参考資料

Application Note

MSPM0 マイコンの MCAN (CAN FD) モジュール入門

TEXAS INSTRUMENTS

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

1 CAN の概要 ... 3
1.1 CAN 規格 ... 3
1.2 MSPM0 MCAN モジュール ... 5
2 MCAN モジュールの SysConfig 構成 ... 6
2.1 MCAN クロック周波数 ... 6
2.2 MCAN の基本構成 ... 6
2.3 高度な構成 ... 18
2.4 保持構成 ... 19
2.5 割り込み ... 19
2.6 ピン構成および PinMux ... 21
3 SDK サンプル コードの紹介 ... 22
3.1 mcan_loopback ... 22
3.2 mcan_message_rx ... 22
3.3 mcan_message_rx_tcan114x ... 22
3.4 mcan_multi_message_tx ... 22
3.5 mcan_multi_message_tx_tcan114x ... 23
3.6 mcan_single_message_tx ... 23
4 MCAN ユースケース ... 24
4.1 TX バッファ素子 ... 24
4.2 RX バッファ素子 ... 25
4.3 TX 処理バッファ モード ... 27
4.4 TX 処理 FIFO モード ... 27
4.5 RX 処理バッファ モード ... 29
4.6 RX 処理 FIFO モード ... 31
4.7 低消費電力モードでの MSPM0 MCAN の使用 ... 33
4.8 バス オフ対応 ... 34

Copyright © 2026 Texas Instruments Incorporated

5 設計テストとデバッグのヒント.....36
5.1 最低限必要なノード数.....36
5.2 トランシーバが必要な理由.....36
5.3 デバッグ チェックリスト.....36
6 まとめ.....37
7 参考資料.....37
8 改訂履歴.....39

図の一覧

図 1-1. 代表的な CAN バス.....3
図 1-2. CAN FD フレーム.....3
図 1-3. ビット時間の構成.....4
図 2-1. MCAN クロック周波数.....6
図 2-2. MCAN の基本構成.....6
図 2-3. トランスミッタ遅延補償 (TDC).....8
図 2-4. ビット タイミング パラメータ.....9
図 2-5. 標準および拡張 ID フィルタの構成.....12
図 2-6. TX MSG RAM.....15
図 2-7. RX MSG RAM.....16
図 2-8. 高度な構成.....18
図 2-9. 保持構成.....19
図 2-10. MCAN の統合.....19
図 2-11. 割り込み.....20
図 2-12. ピン構成および PinMux.....21
図 3-1. MCAN ループバック メッセージ.....22
図 3-2. mcan_multi_message_tx のバス監視ツールの出力.....22
図 3-3. mcan_multi_message_tx_tcan114x のバス監視ツールの出力.....23
図 3-4. mcan_single_message_tx のバス監視ツールの出力.....23

商標

MSPM0™, Code Composer Studio™, LaunchPad™, and BoosterPack™ are trademarks of Texas Instruments.
すべての商標は、それぞれの所有者に帰属します。

Copyright © 2026 Texas Instruments Incorporated

1 CAN の概要

このセクションでは、CAN 規格および MSPM0 MCAN モジュールについて説明します。

1.1 CAN 規格

このセクションでは、CAN バスとフレームの構造、CAN ビットのタイミングについて説明します。

CAN 標準仕様の詳細については、公式ドキュメントを参照してください。

1.1.1 CAN バス

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。

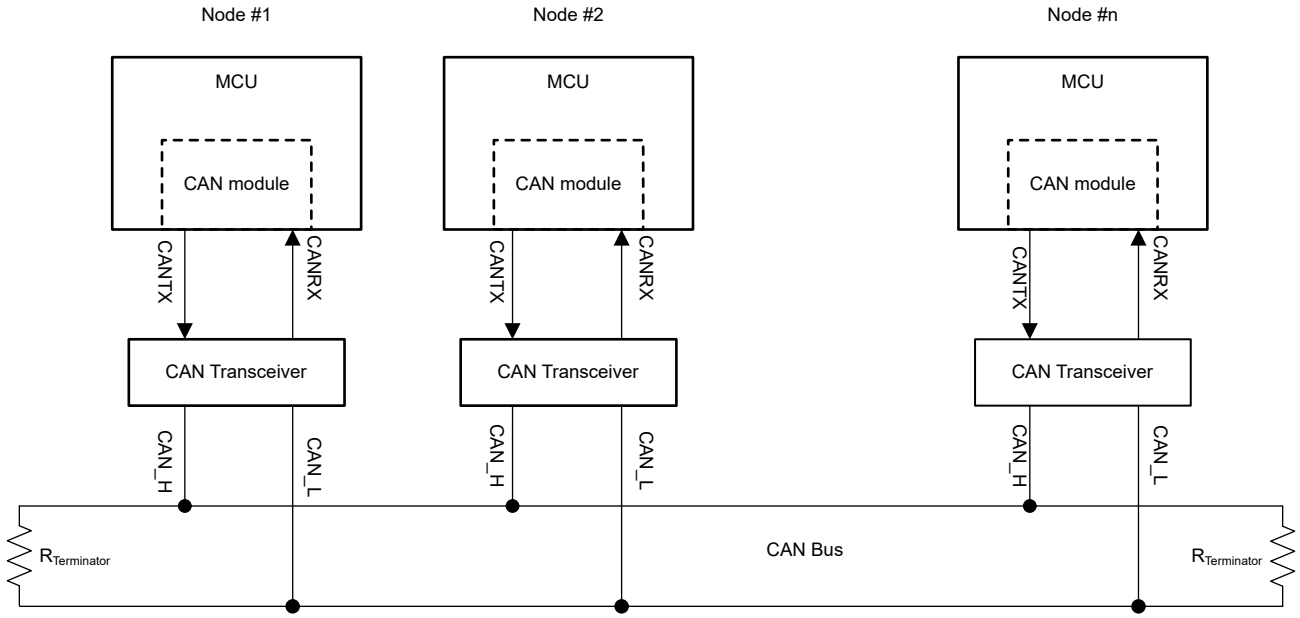


図 1-1. 代表的な CAN バス

1.1.2 CAN フレーム

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。

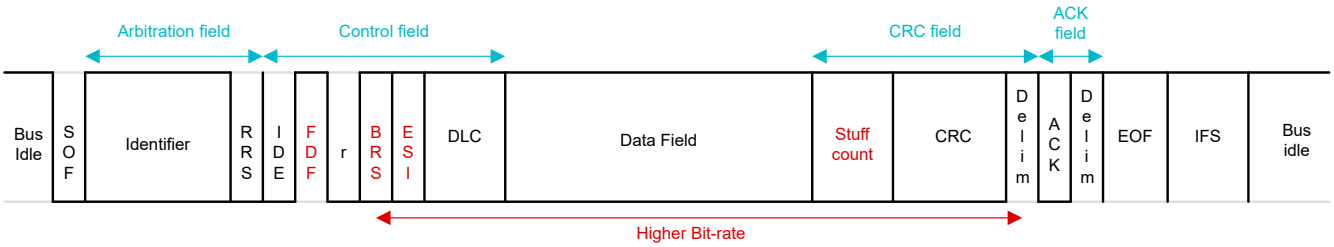


図 1-2. CAN FD フレーム

Copyright © 2026 Texas Instruments Incorporated

1.1.3 CAN ビットのタイミング

CAN ビット レートは、理想的なトランスミッタによる再同期がない場合に送信された 1 秒あたりのビット数です。

CAN-FD プロトコルでは、2 つのビット レートが使用され、低速のアービトレーション フェーズではアービトレーション ビット レートが使用され、データ ビット レートは高速のデータ フェーズで使用されます。

アービトレーションとデータ ビット レートはどちらも、重なりのない別の時間セグメントに分割できます。

- 同期セグメント (Sync_Seg):Sync_Seg は、バス上のさまざまなノードを同期するために使用されます。エッジがこのセグメント内にあることが想定されます。
- 伝播セグメント (Prop_Seg):Prop_Seg は、ネットワーク内の物理的な遅延時間を補償するために使用されます。Prog_Seg = 2*(信号の伝搬 + 入力コンパレータの遅延 + 出力ドライバの遅延)。
- 位相バッファ セグメント 1 (Phase_Seg1)
- 位相バッファ セグメント 2 (Phase_Seg2):Phase_Seg1 と Phase_Seg2 が、エッジ位相誤差を補償するため使用されます。これらのセグメントは、再同期化によって延長または短縮できます。

これらのセグメントは、図 1-3 に示すように、ビット時間を形成します。

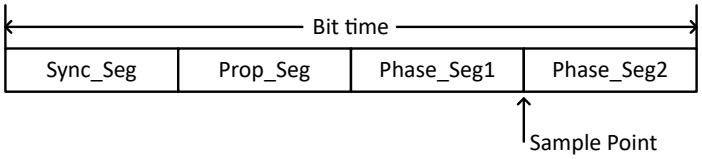


図 1-3. ビット時間の構成

その他の関連する CAN のビット タイミングを、以下に示します。

- サンプル ポイント:サンプル ポイントは、バス レベルが読み取られ、それぞれのビットの値として解釈されるポイントです。これの場所は、Phase_Seg1 の最後にあります。
- 時間量子 (tq):「*tq*」は発振器周期から導出される時間の固定単位です。プログラム可能なプリスケーラがあり、整数値は 1 ～ 32 の範囲です。最小時間量子から始めると、時間量子の長さは「tq = BRP * 最小時間量子」となり、BRP はビット レート プリスケーラの値、最小時間量子は通常、CAN モジュールのクロック ソースです。

