TRAVEO™ T2G ファミリの CAN FD 使用方法

本書について

適用範囲と目的

このアプリケーションノートでは、TRAVEO™ T2G ファミリマイクロコントローラの Controller Area Network with Flexible Data (CAN FD) rate の使用方法について説明します。

対象者

このドキュメントは TRAVEO™ T2G MCU の CAN FD を使用するすべての人を対象にします。

関連製品ファミリー

TRAVEO™ T2G ファミリ CYT2/CYT3/CYT4/CYT6 シリーズ

目次

1 はじめに

このアプリケーションノートでは、TRAVEO™ T2G ファミリデバイスの CAN FD を使用する方法について説明します。

CAN FD は CAN (現在は「クラシック CAN」と呼ばれます) の拡張です。CAN FD は、クラシック CAN の 1 Mbps 制限を超えるビットレートで最大 64 バイトのデータフレームを送信できます。データセグメント内の達成可能な最大バス速度は、トランシーバなどの外部コンポーネントとアプリケーションの特定のネットワークトポロジによってのみ制限されます。5 Mbps をサポートするトランシーバがあります。いくつかの新製品は、8 Mbps の速度を要求しています。

TRAVEO™ T2G の CAN FD コントローラ (M_TTCAN) は、クラシカル CAN だけでなく、CAN FD (ISO 11898-1:2015) および CAN (ISO 11898-4：2004) 上の Time-Triggered (TT) 通信をサポートします。CAN FD コントローラは、ISO 16845:2015 に従って認証されています。

このドキュメントは、CYT2/CYT3/CYT4/CYT6 シリーズデバイスに適用されます。

2 CAN FD 概要

このセクションでは、CAN FD 通信の動作と CAN FD 通信の例と CAN FD メッセージフォーマットとビットタイミングの考慮事項について説明します。

2.1 CAN FD ネットワーク

図 1 に、CAN FD ネットワークの例を示します。

CAN FD ネットワークでは、2 つの通信回線 (CANH, CANL) を使用してノイズに対して回復力を持たせています。複数の電子制御ユニット (ECU) を CAN FD ネットワークに接続できます。データは ECU 間で交換されます。

レシーバノードは、CAN FD トランシーバにより差動バス電圧をデジタル信号に変換します。受信データは、マイクロコントローラの CAN FD 制御ロジックによって処理されます。送信時には、CAN FD 制御ロジックから CAN FD トランシーバにデータが送信され、差動信号を CAN FD ネットワークの CANH および CANL ラインに送信します。

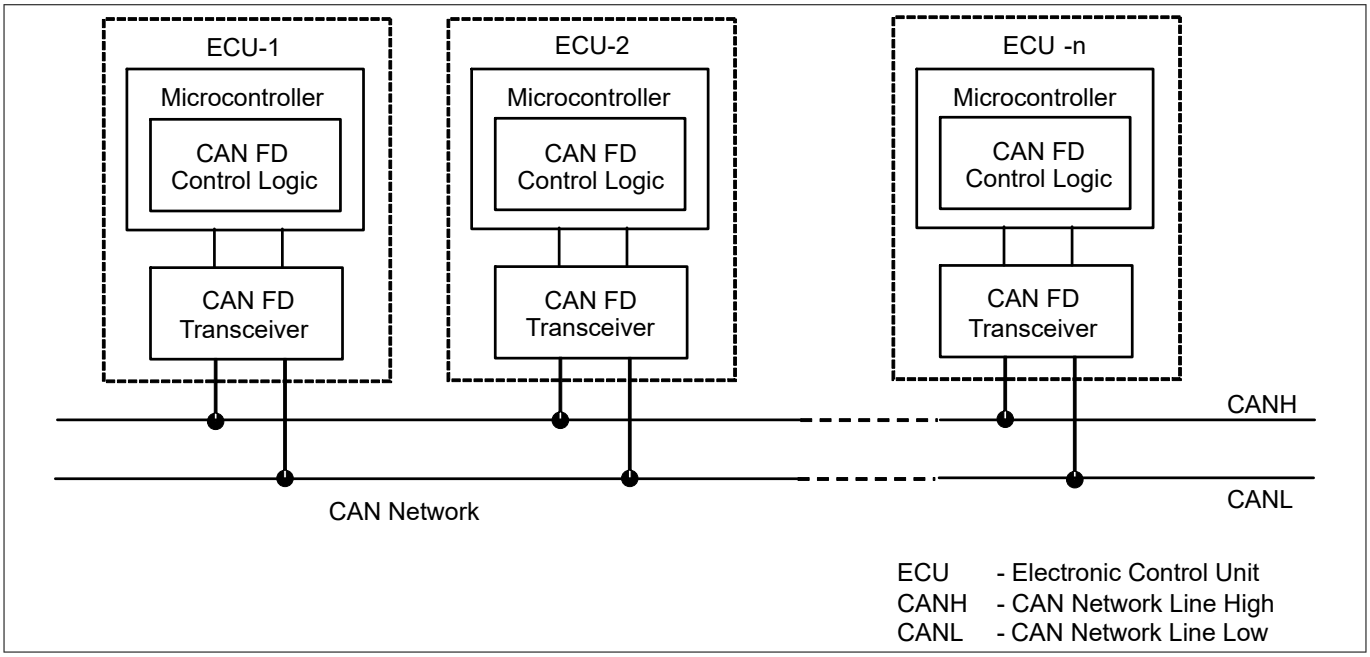

図 1 CAN FD ネットワーク

2.2 CAN FD メッセージ

4 つのフレーム (データフレーム, リモートフレーム, エラーフレーム, オーバーロードフレーム) のうち、ここではデータフレームについて説明します。

図 2 は、クラシック CAN および CAN FD メッセージフレームのデータフレームフォーマットを示します。先述のように、CAN FD はクラシック CAN の拡張であり、両方のメッセージフォーマットはアービトレーションセグメントと CRC フィールドの後が同じです。違いはデータフィールドです。CAN FD フレームはより多くのデータバイトを持ち、アービトレーションボーレートよりも高速で送信できます。

クラシック CAN の最大データ長は 8 バイトで、最大ボーレートは 1 Mbps です。

CAN FD はアービトレーションフェーズで最大 64 バイトのデータ長をサポートし、最大ボーレートは 1 Mbps です。データ通信速度は、クラシック CAN で設定された 1 Mbps を超える場合があり、トランシーバやネットワークトポロジなどの外部コンポーネントによって制限されます。

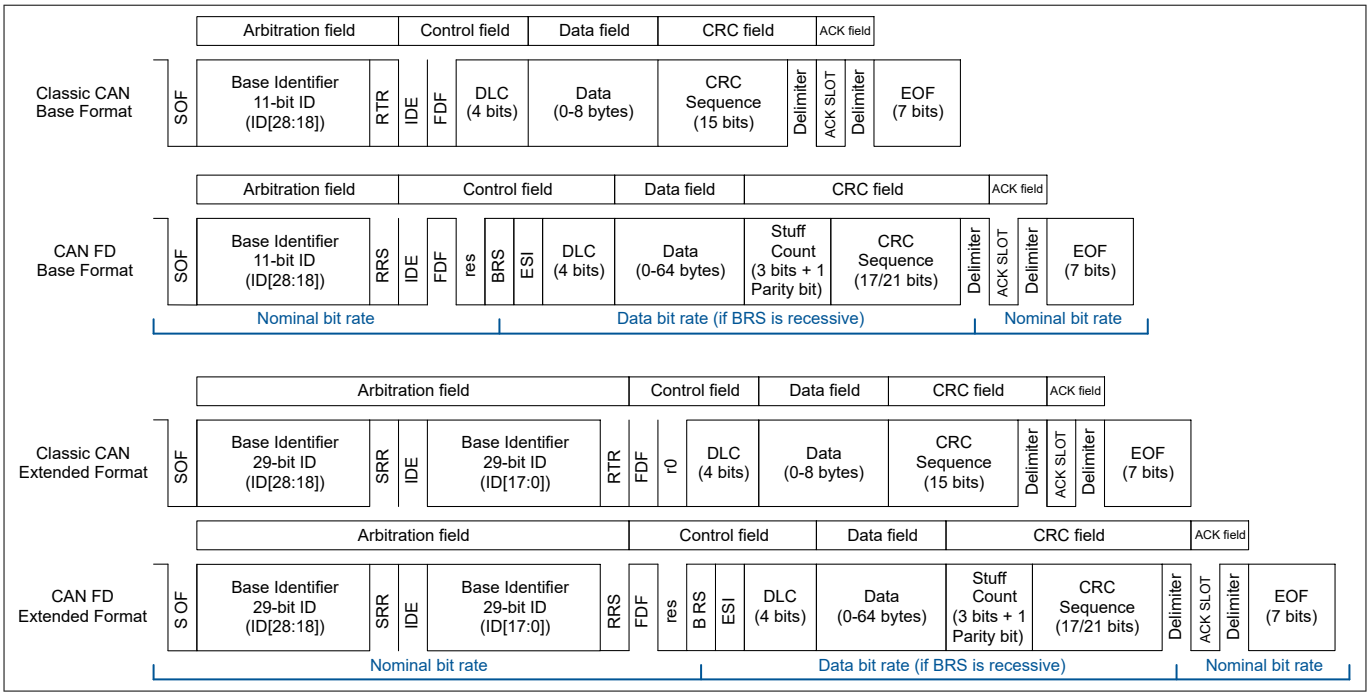

図 2　　データフレームフォーマット

2.2.1　CAN FD フィールド

CAN FD フレームフォーマットのフィールドは、アービトレーションフィールド, コントロールフィールド, データフィールド, CRC フィールドおよび ACK フィールドを含みます。

アービトレーションフィールドはメッセージ ID 番号を含み、同時に送信を開始しようとする他ノードからのメッセージ間の優先順位を決定します。メッセージ ID は、IDE ビットによって構成された 11 ビット (基本フォーマット) または 29 ビット (拡張フォーマット) にできます。

コントロールフィールドの FD フォーマット (FDF) インディケータビットは、フレームタイプを CAN または CAN FD として識別します。FDF ビットは、CAN FD フレームではレセッシブ ('1') であり、CAN フレームではドミナント ('0') です。ビットレートスイッチ (BRS) ビットがレセッシブである場合、データフィールドのビットレートは異なるビットレート、通常は高速に切り替えられます。BRS ビットがドミナントの場合、データフィールドのビットレートはアービトレーションビットレートのままです。エラーステートインジケータ (ESI) ビットは、CAN FD ノードのエラー状態の識別に使用します。BRS ビットと ESI ビットは、CAN FD フレームでのみ使用可能です。

さらに、データ長コード (DLC) は 4 ビットを持ち、送信されるデータのバイト数を示します。この設定可能な範囲は、CAN フレームの場合は 0～8 バイト、CAN FD フレームの場合は最大 64 バイトです。表 1 に、DLC フィールドと送信データバイト数の関係を示します。

表 1　　CAN および CAN FD における DLC コーディング

DLC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CAN 送信データバイト数	0	1	2	3	4	5	6	7	8	8	8	8	8	8	8	8
CAN FD 送信データバイト数	0	1	2	3	4	5	6	7	8	12	16	20	24	32	48	64

データフィールドはメッセージデータを送信し、DLC によって設定されたデータ長でサイズが決まります。

CRC フィールドは、CRC シーケンスと CRC デリミタで構成されます。CAN フレームの場合、CRC シーケンスは 15 ビットの固定長です。CAN FD フレームは、CRC フィールドの先頭に 4 ビットのスタッフカウント、CRC シーケンス

(データ長が 0～16 バイトの場合は 17 ビット、16 バイトを超えるデータ長の場合は 21 ビット) が続きます。いずれの受信ノードも受信したメッセージのデータストリームを分析し、それを送信された CRC と比較して、有効または誤って受信したメッセージを識別できます。

ACK フィールドは、ACK スロットと ACK デリミタで構成されます。送信ノードは、レセッシブビットとして ACK を送信し、1 つ以上の受信ノードは、メッセージ受信が成功した場合にドミナントビットでこれを上書きします。これは、トランスミッタがフレームが正常に受信されたか破損したかの判断に役立ちます。

