Application Note

MSPM0 マイコンの MCAN (CAN FD) モジュール入門

Hao Mengzhen

概要

モジュラー コントローラ エリア ネットワーク (MCAN) ペリフェラルは、MSPM0™ リアルタイム マイコン (MCU) ファミリ内の一部のデバイスに CAN フレキシブル データレート (CAN FD) を実装しています。MCAN ペリフェラルを搭載したデバイスの一部には、MSPM0G350x、MSPM0G351x デバイスがあります。ここに記載されている例は、MCAN モジュールを搭載した任意の MSPM0 MCU で動作するものです。このアプリケーション ノートでは、各種動作モード用に MCAN モジュールをセットアップする方法を示す、いくつかのプログラミング例について説明します。この目的は、MCAN ペリフェラルのプログラミングをユーザーが理解できるようにすることです。

サンプル コードは、MSPM0G3507 でテスト済みです。ただし、これらの例は、MCAN モジュールを搭載したすべての MSPM0 デバイスで実行できるよう簡単に調整できます。ほとんどの例では、動作用に 2 番目の CAN FD ノードが必要です。この要件は、CAN FD または従来型 CAN と CAN FD の両方のプロトコルを使用できる任意の CAN バス分析ツールを搭載した別のマイコンで満たすことができます。現在、多くの USB バス ベースのツールが利用可能です。これらのツールは、バス トラフィックを可視化するだけでなく、フレームを生成することもでき、デバッグに非常に役立ちます。デバッグには、CAN FD トリガまたはデコード機能を内蔵したオシロスコープが不可欠です。

本書全体で、MCAN と CAN FD という用語を同じ意味で使用しています。*CAN FD* はプロトコルを表し、*MCAN* はプロトコルを実装する MCU 内のペリフェラルを指します。本書で説明しているサンプルのプロジェクト ファイルは、MSPM0-SDK の一部としてダウンロードできます。

目次

図の一覧

商標

1 はじめに

CAN は、車載アプリケーション向けに独自に開発されたシリアル プロトコルです。CAN は堅牢性と信頼性のため、産業用機器、医療用電子機器、列車、航空機など多様なアプリケーションに使用できます。CAN プロトコルは、高度なエラー検出機能と閉じ込め機構を備えており、物理的なレベルで簡単な配線が可能です。オリジナルの CAN プロトコル規格は現在、最新の CAN FD 規格と区別するために *Classical* CAN と呼ばれています。CAN バスの代表的な配線を図 1-1 に示します。

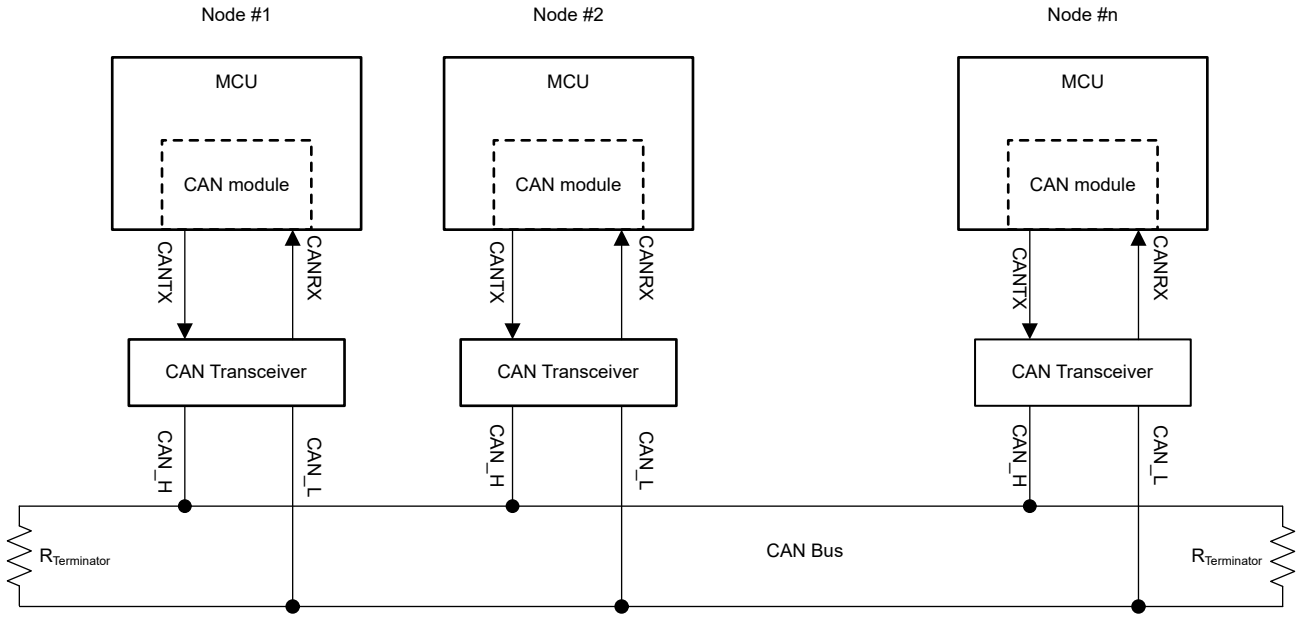

図 1-1. 代表的な CAN バス

CAN フレキシブル データ レート (CAN FD) は、Classical CAN のビット レートと、1 フレームで転送されるバイト数に関して拡張したものであり、これにより通信の実効スループットが向上します。Classical CAN は最大 1Mbps のビット レートとフレームあたり 8 バイトのペイロード サイズに対応していますが、CAN FD は最大 5Mbps のビット レートとフレームあたり最大 64 バイトのペイロード サイズに対応しています。CAN FD フレームのフレーム構造を図 1-2 に示します。

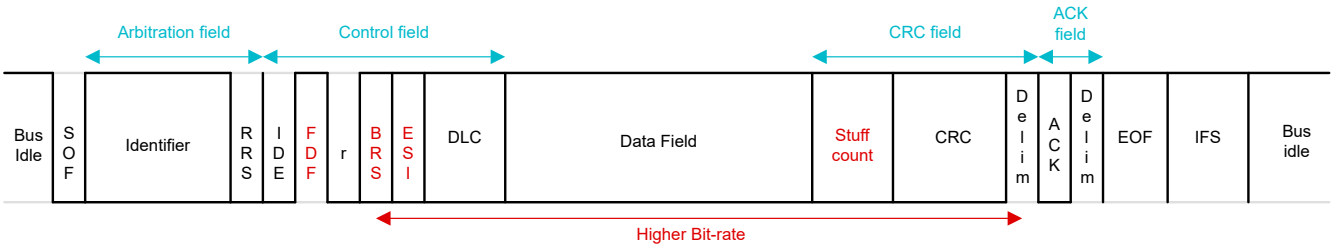

図 1-2. CAN FD フレーム

1.1 MCAN の機能

MCAN モジュールの主な機能は次のとおりです。

- CAN プロトコル 2.0A、B、ISO 11898-1:2015 に準拠
- 完全な CAN FD のサポート (最大 64 データ バイト)
- AUTOSAR および SAE J1939 をサポート
- 最大 32 個の専用送信バッファ
- 構成可能な送信 FIFO、最大 32 個の素子
- 構成可能な送信キュー、最大 32 個の素子
- 構成可能な送信イベント FIFO、最大 32 個の素子
- 最大 14 個の専用受信バッファ
- 2 つの構成可能な受信 FIFO、各最大 14 個の素子、1kB のメッセージ RAM 付き
- 最大 128 個のフィルタ素子
- セルフ テスト用のループバック モード
- マスカブル割り込み (2 つの設定可能な割り込みライン、訂正可能な ECC、カウンタ オーバーフロー、クロックの停止またはウェークアップ)
- マスク不可能割り込み (訂正不可能な ECC)
- 2 つのクロック ドメイン (CAN クロックおよびホスト クロック)
- メッセージ RAM の ECC チェック
- クロックの停止とウェークアップのサポート
- タイムスタンプ カウンタ
- 1Mbps 公称ビット レート、8Mbps データ ビット レート

サポートされない機能:

- ホスト バス ファイアウォール
- クロックのキャリブレーション
- CAN 経由のデバッグ

2 MCAN モジュールの SysConfig 構成

SysConfig は、ハードウェアとソフトウェアの構成に関する課題の簡素化と、ソフトウェア開発の迅速化に役立つ設計を採用した構成ツールです。SysConfig は、Code Composer Studio™ 統合開発環境 (IDE) の一部として利用できるスタンドアロン アプリケーションです。さらに、TI デベロッパー ゾーンにアクセスすると、SysConfig をクラウド環境で実行できます。SysConfig は、ピン、ペリフェラル、無線、ソフトウェア スタック、RTOS、クロック ツリーなどのコンポーネントを構成するための直観的なグラフィカル ユーザー インターフェイスを採用しています。SysConfig は、ソフトウェア開発を迅速化するために、競合の検出、表示、解決を自動的に実行します。

ユーザーが SysConfig を使用してアプリケーションで MCAN モジュールを構成することを TI は推奨しています。SysConfig を使用すると、ユーザーはわずかな労力で、必要に応じて MCAN モジュールを迅速に設定できます。次のセクションでは、MSPM0-SDK 内の mcan_loopback_LP_MSPM0G3507 の例を用いて、SysConfig を使用して MCAN を構成する方法を紹介します。

2.1 MCAN クロック周波数

図 2-1 に、「MCAN Clock Frequency」(MCAN クロック周波数) ブロックに含まれるパラメータを示します。

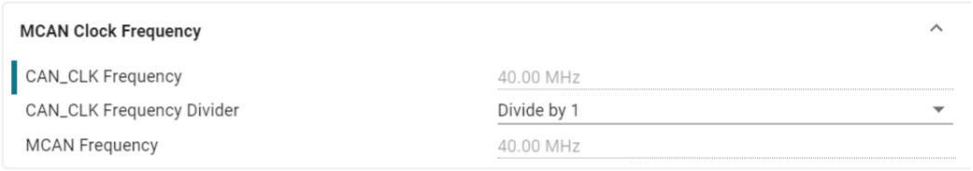

図 2-1. MCAN クロック周波数

MCAN のペリフェラル非同期クロック (CAN_CLK) には、HFXT または SYSPLL のいずれかを介してクロックを供給できます。このクロック構成は、SYSCTL セクションを使用して行う必要があります。MCAN の周波数を 40MHz として構成することを TI は推奨します。

