Application Note

在 TI AM13E2x 上运行 DroneCAN 应用程序

摘要

DroneCAN 是一种轻量级开放式通信协议，广泛用于 UAV 和机器人系统，用于通过 CAN 总线在节点之间实现可靠的数据交换。本应用手册提供了在 TI AM13E230x 微控制器上使用片上 MCAN 外设和 Libcanard 协议栈实施 DroneCAN 的实用指南。本文档介绍了硬件设置、软件集成、一个实用示例以及常见问题，以帮助开发人员在 AM13x 系列器件上构建支持 DroneCAN 的节点。本应用手册适用于所有具有相同 MCAN 外设的 TI MCU。

内容

1 简介.....2
1.1 DroneCAN：协议概述及主要特性.....2
1.2 AM13E230x 概述：.....2
2 系统架构.....4
2.1 软件栈概述.....4
2.2 Libcanard-MCAN 接口层：TX 和 RX 流程.....5
3 软件应用程序概述.....6
3.1 工程结构.....6
3.2 SysConfig 外设配置.....6
3.3 内存池分配.....9
4 硬件设置.....10
5 运行示例和测试结果.....11
6 关键陷阱.....17
7 结语.....20
8 Tools and Software.....21

商标

1 简介

1.1 DroneCAN：协议概述及主要特性

DroneCAN 是一种构建在 CAN 总线之上的轻量级开源通信协议，最初为 UAV 系统开发，现已广泛应用于机器人、工业和航空航天领域。DroneCAN 定义了标准化的消息格式、针对超过 8 字节有效载荷的传输分段、节点 ID 管理以及数据类型签名，以在来自不同供应商的节点之间实现互操作性。Libcanard 是 DroneCAN 协议栈的官方 C 语言实现，专为资源受限的嵌入式系统而设计，所需 RAM 极少，并且除了用户提供的内存池之外，不依赖于动态内存分配器。Libcanard 完全用 C 语言编写，可无缝集成到现有的嵌入式工程和 TI MCU SDK 示例中，无需任何操作系统或中间件层。该协议栈提供了简单的 API，涵盖 DroneCAN 传输编码、发送、接收和解码的整个生命周期。对于已经熟悉基于 CAN 的通信的开发人员而言，这种设计很容易采用。

DroneCAN ID 格式： 图 1-1 显示了 DroneCAN 广播消息的 29 位 ID 结构。

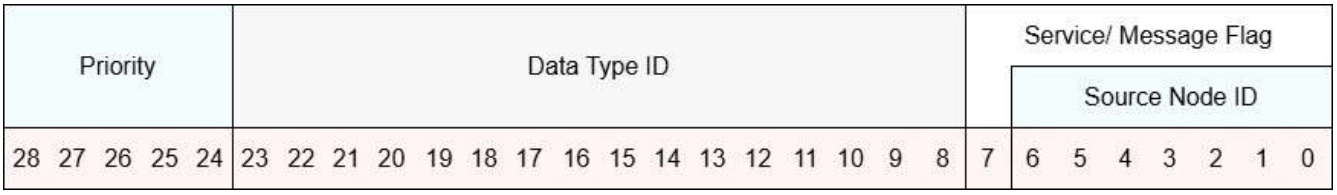

图 1-1. CAN ID 格式

普通的 CAN 帧 ID 通常是 11 位，但 DroneCAN 使用扩展帧 ID (EFF)。在这里，帧具有自描述性，接收工具无需事先了解应用场景即可解码总线上的任何帧，这使得 DroneCAN 的集成变得更加容易。

图 1-2 展示了使用 DroneCAN 进行通信的无人机系统示例。

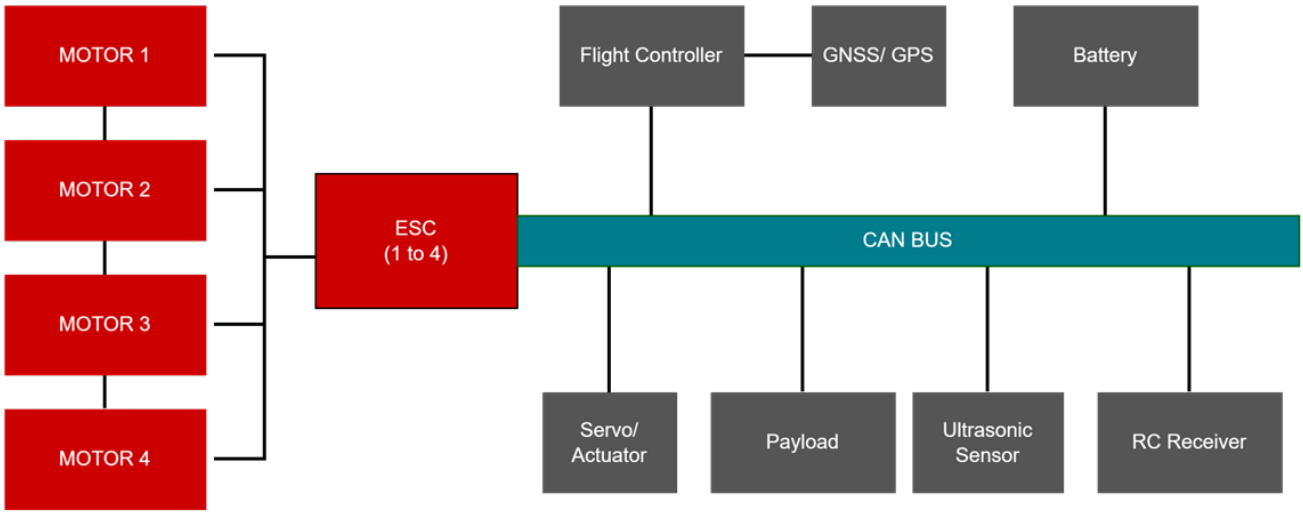

图 1-2. 采用 DroneCAN 的无人机系统

在此，DroneCAN 主要用于飞行控制器（大脑）与电子调速器 (ESC) 之间的通信，后者进一步控制电机的速度和旋转。单条 CAN 总线即可处理不同节点之间的所有通信，而不会出现任何重叠问题。CAN 仅需两根导线即可连接总线上的每个节点，并且该总线完全由硬件处理所有故障检测、帧重传以及其他各种错误，无需任何软件介入。

1.2 AM13E230x 概述：

AM13E230x 是一款基于 32 位 Arm Cortex-M33 的微控制器，专为需要可靠现场通信且对成本敏感的工业和汽车应用而设计。该器件集成了 MCAN 外设，主频高达 200Mhz，支持经典 CAN 和 CAN FD，并具有可配置的波特率。MCAN 外设采用灵活的消息 RAM 架构，支持专用的 TX 缓冲区、RX FIFO 和硬件接受过滤器，能够以最少的 CPU 开销实现高效的多帧传输处理。AM13E230x 支持 SysConfig，允许通过图形界面进行完整的外设配置，

包括位时序、消息 RAM 配置、GPIO 引脚分配、中断路由和 CAN 模式配置，从而显著缩短新用户的启动时间。集成的 CAN 硬件、基于 SysConfig 的配置以及 DriverLib API 支持相结合，使 AM13E230x 成为一个实用且功能强大的 DroneCAN 集成平台。