フレームは、エンドオブフレーム (EOF) を形成する 7 つのレセッシブビットのフラグシーケンスで終了します。

2.2.2 ビットタイミング

クラシック CAN 操作は、メッセージフレーム全体に対して単一のビットタイムを定義します。CAN FD 動作は、ノミナルビットタイムとデータビットタイムの 2 つのビットタイムが定義されます。ノミナルビットタイムはアービトレーションフェーズのためです。データビットタイムはノミナルビットタイム以下であり、データフェーズを加速するために使用できます。

ビットタイムの基本構成は、ノミナルビットタイムとデータビットタイムの両方で共有されます。ビットタイムは、同期セグメント (Sync_Seg), 伝播時間セグメント (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) の 4 つのセグメントに分割できます (図 3 を参照)。バスレベルをビットの値として読み出すサンプリングポイントは、Phase_Seg1 の最後です。

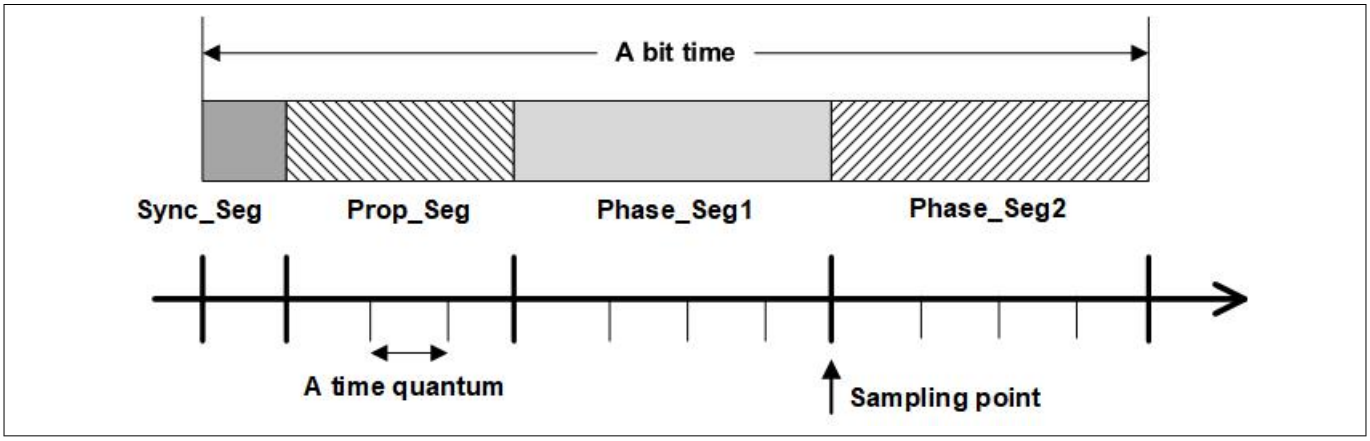

図 3 ビットタイムの構成

各セグメントは、CAN クロックとプリスケーラによって定義されるタイムクォンタムの倍数であるプログラム可能なタイムクォンタで構成されています。これらのパラメータを定義するために使用される値とプリスケーラは、ノミナルビットタイムとデータビットタイムで異なり、ノミナルビットタイミング&プリスケーラレジスタ (NBTP) とデータビットタイミング&プリスケーラレジスタ (DBTP) によって設定されます (表 2 を参照)。

表 2 CAN ビットタイミングパラメータ

記号	説明
Time quantum tq (nominal) and tqd (data)	タイムクォンタム。基本単位時間のタイムクォンタ (CAN クロック周期) に書くプリスケーラを乗算し導出します。 タイムクォンタムは、CAN FD コントローラによってノミナル値として設定されます。tq = (NBTP.NBRP[8:0] +1) × CAN clock period data:tqd = (DBTP.DBRP[4:0] + 1) × CAN clock period

(続く)

表 2 (続き) CAN ビットタイミングパラメータ

記号	説明
Sync_Seg	Sync_Seg は CAN 仕様によって 1 タイムクォンタムに固定されているため設定できません (CAN FD コントローラに内蔵)。 nominal: 1 tq data: 1 tqd
Prop_Seg	Prop_Seg は、ネットワーク内の物理的な遅延時間を保証するために使用されるビットタイムの一部です。CAN FD コントローラは Prop_Seg と Phase_Seg1 の合計を単一のパラメータで構成します。 nominal: Prop_Seg + Phase_Seg1 = NBTP.NTSEG1[7:0] + 1 data: Prop_Seg + Phase_Seg1 = DBTP.DTSEG1[4:0] + 1
Phase_Seg1	Phase_Seg1 はサンプリングポイント前のエッジ位相誤差を保証するために使用されます。再同期ジャンプ幅(Resynchronization jump width)によって短縮できます。. Prop_Seg と Phase_Seg1 の合計は、CAN FD コントローラによってノミナル値として設定されます。nominal: NBTP.NTSEG1[7:0] + 1 data: DBTP.DTSEG1[4:0] + 1
Phase_Seg2	Phase_Seg2 はサンプリングポイント後のエッジ位相誤差を保証するために使用されます。再同期ジャンプ幅(Resynchronization jump width)によって短縮できます。. Phase_Seg2 は CAN FD コントローラによってノミナル値として設定されます。nominal: NBTP.NTSEG2[6:0] + 1 data: DBTP.DTSEG2[3:0] + 1
SJW	再同期ジャンプ幅。(Resynchronization Jump Width) ノード間のタイミング変動を自動的に補正し、Phase_Seg1 と Phase_Seg2 の長さの調整に使用します。SJW は Phase_Seg1 または Phase_Seg2 より長くなりません。 SJW は CAN FD コントローラによってノミナル値として設定されます。nominal: NBTP.NSJW[6:0] + 1 data: DBTP.DSJW[3:0] + 1

ノミナルおよびデータビットタイムは以下の式によって表せます。

ノミナルビットタイム

=Sync_Seg + Prop_Seg + Phase_Seg1 + Phase_Seg2×tq

=1+NBTP.NTSEG17:0+1+NBTP.NTSEG26:0+1×NBTP.NBRP8:0+1×CAN clock period

例 (500 kbps, サンプリングポイント 75%)

=1+13+1+4+1×3+1×140000000=0.000002 500 kbps

データビットタイム

=1+DBTP.DTSEG14:0+1+DBTP.DTSEG23:0+1×DBTP.DBRP4:0+1×CAN clock period

例 (5Mbps, サンプリングポイント 62.5%)

=1+3+1+2+1×0+1×140000000=0.0000002 5 Mbps

例 (2 Mbps, サンプリングポイント 60%)

=1+10+1+7+1×0+1×140000000=0.0000005 2 Mbps

3 TRAVEO™ T2G ファミリの CAN FD コントローラ

このセクションでは、TRAVEO™ T2G ファミリの CAN FD コントローラの概要について説明します。

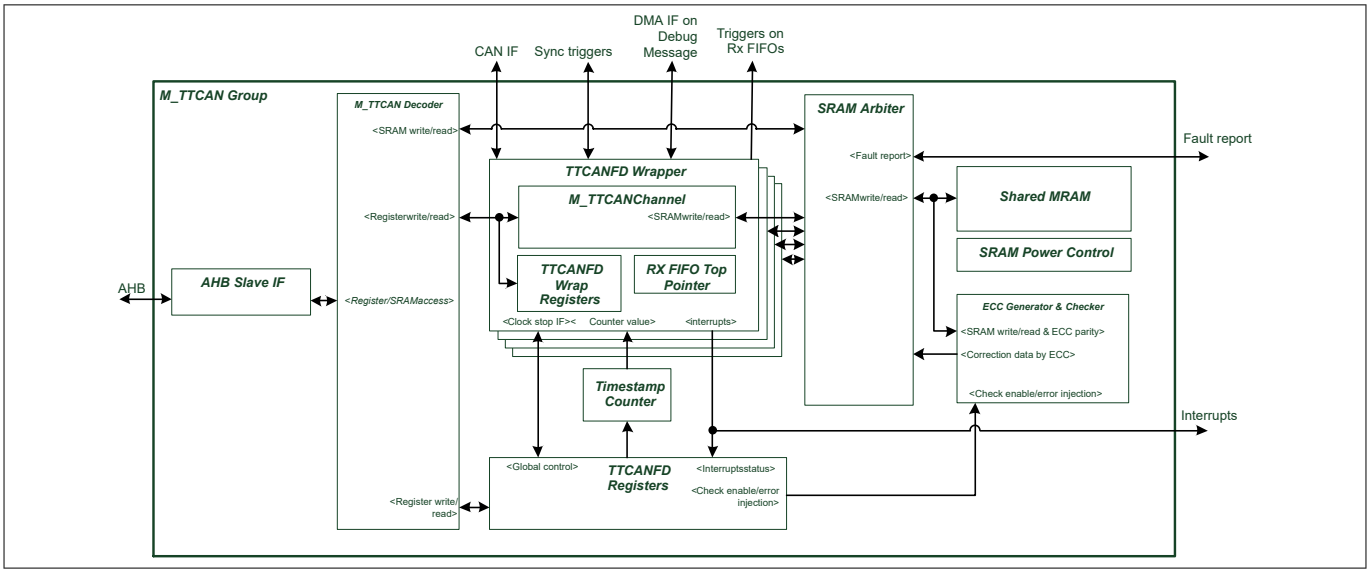

図 4 CAN FD コントローラブロックダイヤグラム

図 4 に TRAVEO™ T2G デバイスの CAN FD コントローラ (M_TTCAN) のブロックダイヤグラムを示します。TRAVEO™ T2G デバイスの M_TTCAN はグループごとに編成され、各グループは Message RAM を共有する 1 つ以上のチャネルで構成されます。使用可能な M_TTCAN グループとチャネルの総数は、デバイスの種類によって異なります。詳細は、デバイスのデータシートを参照してください。

M_TTCAN チャネルは、ISO 11898-1:2015 に準拠したクラシック CAN と CAN FD をサポートします。M_TTCAN は、アクティブおよびスリープパワーモードで利用できます。ディープスリープパワーモードではタイムスタンプカウンタ以外の IP は完全に保持されます。

CAN コアと Tx ハンドラおよび Rx ハンドラは、プロトコル処理を行います。メモリマップド I/O (MMIO) レジスタへのスレーブインタフェースは、CPU による CAN FD コントローラの設定を容易にします。各 M_TTCAN チャネルには、2 つのクロック入力 cclk と hclk があります。cclk は CAN FD 動作、hclk は内部 IP 動作 (レジスタアクセスやメッセージ RAM アクセスなど) に使用されます。

各 M_TTCAN グループは 1 つのメッセージ RAM を持ち、このメッセージ RAM はそのグループに属するチャネル間で共有されます。メッセージ RAM をそのグループのチャネルに割当て、重複した割当てを防止する必要があります。CAN FD コントローラは、メッセージ RAM の領域が複数のチャネル間で重複しているか内部的にチェックしません。メッセージ RAM は、シングルビットエラー訂正機能とダブルビットエラー検出機能で ECC 保護されています。メッセージ RAM への ECC エラーおよび範囲外アクセスは、Fault structure に報告されます。

各 M_TTCAN は 2 つの割込みライン (割込み 0 と割込み 1) を持ちます。割込みソースを割込み 0 または割込み 1 にルーティングできます。割込みソースには、新規メッセージ受信割込み, 送信完了割込みおよび受信 FIFO ウォーターマーク割込みがあります。

割込みラインに加え、統合割込み 0 および統合割込み 1 は各 M_TTCAN グループで使用できます。統合割込み 0 は、グループの全チャネルの割込み 0 ラインの論理和です。同様に、統合割込み 1 は、グループの全チャネルの割込み 1 ラインの論理和です。すべてのチャネル割込みラインと統合割込みラインは、デバイス割込みシステムにルーティングされます。

フレーム受信されるたび Rx ポインタを計算のためのソフトウェアオーバヘッドを除去するため、ハードウェアロジックが実装されています。Rx FIFO のトップポインタは、次の読出しアドレスを計算し、データを読み出すことができる各 FIFO の単一アドレス (RXFTOPn_DATA) を提供します。このロジックは、TTCAN レジスタセット内の特定のアクノリッジインデックス (RXFnA.FnA) も更新し、インデックスもそれに応じて増分されます。