注

CAN_CLK のクロック ソースとして HFXT を使用することを TI は推奨します。SYSOSC の精度は、特別なシナリオではアプリケーションの要件を満たしておらず、エラー フレームが大きくなります。CAN_CLK の推奨周波数は、20MHz、40MHz、80MHz です。

2.2 MCAN の基本構成

図 2-2 に、「MCAN Basic Frequency」(MCAN 基本周波数) ブロックに含まれるパラメータを示します。

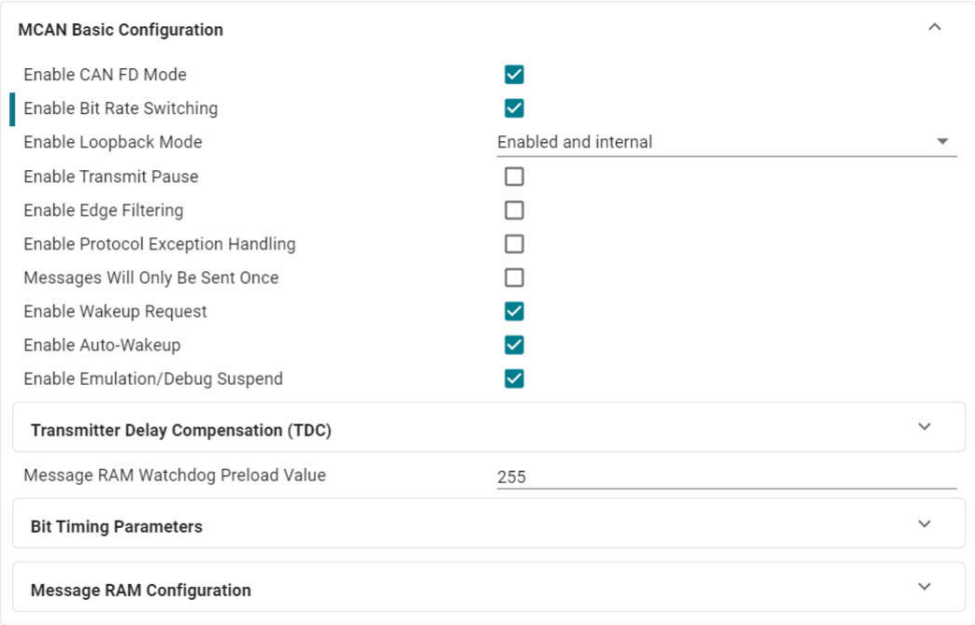

図 2-2. MCAN の基本構成

- Enable CAN FD Mode (CAN FD モードの有効化): CAN フレキシブル データ モードを有効にする必要がある場合に使用します。
- Enable Bit Rate Switching (ビット レート スイッチングの有効化): この機能を有効にすると、MCAN はアービトレーション レートではなく、より高いレートでデータを送信します。この機能は、CAN FD モードが有効化されている場合にのみ機能します。
- Enable Loopback Mode (ループバック モードの有効化): 送信されたメッセージが受信メッセージになります。これにより、ユーザーは CAN トランシーバなしで、CAN_TX ピンの CAN メッセージを監視できます。
- Enable Transmit Pause (送信の一時停止の有効化): 次の送信の前に、CAN ビットを 2 回停止します。この送信一時停止機能は、CAN メッセージ ID が固有で簡単に変更できない CAN ネットワークでの使用を目的としています。これらのメッセージ ID は、他の定義済みメッセージ ID よりも高い優先度を持つことができますが、特定のアプリケーションでは相対的な優先度は逆です。これにより、1 つの ECU が CAN メッセージのバーストを送信でき、別の ECU からの CAN メッセージが遅延 (一時停止) します。
- Enable Edge Filtering (エッジ フィルタリングの有効化): ハード同期のエッジを検出するには、2 つの連続したドミナント タイム クォンタが必要です。この機能を有効にすることで、ノードは信号のエッジを正確に検出し、データ ビット時間が長い場合でもハード同期を実行できるようになり、CAN バス通信の安定性と信頼性が向上します。
- Enable Protocol Exception Handling (プロトコル例外処理の有効化): 将来のプロトコル拡張のために予約されているビットを検出します。
- Messages Will Only Be Sent Once (メッセージを 1 回のみ送信): 自動再送信が無効な場合、送信エラーや NACK、あるいは MCAN モジュールがアービトレーションに負けた場合でも、MCAN モジュールはメッセージを再送信しません。
- Enable Wakeup Request (ウェークアップ要求の有効化): CAN RXD アクティビティ時に MCAN モジュールがウェークアップできるようにします。
- Enable Auto-Wakeup (自動ウェークアップの有効化): MCAN モジュールが MCAN CCCR.INIT ビットを自動的にクリアし、イネーブルなウェークアップ要求時に MCAN を完全にウェークアップできます。クロック停止要求を発行すると、MCAN モジュールはパワーダウン モード (スリープ モード) に移行します。IDLE から ACTIVE への遷移中に、ウェークアップ要求の有効化および自動ウェークアップの有効化機能がイネーブルの場合、MCAN_CCCR.INIT ビットをクリアするために読み取り - 変更 - 書き込みが発行されます。MCAN コアは、クロック停止要求の解除に応答してクロック停止アクノリッジを解除した後、動作を再開します。

注

ハードウェアによってクロック停止要求が解除された後、最初のフレーム (ウェークアップ フレーム) は受信されないことに注意してください。これは、クロック停止が発行された後、IP に実行されているアクティブなクロックがないためです。したがって、クロック停止要求を解除した後、クロックを有効にするウェークアップ フレームを再送信する必要があります。

- Enable Emulation or Debug Suspend (エミュレーションまたはデバッグ中断の有効化):エミュレーションまたはデバッグのため、MCAN モジュールを中断できます。
- Message RAM Watchdog Preload Value (メッセージ RAM ウォッチドッグのプリロード値):RAM ウォッチドッグはメッセージ RAM の READY 出力を監視します。MCAN の汎用マスター インターフェイスによるメッセージ RAM アクセスにより、設定された値でメッセージ RAM ウォッチドッグ カウンタが開始されます。メッセージ RAM が READY 出力をアクティブにして正常に完了したことを示すと、カウンタがリロードされます。カウンタがゼロまでカウントダウンするまでにメッセージ RAM から応答がない場合、カウンタは停止し、割り込みフラグ MCAN_IR.WDI が設定されます。RAM ウォッチドッグ カウンタは、ホスト (システム) クロックからクロック供給されます。

2.2.1 トランスミッタ遅延補償 (TDC)

図 2-3 に、「Transmitter Delay Compensation」(トランスミッタ遅延補償) (TDC) ブロックに含まれるパラメータを示します。

図 2-3. トランスミッタ遅延補償 (TDC)

TDC は、CAN FD システム内のトランシーバのループ遅延に起因する遅延を補償するために使用されるメカニズムです。この遅延により、ノードは高ビット レートのデータ転送中に、サンプル ポイントで有効なビット エラー チェックを実行できなくなる可能性があります。具体的には、TDC はデータ位相にセカンダリ サンプル ポイント (Secondary Sample Point:SSP) を導入し、遅延を考慮して送信されたビットを受信ビットと比較します。これにより、ビット エラーが正しく検出され、処理されることになります。

TDC は、ビット レートが高く、データ ビットが短くなり、ループ遅延が大きくなる場合に必要です。このような遅延により、ノードがビット エラー検出に対する正しいサンプル ポイントを見逃す可能性があります。TDC はデータ位相中のみアクティブであり、アービトレーション位相には影響しません。

TDC を使用すると、データ位相のビット時間が公称ビット時間よりも短くなるため、エラー検出を犠牲にすることなくデータ レートを向上させることができます。

- TDC Filter Window Length (TDC フィルタ ウィンドウ長) (サイクル):このフィルタ機能で SSP 位置の最小値を定義することで、受信 FDF ビット内の支配的なグリッチが受信 res ビットの立ち下がりエッジ前に遅延補償測定を終了して早期 SSP 位置が得られる場合を回避できます。
- TDC Offset (TDC オフセット) (サイクル):このオフセットは、受信ビット内の SSP の位置 (データ位相のビット時間の半分など) を調整するために使用されます。

