参考資料

Application Note

TI AM13E2x での DroneCAN アプリケーションの実行

概要

DroneCAN は、CAN バスを介してノード間で信頼性の高いデータ交換を行うために、UAV やロボット システムで広く使用されている、軽量でオープンな通信プロトコルです。本アプリケーション ノートでは、オンチップの MCAN 周辺機器と Libcanard プロトコル スタックを使用して、TI AM13E230x マイクロコントローラ上で DroneCAN を実装するための実践的なガイドを提供します。本ドキュメントでは、開発者が AM13x ファミリのデバイス上で DroneCAN 対応ノードを構築できるよう、ハードウェアの設定、ソフトウェアの統合、動作例、および一般的な問題について解説します。本アプリケーション ノートは、同じ MCAN 周辺機器を搭載するすべての TI マイコンに適用されます。

目次

商標

1 概要

1.1 DroneCAN：プロトコルの概要と主な機能

DroneCAN は、CAN バスを基盤として構築された軽量なオープンソース通信プロトコルであり、もともとは UAV システム向けに開発されましたが、現在ではロボット工学、産業、航空宇宙分野のアプリケーションで広く採用されています。DroneCAN は、標準化されたメッセージ形式、8 バイトを超えるペイロードの転送セグメンテーション、ノード ID 管理、および異なるベンダーのノード間の相互運用性を実現するためのデータ型シグネチャを定義しています。Libcanard は、DroneCAN スタックの公式 C 言語実装であり、RAM が最小限で、ユーザーが指定したメモリ プール以外の動的メモリ アロケータへの依存がない、リソース制約のある組み込みシステム向けに特別に設計されています。完全に C 言語で記述された Libcanard は、OS やミドルウェア レイヤを必要とせずに、既存の組み込みプロジェクトや TI MCU SDK のサンプルにシームレスに統合できます。このスタックは、DroneCAN の転送エンコーディング、送信、受信、およびデコードの全ライフサイクルを網羅するシンプルな API を公開しています。この設計は、CAN ベースの通信にすでに精通している開発者にとって、容易に採用できるものです。

DroneCAN ID フォーマット: 図 1-1DroneCAN ブロードキャスト メッセージの 29 ビット ID 構造を示しています。

Priority					Data Type ID												Service/ Message Flag	Source Node ID						
28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

図 1-1. CAN ID フォーマット

通常の CAN フレーム ID は通常 11 ビットですが、DroneCAN では拡張フレーム ID (EFF) を使用します。ここでは、フレームが自己記述型となっているため、受信ツールはアプリケーションに関する事前の知識がなくても、通信回線上のあらゆるフレームをデコードすることができ、これにより DroneCAN の統合がはるかに容易になります。

図 1-2 は、通信に DroneCAN を使用しているドローン システムの例を示しています。

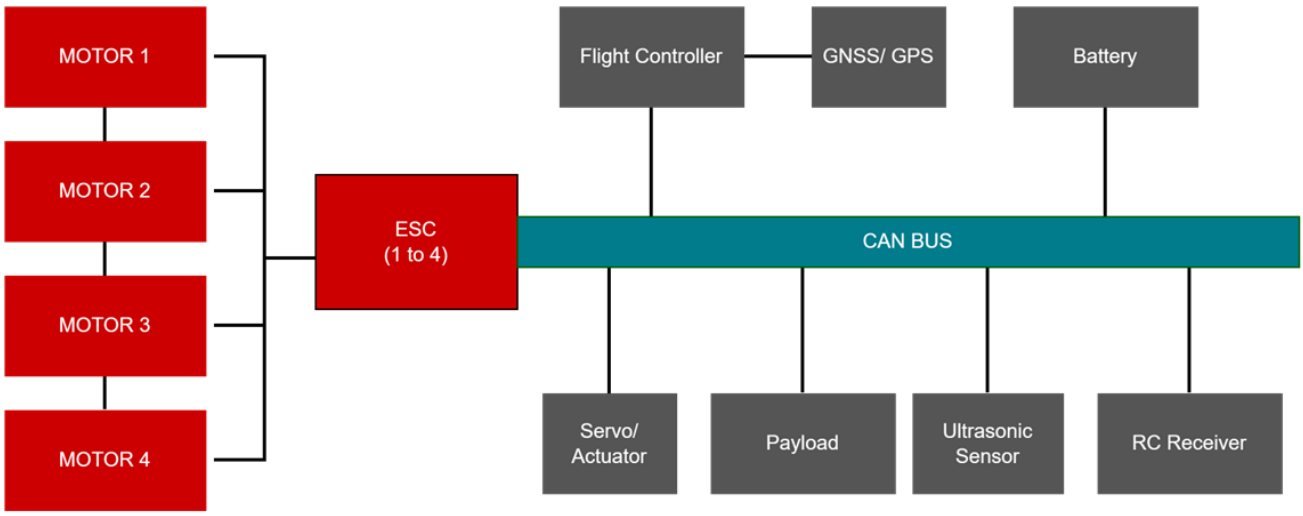


図 1-2. DroneCAN 搭載のドローン システム

ここでは、DroneCAN は主に、フライト コントローラ (中枢) と、モータの速度や回転を制御する電子スピード コントローラ (ESC) との間の通信に使用されます。1 本の CAN バスで、異なるノード間のすべての通信を、干渉の問題なく処理することができます。CAN では、バス上のすべてのノードを接続するのに 2 本の配線だけで済み、このバスは、ソフトウェアを一切介さずに、ハードウェアのみで、すべての障害検出、フレームの再送信、およびその他のエラー処理を完全に処理します。

1.2 AM13E230x の概要:

AM13E230x は、信頼性の高いフィールド通信を必要とする、コスト重視の産業用および自動車用アプリケーション向けに設計された、32 ビット Arm Cortex-M33 ベースのマイクロコントローラです。本デバイスは MCAN 周辺機器を内蔵しており、最大 200Mhz で動作し、設定可能なボー レートで Classical CAN および CAN FD の両方をサポートしています。MCAN 周辺機器は、専用の TX バッファ、RX FIFO、およびハードウェア受入フィルタをサポートする柔軟なメッセージ RAM アーキテクチャを備えており、CPU オーバーヘッドを最小限に抑えながら、効率的なマルチフレーム転送処理を実現します。AM13E230x の SysConfig サポートにより、ビット タイミング、メッセージ RAM 設定、GPIO ピン割り当て、割り込みルーティング、CAN モード設定を含む周辺機器の完全な設定をグラフィカル インターフェースを通じて行うことができ、新規ユーザーの立ち上げ時間を大幅に短縮します。統合された CAN ハードウェア、SysConfig ベースの設定、および DriverLib API サポートの組み合わせにより、AM13E230x は DroneCAN 統合のための実用的かつ高性能なプラットフォームとなっています。