以下のセクションでは、CAN FD コントローラが CAN FD メッセージを送受信する設定方法について説明します。

4 CAN FD 設定

ここでは、インフィニオンが提供するサンプルドライバライブラリ (SDL) を使用して、ユースケースに基づいて CAN FD を構成する方法について説明します。このアプリケーションノートのコードは SDL の一部です。SDL については参考資料 を参照してください。

SDL には基本的に、構成部とドライバ部があります。構成部は、主に目的の操作のためのパラメータ値を構成します。ドライバ部は、構成部のパラメータ値に基づいて各レジスタを構成します。

ご使用のシステムに合わせて構成部を構成できます。

4.1 CAN FD セットアップ

CAN FD をセットアップするには、次の手順を実行します。

1. CAN FD にクロック分周器を設定して割り当て、CAN FD 周辺クロックを初期化します。
2. CAN FD 通信に使用する I/O ポートを有効にします。
3. CAN FD の割込み要因を使用可能な外部 CPU 割込みにマッピングします。
4. CAN FD コントローラを初期化します。

手順 1～3 の設定については、Architecture Technical Reference Manual (TRM) の「Clocking System」、「Input/Output Subsystem」、および「Interrupts」章を参照してください。

4.2 CAN FD 初期化

図 5 に、CAN FD コントローラを初期化するフローを示します。このフローでは、(0) が構成部で実行され、(1) から (9) がドライバ部で実行されます。

(0) システムに応じてパラメータ値を設定してください。

(1) 初期化レジスタ (CCCR.INIT) を "1"に設定して CAN FD 通信を停止してください。次に、コンフィグレーション変更イネーブルレジスタを (CCCR.CCE) を有効にし、書込み保護された CAN FD コンフィグレーションレジスタへの書込みアクセスを許可してください。

(2) 標準 ID フィルタ構成 (SIDFC) レジスタと拡張 ID フィルタ構成レジスタ (XIDFC) を使用して、メッセージ RAM のメッセージフィルタのエレメント数と開始アドレスオフセットを構成してください。拡張 ID メッセージの受け入れフィルタリングに使用されない ID ビットをマスキングするための拡張 ID およびマスク (XIDAM) レジスタを構成してください。

(3) Rx および Tx メッセージは、Rx FIFO 0 コンフィグレーションレジスタ (RXF0C) および Rx FIFO 1 コンフィグレーションレジスタ (RXF1C) を使用して、Rx FIFO のエレメント数とメッセージ RAM の開始アドレスオフセットを設定してください。Rx FIFO トップポインタは、RXFTOP_CTL レジスタ設定により有効または無効にします。

Rx バッファコンフィグレーションレジスタ (RXBC) の Rx バッファ開始アドレスオフセットと Rx バッファ/FIFO エレメント数コンフィグレーション (RXESC) レジスタの Rx バッファまたは FIFO エレメントのデータフィールドサイズを設定してください。

アプリケーションが Tx イベント FIFO を使用する場合、TXEFC レジスタで設定する必要があります。このレジスタには、イベント FIFO サイズ, 開始アドレスオフセットおよびウォーターマークレベルを設定する必要があります。

Tx バッファコンフィグレーションレジスタ (TXBC) を使用して、メッセージ RAM に Tx バッファの数と開始オフセットを設定します。Tx バッファのデータフィールドサイズを Tx バッファエレメントサイズ設定レジスタ (TXESC) で設定してください。

(4) CAN FD チャネルに割り当てるメッセージ RAM 領域をクリアしてください。このメッセージ RAM 領域には、このチャネルの Rx バッファ, Tx バッファおよびフィルタ構成が保持されます。

(5) 動作モード設定:CC 制御レジスタ (CCCR) によりクラシック CAN または CAN FD モード (CCCR.FDOE)、およびビットレートスイッチ (CCCR.BRSE) を設定してください。

(6) ビットタイミング設定:CAN FD モードかつビットレートスイッチが有効の場合、アービトレーションフェーズで使用される NBTP (Nominal Bit Timing & Prescaler Register) とデータフェーズで使用されるデータビットタイミング&

4 CAN FD 設定

プリスケーラレジスタ (DBTP) を設定してください。CAN FD データフェーズでより高いビットレートを使用するためには、トランスミッタ遅延補償レジスタ (TDCR) を設定してください。

(7) メッセージフィルタは、グローバルフィルタ設定レジスタ (GFC) で設定されたフィルタと一致しないメッセージ ID を持つ受信フレームの処理を決定してください。

メッセージ RAM の開始アドレスに開始アドレスオフセット (SIDFC/XIDFC) を加算したアドレスにメッセージフィルタを設定してください。レンジフィルタ, デュアルフィルタまたはクラシックビットマスクフィルタを設定できます。詳細については、アーキテクチャ TRM のメッセージ RAM の章を参照してください。

(8) 送信時に Tx バッファが割込みをアサートするには、Tx バッファ送信割込みイネーブルレジスタ (TXBTIE) を設定してください。同様に、Tx バッファが送信キャンセルの完了時に割込みをアサートするには、Tx バッファキャンセル完了割込みイネーブルレジスタ (TXBCIE) を設定してください。割込みレジスタ (IR) の割込みフラグをクリアし、割込みイネーブル (IE) レジスタの各割込みを有効にしてください。CAN FD コントローラは 2 本の割込みラインを持ちます。割込みライン選択 (ILS) は、割込みが割り当てられているラインを決定します。割込みラインイネーブル (ILE) で割込みラインを有効にしてください。

(9) 初期化レジスタ (CCCR.INIT) を'0'に設定して CAN FD の動作を開始してください。CAN FD チャネルは、CCCR.INIT の読出しが'0'の値になると、メッセージを送受信する準備が完了します。

***注**: 一部の外部トランシーバは、CAN FD 通信前に設定 (たとえば、SPI インタフェース経由で) が必要な場合があります。詳細については、使用しているトランシーバのデータシートを参照してください。*

4 CAN FD 設定

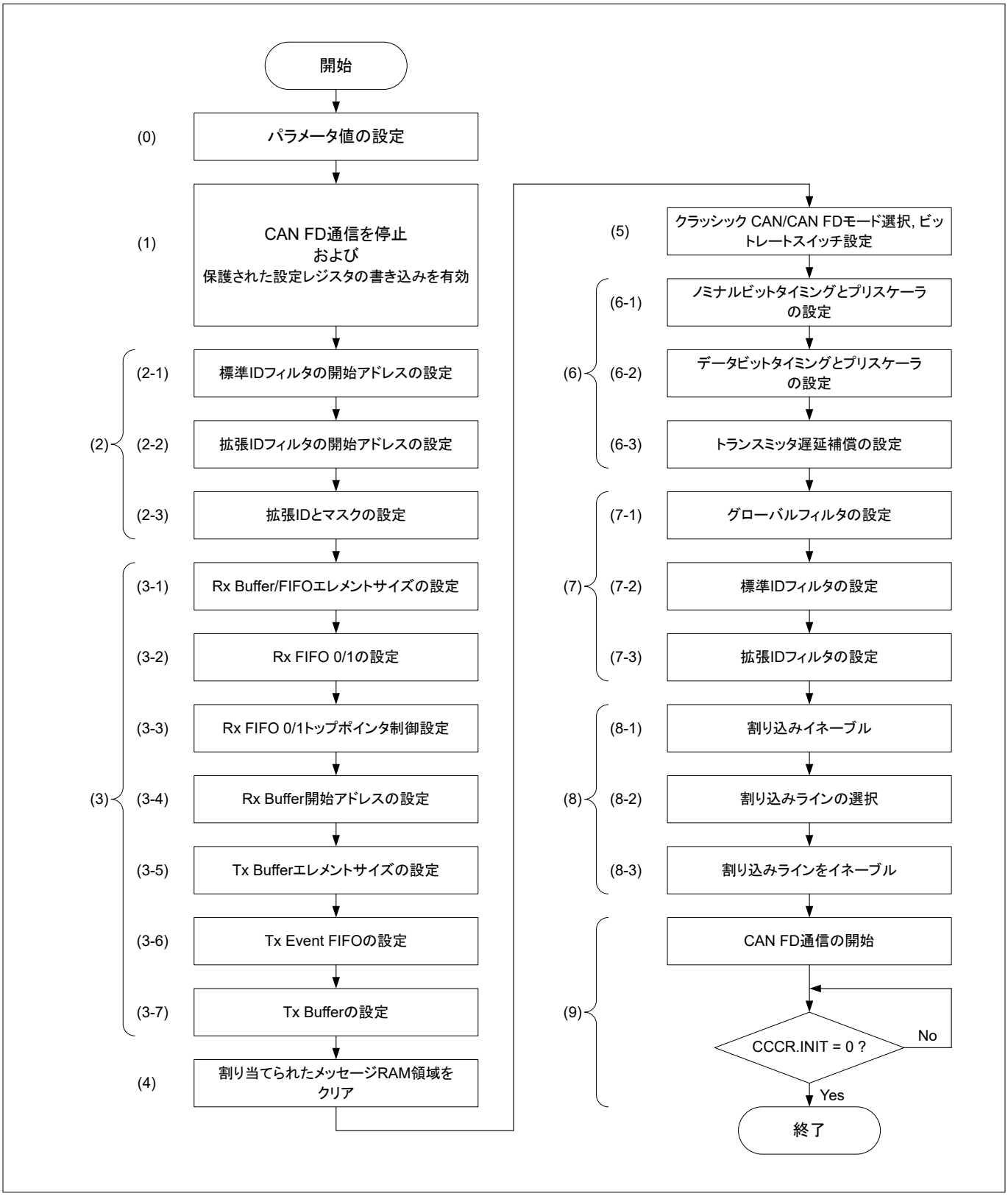

図 5　　CAN FD 初期化フロー例

4.2.1　　ユースケース

ここでは、次のユースケースを使用した CAN FD 初期化の例について説明します。CAN FD の初期化は、SDL を使用して構成されます。

ユースケース:

- モード: CAN FD
- CAN インスタンス: CAN0_CH0
- 割込みハンドラ: CAN メッセージ受信用
- 入力クロック: 40 MHz
- 標準ビットレート (サンプルポイント = 75%)
  - 500 kbps, 1 bit = 8 tq
  - プリスケーラ= 40 MHz / 500 kbps / 8 tq = 10
  - tseg1 = 5 tq, tseg2 = 2 tq, sjw = 2 tq
- 高速ビットレート(サンプルポイント = 75%)
  - 1 Mbps, 1 bit = 8 tq
  - プリスケーラ= 40 MHz / 1 Mbps / 8 tq = 5
  - tseg1 = 5 tq, tseg2 = 2 tq, sjw = 2 tq
- フィルタ構成: 2 つの標準および拡張 ID
- トランシーバ遅延補償:未使用
- Rx/Tx データサイズ: 64 バイト
- Tx イベント FIFO/バッファ数: 4

4.2.2 CAN FD コントローラの構成

CAN FD 初期化用の SDL におけるパラメータ構成を表 3 に示します。

表 3 CAN FD 初期化パラメーター覧

パラメータ		説明	値
Can_Cfg.txCallback		各イベントの割込みハンドラアドレスを設定します。NULL に設定されている場合、処理は行われません。	NULL
Can_Cfg.rxCallback			CAN_RxMsgCallback
Can_Cfg.rxFifoWithTopCallback			NULL
Can_Cfg.statusCallback			NULL
Can_Cfg.errorCallback			NULL
Can_Cfg.canFDMode		設定モードの選択 True: CAN FD モード, False: クラシック CAN モード	true
Can_Cfg.bitrate		標準ビットレート設定。	–
	.prescaler	ビットタイムクァンタを生成するために発振周波数を分周する設定値。設定値は、実際の値から 1 を引いた値です。	10u - 1u
	.timeSegment1	標準時間セグメント 1 の設定。設定値は、実際の値から 1 を引いた値です。	5u - 1u
	.timeSegment2	標準時間セグメント 2 の設定。設定値は、実際の値から 1 を引いた値です。	2u - 1u
	.syncJumpWidth	標準(再)同期ジャンプ幅の設定。設定値は、実際の値から 1 を引いた値です。	2u - 1u

(続く)