Copyright © 2026 Texas Instruments Incorporated

1.2 MSPM0 MCAN モジュール

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

Copyright © 2026 Texas Instruments Incorporated

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、SysConfig ツール 1.25.0 を使用し、LP-MSPM0G3507 を例として、SysConfig で 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。

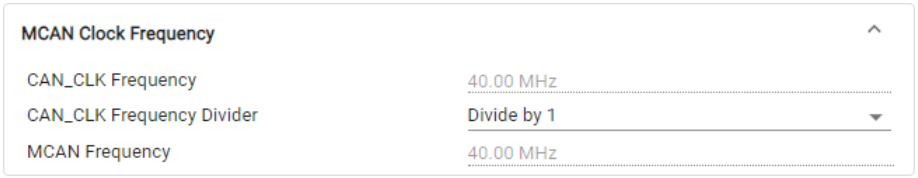


図 2-1. MCAN クロック周波数

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。

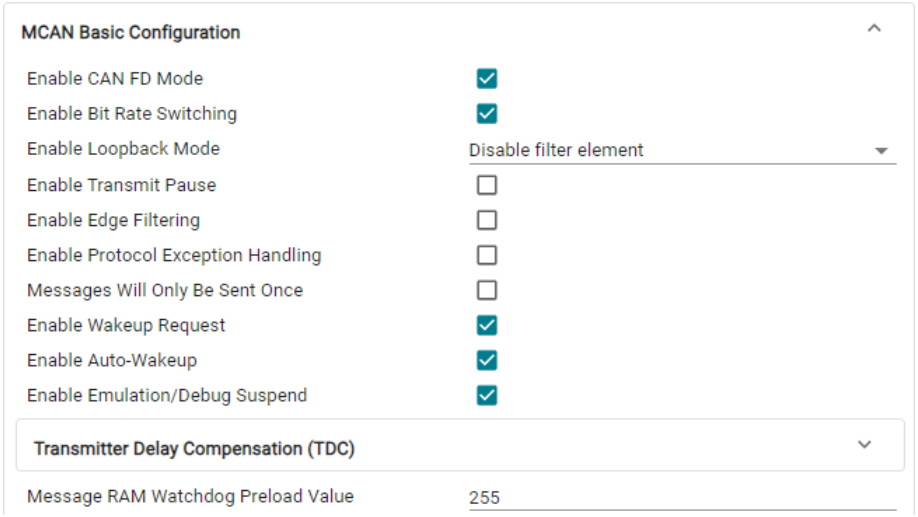


図 2-2. MCAN の基本構成

Copyright © 2026 Texas Instruments Incorporated

- 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。MSPM0 MCAN モジュールの Tx メッセージでは、ユーザーはソース コード内で各メッセージのビット レート スイッチングを制御できます。
- 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。

- エミュレーションまたはデバッグの中断の有効化: MCAN モジュールは、エミュレーションまたはデバッグのために中断できます。
- 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。

Copyright © 2026 Texas Instruments Incorporated

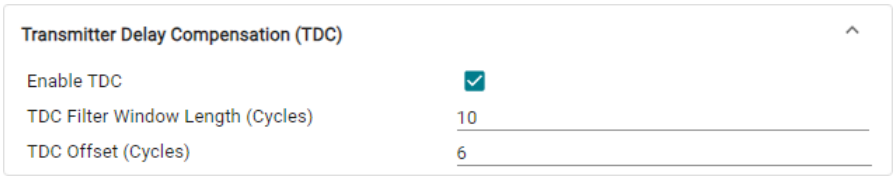


図 2-3. トランスミッタ遅延補償 (TDC)

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。

Copyright © 2026 Texas Instruments Incorporated

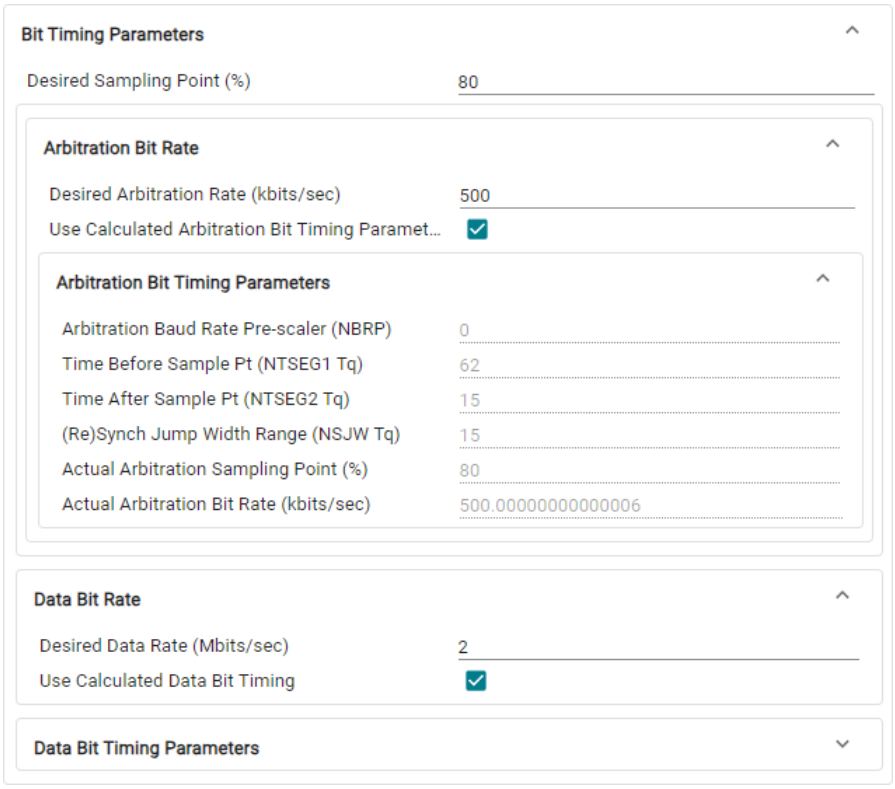


図 2-4. ビット タイミング パラメータ

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 (時間量子 (Time quantum: 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、時間量子 (tq)、同期セグメント (Sync_Seg)、伝搬セグメント (Prop_Seg)、位相バッファ セグメント 1 (Phase_Seg1)、位相バッファ セグメント 2 (Phase_Seg2)、およびサンプル ポイントといった、いくつかの重要な構成要素が含まれています。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。これらのセグメントの詳細な説明については、CAN ビットのタイミングを参照してください。

図 2-4 に各パラメータの説明を示します。

- 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。

CAN ビット タイミングの名前は、ホスト コンピュータの CAN アナライザ ソフトウェア、MCAL の EB tresos、およびマイコンのドキュメントなど、常に異なるシステム間で変更されます。MSPM0 における CAN ビット タイミングの命名の差異や、

Copyright © 2026 Texas Instruments Incorporated

MSPM0 システム構成ツールと一般的な CAN 仕様書との間での対応関係については、表 2-1 を参照することで迅速に把握できます。表 2-1 では、x が付いた名称は、x が N または D を表すことを意味します。たとえば、xBRP は NBRP (アービトレーション ボーレート プリスケーラ) または DBRP (データ ボーレート プリスケーラ) のいずれかを表します。MSPM0 レジスタの詳細については、

表 2-1. ビット タイミングの命名

共通の CAN	SysConfig	MSPM0 レジスタ
tq	MCAN 周波数 / xBRP	MCAN_FCLK/xBRP
Sync_Seg	該当なし	1 で固定し
Prop_Seg	xTSEG1	
Phase_Seg1		
Phase_Seg2	xTSEG2	
SJW		

MSPM0 SysConfig およびレジスタ レベルでは、MSPM0 の Sync_Seg はハードウェア上では 1 つの tq に固定されており、xTSEG1 は共通 CAN の 2 つのセグメント Prop_Seg と Phase_Seg1 と結合されます。SysConfig のビット時間の合計 tq 数は、「tq カウント = 1 + xTSEG1 + xTSEG2」です。サンプリング ポイントは xTSEG1 と xTSEG2 の間です。

Syscfg ビットのタイミング構成例に、共通 CAN と SysConfig、MSPM0 レジスタとの間の CAN ビット タイミング構成の簡単な比較 / マッピング表を示します。MSPM0 MCAN モジュールのビット タイミングは、ビットレート スイッチング イネーブル (500kbit/s + 2Mbit/s)、サンプリング 80% を備えた CAN-FD に設定されています。クロック ソース MCAN_FCLK は分周なしで 40MHz です。