1.2.1 DroneCAN に AM13E230x が選ばれた理由

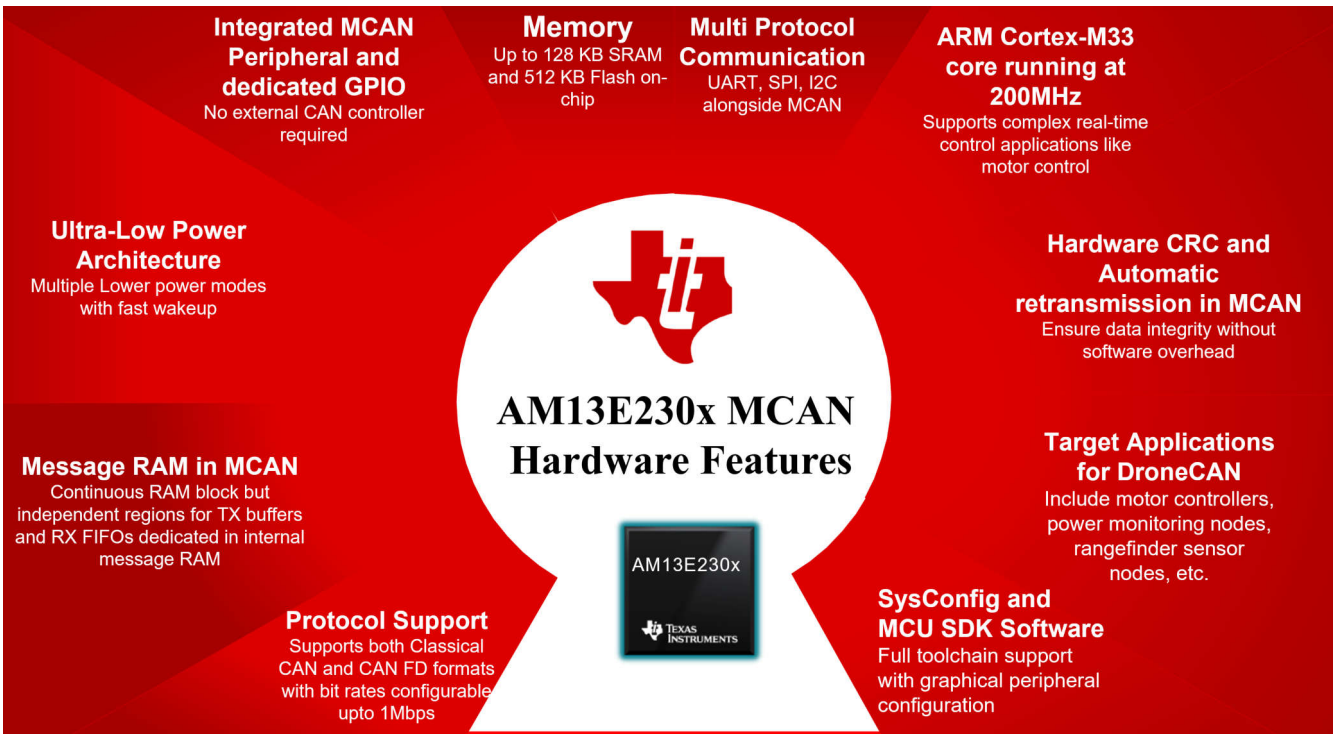


図 1-3. MCAN のハードウェア機能

- AM13E230x MCAN 周辺機器は、専用の GPIO を備えたオンチップ機能であるため、外部の CAN コントローラは必要ありません。
- 専用の CAN コア (コントローラ) により、CAN 通信による CPU 帯域幅の消費が最小限に抑えられるため、DroneCAN を実行している場合でも、本デバイスは非常に効率的なモータ制御を実現します。
- DroneCAN は、すべてのフレームが正しく配信されることを前提としており、MCAN 周辺機器は、ソフトウェアの介入なしに、エラーが発生したフレームの再送信を自動的に処理します。
- MCAN には、TX バッファ、RX FIFO、およびハードウェア受入フィルタ用にそれぞれ独立した領域を持つ内部メッセージ RAM が搭載されています。そのため、受信したすべてのフレームは処理される前に格納およびフィルタリングされます。これは、複数のノード タイプが同時に送信を行い、バスが混雑している場合に非常に効率的です。
- レジスタ空間は、Tx および Rx 転送ハンドラと、実際のハードウェア FIFO およびメッセージ RAM を接続しています。すべてのレジスタ アクセスは、TI Driverlib を通じて抽象化されています。
- CAN と CAN FD の両方がサポートされているため、将来、ハードウェア ボードの変更を行うことなく、同じハードウェアでより高いスループットを必要とするアプリケーションに対応することが可能です。
- この設計は、超低消費電力アーキテクチャを採用しています。高速ウェイクアップ機能を備えた複数の低消費電力モードにより、ほとんどアイドル状態にあるノードからの迅速な応答が可能になります。

- MCAN 周辺機器全体、ビット タイミング、メッセージ RAM の設定、GPIO ピン割り当て、割り込みルーティング、時刻、および CAN モードの設定はすべて Syscfg を通じて行われるため、エンド開発者の負担が軽減されます。
- MCAN に加え、UART、SPI、I2C 通信プロトコルも利用可能であり、これによりデバイスは他のコンポーネントと同時に通信を行うことができます。

2 システム アーキテクチャ

Libcanard は DroneCAN プロトコルに関連するすべての処理を処理しますが、TI MCAN 周辺機器については一切認識していません。Libcanard は Canard フレームのエンコード、デコード、および処理を管理しますが、これらのフレームをバスを介して物理的に転送することはできません。一方、TI MCAN 周辺機器は CAN フレームの送受信を処理できますが、Canard ID 構造や属性については認識していません。このギャップを埋めるために、**専用のインターフェース レイヤ**が必要です。これにより、開発者はアプリケーション内で周辺機器とのやり取りに関するコードを直接記述することなく、AM13x デバイス上で DroneCAN ノードを構築できるようになります。

