Application Note

MSPM0 MCU 上的 MCAN (CAN FD) 模块入门

Hao Mengzhen

摘要

模块化控制器局域网 (MCAN) 外设是在 MSPM0™ 实时微控制器 (MCU) 系列内的部分器件上实现的 CAN 灵活数据速率 (CAN FD)。一些配有 MCAN 外设的器件包括 MSPM0G350x 和 MSPM0G351x 器件。所述的示例将在任何配有 MCAN 模块的 MSPM0 MCU 中运行。本应用手册介绍了几个编程示例，旨在说明如何针对不同的运行模式设置 MCAN 模块。其目的是帮助用户了解 MCAN 外设的编程。

已经在 MSPM0G3507 上测试了示例代码。不过，可以轻松调整这些示例，使其在任何配有 MCAN 模块的 MSPM0 器件上运行。大多数示例都需要第二个 CAN FD 节点才能运行。另一个配有 CAN FD 或配有任何 CAN 总线分析工具的 MCU 可以满足此要求，其中 CAN 总线分析工具能够同时使用经典 CAN 和 CAN FD 协议。目前有许多基于 USB 总线的工具可用。除了提供总线流量的可见性，这些工具还能够生成帧，对调试十分有用。配有内置 CAN FD 触发或解码功能的示波器对调试至关重要。

在本文档中，术语 MCAN 和 CAN FD 可以互换使用。*CAN FD* 是指协议，而 *MCAN* 是指 MCU 中实现协议的外设。本文档中所述的工程文件示例作为 MSPM0-SDK 的一部分提供，可下载。

内容

插图清单

商标

1 简介

CAN 是最初为汽车应用开发的串行协议。由于稳健性和可靠性，CAN 可用于工业设备、医疗电子产品、火车、飞机等各种应用中。CAN 协议具有复杂的错误检测和限制机制，能够在物理层进行简单布线。最初的 CAN 协议标准现在被称为*传统* CAN，以区别于较新的 CAN FD 标准。图 1-1 所示的是 CAN 总线的典型接线。

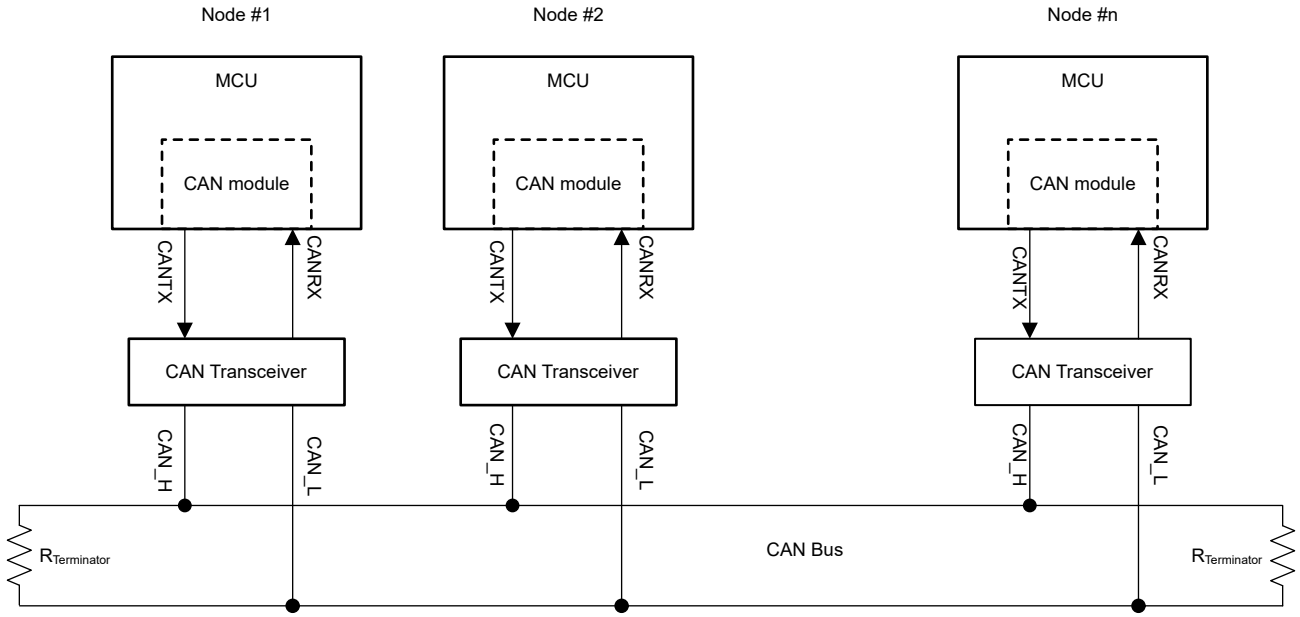

图 1-1. 典型 CAN 总线

CAN 灵活数据速率 (CAN FD) 在较高的位速率和一帧中传输的字节数方面增强了传统 CAN，从而提高了通信的有效吞吐量。传统 CAN 支持高达 1Mbps 的位速率和每帧 8 字节的有效载荷大小，而 CAN FD 支持高达 5Mbps 的比特率和每帧最多 64 字节的有效载荷大小。图 1-2 展示了 CAN FD 帧的帧结构。

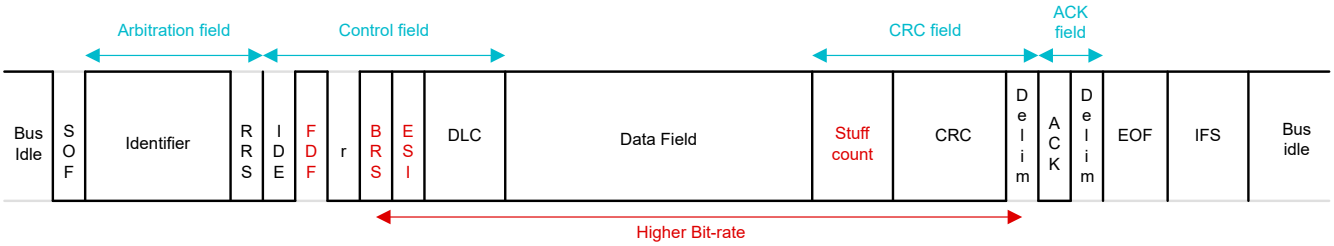

图 1-2. CAN FD 帧

1.1 MCAN 特性

MCAN 模块的显著特性如下：

- 符合 CAN 协议 2.0A、B 和 ISO 11898-1:2015 标准
- 完全支持 CAN FD（最多 64 个数据字节）
- 支持 AUTOSAR 和 SAE J1939
- 多达 32 个专用发送缓冲器
- 可配置的发送 FIFO，最多 32 个元素
- 可配置的发送队列，最多 32 个元素
- 可配置的发送 Event FIFO，最多 32 个元素
- 多达 14 个专用接收缓冲器
- 两个可配置的接收 FIFO，每个 FIFO 最多 14 个元素，具有 1kB 的消息 RAM
- 多达 128 个滤波器元素
- 用于自检的环回模式
- 可屏蔽中断（两条可配置的中断线路、可纠正的 ECC、计数器溢出和时钟停止或唤醒）
- 不可屏蔽中断（不可纠正的 ECC）
- 两个时钟域（CAN 时钟和主机时钟）
- 消息 RAM 的 ECC 检查
- 支持时钟停止和唤醒
- 时间戳计数器
- 1Mbps 标称比特率；8Mbps 数据比特率

不支持的特性：

- 主机总线防火墙
- 时钟校准
- 通过 CAN 进行调试

2 MCAN 模块的 Sysconfig 配置

SysConfig 是一款配置工具，旨在简化硬件和软件配置挑战，从而加速软件开发。SysConfig 可作为 Code Composer Studio™ 集成开发环境的一部分以及作为独立应用提供。此外，可以通过访问 TI 开发人员专区，在云中运行 SysConfig。SysConfig 提供直观的图形用户界面，用于配置引脚、外设、无线电、软件栈、RTOS、时钟树和其他元件。SysConfig 将自动检测、发现和解决冲突，来加快软件开发。

TI 建议用户使用 SysConfig 在应用程序中配置 MCAN 模块。SysConfig 可帮助用户轻松地根据需要快速设置 MCAN 模块。接下来的几节将介绍如何使用 SysConfig 通过 MSPM0-SDK 内的示例 mcan_loopback_lp_MSPM0G3507 来配置 MCAN。

2.1 MCAN 时钟频率

图 2-1 中列示了 MCAN 时钟频率块中包含哪些参数。

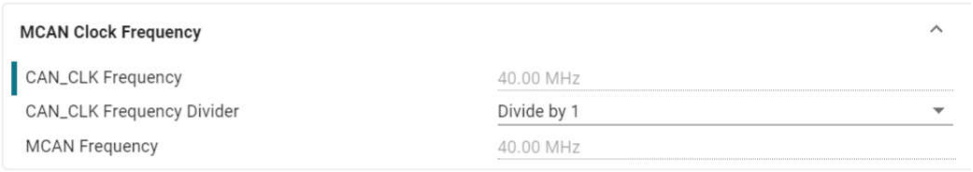

图 2-1. MCAN 时钟频率

可以通过 HFXT 或 SYSPLL 为 MCAN 的外设异步时钟 (CAN_CLK) 计时。选择 SYSCTL 来完成该时钟配置。TI 建议将 MCAN 频率配置为 40MHz。

备注

TI 建议使用 HFXT 作为 CAN_CLK 的时钟源。SYSOSC 的精度不能满足特殊情况下应用的要求，会导致出现更多的错误帧。CAN_CLK 的建议频率为 20MHz、40MHz 和 80MHz。