1.2.1 为什么选择将 AM13E230x 用于 DroneCAN

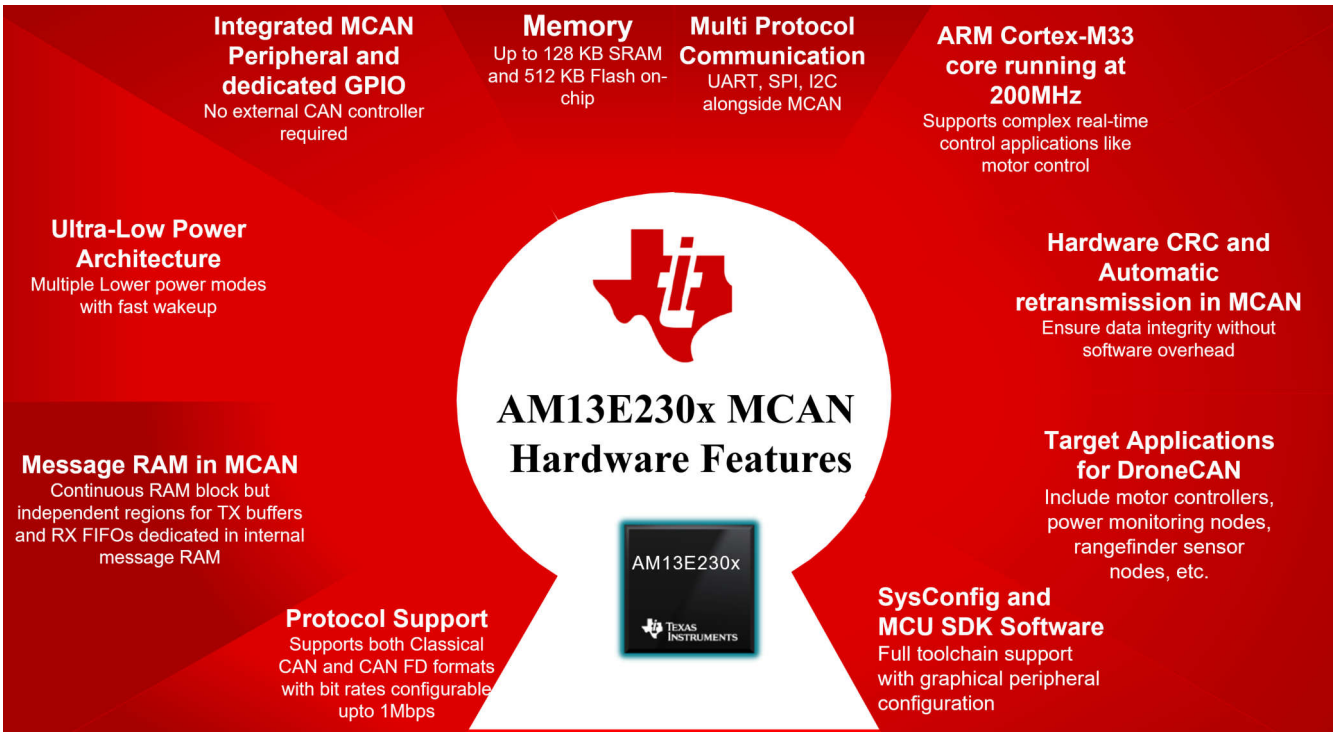

图 1-3. MCAN 硬件特性

- AM13E230x MCAN 外设集成在片上，具有专用 GPIO，因此无需外部 CAN 控制器。
- 专用的 CAN 内核（控制器）可确保 CAN 通信不会占用太多 CPU 带宽，因此即使在运行 DroneCAN 时，该器件也能提供非常高效的电机控制。
- DroneCAN 需要正确传输每个帧，而 MCAN 外设可以自动重传遇到错误的帧，无需任何软件介入。
- MCAN 具有内部消息 RAM，为 TX 缓冲区、RX FIFO 和硬件接受过滤器划分了独立的区域。因此，所有传入的帧在得到处理之前都会经过存储和过滤，当总线繁忙且有多种节点类型同时发送时，这种方式非常高效。
- 寄存器空间将 Tx 和 Rx 传输处理程序与实际硬件 FIFO 和消息 RAM 连接起来。所有寄存器访问都通过 TI Driverlib 进行抽象。
- 同时支持 CAN 和 CAN FD，因此未来可由同一硬件支持更高吞吐量的应用，而无需更改硬件板卡。
- 该设计是一种超低功耗架构。多种低功耗模式都具有快速唤醒功能，使大部分时间处于空闲状态的节点能快速响应。
- 整个 MCAN 外设、位时序、消息 RAM 配置、GPIO 引脚分配、中断路由、计时器和 CAN 模式配置均通过 Syscfg 进行配置，从而减少了终端开发人员的开销。
- 除 MCAN 外，还提供了 UART、SPI 和 I2C 通信协议，使器件能够同时与其他组件进行通信。

2 系统架构

Libcanard 处理所有涉及 DroneCAN 协议的事务，但完全不知道 TI MCAN 外设的情况。Libcanard 管理 Canard 帧的编码、解码和处理，但不能在总线上物理传输这些帧。同时，TI MCAN 外设可以处理 CAN 帧的发送和接收，但不知道 Canard ID 结构或属性。为了弥合这一差距，需要一个**专用的接口层**，使开发人员无需直接在应用程序中编写外设交互代码，即可在 AM13x 器件上构建 DroneCAN 节点。

2.1 软件栈概述

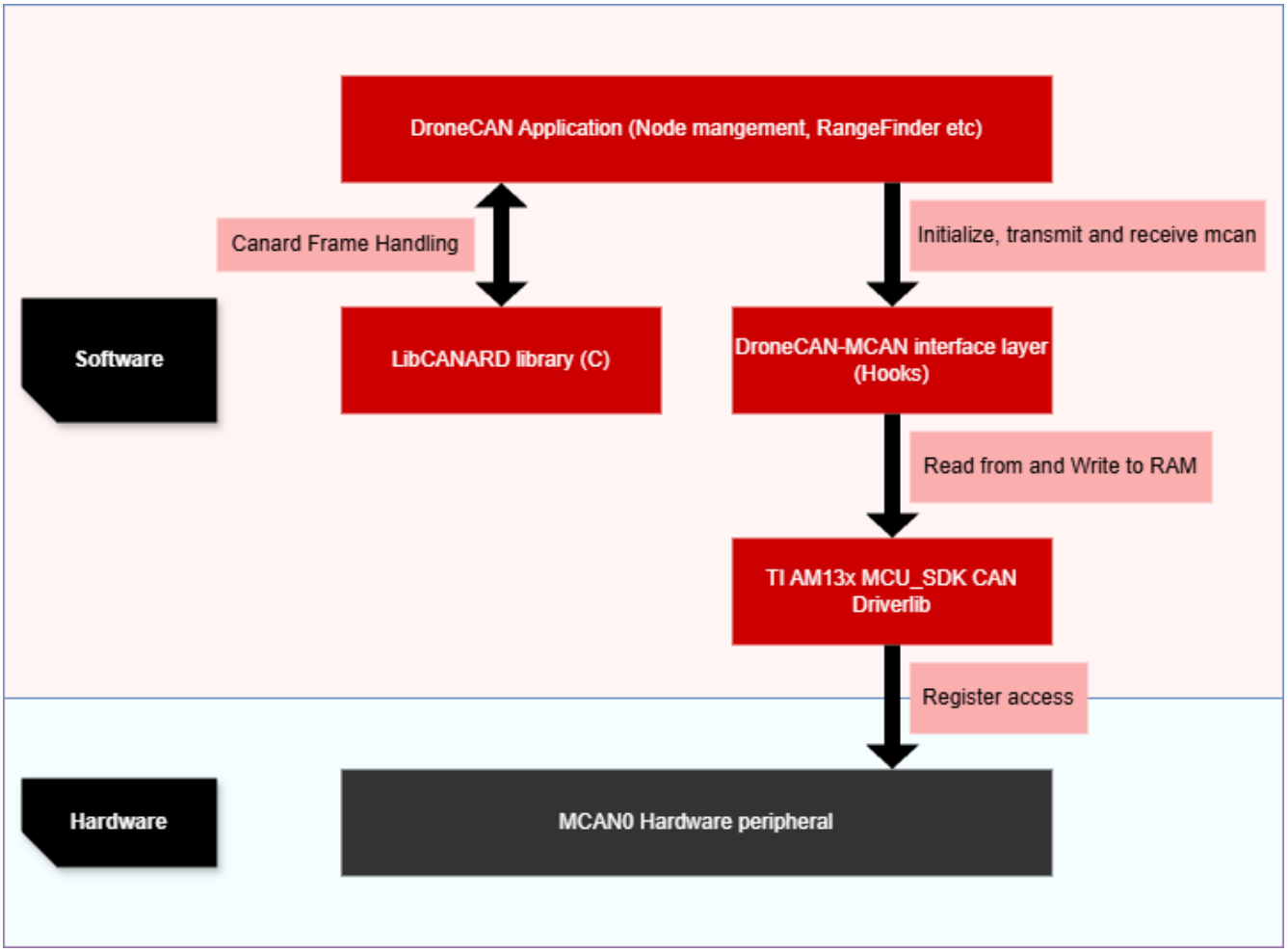

图 2-1. 软件应用程序结构

- **DroneCAN 应用程序**位于顶部，驱动整个系统。该应用程序初始化协议栈，编码和解码传输，管理节点，并执行周期性维护。
- **Libcanard** 是应用程序直接与之交互的协议库。该协议库处理所有 DroneCAN 协议职责，包括帧编码、解码、传输重组、CRC 和内存管理，无需了解硬件知识。
- **DroneCAN-MCAN 接口层**是将应用程序连接到 TI 外设的硬件抽象层。
  - 该应用程序将 Libcanard 帧结构转换为 MCAN 消息 RAM 格式，并执行收发器控制、过滤器编程和模式转换。
  - 所有特定于硬件的复杂性都包含在此层中，使得应用程序和 Libcanard 完全不受任何特定外设代码的影响。
- **MCAN0 硬件外设**是 AM13E230x 上用于驱动 CAN 总线的物理 CAN 控制器。