2.1 ソフトウェア スタックの概要

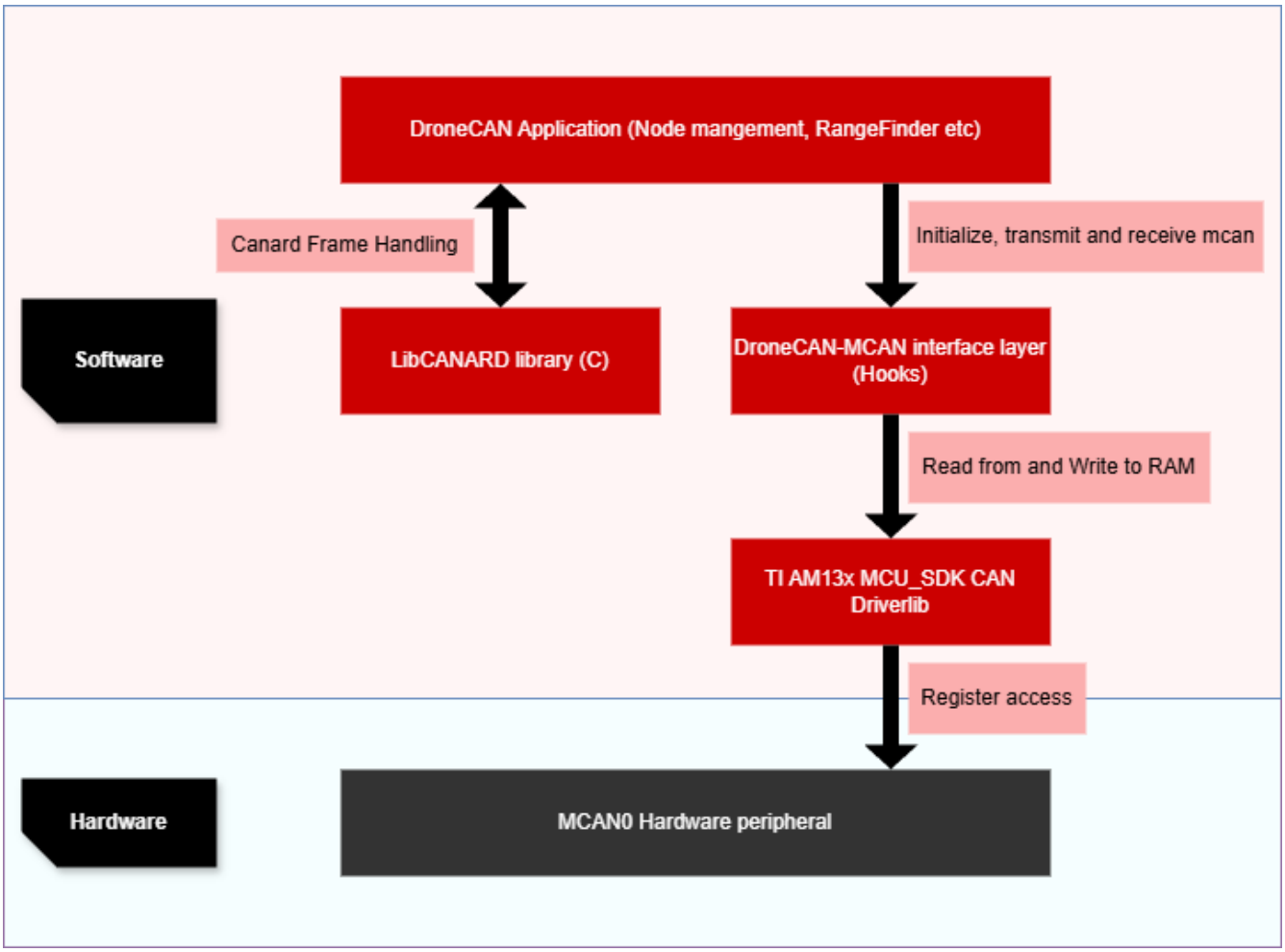


図 2-1. ソフトウェア アプリケーションの構造

- **DroneCAN アプリケーション**は最上位に位置し、システム全体を駆動します。このアプリケーションは、スタックの初期化、転送データのエンコードおよびデコード、ノードの管理、および定期的なメンテナンス処理を行います。
- **Libcanard** は、アプリケーションが直接やり取りを行うプロトコル ライブラリです。このプロトコル ライブラリは、フレームのエンコード、デコード、転送データの再構築、CRC、メモリ管理など、DroneCAN プロトコルのすべての処理を、ハードウェアに関する知識を必要とせずに処理します。
- **DroneCAN-MCAN インターフェース レイヤ**は、アプリケーションと TI 周辺機器を接続するハードウェア抽象化層です。
  - アプリケーションは、Libcanard のフレーム構造を MCAN メッセージ RAM 形式に変換し、トランシーバの制御、フィルタのプログラミング、およびモード遷移を処理します。
  - ハードウェア固有の複雑な処理はすべてこのレイヤ内に収められており、アプリケーションと Libcanard の両方を、周辺機器固有のコードから完全に解放しています。
- **MCAN0 ハードウェア周辺機器**は、AM13E230x 上の物理的な CAN コントローラであり、CAN バスを駆動します。

2.2 Libcanard-MCAN インターフェース レイヤ：TX および RX フロー

このインターフェース レイヤは、主に、MCAN から Libcanard の、およびその逆のフレーム形式の変換、初期化、フィルタ設定、送信、受信などの API を提供します。これは、同様の MCAN 周辺機器を搭載したデバイス上のあらゆる DroneCAN プロジェクトで直接再利用可能です。以下は、Libcanard と統合されたアプリケーションにおけるパケットの送受信の大まかなフローです。

2.2.1 TX のフロー

フレーム転送の一般的なフローは 図 2-2 に示します。

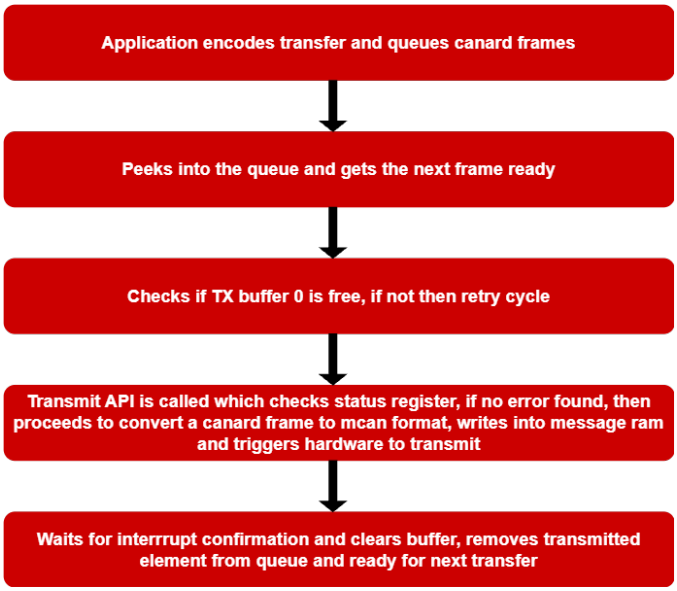


図 2-2. フレーム転送フロー

2.2.2 RX のフロー

フレーム受信の一般的なフローは 図 2-3 に示します。

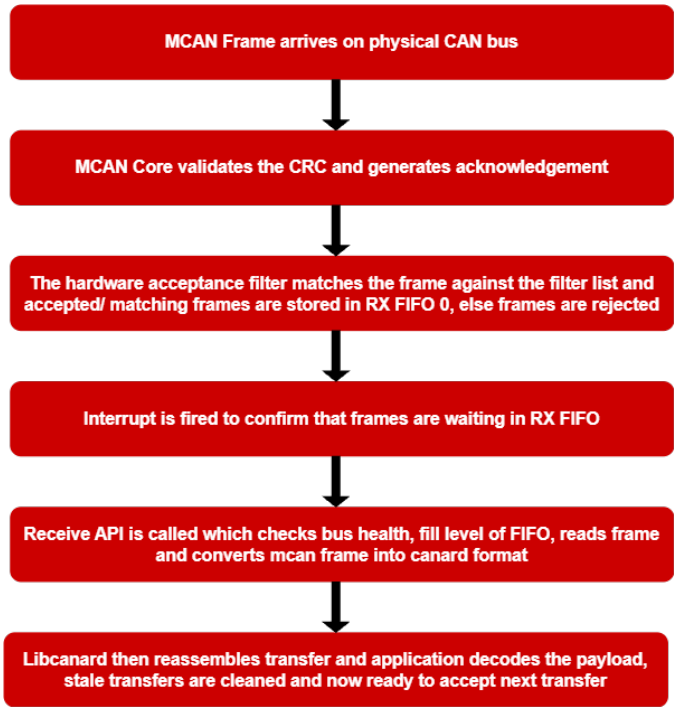


図 2-3. フレーム受信フロー

3 ソフトウェア アプリケーションの概要

AM13E230x Launchpad 上で実装された Libcanard アプリケーションのサンプルは、ご要望に応じて提供可能です。これを入手するには、E2E™ フォーラム、または TI の営業担当者やフィールド担当者に連絡してください。