表 2-2. Syscfg ビットのタイミング構成例

	調停		データ		
ビット長	80tq = 2us		20tq = 0.5us		
ビット レート	0.5MBit/s		2MBit/s		
クロック ソース	40MHz				
時間量子 (tq)	25ns				
共通 CAN パラメータ			共通 CAN パラメータ		
ビット時間パラメータ	調停	データ	ビット時間パラメータ	調停	データ
該当なし			xBRP	0	0
Sync_Seg	1	1	ハードウェアは 1 で固定		
Prop_Seg	63	15	xTSEG1	62	14
Phase_Seg1					
Phase_Seg2	16	4	xTSEG2	15	3
SJW	16	4	xSJW	15	3

Copyright © 2026 Texas Instruments Incorporated

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、複数のフィルタ、送信、および受信メッセージを保存することで、MCAN メッセージ RAM に複数のセクションを設定でき、対応する開始アドレスと終了アドレスを互いに比較することにより、どのセクションも重複しないようにすることです。

1. メッセージ ID フィルタ要素:標準 11 ビット フィルタ、拡張 29 ビット フィルタ。
2. 送信メッセージ:TX バッファ、TX FIFO、および TX イベント FIFO。
3. 受信メッセージ:Rx FIFO0、Rx FIFO1、Rx バッファ。

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。

MCAN メッセージ RAM の構成例に MCAN メッセージ RAM の構成方法の例を示します。サンプル プロジェクトについては MCAN モジュールの SysConfig 構成を参照してください。各構成に関する詳細情報は、本セクションで紹介しています。

表 2-3. MCAN メッセージ RAM の構成例

セクション	サイズ	個数	開始アドレス	終了アドレス
標準 ID フィルタ	4	4	0	16
拡張 ID フィルタ	8	4	16	48
TX バッファ	72	0	該当なし	
TX FIFO	72	4	48	336
TX イベント FIFO	8	4	336	368
RX FIFO0	72	6	368	800
RX FIFO1	72	2	800	944
RX バッファ	72	1	944	1016

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。

Copyright © 2026 Texas Instruments Incorporated

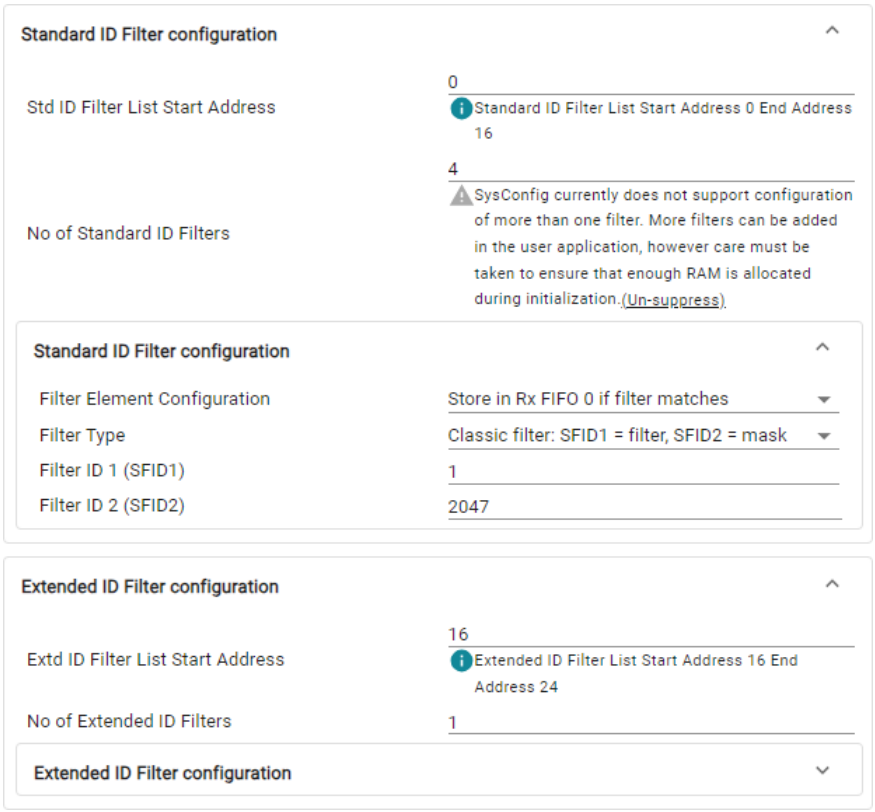


図 2-5. 標準および拡張 ID フィルタの構成

- 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、およびアドレスの重複が発生していない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

Copyright © 2026 Texas Instruments Incorporated

- Standard ID Filter configuration (標準 ID フィルタ構成) → Filter ID 1 (フィルタ ID 1) (SFID1): Standard Filter ID 1 (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): Standard Filter ID 2 (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 複数のフィルタを構成する方法

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。

以下の例は、アプリケーション コード内で標準フィルタを追加する方法を示しています。フィルタ要素「*gMCAN0StdFltWr*」は、標準フィルタ「*sfid1*」とマスク「*sfid2*」を使用するように構成され、一致したメッセージを FIFO0 に格納します。次に、「*DL_MCAN_addStdMsgIDFilter*」を使用して、「*gMCAN0StdFltWr*」を 2 番目のフィルタの RAM アドレス (*filterIndex* = 1) に書き込みます (通常、最初のフィルタ (*filterIndex* = 0) は SysConfig で構成できます)。

```
uint8_t filterIndex = 1U;
static const DL_MCAN_StdMsgIDFilterElement gMCAN0StdFltWr = {
    .sfec  = 0x1,    // Save to FIFO0
    .sft   = 0x2,    // Filter sfid1 with mask sfid2
    .sfid1 = 3,
```


Copyright © 2026 Texas Instruments Incorporated

```
    .sfid2 = 0x7FF,
};
/* Configure Standard ID filter element */
DL_MCAN_addStdMsgIDFilter(MCAN0_INST, filterIndex, &gMCAN0StdFltWr);
```

以下の例は、アプリケーション コード内で拡張フィルタを追加する方法を示しています。フィルタ要素「*gMCAN0ExtFltWr*」は、フィルタ id efid1 と efid2 の間の範囲 (たとえば、0x10 ～ 0x20) に一致するよう構成にされています。たとえば、id = 0x15 の拡張メッセージが一致し、一致したメッセージは FIFO0 に保存されます。次に、「*DL_MCAN_addExtMsgIDFilter*」を使用して、「*gMCAN0ExtFltWr*」を 4 番目のフィルタの RAM アドレス (*filterIndex* = 3) に書き込みます。

```
uint8_t filterIndex = 3U;
static const DL_MCAN_ExtMsgIDFilterElement gMCAN0ExtFltWr = {
    .efec = 0x1,    // Save to FIFO0
    .eft  = 0x0,    // Range filter from efid1 to efid2
    .efid1 = 0x10,
    .efid2 = 0x20,
};
/* Configure Extended ID filter element */
DL_MCAN_addExtMsgIDFilter(MCAN0_INST, filterIndex, &gMCAN0ExtFltWr);
```

構成したフィルタを読み取って確認するには、以下の API を追加して、メッセージ RAM 内のフィルタを MSPM0 の RAM 変数に読み戻すことができます。以下のコードには、標準フィルタと拡張フィルタの両方を読み取る関数が含まれています。

「*DL_MCAN_readStdMsgIDFilter*」API を呼び出すことにより、CPU はインデックス「*filtNum*」を含むメッセージ RAM フィルタ要素を読み取り、フィルタ値を「*gMCAN0StdFltRd*」にコピーします。

```
uint32_t filtNum = 2;
DL_MCAN_StdMsgIDFilterElement gMCAN0StdFltRd;
void DL_MCAN_readStdMsgIDFilter(MCAN_Regs *mcan, uint32_t filtNum,
    DL_MCAN_StdMsgIDFilterElement *elem)
{
    uint32_t startAddr, elemAddr, regVal;

    //startAddr = HW_RD_FIELD32(&mcan->MCANSS.MCAN.MCAN_SIDFC, MCAN_SIDFC_FLSSA);
    startAddr = *(volatile uint32_t *)(&mcan->MCANSS.MCAN.MCAN_SIDFC);
    startAddr = (startAddr & 0x0000FFFC) >> 2;
    startAddr = (uint32_t)(startAddr << 2U);
    elemAddr  = startAddr + (filtNum * 1U * 4U);
    elemAddr += 0x0U;

    //regVal = HW_RD_REG32(((uint32_t) mcan + (uint32_t) elemAddr));
    regVal = *(volatile uint32_t *)( (uint32_t) mcan + (uint32_t) elemAddr );
    elem->sfid2 = (regVal >> 0U)  & 0x000007FF;
    elem->sfid1 = (regVal >> 16U) & 0x000007FF;
    elem->sfec  = (regVal >> 27U) & 0x00000007;
    elem->sft   = (regVal >> 30U) & 0x00000003;
};
DL_MCAN_readStdMsgIDFilter(MCAN0_INST, filtNum, &gMCAN0StdFltRd);
```