2.2 MCAN 基本配置

图 2-2 中列示了 MCAN 基本频率块中包含哪些参数。

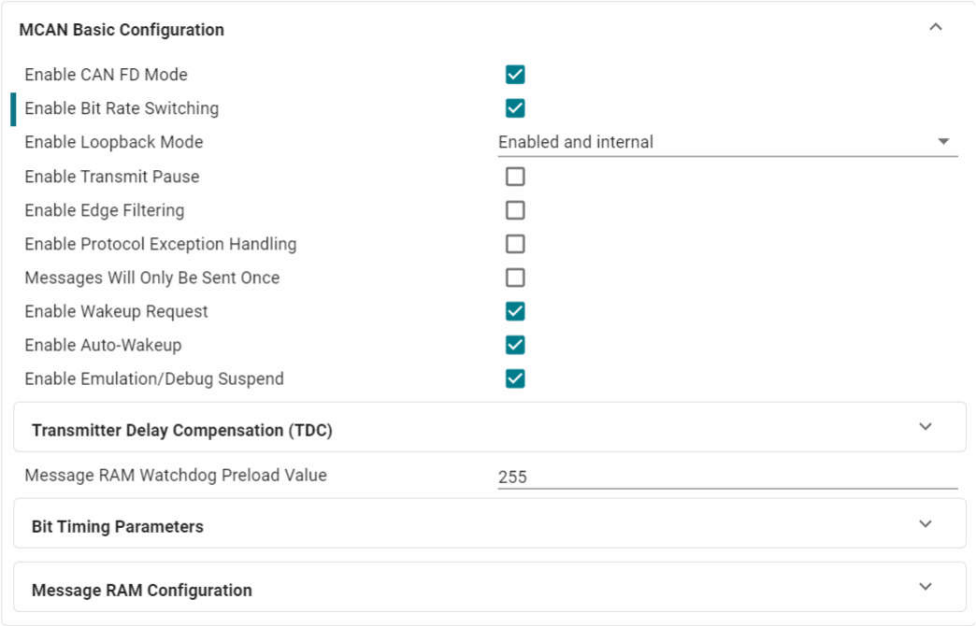

图 2-2. MCAN 基本配置

- Enable CAN FD Mode：在此模式下，需要启用 CAN 灵活数据模式。

- Enable Bit Rate Switching：通过启用该功能，MCAN 以更高的速率（而不是调停速率）发送数据。此功能仅在启用 CAN FD 模式时有效。
- Enable Loopback Mode：发送的消息成为了接收的消息。这使用户可以在无需 CAN 收发器的情况下监控 CAN_TX 引脚上的 CAN 消息。
- Enable Transmit Pause：在下一次传输之前暂停两个 CAN 位时间。传输暂停功能适用于 CAN 消息 ID 是特定的且无法轻易更改的 CAN 网络。这些消息 ID 可能具有比其他定义的消息 ID 更高的优先级，而在特定的应用中其相对优先级可能是相反的。这允许出现以下情况：一个 ECU 发送 CAN 消息突发，导致另一条 ECU 的 CAN 消息延迟（暂停）。
- Enable Edge Filtering：为实现硬同步在检测边沿时，需要两个连续的显性时间份额。启用此功能是为了确保节点可以准确检测信号边沿，并在数据位时间更长的情况下执行硬同步，从而提高 CAN 总线通信的稳定性和可靠性。
- Enable Protocol Exception Handling：检测为将来的协议扩展而保留的位。
- Messages Will Only Be Sent Once：如果禁用了自动重新发送，则在出现传输错误和 NACK 时或 MCAN 模块丢失调停时，MCAN 模块将不再重新发送。
- Enable Wakeup Request：在 CAN RXD 活动时唤醒 MCAN 模块。
- Enable Auto-Wakeup：使 MCAN 模块能够根据已启用的唤醒请求自动将 MCAN CCCR.INIT 位清零，从而完全唤醒 MCAN。发出时钟停止请求会使 MCAN 模块进入断电模式（睡眠模式）。在从空闲状态转换到激活状态期间，如果启用唤醒请求和启用自动唤醒功能，则发出读取-修改-写入命令以将 MCAN_CCCR.INIT 位清零。在 MCAN 内核响应删除时钟停止请求并删除时钟停止确认后，MCAN 内核恢复运行。

备注

请注意，硬件删除时钟停止请求后，不会收到第一个帧（唤醒帧）。这是因为此操作是在发出时钟停止后发生的，IP 中没有运行活动时钟。因此，在删除时钟停止请求之后，必须重新发送启用时钟的唤醒帧。

- Enable Emulation or Debug Suspend：可暂停 MCAN 模块以进行仿真或调试。
- Message RAM Watchdog Preload Value：RAM Watchdog 监控消息 RAM 的 READY 输出。通过 MCAN 的 Generic Master Interface 访问消息 RAM 时，以配置的值启动消息 RAM 监视器计数器。当消息 RAM 通过激活 READY 输出发出成功完成信号时，计数器将重新加载。如果在计数器倒计时至零之前消息 RAM 没有响应，则计数器停止并设置中断标志 MCAN_IR.WDI。RAM 看门狗计数器由主机（系统）时钟计时。

2.2.1 发送器延迟补偿 (TDC)

图 2-3 中列示了发送器延迟补偿 (TDC) 块中包含哪些参数。

图 2-3. 发送器延迟补偿 (TDC)

TDC 机制用于补偿 CAN FD 系统中收发器环路延迟引起的延迟。这种延迟可以防止节点在高比特率数据传输期间在采样点执行有意义的位错误检查。具体而言，TDC 在数据阶段引入了辅助采样点（辅助采样点，即 SSP），在考虑延迟后，将传输的位与接收的位进行比较。这可确保正确检测和处理位错误。

在比特率很高时，TDC 是必需的，这会导致数据位短和明显的环路延迟。这些延迟可能会导致节点错过位错误检测所需的正确采样点。TDC 仅在数据阶段处于活动状态，不会影响调停阶段。

通过使用 TDC，数据阶段的位时间可以比标称位时间更短，从而在不影响错误检测的情况下实现更高的数据速率。

- TDC Filter Window Length (Cycles)：该滤波器功能定义了 SSP 位置的最小值，以避免接收到的 FDF 位内的显性干扰在接收到的 res 位的下降沿之前结束延迟补偿测量（从而导致过早采用 SSP 位置）的情况。
- TDC Offset (Cycles)：该偏移用于调整接收位内 SSP 的位置（例如：数据阶段中位时间的一半）。

有关 TDC 更多详细信息，请参阅 MSPM0 G 系列 80MHz 微控制器技术参考手册。

2.2.2 位时序参数

图 2-4 中列示了哪些参数包含在 Bit Timing Parameters 块中。

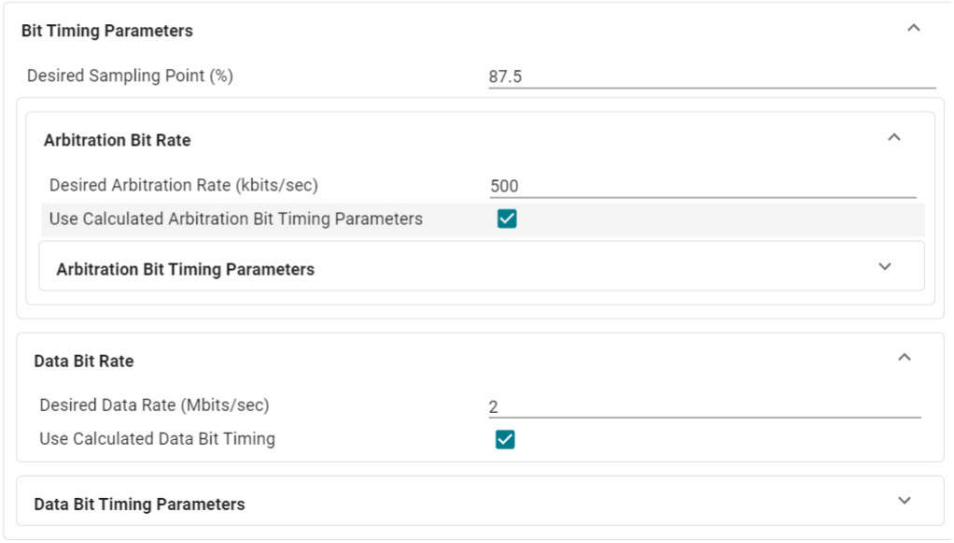

图 2-4. 位时序参数

CAN 总线中的位时序是指用于同步和控制 CAN 通信中数据传输的关键参数。这会将每个位的时间划分为多个时间段（称为 Time Quanta 或 TQ），以确保网络上的所有节点都能准确地接收和发送数据。位时序的设置包括几个关键组成部分：时间量子 (TQ)、同步段 (SyncSeg)、传播段 (PropSeg)、相位缓冲段 1 (PBSeg1)、相位缓冲段 2 (PBSeg2) 和采样点。位时序在数据传输中的作用体现在同步和一致性、数据完整性、容错、数据速率和网络性能以及抗干扰能力等方面。

- Desired Sampling Point (%)：在 CAN FD 中，所需采样点百分比为 15% 至 95%。此参数始终与总线上的其他 CAN 节点相匹配。
- Arbitration Bit Rate Configuration：所需的调停速率 (kbits/sec)：定义了调停速率。
- Use Calculated Arbitration Bit Timing Parameters：启用该配置后，SysConfig 会根据所需的采样点和调停速率，自动计算调停波特率预分频器、采样点前的时间、采样点后时间和 (Re) 同步跳转宽度范围。
- Data Bit Rate Configuration：所需数据速率 (kbits/sec)：定义数据速率。
- Use Calculated Data Bit Timing Parameters：启用该配置后，SysConfig 会根据所需的采样点和数据速率，自动计算数据波特率预分频器、采样点前的时间、采样点后时间和 (Re) 同步跳转宽度范围。