3.1 プロジェクト構造

以下に、CCS DroneCAN プロジェクトの構造例を示します。

- ルート ディレクトリにある example.syscfg ファイルは、ハードウェア設定の唯一の拠点となります。CCS SysConfig エディタでこのファイルを開き、周辺機器の設定を変更してください。
- Libcanard ディレクトリには、変更されていない libcanard ソース ファイルと、canard_am13x インターフェース ファイル (ドライバ内) が含まれています。これらのファイルは、変更を加えることなくプロジェクトにインポートできます。
- ノード ID、メモリ プール サイズ、データ型 ID、タイマー インスタンス、ティック レートなどのアプリケーション定数は、app_cfg.h に一元化されています。必要に応じてこれらの詳細を変更してください。
- アプリケーション レベルの DroneCAN フレームのエンコードおよびデコード ロジックは app_mcan.c にあり、main.c は純粋にエントリ ポイントとして機能します。

```
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 周辺機器の設定

すべての周辺機器の設定は、SysConfig を通じて管理されます。DroneCAN を正常に動作させるには、以下の設定が必要です。

MCAN の一般設定

- 「メッセージは 1 回のみ送信する」のチェックを外し、フレームが自動的に再送信されるようにしてください。これにより、誤ったフレームが永久に失われることを防げます。

メッセージ RAM レイアウト

- 標準フィルタ リスト:2 つの要素、開始アドレス 0
- 拡張フィルタ リスト:2 つの要素、開始アドレス 8
- TX バッファ:開始アドレス 24、1 つの要素
- TX イベント FIFO:開始アドレス 96
- RX FIFO 0:TX 領域に重ならないように、要素数、ウォーターマーク、および開始アドレスが設定されている

受入フィルタ

- 標準 ID フィルタ：従来のビットマスク、一致するフレームを RX FIFO 0 にルーティング
- 拡張 ID フィルタ：範囲フィルタ、29 ビットの全範囲をカバーするために EFID2 を 536870911 に設定
- 一致しない拡張フレームの受入：拒否に設定

割り込み

- 割り込みライン 1 が有効
- RX FIFO 0 の新規メッセージ、TX 完了、バスオフ、パッシブ エラー、および警告ステータスの割り込みソースはすべて、ライン 1 にルーティングされている

タイマ

- 定期的な割り込みを生成するには、タイマー周辺機器のインスタンスが必要です。このアプリケーションでは、タイマー周辺機器を使用して、受信フレームの処理に渡しられたタイムスタンプを維持し、古い転送データをクリーンアップしています。

参考までに、前述の TX および RX の例の SysConfig ファイルを以下に示します。

```
/**
 * 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 メモリ プールの割り当て

プールはアプリケーションを通じて定義および設定できます。この目的のために、リンカー コマンド ファイルに以下のブロックが組み込まれています:

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

メモリ使用量:

実装されたサンプル アプリケーションは、デバッグ ビルドで約 37KB、リリース ビルドで約 35KB のメモリを消費しますが、さらなる最適化の余地があります。つまり、M33 コア上で他のアプリケーションを実行したり、TinyEngine™ 上で AI モデルを実行したりするのに十分なメモリが確保されています。

4 ハードウェア設定

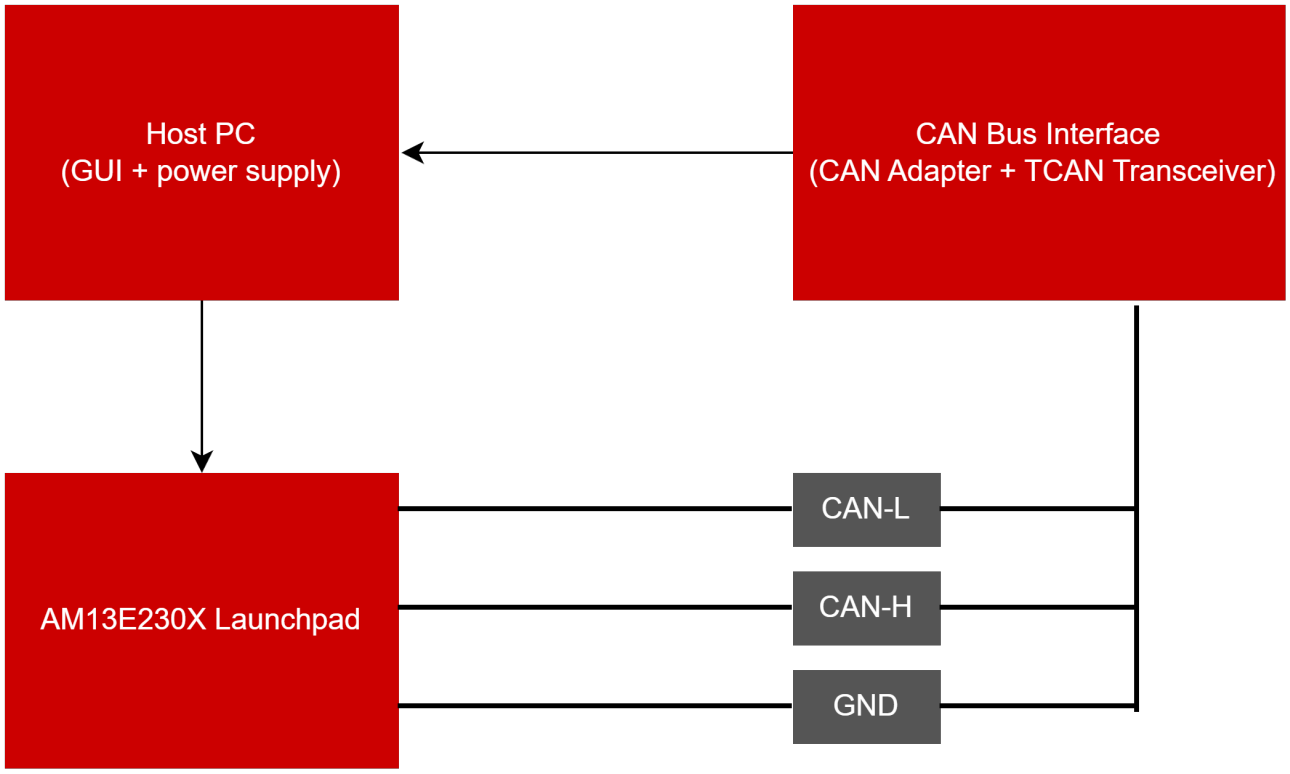


図 4-1. ハードウェア設定

ハードウェア接続:

- **ホスト ワークステーションから AM13E230X へ**:AM13E230X チップに電力を供給し、主要な制御インターフェースとして機能します。
- **AM13E230X から CAN バス インターフェースへ (または CAN バス インターフェースから AM13E230X へ)** 3 線式 CAN 接続:チップと CAN バス インターフェース モジュールを接続する **CAN-H**、**CAN-L**、および **GND**。
- **CAN バス インターフェースから GUI ツールまたはホスト ワークステーションへ**:CAN バス インターフェースは、GUI ツールを実行しているマシンに (USB 経由で) 接続されます。この接続先は、ホスト ワークステーションでも、GUI のみを実行している別のマシンでも構いません。

5 サンプル実行とテスト結果

1. AM13E230x DroneCAN サンプルを入手したら、そのサンプルを CCS 21.0.0 以降にインポートし、SysConfig のバージョンが 1.28.0 以上であることを確認してください。
2. DroneCAN GUI ツールをダウンロードしてください。詳細な手順については、セットアップのウェブページを参照してください。
3. 現時点では、DroneCAN および PCAN のサポートは Linux システムでのみテストされています。
4. PCAN-View のウェブページから PEAK-PCAN をダウンロードしてください。
5. mcan_message_libcanard サンプルには TX および RX の両方の機能が含まれており、app_cfg.h ファイルで定義されている「APP_MODE」マクロを使用してモードを変更できます。
6. TX のテスト:
   a. PCAN ビュー:
      i. PCAN Basic ドライバをインストールし、PCAN ビューを開きます。
      ii. 「接続」をクリックし、「公称ビットレート」を 250kbps、「データビットレート」を 1000kbps に設定します。
      iii. このサンプルでは、APP_MODE が APP_MODE_TX に設定されていることを確認してください。プロジェクトを再ビルドしてデバッグします。
      iv. サンプルを実行すると、送信されたフレームが PCAN の受信領域に表示されます。