「*DL_MCAN_readExtMsgIDFilter*」API を呼び出すことにより、CPU はインデックス「*filtNum*」を含むメッセージ RAM フィルタ要素を読み取り、フィルタ値を「*gMCAN0ExtFltRd*」にコピーします。

```
uint32_t filtNum = 2;
DL_MCAN_ExtMsgIDFilterElement gMCAN0ExtFltRd[CANEXTFLTNUM] = {0};
void DL_MCAN_readExtMsgIDFilter(MCAN_Regs *mcan, uint32_t filtNum,
    DL_MCAN_ExtMsgIDFilterElement *elem)
{
    uint32_t startAddr, elemAddr, regVal;

    //startAddr = HW_RD_FIELD32(&mcan->MCANSS.MCAN.MCAN_XIDFC, MCAN_XIDFC_FLESA);
    startAddr = *(volatile uint32_t *)(&mcan->MCANSS.MCAN.MCAN_XIDFC);
    startAddr = (startAddr & 0x0000FFFC) >> 2;
    startAddr = (uint32_t)(startAddr << 2U);
    elemAddr  = startAddr + (filtNum * 2U * 4U);
    elemAddr += 0x0U;

    //regVal = HW_RD_REG32(((uint32_t) mcan + (uint32_t) elemAddr));
    regVal = *(volatile uint32_t *)( (uint32_t) mcan + (uint32_t) elemAddr );
```


Copyright © 2026 Texas Instruments Incorporated

```
    elem->efid1 = (regVal >> 0U)  & 0x1FFFFFFF;
    elem->efec  = (regVal >> 29U) & 0x00000007;

    elemAddr += 4U;
    //regVal = HW_RD_REG32(((uint32_t) mcan + (uint32_t) elemAddr));
    regVal = *(volatile uint32_t *)( (uint32_t) mcan + (uint32_t) elemAddr );
    elem->efid2 = (regVal >> 0U)  & 0x1FFFFFFF;
    elem->eft   = (regVal >> 30U) & 0x00000003;
};
DL_MCAN_readExtMsgIDFilter(MCAN0_INST, filtNum, &gMCAN0ExtFltRd);
```

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。

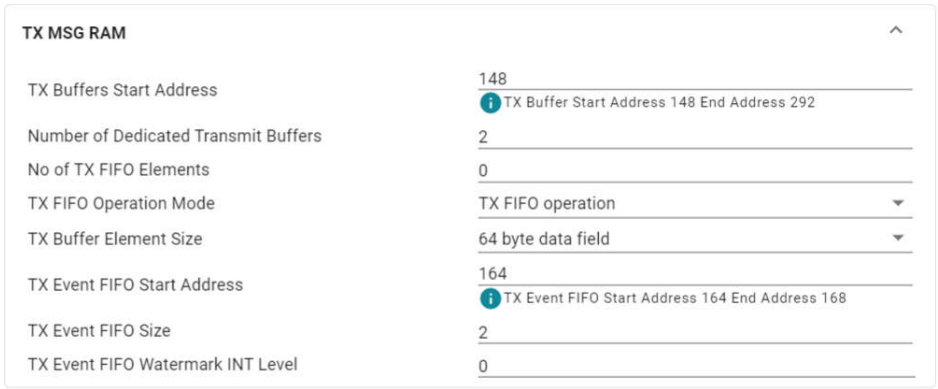


図 2-6. TX MSG RAM

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。

- 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。

表 2-4 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 2-4. 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。

Copyright © 2026 Texas Instruments Incorporated

表 2-4. Tx モード間の違い (続き)

Tx モード	説明
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。

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。

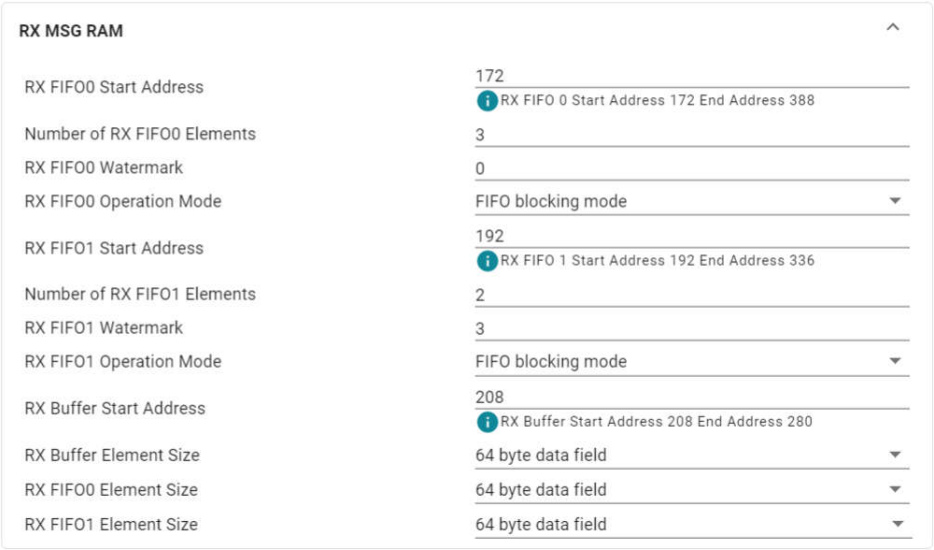


図 2-7. RX MSG RAM

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, RX FIFO0, and RX FIFO1 Element Size (RX バッファ、RX FIFO0 および RX FIFO1 要素サイズ):Rx バッファおよび FIFO 要素サイズを定義します。

Copyright © 2026 Texas Instruments Incorporated

バッファ、FIFO0、FIFO1 はメッセージ RAM 内の独立した開始アドレスを使用し、互いに重複することはできません。各セクションの長さを自由に制御して、アプリケーションの要件を満たすことができます。

「RX バッファ 要素数」に相当するセクションはありません。sfec または efec が 0x7 に設定されている場合、フィルタ要素の sfid2 または efid2 でバッファの書き込みインデックスを制御できます。または、本アプリケーションで使用している「MCAN モジュールの SysConfig 構成」のサンプル コードを参照してください。最大インデックスは、RX バッファに残されている空き領域に基づいて、ユーザーが管理する必要があります。この例では、RX バッファ用に残されている領域は 1 つ分のみです。

Copyright © 2026 Texas Instruments Incorporated

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。

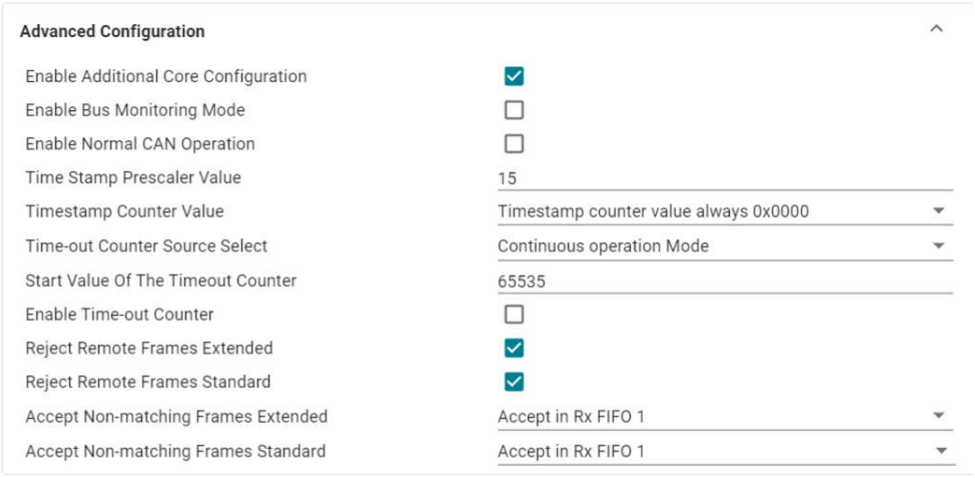


図 2-8. 高度な構成

- 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。

Copyright © 2026 Texas Instruments Incorporated

- 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。

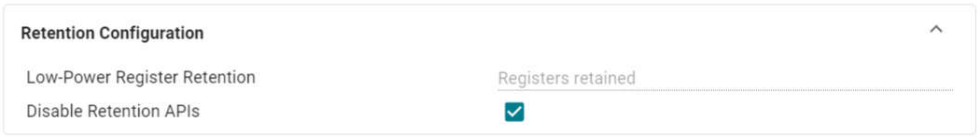


図 2-9. 保持構成

- 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

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 NVIC への MCAN 割り込み要求 (IRQ) を管理します。図 2-10 に、このデバイスでの MCAN モジュールの統合を示します。