2.2.3 消息 RAM 配置

MCAN 模块有一个消息 RAM。消息 RAM 的主要用途是存储：

1. 消息 ID 滤波器元素
2. 传输消息
3. Tx 事件元素
4. 收到的消息

本应用手册介绍了消息 RAM 配置为 1KB 大小，宽度为 32 位。

备注

TI 建议通过 SysConfig 配置消息 RAM。SysConfig 会通知用户当前配置占用的地址空间。这有助于用户避免地址重叠问题。

2.2.3.1 标准和扩展 ID 滤波器配置

图 2-5 中列示了标准和扩展 ID 滤波器配置块中包含哪些参数。

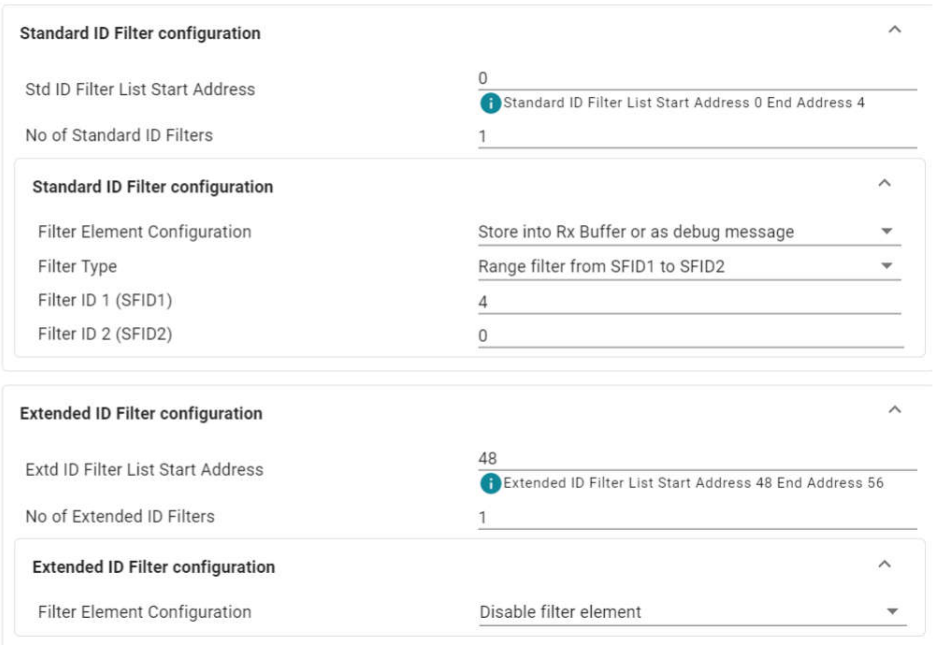

图 2-5. 标准和扩展 ID 滤波器配置

- Std ID Filter List Start Address：每个标准 ID 滤波器占用 4 个消息 RAM 地址。
- Number of Standard ID Filters：最多可以为 11 位标准 ID 配置 128 个滤波器元素。SysConfig 当前不支持配置多个滤波器。可以在用户应用程序中添加更多滤波器，但请确保在初始化期间分配足够的 RAM。
- Standard ID Filter configuration → Filter Element Configuration：所有启用的滤波器元素均用于标准帧的接受过滤。接受过滤在第一个匹配的已启用过滤器元素处或到达过滤器列表末尾时停止。该参数的选项如下所示。
  - 0x0：禁用滤波器元素
  - 0x1：如果滤波器匹配，则存储在 Rx FIFO 0 中
  - 0x2：如果滤波器匹配，则存储在 Rx FIFO 1 中
  - 0x3：如果滤波器匹配，则拒绝 ID
  - 0x4：如果滤波器匹配，则设置优先级
  - 0x5：如果滤波器匹配，则设置优先级并存储在 FIFO 0 中
  - 0x6：如果滤波器匹配，则设置优先级并存储在 FIFO 1 中
  - 0x7：存储到 Rx 缓冲器中，忽略标准滤波器类型的配置
- Standard ID Filter configuration → Filter Type：标准滤波器类型配置。该参数的选项如下所示。
  - 0x0：从 SFID1 到 SFID2 的范围滤波器 (SFID2 ≥ SFID1)
  - 0x1：用于 SFID1 或 SFID2 的双 ID 滤波器
  - 0x2：传统滤波器：SFID1 = 过滤器；SFID2 = 掩码
  - 0x3：禁用的滤波器元素
- Standard ID Filter configuration → Filter ID 1 (SFID1)：标准滤波器 ID 1。在过滤 Rx 缓冲区时，该字段定义要存储的标准消息的 ID。接收到的标识符必须完全匹配，不使用掩码机制。
- Standard ID Filter configuration → Filter ID 2 (SFID2)：标准滤波器 ID 2。根据滤波器元素配置，此 ID 具有不同的定义。如果滤波器元素配置为 0x1 至 0x6，则 SFID2 是标准 ID 滤波器元素的第二个 ID。如果滤波器元素配置为 0x7，则 SFID2 是 Rx 缓冲器的滤波器。

扩展 ID 滤波器的配置如下所示。

- Extd ID Filter List Start Address：每个扩展 ID 滤波器占用 8 个消息 RAM 地址。
- Number of Extended ID Filters：最多可以为 29 位扩展 ID 配置 64 个滤波器元素。SysConfig 当前不支持配置多个滤波器。可以在用户应用程序中添加更多滤波器，但请确保在初始化期间分配足够的 RAM。
- Extended ID Filter configuration → Filter Element Configuration：所有启用的滤波器元素均用于扩展帧的接受过滤。接受过滤在第一个匹配的已启用过滤器元素处或到达过滤器列表末尾时停止。

  - 0x0：禁用滤波器元素
  - 0x1：如果滤波器匹配，则存储在 Rx FIFO 0 中
  - 0x2：如果滤波器匹配，则存储在 Rx FIFO 1 中
  - 0x3：如果滤波器匹配，则拒绝 ID
  - 0x4：如果滤波器匹配，则设置优先级
  - 0x5：如果滤波器匹配，则设置优先级并存储在 FIFO 0 中
  - 0x6：如果滤波器匹配，则设置优先级并存储在 FIFO 1 中
  - 0x7：存储到 Rx 缓冲区中或存储为调试消息，忽略扩展滤波器类型配置
- Extended ID Filter configuration → Filter Type：已扩展的滤波器类型配置。该参数的选项如下所示。
  - 0x0：从 EFID1 到 EFID2 的范围滤波器 (EFID2 ≥ EFID1)
  - 0x1：用于 EFID1 或 EFID2 的双 ID 滤波器
  - 0x2：传统滤波器：EFID1 = 过滤器，EFID2 = 掩码
  - 0x3：从 EFID1 到 EFID2 的范围滤波器 (EFID2 ≥ EFID1)，未应用扩展 ID 和掩码
- Extended ID Filter configuration → Filter ID 1 (EFID1)：已扩展的滤波器 ID 1。已扩展 ID 的滤波器元素的第一个 ID。在过滤 Rx 缓冲区时，该字段定义要存储的扩展消息的 ID。
- Extended ID Filter configuration → Filter ID 2 (EFID2)：已扩展的滤波器 ID 2。根据扩展滤波器元素配置，此 ID 具有不同的定义。如果扩展滤波器元素配置为 0x1 至 0x6，则 EFID2 是扩展 ID 滤波器元素的第二个 ID。如果扩展滤波器元素配置为 0x7，则 EFID2 是 Rx 缓冲器的滤波器。

2.2.3.1.1 如何添加更多滤波器

SysConfig 当前不支持多个滤波器的配置。可以在用户应用程序中添加更多滤波器，但请确保在初始化期间分配足够的 RAM。在应用程序代码中配置滤波器之前，请记住在 SysConfig 中配置滤波器数量。

下例所示的是如何在应用程序代码中添加更多滤波器。在 SysConfig 中，起始地址配置为 0x0，滤波器数量配置为 2。

```
static const DL_MCAN_StdMsgIDFilterElement gMCAN0StdFiltelem_0 = {
    .sfec = 0x1,
    .sft  = 0x0,
    .sfid1 = 3,
    .sfid2 = 4,
};

static const DL_MCAN_StdMsgIDFilterElement gMCAN0StdFiltelem_1 = {
    .sfec = 0x10,
    .sft  = 0x0,
    .sfid1 = 13,
    .sfid2 = 14,
};
/* Configure Standard ID filter element */
DL_MCAN_addStdMsgIDFilter(MCAN0_INST, 0U, (DL_MCAN_StdMsgIDFilterElement *) &gMCAN0StdFiltelem_0);
DL_MCAN_addStdMsgIDFilter(MCAN0_INST, 1U, (DL_MCAN_StdMsgIDFilterElement *) &gMCAN0StdFiltelem_1);
```

下例所示的是如何添加更多扩展滤波器。

```
static const DL_MCAN_ExtMsgIDFilterElement gMCAN0ExtFiltelem_0 = {
    .efec = 0x1,
    .eft  = 0x2,
    .efid1 = 0x3,
    .efid2 = 0x1FFFFFFF,
};
/* Configure Extended ID filter element */
DL_MCAN_addExtMsgIDFilter(MCAN0_INST, 0U, (DL_MCAN_ExtMsgIDFilterElement *) &gMCAN0ExtFiltelem_0);
```

