YTMicro MD24 MCAL应用 - Lin模块配置及实例

版本

Config Tool Version：3.2.0

MD24 MCAL version：3.0.0

前言

1. 本文介绍 LIN 模块原理、驱动配置与 API，以及 MD24 UART LIN 主机和从机配置实例。
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。

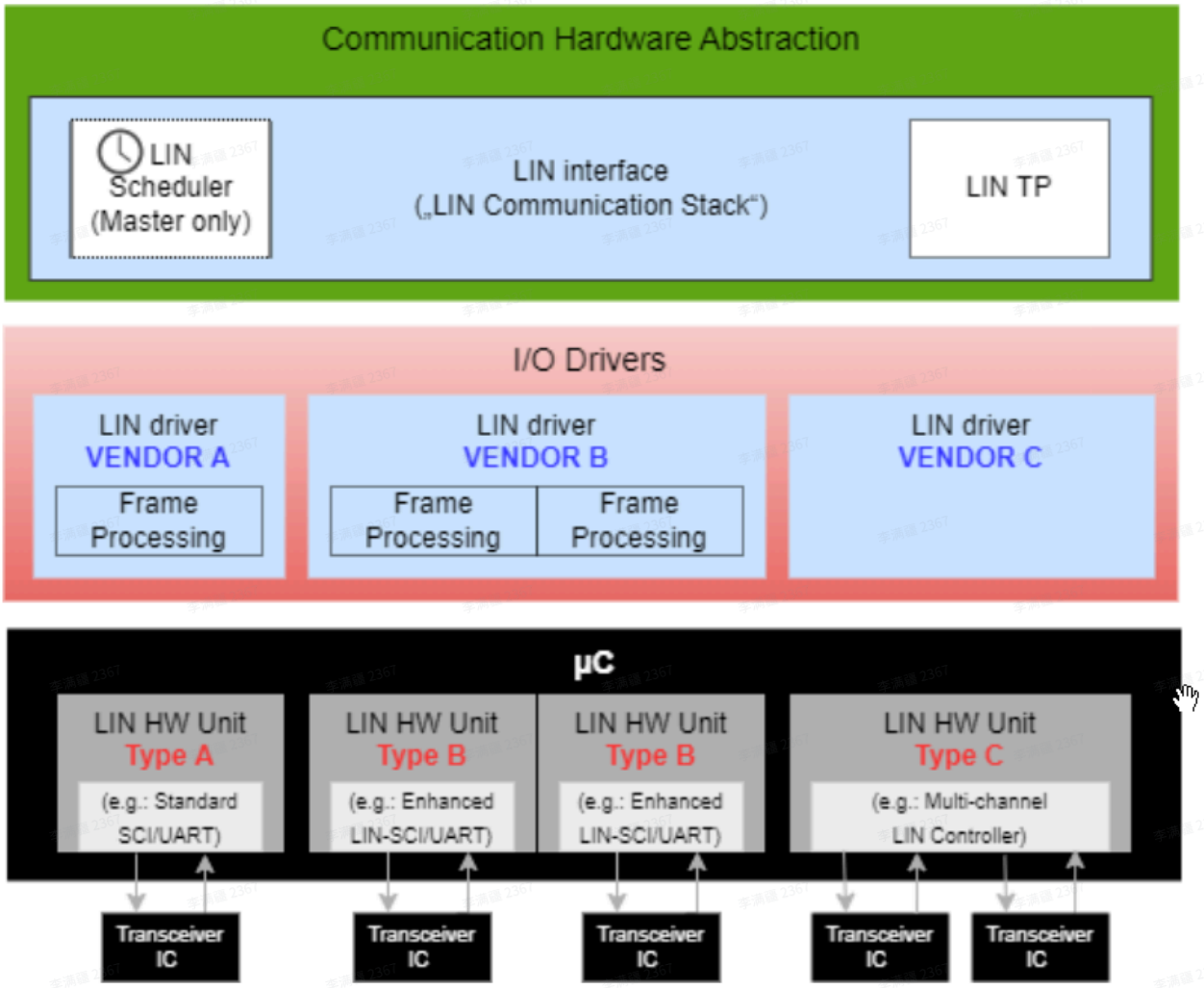


Figure 1.1: Overview LIN Software Architecture Layering

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。

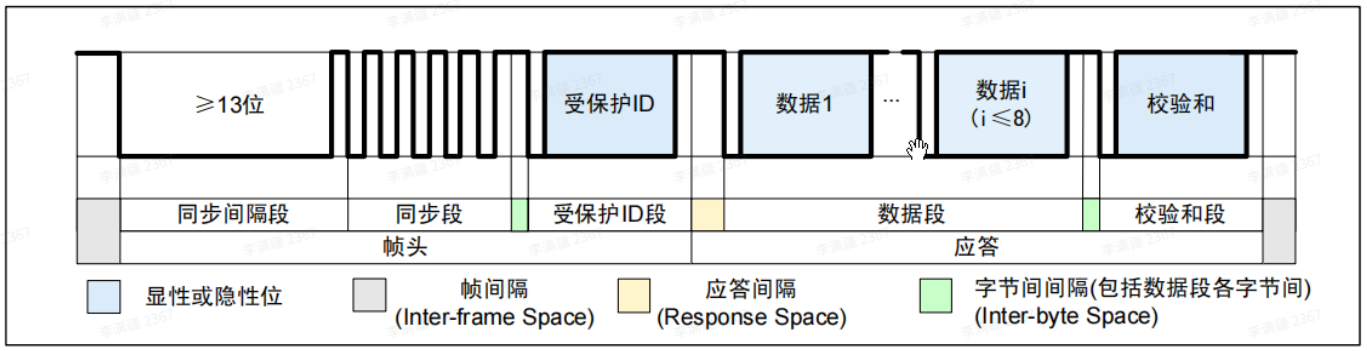


图 3.2 帧的结构

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。

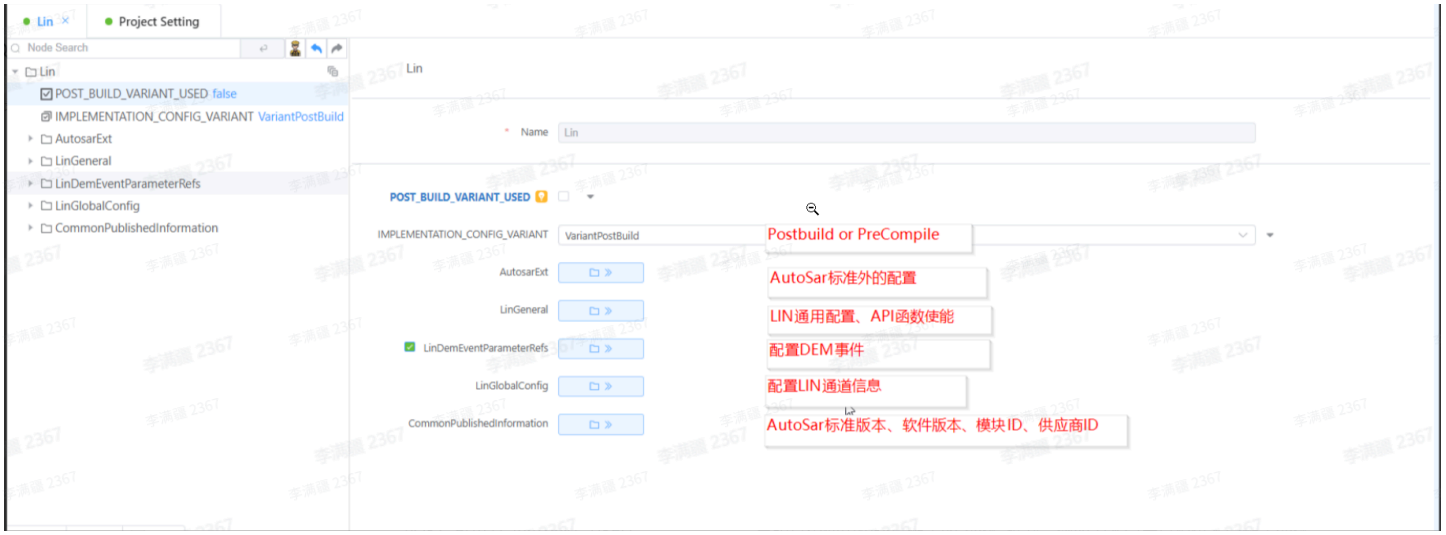
Data数据段、Checksum校验和段：响应包含 1～8 字节数据和 1 字节校验和。

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

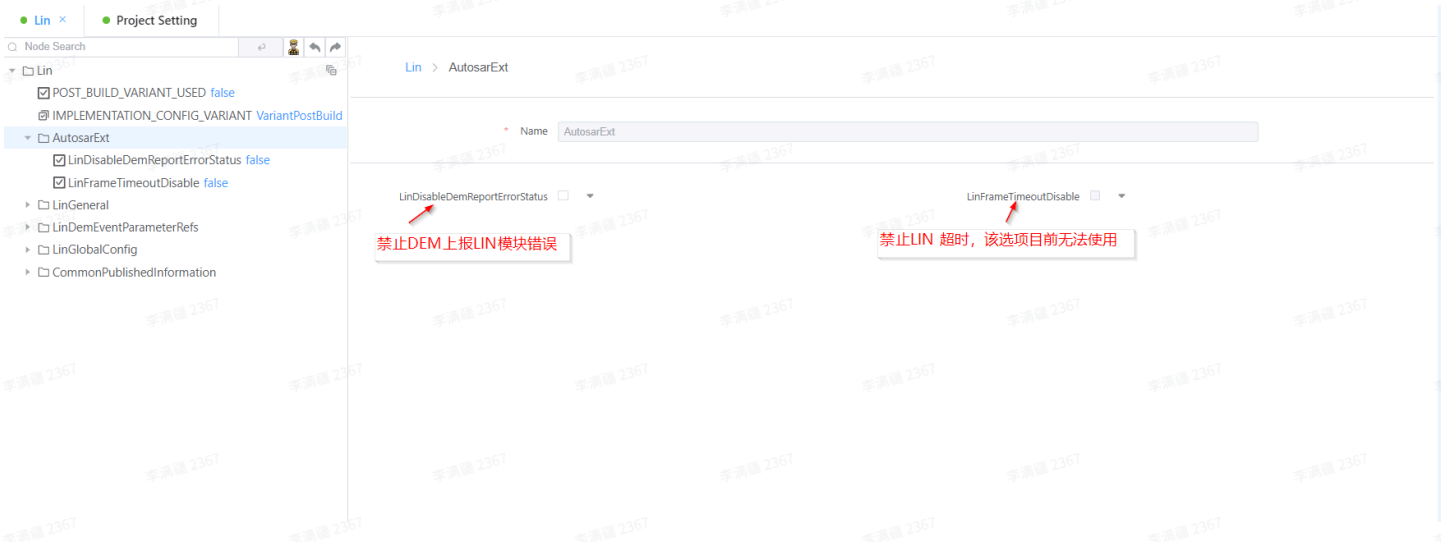


选项	含义
POST_BUILD_VARIANT_USED	声明是否使用构建后配置的“变体”机制，例如为不同产品提供不同配置变体
IMPLEMENTATION_CONFIG_VARIANT	选择模块采用哪种配置实现方式，如编译前配置、构建后配置