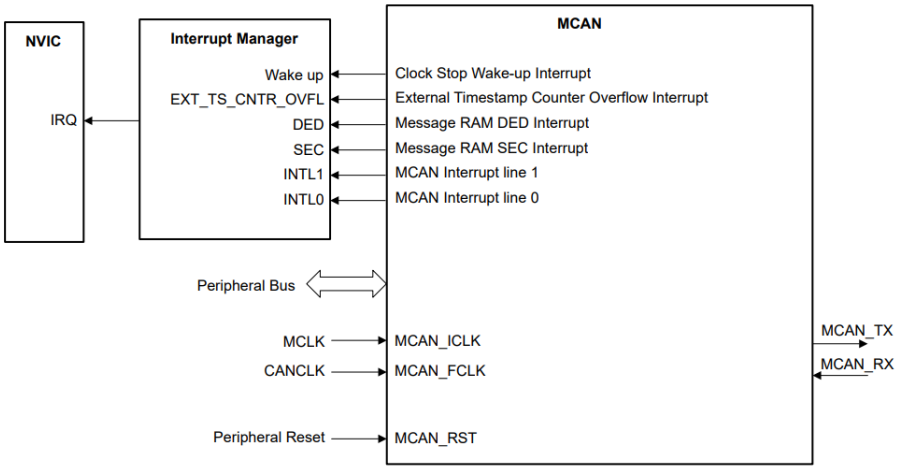


図 2-10. MCAN の統合

図 2-11 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。

Copyright © 2026 Texas Instruments Incorporated

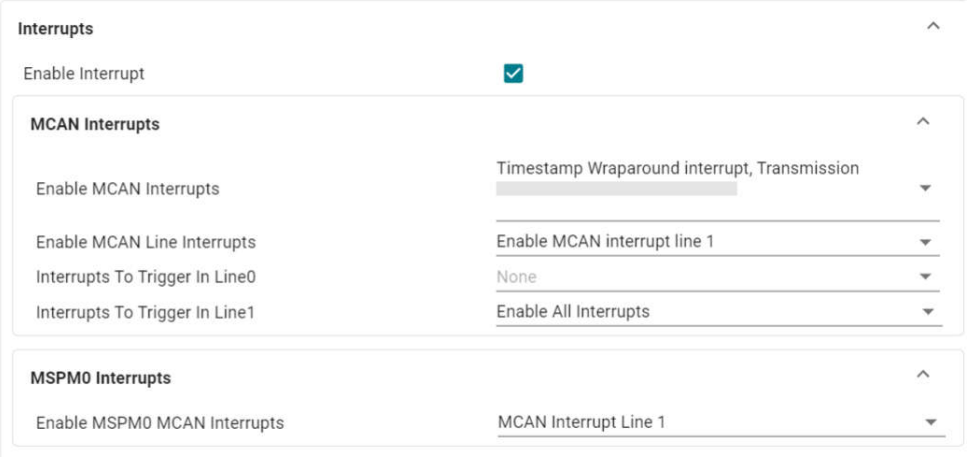


図 2-11. 割り込み

- MCAN 割り込み - 図 2-10 の MCAN ブロック
  - 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 Line1 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 割り込み - 図 2-10 の割り込みマネージャ
  - 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。

MCAN 割り込みは Line0 または Line1 に自由に割り当てることができます。た図 2-11 の例では、すべての割り込みは Line1 に割り当てられ、Line0 に割り込みは割り当てられません。

Copyright © 2026 Texas Instruments Incorporated

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。

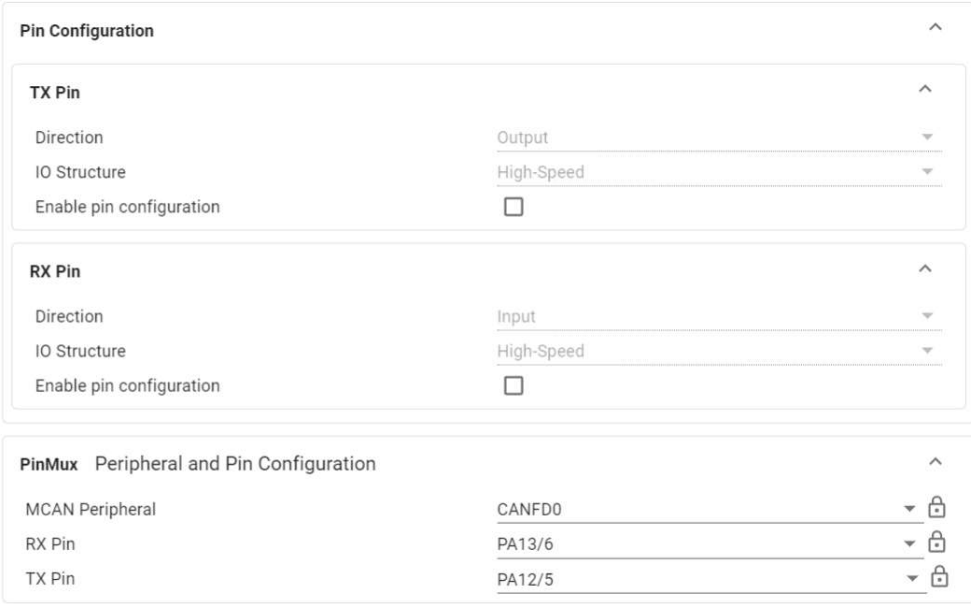


図 2-12. ピン構成および PinMux

- 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。

Copyright © 2026 Texas Instruments Incorporated

3 SDK サンプル コードの紹介

「SDK サンプル コードの紹介」では、driverlib フォルダの 6 つの MSPM0 MCAN サンプル コードを紹介します。ソースには、{mspm0_root_folder}\examples\nortos\LP_MSPM0G3507\driverlib\mcan_* フォルダからアクセスできます。MSPM0 SDK のドキュメントについては、オンライン ドキュメントを参照してください。

3.1 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。

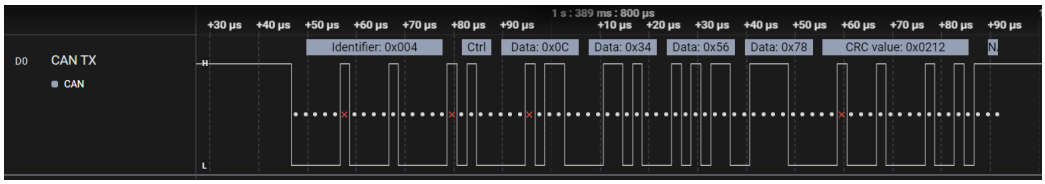


図 3-1. MCAN ループバック メッセージ

3.2 mcan_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。

3.3 mcan_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。

3.4 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

Copyright © 2026 Texas Instruments Incorporated

3.5 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

3.6 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

Copyright © 2026 Texas Instruments Incorporated

4 MCAN ユースケース

SysConfig の構成ガイドおよび SDK サンプルの紹介に加えて、「MCAN ユースケース」では、MSPM0 MCAN の便利な TX/RX 処理メカニズムや、低消費電力モードおよび CAN バス オフ状態における MCAN モジュールの制御方法について、さらに詳しく説明します。これにより、ユーザーが MSPM0 MCAN モジュールへの理解を深め、開発を加速できるようにします。ユーザーは、参考用として「MCAN モジュールの SysConfig 構成」に掲載されているサンプル コードをダウンロードできます。

4.1 TX バッファ素子

MCAN TX バッファ素子は、送信が必要なメッセージ ID、データ、および CAN 制御フィールド情報を格納するための、MSPM0 の RAM 上の変数です。TX バッファ素子は、API DL_MCAN_writeMsgRam を介して MCAN メッセージ RAM に書き込むことができます。

以下のコードは、TX バッファ素子変数の定義と初期化を行い、driverlib API を使用して MCAN メッセージ RAM に書き込む方法を示しています。

```
DL_MCAN_TxBufElement txMsg;
txMsg.id = ((uint32_t)(0x4)) << 18U;    // ID = 0x4
txMsg.rtr = 0U;     // Transmit data frame
txMsg.xtd = 0U;     // 11-bit standard identifier
txMsg.esi = 0U;     // ESI bit in CAN FD format depends only on error passive flag
txMsg.dlc = 1U;     // Transmitting 1 bytes
txMsg.brs = 1U;     // CAN FD frames transmitted with bit rate switching
txMsg.fdf = 1U;     // Frame transmitted in CAN FD format
txMsg.efc = 1U;     // Store Tx events
txMsg.mm = 0xAAU;   // Message marker = 0xAA
txMsg.data[0] = 0x01;   // Data bytes
/* Write Tx Message to the Message RAM. */
DL_MCAN_writeMsgRam(MCAN0_INST, DL_MCAN_MEM_TYPE, Index, &txMsg);
```

表 4-1 に、Tx バッファ素子の各メンバーの説明を示します。

表 4-1. MCAN Tx バッファ素子の構造

メンバー	名称	説明
id	識別子:	識別子、11 ビットの標準識別子は、値を割り当てる前に 18 ビット左シフトする必要があります。
rtr	リモート送信要求	0 = 送信データ フレーム 1 = 送信リモート フレーム
xtd	拡張識別子	0 = 11 ビットの標準識別子 1 = 29 ビットの拡張イベント識別子
esi	エラー状態インジケータ	0 = CAN FD 形式の ESI ビットは、エラー パッシブ フラグのみに依存します 1 = CAN FD 形式の ESI ビットが送信されたリセッシブ
dlc	データ長コード	0 ～ 8 = CAN + CAN FD: 送信フレームには 0 ～ 8 バイトのデータが含まれています 9 ～ 15 = CAN: 送信フレームには 8 バイトのデータが含まれています 9 ～ 15 = CAN FD: 送信フレームには 12/16/20/24/32/48/64 バイトのデータが含まれています
brs	ビット ラットのスイッチング	0 = ビット レートのスイッチングなしで送信された CAN FD フレーム 1 = ビット レートのスイッチングにより送信された CAN FD フレーム

Copyright © 2026 Texas Instruments Incorporated

表 4-1. MCAN Tx バッファ素子の構造 (続き)

