={
        .Pid = 0x2E,
        .Cs = LIN_ENHANCED_CS,
```

接收矩阵

```c
static void Board_Init(void)
{
    Gpt_Init(ConfigPtr: &Gpt_Config);
    Lin_Init(Config: &Lin_Config);
    Platform_Init(Config: NULL_PTR);
    Port_Init(ConfigPtr: &Port_Config);
}

/* USER CODE BEGIN 4 */
/* USER CODE END 4 */
```

LIN初始化

Lin_Master_Demo > Mcal > Lin > src > C Lin.c > Lin_Init

```
31  extern "C" {
574 {
591     {
608         }
609         else
610         {
611             Lin_LocCfgPtr = Config;
612         }
613 #endif /* LIN_FIXED_PB_CONFIG == STD_ON */
614     }
615 #if (LIN_FRAME_TIMEOUT_DISABLE == STD_OFF)  /* timeout update*/
616 #if (LIN_IP_UART == STD_ON)
617     if((Std_ReturnType)E_OK == RetVal)
618     {
619         Gpt_StopTimer(Channel: Lin_LocCfgPtr->LinTimerSourceId);
620         Gpt_EnableNotification(Channel: Lin_LocCfgPtr->LinTimerSourceId);
621         Gpt_StartTimer(Channel: Lin_LocCfgPtr->LinTimerSourceId, Value: Lin_LocCfgPtr->LinTimerClockRef / LIN_CONST_GPT_500US_DIVISOR);
622     }
623 #endif
624 #endif
625     if((Std_ReturnType)E_OK == RetVal)
626     {
627         Lin_Mld_SetInstMap(LinInstMapPtr: Lin_LocCfgPtr->LinInstMapPtr);
628         for (uint8 Index = 0U; Index < (uint8)LIN_CHANNEL_USED_NUM; ++Index)
629         {
630             for (uint8 LinDataLengthLoop = 0U; LinDataLengthLoop < LIN_CONST_MAX_FRAME_BYTES_NUM; ++LinDataLengthLoop)
631             {
632                 Lin_LocSduBuffAddrs[Index][LinDataLengthLoop] = (uint8)0U;
633             }
634             /* Hardware driver init */
635             Lin_Mld_InitChannel(Channel: Index, LinCfgPtr: Lin_LocCfgPtr);
636             /* Set LIN channel state to default LIN_DRV_CH_SLEEP */
637             Lin_ChStates[Index] = LIN_DRV_CH_SLEEP;
638             /* Map configured hardware channels to logical channels */
639         }
640         /* Update LIN status */
641         Lin_LocDrvState = LIN_INIT;
642     }
643 }
644
645 #if (LIN_DEINIT_API_SUPPORT == STD_ON)
646 /**
647  * @brief           Initializes the LIN module.
```

LIN 初始化时，通过GPT启动超时处理，因此GPT 的初始化需要在LIN 初始化之前

```
229 int main(void)
230 {
231     /* USER CODE BEGIN 1 */
232     Mcu_Init(&Mcu_Config);
233     Mcu_InitClock(0);
234     /* USER CODE END 1 */
235     Board_Init();
236     /* USER CODE BEGIN 2 */
237     Lin_WakeupInternal(LinConf_LinChannel_Door);
238     /* USER CODE END 2 */
239
```

唤醒本地master主节点