TDC の測定方法の詳細については、『MSPM0 G シリーズ 80MHz マイコン テクニカル リファレンス マニュアル』を参照してください。

2.2.2 ビット タイミング パラメータ

図 2-4 に、「Bit Timing Parameters」(ビット タイミング パラメータ) ブロックに含まれるパラメータを示します。

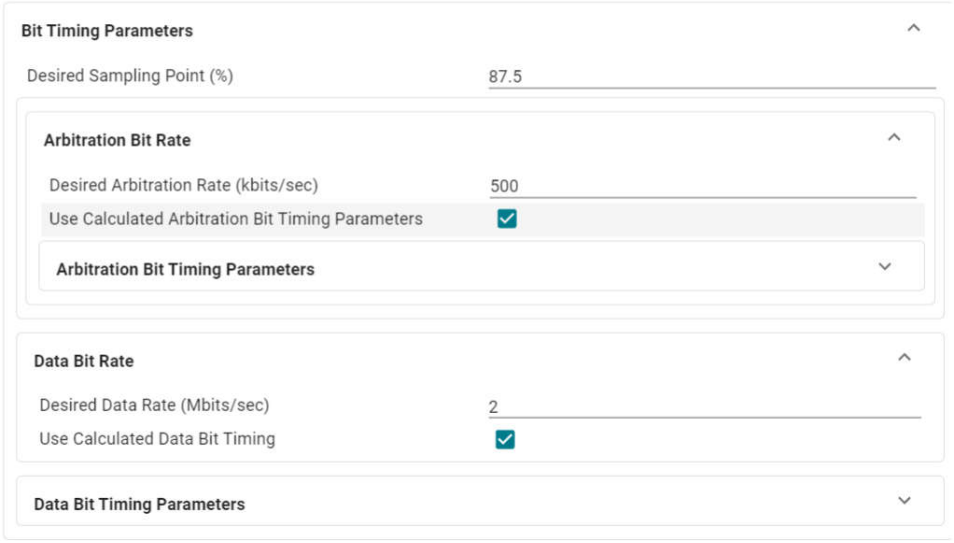

図 2-4. ビット タイミング パラメータ

CAN バスのビット タイミングは、CAN 通信でのデータ転送の同期と制御に使用される重要なパラメータを指します。これにより、各ビットの時間を複数の期間 (Time Quanta または TQ と呼ばれる) に分割し、ネットワーク上のすべてのノードが正確にデータを送受信できるようにします。ビット タイミングの設定には、次に示すいくつかの重要なコンポーネントが含まれます。タイムクォンタム (TQ)、同期セグメント (SyncSeg)、伝搬セグメント (PropSeg)、位相バッファ セグメント 1 (PBSeg1)、位相バッファ セグメント 2 (PBSeg2)、およびサンプル ポイント。データ転送におけるビットタイミングの役割は、同期と整合性、データ完全性、耐故障性、データ レートとネットワーク パフォーマンス、および干渉防止機能などの側面に反映されます。

- Desired Sampling Point (望ましいサンプリング ポイント) (%):CAN FD の場合、必要なサンプリング ポイントの割合は 15% から 95% です。このパラメータは、常にバス上の他の CAN ノードと一致します。
- Arbitration Bit Rate Configuration (アービトレーション ビット レート構成): 目的のアービトレーション レート (kbits/sec):アービトレーション レートを定義します。
- Use Calculated Arbitration Bit Timing Parameters (計算されたアービトレーション ビット タイミング パラメータの使用):この構成がイネーブルの場合、SysConfig は、目的のサンプリング ポイントとアービトレーション レートに基づいて、アービトレーション ボーレート プリスケーラ、サンプリング Pt 前の時間、サンプリング Pt 後の時間、および (Re) 同期ジャンプ幅範囲を自動的に計算します。
- Data Bit Rate Configuration (データ ビット レートの構成): 目的のデータ レート (kbits/sec):データ レートを定義します。
- Use Calculated Data Bit Timing Parameters (計算されたデータ ビット タイミング パラメータの使用):この構成がイネーブルの場合、SysConfig は、目的のサンプリング ポイントとデータ レートに基づいて、データ ボーレート プリスケーラ、サンプリング Pt 前の時間、サンプリング Pt 後の時間、および (Re) 同期ジャンプ幅範囲を自動的に計算します。

2.2.3 メッセージ RAM の構成

MCAN モ ジュールには、メッセージ RAM があります。メッセージ RAM の主な目的は、次のものを保存することです。

1. メッセージ ID フィルタ要素
2. 送信メッセージ
3. Tx イベント要素
4. 受信メッセージ

本アプリケーション ノートでは、メッセージ RAM のサイズは 1kB サイズ、幅は 32 ビットに設定されています。

注

SysConfig を使用してメッセージ RAM を構成することを TI は推奨します。SysConfig は、現在の設定で使用されているアドレス空間をユーザーに通知します。これにより、ユーザーはアドレスの重複の問題を回避できます。

2.2.3.1 標準および拡張 ID フィルタの構成

図 2-5 に、「Standard and Extended ID Filter」(標準および拡張 ID フィルタ) 構成ブロックに含まれるパラメータを示します。

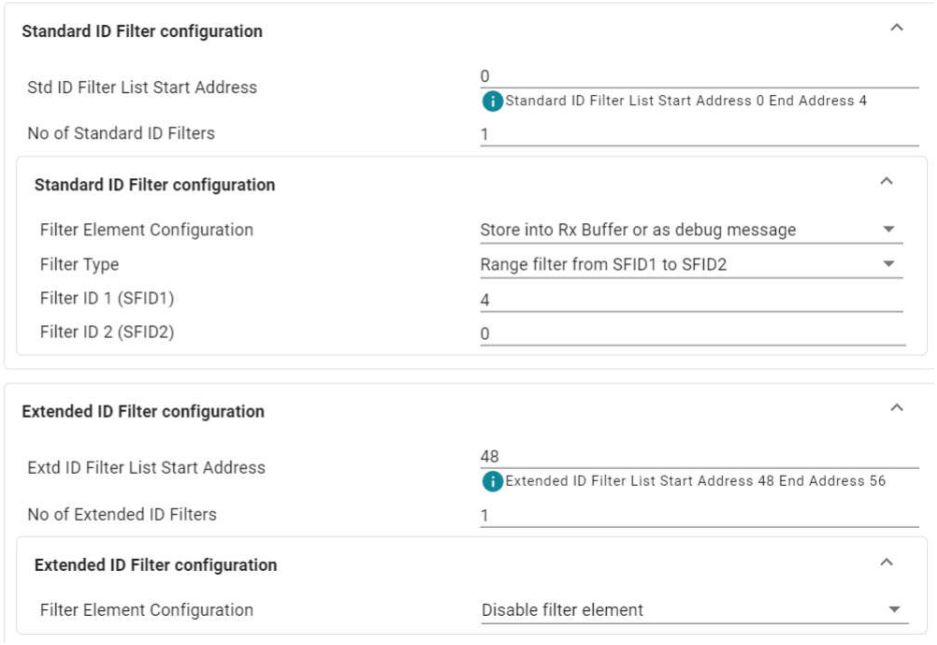

図 2-5. 標準および拡張 ID フィルタの構成

- Std ID Filter List Start Address (標準 ID フィルタ リスト開始アドレス): 標準 ID フィルタはそれぞれ 4 つのメッセージ RAM アドレスを受け取ります。
- Number of Standard ID Filters (標準 ID フィルタ数): 11 ビット標準 ID に対して最大 128 個のフィルタ要素を構成できます。SysConfig では現在、複数のフィルタの構成をサポートしていません。ユーザー アプリケーションではさらに多くのフィルタを追加できますが、初期化時に十分な RAM が割り当てられていることを確認してください。
- Standard ID Filter configuration (標準 ID フィルタ構成) → Filter Element Configuration (フィルタ要素構成): 有効なフィルタ要素はすべて、標準フレームの受け入れフィルタリングに使用されます。受け入れフィルタリングは、最初に一致する有効なフィルタ要素で、またはフィルタ リストの最後に到達すると停止します。このパラメータのオプションを以下に示します。
  - 0x0: フィルタ要素を無効化し
  - 0x1: フィルタが一致した場合は Rx FIFO 0 に格納
  - 0x2: フィルタが一致した場合は Rx FIFO 1 に格納
  - 0x3: フィルタが一致している場合は ID を拒否
  - 0x4: フィルタが一致している場合は優先度を設定
  - 0x5: フィルタが一致している場合は優先度を設定し、FIFO 0 に格納
  - 0x6: フィルタが一致している場合は優先度を設定し、FIFO 1 に格納
  - 0x7: Rx バッファに格納 (標準フィルタ タイプの構成を無視)
- Standard ID Filter configuration (標準 ID フィルタ構成) →Filter Type (フィルタ タイプ): 標準フィルタ タイプ構成を示します。このパラメータのオプションを以下に示します。
  - 0x0: SFID1 から SFID2 までの範囲フィルタ (SFID2 ≥ SFID1)
  - 0x1: SFID1 または SFID2 用デュアル ID フィルタ
  - 0x2: 従来型フィルタ: SFID1 = フィルタ、SFID2 = マスク

  - 0x3:フィルタ要素が無効化
- Standard ID Filter configuration (標準 ID フィルタ構成) → Filter ID 1 (フィルタ ID 1) (SFID1):標準フィルタ ID 1。Rx バッファをフィルタリングする場合、このフィールドは保存する標準メッセージの ID を定義します。受信した識別子は正確に一致する必要があり、マスキング メカニズムは使用されません。
- Standard ID Filter configuration (標準 ID フィルタ構成) → Filter ID 2 (フィルタ ID 2) (SFID2):標準フィルタ ID 2。この ID は、フィルタ要素の構成に応じて定義が異なります。フィルタ要素の構成が 0x1 から 0x6 の場合、SFID2 は標準 ID フィルタ要素における 2 番目の ID を表します。フィルタ要素の構成が 0x7 の場合、SFID2 は Rx バッファ用のフィルタになります。