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：

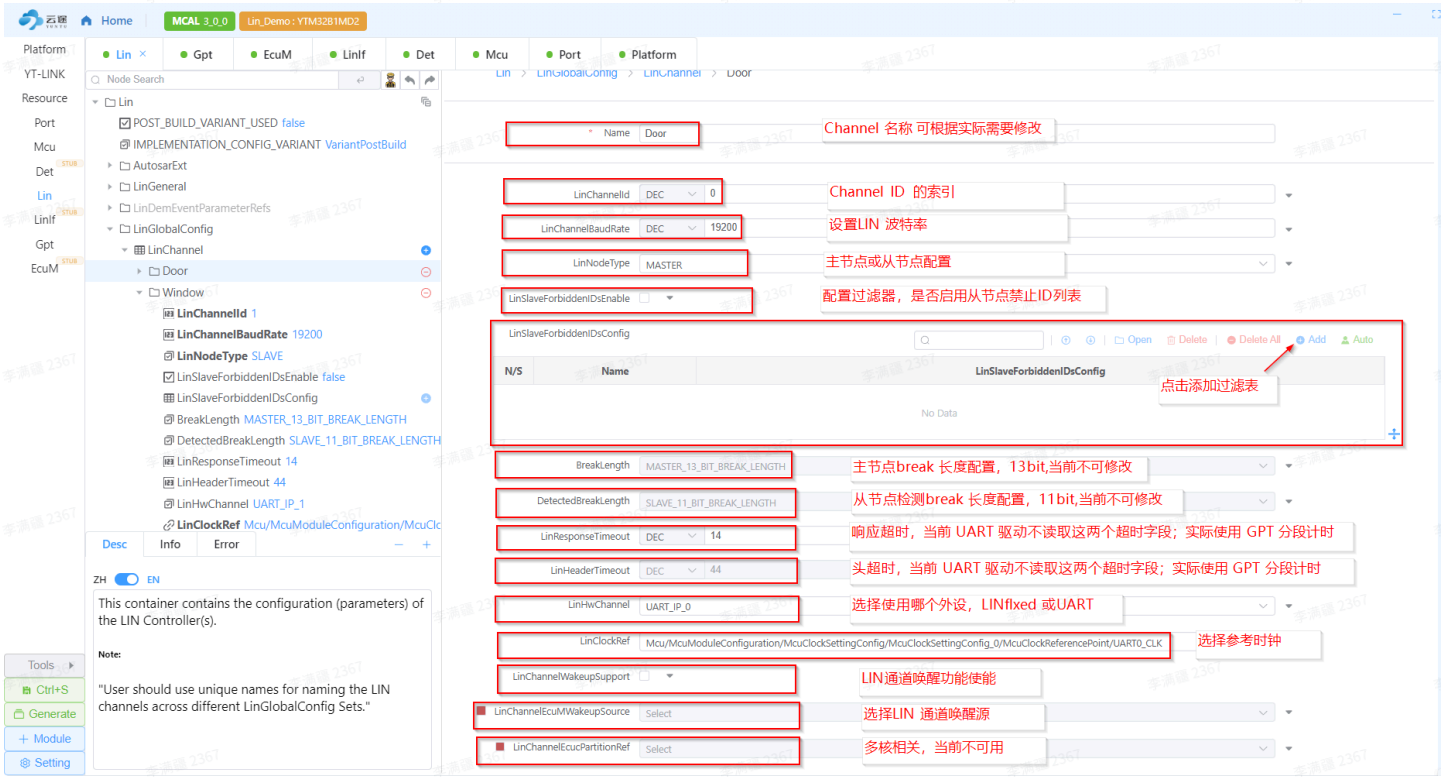
开启 `LIN_ENABLE_USER_MODE_SUPPORT` 后，会通过 `OsIf_Trusted_Call...` 包装底层操作；关闭时则直接调用。

- 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。
- linuseSoftwareChecksum: 仅限硬件为Linflxed 外设时可以启用。

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，需要根据芯片型号选择
- 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:

- 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	启用通道外部总线唤醒支持

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：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，例如 MD24 当前 UART 驱动以 **RX 活动边沿中断**进入唤醒处理。

- 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 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。

2.3.3 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（UART LIN主机+从机）

3.1 应用场景

- 参考工程为 MCAL_3.0.0/Lin_Demo，目标器件为 YTM32B1MD24，使用开发板 YTM32B1MD2-EVB-Q100-Rev.B。

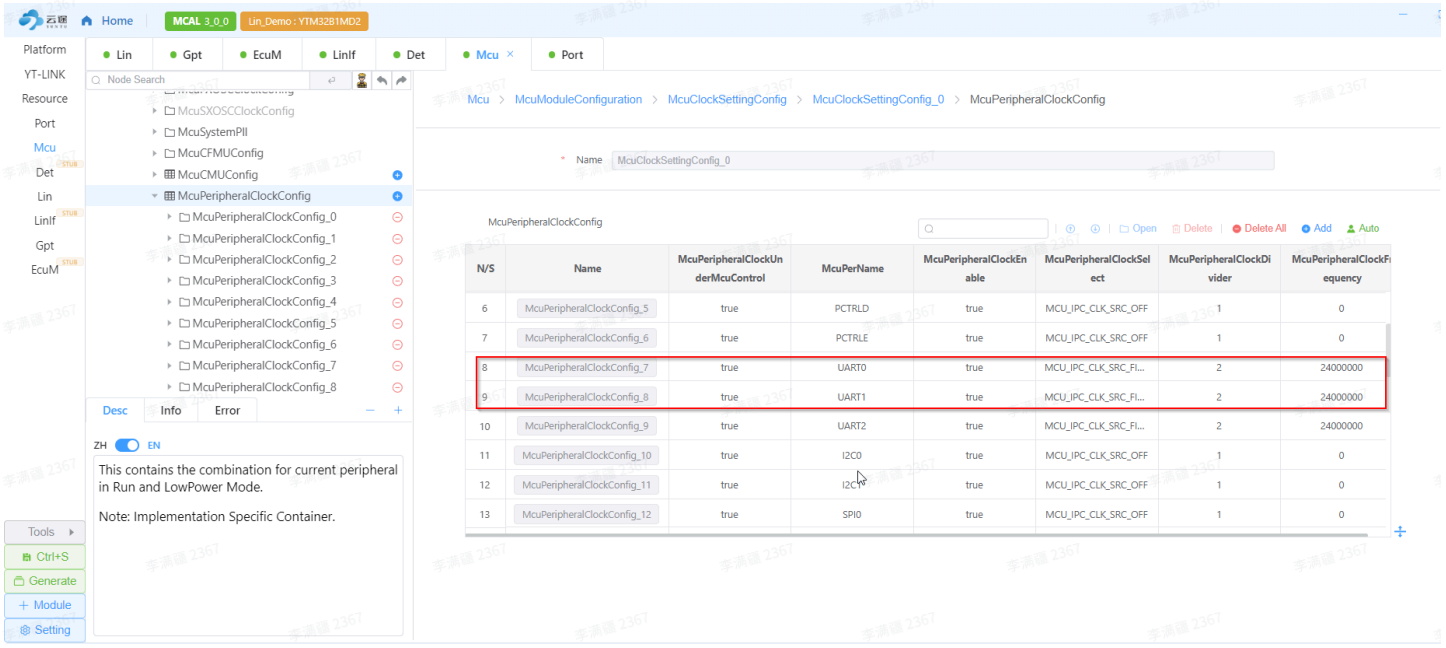
逻辑通道	角色 / 外设	UART 引脚及复用	EVB 总线接口
Door（0）	主节点 / UART0	PTB0：RX；PTB1：TX 均为 ALT2、无内部上下拉	J23
Window（1）	从节点 / UART1	PTC8：RX；PTC9：TX 均为 ALT2、无内部上下拉	J27

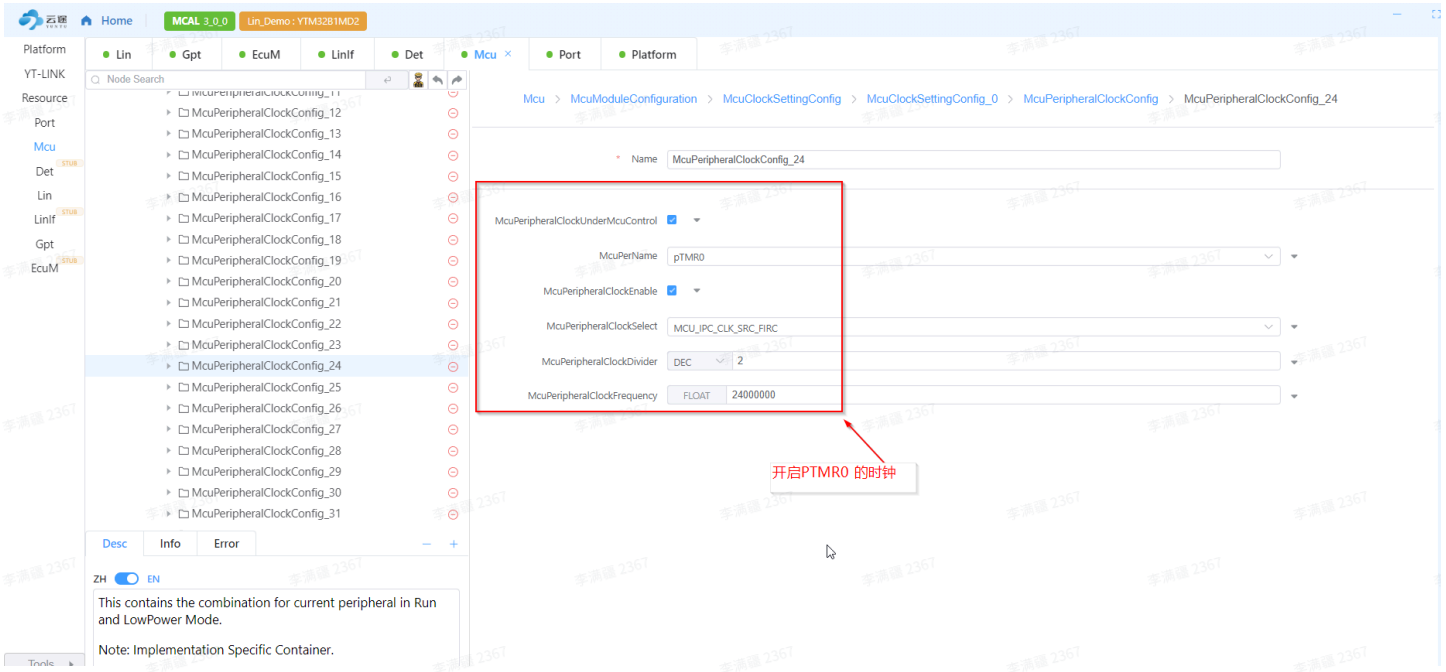
- 配置波特率为19200
- 调用Lin_SendFrame发送了ID从0x1F到0x28的LIN_MASTER_RESPONSE类型的帧; 发送ID从0x29到0x32的LIN_SLAVE_RESPONSE类型的帧，其中id为0x32的帧slave节点不响应。