2.2 Libcanard-MCAN 接口层：TX 和 RX 流程

该接口层主要提供一些 API，处理 MCAN 帧格式与 Libcanard 帧格式的相互转换、初始化、过滤器配置、发送和接收。这可以在任何具有类似 MCAN 外设的器件的 DroneCAN 工程中直接复用。下面给出了集成 Libcanard 的应用程序中发送和接收数据包的大致流程。

2.2.1 TX 流程

图 2-2 展示了传输帧的典型流程。

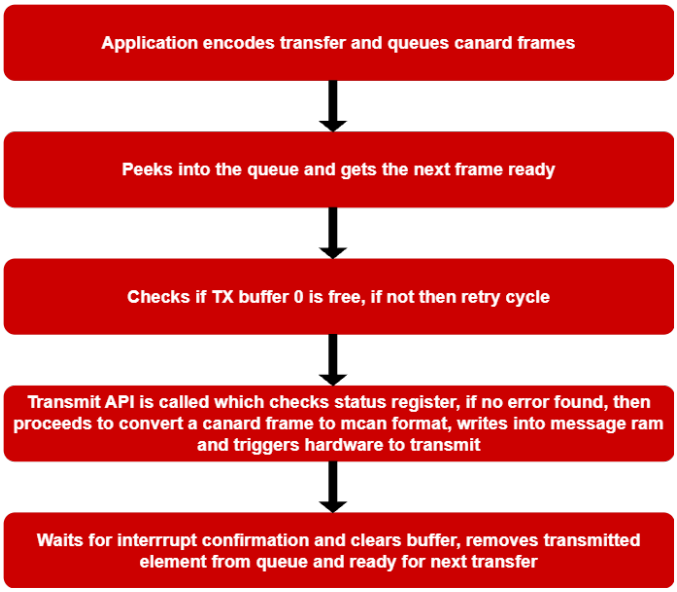

图 2-2. 帧传输流程

2.2.2 RX 流程

图 2-3 展示了接收帧的典型流程。

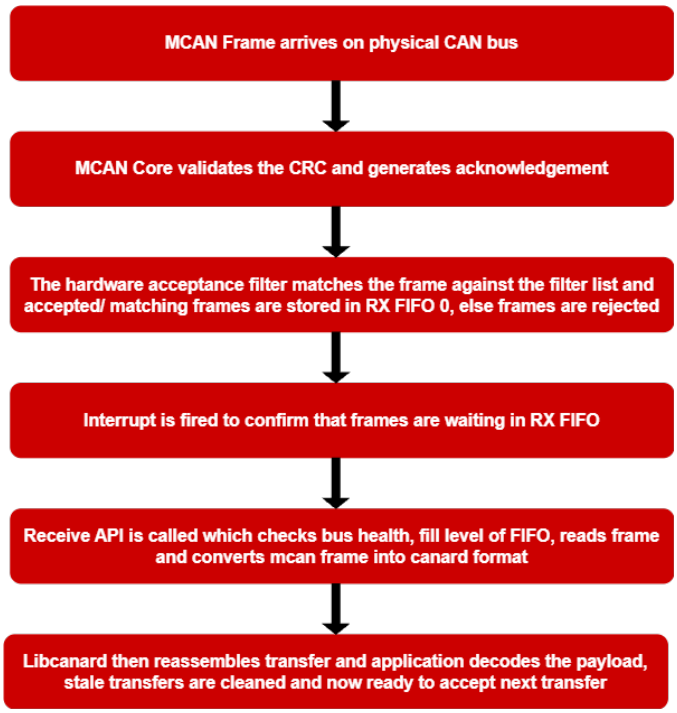

图 2-3. 帧接收流程

3 软件应用程序概述

在 AM13E230x Launchpad 上实现的示例 Libcanard 应用程序可按需提供。请访问 E2E™ 论坛或通过您的 TI 销售或现场联系人获取该应用程序的访问权限。

3.1 工程结构

下面讨论了 CCS DroneCAN 工程结构的一个示例：

- 根目录下的 example.syscfg 文件是硬件配置的单一位置。在 CCS SysConfig 编辑器中打开该文件以修改任何外设设置。
- Libcanard 目录包含未修改的 libcanard 源文件和接口文件 canard_am13x（位于 drivers 目录内）。这些文件会导入工程，无需任何更改。
- 应用程序常量，包括节点 ID、内存池大小、数据类型 ID、计时器实例和时钟速率，都集中在 app_cfg.h 中。根据需要修改这些详细信息。
- 应用程序级的 DroneCAN 帧编码和解码逻辑位于 app_mcan.c 中，main.c 仅作为入口点。

```
example_root_folder
│
├── example.syscfg              ← Hardware configuration. Open this
│                                 in CCS SysConfig editor to change
│                                 MCAN settings, GPIO, timers.
│
├── am13e230x_lp/
│   └── app_cfg.h               ← Application constants: node ID,
│                                 memory pool size, DTID values,
│                                 timer instance, tick rate, etc.
│
├── libcanard/                  ← Migrate the entire libcanard
│   ├── canard.c                   directory into the project root.
│   ├── canard.h                   Do not modify these files.
│   ├── canard_internals.h
│   └── drivers/
│       ├── canard_am13x.c      ← TI driver layer implementation.
│       └── canard_am13x.h      ← TI driver layer header.
│
├── app_mcan.c                  ← Application layer. Your DroneCAN
│                                 encode/decode logic lives here.
│
├── main.c ← Entry point.
│
└── targetConfigs/              ← CCS debug configuration.

WARNING: Not Thread-Safe

This driver does not use IRQ or critical sections. It is safe to call the API functions from IRQ
context **provided the application verifies that no concurrent calls occur from thread and IRQ
context simultaneously**. The application is responsible for all guarding.
```

3.2 SysConfig 外设配置

所有外设配置均通过 SysConfig 进行管理。需要执行以下设置才能正确使用 DroneCAN：

MCAN 常规设置

- 取消选中“Messages Will Only Be Sent Once”，以便帧可以自动重传，这样错误帧就不会永久丢失。

消息 RAM 布局

- 标准过滤器列表：2 个元素，起始地址为 0
- 扩展过滤器列表：2 个元素，起始地址为 8
- TX 缓冲区：起始地址为 24，1 个元素
- TX 事件 FIFO：起始地址为 96
- RX FIFO 0：根据 TX 区域进行配置，设置元素计数、水位线和起始地址，且与 TX 区域无重叠