拡張 ID フィルタの構成を以下に示します。

- Extd ID Filter List Start Address (拡張 ID フィルタ リスト開始アドレス):拡張 ID フィルタはそれぞれ 8 つのメッセージ RAM アドレスを受け取ります。
- Number of Extended ID Filters (拡張 ID フィルタ数):29 ビット拡張 ID に対して最大 64 個のフィルタ要素を構成できます。SysConfig では現在、複数のフィルタの構成をサポートしていません。ユーザー アプリケーションではさらに多くのフィルタを追加できますが、初期化時に十分な RAM が割り当てられていることを確認してください。
- Extended ID Filter configuration (拡張 ID フィルタ構成) →Filter Element Configuration (フィルタ要素構成):有効なフィルタ要素はすべて、拡張フレームの受け入れフィルタリングに使用されます。受け入れフィルタリングは、最初に一致する有効なフィルタ要素で、またはフィルタ リストの最後に到達すると停止します。
  - 0x0:フィルタ要素を無効化し
  - 0x1:フィルタが一致した場合は Rx FIFO 0 に格納
  - 0x2:フィルタが一致した場合は Rx FIFO 1 に格納
  - 0x3:フィルタが一致している場合は ID を拒否
  - 0x4:フィルタが一致している場合は優先度を設定
  - 0x5:フィルタが一致している場合は優先度を設定し、FIFO 0 に格納
  - 0x6:フィルタが一致している場合は優先度を設定し、FIFO 1 に格納
  - 0x7:Rx バッファに格納、またはデバッグ メッセージとして格納 (拡張フィルタ タイプの構成を無視)
- Extended ID Filter configuration (拡張 ID フィルタ構成) → Filter Type (フィルタ タイプ):拡張フィルタ タイプの構成を示します。このパラメータのオプションを以下に示します。
  - 0x0:EFID1 から EFID2 までの範囲フィルタ (EFID2 ≥ EFID1)
  - 0x1:EFID1 または EFID2 用デュアル ID フィルタ
  - 0x2:従来型フィルタ:EFID1 = フィルタ、EFID2 = マスク
  - 0x3:EFID1 から EFID2 (EFID2 ≥ EFID1) までの範囲フィルタで、拡張 ID およびマスクは非適用さ
- Extended ID Filter configuration (拡張 ID フィルタ構成) → Filter ID 1 (フィルタ ID 1) (EFID1):拡張フィルタ ID 1 を示します。拡張 ID フィルタ要素の最初の ID です。Rx バッファをフィルタリングする場合、このフィールドは保存する拡張メッセージの ID を定義します。
- Extended ID Filter configuration (拡張 ID フィルタ構成) → Filter ID 2 (フィルタ ID 2) (EFID2):拡張フィルタ ID 2 を示します。この ID は、拡張フィルタ要素の構成に応じて定義が異なります。拡張フィルタ要素の構成が 0x1 から 0x6 の場合、EFID2 は拡張 ID フィルタ要素における 2 番目の ID を表します。拡張フィルタ要素の構成が 0x7 の場合、EFID2 は Rx バッファ用のフィルタになります。

2.2.3.1.1 フィルタを追加する方法

SysConfig は現在、複数のフィルタの構成をサポートしていません。ユーザー アプリケーションではさらに多くのフィルタを追加できますが、初期化時に十分な RAM が割り当てられていることを確認してください。アプリケーション コードでフィルタを構成する前に、SysConfig でフィルタの数を構成することを忘れないでください。

アプリケーション コードにさらにフィルタを追加する方法の一例を以下に示します。開始アドレスは 0x0 に構成され、SysConfig ではフィルタ数は 2 に構成されます。

```
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);
```