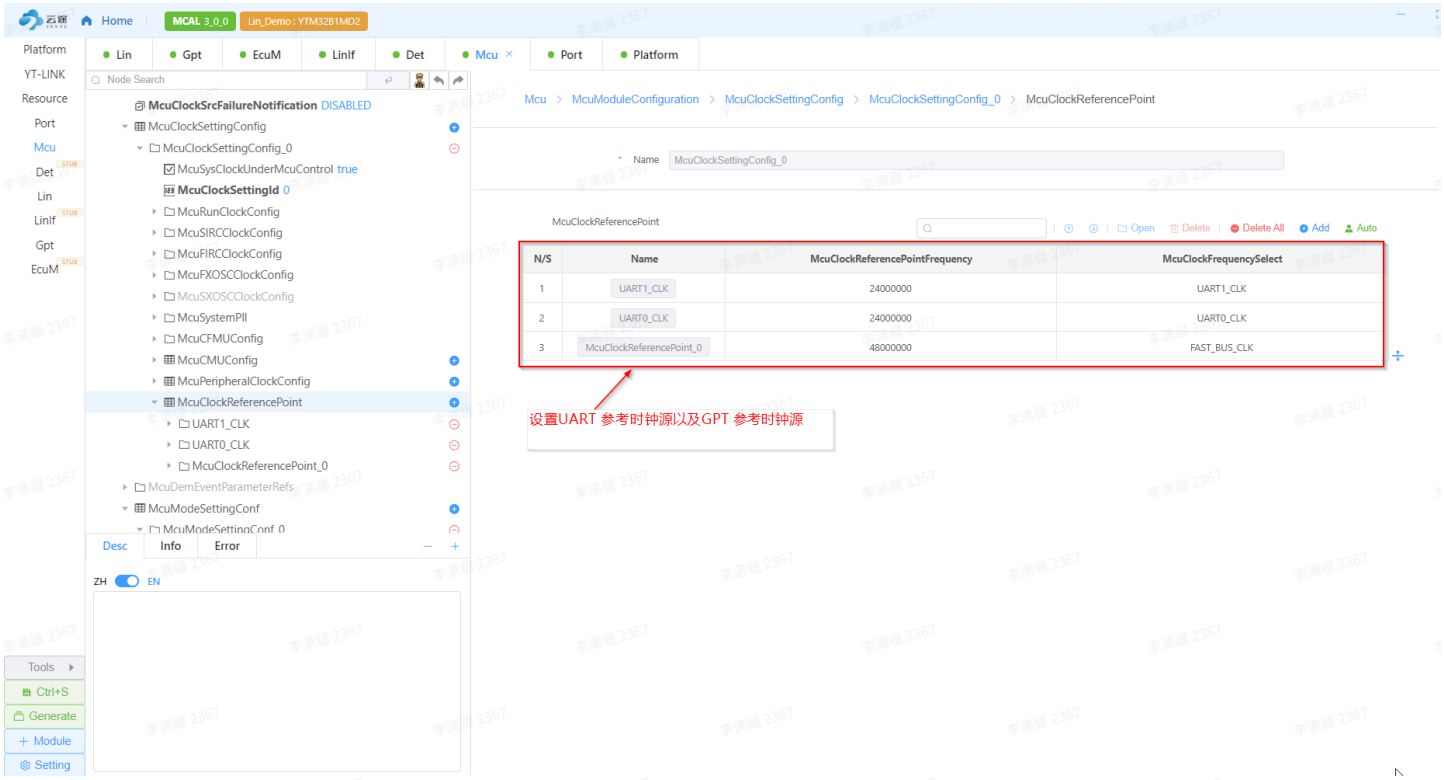
3.2 配置步骤

3.2.1 时钟配置

在MCU模块中配置LIN的reference clock。

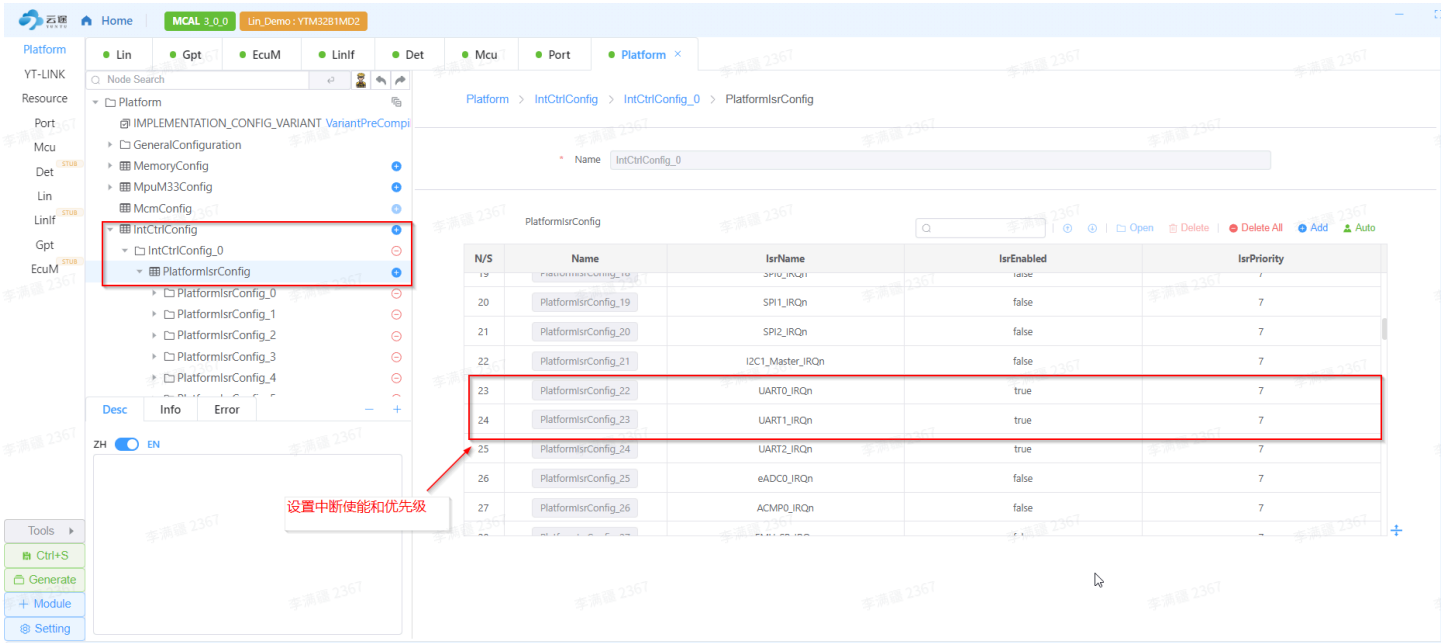


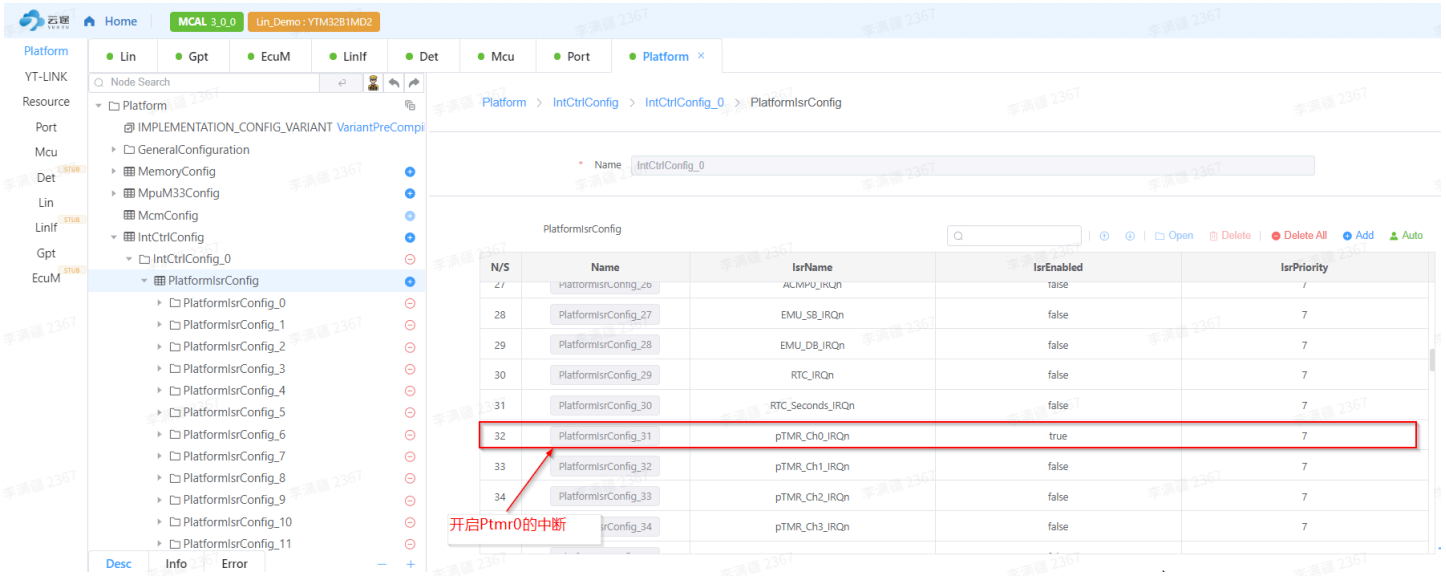




3.2.2 中断配置

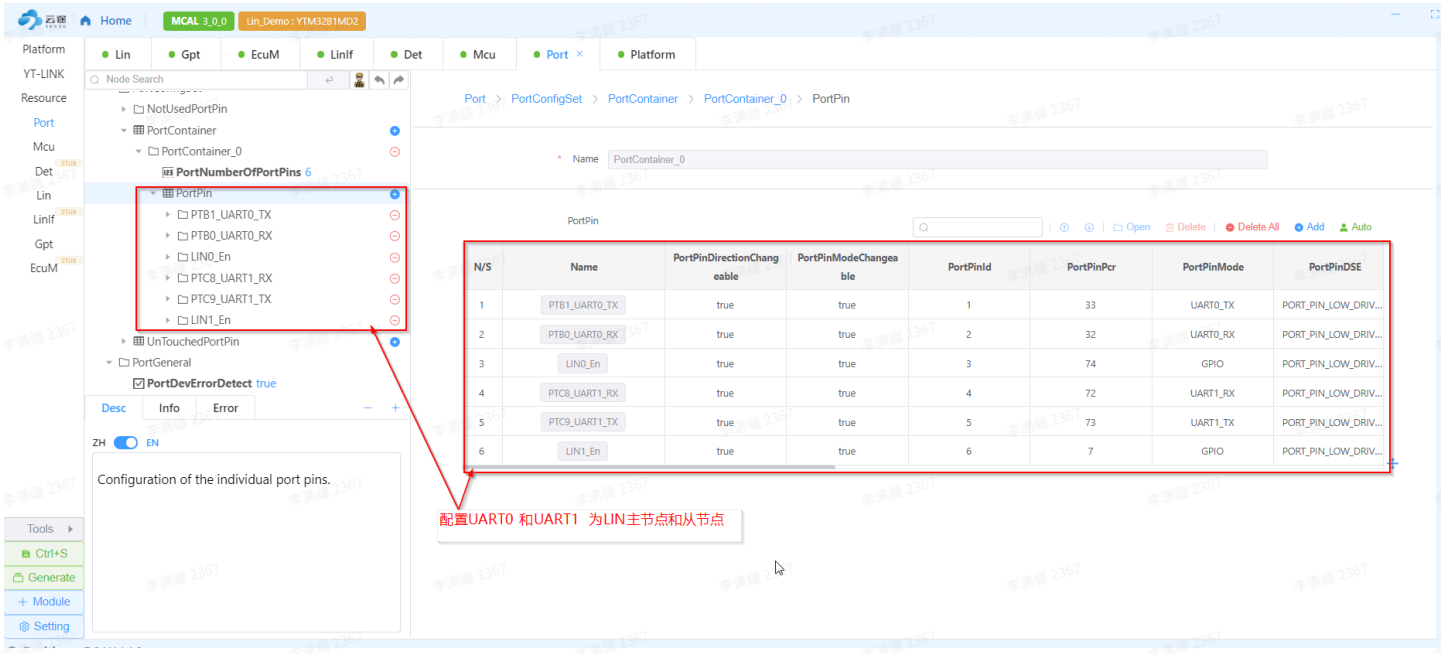
根据 UART 驱动的收发方式配置所需中断，在 Platform 中设置对应中断源和处理入口；