2.2.3.2 TX MSG RAM

图 2-6 中列示了 TX MSG RAM 块中包含哪些参数。

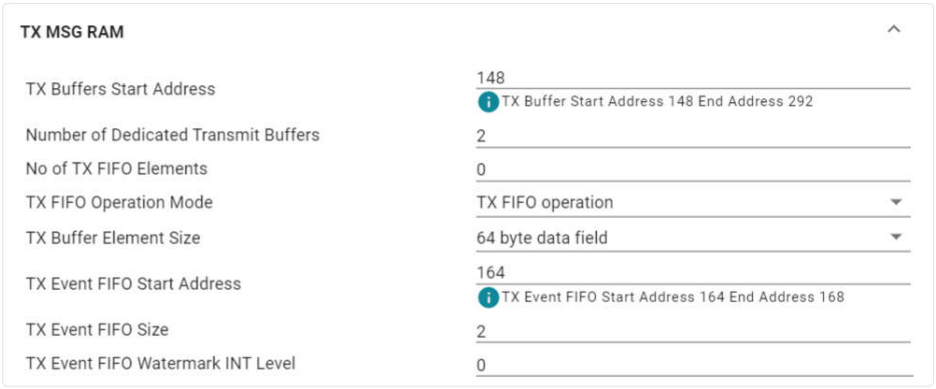

图 2-6. TX MSG RAM

Tx 缓冲区部分可配置为保存专用 Tx 缓冲区、Tx FIFO 和 Tx Queue。如果 Tx 缓冲区部分由专用 Tx 缓冲区、Tx FIFO 和 Tx Queue 共享，则专用 Tx 缓冲区从 Tx 缓冲区部分的开头开始，后跟分配给 Tx FIFO 或 Tx Queue 的缓冲区。表 2-1 展示了 Tx 缓冲区模式，Tx FIFO 模式和 Tx 队列模式之间的差异。

表 2-1. Tx 模式之间的差异

Tx 模式	说明
Tx 缓冲模式	专用 Tx 缓冲区用于在主机 CPU 完全控制下的消息传输。
Tx FIFO 模式	Tx FIFO 允许从不同的 Tx 缓冲区发送具有相同消息 ID 的消息，其发送顺序为这些消息写入 Tx FIFO 的顺序。
Tx 队列模式	存储在 Tx 队列中的消息从优先级最高的消息（最低消息 ID）开始发送。

- TX Buffers Start Address：消息 RAM 中 Tx 缓冲区的起始地址。
- Number of Dedicated Transmit Buffers：定义配置为专用 Tx 缓冲区的元素数量。
- No of TX FIFO Elements：定义配置为 Tx FIFO 或 Tx 队列的元素数量。
- TX FIFO Operation Mode：定义 Tx FIFO 模式或 Tx 队列模式。
- TX Buffer Element Size：定义 Tx 缓冲区数据字段大小。如果 Tx 缓冲区元素的数据长度代码 (DLC) 配置为一个高于 Tx 缓冲区数据字段大小的值，则未由 Tx 缓冲区定义的字节将作为 *0xCC*（填充字节）传输。

- TX Event FIFO Start Address：Tx 事件 FIFO 存储有关已发送消息的信息。为了支持 Tx 事件处理，消息 RAM 实现了 Tx 事件 FIFO 段。通过读取 Tx 事件 FIFO，主机 CPU 按照消息发送的顺序获取该信息。在 CAN 总线上传输消息后，消息 ID 和时间戳存储在 Tx 事件 FIFO 元素中。为了将 Tx 事件链接到 Tx 事件 FIFO 元素，发送的 Tx 缓冲区中的消息标记被复制到 Tx 事件 FIFO 元素中。
- TX Event FIFO Size：最多可配置 32 个 Tx 事件 FIFO 元素。
- TX Event FIFO Watermark INT Level：定义 Tx 事件 FIFO 填充级别阈值。可以配置 Tx 事件 FIFO 水线，以避免 Tx 事件 FIFO 溢出。

2.2.3.3 RX MSG RAM

图 2-7 中列示了 RX MSG RAM 块中包含哪些参数。

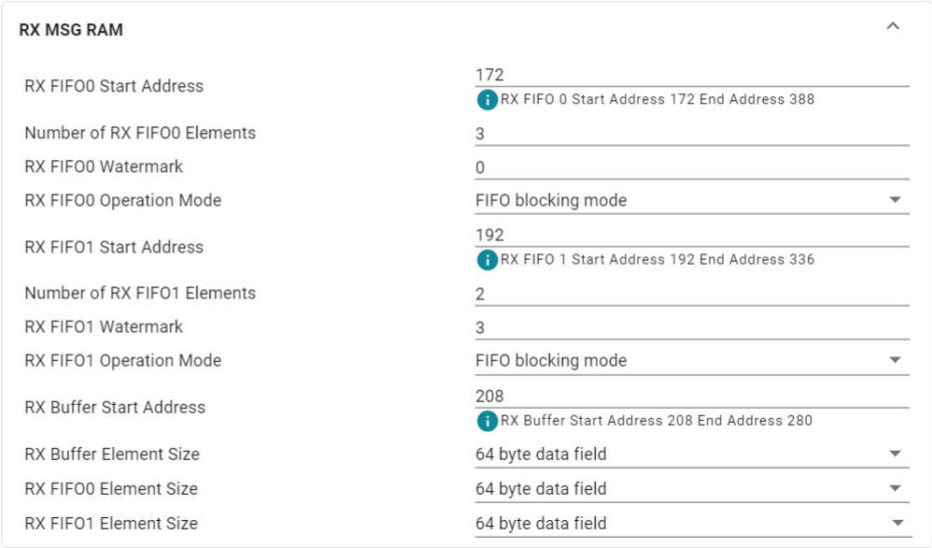

图 2-7. RX MSG RAM

消息 RAM 中最多可配置 64 个 Rx 缓冲区和两个 Rx FIFO。每个 Rx FIFO 段可配置为存储最多 64 条接收到的消息。通过配置元素大小，存储最多 64 字节数据字段的 CAN FD 消息。

- RX FIFO0 and RX FIFO1 Start Address：定义消息 RAM 中 Rx FIFO 的起始地址。
- Number of RX FIFO0 and RX FIFO1 Elements：每个 Rx FIFO 可配置为最多存取 64 条已接收到的消息。
- RX FIFO0 and RX FIFO1 Watermark：Rx FIFO 水线可用于防止 Rx FIFO 溢出。如果 Rx FIFO 填充级别达到 Rx FIFO 水线，则会设置中断标志 MCAN_IR.RF0W/MCAN_IR.RF1W。
- RX FIFO0 and RX FIFO1 Operation Mode：
  - Rx FIFO Blocking Mode：Rx FIFO 阻塞模式是 Rx FIFO 的默认工作模式。如果达到 Rx FIFO 已满状态，则不会再将任何消息写入相应的 Rx FIFO，直到至少一条消息被读出并且 Rx FIFO Get 索引已递增。
  - Rx FIFO Overwrite Mode：当达到 Rx FIFO 已满状态时，FIFO 的下一条被接受的消息将覆盖最早的 FIFO 消息。
- RX FIFO0 and RX FIFO1 Element Size：定义 Rx FIFO 元素大小。
- RX Buffer Start Address：定义消息 RAM 中 Rx 缓冲区的起始地址。
- RX Buffer Element Size：定义 Rx 缓冲区元素大小。