表 3 (続き) CAN FD 初期化パラメーター覧

パラメータ		説明	値
Can_Cfg.fastBitrate		高速ビットレート設定。CAN FD モードを設定する場合に必要です。	–
	.prescaler	ビットタイムクァンタを生成するために発振周波数を分周する設定値。設定値は、実際の値から 1 を引いた値です。	5u - 1u
	.timeSegment1	時間セグメント 1 の設定。設定値は、実際の値から 1 を引いた値です。	5u - 1u
	.timeSegment2	時間セグメント 2 の設定。設定値は、実際の値から 1 を引いた値です。	2u - 1u
	.syncJumpWidth	(再)同期ジャンプ幅の設定。設定値は、実際の値から 1 を引いた値です。	2u - 1u
Can_Cfg.tdcConfig		トランスミッタ遅延補正設定。CAN FD モードを設定する場合に必要です。	–
	.tdcEnabled	トランスミッタ遅延補正を有効にします。 True: 有効, False: 無効	false
	.tdcOffset	トランスミッタ遅延補正オフセットを設定します。	0
	.tdcFilterWindow	トランスミッタ遅延補正フィルタウィンドウ長を設定します。	0
Can_Cfg.sidFilterConfig		標準メッセージ ID フィルタを設定します。	–
	.numberOfSIDFilters	標準メッセージ ID フィルタ数。	sizeof(stdIdFilter) / sizeof(stdIdFilter[0])
	.sidFilter	標準メッセージ ID フィルタアドレスを設定します。	stdIdFilter
Can_Cfg.extidFilterConfig		拡張メッセージ ID フィルタを設定します。	–
	.numberOfEXTIDFilters	拡張メッセージ ID フィルタ数。	sizeof(extIdFilter) / sizeof(extIdFilter[0])
	.extidFilter	拡張メッセージ ID フィルタアドレスを設定します。	extIdFilter
	.extIDANDMask	拡張フレームのアクセプタンスフィルタリングのために、受信フレームのメッセージ ID と AND 演算される値を設定します。	0x1fffffff
Can_Cfg.globalFilterConfig		グローバルフィルタを設定します。	–
	.nonMatchingFramesStandard	フィルタリストのどの要素とも一致しない受信メッセージの処理方法を定義します。 Rx FIFO 0 で受け付け、Rx FIFO 1 で受け付け、拒否	CY_CANFD_ACCEPT_IN_RXFIFO_0
	.nonMatchingFramesExtended		CY_CANFD_ACCEPT_IN_RXFIFO_1
	.rejectRemoteFramesStandard	リモートフレームを拒否します。 True: リジェクトします, False: フィルタします	true
	.rejectRemoteFramesExtended		true

(続く)

表 3 (続き) CAN FD 初期化パラメーター覧

パラメータ		説明	値
Can_Cfg.rxBufferDataSize		Rx イベント FIFO サイズを設定します。	CY_CANFD_BUFFER_DATA_SIZE_64
Can_Cfg.rxFifo1DataSize		Rx データバッファサイズを設定します。	CY_CANFD_BUFFER_DATA_SIZE_64
Can_Cfg.rxFifo0DataSize		Rx FIFO1 サイズを設定します。	CY_CANFD_BUFFER_DATA_SIZE_64
Can_Cfg.txBufferDataSize		Tx バッファデータサイズを設定します。	CY_CANFD_BUFFER_DATA_SIZE_64
Can_Cfg.rxFifo0Config		Rx FIFO 0 を構成します。	–
	.mode	FIFO 0 オペレーションモード Blocking mode, Overwrite mode	CY_CANFD_FIFO_MODE_BLOCKING
	.watermark	Rx FIFO 0 ウォーターマーク割込みのレベルを設定します。	10u
	.numberOfFifoElements	Rx FIFO 0 のエレメント数を設定します。	8u
	.topPointerLogicEnabled	FIFO トップポインタロジックが FIFO 先頭アドレスとメッセージワードカウンタを設定できるようにします。True: 有効, False: 無効	false
Can_Cfg.rxFifo1Config		Rx FIFO 1 を構成します。	–
	.mode	FIFO 1 オペレーションモード ブロッキングモード, オーバライトモード	CY_CANFD_FIFO_MODE_BLOCKING
	.watermark	Rx FIFO 1 ウォーターマーク割込みのレベルを設定します。	10u
	.numberOfFifoElements	Rx FIFO 1 のエレメント数を設定します。	8u
	.topPointerLogicEnabled	FIFO トップポインタロジックが FIFO 先頭アドレスとメッセージワードカウンタを設定できるようにします。True: 有効, False: 無効	false
Can_Cfg.noOfRxBuffers		Tx イベント FIFO 数を設定します。	4u
Can_Cfg.noOfTxBuffers		専用 Tx バッファ数を設定します。	4u

Code Listing 1 は構成部にて CAN FD を初期化するサンプルプログラムを示します。

4 CAN FD 設定

Code Listing 1 構成部での CAN FD 初期化の例

```
/* Standard ID Filter configuration */
static const cy_stc_id_filter_t stdIdFilter[] =
{
    /*  Standard ID filter.  */
    CANFD_CONFIG_STD_ID_FILTER_CLASSIC_RXBUFF(0x010u, 0u),    /* ID=0x010, store into RX buffer
Idx0 */
    CANFD_CONFIG_STD_ID_FILTER_CLASSIC_RXBUFF(0x020u, 1u),    /* ID=0x020, store into RX buffer
Idx1 */
};

/* Extended ID Filter configration */
static const cy_stc_extid_filter_t extIdFilter[] =
{
    /*  Extended ID filter.  */
    CANFD_CONFIG_EXT_ID_FILTER_CLASSIC_RXBUFF(0x10010u, 2u),  /* ID=0x10010, store into RX
buffer Idx2 */
    CANFD_CONFIG_EXT_ID_FILTER_CLASSIC_RXBUFF(0x10020u, 3u),  /* ID=0x10020, store into RX
buffer Idx3 */
};

/* CAN configuration */
/*  Configure interrupt handler for each event. Registers CAN message reception event. Others
are NULL  */
cy_stc_canfd_config_t canCfg =
{
    .txCallback     = NULL,                                   // Unused.
    .rxCallback     = CAN_RxMsgCallback,
    .rxFifoWithTopCallback = NULL,                            //CAN_RxFifoWithTopCallback,
    .statusCallback = NULL,                                   // Un-supported now
    .errorCallback  = NULL,                                   // Un-supported now

    .canFDMode      = true,                                   // Use CANFD mode
    // 40 MHz
    .bitrate        =                                         // Nominal bit rate settings
(sampling point = 75%)
    {
         /*  Normal bit rate setting. Prescaler = 10, tseg1 = 4, tseg2 = 1, sjw = 1. Set to
minus 1.   */
        .prescaler      = 10u - 1u,                           // cclk/10, When using 500kbps,
1bit = 8tq
        .timeSegment1   = 5u - 1u,                            // tseg1 = 5tq
        .timeSegment2   = 2u - 1u,                            // tseg2 = 2tq
        .syncJumpWidth  = 2u - 1u,                            // sjw   = 2tq
    },

    .fastBitrate    =                                         // Fast bit rate settings (sampling
point = 75%)
    {
         /*  Fast bit rate setting. Prescaler = 5, tseg1 = 4, tseg2 = 1, sjw = 1. Set to minus
1.   */
        .prescaler      = 5u - 1u,                            // cclk/5, When using 1Mbps, 1bit =
```

```
8tq
        .timeSegment1   = 5u - 1u,                          // tseg1 = 5tq,
        .timeSegment2   = 2u - 1u,                          // tseg2 = 2tq
        .syncJumpWidth  = 2u - 1u,                          // sjw   = 2tq
    },
    .tdcConfig      =                                       // Transceiver delay compensation,
unused.
    {
         /*  Configure Transmitter Delay Compensation. Set tdcEnabled to false (this is
unused).  */
        .tdcEnabled     = false,
        .tdcOffset      = 0,
        .tdcFilterWindow= 0,
    },
    .sidFilterConfig    =                                   // Standard message ID filters
setting.
    {
        .numberOfSIDFilters = sizeof(stdIdFilter) / sizeof(stdIdFilter[0]),
        .sidFilter          = stdIdFilter,
    },
    .extidFilterConfig  =                                   // Extended message ID filters
setting.
    {
        .numberOfEXTIDFilters   = sizeof(extIdFilter) / sizeof(extIdFilter[0]),
        .extidFilter            = extIdFilter,
        .extIDANDMask           = 0x1fffffff,               // No pre filtering.
    },
    .globalFilterConfig =                                   // Global filter setting.
    {
        .nonMatchingFramesStandard = CY_CANFD_ACCEPT_IN_RXFIFO_0,  // Reject none match IDs
        .nonMatchingFramesExtended = CY_CANFD_ACCEPT_IN_RXFIFO_1,  // Reject none match IDs
        .rejectRemoteFramesStandard = true,                        // No remote frame
        .rejectRemoteFramesExtended = true,                        // No remote frame
    },
     /*  Configure FIFO and data buffer size: 64 bytes.  */
    .rxBufferDataSize = CY_CANFD_BUFFER_DATA_SIZE_64,
    .rxFifo1DataSize  = CY_CANFD_BUFFER_DATA_SIZE_64,
    .rxFifo0DataSize  = CY_CANFD_BUFFER_DATA_SIZE_64,
    .txBufferDataSize = CY_CANFD_BUFFER_DATA_SIZE_64,
    .rxFifo0Config    = // RX FIFO0, unused.
    {
    /*  Configure Rx FIFO 0. Set topPointerLogicEnabled to false (RxFIFO 0 is unused).  */
        .mode = CY_CANFD_FIFO_MODE_BLOCKING,
        .watermark = 10u,
        .numberOfFifoElements = 8u,
        .topPointerLogicEnabled = false,
    },
    .rxFifo1Config    =                                      // RX FIFO1, unused.
    {
         /*  Configure Rx FIFO 1.  */
        .mode = CY_CANFD_FIFO_MODE_BLOCKING,
        .watermark = 10u,
        .numberOfFifoElements = 8u,
```

```
        .topPointerLogicEnabled = false,                    // true,
    },
    .noOfRxBuffers  = 4u,
    .noOfTxBuffers  = 4u,         /*  Configure Tx event FIFO and Buffer.
    Set to 4.
    */
};

int main(void)
{
  :
  /*  CAN Channel setting. CAN0 Channel 0.  */
Cy_CANFD_Init(CY_CANFD_TYPE, &canCfg);
  :
}
```

4.2.3 メッセージ RAM の構成

ここでは、CAN メッセージ RAM の構成とメッセージ RAM 全体のサイズが TRAVEO™デバイスごとに異なる可能性があることを示します。各 CAN マクロの下でチャネルごとに割り当てられるサイズを指定する必要があります。SDL の一部として、構成が 1 例として提供されており、それぞれのアプリケーションの要件に基づいて変更できます。

CYT2B7 には、CAN マクロごとに 24KB のメッセージ RAM があります。Code Listing 2 は、チャネルごとに 8KB を割り当てる構成例を示しています。このコードを変更して、10KB + 10KB + 4KB を割り当てることができます。

Code Listing 2 メッセージ RAM の構成例

```
/** Offset of CAN FD Message RAM (MRAM). Total 24k MRAM per CAN FD instance is shared by all
CAN FD channels
 *  within an instance and allocation for each channel is done by user. Below shown is example
allocation */

/** Defining MRAM size (in bytes) per channel for CAN0 */
#define CY_CANFD0_0_MSGRAM_SIZE         ((CANFD0_MRAM_SIZE*1024)/CANFD0_CAN_NR)
#define CY_CANFD0_1_MSGRAM_SIZE         ((CANFD0_MRAM_SIZE*1024)/CANFD0_CAN_NR)
#define CY_CANFD0_2_MSGRAM_SIZE         ((CANFD0_MRAM_SIZE*1024)/CANFD0_CAN_NR)
```

4.2.4 ドライバ部の CAN FD 初期化コード例

Code Listing 3 は、CAN FD を初期化するドライバ部のサンプルプログラムを示します。

以下の説明は、SDL のドライバ部のレジスタ表記を理解するのに役立ちます。

- pstcCanFDChMTTCAN->unCCCR.u32Register は、レジスタ TRM に記載されている CANFDx_CHy_CCCR レジスタです。他のレジスタについても同様です。「x」は CAN FD インスタンスを示し、「y」は CAN FD インスタンスのチャネル番号を示します。
- パフォーマンス改善策