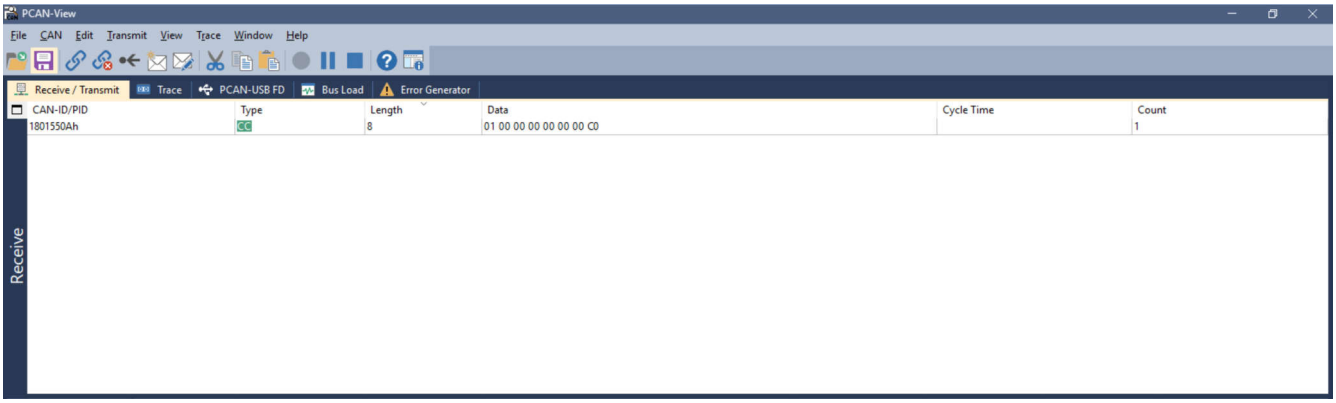


図 5-1. PCAN フレーム

   v. この例の予想される出力を、図 5-2 に示します。

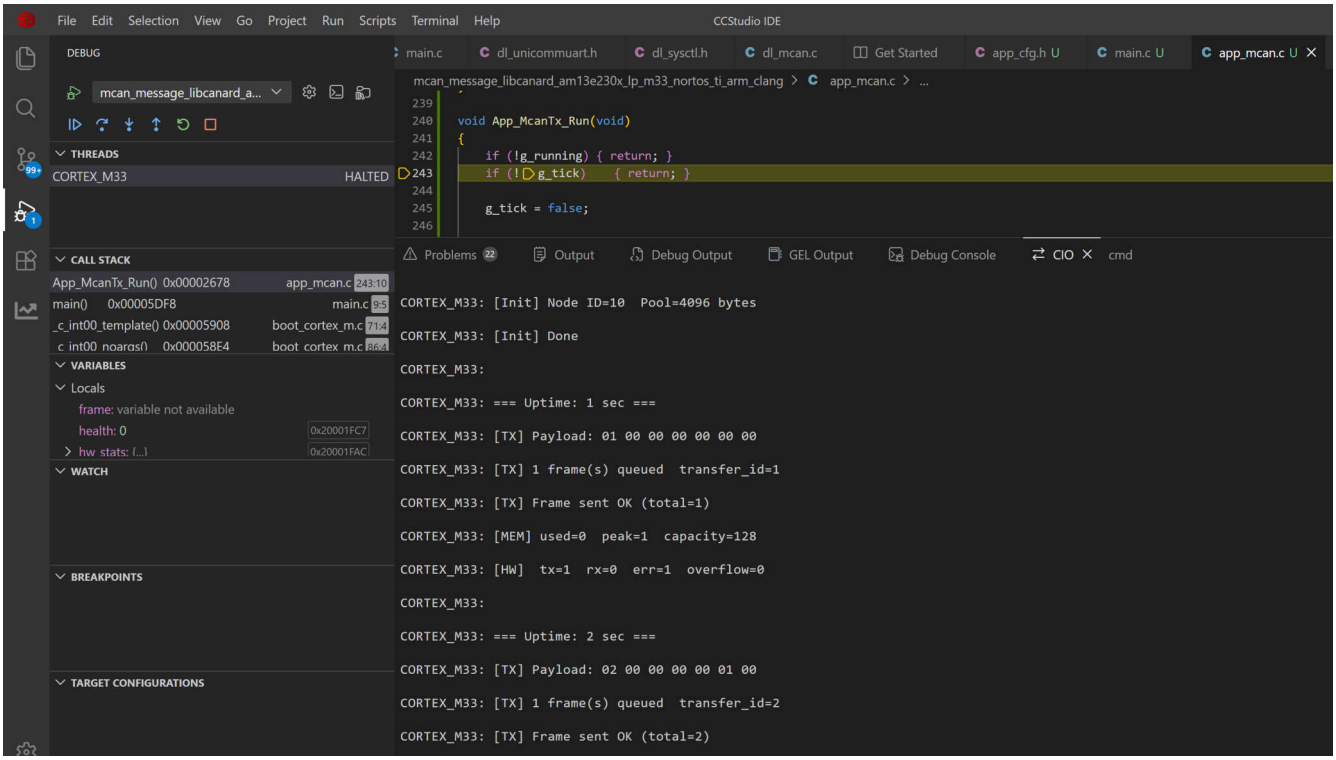


図 5-2. TX コンソールのログ出力

b. DroneCAN GUI ツール:
   i. DroneCAN GUI ツールを起動する前に、システム上で CAN バスを設定してください。
   ii. 以下のコマンドを使用してください:

```
sudo ip link set can0 up

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

   iii. 正確なビット レートを把握するには、テキスト モードで syscfg を開き、desiredNomRate に設定されている値を確認して、その値を使用してください。ここでは NomRate が 250 に設定されているため、CAN の設定時には 250000 を使用してください。
   iv. 次に、DroneCAN GUI ツールを起動し、GUI 上でこのビット レートを再度設定し、ローカル ノード ID を設定して、チェックボックスを選択してください。
   v. 次に、「パネル」を開き、「バス モニタ」に移動して、動画アイコンを選択し、フレームの受信を開始してください。
   vi. デバッグを行い、TX サンプルを実行すると、DroneCAN 上でフレームが確認できるようになり、その他にいくつかのハートビート フレームも表示されます。

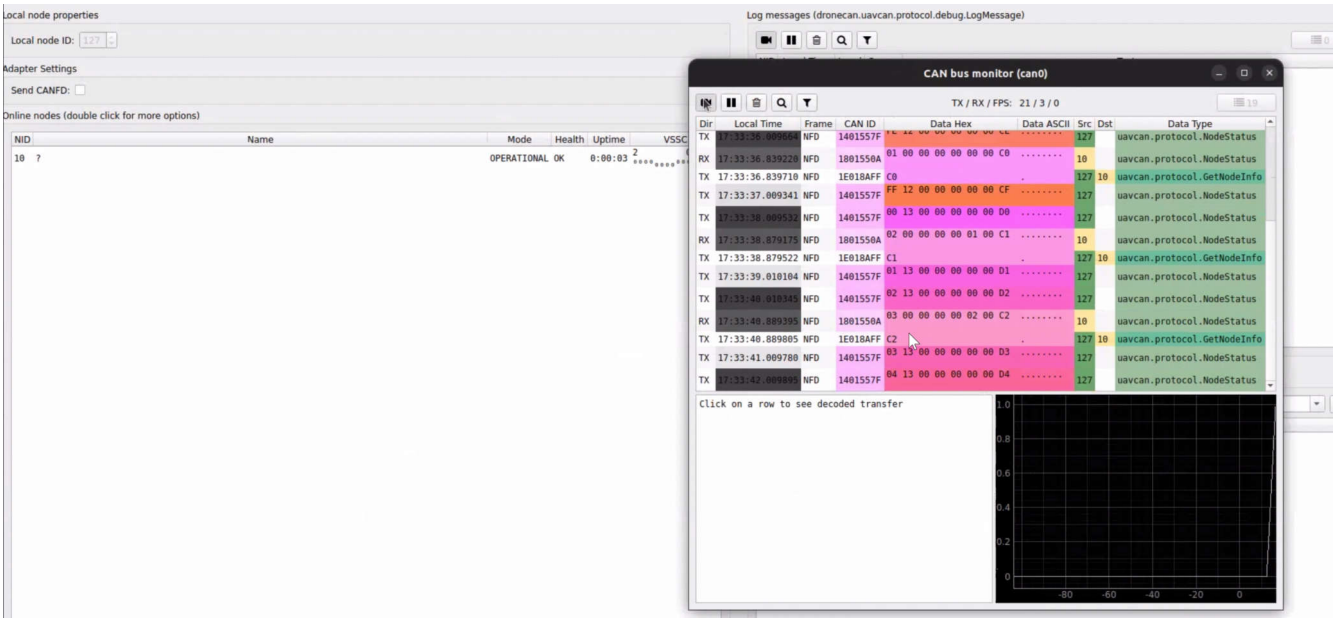


図 5-3. DroneCAN フレーム

vii. ここでは 3 種類のフレームが表示されています

1. ピンクの行は、ノード 10 からブロードキャストされたノード ステータスを表しています。
2. オレンジ色の行は、ノード 127 からのハートビート メッセージを表しています。
3. 緑色の行は、ノード情報を取得するための GUI ツールからのサービス リクエストを表しています。

7. RX のテスト:
   a. PCAN ビュー:
      i. PCAN からフレームを送信するには、「送信」に移動し、「新規メッセージ」を選択してから、必要に応じて「CAN FD」にチェックを入れます。

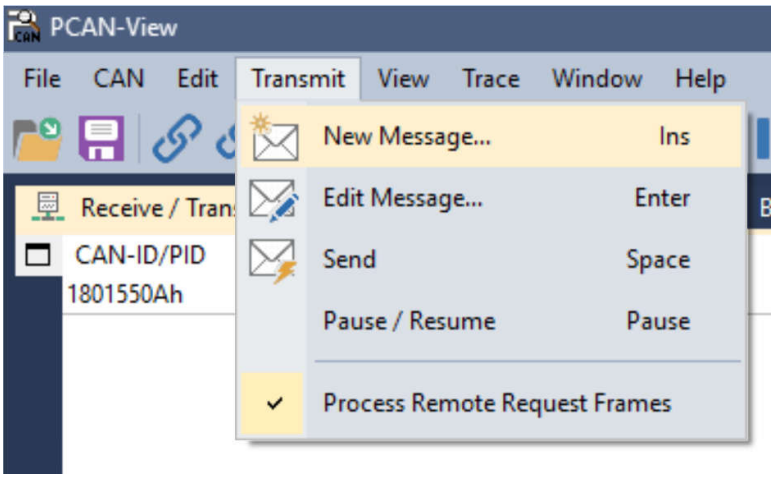


図 5-4. 新しい PCAN フレームの作成