2.3 高级配置

图 2-8 中列示了高级配置块中包含哪些参数。

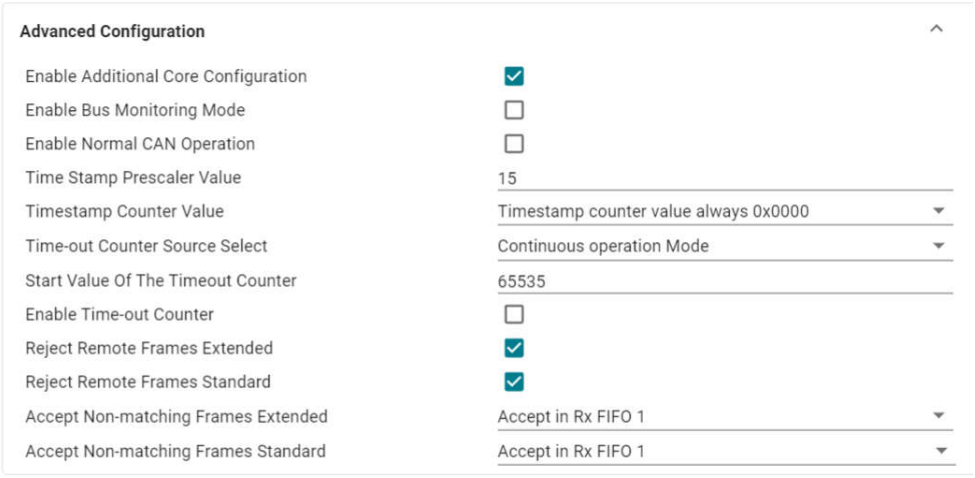

图 2-8. 高级配置

- Enable Additional Core Configuration：启用或禁用附加功能，例如时间戳和处理不匹配的帧。
- Enable Bus Monitoring Mode：在总线监控模式中（请参阅 ISO 11898-1:2015 总线监控部分），MCAN 模块能够接收有效数据和远程帧，但无法开始传输。MCAN 模块仅在 CAN 总线上发送隐性位。如果 MCAN 模块需要发送显性位（ACK 位、过载标志和活动错误标志），则该位会在内部重新路由，以便 MCAN 模块监控该显性位。CAN 总线可能仍处于隐性状态。总线监控模式可用于分析 CAN 总线上的流量，而不会因显性位的传输而影响总线。
- Enable Normal CAN Operation：将正常 CAN 运行模式定义为受限运行模式。在受限运行模式下，CAN 节点能够接收数据和远程帧并对有效帧进行确认，但节点不会发送数据帧、远程帧、活动错误帧或过载帧。如果出现错误情况或过载情况，则节点不会发送显性位；相反，节点等待出现总线空闲条件，以将自身重新与 CAN 通信同步。当 Tx 处理程序无法及时从消息 RAM 读取数据时，会自动进入受限运行模式。要离开受限运行模式，主机 CPU 必须重置 MCAN_CCCR.ASM 位。该模式可用于需要适应不同 CAN 比特率的应用。在这种情况下，应用会测试不同的比特率，并在节点接收到有效帧后退出受限运行模式。
- Time Stamp Prescaler Value：MCAN 模块集成了一个用于时间戳生成的 16 位绕回计数器。时间戳计数器预分频器 MCAN_TSCC.TCP 字段可配置为以 CAN 位时间 (1-16) 的倍数对计数器进行计时。在帧接收/传输开始时，会捕获计数器值并将其存储到 Rx 缓冲区、Rx FIFO 或 Tx Event FIFO 元素的时间戳部分中。
- Timestamp Counter Value：将时间戳计数器值配置为 0x0，来自内部 16 位计数器或外部时间戳。
- Time-out Counter Source Select：MCAN 模块集成了一个 16 位超时计数器。超时计数器用于为 Rx FIFO 0、Rx FIFO 1 和 Tx 事件 FIFO 消息 RAM 元素指示超时状态。在连续模式下，计数器立即以 MCAN_TOCC.TOP 字段配置的值重新启动。如果超时计数器由 FIFO 之一控制，则空 FIFO 会将计数器预设为 MCAN_TOCC.TOP 字段配置的值。存储第一个 FIFO 元素时，开始向下计数。
- Start Value Of The Timeout Counter：定义超时持续时间。
- Enable Time-out Counter：使能超时功能。
- Reject Remote Frames Extended：过滤或拒绝所有具有 29 位扩展 ID 的远程帧。
- Reject Remote Frames Standard：过滤或拒绝所有具有 11 位标准 ID 的远程帧。
- Accept Non-matching Frames Extended：定义如何处理接收到的具有 29 位 ID 且与滤波器列表中任何元素均不匹配的消息。
- Accept Non-matching Frames Standard：定义如何处理接收到的具有 11 位 ID 且与滤波器列表中任何元素均不匹配的消息。

2.4 保留配置

图 2-9 中列示了保留配置块中包含哪些参数。

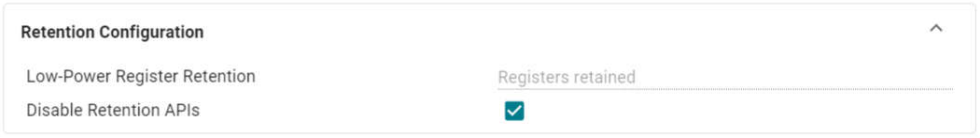

图 2-9. 保留配置

- Low-Power Register Retention：驻留在 PD1 域中的一些 MSPM0G 外设在进入停止或待机模式时不保留寄存器内容。开发人员可以决定在应用中使用 SysConfig 的默认初始化来重新初始化外设。这种方法更节省内存。或者，用户也可以使用提供的 DriverLib API 在进入低功耗模式之前和之后保存和恢复外设的寄存器配置。如果在运行时修改外设配置，建议使用此方法。
- Disable Retention APIs：选中时，无论选择何种外设，都不会生成保留 API。

2.5 中断

图 2-10 中列示了中断块中包含哪些参数。

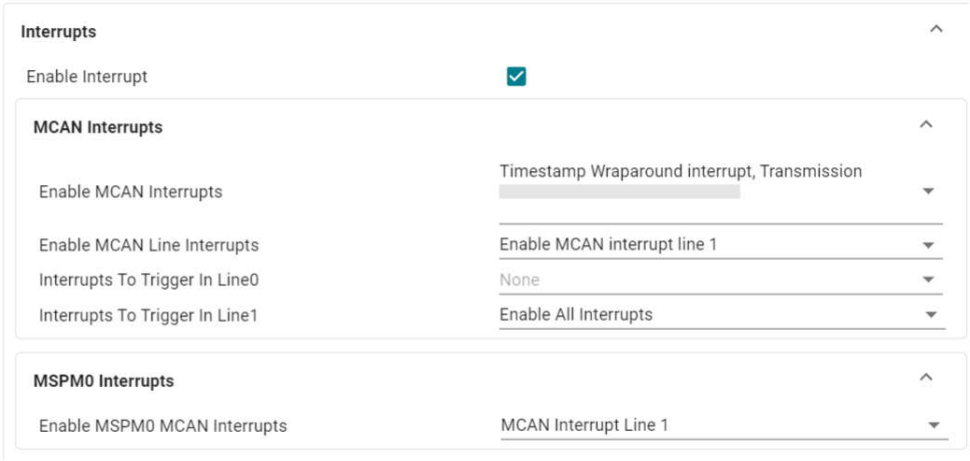

图 2-10. 中断

MCAN 模块包含一个事件发布者 (CPU_INT)，用于通过静态事件路由管理对 CPU 子系统的 MCAN 中断请求 (IRQ)。图 2-11 展示了器件中的 MCAN 模块集成。

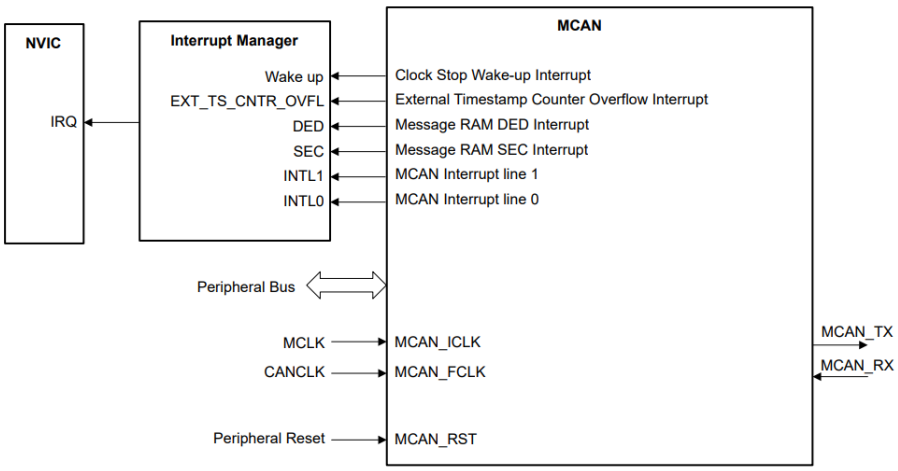

图 2-11. MCAN 集成

- MCAN 中断
  - Enable MCAN Interrupts：MCAN 内核有 2 条中断线路和 30 个内部中断源。每个源都可以配置为驱动两条中断线路之一。MCAN 内核提供两个中断请求，即，中断 Line0 和中断 Line1。
  - Enable MCAN Line Interrupts：定义应用中使用的中断线路。
  - Interrupts To Trigger In Line0：定义了哪些中断源被分配给了 Interrupt Line0。
  - Interrupts To Trigger In Line1：定义了哪些中断源被分配给了 Interrupt Line1。
- MSPM0 中断
  - Enable MSPM0 MCAN Interrupts：MCAN 模块提供多个不同的中断源，这些中断源可配置为产生 CPU 中断事件。为了降低中断优先级，在此配置 MCAN 的 CPU 中断事件。

2.6 引脚配置和 PinMux

图 2-12 中列示了引脚配置和 PinMux 块中包含哪些参数。

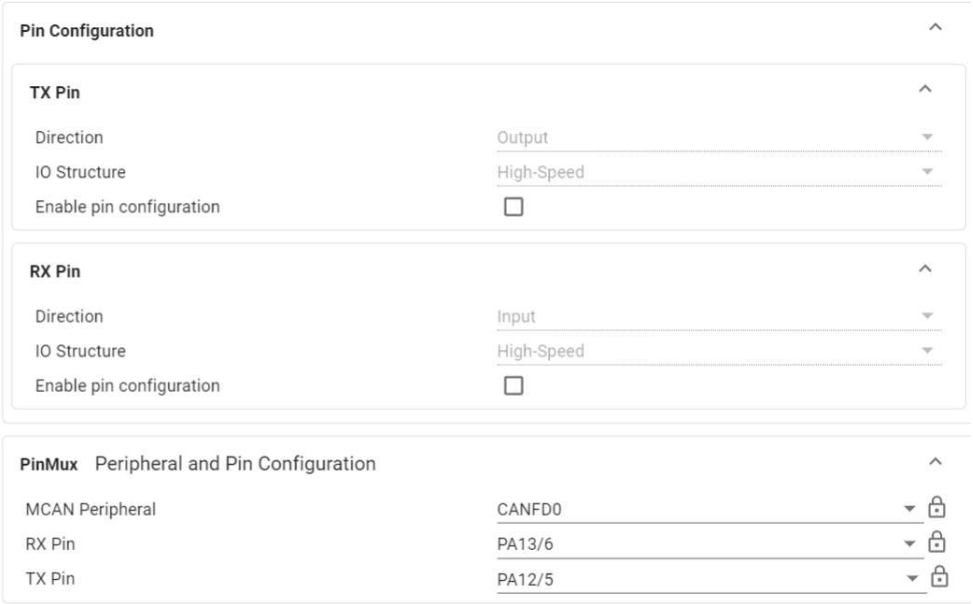

图 2-12. 引脚配置和 PinMux

- 引脚配置
  - TX 引脚和 RX 引脚：在专用引脚上配置数字 IOMUX 功能。例如内部上拉或下拉电阻、反相输出、驱动强度和高阻抗设置。TI 建议为 CAN 应用保留 IOMUC 配置的默认配置。
- 引脚 Mux
  - MCAN 外设：一些 MSPM0 器件集成了多个 MCAN 模块。用户可以选择在此处配置哪个 MCAN 模块。
  - TX/RX 引脚：为 MCAN TX 和 RX 功能配置 GPIO 引脚。