4 CAN FD 設定

レジスタ設定のパフォーマンス向上のため、SDL は完全な 32 ビットデータをレジスタに書き込みます。各ビットフィールドは、事前にビット書込み可能なバッファに生成され、最終的な 32 ビットデータとしてレジスタに書き込まれます。

```
unRXESC.stcField.u3RBDS = pstcConfig->rxBufferDataSize;
unRXESC.stcField.u3F1DS = pstcConfig->rxFifo1DataSize;  /* 1. Generate 32-bit data on the
buffer. */
unRXESC.stcField.u3F0DS = pstcConfig->rxFifo0DataSize;
pstcCanFDChMTTCAN->unRXESC.u32Register = unRXESC.u32Register;  /* 2. Write to register as
complete 32-bit data. */
```

レジスタの共用体と構造体表現の詳細については、hdr/rev_x の cyip_canfd.h を参照してください。

4 CAN FD 設定

Code Listing 3 ドライバ部の CAN FD 初期化の例

```
cy_en_canfd_status_t Cy_CANFD_Init(cy_pstc_canfd_type_t pstcCanFD, const cy_stc_canfd_config_t*
pstcConfig)
{
// Local variable declarations
cy_stc_canfd_context_t* pstcCanFDContext;
uint32_t*               pu32Adrs;
uint32_t                u32Count;
uint32_t                u32SizeInWord;

cy_stc_id_filter_t*     pstcSIDFilter;
cy_stc_extid_filter_t*  pstcEXTIDFilter;
cy_stc_canfd_msgram_config_t stcMsgramConfig;
volatile stc_CANFD_CH_M_TTCAN_t* pstcCanFDChMTTCAN;

/* Shadow data to avoid RMW and speed up HW access */
/*  Set data on the buffer to '0' for performance improvement.  */
un_CANFD_CH_SIDFC_t unSIDFC = { 0 };
un_CANFD_CH_XIDFC_t unXIDFC = { 0 };
un_CANFD_CH_XIDAM_t unXIDAM = { 0 };
un_CANFD_CH_RXF0C_t unRXF0C = { 0 };
un_CANFD_CH_RXF1C_t unRXF1C = { 0 };
un_CANFD_CH_RXBC_t  unRXBC  = { 0 };
un_CANFD_CH_TXEFC_t unTXEFC = { 0 };
un_CANFD_CH_TXBC_t  unTXBC  = { 0 };
un_CANFD_CH_CCCR_t  unCCCR  = { 0 };
un_CANFD_CH_NBTP_t  unNBTP  = { 0 };
un_CANFD_CH_DBTP_t  unDBTP  = { 0 };
un_CANFD_CH_TDCR_t  unTDCR  = { 0 };
un_CANFD_CH_GFC_t   unGFC   = { 0 };
un_CANFD_CH_RXESC_t unRXESC = { 0 };
un_CANFD_CH_TXESC_t unTXESC = { 0 };
un_CANFD_CH_IE_t    unIE    = { 0 };
un_CANFD_CH_ILS_t   unILS   = { 0 };
un_CANFD_CH_ILE_t   unILE   = { 0 };
un_CANFD_CH_RXFTOP_CTL_t unRXFTOP_CTL = { 0 };

/* Check for NULL pointers */
if ( pstcCanFD  == NULL ||
     pstcConfig == NULL ||
     /*  Check if configuration parameter values are valid.  */
     ((pstcConfig->sidFilterConfig.numberOfSIDFilters != 0) && (pstcConfig-
>sidFilterConfig.sidFilter == NULL)) ||((pstcConfig->extidFilterConfig.numberOfEXTIDFilters !=
0) && (pstcConfig->extidFilterConfig.extidFilter == NULL))
      )
  {
return CY_CANFD_BAD_PARAM;
  }


/* Get pointer to internal data structure */
pstcCanFDContext = Cy_CANFD_GetContext(pstcCanFD);
```