次の例は、拡張フィルタを追加する方法を示しています。

```
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

図 2-6 に、TX MSG RAM ブロックに含まれるパラメータを示します。

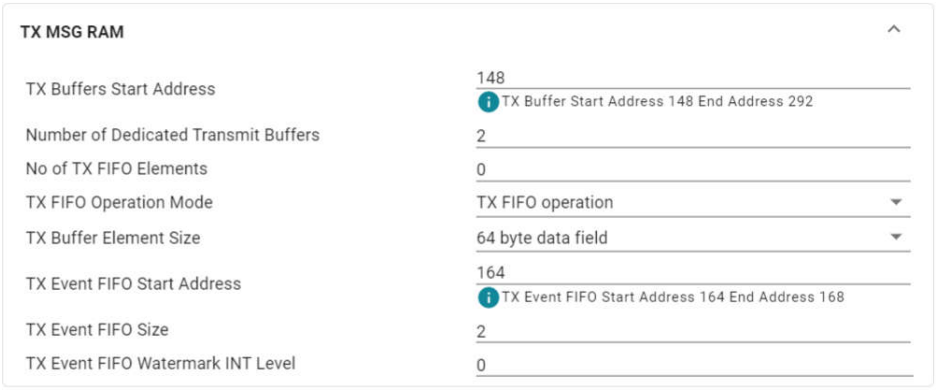

図 2-6. TX MSG RAM

Tx バッファ セクションは、Tx FIFO および Tx キューと同様に、専用の Tx バッファを保持するように構成することができます。Tx バッファ セクションが専用 Tx バッファと Tx FIFO および Tx キューで共有される場合、専用 Tx バッファは Tx バッファ セクションの先頭から開始され、それに続いて Tx FIFO または Tx キューに割り当てられたバッファが続きます。表 2-1 に、Tx バッファ モード、Tx FIFO モード、および Tx キュー モードの違いを示します。

表 2-1. Tx モード間の違い

Tx モード	説明
Tx バッファ モード	専用 Tx バッファは、ホスト CPU が完全に制御して、メッセージを転送することを目的としています。
Tx FIFO モード	Tx FIFO では、異なる Tx バッファから同じメッセージ ID を持つメッセージを、これらのメッセージが Tx FIFO に書き込まれた順序で転送できます。
Tx キュー モード	Tx キューに保存されているメッセージは、最も高い優先度のメッセージ (最も低いメッセージ ID) から開始して転送されます。

- TX Buffers Start Address (TX バッファの開始アドレス): メッセージ RAM 内の Tx バッファの開始アドレスを定義します。

- Number of Dedicated Transmit Buffers (専用送信バッファ数): 専用 Tx バッファとして構成される要素の数を定義します。
- No of TX FIFO Elements (TX FIFO 要素数): Tx FIFO または Tx キューとして構成される要素の数を定義します。
- TX FIFO Operation Mode (TX FIFO 動作モード): Tx FIFO モードまたは Tx キュー モードを定義します。
- TX Buffer Element Size (TX バッファ要素サイズ): Tx バッファ データ フィールドのサイズを定義します。Tx バッファ要素のデータ長コード DLC が Tx バッファ データ フィールド サイズより大きい値に構成されている場合、Tx バッファで定義されていないバイトは *0xCC* (パディング バイト) として送信されます。
- TX Event FIFO Start Address (TX イベント FIFO 開始アドレス): 送信されたメッセージに関する情報を Tx イベント FIFO に保存します。Tx イベント処理をサポートするため、メッセージ RAM に Tx イベント FIFO セクションが実装されています。Tx イベント FIFO を読み取ることにより、ホスト CPU はメッセージが送信された順序でこの情報を取得します。CAN バスでメッセージが転送された後、メッセージ ID とタイムスタンプは Tx イベント FIFO 要素に格納されます。Tx イベントを Tx イベント FIFO 要素にリンクするには、送信された Tx バッファからのメッセージ マーカーが Tx イベント FIFO 要素にコピーされます。
- TX Event FIFO Size (TX イベント FIFO サイズ): 最大 32 個の Tx イベント FIFO 要素を構成できます。
- TX Event FIFO Watermark INT Level (TX イベント FIFO ウォーターマーク INT レベル): Tx イベント FIFO のフィル レベル スレッショルドを定義します。Tx イベント FIFO ウォーターマークは、Tx イベント FIFO のオーバーフローを回避するように構成できます。

2.2.3.3 RX MSG RAM

図 2-7 に、RX MSG RAM ブロックに含まれるパラメータを示します。

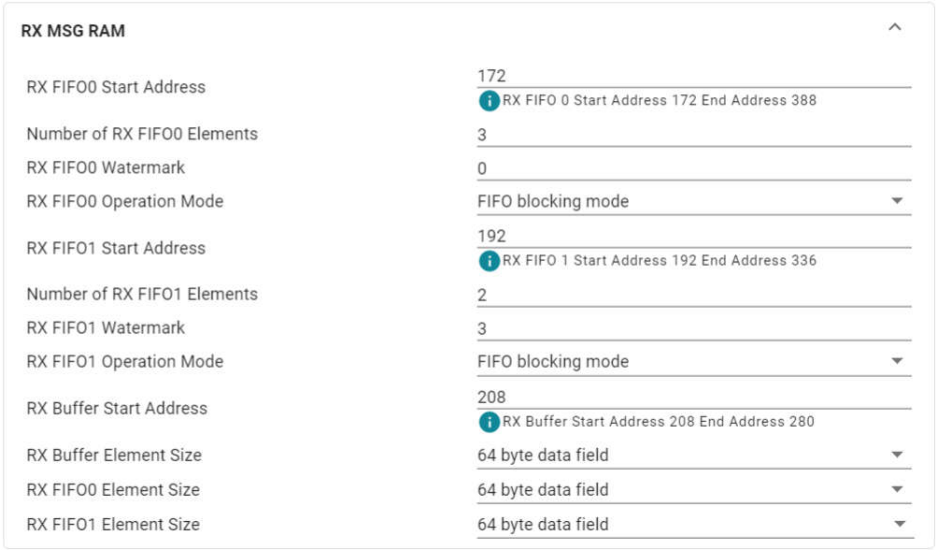

図 2-7. RX MSG RAM

メッセージ RAM には、最大 64 個の Rx バッファと 2 つの Rx FIFO を構成できます。各 Rx FIFO セクションは、最大 64 個の受信メッセージを保存するように構成できます。要素サイズは、最大 64 バイトのデータ フィールドを持つ CAN FD メッセージを保存するように構成できます。

- RX FIFO0 and RX FIFO1 Start Address (RX FIFO0 および RX FIFO1 開始アドレス): メッセージ RAM 内の Rx FIFO の開始アドレスを定義します。
- Number of RX FIFO0 and RX FIFO1 Elements (RX FIFO0 および RX FIFO1 要素の数): 各 Rx FIFO は、最大 64 個の受信メッセージを保存するように構成できます。
- RX FIFO0 and RX FIFO1 Watermark (RX FIFO0 および RX FIFO1 ウォーターマーク): Rx FIFO ウォーターマークを使用して、Rx FIFO オーバーフローを防止できます。Rx FIFO のフィル レベルが Rx FIFO ウォーターマークに達すると、割り込みフラグ MCAN_IR.RF0W/MCAN_IR.RF1W が設定されます。
- RX FIFO0 and RX FIFO1 Operation Mode (RX FIFO0 および RX FIFO1 動作モード):

  - Rx FIFO Blocking Mode (Rx FIFO ブロッキング モード): Rx FIFO ブロッキング モードは、Rx FIFO のデフォルトの動作モードです。Rx FIFO がフル状態になると、少なくとも 1 つのメッセージが読み出されて Rx FIFO ゲットインデックスがインクリメントされるまで、それ以上のメッセージは対応する Rx FIFO に書き込まれません。
  - Rx FIFO Overwrite Mode (Rx FIFO 上書きモード): Rx FIFO がフル状態に達すると、FIFO の次の受信メッセージが最も古い FIFO メッセージを上書きします。
- RX FIFO0 and RX FIFO1 Element Size (RX FIFO0 および RX FIFO1 要素サイズ): Rx FIFO 要素サイズを定義します。
- RX Buffer Start Address (RX バッファの開始アドレス): メッセージ RAM 内の Rx バッファの開始アドレスを定義します。
- RX Buffer Element Size (RX バッファ要素サイズ): Rx バッファ要素サイズを定義します。

2.3 高度な構成

図 2-8 に、「Advanced Configuration」(高度な構成) ブロックに含まれるパラメータを示します。

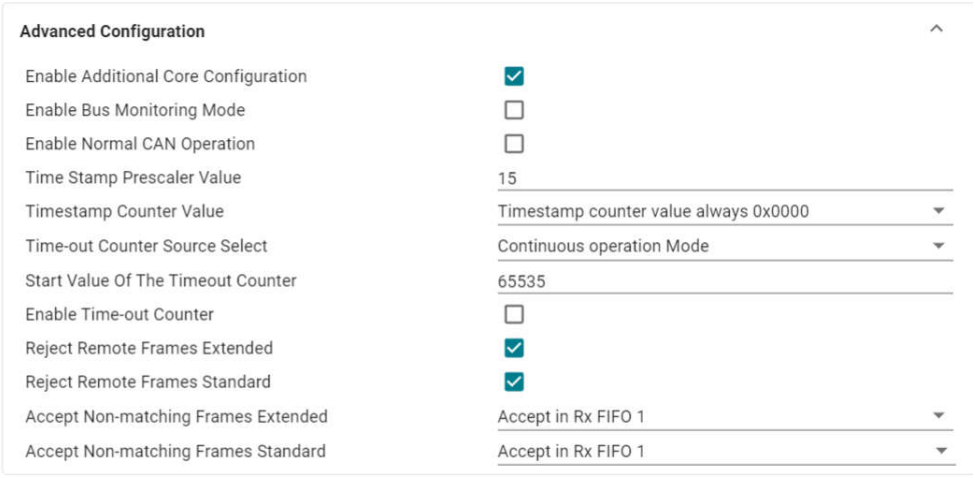

図 2-8. 高度な構成

- Enable Additional Core Configuration (追加のコア構成の有効化):タイムスタンプや一致しないフレームの処理などの追加機能を有効または無効にします。
- Enable Bus Monitoring Mode (バス監視モードの有効化):バス監視モード (「ISO 11898-1:2015」「バス監視」セクションを参照) では、MCAN モジュールは有効なデータとリモート フレームを受信できますが、転送を開始できません。MCAN モジュールは、CAN バス上でリセッシブ ビットのみを送信します。MCAN モジュールがドミナント ビット (ACK ビット、過負荷フラグ、アクティブ エラー フラグ) を送信する必要がある場合、MCAN モジュールがこのドミナント ビットを監視するように、このビットは内部で再ルーティングされます。CAN バスはリセッシブ状態を維持できます。バス監視モードを使用すると、ドミナント ビットの転送によってバスに影響を及ぼさずに、CAN バス上のトラフィックを分析できます。
- Enable Normal CAN Operation (通常 CAN 動作の有効化):通常 CAN 動作モードを制限動作モードに定義します。制限動作モードでは、CAN ノードはデータ フレームとリモート フレームを受信して、有効なフレームにアクノリッジを返すことができますが、ノードはデータ フレーム、リモート フレーム、アクティブ エラー フレーム、または過負荷フレームを送信しません。エラー状態または過負荷状態の場合、ノードはドミナント ビットを送信せず、ノードはバス アイドル条件の発生を待機して CAN 通信と再同期します。Tx ハンドラがメッセージ RAM から時間内にデータを読み取ることができない場合、制限動作モードが自動的に開始されます。制限動作モードを終了するには、ホスト CPU は MCAN_CCCR.ASM ビットをリセットする必要があります。このモードは、各種の CAN ビット レートに適応するアプリケーションで使用できます。この場合、アプリケーションはさまざまなビット レートをテストし、ノードが有効なフレームを受信すると制限動作モードを終了します。
- Time Stamp Prescaler Value (タイム スタンプ プリスケーラ値):MCAN モジュールは、タイムスタンプ生成用の 16 ビットの折り返しカウンタを内蔵しています。タイムスタンプ カウンタ プリスケーラ MCAN_TSCC.TCP フィールドについて、カウンタを CAN ビット時間 (1 ～ 16) の倍数でクロックするように設定できます。フレームの受信または転送の開始時に、カウンタ値がキャプチャされ、Rx バッファ、Rx FIFO、または Tx イベント FIFO 要素のタイムスタンプ セクションに格納されます。
- Timestamp Counter Value (タイムスタンプ カウンタ 値):タイムスタンプ カウンタ値を 0x0 に対して、内部 16 ビットカウンタから、または外部タイムスタンプから設定します。
- Time-out Counter Source Select (タイムアウト カウンタ ソース選択):MCAN モジュールには 16 ビット タイムアウト カウンタが内蔵されています。タイムアウト カウンタは、Rx FIFO 0、Rx FIFO 1、および Tx イベント FIFO メッセージ RAM 要素のタイムアウト条件を通知するために使用されます。連続モードでは、MCAN_TOCC.TOP フィールドで設定された値で、カウンタは即座に再起動されます。タイムアウト カウンタがいずれかの FIFO によって制御されている場合、空の FIFO は、MCAN_TOCC.TOP フィールドで設定された値にカウンタをプリセットします。ダウンカウントは、最初の FIFO 要素が格納されると開始されます。
- Start Value Of The Timeout Counter (タイムアウト カウンタの開始値):タイムアウト時間を定義します。