3 演示项目说明

- *mcan_loopback*

此示例说明了 MCAN 模块的环回功能。回送操作完全在模块内部执行。然而，传输的数据在 MCANTX 引脚中可见。该测试用例的一个优势是不需要收发器，因此环回操作可以在 LaunchPad™ 板上运行。为了便于轻松分析逻辑分析仪上的数据，仅传输四个字节的数据。然而，数据在禁用比特率切换的情况下作为 CAN 帧发送。

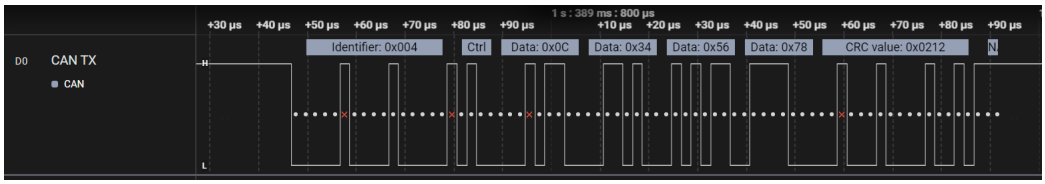

图 3-1. MCAN 环回消息

- *mcan_multi_message_tx*

该示例演示了用于发送多条消息的 MCAN 外部传输功能。在两个 CAN 节点之间进行外部通信。接收节点可以是另一个能够接收或确认已发送帧的 MCU 或 CAN 总线分析工具。通过 CAN 收发器连接两个 CAN 节点。此示例可与 mcan_multi_message_rx 示例项目一起使用。使用 250kbps 标称比特率和数据比特率。

TX 消息在缓冲模式下存储在 CAN 消息 RAM 中。然后，使用软件调用添加请求让传输 API 在所需的 TX 缓冲区中传输消息。

Index	Time	Device	Channel	Frame ID	Type	CANType	RT	Len	Data
					ALL	ALL	ALL		
0	0.000000	Devic...	0	0x3	StandardFrame	CANFD Accelerate	Rx	1	00
1	0.000200	Devic...	0	0x4	StandardFrame	CANFD Accelerate	Rx	1	00

图 3-2. 用于 mcan_multi_message_tx 的总线监控工具输出

- *mcan_multi_message_rx*

该示例演示了 MCAN 接收功能。传输节点可以是另一个能够传输 CAN FD 帧的 MCU 或 CAN 总线分析工具。使用 250kbps 的标称比特率和 2Mbps 的数据比特率。仅接收标准消息 ID 为 0x3 和 0x4 的帧。如果将另一个具有 MCAN 模块的 MCU 用作发送器，则可以为传输功能运行 mcan_multi_message_tx 示例项目。

- *mcan_multi_message_tx_tcan114x*

该示例演示了使用 BOOSTXL-TCAN1145 BoosterPack™ 发送多条消息的 MCAN 外部发送功能。在两个 CAN 节点之间进行外部通信。接收节点可以是另一个能够接收或确认已发送帧的 MCU 或 CAN 总线分析工具。通过 CAN 收发器连接两个 CAN 节点。可将此示例与 mcan_multi_message_rx_tcan114x 示例项目一同使用。使用 250kbps 的标称比特率和 2Mbps 的数据比特率。

软件首先通过 SPI 初始化 TCAN114x 模块。同时，TX 消息在缓冲模式下被存储在 CAN 消息 RAM 中。然后，使用软件调用添加请求让传输 API 在所需的 TX 缓冲区中传输消息。

Index	Time	Device	Channel	Frame ID	Type	CANType	RT	Len	Data
					ALL	ALL	ALL		
0	0.000000	Devic...	0	0x4	StandardFrame	CANFD Accelerate	Rx	1	00
1	2.122000	Devic...	0	0x3	StandardFrame	CANFD Accelerate	Rx	1	00

图 3-3. mcan_multi_message_tx_tcan114x 的总线监控工具输出

- *mcan_multi_message_rx_tcan114x*

此示例使用 BOOSTXL-TCAN1145 BoosterPack 演示了 MCAN 接收功能。传输节点可以是另一个能够传输 CAN FD 帧的 MCU 或 CAN 总线分析工具。使用 250kbps 的标称比特率和 2Mbps 的数据比特率。仅接收标准消息 ID

为 0x3 和 0x4 的帧。如果使用另一个配有 MCAN 模块的 MCU 作为发送器，则可为发送功能运行 mcan_multi_message_tx_tcan114x 示例项目。

- *mcan_single_message_tx*

该示例演示了用于发送信号消息的 MCAN 外部发送功能。在两个 CAN 节点之间进行外部通信。接收节点可以是另一个能够接收或确认已发送帧的 MCU 或 CAN 总线分析工具。通过 CAN 收发器连接两个 CAN 节点。此示例可与 mcan_multi_message_rx 示例项目一起使用。使用 250kbps 的标称比特率和 2Mbps 的数据比特率。

TX 消息在缓冲模式下存储在 CAN 消息 RAM 中。使用软件调用添加请求让传输 API 在 TX 缓冲区中传输消息。

Index	Time	Device	Channel	Frame ID	Type	CANType	RT	Len	Data
					ALL	ALL	ALL		
0	0.000000	Devic...	0	0x4	StandardFrame	CANFD Accelerate	Rx	1	00

图 3-4. mcan_single_message_tx 总线监控工具的输出

3.1 TX 缓冲模式

在 SDK 中，TX 在当前演示项目中配置为缓冲模式。下述讨论了如何在缓冲器模式下使用 TX。

```
DL_MCAN_TxBufElement txMsg;
/* Initialize message to transmit. */
/* Identifier Value. */
txMsg.id = ((uint32_t)(0x4)) << 18U;
/* Transmit data frame. */
txMsg.rtr = 0U;
/* 11-bit standard identifier. */
txMsg.xtd = 0U;
/* ESI bit in CAN FD format depends only on error passive flag. */
txMsg.esi = 0U;
/* Transmitting 4 bytes. */
txMsg.dlc = 1U;
/* CAN FD frames transmitted with bit rate switching. */
txMsg.brs = 1U;
/* Frame transmitted in CAN FD format. */
txMsg.fdf = 1U;
/* Store Tx events. */
txMsg.efc = 1U;
/* Message Marker. */
txMsg.mm = 0xAAU;
/* Data bytes. */
txMsg.data[0] = LED_STATUS_ON;

/* Write Tx Message to the Message RAM. */
DL_MCAN_writeMsgRam(MCAN0_INST, DL_MCAN_MEM_TYPE_BUF, 0U, &txMsg);

/* Add request for transmission. */
DL_MCAN_TXBufAddReq(MCAN0_INST, 0U);
```

首先，通过调用 DL_MCAN_writeMsgRam()，将 TX 消息以缓冲区类型与缓冲区编号保存到消息 RAM 中。然后，使用添加请求通过调用 DL_MCAN_TXBufAddReq() 与此消息的缓冲区编号信息一起传输。

3.2 TX FIFO 模式

在 SDK 中，TX 在当前演示项目中配置为缓冲模式。下文描述了如何在 FIFO 模式下使用 TX。

```
uint8_t MCAN_send_frame(uint32_t id, uint8_t* data, uint16_t len)
{
    frame_send_success = false;
    DL_MCAN_TxBufElement txMsg;
    DL_MCAN_TxFIFOStatus txfifoStatus;

    /* Initialize message to transmit. */
    /* Identifier Value. */
    txMsg.id = id;
    /* Transmit data frame. */
    txMsg.rtr = 0U;
    /* 11-bit standard identifier. */
    txMsg.xtd = 0U;
    /* ESI bit in CAN FD format depends only on error passive flag. */
    txMsg.esi = 0U;
    /* Transmitting 4 bytes. */
    txMsg.dlc = encode_dlc(len);
    /* CAN FD frames transmitted with bit rate switching. */
    txMsg.brs = protocol_mode;
    /* Frame transmitted in CAN FD format. */
    txMsg.fdf = protocol_mode;  //protocol_mode;
    /* Store Tx events. */
    txMsg.efc = 1U;
    /* Message Marker. */
    txMsg.mm = 0xAAU;
    /* Data bytes. */
    for (int i = 0; i < len; i++) txMsg.data[i] = data[i];
    while (DL_MCAN_OPERATION_MODE_NORMAL != DL_MCAN_getOpMode(CANFD0))
        ;

    /* Write Tx Message to the Message RAM (FIFO). */
    DL_MCAN_writeMsgRam(CANFD0, DL_MCAN_MEM_TYPE_FIFO, 0, &txMsg);

    /* Get put index and other TxFIFO details in txfifoStatus*/
    DL_MCAN_getTxFIFOQueStatus(CANFD0, &txfifoStatus);

    /* Enable Transmission interrupt.*/
    DL_MCAN_TXBufTransIntrEnable(CANFD0, txfifoStatus.putIdx, 1U);

    /* Add request for transmission. */
    DL_MCAN_TXBufAddReq(CANFD0, txfifoStatus.putIdx);
    if (txMsg.id == MCAN_HOST_ID) {
        while (frame_send_success == false) {
            ;
        }
    }
    return 0;
}
```