メンバー	名称	説明
fdf	FD フォーマット	0 = Classic CAN 形式で送信されるフレーム 1 = CAN FD 形式で送信されるフレーム
efc	イベント FIFO 制御	0 = Tx イベントを保存しません 1 = Tx イベントを保存します
mm	メッセージ マーカー	TX イベント FIFO に入力するメッセージ マーカー
データ	データ バイト	最初の dlc で指定されたバイト数のみが有効です。

4.2 RX バッファ素子

MCAN RX バッファ素子は、CAN バスから受信したメッセージ ID、データ、および CAN 制御フィールド情報を格納するための MSPM0 の RAM 変数です。RX バッファ素子は、API DL_MCAN_readMsgRam を使用して、MCAN メッセージ RAM から読み取ることができます。

以下のコードは、RX バッファ素子を初期化し、driverlib API を使用して、MCAN メッセージ RAM から RX バッファ素子に CAN メッセージを読み取る方法を示しています。

```
DL_MCAN_RxBufElement rxMsg;
/* Read from Message RAM to Rx Buffer Element. */
DL_MCAN_readMsgRam(MCAN0_INST, DL_MCAN_MEM_TYPE, bufIndex, fifoIndex, &rxMsg);
rxMsgId = rxMsg.id;
rxMsgDlc = rxMsg.dlc;
for(int i = 0; i <= rxMsgDlc; i++) {
    rxMsgData[i] = rxMsg.data[i];
}
```

MCAN Rx バッファ素子の構造に、RX バッファ素子の各メンバーの説明を示します。

表 4-2. MCAN Rx バッファ素子の構造

メンバー	名称	説明
id	識別子:	識別子、11 ビットの標準識別子は、メッセージ RAM から読み取った後、18 ビット右シフトする必要があります。
rtr	リモート送信要求	0 = 受信フレームはデータ フレームです 1 = 受信フレームはリモート フレームです
xtd	拡張識別子	0 = 11 ビットの標準識別子 1 = 29 ビットの拡張イベント識別子
esi	エラー状態インジケータ	0 = 送信ノードはエラー アクティブです 1 = 送信ノードはエラー パッシブです
rxts	Rx タイムスタンプ	タイムスタンプ値
dlc	データ長コード	0 ～ 8 = CAN + CAN FD:受信フレームには 0 ～ 8 バイトのデータが含まれています 9 ～ 15 = CAN:受信フレームには 8 バイトのデータが含まれています 9 ～ 15 = CAN FD:受信フレームには 12/16/20/24/32/48/64 バイトのデータが含まれています
brs	ビット ラットのスイッチング	0 = ビット レート切り替えなしで受信されたフレーム 1 = ビット レート切り替えで受信されたフレーム

Copyright © 2026 Texas Instruments Incorporated

表 4-2. MCAN Rx バッファ素子の構造 (続き)

メンバー	名称	説明
fdf	FD フォーマット	0 = 標準フレーム フォーマット 1 = CAN FD フレーム フォーマット (新しい DLC コーディングと CRC)
fidx	フィルタ インデックス	フィルタのインデックス
anmf	受け入れられた不整合フレーム	0 = 受信フレーム マッチング フィルタ インデックス FIDX 1 = 受信したフレームが Rx フィルタ要素と一致しませんでした
データ	データ バイト	最初の dlc で指定されたバイト数のみが有効です。

Copyright © 2026 Texas Instruments Incorporated

4.3 TX 処理バッファ モード

TX バッファ モードで CAN メッセージを受信するには、SysConfig の「Number of Dedicated Transmit Buffers (専用送信バッファの数)」で、Tx バッファ用のメッセージ RAM 領域を確保する必要があります。

以下に、TX バッファの処理メカニズムの例を示します。送信要求保留レジスタをポーリングして使用可能なバッファを検索し、その後、CAN メッセージをバッファに書き込みます。

- TX バッファ素子を書き込むために、メッセージ RAM 内の空き TX バッファを検索するか、対象となる TX バッファが空いているかどうかを確認します。
- API DL_MCAN_writeMsgRam を使用して、DL_MCAN_MEM_TYPE_BUF モードでメッセージ RAM に書き込みます。
- CANTxBufferPutIndex に送信要求を追加します。
- 必要に応じて、送信が完了するのを待ちます。

```
uint8_t MCAN_TxBufPoll(MCAN_Regs *mcan, DL_MCAN_TxBufElement *txMsg, uint8_t length)
{
    uint8_t CANTxBufferPutIndex = 0;
    uint8_t loop_i = 0;

    for(loop_i = 0; loop_i < length; loop_i++) {
        /* Find the empty Tx buffer to start transfer. */
        do {
            CANTxBufferPutIndex = (__builtin_ctz(~DL_MCAN_getTxBufReqPend(mcan)));
        }
        while(CANTxBufferPutIndex >= MCAN0_INST_MCAN_TX_BUFF_SIZE);
        /* Write Tx Message to the Message RAM. */
        DL_MCAN_writeMsgRam(mcan, DL_MCAN_MEM_TYPE_BUF, CANTxBufferPutIndex, &txMsg[loop_i]);
        /* Add request for transmission. */
        DL_MCAN_TXBufAddReq(mcan, CANTxBufferPutIndex);
    }

    /* Wait until all pending transmission requirements are proceed */
    while(DL_MCAN_getTxBufReqPend(mcan) != 0x00);

    return loop_i;
}
```

TX バッファによる CAN メッセージの送信は、CPU によって完全に制御されます。CPU は、送信要求 API DL_MCAN_TXBufAddReq を使用することで、どのインデックスのバッファから最初に送信するかを制御できます。TX バッファに複数のメッセージを書き込み、同時に送信要求を追加した場合、id 番号が最も小さいメッセージが先に送信されます。TX バッファに同じ id の複数のメッセージが書き込まれた場合、バッファ番号が最も小さいメッセージが先に送信されます。

4.4 TX 処理 FIFO モード

TX FIFO モードで CAN メッセージを送信するには、SysConfig の「No of TX FIFO Elements (TX FIFO 要素数)」で Tx FIFO 用のメッセージ RAM 領域を確保する必要があります。

以下に、TX FIFO の処理メカニズムの例を示します。TX FIFO のステータスをポーリングして、TX FIFO が満杯でない状態になるまで待機した後、CAN メッセージを FIFO に書き込みます。

- TX FIFO が空になるまで確認して、待機します。
- API DL_MCAN_writeMsgRam を使用して、DL_MCAN_MEM_TYPE_FIFO モードでメッセージ RAM に書き込みます。
- txFS.putIdx に送信要求を追加します。
- 送信完了を待機することが推奨されます。

```
uint8_t MCAN_TxFIFOPoll(MCAN_Regs *mcan, DL_MCAN_TxBufElement *txMsg, uint8_t length)
{
    uint8_t loop_i = 0;

    while(loop_i < length) {
        do{
            /* Update the Tx FIFO Status. */
            DL_MCAN_getTxFIFOQueStatus(mcan, &txFS);
```


Copyright © 2026 Texas Instruments Incorporated

```
            /* Add timeout handling here */
        }
        while(txFS.fifoFull);
        /* Write Tx Message to the Message RAM. */
        DL_MCAN_writeMsgRam(mcan, DL_MCAN_MEM_TYPE_FIFO, txFS.putIdx, &txMsg[loop_i++]);
        /* Add request for transmission. */
        DL_MCAN_TXBufAddReq(mcan, txFS.putIdx);
    }

    /* Wait until all pending transmission requirements are proceed */
    while(DL_MCAN_getTxBufReqPend(mcan) != 0x00);

    return loop_i;
}
```

Tx FIFO メッセージのデータは、Get インデックスの MCAN_TXFQS.TFGI フィールドで参照されるメッセージから順に送信されます。各送信の後、Get インデックスは Tx FIFO が空になるまでインクリメントされます。

TX バッファモードと FIFO モードの両方において、MCAN 割り込みは送信完了を通知する機能をサポートしています。TX 割り込みを処理するには、以下の手順を使用します。ユーザーは、上記の TX バッファおよび FIFO のポーリング処理のサンプルを参考にして、割り込みハンドラ「*MCAN_TxFIFOInt_Service*」を作成できます。または、「MCAN モジュールの SysConfig 構成」に記載のデモ コードを参照してください。

- Syscfg の MCAN 設定で、MCAN および MSPM0 の割り込みが正しく構成されていることを確認します。
- MCAN_TXBTIE で、対応する TX バッファ割り込みを有効にします。
- MCAN の TC 送信完了割り込みを有効にします。
- バッファ モードまたは FIFO モードを使用して、送信を開始します。
- IRQHandler で TC 割り込みを処理します。
- TX イベント FIFO 割り込みを使用することが推奨されます。

```
    /* Enable all Tx Buffer's Interrupt */
    mcan->MCANSS.MCAN.MCAN_TXBTIE = 0xFFFFFFFF;
    /* Enable MCAN Transmission Completed interrupt. */
    DL_MCAN_enableIntr(mcan, DL_MCAN_INTERRUPT_TC, 1U);

void MCAN0_INST_IRQHandler(void)
{
    switch (DL_MCAN_getPendingInterrupt(MCAN0_INST)) {
        case DL_MCAN_IIDX_LINE1:
            /* Check MCAN interrupts fired during TX/RX of CAN package */
            gInterruptLine1Status = DL_MCAN_getIntrStatus(MCAN0_INST);

            /* Transmission Complete from one interrupt enabled buffer/FIFO */
            if(gInterruptLine1Status&DL_MCAN_INTERRUPT_TC) {
                MCAN_TxFIFOInt_Service(MCAN0_INST);
            }

        default:
            break;
    }
}
```