- Enable Time-out Counter (タイムアウト カウンタの有効化):タイムアウト機能を有効化します。

- Reject Remote Frames Extended (拡張リモート フレームの除去):29 ビットの拡張 ID を持つすべてのリモート フレームをフィルタまたは除去します。
- Reject Remote Frames Standard (標準リモート フレームの除去):11 ビットの標準 ID を持つすべてのリモート フレームをフィルタまたは除去します。
- Accept Non-matching Frames Extended (拡張不一致フレームの受け入れ):フィルタ リストのどの要素にも一致しない 29 ビットの ID を持つ受信メッセージの処理方法を定義します。
- Accept Non-matching Frames Standard (標準不一致フレームの受け入れ):フィルタ リストのどの要素にも一致しない 11 ビットの ID を持つ受信メッセージの処理方法を定義します。

2.4 保持構成

図 2-9 に、「Retention Configuration」(保持構成) ブロックに含まれるパラメータを示します。

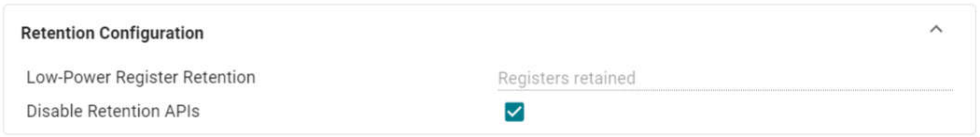

図 2-9. 保持構成

- Low-Power Register Retention (低消費電力レジスタ保持):PD1 ドメインに存在する一部の MSPM0G ペリフェラルは、STOP または STANDBY モードに移行するとき、レジスタの内容を保持しません。開発者は、アプリケーション内の SysConfig からのデフォルト初期化を使用して、ペリフェラルを再初期化することを決定できます。この方法により、メモリ効率が向上します。また、ユーザーは、提供されている DriverLib API を使用して、低消費電力モードに移行する前および後に、ペリフェラルのレジスタ構成を保存および復元することもできます。この方法は、実行時にペリフェラル構成が変更される場合に推奨されます。
- Disable Retention APIs (保持 API の無効化):このオプションを選択すると、選択したペリフェラルに関係なく保持 API が生成されません。

2.5 割り込み

図 2-10 に、Interrupts (割り込み) ブロックに含まれるパラメータを示します。

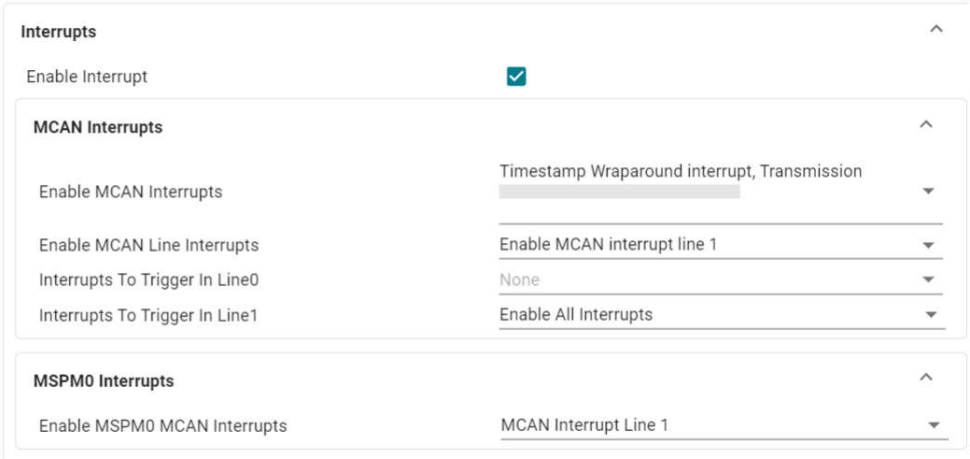

図 2-10. 割り込み

MCAN モジュールには、1 つのイベント パブリッシャ (CPU_INT) が含まれており、静的イベント ルートによって CPU サブシステムへの MCAN 割り込み要求 (IRQ) を管理します。図 2-11 に、このデバイスでの MCAN モジュールの統合を示します。

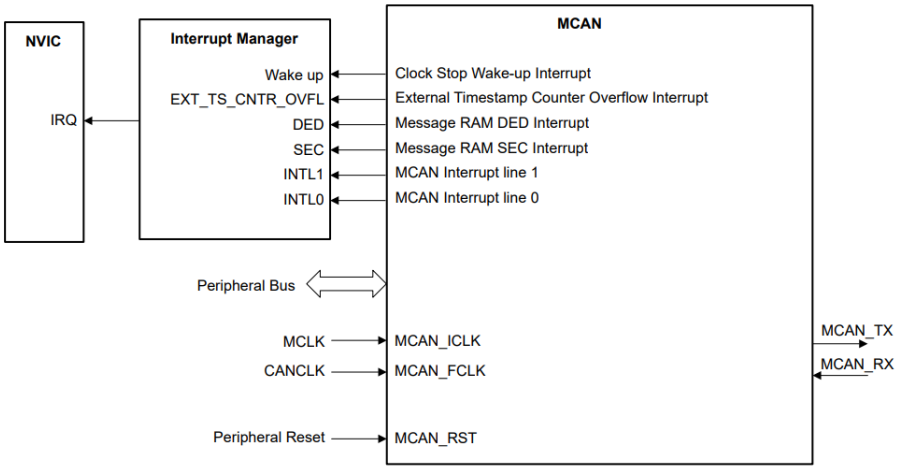

図 2-11. MCAN の統合

- MCAN 割り込み
  - Enable MCAN Interrupts (MCAN 割り込みの有効化):MCAN コアには 2 つの割り込みラインと 30 の内部割り込みソースがあります。各ソースは、2 つの割り込みラインのうち 1 つを駆動するように構成できます。MCAN コアには、2 つの割り込み要求 Interrupt Line0 と Interrupt Line1 があります。
  - Enable MCAN Line Interrupts (MCAN ライン割り込みの有効化):アプリケーションで使用する割り込みラインを定義します。
  - Interrupts To Trigger in Line0 (Line0 のトリガへの割り込み):Interrupt Line0 に割り当てられる割り込みソースを定義します。
  - Interrupts To Trigger in Line1 (Line1 のトリガへの割り込み):Interrupt Line1 に割り当てられる割り込みソースを定義します。
- MSPM0 の割り込み
  - Enable MSPM0 MCAN Interrupts (MSPM0 MCAN 割り込みの有効化):MCAN モジュールは各種の割り込みソースを備えており、CPU 割り込みイベントのソースとして構成できます。割り込みの優先度を下げるため、MCAN からの CPU 割り込みイベントはここで構成されます。

2.6 ピン構成および PinMux

図 2-12 に、「Pin Configuration」(ピン構成) および PinMux ブロックに含まれるパラメータを示します。

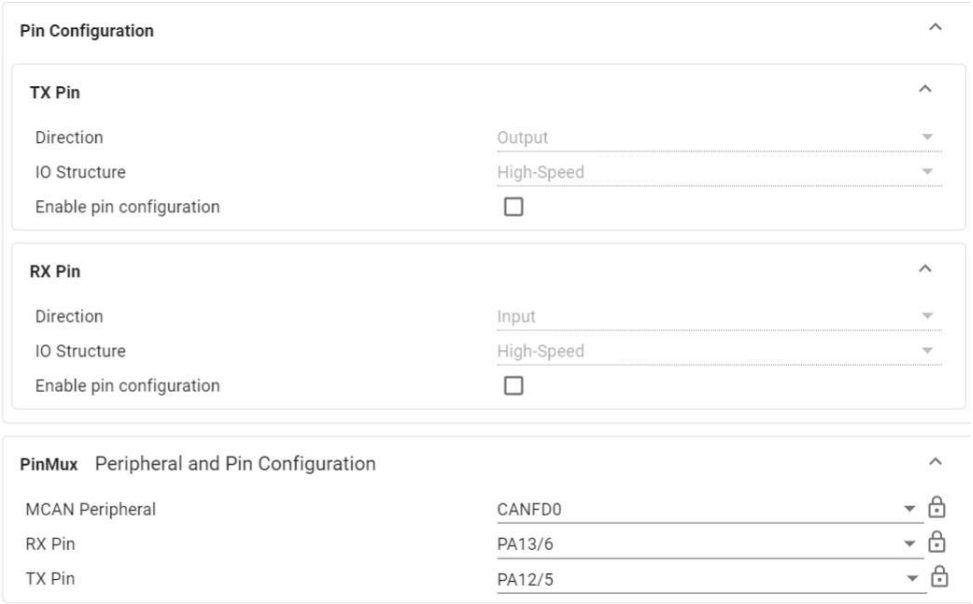

図 2-12. ピン構成および PinMux

- ピン構成
  - TX ピンと RX ピン:デジタル IOMUX 機能を専用ピンに構成します。内部プルアップまたはプルダウン抵抗、反転出力、駆動強度、高インピーダンス設定などがあります。CAN アプリケーション用に IOMUC 構成のデフォルト構成を維持することを TI は推奨します。
- Pin Mux (ピン マルチプレクサ)
  - MCAN ペリフェラル:一部の MSPM0 デバイスは複数の MCAN モジュールに統合されています。ユーザーは、ここで構成する MCAN モジュールを選択できます。
  - TX/RX ピン:MCAN TX および RX 機能の GPIO ピンを構成します。

3 デモ プロジェクトの説明

- *mcan_loopback*

次の例は、MCAN モジュールのループバック機能を示しています。ループバック動作は、モジュールの内部に完全に存在します。ただし、送信されたデータは MCANTX ピンに表示されます。このテスト ケースの利点は、トランシーバが不要なため、LaunchPad™ ボードでループバック動作を実行できることです。ロジック アナライザでのデータの分析を容易にするために、送信されるデータは 4 バイトのみです。ただし、データは CAN フレームとして送信され、ビット レート スイッチングが無効化されます。

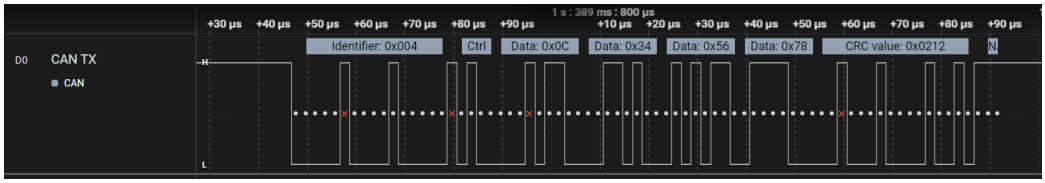

図 3-1. MCAN ループバック メッセージ

- *mcan_multi_message_tx*

この例では、複数のメッセージを送信する MCAN External Transmit (外部送信) 関数を示します。外部通信は 2 つの CAN ノード間で行われます。受信ノードには、送信されたフレームを受信または確認応答できる別の MCU または CAN バス分析ツールを使用できます。CAN トランシーバ経由で両方の CAN ノードを接続してください。この例は、mcan_multi_message_rx サンプル プロジェクトとともに使用できます。公称ビット レートの 250kbps とデータ ビット レートが使用されます。

TX メッセージは、バッファ モードとして CAN メッセージ RAM に格納されます。次に、転送 API のソフトウェア コール アド要求を使用して、目的の TX バッファでメッセージを送信します。

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

図 3-2. mcan_multi_message_tx のバス監視ツールの出力

- *mcan_multi_message_rx*

この例は、MCAN 受信機能を示しています。送信ノードには、CAN FD フレームを送信できる別の MCU または CAN バス分析ツールを使用できます。公称ビット レートの 250kbps とデータ ビット レートの 2Mbps が使用されます。標準メッセージ ID が 0x3 および 0x4 のフレームのみが受信されます。MCAN モジュールを搭載した別の MCU をトランスミッタとして使用する場合は、mcan_multi_message_tx サンプル プロジェクトを送信機能として実行できます。

- *mcan_multi_message_tx_tcan114x*

この例では、BOOSTXL-TCAN1145 BoosterPack™ を使用して複数のメッセージを送信する MCAN 外部送信機能を示します。外部通信は 2 つの CAN ノード間で行われます。受信ノードには、送信されたフレームを受信または確認応答できる別の MCU または CAN バス分析ツールを使用できます。CAN トランシーバ経由で両方の CAN ノードを接続してください。この例は、mcan_multi_message_rx_tcan114x サンプル プロジェクトとともに使用できます。公称ビット レートの 250kbps とデータ ビット レートの 2Mbps が使用されます。

ソフトウェアは最初に SPI 経由で TCAN114x モジュールを初期化します。一方、TX メッセージはバッファ モードとして CAN メッセージ RAM に格納されます。次に、転送 API のソフトウェア コール アド要求を使用して、目的の TX バッファでメッセージを送信します。

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

図 3-3. mcan_multi_message_tx_tcan114x のバス監視ツールの出力

- *mcan_multi_message_rx_tcan114x*

この例では、BOOSTXL-TCAN1145 BoosterPack を使用した MCAN 受信機能を示します。送信ノードには、CAN FD フレームを送信できる別の MCU または CAN バス分析ツールを使用できます。公称ビット レートの 250kbps とデータ ビット レートの 2Mbps が使用されます。標準メッセージ ID が 0x3 および 0x4 のフレームのみが受信されます。MCAN モジュールを搭載した別の MCU をトランスミッタとして使用する場合は、mcan_multi_message_tx_tcan114x サンプル プロジェクトを送信機能として実行できます。

- *mcan_single_message_tx*

この例では、信号メッセージを送信する MCAN 外部送信機能を示します。外部通信は 2 つの CAN ノード間で行われます。受信ノードには、送信されたフレームを受信または確認応答できる別の MCU または CAN バス分析ツールを使用できます。CAN トランシーバ経由で両方の CAN ノードを接続してください。この例は、mcan_multi_message_rx サンプル プロジェクトとともに使用できます。公称ビット レートの 250kbps とデータ ビット レートの 2Mbps が使用されます。

TX メッセージは、バッファ モードとして CAN メッセージ RAM に格納されます。次に、転送 API のソフトウェア コール アド要求を使用して、TX バッファでメッセージを送信します。

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

図 3-4. mcan_single_message_tx のバス監視ツールの出力

3.1 TX バッファ モード

TX は、SDK の現在のデモ プロジェクトにおいてバッファ モードとして構成されています。以下のコンテンツでは、バッファ モードで TX を使用する方法について説明します。