4 CAN FD 設定

```
/* Check for NULL */
if (pstcCanFDContext == NULL)
  {
return CY_CANFD_BAD_PARAM;
  }
/* Set notification callback functions */
/*  Configure interrupt handler for callback events.  */
pstcCanFDContext->canFDInterruptHandling.canFDTxInterruptFunction  = pstcConfig->txCallback;
pstcCanFDContext->canFDInterruptHandling.canFDRxInterruptFunction  = pstcConfig->rxCallback;
pstcCanFDContext->canFDInterruptHandling.canFDRxWithTopPtrInterruptFunction = pstcConfig-
>rxFifoWithTopCallback;
pstcCanFDContext->canFDNotificationCb.canFDStatusInterruptFunction = pstcConfig->statusCallback;
pstcCanFDContext->canFDNotificationCb.canFDErrorInterruptFunction  = pstcConfig->errorCallback;

/* Get the pointer to M_TTCAN of the CAN FD channel */
/*  Get base address of CANx channel register.  */
pstcCanFDChMTTCAN = &pstcCanFD->M_TTCAN;

/* Set CCCR.INIT to 1 and wait until it will be updated. */
pstcCanFDChMTTCAN->unCCCR.u32Register = 0x1ul;
while(pstcCanFDChMTTCAN->unCCCR.stcField.u1INIT != 1)
    {
    }

    /* Cancel protection by setting CCE */
    /*  (1) Stop CAN FD communication and enable to write to Protected Configuration Register.
*/
    pstcCanFDChMTTCAN->unCCCR.u32Register = 0x3ul;


    /* Standard ID filter */
    /*  (2-1) Configure Standard ID filter.  */
    unSIDFC.stcField.u8LSS = pstcConfig->sidFilterConfig.numberOfSIDFilters;  // Number of SID
filters
    unSIDFC.stcField.u14FLSSA = stcMsgramConfig.offset >> 2;    // Start address (word) of SID
filter configuration in message RAM
    pstcCanFDChMTTCAN->unSIDFC.u32Register = unSIDFC.u32Register;

    /* Extended ID filter */
    unXIDFC.stcField.u7LSE = pstcConfig->extidFilterConfig.numberOfEXTIDFilters; // Number of
ext id filters
    unXIDFC.stcField.u14FLESA = pstcCanFDChMTTCAN->unSIDFC.stcField.u14FLSSA +
      (pstcConfig->sidFilterConfig.numberOfSIDFilters * SIZE_OF_SID_FILTER_IN_WORD); // Start
address (word) of ext id filter configuration in message RAM
      /*  (2-2) Configure Extended ID Filter.Start address is placed under Standard ID filter
area.  */
    pstcCanFDChMTTCAN->unXIDFC.u32Register = unXIDFC.u32Register;

    /* Extended ID AND Mask */
    unXIDAM.stcField.u29EIDM = pstcConfig->extidFilterConfig.extIDANDMask;
    /*  (2-3) Configure Extended ID Mask.  */
    pstcCanFDChMTTCAN->unXIDAM.u32Register = unXIDAM.u32Register;
```

```
    /* Configuration of Rx Buffer and Rx FIFO */
    unRXESC.stcField.u3RBDS = pstcConfig->rxBufferDataSize;
    unRXESC.stcField.u3F1DS = pstcConfig->rxFifo1DataSize;
    unRXESC.stcField.u3F0DS = pstcConfig->rxFifo0DataSize;
    /*  (3-1) Configure Rx Buffer/FIFO Element size.  */
    pstcCanFDChMTTCAN->unRXESC.u32Register = unRXESC.u32Register;

    /* Rx FIFO 0 */
    unRXF0C.stcField.u1F0OM = pstcConfig->rxFifo0Config.mode;
    unRXF0C.stcField.u7F0WM = pstcConfig->rxFifo0Config.watermark;
    unRXF0C.stcField.u7F0S = pstcConfig->rxFifo0Config.numberOfFifoElements;
    unRXF0C.stcField.u14F0SA = pstcCanFDChMTTCAN->unXIDFC.stcField.u14FLESA +
    /*  (3-2) Configure Rx FIFO 0. Start address is placed under Extend ID filter area.  */
      (pstcConfig->extidFilterConfig.numberOfEXTIDFilters * SIZE_OF_EXTID_FILTER_IN_WORD);
    pstcCanFDChMTTCAN->unRXF0C.u32Register = unRXF0C.u32Register;

    /* Rx FIFO 1 */
    unRXF1C.stcField.u1F1OM = pstcConfig->rxFifo1Config.mode;
    unRXF1C.stcField.u7F1WM = pstcConfig->rxFifo1Config.watermark;
    unRXF1C.stcField.u7F1S = pstcConfig->rxFifo1Config.numberOfFifoElements;
    unRXF1C.stcField.u14F1SA = pstcCanFDChMTTCAN->unRXF0C.stcField.u14F0SA +
    /*  (3-2) Configure Rx FIFO Start address is placed under Rx FIFO 0 area.  */
      (pstcConfig->rxFifo0Config.numberOfFifoElements * (2 + dataBufferSizeInWord[pstcConfig-
>rxFifo0DataSize]));
    pstcCanFDChMTTCAN->unRXF1C.u32Register = unRXF1C.u32Register;

    /* Rx FIFO 0,1 Top pointer logic config */
    /*  (3-3) Configure RxFIFO 0,1 Top pointer logic.  */
    unRXFTOP_CTL.stcField.u1F0TPE = (pstcConfig->rxFifo0Config.topPointerLogicEnabled ==
false) ? 0 : 1;
    unRXFTOP_CTL.stcField.u1F1TPE = (pstcConfig->rxFifo1Config.topPointerLogicEnabled ==
false) ? 0 : 1;
    pstcCanFD->unRXFTOP_CTL.u32Register = unRXFTOP_CTL.u32Register;

    /* Rx buffer */
    /*  (3-4) Configure Rx Buffer. Start address is placed under Rx FIFO 1 area.  */
    unRXBC.stcField.u14RBSA = pstcCanFDChMTTCAN->unRXF1C.stcField.u14F1SA +
      (pstcConfig->rxFifo1Config.numberOfFifoElements * (2 + dataBufferSizeInWord[pstcConfig-
>rxFifo1DataSize]));
    pstcCanFDChMTTCAN->unRXBC.u32Register = unRXBC.u32Register;

    /* Configuration of Tx Buffer and Tx FIFO/Queue */
    unTXESC.stcField.u3TBDS = pstcConfig->txBufferDataSize;
    /*  (3-5) Configure Tx Buffer Element Size.  */
    pstcCanFDChMTTCAN->unTXESC.u32Register = unTXESC.u32Register;

    /* Tx FIFO/QUEUE (not use) */
    unTXEFC.stcField.u6EFWM = 0;  /* Watermark interrupt disabled */
    unTXEFC.stcField.u6EFS = 0;   /* Tx Event FIFO disabled */
    /*  (3-6) Configure Tx Event FIFO. Start address is placed under Tx Buffer area.  */
    unTXEFC.stcField.u14EFSA = pstcCanFDChMTTCAN->unRXBC.stcField.u14RBSA +
      (pstcConfig->noOfRxBuffers * (2 + dataBufferSizeInWord[pstcConfig->rxBufferDataSize]));
```

```
    pstcCanFDChMTTCAN->unTXEFC.u32Register = unTXEFC.u32Register;

    /* Tx buffer */
    unTXBC.stcField.u1TFQM = 0;   /* Tx FIFO operation */
    unTXBC.stcField.u6TFQS = 0;   /* No Tx FIFO/Queue */
    unTXBC.stcField.u6NDTB = pstcConfig->noOfTxBuffers;   /* Number of Dedicated Tx Buffers */
    /*  (3-7) Configure Tx Buffer. Start address is placed under Tx FIFO area.  */
    unTXBC.stcField.u14TBSA = pstcCanFDChMTTCAN->unTXEFC.stcField.u14EFSA +
      (10 * SIZE_OF_TXEVENT_FIFO_IN_WORD);    // Reserving memory for 10 TxEvent Fifo elements
for easy future use
    pstcCanFDChMTTCAN->unTXBC.u32Register = unTXBC.u32Register;

    /* Initialize message RAM area(Entire region zeroing) */
    pu32Adrs = (uint32_t *)(((uint32_t)pstcCanFD & 0xFFFF0000ul) +
(uint32_t)CY_CANFD_MSGRAM_START + stcMsgramConfig.offset);
    u32SizeInWord = stcMsgramConfig.size >> 2;   /*  (4) Clear the Message RAM area.  */
    for(u32Count = 0; u32Count < u32SizeInWord; u32Count++)
    {
        *pu32Adrs++ = 0ul;
    }

    /* Configuration of CAN bus */
    /* CCCR register */
    unCCCR.stcField.u1TXP = 0;     /* Transmit pause disabled */
    unCCCR.stcField.u1BRSE = ((pstcConfig->canFDMode == true) ? 1 : 0);    /* Bit rate switch */
    unCCCR.stcField.u1FDOE = ((pstcConfig->canFDMode == true) ? 1 : 0);    /* FD operation */
    unCCCR.stcField.u1TEST = 0;    /* Normal operation */
    unCCCR.stcField.u1DAR = 0;     /* Automatic retransmission enabled */
    unCCCR.stcField.u1MON_ = 0;    /* Bus Monitoring Mode is disabled */
    unCCCR.stcField.u1CSR = 0;     /* No clock stop is requested */
    unCCCR.stcField.u1ASM = 0;     /* Normal CAN operation. */
    /*  (5) Select CAN/CAN FD Mode and Set Bit Rate Switch.  */
    pstcCanFDChMTTCAN->unCCCR.u32Register = unCCCR.u32Register;

    /* Nominal Bit Timing & Prescaler Register */
    unNBTP.stcField.u9NBRP = pstcConfig->bitrate.prescaler;
    unNBTP.stcField.u8NTSEG1 = pstcConfig->bitrate.timeSegment1;
    unNBTP.stcField.u7NTSEG2 = pstcConfig->bitrate.timeSegment2;
    unNBTP.stcField.u7NSJW = pstcConfig->bitrate.syncJumpWidth;
    /* (6-1) Configure Nominal Bit Timing & Prescaler. */
    pstcCanFDChMTTCAN->unNBTP.u32Register = unNBTP.u32Register;


    if(pstcConfig->canFDMode == true)
    {
        /* Data Bit Timing & Prescaler */
        unDBTP.stcField.u5DBRP = pstcConfig->fastBitrate.prescaler;
        unDBTP.stcField.u5DTSEG1 = pstcConfig->fastBitrate.timeSegment1;
        /*  (6-2) Configure Data Bit Timing & Prescaler.This configuration is only for CAN FD
mode.  */
        unDBTP.stcField.u4DTSEG2 = pstcConfig->fastBitrate.timeSegment2;
        unDBTP.stcField.u4DSJW = pstcConfig->fastBitrate.syncJumpWidth;
        unDBTP.stcField.u1TDC = ((pstcConfig->tdcConfig.tdcEnabled == true) ? 1 : 0); /*
```

```
Transceiver Delay Compensation enabled */
        pstcCanFDChMTTCAN->unDBTP.u32Register = unDBTP.u32Register;

        /* Transmitter Delay Compensation */
        /*  (6-3) Configure Transmitter Delay Compensation. This configuration is only for CAN
FD mode.  */
        unTDCR.stcField.u7TDCO = pstcConfig->tdcConfig.tdcOffset;       /* Transmitter Delay
Compensation Offset */
        unTDCR.stcField.u7TDCF = pstcConfig->tdcConfig.tdcFilterWindow; /* Transmitter Delay
Compensation Filter Window Length */
        pstcCanFDChMTTCAN->unTDCR.u32Register = unTDCR.u32Register;
    }

    /* Configuration of Global Filter */
    unGFC.stcField.u2ANFS = pstcConfig->globalFilterConfig.nonMatchingFramesStandard;
    /*  (7-1) Configure Global Filter.  */
    unGFC.stcField.u2ANFE = pstcConfig->globalFilterConfig.nonMatchingFramesExtended;
    unGFC.stcField.u1RRFS = ((pstcConfig->globalFilterConfig.rejectRemoteFramesStandard ==
true) ? 1 : 0);
    unGFC.stcField.u1RRFE = ((pstcConfig->globalFilterConfig.rejectRemoteFramesExtended ==
true) ? 1 : 0);
    pstcCanFDChMTTCAN->unGFC.u32Register = unGFC.u32Register;

    /* Standard Message ID Filters */
    /*  (7-2) Configure Standard ID Filter.  */
    pstcSIDFilter = (cy_stc_id_filter_t *)(((uint32_t)pstcCanFD & 0xFFFF0000ul) +
                                           (uint32_t)CY_CANFD_MSGRAM_START      +
                                           (pstcCanFDChMTTCAN->unSIDFC.stcField.u14FLSSA <<
2u));
    for(u32Count = 0; u32Count < pstcConfig->sidFilterConfig.numberOfSIDFilters; u32Count++)
    {
        pstcSIDFilter[u32Count] = pstcConfig->sidFilterConfig.sidFilter[u32Count];
    }

    /* Extended Message ID Filters */
    /*  (7-3) Configure Extended ID Filter.  */
    pstcEXTIDFilter = (cy_stc_extid_filter_t *)(((uint32_t)pstcCanFD & 0xFFFF0000ul) +
                                                (uint32_t)CY_CANFD_MSGRAM_START      +
                                                (pstcCanFDChMTTCAN->unXIDFC.stcField.u14FLESA
<< 2u));
    for(u32Count = 0; u32Count < pstcConfig->extidFilterConfig.numberOfEXTIDFilters; u32Count++)
    {
        pstcEXTIDFilter[u32Count] = pstcConfig->extidFilterConfig.extidFilter[u32Count];
    }

    /* Configuration of Interrupt */
    /* Interrupt Enable */
    unIE.stcField.u1ARAE  = 0; /* Access to Reserved Address */
    unIE.stcField.u1PEDE  = 0; /* Protocol Error in Data Phase */
    unIE.stcField.u1PEAE  = 0; /* Protocol Error in Arbitration Phase */
    unIE.stcField.u1WDIE  = 0; /* Watchdog */
    unIE.stcField.u1BOE   = 0; /* Bus_Off Status */
    unIE.stcField.u1EWE   = 0; /* Warning Status */
```

```
    unIE.stcField.u1EPE   = 0; /* Error Passive */
    unIE.stcField.u1ELOE  = 0; /* Error Logging Overflow */
    unIE.stcField.u1BEUE  = 0; /* Bit Error Uncorrected */
    unIE.stcField.u1BECE  = 0; /* Bit Error Corrected */
    unIE.stcField.u1DRXE  = 1; /* Message stored to Dedicated Rx Buffer */
    unIE.stcField.u1TOOE  = 0; /* Timeout Occurred */
    unIE.stcField.u1MRAFE = 0; /* Message RAM Access Failure */
    unIE.stcField.u1TSWE  = 0; /* Timestamp Wraparound */
    unIE.stcField.u1TEFLE = 0; /* Tx Event FIFO Event Lost */
    unIE.stcField.u1TEFFE = 0; /* Tx Event FIFO Full */
    unIE.stcField.u1TEFWE = 0; /* Tx Event FIFO Watermark Reached */
    unIE.stcField.u1TEFNE = 0; /* Tx Event FIFO New Entry */
    unIE.stcField.u1TFEE  = 0; /* Tx FIFO Empty */
    unIE.stcField.u1TCFE  = 0; /* Transmission Cancellation Finished */
    unIE.stcField.u1TCE   = 0; /* Transmission Completed */
    unIE.stcField.u1HPME  = 0; /* High Priority Message */
    unIE.stcField.u1RF1LE = 0; /* Rx FIFO 1 Message Lost */
    unIE.stcField.u1RF1FE = 0; /* Rx FIFO 1 Full */
    unIE.stcField.u1RF1WE = 0; /* Rx FIFO 1 Watermark Reached */
    unIE.stcField.u1RF1NE = 1; /* Rx FIFO 1 New Message */
    unIE.stcField.u1RF0LE = 0; /* Rx FIFO 0 Message Lost */
    unIE.stcField.u1RF0FE = 0; /* Rx FIFO 0 Full */
    unIE.stcField.u1RF0WE = 0; /* Rx FIFO 0 Watermark Reached */
    unIE.stcField.u1RF0NE = 1; /* Rx FIFO 0 New Message */    /*  (8-1) Configure Interrupt
Enable.  */
    pstcCanFDChMTTCAN->unIE.u32Register = unIE.u32Register;

    /* Interrupt Line Select */
    unILS.stcField.u1ARAL  = 0; /* Access to Reserved Address */
    unILS.stcField.u1PEDL  = 0; /* Protocol Error in Data Phase */
    unILS.stcField.u1PEAL  = 0; /* Protocol Error in Arbitration Phase */
    unILS.stcField.u1WDIL  = 0; /* Watchdog */
    unILS.stcField.u1BOL   = 0; /* Bus_Off Status */
    unILS.stcField.u1EWL   = 0; /* Warning Status */
    unILS.stcField.u1EPL   = 0; /* Error Passive */
    unILS.stcField.u1ELOL  = 0; /* Error Logging Overflow */
    unILS.stcField.u1BEUL  = 0; /* Bit Error Uncorrected */
    unILS.stcField.u1BECL  = 0; /* Bit Error Corrected */
    unILS.stcField.u1DRXL  = 0; /* Message stored to Dedicated Rx Buffer */
    unILS.stcField.u1TOOL  = 0; /* Timeout Occurred */
    unILS.stcField.u1MRAFL = 0; /* Message RAM Access Failure */
    unILS.stcField.u1TSWL  = 0; /* Timestamp Wraparound */
    unILS.stcField.u1TEFLL = 0; /* Tx Event FIFO Event Lost */
    unILS.stcField.u1TEFFL = 0; /* Tx Event FIFO Full */
    unILS.stcField.u1TEFWL = 0; /* Tx Event FIFO Watermark Reached */
    unILS.stcField.u1TEFNL = 0; /* Tx Event FIFO New Entry */
    unILS.stcField.u1TFEL  = 0; /* Tx FIFO Empty */
    unILS.stcField.u1TCFL  = 0; /* Transmission Cancellation Finished */
    unILS.stcField.u1TCL   = 0; /* Transmission Completed */
    unILS.stcField.u1HPML  = 0; /* High Priority Message */
    unILS.stcField.u1RF1LL = 0; /* Rx FIFO 1 Message Lost */
    unILS.stcField.u1RF1FL = 0; /* Rx FIFO 1 Full */
    unILS.stcField.u1RF1WL = 0; /* Rx FIFO 1 Watermark Reached */
```

```
    unILS.stcField.u1RF1NL = 0; /* Rx FIFO 1 New Message */
    unILS.stcField.u1RF0LL = 0; /* Rx FIFO 0 Message Lost */
    unILS.stcField.u1RF0FL = 0; /* Rx FIFO 0 Full */
    unILS.stcField.u1RF0WL = 0; /* Rx FIFO 0 Watermark Reached */
    unILS.stcField.u1RF0NL = 0; /* Rx FIFO 0 New Message */    /*  (8-2) Configure Interrupt
line Select.  */
    pstcCanFDChMTTCAN->unILS.u32Register = unILS.u32Register;

    /* Interrupt Line Enable */
    unILE.stcField.u1EINT0 = 1;  /* Enable Interrupt Line 0 */    /*  (8-3) Configure Interrupt
line Enable.  */
    unILE.stcField.u1EINT1 = 0;   /* Disable Interrupt Line 1 */
    pstcCanFDChMTTCAN->unILE.u32Register = unILE.u32Register;

    /* CAN-FD operation start */
    /* Set CCCR.INIT to 0 and wait until it will be updated */
    unCCCR.stcField.u1INIT = 0;   /*  (9) Start CAN FD communication.  */
    pstcCanFDChMTTCAN->unCCCR.u32Register = unCCCR.u32Register;
    while(pstcCanFDChMTTCAN->unCCCR.stcField.u1INIT != 0)
    {
    }

    return CY_CANFD_SUCCESS;
}
```

4.3 メッセージ送信

図 6 は、メッセージ送信フロー例です。この例では、Tx 割込みは使用しません。このフローでは、(0) が構成部分で実行され、(1) から(5) がドライバ部分で実行されます。