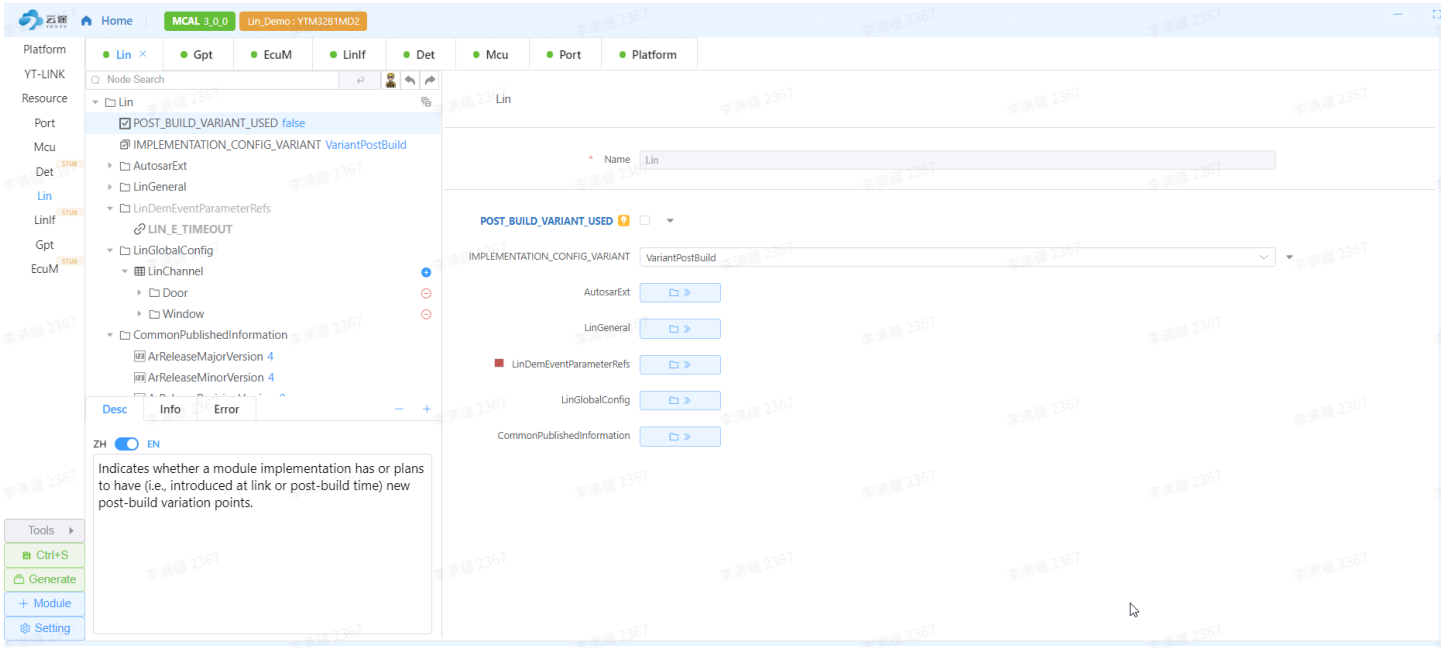


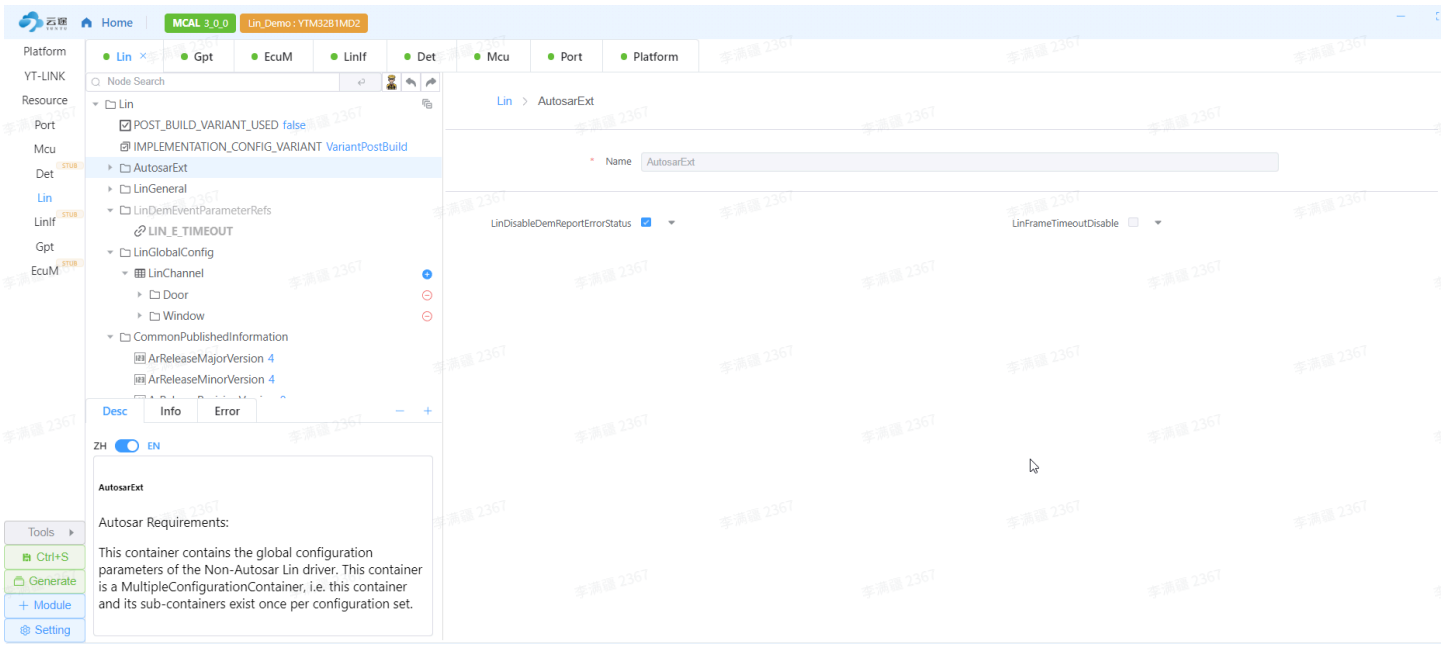
3.2.3 Port配置

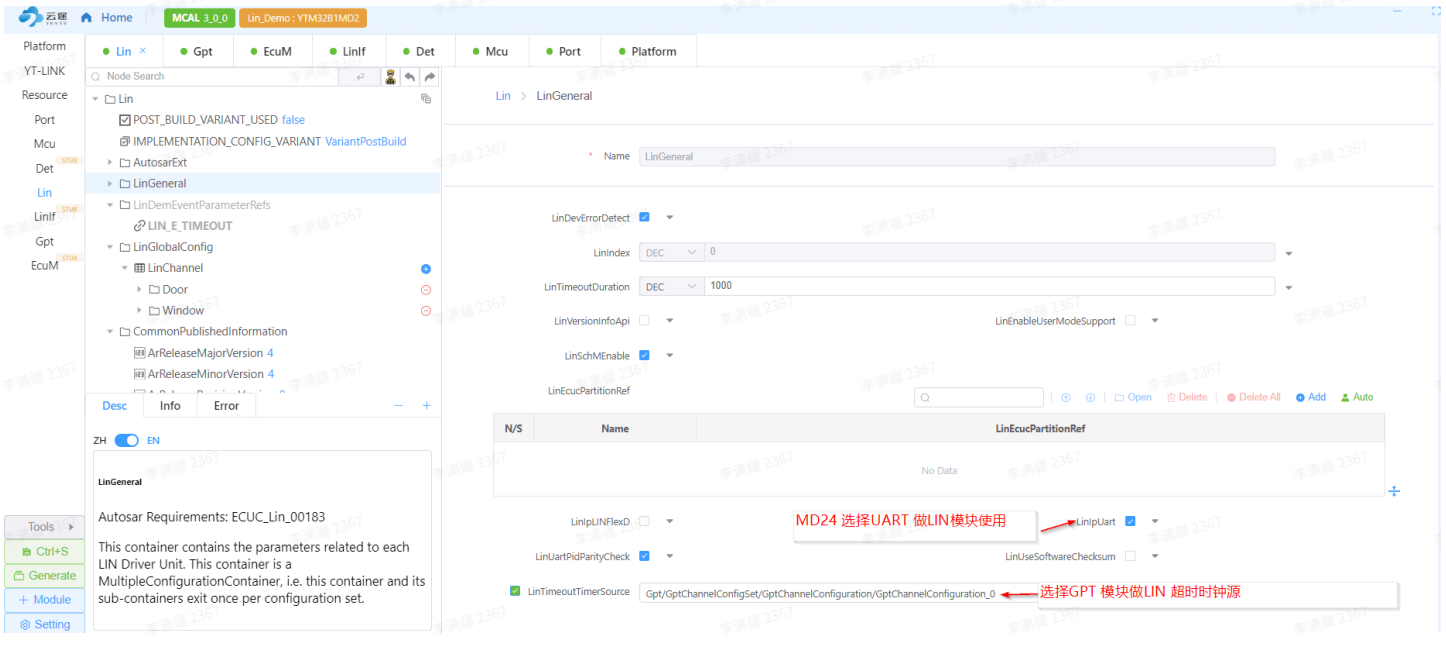
配置两路 LIN 的 TX/RX 复用及收发器控制引脚，见下图。

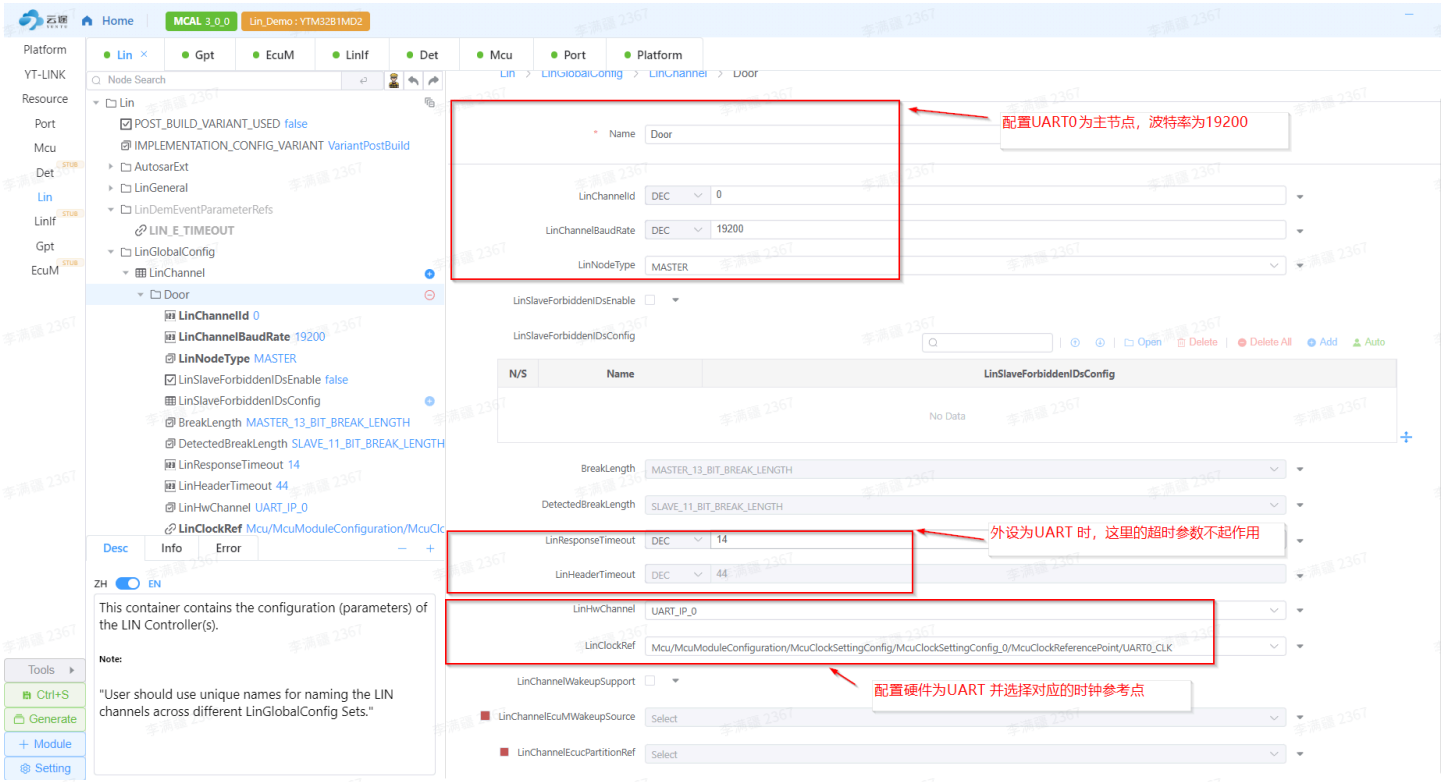


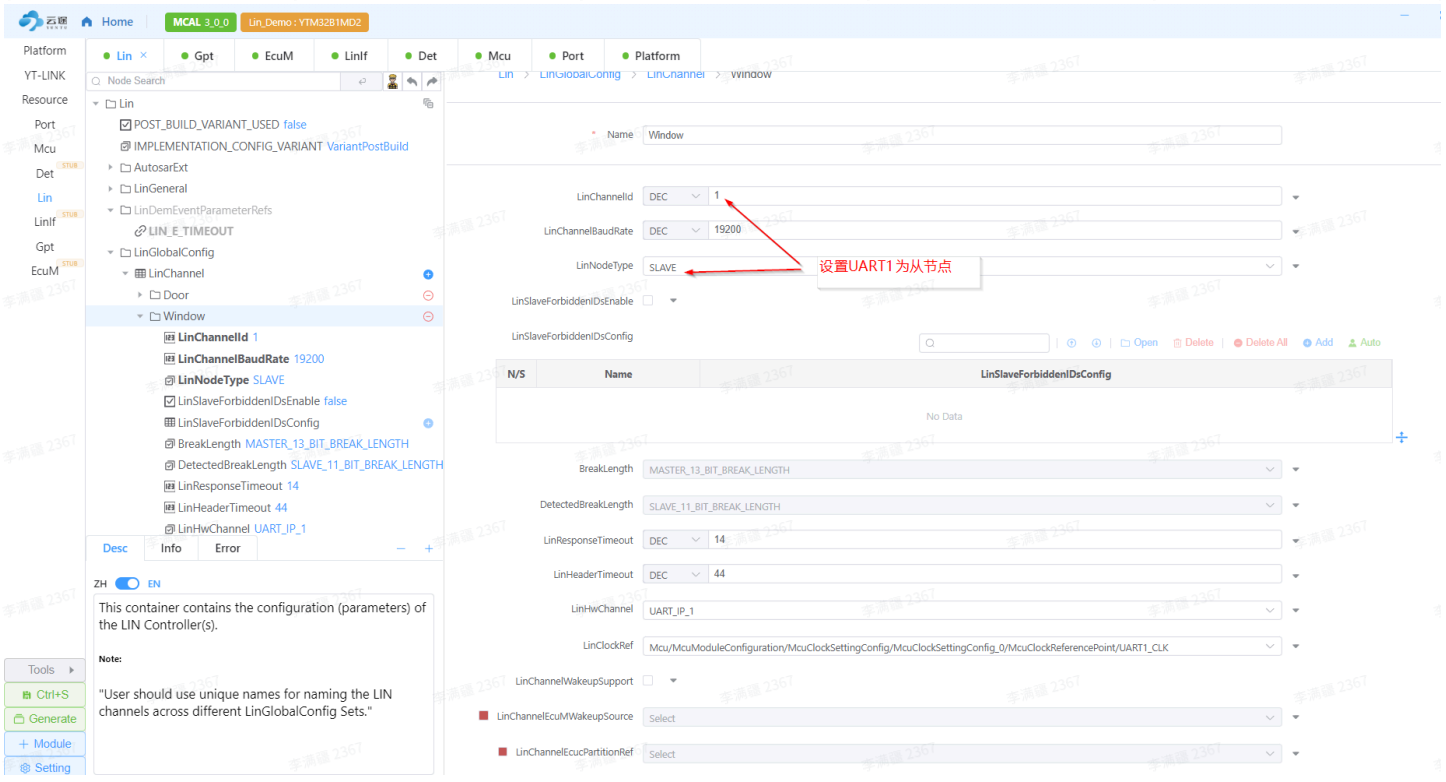
3.2.4 配置LIN模块


Home
MCAL 3_0_0
Lin_Demo : YTM32B1MD2
Platform
YT-LINK
Resource
Port
Mcu
Det
Lin
LinIf
Gpt
EcuM
Lin
Gpt
EcuM
LinIf
Det
Mcu
Port
Platform
Node Search
Lin
POST_BUILD_VARIANT_USED false
IMPLEMENTATION_CONFIG_VARIANT VariantPostBuild
AutosarExt
LinGeneral
LinDemEventParameterRefs
LIN_E_TIMEOUT
LinGlobalConfig
LinChannel
Door
Window
CommonPublishedInformation
ArReleaseMajorVersion 4
ArReleaseMinorVersion 4
Desc
Info
Error
ZH
EN
Indicates whether a module implementation has or plans to have (i.e., introduced at link or post-build time) new post-build variation points.
Tools
Ctrl+S
Generate
Module
Setting
Lin
Name
Lin
POST_BUILD_VARIANT_USED
IMPLEMENTATION_CONFIG_VARIANT
VariantPostBuild
AutosarExt
LinGeneral
LinDemEventParameterRefs
LinGlobalConfig
CommonPublishedInformation