```c
    while (1)
    {
        Cnt++;
        if (Cnt >= 10000)
        {
            Cnt = 0;
            LinStatus = Lin_GetStatus(Channel: 0U, Lin_SduPtr: &lin_RxData); /* Get the status of LIN*/
            if ((LinStatus == LIN_OPERATIONAL) || (LinStatus == LIN_TX_OK) || (LinStatus == LIN_RX_NO_RESPONSE) || (LinStatus == LIN_RX_OK))
            {
                /*LIN has received the data.*/
                if (LinStatus == LIN_RX_OK)
                {
                    for (int Ptr = 0; Ptr < 8; Ptr++)
                    {
                        lin_RxDataBuffer[Ptr] = *(lin_RxData + Ptr);
                    }
                }
                /*RxPdu and TxPdu are transmitted alternately.*/
                if (SendDataType == 0)
                {
                    SendDataType = 1;
                    Lin_SendFrame(Channel: LinConf_LinChannel_Door, PduInfoPtr: linTxPduMatrix + DataLoop0);
                    DataLoop0++;
                    if (DataLoop0 >= 10)
                    {
                        DataLoop0 = 0;
                    }
                }
                else
                {
                    SendDataType = 0;
                    Lin_SendFrame(Channel: LinConf_LinChannel_Door, PduInfoPtr: linRxPduMatrix + DataLoop1);
                    DataLoop1++;
                    if (DataLoop1 >= 10)
                    {
                        DataLoop1 = 0;
                    }
                }
            }
        }
        /* USER CODE END WHILE */
        /* USER CODE BEGIN 3 */
```

获取LIN 状态信息

该传输函数只能主节点调用，主节点根据传输类型发送数据帧

Lin_Master_Demo › Mcal › Lin › src › C Lin.c › Lin_SendFrame

```c
extern "C" {
{
    {
                                Apiid: (uint8)LIN_SID_SEND_FRAME,Errorid: LIN_E_STATE_TRANSITION);
#endif /* LIN_DEV_ERROR_DETECT == STD_ON */
    }
    else
    {
        /* LIN master device send header first */
        if (E_NOT_OK == Lin_Mld_SendHeader(Channel, PduInfoPtr))
        {
            /* Report production error */
#if (LIN_DEV_ERROR_DETECT == STD_ON)
            (void) Det_ReportError(ModuleId: (uint16)LIN_MODULE_ID,InstanceId: Channel,
                                   ApiId: (uint8)LIN_SID_SEND_FRAME,ErrorId: LIN_E_TIMEOUT);
#endif /* LIN_DEV_ERROR_DETECT == STD_ON */
#if (LIN_DEM_DISABLE == STD_OFF)
            (void) Dem_SetEventStatus(Lin_LocCfgPtr->LinDemConfigPtr->EventId, DEM_EVENT_STATUS_FAILED);
#endif
        }
        else
        {
            /* Header sent successfully */
            /* Set header status to TRUE if master need to send response frame */
            if (LIN_FRAMERESPONSE_TX == PduInfoPtr->Drc)
            {
                Lin_Mld_SendResponse(Channel, PduInfoPtr);
                RetVal = (uint8)E_OK;
            }
            /* Note: need to call Lin_SendHeader() first */
            RetVal = (uint8)E_OK;
        }
    }
    return RetVal;
}