      ii. 適切な ID 構造に従って、カスタム DroneCAN フレームを作成してください (AppNote の「DroneCAN フレーム ID 構造」を参照してください)。拡張フレームおよびビット レート切り替えについて確認してください。

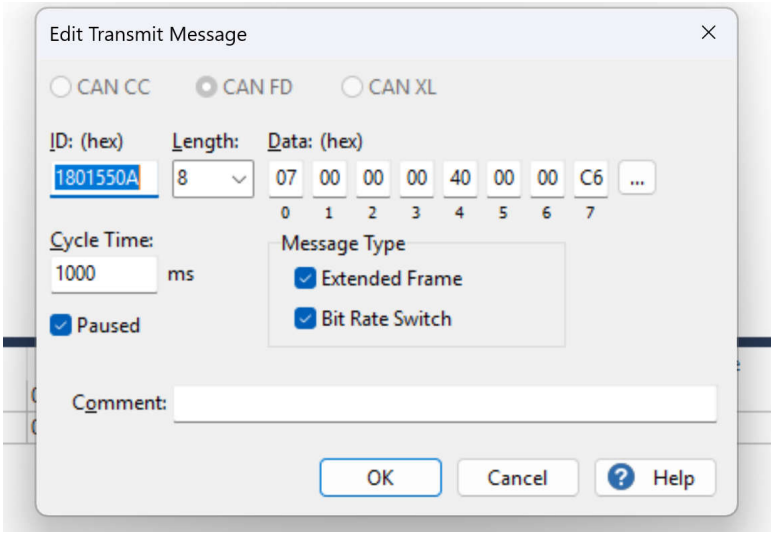


図 5-5. PCAN フレームの属性

      iii. フレームを連続して送信するには、サイクル タイムを設定して、特定の間隔でフレームを絶えず送信するようにします。
      iv. PCAN からフレームの送信を開始する前に、例のコンソールでこれらのフレームを表示するには、ボーレート (SysConfig で定義されている値、targetBaudRate という変数を確認し、その値を使用してください) を設定してシリアル ポート COM8 をオープンします。
      v. シリアル コンソールには、デコードされたフレームが表示されます。

b. DroneCAN GUI ツール:
   i. DroneCAN からフレームを送信するには、Python スクリプトを記述します。
   ii. 「パネル」に移動し、「対話型コンソール」を選択して Jupyter コンソールを開きます。
   iii. 以下は、ノード 10 に対応するカスタマイズされたフレームを生成する Python スクリプトの例です。

```
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. シリアル コンソールには、2 種類のデコードされた転送データが表示されます。1 つはデコードされたノード ステータス、もう 1 つはホスト ノードのハートビートです。

```
================================================================
  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
================================================================
```

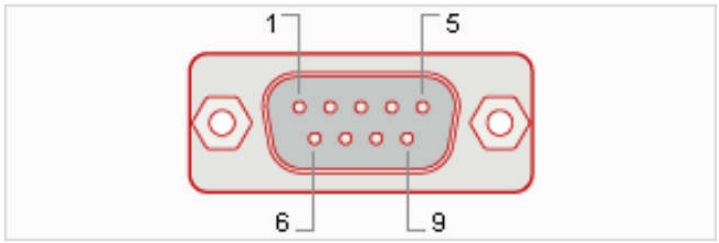
図 5-6. RX シリアル ポート上のデコードされたノード ステータス

```
================================================================
  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
================================================================
```