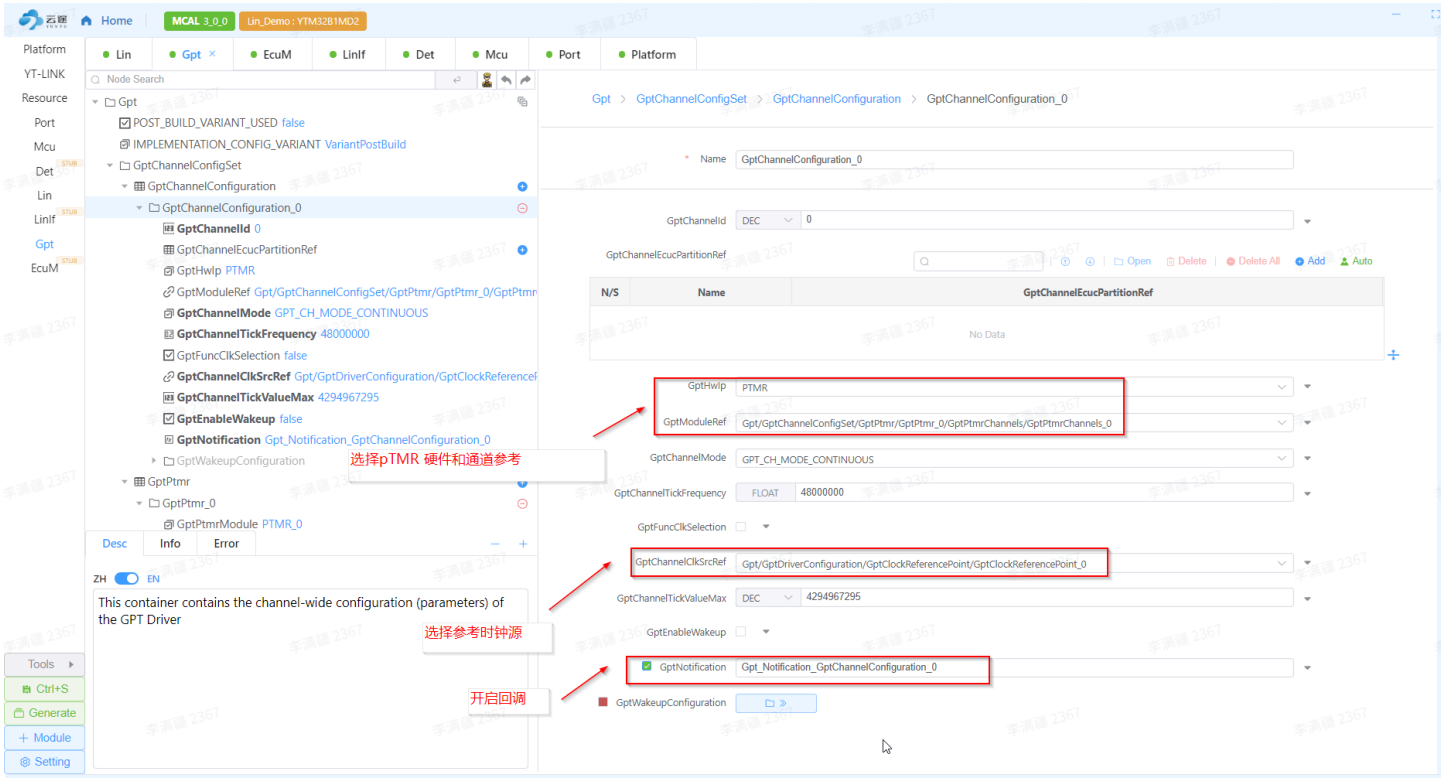

Home
MCAL 3_0_0
Lin_Demo : YTM32B1MD2
Lin
Gpt
EcuM
LinIf
Det
Mcu
Port
Platform
Node Search
Lin
POST_BUILD_VARIANT_USED false
IMPLEMENTATION_CONFIG_VARIANT VariantPostBuild
AutosarExt
LinGeneral
LinDemEventParameterRefs
LIN_E_TIMEOUT
LinGlobalConfig
LinChannel
Door
Window
CommonPublishedInformation
ArReleaseMajorVersion 4
ArReleaseMinorVersion 4
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.
Tools
Ctrl+S
Generate
Module
Setting
Lin > AutosarExt
Name
AutosarExt
LinDisableDemReportErrorStatus
LinFrameTimeoutDisable

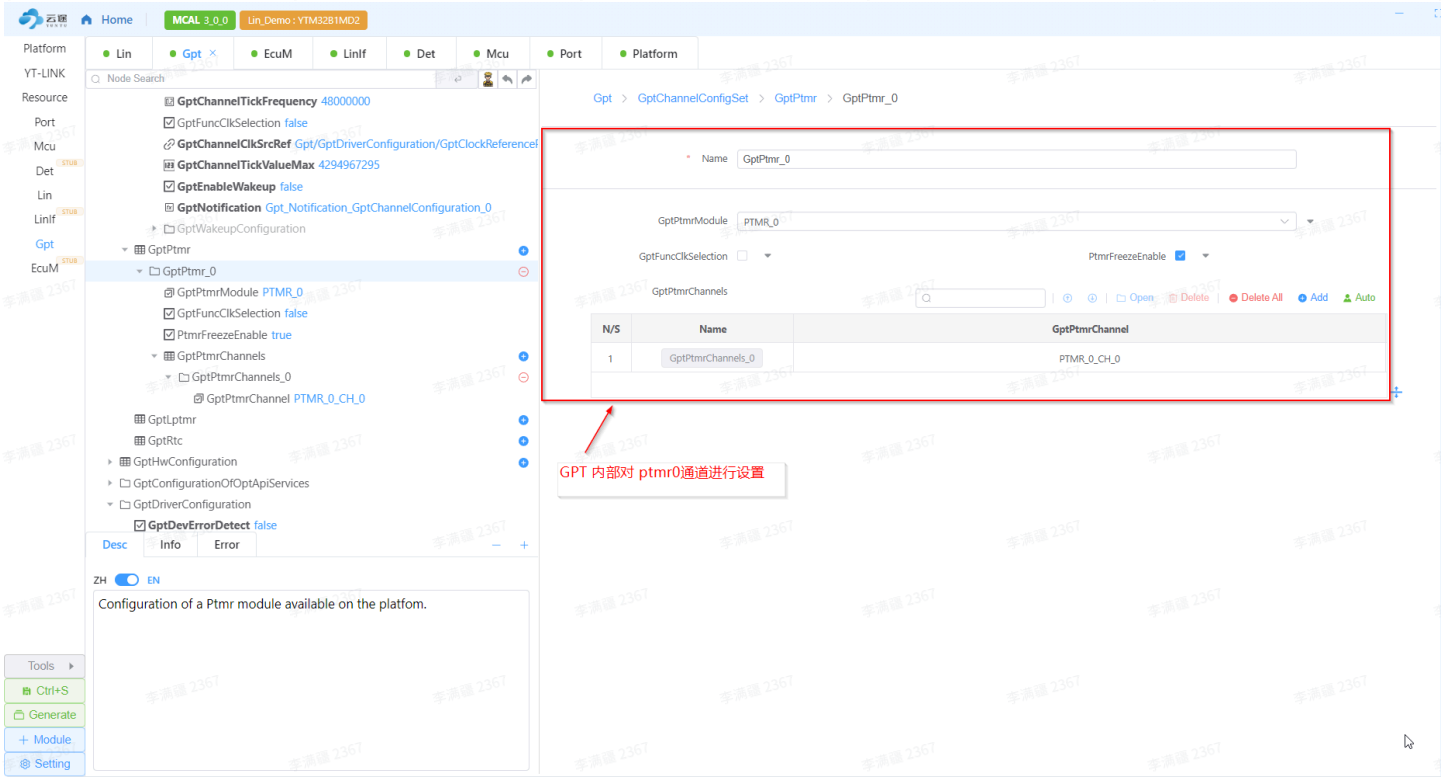

Home
MCAL 3_0_0
Lin_Demo : YTM32B1MD2
Lin
Gpt
EcuM
LinIf
Det
Mcu
Port
Platform
Node Search
Lin
POST_BUILD_VARIANT_USED false
IMPLEMENTATION_CONFIG_VARIANT VariantPostBuild
AutosarExt
LinGeneral
LinDemEventParameterRefs
LIN_E_TIMEOUT
LinGlobalConfig
LinChannel
Door
Window
CommonPublishedInformation
ArReleaseMajorVersion 4
ArReleaseMinorVersion 4
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.
Tools
Ctrl+S
Generate
Module
Setting
Lin > LinGeneral
Name
LinGeneral
LinDevErrorDetect
LinIndex
DEC
0
LinTimeoutDuration
DEC
1000
LinVersionInfoApi
LinEnableUserModeSupport
LinSchMEnable
LinEcucPartitionRef
Open
Delete
Delete All
Add
Auto
N/S
Name
LinEcucPartitionRef
No Data
LinIpLINFlexD
MD24 选择UART 做LIN模块使用
LinIpUart
LinUartPidParityCheck
LinUseSoftwareChecksum
LinTimeoutTimerSource
Gpt/GptChannelConfigSet/GptChannelConfiguration/GptChannelConfiguration_0
选择GPT 模块做LIN 超时时钟源