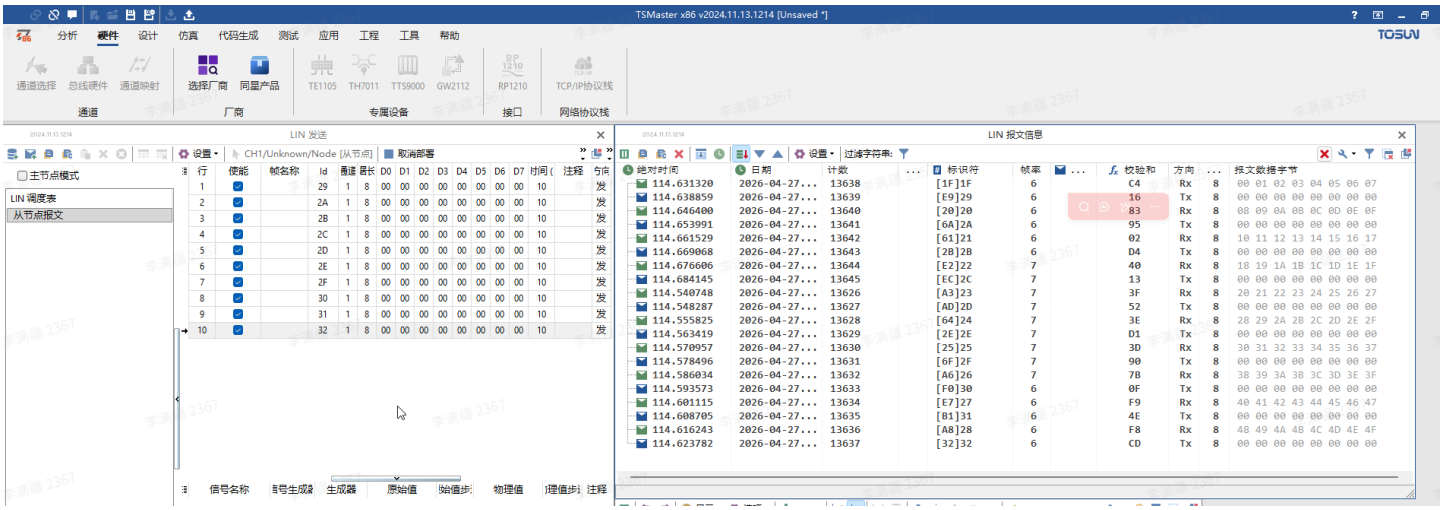
/**
```

发送帧头

发送数据帧

4.4 现象

TS Master 作为从节点，MC03 EVB 作为主节点，测试现象如下：



Tips:

程序中使用的为的PID帧，使用同星上位机设置从机发送帧时需要将PID&3F解析出帧ID进行设置。例如 PID=0xE9 对应 ID=0x29。

5. LIN模块应用实例3(UART LIN从机)

5.1 应用场景

- 基于MC0 Demo板，MC03 使用UART 外设来实现LIN slave功能，上位机作为主节点。
- 配置波特率为19200
- 上位机发送帧头。ID=0x1F～0x28 时，上位机发布响应，MC03 接收；ID=0x29～0x32 时，MC03 发布响应，上位机接收。

5.2 配置步骤

5.2.1 时钟配置