図 5-7. ホストからのハートビート ノード ステータス

6 主な注意点

- **DroneCAN GUI ツールが開きませんか？**システムで CAN バスを設定する際、上部で正しいボー レートが設定されていることを確認してください。コマンド「sudo ip link show」を使用して、接続先を確認してください。syscfg ファイル内で正しいボー レートを特定し、syscfg をテキスト モードで開いて、変数 targetBaudRate の値を確認してください。
- **PCAN デバイスは接続されているのにフレームが表示されませんか？**PEAK-PCAN デバイスのライトが緑色に点灯している場合、トランシーバの接続に問題はなく、アプリケーション側に問題があります。ライトが赤く点灯し続けている場合は、CAN ハードウェアの接続が間違っています。PCAN の設定を確認し、それに応じて接続し直してください。



1	not connected / optional +5 V
2	CAN-L
3	GND
6	GND
7	CAN-H

図 6-1. PCAN の接続

- **PCAN または GUI ツールからフレームが送信されているのに、受信されていませんか？**syscfg の Rx FIFO 0 において、不一致のフレームも受信対象に設定されていることを確認してください。

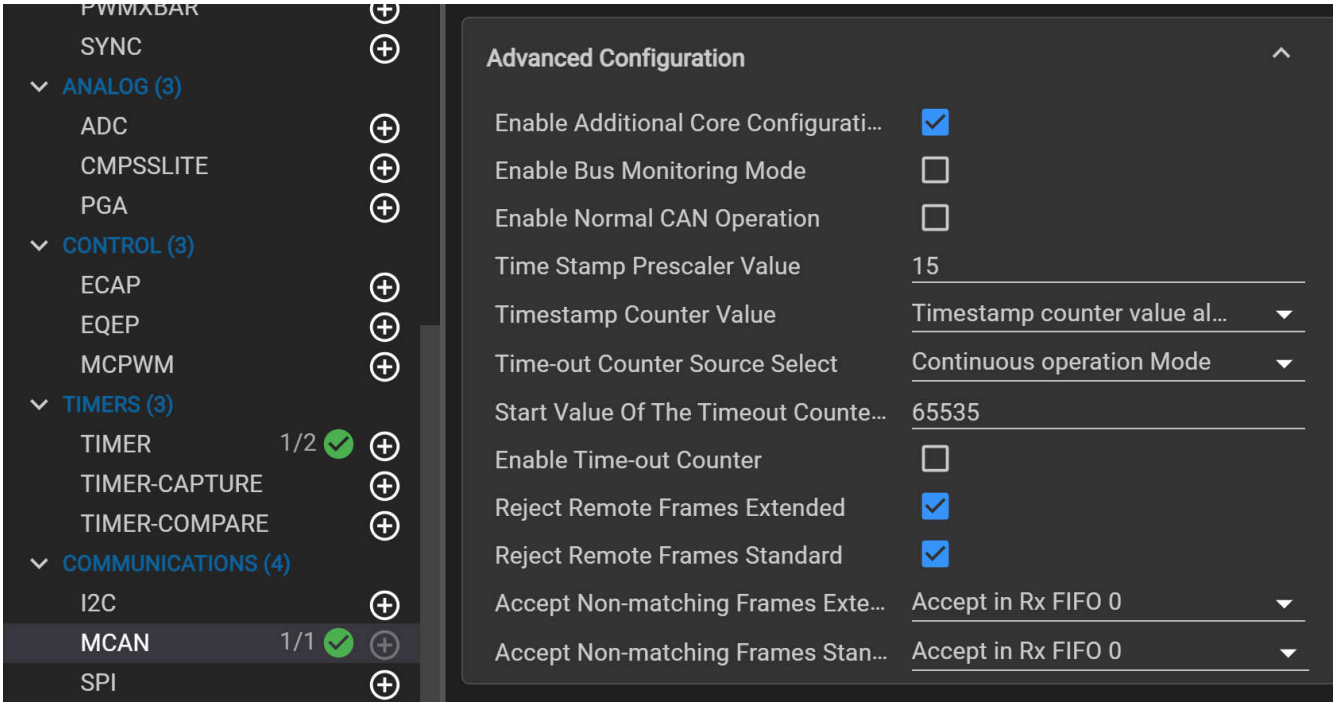


図 6-2. SysConfig ビュー

- **RX テスト中にシリアル ログが表示されませんか？**RX コンソールの出力を表示するには、アプリケーションを起動する前に、COM8 で 115200 ボーでシリアル ターミナルを開いてください。ボー レートは、Syscfg の UART 設定で確認できます。

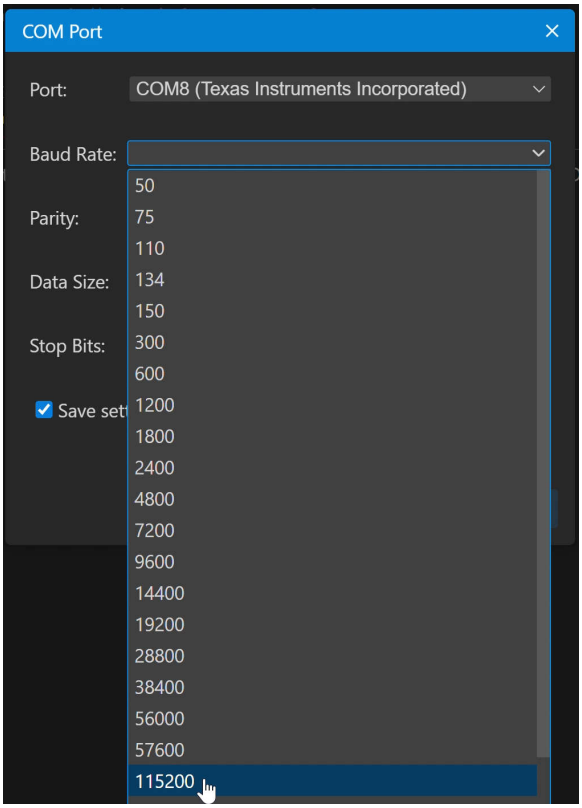


図 6-3. シリアル ポートおよびボー レートの設定

- **PCAN に送信されたフレームが表示されませんか？**デバイスを一度切り離し、再度接続して、ビット レートを確認してください。アービトレーションのビット レートは desiredNomRate です。データ ビット レートは、通常、CAN FD の場合 1Mbps に設定できます。
- **DroneCAN GUI ツールを使用している際、ハートビート フレームしか表示されませんか？**通常、処理中はハートビート フレームがカスタム フレームよりも高速に送信されるため、オーバーラップが発生し、本来送信されるべきフレームが失われることがあります。最初に送信されたフレームが失われた場合は、DroneCAN GUI ツールの Jupyter コンソールで、ハートビート フレームを抑制し、パケットを再送信するコードを Python コードに追加してください。
- **DroneCAN GUI ツールは起動するのに、バス モニタがフレームを受信しませんか？**DroneCAN を実行しているデバイスにノード ID が設定されていることを確認してください。サンプルをテストする前に、ローカル ノード ID を入力し、ボックスにチェックを入れてください。
- **ビルド中に「canard_pool undefined」エラーが発生しますか？**リンカ コマンド (.cmd) ファイルを確認し、RAM 内のメモリを割り当て始めるために .canard_pool セクションが正しく定義されているか確認してください。
- **ビルドは成功したのに、アプリケーションが 1 回の送信後に停止し、処理が続行されない、あるいはエラーも表示されませんか？**アプリケーションは、最初のフレームの転送成功の確認応答を待っています。この問題は、アプリケーションの問題ではなく、接続の問題です。トランシーバの接続を再度確認し、しっかりと固定してください。
- **デバッグ用ステートメントを追加した後、アプリケーションが期待通りに動作しなくなりましたか？**printf 関数を使用してデバッグ用ステートメントを記述すると、同時に実行されているログ出力ステートメントと干渉し、割り込みが発生します。コードのどの部分をデバッグする場合でも、常にログ出力ステートメントを使用してください。
- **libcanard ライブラリからさらにいくつかの API を呼び出した後、アプリケーションが期待通りに動作しなくなりましたか？**これらはアプリケーションから直接呼び出されていることを確認してください。インターフェース レイヤのファイルは編集しないでください。例えば、canard_am13x.c や canard_am13x.h などです。
- **依存関係のバージョン競合によりプロジェクトがビルドできませんか？**SysConfig のバージョンが 1.28.0 以上、CCS のバージョンが 21.0.0 以上であることを確認してください。