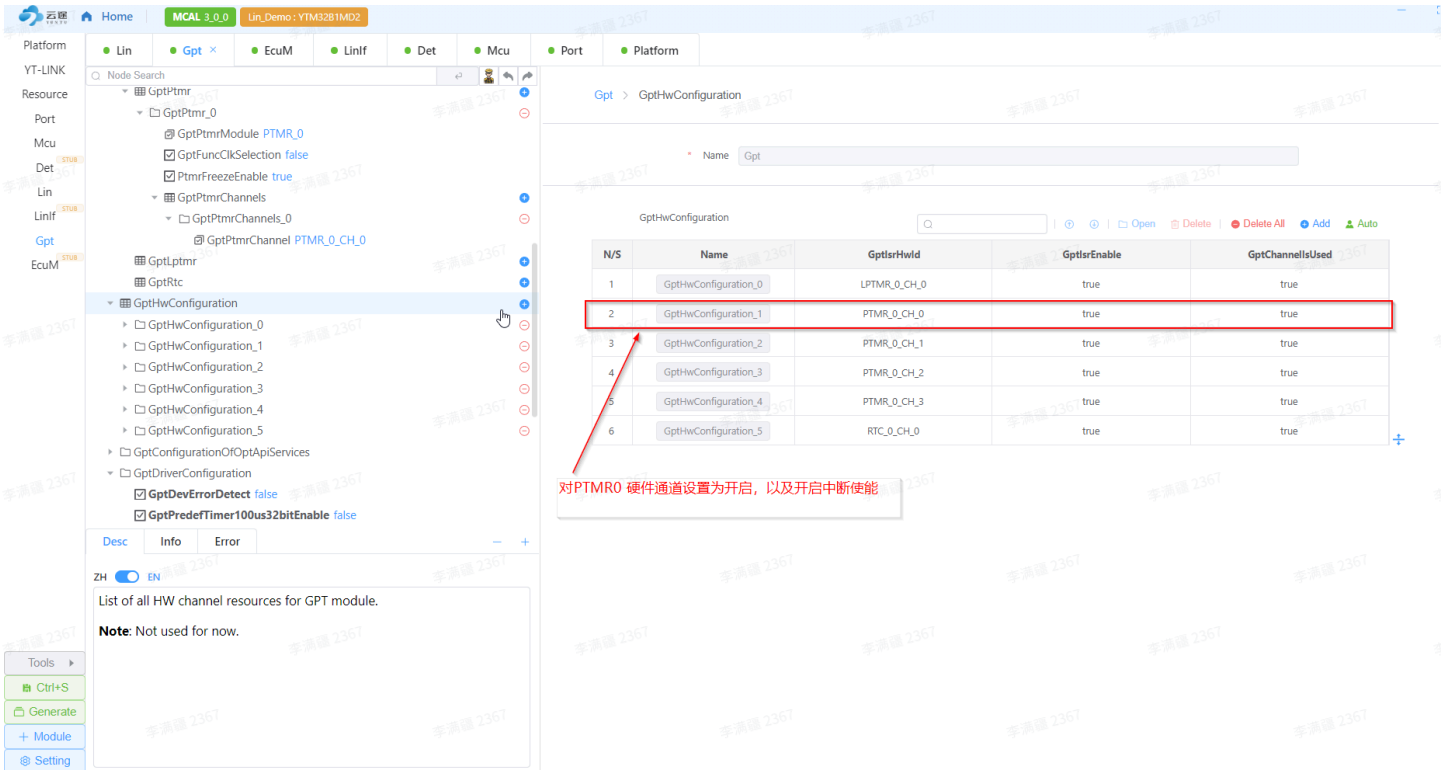

Home
MCAL 3_0_0
Lin_Demo : YTM32B1MD2
Platform
YT-LINK
Resource
Port
Mcu
Det
Lin
LinIf
Gpt
EcuM
Lin
Gpt
EcuM
LinIf
Det
Mcu
Port
Platform
Node Search
Lin
POST_BUILD_VARIANT_USED false
IMPLEMENTATION_CONFIG_VARIANT VariantPostBuild
AutosarExt
LinGeneral
LinDemEventParameterRefs
LIN_E_TIMEOUT
LinGlobalConfig
LinChannel
Door
LinChannelId 0
LinChannelBaudRate 19200
LinNodeType MASTER
LinSlaveForbiddenIDsEnable false
LinSlaveForbiddenIDsConfig
BreakLength MASTER_13_BIT_BREAK_LENGTH
DetectedBreakLength SLAVE_11_BIT_BREAK_LENGTH
LinResponseTimeout 14
LinHeaderTimeout 44
LinHwChannel UART_IP_0
LinClockRef Mcu/McuModuleConfiguration/McuClo
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."
Tools
Ctrl+S
Generate
Module
Setting
Lin > LinGlobalConfig > LinChannel > Door
Name Door
配置UART0为主节点，波特率为19200
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
LinResponseTimeout DEC 14
外设为UART 时，这里的超时参数不起作用
LinHeaderTimeout DEC 44
LinHwChannel UART_IP_0
LinClockRef Mcu/McuModuleConfiguration/McuClockSettingConfig/McuClockSettingConfig_0/McuClockReferencePoint/UART0_CLK
配置硬件为UART 并选择对应的时钟参考点
LinChannelWakeupSupport
LinChannelEcuMWakeupSource Select
LinChannelEcucPartitionRef Select

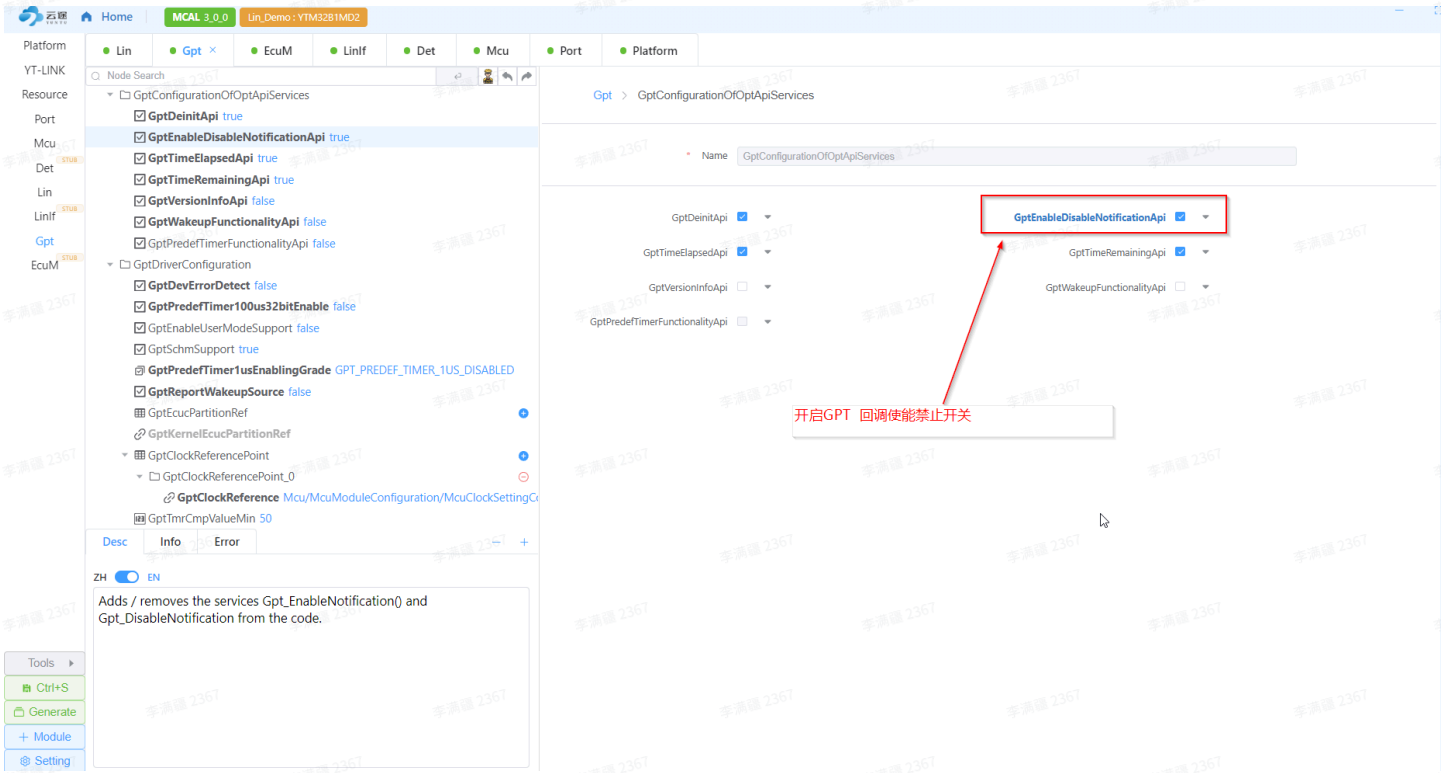

Home
MCAL 3_0_0
Lin_Demo : YTM32B1MD2
Platform
YT-LINK
Resource
Port
Mcu
Det
Lin
LinIf
Gpt
EcuM
Lin
Gpt
EcuM
LinIf
Det
Mcu
Port
Platform
Node Search
Lin
POST_BUILD_VARIANT_USED false
IMPLEMENTATION_CONFIG_VARIANT VariantPostBuild
AutosarExt
LinGeneral
LinDemEventParameterRefs
LIN_E_TIMEOUT
LinGlobalConfig
LinChannel
Door
Window
LinChannelId 1
LinChannelBaudRate 19200
LinNodeType SLAVE
LinSlaveForbiddenIDsEnable false
LinSlaveForbiddenIDsConfig
BreakLength MASTER_13_BIT_BREAK_LENGTH
DetectedBreakLength SLAVE_11_BIT_BREAK_LENGTH
LinResponseTimeout 14
LinHeaderTimeout 44
LinHwChannel UART_IP_1
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."
Tools
Ctrl+S
Generate
Module
Setting
Lin > LinGlobalConfig > LinChannel > Window
Name Window
LinChannelId DEC 1
LinChannelBaudRate DEC 19200
LinNodeType SLAVE
设置UART1为从节点
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
LinResponseTimeout DEC 14
LinHeaderTimeout DEC 44
LinHwChannel UART_IP_1
LinClockRef Mcu/McuModuleConfiguration/McuClockSettingConfig/McuClockSettingConfig_0/McuClockReferencePoint/UART1_CLK
LinChannelWakeupSupport
LinChannelEcuMWakeupSource Select
LinChannelEcucPartitionRef Select