参考 4.2.1 的 UART 时钟配置；本章从机生成配置使用 24 MHz 时钟引用，需与实际时钟树一致。

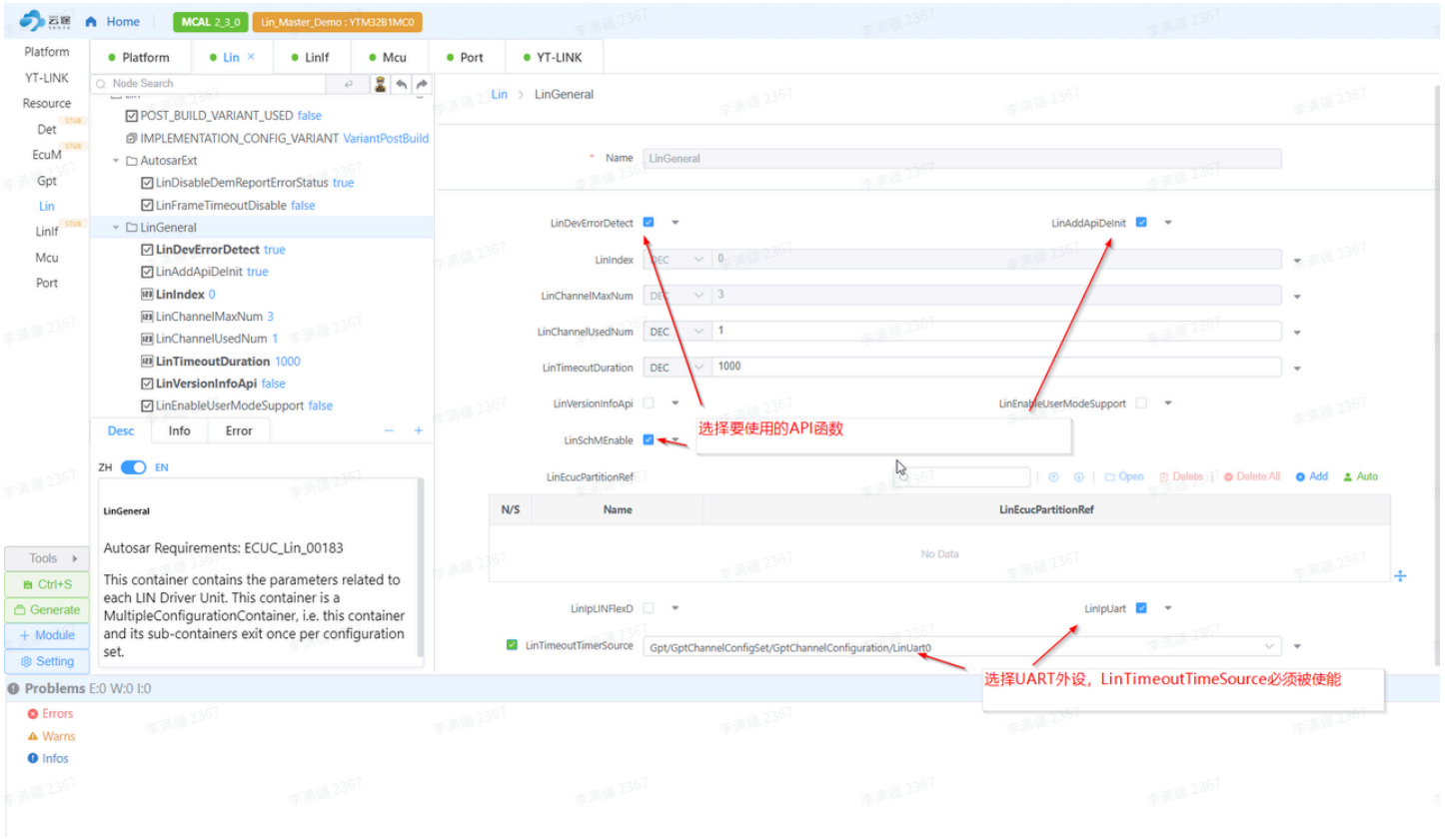
5.2.2 中断配置

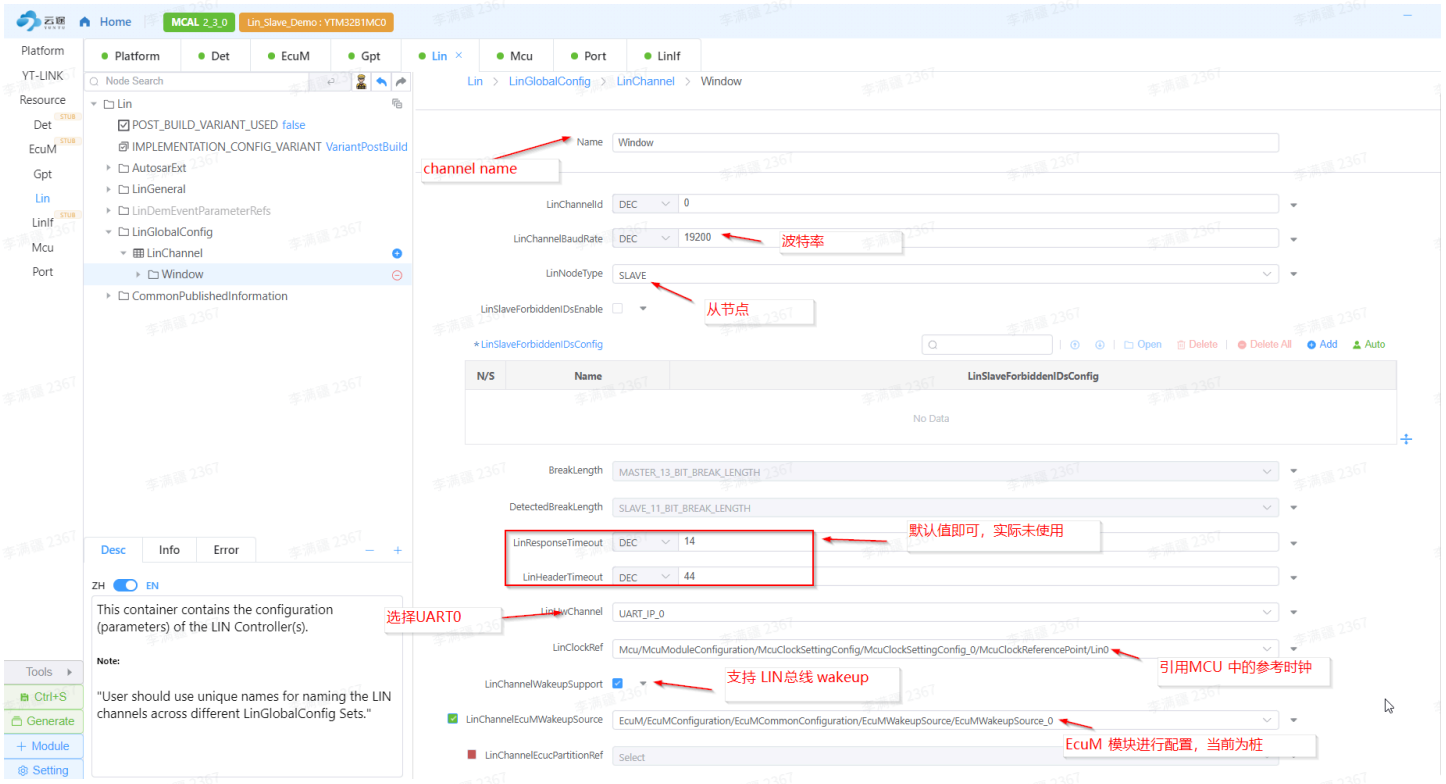
参考 4.2.2 配置对应 UART 中断；中断入口及通道映射按 Lin_Slave_Demo 核对。

5.2.3 Port配置

参考 4.2.3 配置 UART TX/RX 复用及收发器引脚；按从机连接核对收发器工作模式。

5.2.4 配置LIN模块


选择要使用的API函数
选择UART外设，LinTimeoutTimeSource必须被使能


channel name
波特率
从节点
默认值即可，实际未使用
选择UART0
支持 LIN总线 wakeup
引用MCU 中的参考时钟
EcuM 模块进行配置，当前为桩

5.3 API说明