接受过滤

- 标准 ID 过滤器：经典位掩码，将匹配的帧路由到 RX FIFO 0

- 扩展 ID 过滤器：范围过滤器，EFID2 设置为 536870911 以覆盖完整的 29 位范围
- 接受非匹配扩展帧：设置为拒绝

中断

- 启用中断线 1
- RX FIFO 0 新消息、TX 完成、总线关闭、错误被动和警告状态中断源均路由到中断线 1

计时器

- 需要一个计时器外设实例来产生周期性中断。应用程序使用计时器外设来维护时间戳，该时间戳传递给处理接收到的帧和清理过期传输的函数。

作为参考，下面给出了所提及的 TX 和 RX 示例的 SysConfig 文件：

```
/**
 * These arguments were used when this file was generated. They will be automatically applied on
subsequent loads
 * through the GUI or CLI. Run CLI with '--help' for additional information on how to override
these arguments.
 * @cliArgs --device "AM13E230x" --part "AM13E230x" --package "LQFP64_G" --product "TI MCU
SDK@26.01.00"
 * @v2CliArgs --device "AM13E23019" --package "LQFP64_G" --product "TI MCU SDK@26.01.00"
 * @versions {"tool":"1.27.0+4565"}
 */

/**
 * Import the modules used in this configuration.
 */
const config = scripting.addModule("/drivers/config");
const gpio   = scripting.addModule("/drivers/gpio");
const gpio1  = gpio.addInstance();
const mcan   = scripting.addModule("/drivers/mcan", {}, false);
const mcan1  = mcan.addInstance();
const sysctl = scripting.addModule("/drivers/sysctl");
const timer  = scripting.addModule("/drivers/timer", {}, false);
const timer1 = timer.addInstance();
const uart   = scripting.addModule("/drivers/uart", {}, false);
const uart1  = uart.addInstance();

/**
 * Write custom configuration values to the imported modules.
 */
gpio1.$name                         = "GPIO_GRP_0";
gpio1.port                          = "PORTA";
gpio1.associatedPins[0].$name       = "TCAN_STB";
gpio1.associatedPins[0].pin.$assign = "PA24";

mcan1.$name                             = "MCAN";
mcan1.wkupReqEnable                     = true;
mcan1.emulationEnable                   = true;
mcan1.autoWkupEnable                    = true;
mcan1.tdcEnable                         = true;
mcan1.additionalCoreConfig              = true;
mcan1.rrfe                              = true;
mcan1.rrfs                              = true;
mcan1.txEventFIFOWaterMark              = 0;
mcan1.txEventFIFOSize                   = 0;
mcan1.txBufNum                          = 1;
mcan1.txBufElemSize                     = "DL_MCAN_ELEM_SIZE_8BYTES";
mcan1.nomTimeSeg1_manual                = 126;
mcan1.nomTimeSeg2_manual                = 31;
mcan1.nomSynchJumpWidth_manual          = 31;
mcan1.dataTimeSeg1_manual               = 14;
mcan1.dataTimeSeg2_manual               = 3;
mcan1.dataSynchJumpWidth_manual         = 3;
mcan1.spPercent                         = 80;
mcan1.desiredNomRate                    = 250;
mcan1.rxFIFO1startAddr                  = 0;
mcan1.rxFIFO1size                       = 0;
mcan1.rxFIFO1waterMark                  = 0;
mcan1.m0interrupts                      = ["DL_MCAN_INTERRUPT_LINE1"];
mcan1.registerInterrupt                 = true;
mcan1.enableInterrupt                   = true;
mcan1.interruptLine                     = ["DL_MCAN_INTR_LINE_NUM_1"];
mcan1.interruptLine1Flag                =
```

```
["DL_MCAN_INTR_SRC_BUS_OFF_STATUS","DL_MCAN_INTR_SRC_ERR_PASSIVE","DL_MCAN_INTR_SRC_PROTOCOL_ERR_ARB
","DL_MCAN_INTR_SRC_PROTOCOL_ERR_DATA","DL_MCAN_INTR_SRC_RX_FIFO0_FULL","DL_MCAN_INTR_SRC_RX_FIFO0_N
EW_MSG","DL_MCAN_INTR_SRC_RX_FIFO0_WATERMARK","DL_MCAN_INTR_SRC_TRANS_COMPLETE","DL_MCAN_INTR_SRC_WA
RNING_STATUS"];
mcan1.interruptFlags                    =
["DL_MCAN_INTR_SRC_BUS_OFF_STATUS","DL_MCAN_INTR_SRC_ERR_PASSIVE","DL_MCAN_INTR_SRC_PROTOCOL_ERR_ARB
","DL_MCAN_INTR_SRC_PROTOCOL_ERR_DATA","DL_MCAN_INTR_SRC_RX_FIFO0_FULL","DL_MCAN_INTR_SRC_RX_FIFO0_N
EW_MSG","DL_MCAN_INTR_SRC_RX_FIFO0_WATERMARK","DL_MCAN_INTR_SRC_TRANS_COMPLETE","DL_MCAN_INTR_SRC_WA
RNING_STATUS"];
mcan1.lss                               = 2;
mcan1.lse                               = 2;
mcan1.flesa                             = 8;
mcan1.txStartAddr                       = 24;
mcan1.txEventFIFOStartAddr              = 96;
mcan1.rxBufStartAddr                    = 96;
mcan1.rxFIFO0startAddr                  = 112;
mcan1.rxFIFO0size                       = 10;
mcan1.rxFIFO0waterMark                  = 8;
mcan1.stdFiltType                       = "10";
mcan1.stdFiltElem                       = "001";
mcan1.extFiltElem                       = "001";
mcan1.extFiltType                       = "00";
mcan1.extFiltID2                        = 536870911;
mcan1.stdFiltID1                        = 3;
mcan1.stdFiltID2                        = 4;
mcan1.peripheral.$assign                = "MCAN0";
mcan1.peripheral.txPin.$assign          = "PA12";
mcan1.peripheral.rxPin.$assign          = "PA11";
mcan1.txPinConfig.direction             = scripting.forceWrite("OUTPUT");
mcan1.txPinConfig.hideOutputInversion   = scripting.forceWrite(false);
mcan1.txPinConfig.onlyInternalResistor  = scripting.forceWrite(false);
mcan1.txPinConfig.passedPeripheralType  = scripting.forceWrite("Digital");
mcan1.txPinConfig.$name                 = "ti_driverlib_gpio_GPIOPinGeneric0";
mcan1.rxPinConfig.hideOutputInversion   = scripting.forceWrite(false);
mcan1.rxPinConfig.onlyInternalResistor  = scripting.forceWrite(false);
mcan1.rxPinConfig.passedPeripheralType  = scripting.forceWrite("Digital");
mcan1.rxPinConfig.$name                 = "ti_driverlib_gpio_GPIOPinGeneric1";

sysctl.useHFCLK     = true;
sysctl.SYSPLLSource = "DL_SYSCTL_SYSPLL_REF_HFCLK";
sysctl.SYSPLL_Qdiv  = 32;

timer1.$name              = "APP_TIMER_0";
timer1.registerInterrupt  = true;
timer1.timerClkDiv        = "DL_TIMER_CLOCK_DIVIDE_8";
timer1.timerClkPrescale   = 256;
timer1.timerMode          = "DL_TIMER_TIMER_MODE_PERIODIC";
timer1.interrupts         = ["DL_TIMER_INTERRUPT_ZERO_EVENT"];
timer1.timerPeriod        = "1000 ms";

uart1.$name                             = "APP_MCANTX_LOGUART_0";
uart1.targetBaudRate                    = 115200;
uart1.peripheral.$assign                = "UC4";
uart1.peripheral.rxPin.$assign          = "PA1";
uart1.peripheral.txPin.$assign          = "PA0";
uart1.txPinConfig.direction             = scripting.forceWrite("OUTPUT");
uart1.txPinConfig.hideOutputInversion   = scripting.forceWrite(false);
uart1.txPinConfig.onlyInternalResistor  = scripting.forceWrite(false);
uart1.txPinConfig.passedPeripheralType  = scripting.forceWrite("Digital");
uart1.txPinConfig.$name                 = "drivers_gpio_v0_gpioPinConfig0";
uart1.rxPinConfig.hideOutputInversion   = scripting.forceWrite(false);
uart1.rxPinConfig.onlyInternalResistor  = scripting.forceWrite(false);
uart1.rxPinConfig.passedPeripheralType  = scripting.forceWrite("Digital");
uart1.rxPinConfig.$name                 = "drivers_gpio_v0_gpioPinConfig1";

/**
 * Pinmux option for unlocked pins/peripherals. This means that minor changes to the automatic
solver in a future
 * version of the tool will not impact the pinmux you originally saw.  These lines can be
completely deleted in order to
 * re-solve from scratch.
 */
sysctl.peripheral.$suggestSolution       = "SYSCTL";
sysctl.peripheral.x1Pin.$suggestSolution = "PC16_X1";
sysctl.peripheral.x2Pin.$suggestSolution = "PC17_X2";
timer1.peripheral.$suggestSolution       = "TIMG4_0";
```

3.3 内存池分配

可通过应用程序来定义和配置内存池。为此，在连接器命令文件中包含了以下模块：

```
SECTIONS
{
    .canard_pool : ALIGN(8),
                   RUN_START(__CANARD_POOL_START),
                   RUN_END(__CANARD_POOL_END)
                   > RAM_S
}
```

内存占用：

实现的示例应用程序在调试版本中约占 37KB 内存，在发布版本中约占 35KB 内存，具有进一步优化的空间。这意味着，保留了足够的内存用于在 M33 内核上运行其他应用程序，或在 TinyEngine™ 上运行 AI 模型。