メッセージは、メッセージ RAM 領域の Tx バッファを介して送信されます。保留中の要求 (TXBRP) がないことを確認します。保留中の要求がない場合は、メッセージ RAM の Tx バッファアドレスを計算し、CAN FD コントローラが送信するフレームの制御情報とデータを書き込みます。メッセージ送信は、Tx バッファ追加要求レジスタ (TXBAR) の書込みによって開始されます。

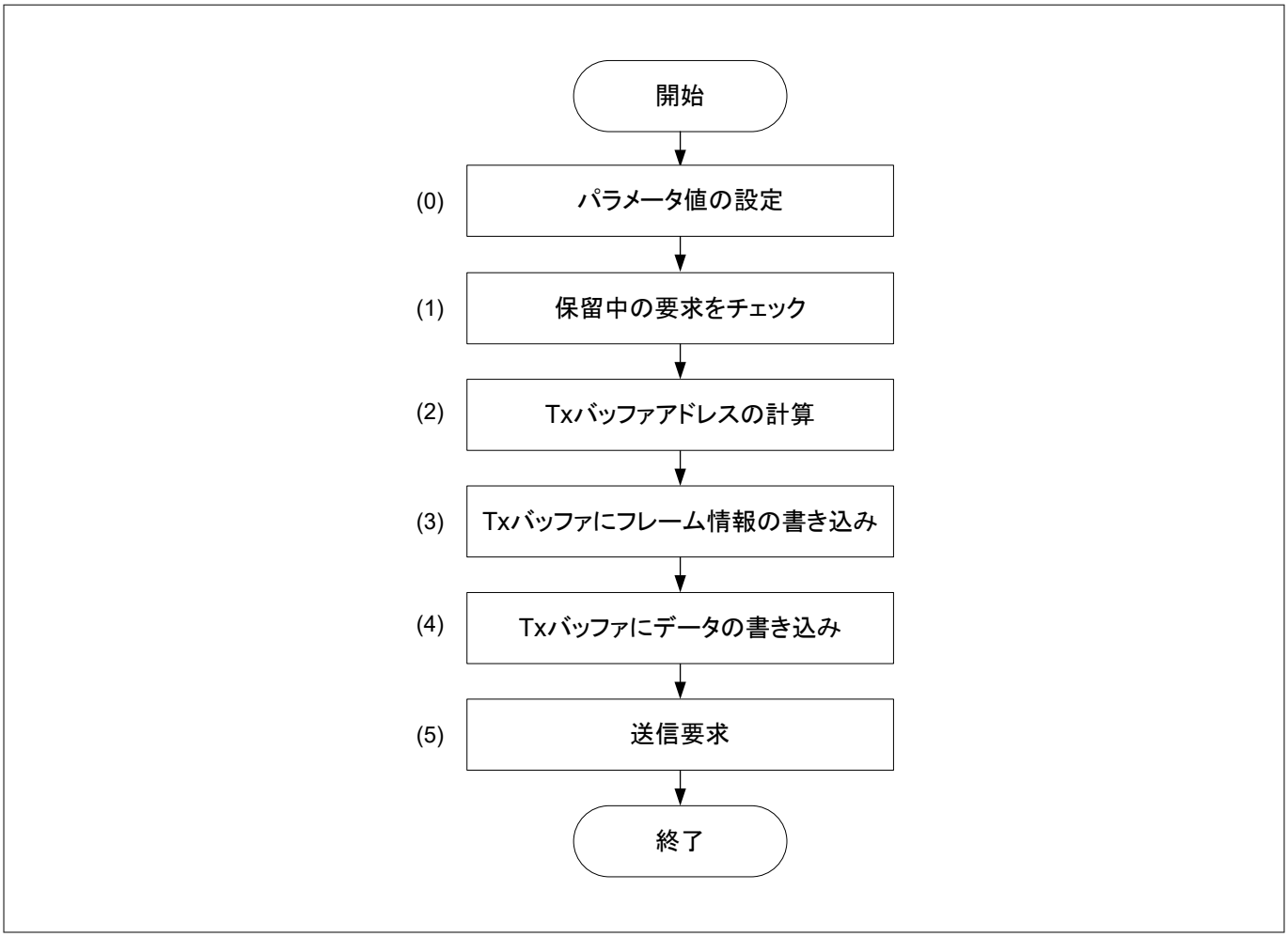

図 6　　メッセージ送信フロー例

4.3.1　　ユースケース

ここでは、以下のユースケースと 4.2.1 ユースケースで説明したユースケースを使用した CAN FD メッセージ送信の例について説明します。CAN FD メッセージ送信は、SDL を使用して構成されます。

ユースケース:

- FD フォーマット (FDF): 1 (CAN FD フォーマットで送信されたフレーム)
  - ビットレートスイッチング (BRS): 1 (CAN FD フレームはビットレートスイッチングで送信されます)
- 拡張識別子 (XTD): 0 (11 ビット標準識別子)
- 識別子 (ID): 0x525
- データ長コード (DLC): 15

4.3.2　　構成

表 4 に、メッセージ送信用の SDL の構成部分のパラメータを示します。

4 CAN FD 設定

表 4　　メッセージ送信パラメーター覧

パラメーター	説明	値
.canFDFormat	CAN フレームフォーマットの選択 True: CAN FD, False: クラシック CAN	true
.idConfig.extended	CAN ID フォーマットの選択 True: 拡張 ID, False: 標準 ID	false
.idConfig.identifier	CAN ID の設定	0x525
.dataConfig.dataLengthCode	CAN データ長コードの設定	15
.dataConfig.data	送信データの設定	任意の値

Code Listing 4 は、構成部での CAN FD メッセージ送信のサンプルプログラムを示します。

Code Listing 4 構成部でのメッセージ送信の例

```
int main(void)
{
   :
/* Prepare CANFD message to transmit*/
   cy_stc_canfd_msg_t stcMsg;

   stcMsg.canFDFormat = true;                /* CAN FD format */
   stcMsg.idConfig.extended = false;         /*  11-bit standard identifier  */
   stcMsg.idConfig.identifier = 0x525;       /*  CAN ID  */
   stcMsg.dataConfig.dataLengthCode = 15;    /*  CAN Data Length Code  */
   stcMsg.dataConfig.data[0]  = 0x70190523;  /*  Transmission data  */
   stcMsg.dataConfig.data[1]  = 0x70190819;  /*  Transmission data  */
   stcMsg.dataConfig.data[2]  = 0x33332222;  /*  Transmission data  */
   stcMsg.dataConfig.data[3]  = 0x33332222;  /*  Transmission data  */
   stcMsg.dataConfig.data[4]  = 0x55554444;  /*  Transmission data  */
   stcMsg.dataConfig.data[5]  = 0x77776666;  /*  Transmission data  */
   stcMsg.dataConfig.data[6]  = 0x99998888;  /*  Transmission data  */
   stcMsg.dataConfig.data[7]  = 0xBBBBAAAA;  /*  Transmission data  */
   stcMsg.dataConfig.data[8]  = 0xDDDDCCCC;  /*  Transmission data  */
   stcMsg.dataConfig.data[9]  = 0xFFFFEEEE;  /*  Transmission data  */
   stcMsg.dataConfig.data[10] = 0x78563412;
   stcMsg.dataConfig.data[11] = 0x00000000;
   stcMsg.dataConfig.data[12] = 0x11111111;
   stcMsg.dataConfig.data[13] = 0x22222222;
   stcMsg.dataConfig.data[14] = 0x33333333;
   stcMsg.dataConfig.data[15] = 0x44444444;

/*  CAN Transmission setting
CAN0 Channel 0
Message buffer 0
Transmission data
*/
Cy_CANFD_UpdateAndTransmitMsgBuffer(CY_CANFD_TYPE, 0, &stcMsg);
:
}
```

4.3.3 メッセージ送信のサンプルプログラム

Code Listing 5 は、ドライバ部の CAN FD メッセージ送信のプログラム例を示します。

Code Listing 5 ドライバ部のメッセージ送信の例

```
cy_en_canfd_status_t Cy_CANFD_UpdateAndTransmitMsgBuffer( cy_pstc_canfd_type_t pstcCanFD,
uint8_t u8MsgBuf, cy_stc_canfd_msg_t* pstcMsg )
{
    // Local variable declarations
    cy_stc_canfd_context_t* pstcCanFDContext;
    uint16_t u16DlcTemp;
    uint16_t u16Count;
    uint8_t u8DataLengthWord;

    cy_stc_canfd_tx_buffer_t* pstcCanFDTxBuffer;
    volatile stc_CANFD_CH_M_TTCAN_t* pstcCanFDChMTTCAN;

    /* Check for NULL pointers */
    /*  Check if configuration parameter values are valid  */
    if ( pstcCanFD  == NULL ||
         pstcMsg == NULL
       )
    {
        return CY_CANFD_BAD_PARAM;
    }

    if(u8MsgBuf > 31)
    {
        return CY_CANFD_BAD_PARAM;
    }

    /* Get pointer to internal data structure */
    pstcCanFDContext = Cy_CANFD_GetContext(pstcCanFD);

    /* Check for NULL */
    if (pstcCanFDContext == NULL)
    {
        return CY_CANFD_BAD_PARAM;
    }

    /* Get the pointer to M_TTCAN of the CAN FD channel */
    pstcCanFDChMTTCAN = &pstcCanFD->M_TTCAN;

    /* Check if CAN FD controller is in not in INIT state and Tx buffer is empty or not */
    if((pstcCanFDChMTTCAN->unCCCR.stcField.u1INIT != 0) ||
       ((pstcCanFDChMTTCAN->unTXBRP.u32Register & (1ul << u8MsgBuf)) != 0)   /* (1) Check
Pending Request */
      )
    {
        return CY_CANFD_BAD_PARAM;
    }

    /* Get Tx Buffer address */
    /* (2) Get the Tx Buffer Address with a calculation function.  (See Code Listing 5) */
    pstcCanFDTxBuffer = (cy_stc_canfd_tx_buffer_t*)Cy_CANFD_CalcTxBufAdrs(pstcCanFD, u8MsgBuf);
```

```
    if(pstcCanFDTxBuffer == NULL)
    {
        return CY_CANFD_BAD_PARAM;
    }

    pstcCanFDTxBuffer->t0_f.rtr = 0; /* Transmit data frame. */
    pstcCanFDTxBuffer->t0_f.xtd = (pstcMsg->idConfig.extended == true) ? 1 : 0;
    pstcCanFDTxBuffer->t0_f.id  = (pstcCanFDTxBuffer->t0_f.xtd == 0) ?
                                        (pstcMsg->idConfig.identifier << 18) : pstcMsg-
>idConfig.identifier;

    pstcCanFDTxBuffer->t1_f.efc = 0; /* Tx Event Fifo not used *//*(3) Write Frame Information
to TX Buffer */
    pstcCanFDTxBuffer->t1_f.mm  = 0; /* Not used */
    pstcCanFDTxBuffer->t1_f.dlc = pstcMsg->dataConfig.dataLengthCode;
    pstcCanFDTxBuffer->t1_f.fdf = (pstcMsg->canFDFormat == true) ? 1 : 0;
    pstcCanFDTxBuffer->t1_f.brs = (pstcMsg->canFDFormat == true) ? 1 : 0;

    /* Convert the DLC to data byte word */
    if (pstcMsg->dataConfig.dataLengthCode < 8 )
    {
        u16DlcTemp = 0;
    }
    else
    {
        u16DlcTemp = pstcMsg->dataConfig.dataLengthCode - 8;
    }
    u8DataLengthWord = dataBufferSizeInWord[u16DlcTemp];

    /* Data set */
    for ( u16Count = 0; u16Count < u8DataLengthWord; u16Count++ )  /*(4) Write Data to Tx
Buffer */
    {
        pstcCanFDTxBuffer->data_area_f[u16Count] = pstcMsg->dataConfig.data[u16Count];
    }

    /*  (5) Transmission Request  */
    pstcCanFDChMTTCAN->unTXBAR.u32Register = 1ul << u8MsgBuf;     // Transmit buffer add request

    return CY_CANFD_SUCCESS;
}
```