```c
int main(void)
{
    /* USER CODE BEGIN 1 */
    Mcu_Init(&Mcu_Config);
    Mcu_InitClock(0);
    /* USER CODE END 1 */
    Board_Init();
    /* USER CODE BEGIN 2 */
    Lin_WakeupInternal(LinConf_LinChannel_Window);
    /* USER CODE END 2 */

    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1)
    {
#if (LIN_SLAVE_API_SUPPORT == STD_ON)
        if (TRUE == TransmitDoneFlag)
        {
            TransmitDoneFlag = FALSE;
        }
        if (TRUE == ReceiveDoneFlag)
        {
            ReceiveDoneFlag = FALSE;
        }
        /* USER CODE END WHILE */
        /* USER CODE BEGIN 3 */
#endif
    }
    /* USER CODE END 3 */
}

static void Board_Init(void)
{
    Gpt_Init(&Gpt_Config);
    Lin_Init(&Lin_Config);
    Platform_Init(NULL_PTR);
    Port_Init(&Port_Config);
}

/* USER CODE BEGIN 4 */
/* USER CODE END 4 */
```

唤醒本地从节点

复位在LinIf 回调函数中置位的发送和接受完成标志

初始化

```c
Std_ReturnType LinIf_HeaderIndication(NetworkHandleType Channel, Lin_PduType *PduPtr)
{
    Std_ReturnType RetVal = E_NOT_OK;
    if (LinConf_LinChannel_Window == Channel)
    {
        if ((0x29U <= (PduPtr->Pid & 0x3fU)) && (0x32U >= (PduPtr->Pid & 0x3fU)))
        {
            PduPtr->Cs = LIN_ENHANCED_CS;
            PduPtr->Dl = 8U;
            PduPtr->Drc = LIN_FRAMERESPONSE_TX;
            SduPtr[0] = PduPtr->Pid;
            PduPtr->SduPtr = SduPtr;
            RetVal = E_OK;
        }
        else if ((0x1fU <= (PduPtr->Pid & 0x3fU)) && (0x28U >= (PduPtr->Pid & 0x3fU)))
        {
            PduPtr->Cs = LIN_ENHANCED_CS;
            PduPtr->Drc = LIN_FRAMERESPONSE_RX;
            RetVal = E_OK;
        }
        else
        {
            PduPtr->Drc = LIN_FRAMERESPONSE_IGNORE;
            RetVal = E_OK;
        }
    }
    return RetVal;
}