Copyright © 2026 Texas Instruments Incorporated

4.5 RX 処理バッファ モード

RX バッファ モードで CAN メッセージを受信するには、SysConfig の「RX Buffer Start Address (RX バッファの開始アドレス)」で **Dedicated Rx Buffer (専用 Rx バッファ)** 用のメッセージ RAM 領域を確保する必要があります。有効な開始アドレスを設定し、使用する RX バッファ数に対して十分な空き領域があることを確認してください。この領域は手動で計算する必要があります。計算方法については、「MCAN メッセージ RAM の構成例」およびテクニカル リファレンス マニュアルの「Rx バッファおよび FIFO 素子」を参照してください。開始アドレス以降の空き領域はすべて RX バッファとして使用され、他の用途には使用できません。

CAN RAM に CAN メッセージを正しく受信するには、ID フィルタを正しく構成する必要があります。フィルタの構成ガイドは「標準および拡張 ID フィルタの構成」を参照してください。ここでは、ID = 3 メッセージをインデックス = 0 RX バッファに格納し、API DL_MCAN_addStdMsgIDFilter を使用して構成を CAN RAM に保存する標準 ID フィルタの例を示します。

```
static const DL_MCAN_StdMsgIDFilterElement gMCAN0StdFiltelem = {
    .sfec = 0x7,  // Save message to DRX
    .sft  = 0x0,  // Ignored when sfec = 0x7
    .sfid1 = 3,   // Filter ID, message must match this ID
    .sfid2 = 0x0, // RX buffer index when sfec = 0x7
};
/* Configure Standard ID filter element */
DL_MCAN_addStdMsgIDFilter(MCAN0_INST, 0U, (DL_MCAN_StdMsgIDFilterElement *) &gMCAN0StdFiltelem);
```

以下に、RX バッファの処理メカニズムの例を示します。MCAN 新規データ レジスタ (MCAN_NDATx) をポーリングして、メッセージを受信したバッファを特定し、その後、バッファからメッセージを読み取ります。

1. 指定された bufIdx に基づいて、NDATIdx と canBufStatus を設定します。
2. MCAN_NDATx をポーリングして、CAN RX バッファに有効な CAN メッセージが受信されるまで待機します。
3. API DL_MCAN_readMsgRam を使用して DL_MCAN_MEM_TYPE_BUF モードでメッセージ RAM を読み取ります。
4. 対応するステータス ビットに 1 を書き込み、ステータスをクリアします。

```
uint8_t MCAN_RxBufPoll(MCAN_Regs *mcan, DL_MCAN_RxBufElement *rxMsg, uint8_t bufIdx)
{
    uint8_t NDATIdx = 0;
    volatile uint32_t *canBufStatus = NULL;

    if(bufIdx <= 31) {
        NDATIdx = bufIdx;
        canBufStatus = &mcan->MCANSS.MCAN.MCAN_NDAT1;
    }
    else {
        NDATIdx = bufIdx - 32;
        canBufStatus = &mcan->MCANSS.MCAN.MCAN_NDAT2;
    }

    /* Wait for receiving */
    while( !((*canBufStatus) & (0x00000001 << NDATIdx)) );

    if( (*canBufStatus) & (0x00000001 << NDATIdx) ) {
        /* Read message to rxMsg from CAN buffer */
        DL_MCAN_readMsgRam(mcan, DL_MCAN_MEM_TYPE_BUF, bufIdx, 0, &rxMsg[0]);
        /* Clear corresponding new data status */
        *canBufStatus = 0x00000001 << NDATIdx;

        return 1;
    }

    return 0;
}
```

RX バッファは割り込み処理をサポートしており、以下に RX バッファの割り込み処理の例を示します。RX バッファは、RX FIFO とは独立した専用の割り込みステータス DL_MCAN_INTERRUPT_DRX を使用し、API DL_MCAN_enableIntr

Copyright © 2026 Texas Instruments Incorporated

で有効にできます。上記の RX バッファ処理の例を参考にして、割り込みハンドラ「*MCAN_RxBufInt_Service*」を書き込むか、「MCAN モジュールの SysConfig 構成」に記載されているデモ コードを参照してください。

```
DL_MCAN_enableIntr(mcan, DL_MCAN_INTERRUPT_DRX, 1U);

void MCAN0_INST_IRQHandler(void)
{
    switch (DL_MCAN_getPendingInterrupt(MCAN0_INST)) {
        case DL_MCAN_IIDX_LINE1:
            /* Check MCAN interrupts fired during TX/RX of CAN package */
            gInterruptLine1Status = DL_MCAN_getIntrStatus(MCAN0_INST);

            /* Message Stored to Dedicated Rx Buffer interrupt */
            if(gInterruptLine1Status&DL_MCAN_INTERRUPT_DRX) {
                MCAN_RxBufInt_Service(MCAN0_INST);
            }

        default:
            break;
    }
}
```

注

一般的な使用事例では、特定の CAN ID 用に専用の RX バッファを構成することは、ハードウェアでフィルタ処理されたメッセージ ルーティングを実現するのに便利な方法です。メッセージが ID に関係なく順番にキューイングされる RX FIFO とは異なり、専用 RX バッファでは、特定の CAN ID のメッセージが常に同じメモリ位置に書き込まれます。これにより、優先度の高いメッセージや高頻度で送信されるメッセージが、他の通信によって上書きされたり、遅延したりすることを防ぎ、ゲートウェイでのルーティング、重要な診断情報の追跡、および低遅延のコマンド処理が可能になります。1 つまたは複数の ID フィルタを 1 つの RX バッファに割り当てることができます。また、各 RX バッファを異なるコールバック関数に振り分け、NDAT0 および NDAT1 (バッファ インデックス) に基づいて処理することができます。

Copyright © 2026 Texas Instruments Incorporated

4.6 RX 処理 FIFO モード

RX FIFO モードで CAN メッセージを受信するには、SysConfig の「Number of RX FIFOx Elements (RX FIFOx 要素数)」で RX FIFO 用のメッセージ RAM 領域を確保する必要があります。TX FIFO とは異なり、MSPM0 MCAN モジュールは 2 つの RX FIFO をサポートしており、ユーザーは 1 つまたは複数の ID フィルタを 1 つの RX FIFO に割り当てることができます。

CAN RAM に CAN メッセージを受信するには、ID フィルタを正しく構成する必要があります。フィルタの構成ガイドは「標準および拡張 ID フィルタの構成」を参照してください。ここでは、ID = 3 メッセージを RX FIFO0 に格納し、API DL_MCAN_addStdMsgIDFilter を使用して構成を CAN RAM に保存する標準 ID フィルタの例を示します。sfid2 = 0x7FF (マスク ビット = 1 に対応する各ビットについて ID が照合されるため) であるため、メッセージの ID は sfid1 と完全に一致する必要があります。

```
static const DL_MCAN_StdMsgIDFilterElement gMCAN0StdFiltelem = {
    .sfec = 0x1,  // Save to FIFO0
    .sft  = 0x2,  // sfid1 is id, sfid2 is mask
    .sfid1 = 3,   // Filter ID, message must match this ID
    .sfid2 = 0x7FF, // RX buffer index when sfec = 0x7
};
/* Configure Standard ID filter element */
DL_MCAN_addStdMsgIDFilter(MCAN0_INST, 0, &gMCAN0StdFiltelem);
```

以下に、RX FIFO 処理メカニズムの例を示します。ポーリングを行い、RX FIFO が空ではないのを待機した後、RX FIFO から読み取り、RX FIFO ACK を CAN に書き込みます。

1. RX FIFO の割り込みステータスおよび RX FIFO の空き状態をポーリングし、RX FIFO0 にメッセージが格納されるまで待機します。
2. API DL_MCAN_readMsgRam を使用して DL_MCAN_MEM_TYPE_FIFO モードでメッセージ RAM を読み取ります。
3. RX FIFO に ACK を書き込みます。

```
uint8_t MCAN_RxFIFOPoll(MCAN_Regs *mcan, DL_MCAN_RxBufElement *rxMsg, uint8_t length)
{
    uint8_t loop_i = 0;

    for(loop_i = 0; loop_i < length; loop_i++) {
        rxFS.num = DL_MCAN_RX_FIFO_NUM_0;
        rxFS.fillLvl = 0;

        /* Waiting for New CAN Message Stored into FIFO 0 */
        while( !(DL_MCAN_getIntrStatus(mcan)&(DL_MCAN_INTERRUPT_RF0N|DL_MCAN_INTERRUPT_RF0F)) );
        /* Clear Receive FIFO 0 New Message Interrupt Flag */
        DL_MCAN_clearIntrStatus(mcan, DL_MCAN_INTERRUPT_RF0N|DL_MCAN_INTERRUPT_RF0F,
DL_MCAN_INTR_SRC_MCAN_LINE_1);

        /* Waiting for FIFO 0 Fill Level Not 0 */
        while ((rxFS.fillLvl) == 0) {
            DL_MCAN_getRxFIFOStatus(mcan, &rxFS);
        }
        DL_MCAN_readMsgRam(mcan, DL_MCAN_MEM_TYPE_FIFO, 0U, rxFS.num, &rxMsg[loop_i]);
        DL_MCAN_writeRxFIFOAck(mcan, rxFS.num, rxFS.getIdx);
    }

    return loop_i;
}
```