首先，通过调用 DL_MCAN_writeMsgRam() 将 TX 消息保存到 FIFO 类型的消息 RAM 中。然后，通过调用 DL_MCAN_getTxFIFOQueStatus() 获取该消息的索引，该索引指示已保存消息在 FIFO 中的位置。之后，通过调用 DL_MCAN_TXBufTransIntrEnable()，使用此消息的 put 索引信息启用传输中断。最后，通过调用 DL_MCAN_TXBufAddReq()，使用此消息的 put 索引信息调用传输用的添加请求。

3.3 RX 缓冲模式

在 SDK 中的当前演示项目中，将 RX 配置为 FIFO 模式。下一节介绍了如何在缓冲模式下使用 RX。

```
static const DL_MCAN_StdMsgIDFilterElement gMCAN0StdFiltelem = {
    .sfec = 0x7,
    .sft  = 0x0,
    .sfid1 = 3,
    .sfid2 = 0x0,
};
/* Configure Standard ID filter element */
DL_MCAN_addStdMsgIDFilter(MCAN0_INST, 0U, (DL_MCAN_StdMsgIDFilterElement *) &gMCAN0StdFiltelem);
```

当滤波器元素配置为 *Store into Rx buffer 或调试消息*时，将启用 RX 缓冲区模式。首先，添加一个 .sfec 为 0x7 的滤波器。将 .sfec 设置为 0x7 时，.sft 无关紧要。sfid1 是滤波器 ID，接收消息 ID 必须与该 ID 相匹配才能接收。sfid2 能配置使用哪个 Rx 缓冲器来存储已接收的消息。

```
/* New message received by Rx Buffer (Filter matched) */
if(gInterruptLineStatus & DL_MCAN_INTR_SRC_DEDICATED_RX_BUFF_MSG){
    gInterruptLine1Status |= DL_MCAN_INTR_SRC_DEDICATED_RX_BUFF_MSG;
    DL_MCAN_getNewDataStatus(MCAN0_INST, &newDataStatus);

    if(newDataStatus.statusLow) {
        /* Check Rx Buffer0 status */
        if(newDataStatus.statusLow & 0x1){
            DL_MCAN_readMsgRam(MCAN0_INST, DL_MCAN_MEM_TYPE_BUF, MCAN_RX_MSG_BUFFER_INDEX, 0U,
&rxMsg);
            ProcessCanRxMsg(&rxMsg);
        }
        /* Check remaining Rx Buffer */
    }
    if(newDataStatus.statusHigh) {
        ; /* 32-63, not applicable in demo */
    }
    /* Clear all new data status */
    DL_MCAN_clearNewDataStatus(MCAN0_INST, &newDataStatus);
}
```

当接收到一条与滤波器配置匹配的消息时，该 RX 消息被存储到 RX 缓冲区 0 到 63 的范围。首先，获取新的数据状态。如果 RX 缓冲区 0 至 RX 缓冲区 31 中接收到新消息，则 newDataStatus.statusLow 的相应位将设置为 1。如果 RX 缓冲区 32 到 RX 缓冲区 63 中接收到新消息，则 newDataStatus.statusHigh 的相应位将设置为 1。然后，检查 RX 缓冲区 0 中是否接收到新消息。如果是，则使用 MCAN_RX_MSG_BUFFER_INDEX 值调用 DL_MCAN_readMsgRam() 来读取 RX 消息。在本例中，RX 缓冲区 0 的索引值为 0。最后，在处理消息后，调用 DL_MCAN_clearNewDataStatus() 清除新数据状态。

3.4 RX FIFO 模式

在 SDK 中的当前演示项目中，将 RX 配置为 FIFO 模式。下节介绍了 DL_MCAN_RxFIFOStatus 的结构，该结构通常用于 RX FIFO 模式。TI 建议用户在从 RX FIFO 提取消息时检查此结构。

```
/**
 *  @brief  Structure for MCAN Rx FIFO Status.
 */
typedef struct {
    /*! Rx FIFO number
     *   One of @ref DL_MCAN_RX_FIFO_NUM
     */
    uint32_t num;
    /*! Rx FIFO Fill Level */
    uint32_t fillLvl;
    /*! Rx FIFO Get Index */
    uint32_t getIdx;
    /*! Rx FIFO Put Index */
    uint32_t putIdx;
    /*! Rx FIFO Full
     *   0 = Rx FIFO not full
     *   1 = Rx FIFO full
     */
    uint32_t fifoFull;
    /*! Rx FIFO Message Lost */
    uint32_t msgLost;
} DL_MCAN_RxFIFOStatus;
```

- *num* 指示当前操作的 Rx FIFO 实例编号。MCAN 可以支持多个 Rx FIFO（如 Rx FIFO 0 和 Rx FIFO 1），必须指定具体实例。例如，DL_MCAN_RX_FIFO_NUM 枚举包含 DL_MCAN_RX_FIFO_0 和 DL_MCAN_RX_FIFO_1。
- fillLvl 指示当前存储在 Rx FIFO 中的有效消息数量。例如，当 fillLvl 为 5 时，这意味着 FIFO 中有 5 条未读消息。当 fillLvl 达到 Rx FIFO 的深度（例如 6 条消息）时，fifoFull 字段设置为 1，表示 FIFO 已满。
- getIdx 指向下一条要读取的消息。CPU 通过递增 getIdx，按顺序读取 FIFO 中的消息。当 Rx FIFO 已满且处于覆盖模式时，新消息会覆盖最早的消息。此时，用户开始从 getIdx + 1 读取，以避免读取正在被覆盖的旧数据。
- putIdx 指向下一个可写消息的位置。接收到新消息时，该消息会写入 putIdx 指向的位置，然后 putIdx 递增。
- fifoFull 表示 Rx FIFO 是否已满。当 Rx FIFO 已满时，请尝试以下选项：
  - 阻塞模式：将丢弃新消息并触发 msgLost 计数。
  - 覆盖模式：新消息会覆盖最早的消息，putIdx 和 getIdx 同时递增。
- msgLost 统计由于 Rx FIFO 已满而丢弃的消息数量。在下列情况下会触发此举：
  - 阻塞模式：当 Rx FIFO 已满时，将接收到一条新的消息，而该消息将被拒绝和丢弃。
  - 覆盖模式：覆盖旧消息时，被覆盖的消息将计为丢失。

4 解决/避免 CAN 通信问题的调试和设计提示

本节说明了实现 CAN 总线时的一些常见错误和疏忽。随后是一些有助于总线问题故障排除的调试提示。

4.1 所需的最少节点数

除非在自检模式下工作，否则 CAN 总线上至少需要两个节点，原因如下。当节点在 CAN 总线上传输帧时，该节点需要网络上至少一个其他节点的确认 (ACK)。只要 CAN 节点成功接收到消息，节点就会自动发送 ACK，除非该功能已关闭（*静默模式*。静默节点是节点接收帧但不提供 ACK 的位置；MCAN 中的总线监控模式）。提供 ACK 的节点不需要是帧的预期接收者，尽管可能会发生这种情况。（总线上的所有活动节点都提供 ACK，无论节点是否是该帧的预期接收者都是如此）。

当发送节点没有接收到 ACK 时，这会导致一个 ACK 错误并且发送节点一直在将帧永远重新传输出去。发送错误计数器 (TEC) 递增至 128 并在此处停止。REC 保持为 0。节点不会脱离 bus-off。也不会生成中断。如果另一个节点进入网络，则 TEC 在每次成功传输时开始递减（一直递减到 0）。

4.2 为何需要收发器

用户无法将节点 A 的 MCAN_TX 直接连接到节点 B 的 MCAN_RX，反之亦然，并期望 CAN 通信成功。在这种情况下，CAN 与 UART 或 SPI 等其他串行接口不同。例如，UART 可以与 RS232 收发器搭配使用，也可以通过直接连接（一个节点的 UART_TX 到另一个节点的 UART_RX，反之亦然）来使用。然而，出于以下原因，CAN 总线需要一个 CAN 收发器。除了转换单端 CAN 信号以进行差分传输之外，收发器还将 CAN_TX 引脚环回到节点的 CAN_RX 引脚。这是因为 CAN 节点需要能够监控传输。

- 这与 CAN 协议规定的 ACK 要求有关。当节点在 CAN 总线上传输帧时，该节点需要网络上至少一个其他节点的 ACK。在 ACK 阶段，发送器输出 1 并期望读回 0。
- 在调停期间，具有较高优先级消息 ID 的节点需要能够将 1 覆盖为 0。在这里，发送器需要读回传输的数据。当节点在调停阶段输出 1 并回读 0 时，该节点将失去调停。

为了节省收发器的成本，某些应用（所有节点都位于同一 PCB 上且靠近）使用二极管等分立式元件来满足 CAN 节点监控传输的要求。

4.3 总线关闭状态

当 CAN 节点由于严重的总线错误而被强制隔离时，会出现总线关闭状态。在该状态下，节点停止发送和接收数据，从而有效地与总线断开连接。该机制可保护总线免受持续错误的影响并防止故障传播。

主要触发是指当 TEC 超过阈值（通常为 255）时引起的触发。常见场景包括：

- 硬件故障：CAN 控制器或收发器损坏。
- 物理层问题：开路或短路、端接电阻器不正确或信号干扰（例如强 EMI）。
- 配置错误：波特率不匹配，导致调停失败或位错误。
- 软件错误：重复传输格式错误的帧。