void LinIf_RxIndication(NetworkHandleType Channel, uint8 *Lin_SduPtr)
{
    ReceiveDoneFlag = TRUE;
    for (uint8 Index = 0; Index < 8; Index++)
    {
        ReceiveData[Index] = *(Lin_SduPtr + Index);
    }
}

void LinIf_TxConfirmation(NetworkHandleType Channel)
{
    TransmitDoneFlag = TRUE;
}

/* USER CODE END 0 */
```

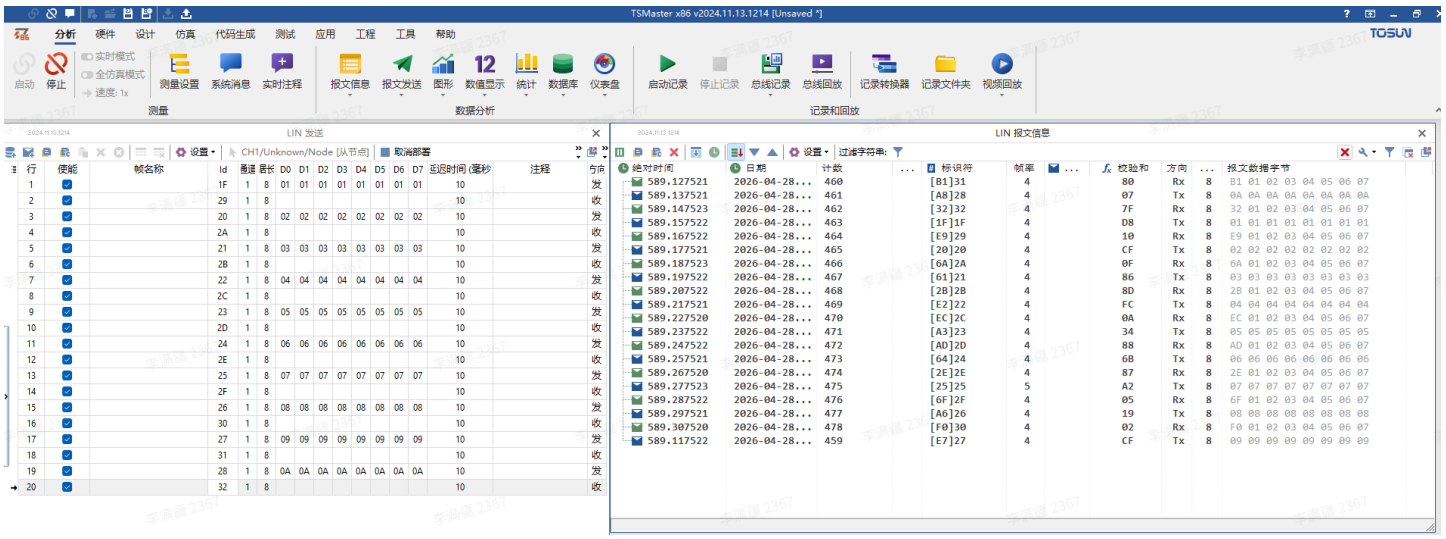
根据PID解析出ID,再根据ID判断接受还是发送，再分别进行处理

把接收的内容写入buffer,以及写接收完成标志位

发送完成后写发送完成标志位

5.4 现象

使用 TS Master 作为主节点：对 ID=0x1F～0x28 发送帧头和数据，对 ID=0x29～0x32 发送帧头并接收 MC03 从节点的响应。现象如下：



文档历史

版本号	日期	修订记录
V1.0	2026.4.28	初始版本