4 硬件设置

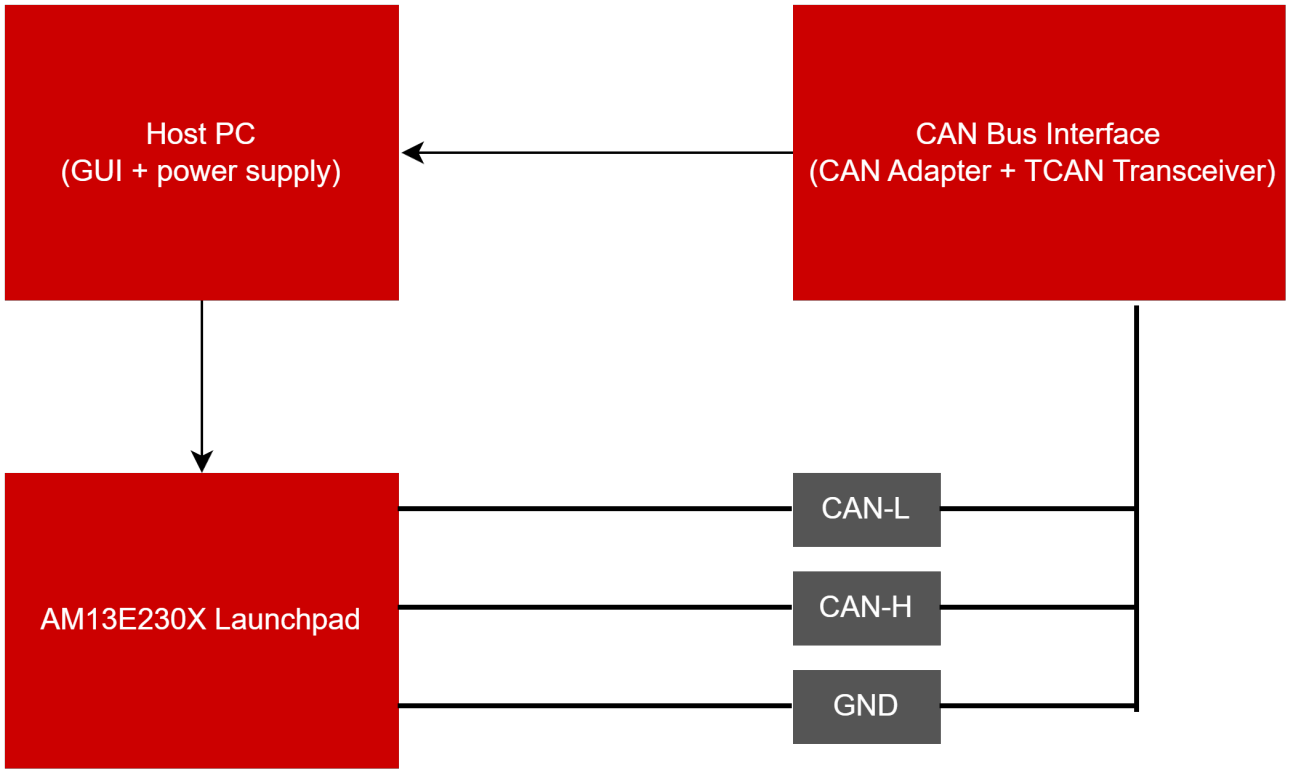

图 4-1. 硬件设置

硬件连接：

- **主机工作站至 AM13E230X**：为 AM13E230X 芯片供电并用作主控制接口。
- **AM13E230X 至 CAN 总线接口（或 CAN 总线接口至 AM13E230X）**三线 CAN 连接：**CAN-H**、**CAN-L** 和 **GND** 将芯片链接至 CAN 总线接口模块。
- **CAN 总线接口至 GUI 工具或主机工作站** CAN 总线接口（通过 USB）连接到运行 GUI 工具的任意机器。该连接可以是主机工作站，也可以是仅运行 GUI 的独立机器。

5 运行示例和测试结果

1. 获得 AM13E230x DroneCAN 示例的访问权限后，将示例导入 CCS 21.0.0 或更高版本，并确保 SysConfig 版本为 1.28.0 或更高。
2. 下载 DroneCAN GUI 工具。有关详细说明，请参阅设置网页。
3. 目前，DroneCAN 支持功能以及 PCAN 仅在 Linux 系统上经过测试。
4. 从 PCAN-View 网页下载 PEAK-PCAN。
5. mcan_message_libcanard 示例同时包含 TX 和 RX 功能，可以使用 app_cfg.h 文件中定义的宏“APP_MODE”来更改模式。
6. TX 测试：
   a. PCAN-View：
      i. 安装 PCAN 基本驱动程序并打开 PCAN 视图。
      ii. 点击 connect，将标称比特率设置为 250kbps，数据比特率设置为 1,000kbps。
      iii. 在示例中，验证 APP_MODE 已设置为 APP_MODE_TX。重新构建并调试工程。
      iv. 运行示例后，可以在 PCAN 接收空间看到发送的帧。