- **「ファイルが見つかりません」というエラーでビルドに失敗していますか？**プロジェクトのルートから正しいパスでインポートされるよう、ビルドおよび arm コンパイラの下に libcanard ソースファイルが適切に含まれていることを確認してください。

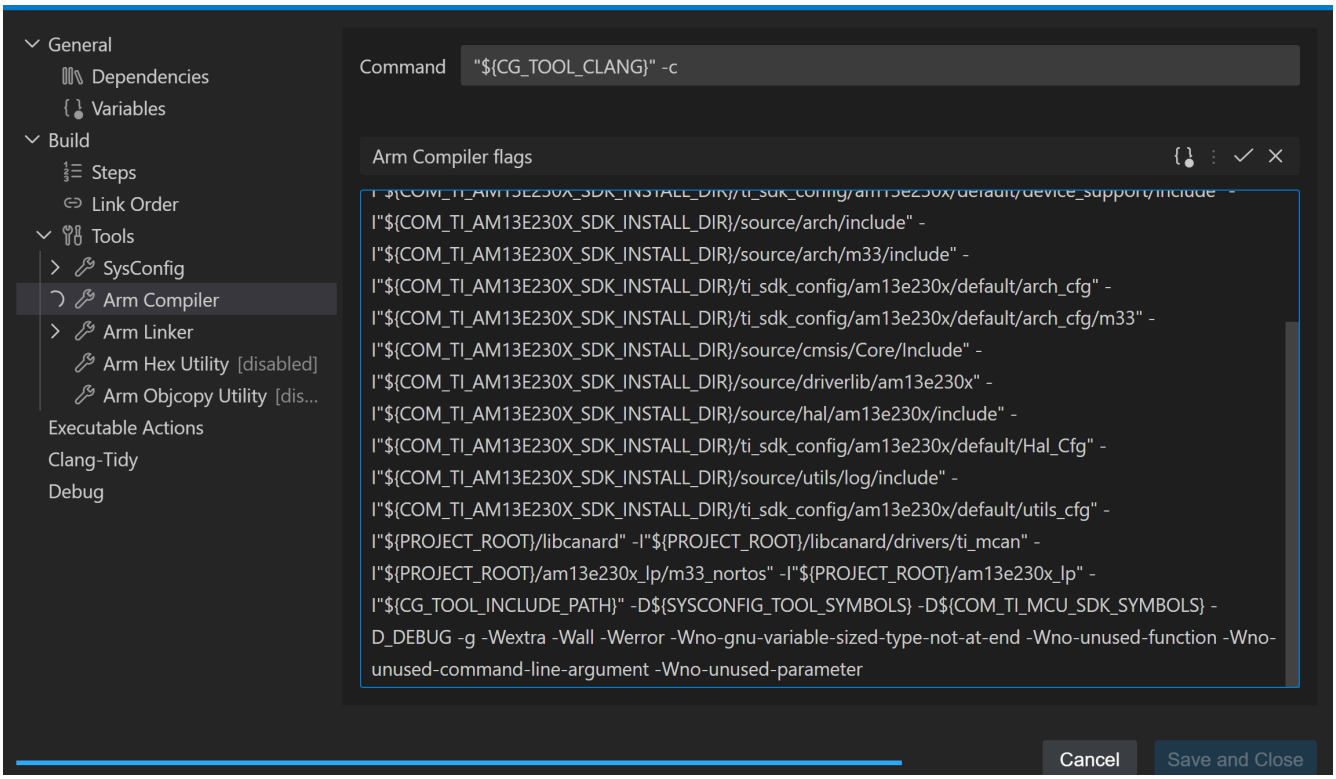


図 6-4. コンパイラのインクルード パスの設定

- **複数のフレームを転送中にメモリの問題が発生した場合は？**Canard のプール サイズはアプリケーション内でのみ定義する必要があります。各マルチフレームは 32 バイトのメモリを占有するため、メモリ変数をそれに応じて設定してください。

7 まとめ

このアプリケーション ノートでは、オンチップの MCAN 周辺機器を使用して、AM13E230x マイクロコントローラ上で libcanard DroneCAN スタックを統合する方法を示しました。canard_am13x インターフェース レイヤは、libcanard と TI DriverLib の間に、クリーンで再利用可能なブリッジを提供し、最小限の統合作業で AM13x ファミリ デバイス上で標準準拠の DroneCAN ノード開発を可能にします。SysConfig ベースの周辺機器設定および提供されたサンプル アプリケーションと組み合わせることで、開発者は、あらゆる AM13x ベースの設計において DroneCAN ノードを構築するための、実用的で即戦力となる基盤を手に入れることができます。

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

重要なお知らせと免責事項

TI は、技術データと信頼性データ (データシートを含みます)、設計リソース (リファレンス デザインを含みます)、アプリケーションや設計に関する各種アドバイス、Web ツール、安全性情報、その他のリソースを、欠陥が存在する可能性のある「現状のまま」提供しており、商品性および特定目的に対する適合性の黙示保証、第三者の知的財産権の非侵害保証を含むいかなる保証も、明示的または黙示的にかかわらず拒否します。

これらのリソースは、TI 製品を使用する設計の経験を積んだ開発者への提供を意図したものです。(1) お客様のアプリケーションに適した TI 製品の選定、(2) お客様のアプリケーションの設計、検証、試験、(3) お客様のアプリケーションに該当する各種規格や、その他のあらゆる安全性、セキュリティ、規制、または他の要件への確実な適合に関する責任を、お客様のみが単独で負うものとします。

上記の各種リソースは、予告なく変更される可能性があります。これらのリソースは、リソースで説明されている TI 製品を使用するアプリケーションの開発の目的でのみ、TI はその使用をお客様に許諾します。これらのリソースに関して、他の目的で複製することや掲載することは禁止されています。TI や第三者の知的財産権のライセンスが付与されている訳ではありません。お客様は、これらのリソースを自身で使用した結果発生するあらゆる申し立て、損害、費用、損失、責任について、TI およびその代理人を完全に補償するものとし、TI は一切の責任を拒否します。

TI の製品は、 TI の販売条件 、 TI の総合的な品質ガイドライン 、 ti.com または TI 製品などに関連して提供される他の適用条件に従い提供されます。TI がこれらのリソースを提供することは、適用される TI の保証または他の保証の放棄の拡大や変更を意味するものではありません。 TI がカスタム、またはカスタマー仕様として明示的に指定していない限り、TI の製品は標準的なカタログに掲載される汎用機器です。

お客様がいかなる追加条項または代替条項を提案する場合も、TI はそれらに異議を唱え、拒否します。

最終更新日：2025 年 10 月