```
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);
```

まず、TX メッセージは、DL_MCAN_writeMsgRam() を呼び出すことにより、バッファ タイプのメッセージ RAM にバッファ番号で保存されます。次に、DL_MCAN_TXBufAddReq() を呼び出して、このメッセージのバッファ番号情報とともに送信する追加要求を使用します。

3.2 TX FIFO モード

TX は、SDK の現在のデモ プロジェクトにおいてバッファ モードとして構成されています。以下のセクションでは、FIFO モードで TX を使用する方法について説明します。

```
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;
}
```

まず、DL_MCAN_writeMsgRam() を呼び出すことにより、TX メッセージが FIFO タイプのメッセージ RAM に保存されます。次に、DL_MCAN_getTxFIFOQueStatus() を呼び出して、FIFO に保存されたメッセージの場所を示すこのメッセージのインデックスを取得します。その後、DL_MCAN_TXBufTransIntrEnable() を呼び出して、このメッセージの PUT インデックス情報を指定して転送割り込みを有効にします。最後に、DL_MCAN_TXBufAddReq() を呼び出して、送信するこのメッセージの PUT インデックス情報を指定して転送用の ADD 要求を呼び出します。

3.3 RX バッファ モード

RX は、SDK の現在のデモ プロジェクトにおいて FIFO モードとして構成されています。以下のセクションでは、バッファ モードで RX を使用する方法について説明します。

```
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);
```

RX バッファ モードは、フィルタ素子が *Rx バッファにストアとして、またはデバッグ メッセージとして*構成されている場合に有効化されます。まず、.sfecfeca.a.0x7 のフィルタを追加します。.sft は .sfec が 0x7 として設定されている場合は関係ありません。.sfid1 はフィルタ ID で、受信メッセージ ID はこの ID と一致しなければ受信できません。.sfid2 は、受信メッセージの格納に使用する Rx バッファを構成します。

```
/* 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);
}
```

フィルタ構成と一致するメッセージを受信すると、RX メッセージは 0 ～ 63 個の RX バッファに格納されます。まず、新しいデータ ステータスを取得します。RX バッファ 0 から RX バッファ 31 に新しいメッセージが受信されると、newDataStatus.statusLow の対応するビットが 1 に設定されます。RX バッファ 32 から RX バッファ 63 に新しいメッセージが受信されると、newDataStatus.statusHigh の対応するビットが 1 に設定されます。次に、RX バッファ 0 で新しいメッセージが受信されているかどうかを確認します。新しいメッセージがある場合は、MCAN_RX_MSG_BUFFER_INDEX の値を指定して DL_MCAN_readMsgRam() を呼び出して、RX メッセージを読み出します。この場合、RX バッファ 0 のインデックス値は 0 です。最後に、メッセージを処理した後、DL_MCAN_clearNewDataStatus() を呼び出して新しいデータ ステータスをクリアします。

3.4 RX FIFO モード

RX は、SDK の現在のデモ プロジェクトにおいて FIFO モードとして構成されています。以下のセクションでは、RX FIFO モードで一般的に使用される DL_MCAN_RxFIFOStatus の構造を紹介します。TI は、RX FIFO からメッセージを取得する際にこの構造を確認することをユーザーに推奨しています。

```
/**
 *  @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* は、現在の動作の Rx FIFO インスタンス番号を示します。MCAN は複数の Rx FIFO (Rx FIFO 0、Rx FIFO 1 など) をサポートでき、特定のインスタンスを指定する必要があります。たとえば、DL_MCAN_RX_FIFO_NUM 列挙には、DL_MCAN_RX_FIFO_0 と DL_MCAN_RX_FIFO_1 が含まれています。
- fillLvl は、Rx FIFO に現在保存されている有効なメッセージの数を示します。たとえば、fillLvl が 5 の場合、これは FIFO に 5 つの未読メッセージがあることを意味します。fillLvl が Rx FIFO の深さ (6 つのメッセージなど) に達すると、fifoFull フィールドが 1 に設定され、FIFO が満杯であることを示します。
- getIdx は、次に読み込まれるメッセージを指します。CPU は、getIdx をインクリメントして FIFO 内のメッセージを順次読み取ります。Rx FIFO が満杯で上書きモードの場合、新しいメッセージによって最も古いメッセージが上書きされます。この時点で、ユーザーは上書きされている古いデータを読み取るのを避けるために getIdx + 1 からの読み取りを開始します。
- putIdx は、次の書き込み可能なメッセージの場所を指します。新しいメッセージを受信すると、メッセージは putIdx が指す位置に書き込まれ、putIdx が増加します。
- fifoFull は、Rx FIFO がフルであるかどうかを示します。Rx FIFO がフルになったら、次のオプションを試してください。
  - ブロッキング モード：新しいメッセージを破棄し、msgLost カウントをトリガします。
  - 上書きモード：新しいメッセージが最も古いメッセージを上書きします。putIdx と getIdx は同時に増加します。
- msgLost は、Rx FIFO がフルにより廃棄されたメッセージの数をカウントします。これは、次の状況でトリガされます。
  - ブロッキングモード：Rx FIFO がフルになると、新しいメッセージを受信しても、そのメッセージを拒否、破棄します。
  - 上書きモード：古いメッセージを上書きすると、上書きされたメッセージは失われたメッセージとしてカウントされます。

4 CAN 通信の問題を解決 / 回避するためのデバッグと設計のヒント

このセクションでは、CAN バス実装時の一般的な誤りと見落としについて説明します。その後、バスの問題のトラブルシューティングに役立つデバッグのヒントを紹介します。

4.1 最低限必要なノード数

セルフ テスト モードで動作しない限り、次の理由で CAN バスに少なくとも 2 つのノードが必要です。あるノードが CAN バス上でフレームを送信するとき、ノードはネットワーク上にある少なくとも 1 つの他のノードからのアクノリッジ (受信確認) (ACK) を待ちます。CAN ノードが正常にメッセージを受信すると、その機能がオフ (*サイレントモード*) になっていない限り、ノードは ACK を自動的に送信します。(サイレント ノードとは、ノードがフレームを受信しても ACK を返さない状態を指し、これは MCAN のバス監視モードに相当します)。ACK を返すノードは、必ずしもそのフレームの本来の受信対象である必要はありませんが、そうである場合もあります。(バス上のすべてのアクティブ ノードは、ノードがそのフレームの本来の受信対象であるかどうかに関係なく、ACK を返します)。

送信ノードが ACK を受信しない場合、ACK エラーが発生し、送信ノードはフレームを永続的に再送信し続けます。送信エラー カウンタ (TEC) が 128 までインクリメントされ、そこで停止します。REC は 0 のままです。ノードはバス オフになりません。割り込みも生成されません。別のノードがネットワークに接続されると、TEC は送信が成功するたびにデクリメントを開始します (0 まで)。

4.2 トランシーバが必要な理由

ユーザーは、ノード A の MCAN_TX をノード B の MCAN_RX に直接接続したり、その逆に接続したりすることはできず、CAN 通信が正常に行われることが予想されます。この場合、CAN は UART や SPI など他のシリアル インターフェイスとは異なります。たとえば、UART は RS232 トランシーバまたは直接接続 (1 つのノードの UART_TX から別のノードの UART_RX へ、またはその逆へ) 経由で動作できます。ただし、CAN バスには、次の理由により CAN トランシーバが必要です。シングル エンド CAN 信号を差動転送用に変換するだけでなく、トランシーバは CAN_TX ピンをノードの CAN_RX ピンにループバックします。これは、CAN ノードが転送を監視できる必要があるからです。

- これは、CAN プロトコルで要求されている ACK 要件に対応する必要があります。あるノードが CAN バス上でフレームを送信するとき、ノードはネットワーク上にある少なくとも 1 つの他のノードからの ACK を待ちます。ACK 位相の場合、トランスミッタは 1 を出力し、0 の読み戻しを待ちます。
- アービトレーション中、優先度の高いメッセージ ID を持つノードは、0 で 1 を無効にできる必要があります。この場合、トランスミッタは送信したデータを読み戻せる必要があります。アービトレーション位相中にノードが 1 を出力し、0 を読み戻すと、ノードはアービトレーションを失います。

トランシーバのコストを節約するために、(すべてのノードが同じ PCB 上にあり、近接した場所にある) 一部のアプリケーションでは、ダイオードなどのディスクリート部品を使用して、CAN ノードで転送を監視するという要件を満たすことができます。

4.3 バス オフ ステータス

バス オフ状態は、重大なバスエラーのために CAN ノードが強制的に絶縁された場合に発生します。この状態では、ノードはデータの送受信を停止し、実質的にバスとの接続を切断します。このメカニズムにより、バスが永続的なエラーから保護され、故障伝搬が防止されます。

1 次トリガは、TEC がスレッショルド (通常 255) を超えたときです。一般的なシナリオは次のとおりです。

- ハードウェア障害:CAN コントローラまたはトランシーバの破損。
- 物理層の問題:開放または短絡、不適切な終端抵抗、または信号干渉 (強い EMI など)。
- 構成エラー:アービトレーション エラーまたはビット エラーの原因となるボーレートの不一致。
- ソフトウェア エラー:不正なフレームの繰り返し送信。