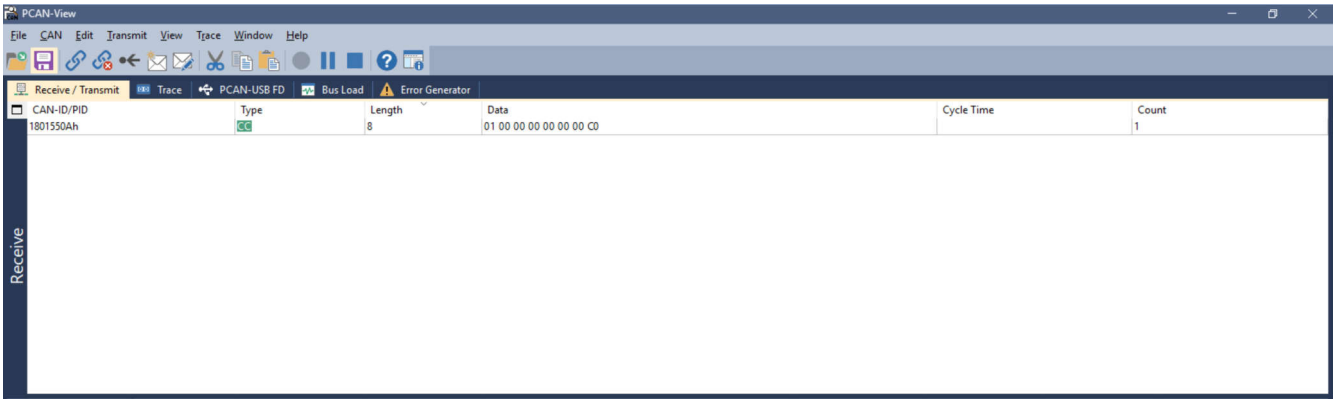

图 5-1. PCAN 帧

   v. 图 5-2 显示了此示例的预期输出。

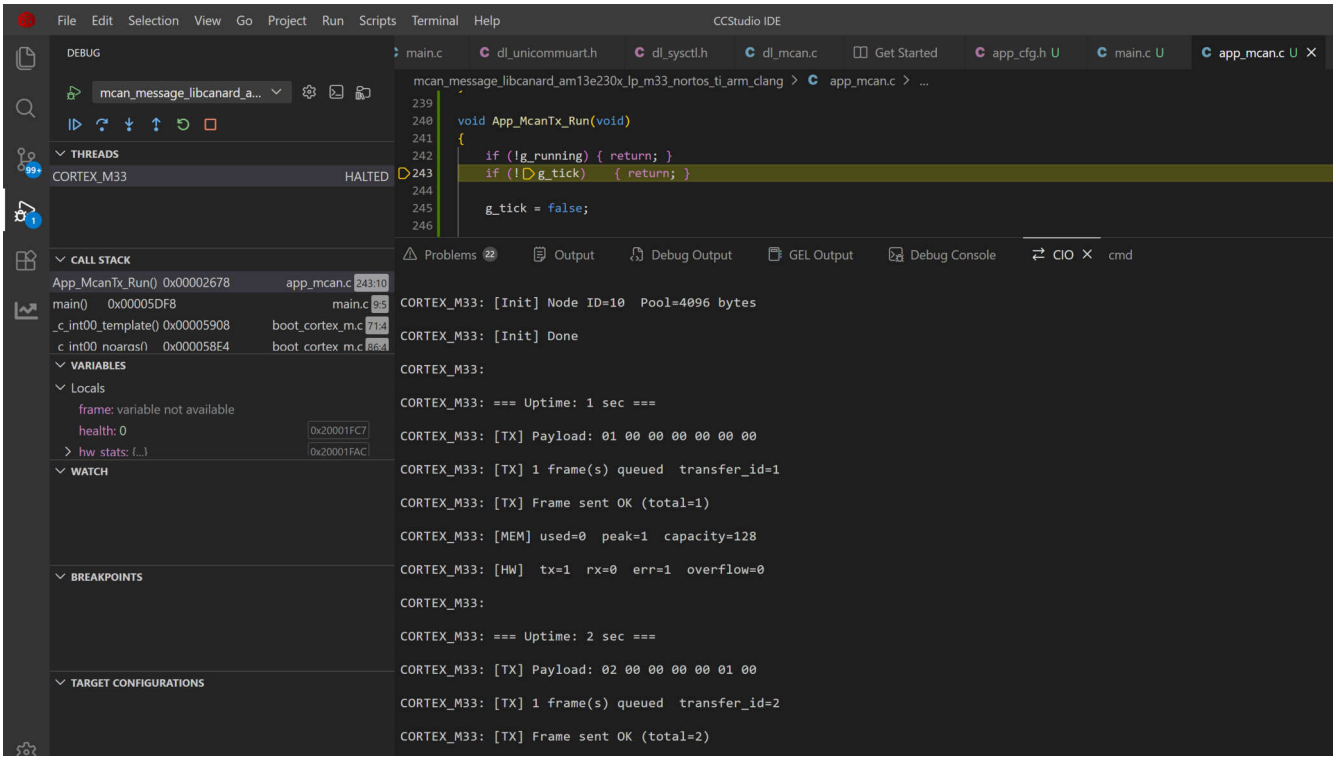

图 5-2. TX 控制台日志输出

b. DroneCAN GUI 工具：
   i. 在启动 DroneCAN GUI 工具之前，请在系统上设置 CAN 总线。
   ii. 使用以下命令：

```
sudo ip link set can0 up

sudo ip link set can0 type can bitrate 250000
```

   iii. 要获知确切的比特率，以文本模式打开 syscfg，查找为 desiredNomRate 设置的值并使用该值。这里 NomRate 设置为 250，因此在设置 CAN 时使用 250,000。
   iv. 现在启动 DroneCAN GUI 工具，在 GUI 上再次设置此比特率，设置一个本地节点 ID，并选中复选框。
   v. 现在打开 Panels，导航到 Bus Monitor 并选择视频图标以开始接收帧。
   vi. 调试并运行 TX 示例后，DroneCAN 上的帧即可见，同时还会显示一些其他的检测信号帧。

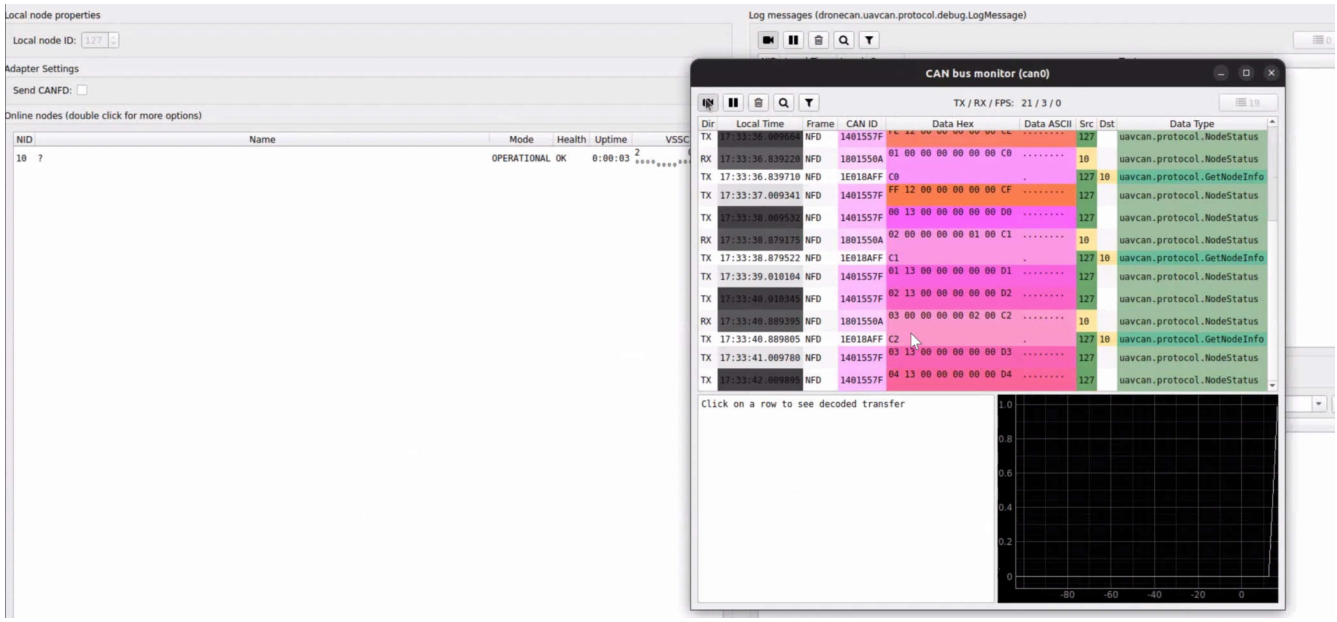

图 5-3. DroneCAN 帧

vii. 此处可见三种类型的帧
   1. 粉色行代表来自节点 10 的节点状态广播。
   2. 橙色行代表来自节点 127 的检测信号消息。
   3. 绿色行代表来自 GUI 工具的获取节点信息的服务请求。

7. RX 测试：
   a. PCAN-View：
      i. 要从 PCAN 发送帧，请转到 Transmit 并导航到 New Message，然后根据需要勾选 CAN FD。

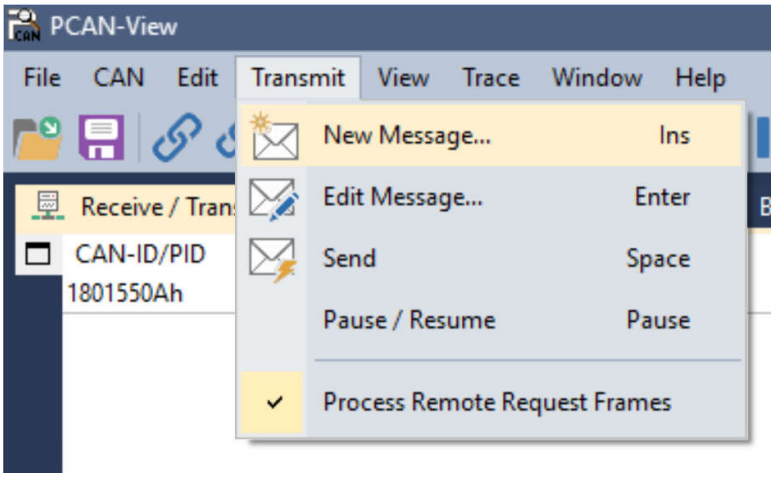

图 5-4. 创建新 PCAN 帧

      ii. 按照正确的 ID 结构（查看应用手册中的 DroneCAN 帧 ID 结构）创建您自己的自定义 DroneCAN 帧。勾选 extended frame 和 bit rate switch。

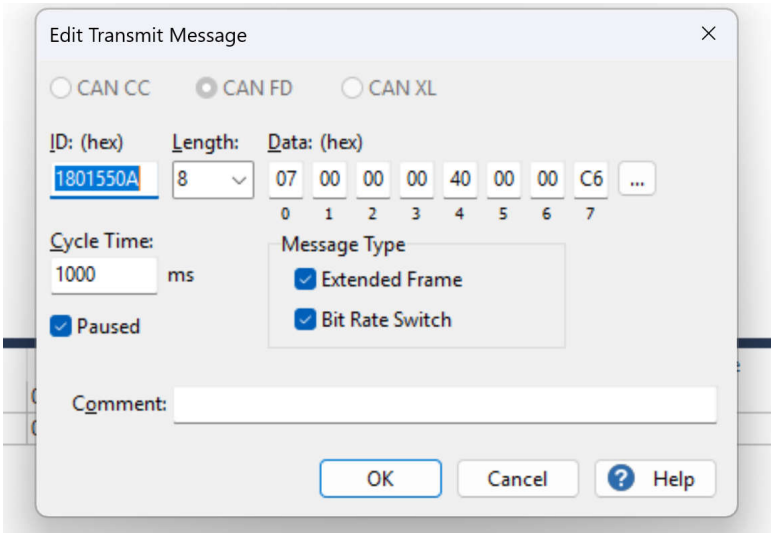

图 5-5. PCAN 帧的属性

      iii. 要连续发送帧，请设置周期时间，以特定的间隔持续发送帧。
      iv. 现在，在开始从 PCAN 发送帧之前，要在示例的控制台中查看这些帧，请打开串行端口 COM8，并设置波特率（如 SysConfig 中所定义，查找名为 targetBaudRate 的变量并使用该值）。
      v. 串行控制台将显示解码后的帧。