RX FIFO は割り込み処理をサポートしており、以下に RX FIFO の割り込み処理の例を示します。どちらの RX FIFO にも、それぞれ独自の割り込みステータスがあり、その一覧を以下に示します。

- DL_MCAN_INTERRUPT_RF0N、DL_MCAN_INTERRUPT_RF1N：RX FIFO に新しいメッセージが格納されました。
- DL_MCAN_INTERRUPT_RF0F、DL_MCAN_INTERRUPT_RF1F：RX FIFO に新しいメッセージが格納され、満杯になりました。
- DL_MCAN_INTERRUPT_RF0L、DL_MCAN_INTERRUPT_RF1L：RX FIFO メッセージ消失が発生しました。

Copyright © 2026 Texas Instruments Incorporated

- **注**

  RX FIFO がメッセージを正常に受信するすべての状況に対応するには、RX FIFO の新規メッセージ割り込みと満杯割り込みを併用する必要があります。これは、メッセージの書き込みによって FIFO が満杯になった場合、その後は RX FIFO の新規メッセージ割り込みが発生しないためです。RX FIFO のメッセージ消失割り込みは、エラー処理用の割り込みとして使用されます。これは、RX FIFO が満杯になった後も読み取り処理が行われず、新しいメッセージがブロックされて破棄される、または古いメッセージが上書きされることを通知します。

```
/* Enable MCAN receive FIFO 0 new message interrupt. */
DL_MCAN_enableIntr(mcan, DL_MCAN_INTERRUPT_RF0N|DL_MCAN_INTERRUPT_RF0F, 1U);

void MCAN0_INST_IRQHandler(void)
{
    switch (DL_MCAN_getPendingInterrupt(MCAN0_INST)) {
        case DL_MCAN_IIDX_LINE1:
            /* Check MCAN interrupts fired during TX/RX of CAN package */
            gInterruptLine1Status = DL_MCAN_getIntrStatus(MCAN0_INST);

            /* New message received by Rx FIFO 0 */
            if( (gInterruptLine1Status & DL_MCAN_INTERRUPT_RF0N) ||
                (gInterruptLine1Status & DL_MCAN_INTERRUPT_RF0F)
              ) {
                MCAN_RxFIFOInt_Service(MCAN0_INST);
            }

        default:
            break;
    }
}
```

上記の RX FIFO ポーリング処理の例を参考にして、割り込みハンドラ「*MCAN_RxFIFOInt_Service*」を書き込むか、「MCAN モジュールの SysConfig 構成」に記載されているデモ コードを参照してください。

Copyright © 2026 Texas Instruments Incorporated

4.7 低消費電力モードでの MSPM0 MCAN の使用

ユーザーは、アプリケーションの消費電力の要件を満たすために、MSP デバイス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 GPIO 入力ピンとして構成できます。MCAN RX ピンでの立ち下がりエッジ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、マイコンは立ち下がりエッジ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 モジュールと MCAN フィルタを再構成します。MCAN はこのように通常機能を復元し、Rx ピンによるウェークアップをサポートします。

注

最初の MCAN メッセージは、MSP デバイス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、キー メッセージをウェークアップ メッセージとして使用しないでください。

コード例を以下に示します。ユーザーは、API「*MCAN_LowPowerMode*」を呼び出すことで低消費電力モードに移行する必要があります。

```
void MCAN0_Restart(void)
{
    /* Restart the MCAN */
    DL_MCAN_reset(CANFD0);
    delay_cycles(16);
    DL_MCAN_disablePower(CANFD0);
    delay_cycles(32);
    DL_MCAN_enablePower(CANFD0);
    /* Wait until MCAN module is ready */
    while(!(DL_SYSCTL_getStatus()&DL_SYSCTL_STATUS_MCAN0_READY));
    /* Init MCAN */
    SYSCFG_DL_MCAN0_init();
}

/**
 * Demo for enter and exit STANBY0 mode steps while using MCAN
 * Switch CAN Rx to GPIO input capture in STANDBY mode
 */
void MCAN_LowPowerMode(void)
{
    /* Switch CAN Rx pin to GPIO input mode */
    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);
    /* Enable GPIO interrupt */
    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);

    /* Disable SYSPLL */
    DL_SYSCTL_switchMCLKfromHSCLKtoSYSOSC();
    /* Set default configuration */
    DL_SYSCTL_disableSYSPLL();
    /* Switch low power mode to STANDBY0 */
    DL_SYSCTL_setPowerPolicySTANDBY0();
    /* Enter low power mode here */
    __WFI();
    /* Switch low power mode to SLEEP0 */
    DL_SYSCTL_setPowerPolicyRUN0SLEEP0();
    /* Init the SYSCTL (Clock and power policy) */
    SYSCFG_DL_SYSCTL_init();

    /* Switch GPIO input mode to CAN Rx pin */
    DL_GPIO_initPeripheralInputFunction(GPIO_MCAN0_IOMUX_CAN_RX, GPIO_MCAN0_IOMUX_CAN_RX_FUNC);

    MCAN0_Restart();
}
```

注

低消費電力モードから復帰した後は、CAN フィルタおよび CAN 割り込みの両方を、手動で通常動作モードに戻るよう再設定する必要があります。

Copyright © 2026 Texas Instruments Incorporated

4.8 バス オフ対応

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。この方法では、CAN コントローラをすばやく復帰させることができます。
- 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。この方法は CAN の要件を満たしています。

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_LINE1:
            /* Check MCAN interrupts fired during TX/RX of CAN package */
            gInterruptLine1Status = DL_MCAN_getIntrStatus(MCAN0_INST);

            /* MCAN bus off status changed */
            if(gInterruptLine1Status&DL_MCAN_INTERRUPT_BO) {
                DL_MCAN_getProtocolStatus(MCAN0_INST, &gProtStatus);
                if(gProtStatus.busOffStatus == 1) { // Busoff entered
                    CANBusOff = 1;
                }
                else {  // Busoff exited
                    CANBusOff = 0;
                }
            }

        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。「パッシブ リカバリ」または「強制リカバリ」のいずれかの方法を選択します。

```
main loop:
{
        /* CAN bus off recovery */
        if(CANBusOff == 1) {
            /* Passive recovery: detected 128 consecutive inst/ of 11 recessive bits */
            MCAN_BusoffRecovery();  // CANBusOff reset to 0 in interrupt
                                    // This API can be directly put into CAN interrupt
```


Copyright © 2026 Texas Instruments Incorporated

```
            /* Force recovery: reset CAN to restore from bus off */
            // MCAN0_Restart();
            // CANBusOff = 0;
        }
}

void MCAN0_Restart(void)
{
    /* Restart the MCAN */
    DL_MCAN_reset(CANFD0);
    delay_cycles(16);
    DL_MCAN_disablePower(CANFD0);
    delay_cycles(32);
    DL_MCAN_enablePower(CANFD0);
    /* Wait until MCAN module is ready */
    while(!(DL_SYSCTL_getStatus()&DL_SYSCTL_STATUS_MCAN0_READY));
    /* Init MCAN */
    SYSCFG_DL_MCAN0_init();
}
```


Copyright © 2026 Texas Instruments Incorporated

5 設計テストとデバッグのヒント

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。

5.1 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)。

5.2 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。

5.3 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。

5.3.1 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。

Copyright © 2026 Texas Instruments Incorporated

5.3.2 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)。

5.3.3 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。

6 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。また、量産経験に基づき、ユーザー定義のサンプル コードを使用した一般的な使用事例をいくつか紹介します。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。

7 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

Copyright © 2026 Texas Instruments Incorporated

- 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

Copyright © 2026 Texas Instruments Incorporated

8 改訂履歴

Changes from Revision * (April 2025) to Revision A (October 2026)	Page
• CAN 基本規格および MSP MCAN モジュールの概要セクションを追加	3
• ドキュメント全体にわたって表、図、相互参照の採番方法を更新	3
• CAN ビットのタイミングの概要を追加	4
• MCAN の機能を MSPM0 MCAN モジュールに変更	5
• すべての図を最適化	6
• Syscfg の CAN ビット タイミング表を追加	8
• 表 2-3 を追加	11
• MSP SDK サンプル コードを紹介するための独立したセクションを追加。	22
• 既存のバッファ / FIFO セクションを最適化	24
• CAN 低消費電力モード機能に、SYSPLL 低消費電力モード処理コードを追加。	33
• CAN バス オフからの復帰方法として、パッシブ リカバリをデフォルトで使用。	34

Copyright © 2026 Texas Instruments Incorporated

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。

Copyright © 2026, Texas Instruments Incorporated

最終更新日：2025 年 10 月