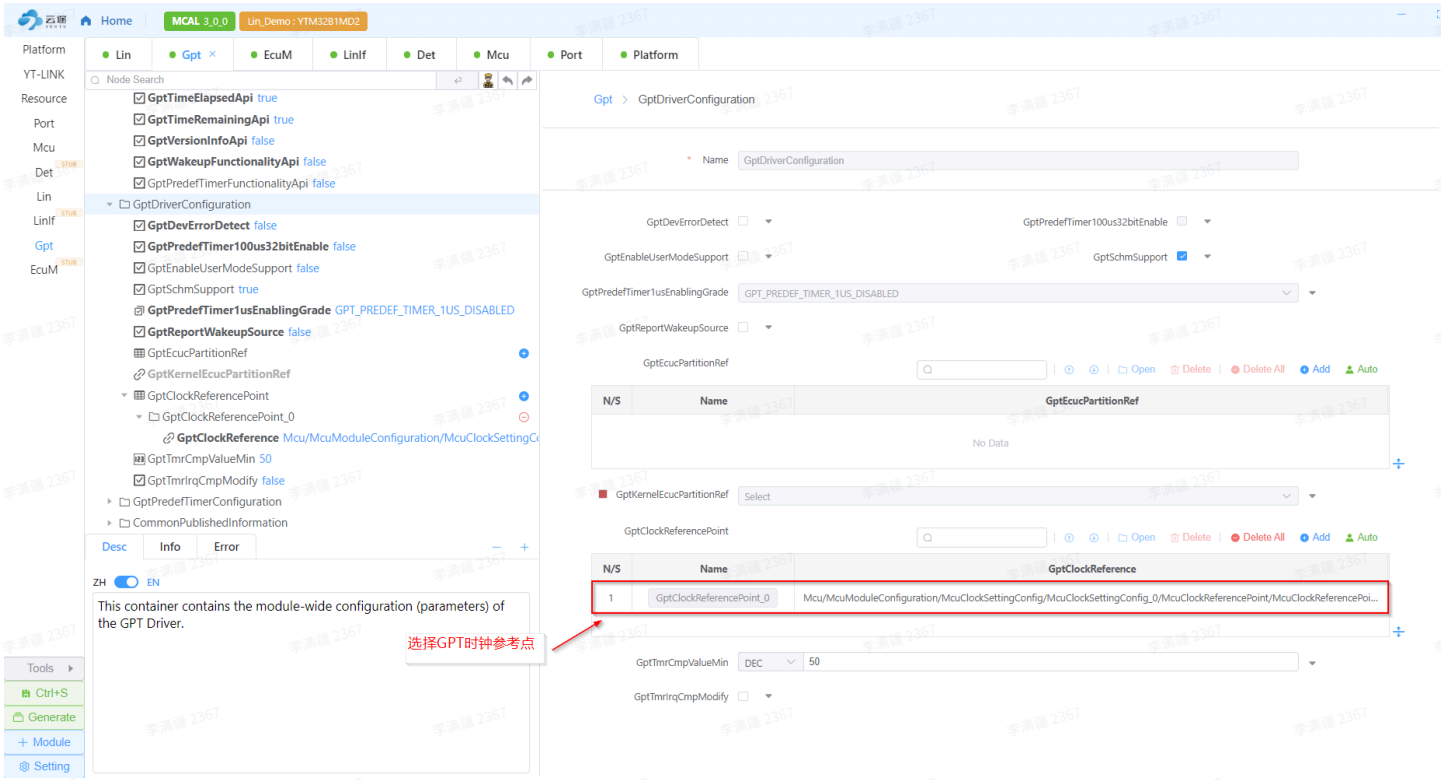
3.2.5 配置GPT 模块


选择pTMR 硬件和通道参考
选择参考时钟源
开启回调


GPT 内部对 ptmr0通道进行设置


Home
MCAL 3_0_0
Lin_Demo : YTM32B1MD2
Platform
YT-LINK
Resource
Port
Mcu
Det
Lin
LinIf
Gpt
EcuM
Lin
Gpt
EcuM
LinIf
Det
Mcu
Port
Platform
Node Search
GptPtmr
GptPtmr_0
GptPtmrModule PTMR_0
GptFuncClkSelection false
PtmrFreezeEnable true
GptPtmrChannels
GptPtmrChannels_0
GptPtmrChannel PTMR_0_CH_0
GptLptmr
GptRtc
GptHwConfiguration
GptHwConfiguration_0
GptHwConfiguration_1
GptHwConfiguration_2
GptHwConfiguration_3
GptHwConfiguration_4
GptHwConfiguration_5
GptConfigurationOfOptApiServices
GptDriverConfiguration
GptDevErrorDetect false
GptPredefTimer100us32bitEnable false
Desc
Info
Error
ZH
EN
List of all HW channel resources for GPT module.
Note: Not used for now.
Tools
Ctrl+S
Generate
Module
Setting
Gpt > GptHwConfiguration
Name
Gpt
GptHwConfiguration
Open
Delete
Delete All
Add
Auto
N/S
Name
GptIsrHwId
GptIsrEnable
GptChannelIsUsed
1 GptHwConfiguration_0 LPTMR_0_CH_0 true true
2 GptHwConfiguration_1 PTMR_0_CH_0 true true
3 GptHwConfiguration_2 PTMR_0_CH_1 true true
4 GptHwConfiguration_3 PTMR_0_CH_2 true true
5 GptHwConfiguration_4 PTMR_0_CH_3 true true
6 GptHwConfiguration_5 RTC_0_CH_0 true true
对PTMR0 硬件通道设置为开启，以及开启中断使能


Home
MCAL 3_0_0
Lin_Demo : YTM32B1MD2
Platform
YT-LINK
Resource
Port
Mcu
Det
Lin
LinIf
Gpt
EcuM
Lin
Gpt
EcuM
LinIf
Det
Mcu
Port
Platform
Node Search
GptConfigurationOfOptApiServices
GptDeinitApi true
GptEnableDisableNotificationApi true
GptTimeElapsedApi true
GptTimeRemainingApi true
GptVersionInfoApi false
GptWakeupFunctionalityApi false
GptPredefTimerFunctionalityApi false
GptDriverConfiguration
GptDevErrorDetect false
GptPredefTimer100us32bitEnable false
GptEnableUserModeSupport false
GptSchmSupport true
GptPredefTimer1usEnablingGrade GPT_PREDEF_TIMER_1US_DISABLED
GptReportWakeupSource false
GptEcucPartitionRef
GptKernelEcucPartitionRef
GptClockReferencePoint
GptClockReferencePoint_0
GptClockReference Mcu/McuModuleConfiguration/McuClockSettingC
GptTmrCmpValueMin 50
Desc
Info
Error
ZH
EN
Adds / removes the services Gpt_EnableNotification() and Gpt_DisableNotification from the code.
Tools
Ctrl+S
Generate
Module
Setting
Gpt > GptConfigurationOfOptApiServices
Name
GptConfigurationOfOptApiServices
GptDeinitApi
GptEnableDisableNotificationApi
GptTimeElapsedApi
GptTimeRemainingApi
GptVersionInfoApi
GptWakeupFunctionalityApi
GptPredefTimerFunctionalityApi
开启GPT 回调使能禁止开关



3.3 API说明

- **主从收发流程**

本例 Door 为逻辑通道 0，对应 URAT0；Window 为逻辑通道 1，对应 UART1。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。

- 下面为代码截图：

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
336
337   static void Board_Init(void)
338   {
339       Port_Init(&Port_Config);
340       Platform_Init(NULL_PTR);
341       Gpt_Init(&Gpt_Config);
342       Lin_Init(&Lin_Config);
343   }
344
345   /* USER CODE BEGIN 4 */
346   /* USER CODE END 4 */
347
```

GPT 模块要先于LIN 模块初始化

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 */
}
```

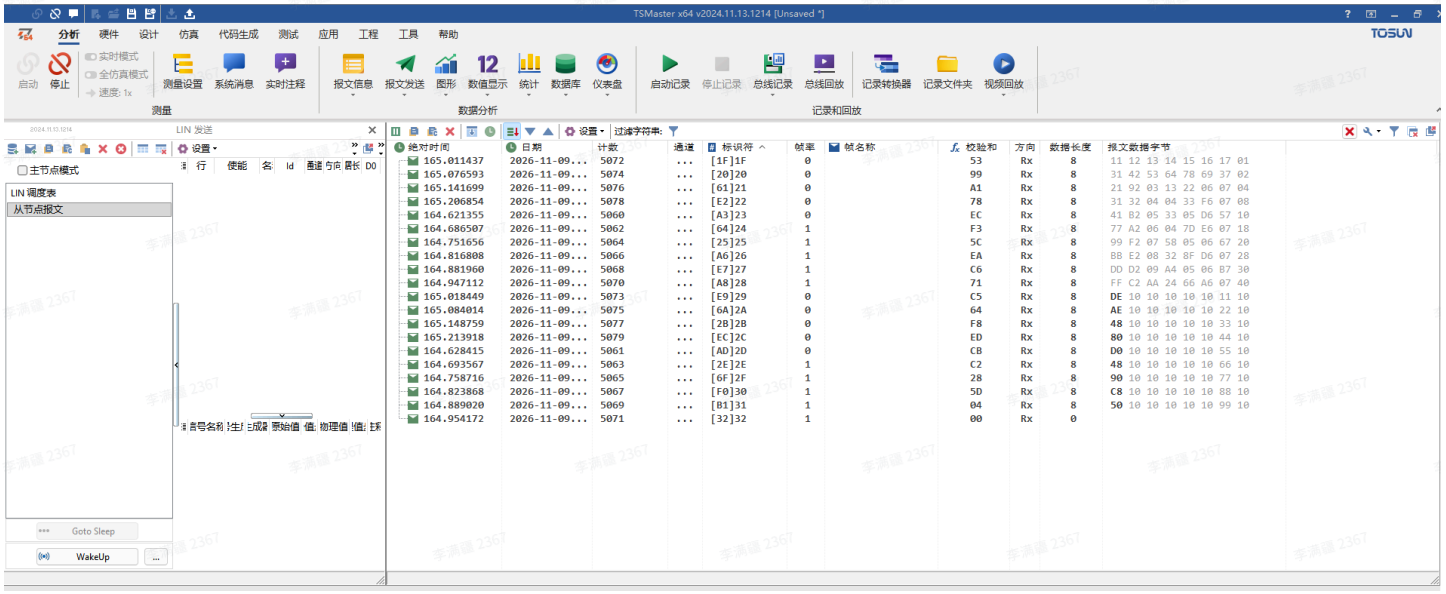
检查是否存在外部唤醒事件

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LIN0（UART0）做主机，LIN1（UART1）做从机，使用TSmaster 上位机从节点功能做数据被动监控，不发送帧头和响应。



结果分析：

ID	响应方向	配置 / 原因	期望现象与原记录
0x32	从机 → 主机	LinIf_HeaderIndication() 仅允许 0x29～0x31 发送	回调拒绝响应，属于应用回调过滤。

0x1F～0x28	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；
0x29～0x31	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；

文档历史

版本号	日期	修订记录
V1.0	2026.11.09	初始版本