b. DroneCAN GUI 工具：
   i. 要从 DroneCAN 发送帧，需要编写一段 python 脚本。
   ii. 导航至 Panels，然后选择 Interactive console 以打开 jupyter 控制台。
   iii. 下面显示了一段示例 python 脚本，用于生成与节点 10 对应的自定义帧。

```
import time

# Clean state tracker for the incrementing byte
tx_state = {'counter': 0}

def send_classical_can_frame():
    try:
        # 1. Build your exact 8-byte payload structure
        payload_bytes = bytearray([tx_state['counter'] & 0xFF, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xC6])

        # 2. Modify the ID flags:
        # (1 << 31) forces Extended 29-bit ID representation (REQUIRED for 0x1801550A)
        # REMOVED: (1 << 30) - This completely disables the CAN FD BRS high-speed bit
rate switch
        can_id_flags = 0x1801550A | (1 << 31)

        # 3. Inject directly into the positional driver hook
        can_send(can_id_flags, payload_bytes)

        # Print confirmation logs locally inside your Linux window
        hex_string = " ".join(f"{b:02X}" for b in payload_bytes)
        print(f"[CLASSICAL CAN TX] ID: 0x1801550A | Data: {hex_string}")

        # Increment counter stepping up sequentially
        tx_state['counter'] = (tx_state['counter'] + 1) % 256

    except Exception as err:
        print(f"[TX ERROR]: {err}")

# Wipe all old background timers completely
stop()

# Force kill any lingering heartbeat ghosts on the driver instance
try:
    __get_node__().heartbeat_publisher.stop()
except:
    pass

# Start the new Classical CAN loop executing at a 1-second interval
classical_task = periodic(1.0, send_classical_can_frame)
print("\n>>> ACTIVE: Classical CAN Extended frame loop running at 1Hz! <<<")


#NOTE: The above example sends a CAN frame whose source ID corresponds to node 10(0A),
to change this, change the last 2 bytes to the respective source node
```

iv. 串行控制台显示两种类型的解码传输：一种是解码后的节点状态，另一种是主节点的节点检测信号。

```
============================================================
  Transfer #31
============================================================
  DTID:        341 (NodeStatus)
  Type:        Broadcast
  Source Node: 127
  Transfer ID: 28
  Priority:    20 (Custom)
  Length:      7 bytes
  Payload:     8C 15 00 00 00 00 00
  --- NodeStatus ---
  Uptime:        5516 sec
  Health:        0 (OK)
  Mode:          0 (OPERATIONAL)
  Sub-Mode:      0
  Vendor Status: 0x0000
============================================================
```

图 5-6. RX 串行端口上解码后的节点状态

```
============================================================
  Transfer #32
============================================================
  DTID:        341 (NodeStatus)
  Type:        Broadcast
  Source Node: 10
  Transfer ID: 6
  Priority:    24 (LOW)
  Length:      7 bytes
  Payload:     0F 00 00 00 00 00 00
  --- NodeStatus ---
  Uptime:        15 sec
  Health:        0 (OK)
  Mode:          0 (OPERATIONAL)
  Sub-Mode:      0
  Vendor Status: 0x0000
============================================================
```

图 5-7. 来自主机的检测信号节点状态

6 关键陷阱

- **DroneCAN GUI 工具未打开？**在系统上设置 CAN 总线时，请在顶部为总线设置正确的波特率。使用命令 *sudo ip link show* 验证位置。在 syscfg 文件中查找正确的波特率，以文本模式打开 syscfg，查看变量 targetBaudRate 的值。
- **PCAN 设备已连接但看不到任何帧？**如果 PEAK-PCAN 设备显示绿灯，则说明收发器连接没有问题，问题出在应用程序上。如果持续亮红灯，则表示 CAN 硬件连接错误。检查 PCAN 配置并相应地进行连接。

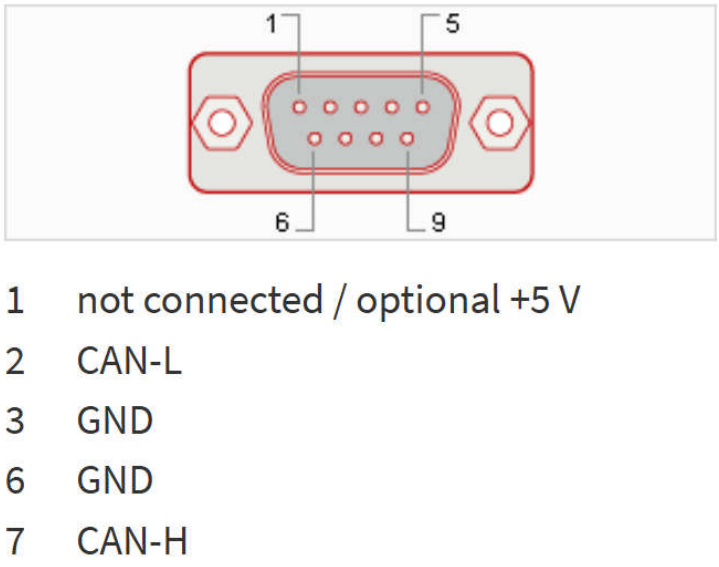

图 6-1. PCAN 连接

- **帧从 PCAN 或 GUI 工具发出，但未被接收到？**验证在 syscfg 中，接收 Rx FIFO 0 是否也设置为接受非匹配帧。

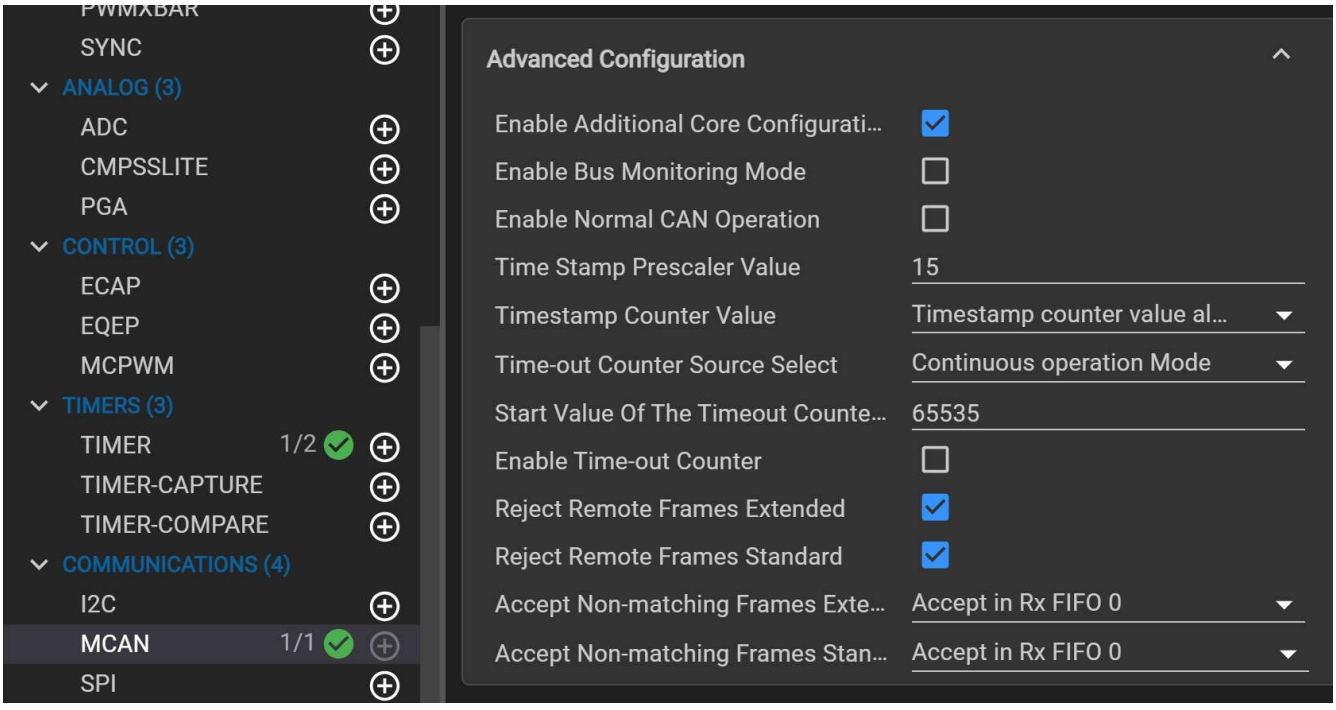

图 6-2. SysConfig 视图

- **RX 测试期间看不到串行日志？**要查看 RX 控制台输出，请在启动应用程序之前，以 115,200 波特率打开 COM8 上的串行终端。可以在 Syscfg UART 配置中确认波特率。