たとえば、転送エラーのたびに TEC は 1 ずつインクリメントされます。エラーが急速に蓄積され、TEC が 255 を超えると、ノードはバス オフ状態になります。

このデバイスは、次のいずれかのアクションを実行することで、バス オフ状態から回復できます。

- ハードウェア リセット: CAN コントローラをリセットして (ソフトウェア リセットまたは電源の再投入によって)、エラー カウンタをクリアします。
- 回復シーケンスの受信: ノードを アクティブなエラー状態に自動的に復元する 11 のリセッシブ ビット (バス アイドル信号) の 129 の連続シーケンスを検出します。これには、十分なバス アイドル時間 (合計期間 = 129 × 11 ビット) が必要です。

以下に、まず SysConfig で Bus_Off ステータス割り込みを有効化してバス オフ ステータスを検出する方法の一例を示します。バス オフ ステータスが変化すると、MCAN は Bus_Off ステータス割り込みをトリガして、この状況をユーザーに通知します。ユーザーは、バス オフ状態が割り込みルーチンであるかどうかを確認する必要があります。

```
/**
 * 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;
    }
}
```

次に、メイン ループの CANBusOff フラグを確認します。バス オフ状態が検出されたら、MCAN モジュールを再起動します。

```
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 低消費電力モードでの MCAN の使用

ユーザーは、アプリケーションの要件を満たすために、マイコンを低消費電力モードにする必要があります。ただし、MCAN モジュールは低消費電力モードではディスエーブルになっています。回避方法として、低消費電力モードに移行する前に、ユーザーは MCAN RX ピンを入力ピンとして構成できます。そのピンでエッジ障害割り込みを有効化します。MCAN の RX ピンで 1 つのメッセージが受信されると、マイコンはエッジ障害割り込みによってウェークアップされます。次に、MCAN RX ピンを MCAN 機能として再構成し、MCAN モジュールを再構成します。MCAN は、この方法で通常の機能に復元します。

注

最初の MCAN メッセージは、マイコンのウェークアップ信号として使用されます。このウェークアップ機能のテスト メッセージを使用します。

コード例を以下に示します。

```
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 デバッグ チェックリスト

このセクションでは、いくつかの一般的な誤りを紹介し、デバッグの便利なヒントを提供します。

4.5.1 プログラミングの問題

- MCAN モジュールへのクロックは有効になっていますか。SysConfig で CANCLK 構成を確認してください。エラー率を下げるため、CANCLK ソースとして外部水晶振動子を使用することを TI は推奨します。
- 最初に割り込みなしでコードを試してみてください。代わりにポーリングを使用します。ポーリングが機能すると、ユーザーは後で割り込みを追加できます。
- 最初にバス上で通信を開始しようとする場合は、送信ノードと受信ノードのメールボックスが同じメッセージ ID でプログラムされていることを確認してください。最初はフィルタ機能を使用しないでください。ハードウェアの問題が確認されない場合は、後でフィルタリングを追加できます。

4.5.2 物理層の問題

- バスは両端でのみ (120Ω で) 正しく終端されていますか。バスは両端でのみ 120Ω の抵抗で終端する必要があります。バス上には終端抵抗を 2 個までしか設置できません。ただし、分割終端方式を採用する場合は、両端にそれぞれ 2 個の抵抗を配置します。CAN バス システムの設計時に、終端抵抗をシステムの筐体の外部からイネーブルまたはディスエーブルにできます。この方式は、ノードをネットワークに追加またはネットワークから削除する必要がある場合に簡単に使用できます。
- すべての CAN ノードが同じビット レートに構成されていますか。ノードのビット レートが一致していないと、バス上にエラー フレームが繰り返し発生します。オシロスコープで CAN_TX ピンの出力をキャプチャして、ビット時間を物理的に確認してください。
- ユーザーはビット レートを低くしてみましたか。たとえば、50kbps などです。伝搬遅延に関するタイミングの問題は、より低いビットレートを試すことで検出できる可能性があります。ビット タイミング パラメータが SysConfig で正しく設定されていることを確認してください。
- ユーザーはバスの長さとノード数を減らしてみましたか。
- エラー状態が発生する前に、バスに任意のエラー フレームが見られましたか。タイミング違反やノイズの問題が発生する可能性があります。
- バス内にノードはいくつありますか。(非自己テスト モードでは、CAN プロトコルで要求されているアクノリッジ (ACK) 要件により、ネットワーク上に少なくとも 2 つのノードが存在する必要があります)。

4.5.3 ハードウェアのデバッグのヒント

- ACK 位相までの波形を確認するには、トランシーバをノードに接続する必要があります。トランシーバがないと、ノードはただちにエラー状態になります。
- CAN フレームが送信 MCU の MCAN_TX ピンで正しく認識され、予期されるビットレートであるかどうかを確認してください。予期されるデータが MCAN_TX ピンに見られる場合は、MCAN_RX ピンのデータを確認してください。MCAN_RX ピンに同じデータが見られる場合、トランシーバはデータを正しくループ バックしています。
- CAN FD トリガを内蔵したオシロスコープを使用する場合、トリガするように構成された信号がボード上でプローブされている信号と一致していることを確認します。多くのオシロスコープは、Start-Of_Frame (SOF)、リモート フレーム、エラー フレーム、および特定のメッセージ ID に加えて、CAN 送信 (CANTX)、CAN 受信 (CANRX)、CAN_H、および CAN_L 信号をトリガできます。
- スコープが波形をデコードしない場合は、チャネルの入力スレッショルド値が正しいことを確認してください。これは、信号に通常使用される*トリガ レベル*に似ています。
- 公称位相とデータ位相の両方のビットレートが、オシロスコープで正しく構成されていることを確認してください。それ以外の場合は、誤ったデータが表示されます。
- CAN バス アナライザ ツール : 公称位相とデータ位相の両方のビットレートが正しく構成されていることを確認します。

5 まとめ

このドキュメントでは最初に、CAN バスの概要と MSPM0 マイコンに内蔵された MCAN モジュールの機能を記載しています。このアプリケーション ノートでは、SysConfig を使用して CAN モジュールを構成する方法について詳しく説明しているため、ユーザーはアプリケーションの要件に合わせて関連パラメータを調整できます。このドキュメントでは、MSPM0 SDK に付属する MCAN モジュールに関連するデモの例について簡単に説明します。最後に、ソフトウェアとハードウェアの側面から、一般的なデバッグの問題と関連するデバッグの提案をいくつか紹介します。

6 参考資料

- テキサス インスツルメンツ、『*MSPM0 G シリーズ 80MHz マイコン*』テクニカル リファレンス マニュアル
- テキサス インスツルメンツ、『*MSPM0-SDK*』
- テキサス インスツルメンツ、『*コントローラ エリア ネットワーク (CAN) の概要*』アプリケーション ノート
- テキサス インスツルメンツ、『*コントローラ エリア ネットワークの物理層要件*』アプリケーション ノート
- テキサス インスツルメンツ、『*コントローラ エリア ネットワーク (CAN) の物理層のデバッグの基礎*』アナログ設計ジャーナル
- テキサス インスツルメンツ、『*CAN ビット タイミング パラメータ用のカリキュレータ*』アプリケーション ノート
- テキサス インスツルメンツ、『*3.3V CAN (コントローラ エリア ネットワーク) トランシーバの概要*』アプリケーション ノート
- テキサス インスツルメンツ、『*チョークレス トランシーバによる CAN バスの実装の簡素化*』マーケティング ホワイト ペーパー

- テキサス インスツルメンツ、『CAN バス接続の重要な間隔』アプリケーション ノート
- テキサス インスツルメンツ、『CAN バスでのメッセージ優先反転』アナログ設計ジャーナル

重要なお知らせと免責事項

テキサス・インスツルメンツは、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、テキサス・インスツルメンツ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 テキサス・インスツルメンツ製品の選定、(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 テキサス・インスツルメンツ製品を使用するアプリケーションの開発の目的でのみ、テキサス・インスツルメンツ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。テキサス・インスツルメンツ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、テキサス・インスツルメンツおよびその代理人を完全に補償するものとし、テキサス・インスツルメンツは一切の責任を拒否します。

テキサス・インスツルメンツの製品は、テキサス・インスツルメンツの販売条件、または ti.com やかかる テキサス・インスツルメンツ製品の関連資料などのいずれかを通じて提供する適用可能な条項の下で提供されています。テキサス・インスツルメンツがこれらのリソースを提供することは、適用されるテキサス・インスツルメンツ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。

お客様がいかなる追加条項または代替条項を提案した場合でも、テキサス・インスツルメンツはそれらに異議を唱え、拒否します。

郵送先住所:Texas Instruments, Post Office Box 655303, Dallas, Texas 75265

重要なお知らせと免責事項

テキサス・インスツルメンツは、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、 テキサス・インスツルメンツ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 テキサス・インスツルメンツ製品の選定、(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 テキサス・インスツルメンツ製品を使用するアプリケーションの開発の目的でのみ、 テキサス・インスツルメンツ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。 テキサス・インスツルメンツ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、 テキサス・インスツルメンツおよびその代理人を完全に補償するものとし、 テキサス・インスツルメンツは一切の責任を拒否します。

テキサス・インスツルメンツの製品は、 テキサス・インスツルメンツの販売条件、または ti.com やかかる テキサス・インスツルメンツ製品の関連資料などのいずれかを通じて提供する適用可能な条項の下で提供されています。 テキサス・インスツルメンツがこれらのリソースを提供することは、適用される テキサス・インスツルメンツ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。

お客様がいかなる追加条項または代替条項を提案した場合でも、 テキサス・インスツルメンツはそれらに異議を唱え、拒否します。

郵送先住所：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265