例如，每个传输误差都会使 TEC 递增 1。如果误差快速累积且 TEC 超过 255，则节点进入总线关断状态。

该器件可以通过执行以下操作之一从总线关闭状态中恢复：

- 硬件复位：复位 CAN 控制器（通过软件复位或下电上电）以清除错误计数器。
- 接收恢复序列：检测 129 个连续 11 个隐性位序列（总线空闲信号），从而自动将节点恢复为*活动错误*状态。这需要足够的总线空闲时间（总持续时间= 129 × 11 位）。

下述示例说明了如何首先在 SysConfig 中启用中断 Bus_Off 状态来检测总线关闭状态。当总线关闭状态发生变化时，MCAN 会触发中断 Bus_Off 状态以将这种情况通知用户。用户需要检查总线关闭状态是否在中断例程中。

```
/**
 * CAN protocol status
 *      Bus off
 */
DL_MCAN_ProtocolStatus gProtStatus;
volatile uint8_t CANBusOff = 0;
void MCAN0_INST_IRQHandler(void)
{
    switch (DL_MCAN_getPendingInterrupt(MCAN0_INST)) {
        case DL_MCAN_IIDX_LINE0:
            break;
        case DL_MCAN_IIDX_LINE1:
            /* MCAN bus off status changed */
            if(gInterruptLine1Status&DL_MCAN_INTERRUPT_BO) {
                DL_MCAN_getProtocolStatus(MCAN0_INST, &gProtStatus);
                if(gProtStatus.busOffStatus == 1) {
                    CANBusOff = true;
                }
                else {
                    CANBusOff = false;
                }
            }
            /* Clear all MCAN interrupt status */
            DL_MCAN_clearIntrStatus(MCAN0_INST, gInterruptLine1Status,
DL_MCAN_INTR_SRC_MCAN_LINE_1);
            break;
        default:
            break;
    }
}
```

然后，检查主环路中的 CANBusOff 标志。检测到总线关闭状态时，重新启动 MCAN 模块。

```
main loop:
{
    if(CANBusOff == 1) {
        /* Re-start MCAN */
        MCAN0_Restart();
        CANBusOff = 0;
    }
}
/*MCAN restart function*/
void MCAN0_Restart(void)
{
    DL_MCAN_reset(CANFD0);
    delay_cycles(16);
    DL_MCAN_disablePower(CANFD0);
    delay_cycles(32);
    DL_MCAN_enablePower(CANFD0);
    // MCAN RAM need at least 50us to finish init
    // 1600 CPU cycles@CPU32MHz
    // 4000 CPU cycles@CPU80MHz
    delay_cycles(4000);
    SYSCFG_DL_MCAN0_init();
}
```

4.4 在低功耗模式下使用 MCAN

用户需要将 MCU 置于低功耗模式，以满足应用要求。然而，在低功耗模式下禁用 MCAN。作为权变措施，用户可以在进入低功耗模式之前将 MCAN RX 引脚配置为输入引脚。在该引脚上启用边沿失败中断。当在 MCAN RX 引脚上接收到一条消息时，MCU 由边沿失败中断唤醒。然后，将 MCAN RX 引脚重新配置为 MCAN 功能并重新配置 MCAN 模块。MCAN 以这种方式恢复正常功能。

备注

第一条 MCAN 消息用作 MCU 的唤醒信号。对该唤醒功能使用测试消息。

下方显示了示例代码。

```
void MCAN_LowPowerMode(void)
{
    DL_GPIO_initDigitalInputFeatures(GPIO_MCAN0_IOMUX_CAN_RX,
         DL_GPIO_INVERSION_DISABLE, DL_GPIO_RESISTOR_PULL_UP,
         DL_GPIO_HYSTERESIS_DISABLE, DL_GPIO_WAKEUP_DISABLE);
    DL_GPIO_setLowerPinsPolarity(GPIO_MCAN0_CAN_RX_PORT, DL_GPIO_PIN_13_EDGE_FALL);
    DL_GPIO_clearInterruptStatus(GPIO_MCAN0_CAN_RX_PORT, GPIO_MCAN0_CAN_RX_PIN);
    DL_GPIO_enableInterrupt(GPIO_MCAN0_CAN_RX_PORT, GPIO_MCAN0_CAN_RX_PIN);

    DL_SYSCTL_setPowerPolicySTANDBY0();
    __WFI();
    DL_SYSCTL_setPowerPolicyRUN0SLEEP0();

    DL_GPIO_initPeripheralInputFunction(
        GPIO_MCAN0_IOMUX_CAN_RX, GPIO_MCAN0_IOMUX_CAN_RX_FUNC);

    MCAN0_Restart();
}
/*MCAN restart function*/
void MCAN0_Restart(void)
{
    DL_MCAN_reset(CANFD0);
    delay_cycles(16);
    DL_MCAN_disablePower(CANFD0);
    delay_cycles(32);
    DL_MCAN_enablePower(CANFD0);
    // MCAN RAM need at least 50us to finish init
    // 1600 CPU cycles@CPU32MHz
    // 4000 CPU cycles@CPU80MHz
    delay_cycles(4000);
    SYSCFG_DL_MCAN0_init();
}
```

4.5 调试检查清单

本节重点介绍一些常见错误并提供有用的调试技巧。

4.5.1 编程问题

- 是否启用了 MCAN 模块的时钟？查看 SysConfig 中的 CANCLK 配置。TI 建议使用外部晶体作为 CANCLK 源，降低误差率。
- 首先尝试不使用中断的代码。请改用轮询。一旦轮询工作正常，用户可以在稍后添加中断。
- 当尝试首次在总线上发起通信时，请确保使用相同的消息 ID 对发送节点和接收节点中的邮箱进行编程。最初请勿使用滤波器功能。如果未确认硬件问题，则可以稍后添加滤波。

4.5.2 物理层问题

- 总线是否仅在两端正确端接（使用 120Ω）？只能在两端使用 120Ω 电阻器时端接总线。总线上不能有超过两个终端电阻器，除非遵循分裂终端原则，在这种情况下，其中两端都有两个电阻器。在设计 CAN 总线系统时，可以从系统外壳外部启用或禁用端接电阻器。如必须在网络中添加或删除节点，这种方案很容易实现。
- 是否所有 CAN 节点都配置为相同的比特率？不匹配的节点比特率会在总线上重复引入错误帧。捕获示波器上 CAN_TX 引脚的输出，以物理验证位时间。
- 用户是否尝试过更低的比特率？例如，50kbps。当尝试使用较低的比特率时，可能会遇到与传播延迟有关的时序问题。确保在 SysConfig 中正确配置位时序参数。
- 用户是否尝试过缩短总线长度和减少节点数量？
- 在错误条件出现前，总线上是否有错误帧？这可能是时序违规或噪声问题。
- 总线中有多少个节点？（在非自检模式下，由于 CAN 协议规定的确认 (ACK) 要求，网络上必须至少有两个节点）。

4.5.3 硬件调试提示

- 要查看直至 ACK 阶段的波形，必须将收发器连接到节点。如果没有收发器，该节点立即进入错误状态。
- 检查是否能在发送 MCU 的 MCAN_TX 引脚上合适地看到 CAN 帧且 CAN 帧是否具备预期比特率。如果在 MCAN_TX 引脚上看到了预期数据，则检查 MCAN_RX 引脚上的数据。如果在 MCAN_RX 引脚上看到相同的数据，则收发器能正确地环回数据。
- 如果使用具有内置 CAN FD 触发器的示波器，请确保配置为触发的信号与在电路板上探测的信号相匹配。除了 Start-of_Frame (SOF)、远程帧、错误帧和特定消息 ID，许多示波器还能够触发 CAN 发送 (CANTX)、CAN 接收 (CANRX)、CAN_H 和 CAN_L 信号。
- 如果示波器没有解码波形，请确保通道的输入阈值正确。这与通常用于信号中的*触发电平*类似。
- 确保在示波器中正确配置标称阶段和数据阶段的比特率。否则，这会显示不正确的数据。
- CAN 总线分析器工具：确保正确配置标称阶段和数据阶段的比特率。

5 总结

本文档首先概述了 CAN 总线和 MSPM0 MCU 中集成的 MCAN 模块的特性。应用手册详细介绍了如何使用 SysConfig 配置 CAN 模块，以便用户能够调整相关参数以满足应用要求。本文档简要介绍了 MSPM0 SDK 中提供的与 MCAN 模块相关的演示示例。最后，介绍了一些常见的调试问题以及软件和硬件方面的相关调试建议。

6 参考资料

- 德州仪器 (TI)，*MSPM0 G 系列 80MHz 微控制器*，技术参考手册
- 德州仪器 (TI)，*MSPM0-SDK*
- 德州仪器 (TI)，*控制器局域网 (CAN) 简介*，应用报告
- 德州仪器 (TI)，*控制器局域网物理层要求*，应用报告
- 德州仪器 (TI)，*控制器局域网 (CAN) 物理层调试基础知识*，模拟设计期刊
- 德州仪器 (TI)，*CAN 位时序参数计算器*，应用手册
- 德州仪器 (TI)，*3.3V CAN（控制器局域网）收发器概述*，应用手册
- 德州仪器 (TI)，*使用无扼流圈收发器简化 CAN 总线实施*，营销白皮书
- 德州仪器 (TI)，*CAN 总线连接的关键间距*，应用手册
- 德州仪器 (TI)，*CAN 总线上的消息优先级反转*，模拟设计期刊

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。 严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265