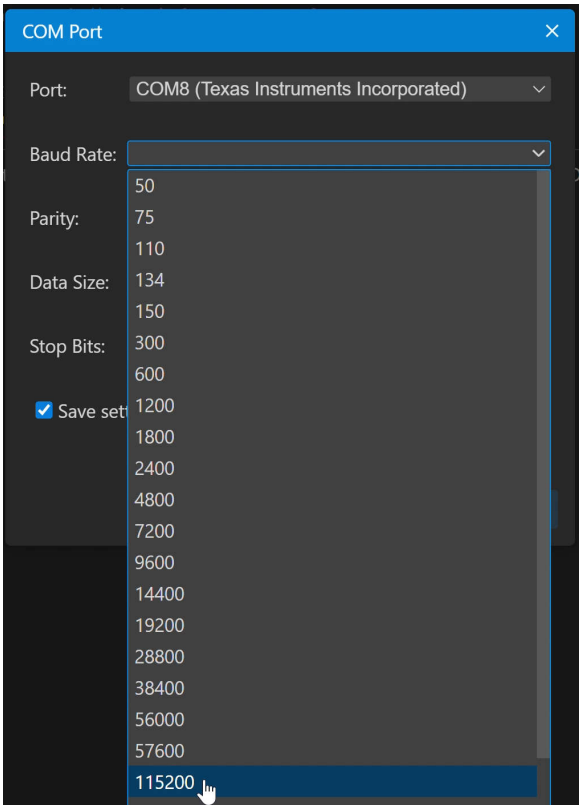

图 6-3. 串行端口和波特率配置

- **发送的帧在 PCAN 上看不到？**断开设备，重新连接，然后检查比特率。仲裁比特率为 desiredNomRate。对于 CAN FD，数据比特率通常可设置为 1Mbps。
- **使用 DroneCAN GUI 工具时，只能看到检测信号帧？**通常，在处理过程中，检测信号帧的发送速度比自定义帧快，并且由于重叠，原本要发送的帧可能会丢失。如果原本要发送的帧丢失，请在 DroneCAN GUI 工具的 Jupyter 控制台中，在 python 代码中包含一段用于抑制检测信号帧的代码，然后重新发送数据包。
- **DroneCAN GUI 工具已打开，但总线监测器收不到任何帧？**确保运行 DroneCAN 的器件设置了节点 ID。在测试示例之前，输入本地节点 ID 并勾选复选框。
- **构建时出现错误：canard_pool 未定义？**检查链接器命令 (.cmd) 文件，确认 .canard_pool 段已正确定义，可开始占用 RAM 中的内存。
- **构建成功，但应用程序在发送一帧后停止，没有继续运行也没有报错？**应用程序正在等待对第一帧成功传输的确认。此问题是连接问题，而非应用程序问题。再次检查并紧固收发器连接。
- **添加更多调试语句后，应用程序未按预期运行？**当使用 printf 函数编写调试语句时，这些语句会干扰同时运行的日志语句，并导致中断。始终使用日志语句来调试代码的任何部分。
- **在调用 Libcanard 库中的更多 API 后，应用程序未按预期运行？**确保这些 API 是直接从应用程序调用的，不要编辑接口层文件。例如，canard_am13x.c 和 canard_am13x.h。
- **由于依赖项的版本冲突而无法构建工程？**验证 SysConfig 版本为 1.28.0 或更高，以及 CCS 版本为 21.0.0 或更高。
- **构建失败，出现文件未找到错误？**确保在构建和 arm 编译器设置下包含 libcanard 源文件，以便使用正确的路径从工程根目录正确导入。

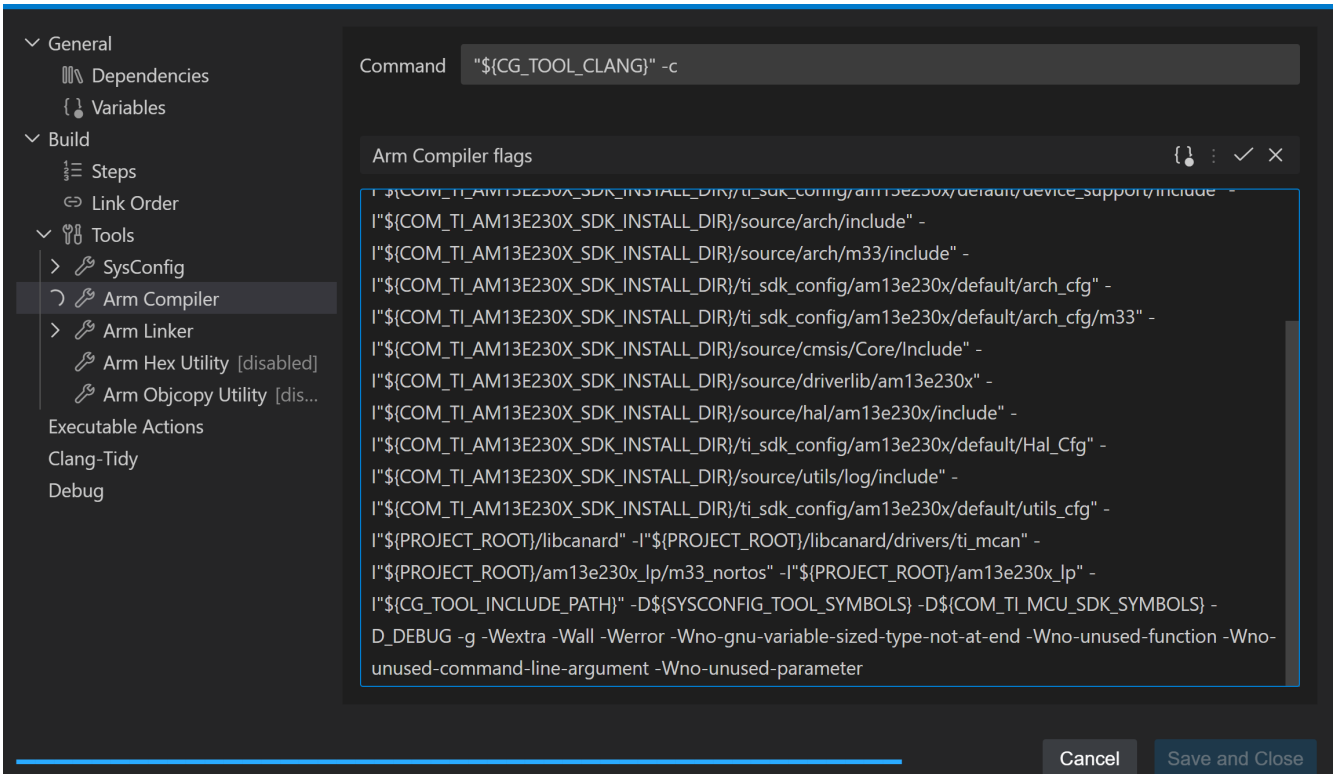

图 6-4. 编译器包含路径配置

- **在传输多个帧时遇到内存问题？**Canard 内存池大小必须仅在应用程序中定义；每次多帧传输占用 32 字节内存，因此请相应地设置内存变量。

7 结语

本应用手册演示了如何使用片上 MCAN 外设在 AM13E230x 微控制器上集成 libcanard DroneCAN 堆栈。canard_am13x 接口层在 libcanard 和 TI DriverLib 之间建立起一座清晰、可复用的桥梁，能够以极少的集成工作量在 AM13x 系列器件上开发符合标准的 DroneCAN 节点。结合基于 SysConfig 的外设配置和所提供的示例应用程序，开发人员拥有一个实用且可直接使用的基础平台，用于在任何基于 AM13x 的设计中构建 DroneCAN 节点。

8 Tools and Software

Tools

E2E Forum	Texas Instruments support forums
Libcanard Github Repo	Support forum for TI AM13E230x

Software

About DroneCAN	Official DroneCAN website

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、与某特定用途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安全、安保法规或其他要求。

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。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 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。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

最后更新日期：2025 年 10 月