Code Listing 6 は、ドライバ部分での Tx バッファアドレス計算のサンプルプログラムを示します。

Code Listing 6 ドライバ部の Tx バッファアドレス計算の例

```
static uint32_t* Cy_CANFD_CalcTxBufAdrs(cy_pstc_canfd_type_t pstcCanFD, uint8_t u8MsgBuf)
{
    uint32_t* pu32Adrs;

    if ( u8MsgBuf > 31)
    {
        /* Set 0 to the return value if the index is invalid */
        pu32Adrs = NULL;
    }
    else
    {
        /* Set the message buffer address to the return value if the index is available */
        pu32Adrs = (uint32_t*)(((uint32_t)pstcCanFD & 0xFFFF0000ul) +
(uint32_t)CY_CANFD_MSGRAM_START);
        pu32Adrs += pstcCanFD->M_TTCAN.unTXBC.stcField.u14TBSA;
        /*  Calculate Tx Buffer Address  */
        pu32Adrs += u8MsgBuf * (2 + dataBufferSizeInWord[pstcCanFD-
>M_TTCAN.unTXESC.stcField.u3TBDS]);
    }
    return pu32Adrs;
}
```

4.4 メッセージ受信

フィルタ構成に基づいて、メッセージ受信は専用の Rx バッファまたは Rx FIFO 0/1 で行います。ここでは、メッセージの受信方法について説明します。

4.4.1 専用 Rx バッファによるメッセージ受信

図 7 に専用 Rx バッファと Rx 割込みを使用したメッセージ受信フローの例を示します。

受信メッセージが受入れフィルタリングを経由してメッセージ RAM 領域の専用 Rx バッファの 1 つに格納されると、Rx 割込みがイネーブルの場合、割込みが発生します。メッセージが専用 Rx バッファに格納されると、割込みレジスタ (IR.DRX) とニューデータレジスタ 1/2 (NDAT 1/2) の対応するビットがセットされます。割込みハンドリングは、受信したメッセージを保持しているメッセージ RAM 内の Rx バッファの絶対アドレスの計算と、計算されたアドレスから受信したメッセージ情報の読出しを含みます。メッセージが Rx バッファから読み出された後、この Rx バッファが次のメッセージを受信できるように、NDAT 1/2 レジスタ内の対応するフラグをクリアする必要があります。

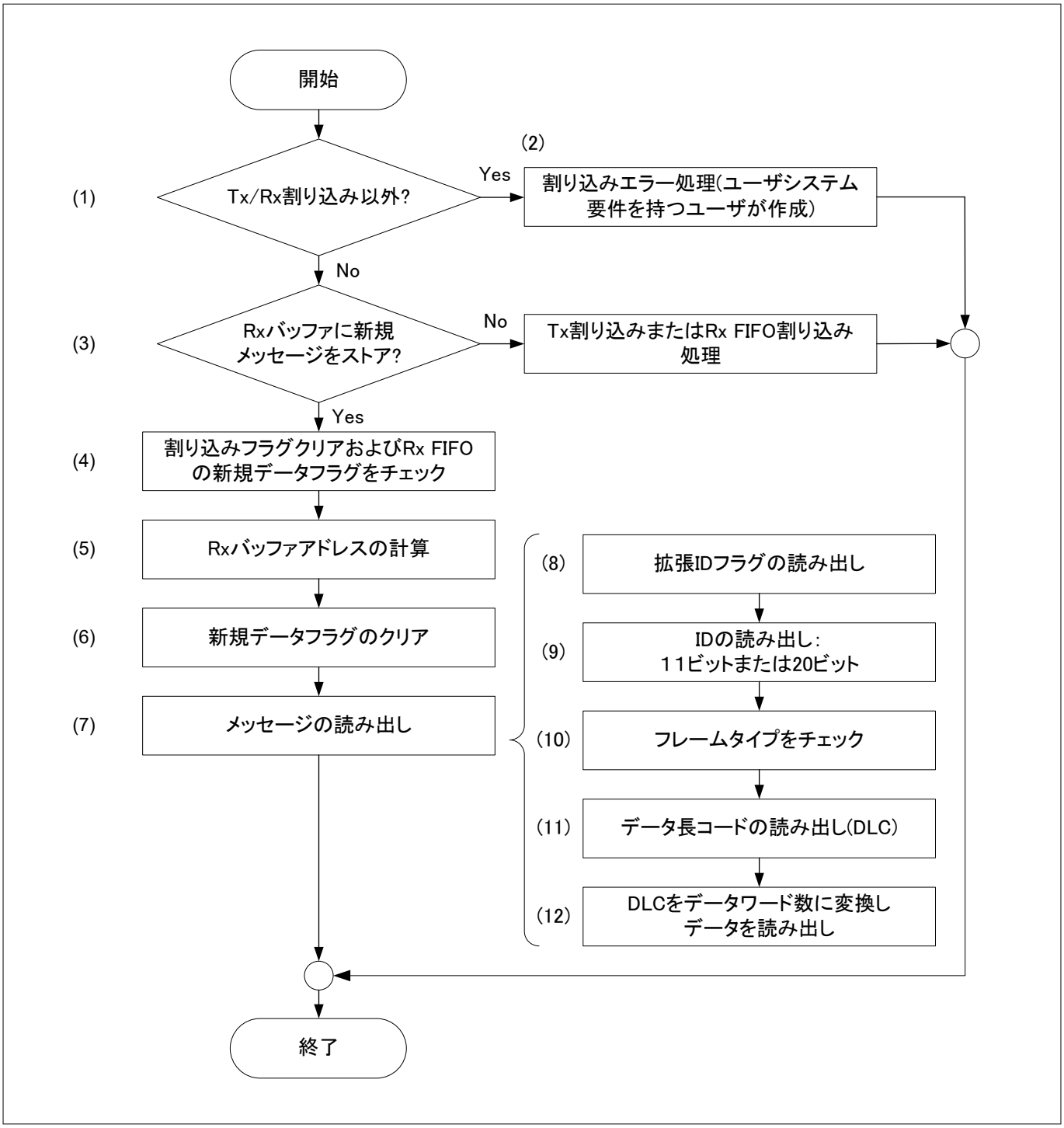

図 7　　専用 Rx バッファによるメッセージ受信フロー例

4.4.1.1　ユースケース

ここでは、4.2.1 ユースケースで説明した使用例を使用して、専用の Rx バッファでメッセージを受信する例を説明します。

4.4.1.2　構成

ここでは、CAN FD コントローラの構成で説明した構成を使用した専用 Rx バッファでのメッセージ受信の例を説明します。

4.4.1.3 専用受信バッファでのメッセージ受信のサンプルプログラム

Code Listing 7 は、専用の Rx バッファでのメッセージ受信のプログラム例を示します。このプログラムは、専用の Rx バッファ 0～31 および Rx 割込みを使用します。

Code Listing 7 専用受信バッファでのメッセージ受信の例

```
void Cy_CANFD_IrqHandler( cy_pstc_canfd_type_t pstcCanFD )
{
    uint32_t*                       pu32Adrs = 0;
    uint8_t                         u8MessageBufferNumber ;
    cy_stc_canfd_msg_t              stcCanFDmsg;
    volatile stc_CAN_CH_M_TTCAN_t*  pstcCanFDChMTTCAN;
    cy_stc_canfd_context_t*         pstcCanFDContext;
    uint8_t                         u8BufferSizeTemp = 0;
    uint32_t                        au32RxBuf[18];

    /* Get pointer to internal data structure */
    pstcCanFDContext = Cy_CANFD_GetContext(pstcCanFD);

    /* Get the pointer to M_TTCAN of the CAN FD channel */
    pstcCanFDChMTTCAN = &pstcCanFD->M_TTCAN;

    /* Other than Tx/Rx interrupt occurred */
    /*  (1) Check for interrupts other than Tx/Rx   */
    if (pstcCanFDChMTTCAN->unIR.u32Register & 0x3ff7E0EE)
    {
    /* (2) Error Interrupt handling (created by user with user system requirement)*/
        Cy_CANFD_ErrorHandling(pstcCanFD);
    }

    /* (3) Check if New Message stored in Dedicated Rx Buffer  */
    if(pstcCanFDChMTTCAN->unIR.stcField.u1DRX == 1) /* At least one message stored into an Rx
Buffer */
    {
        /* Clear the Message stored to Dedicated Rx Buffer flag */
        /* (4) Clear Interrupt flag and check New Data flag of Dedicated Rx Buffers 0 -31 */
        pstcCanFDChMTTCAN->unIR.stcField.u1DRX = 1UL;

        if(pstcCanFDChMTTCAN->unNDAT1.u32Register != 0)       // Message buffers 0-31
        {
            for(u8MessageBufferNumber = 0; u8MessageBufferNumber < 32; u8MessageBufferNumber++)
            {
                if((pstcCanFDChMTTCAN->unNDAT1.u32Register & (1ul << u8MessageBufferNumber)) !=
0)
                {
                    // Calculate Rx Buffer address
                    /* (5) Get the Rx Buffer address for which New Data flag is set with a
calculation function (See Code Listing 8) */
                    pu32Adrs = Cy_CANFD_CalcRxBufAdrs(pstcCanFD, u8MessageBufferNumber);

                    // Clear NDAT1 register
                    /* (6) Clear New Data flag */
                    pstcCanFDChMTTCAN->unNDAT1.u32Register = (1ul << u8MessageBufferNumber);

                    break;
                }
            }
```

```
        }
        else if(pstcCanFDChMTTCAN->unNDAT2.u32Register != 0)  // Message buffers 32-63
        {
            for(u8MessageBufferNumber = 0; u8MessageBufferNumber < 32; u8MessageBufferNumber++)
            {
                if((pstcCanFDChMTTCAN->unNDAT2.u32Register & (1ul << u8MessageBufferNumber)) !=
0)
                {
                    u8MessageBufferNumber += 32;

                    // Calculate Rx Buffer address
                    pu32Adrs = Cy_CANFD_CalcRxBufAdrs(pstcCanFD, u8MessageBufferNumber);

                    // Clear NDAT2 register
                    pstcCanFDChMTTCAN->unNDAT2.u32Register = (1ul << (u8MessageBufferNumber -
32));

                    break;
                }
            }
        }
        if(pu32Adrs)
        {
            /* (7) Read the message from Rx Buffer (See Code Listing 9) */
            Cy_CANFD_ExtractMsgFromRXBuffer((cy_stc_canfd_rx_buffer_t *) pu32Adrs,
&stcCanFDmsg);

            /* CAN-FD message received, check if there is a callback function */
            /* Call callback function if it was set previously. */
            if (pstcCanFDContext->canFDInterruptHandling.canFDRxInterruptFunction != NULL)
            {
               /* Message handling by application */
                pstcCanFDContext->canFDInterruptHandling.canFDRxInterruptFunction(false,
u8MessageBufferNumber, &stcCanFDmsg);
            }
        }
    }
}
```

Code Listing 8 は、Rx バッファアドレス計算のプログラム例を示します。

Code Listing 8 Rx バッファアドレス計算の例

```
static uint32_t* Cy_CANFD_CalcRxBufAdrs(cy_pstc_canfd_type_t pstcCanFD, uint8_t u8MsgBuf)
{
    uint32_t* pu32Adrs;

    if (u8MsgBuf > 63)
    {
        /* Set 0 to the return value if the index is invalid */
        pu32Adrs = NULL;
    }
    else
    {
        /*  Calculate Rx Buffer address  */
        /* Set the message buffer address to the return value if the index is available */
        pu32Adrs =  (uint32_t*)(((uint32_t)pstcCanFD & 0xFFFF0000ul) +
(uint32_t)CY_CANFD_MSGRAM_START);
        pu32Adrs += pstcCanFD->M_TTCAN.unRXBC.stcField.u14RBSA;
        pu32Adrs += u8MsgBuf * (2 + dataBufferSizeInWord[pstcCanFD-
>M_TTCAN.unRXESC.stcField.u3RBDS]);
    }
    return pu32Adrs;
}
```

Code Listing 9 は、Rx バッファからのメッセージ抽出のプログラム例を示します。

Code Listing 9 Rx バッファからのメッセージ抽出の例

```
/*** Internal function to extract received message from Rx Buffer ***/
void Cy_CANFD_ExtractMsgFromRXBuffer(cy_stc_canfd_rx_buffer_t *pstcRxBufferAddr,
cy_stc_canfd_msg_t *pstcCanFDmsg)
{
    uint16_t     u16Count = 0;
    uint16_t     u16DlcTemp = 0;

    if(0 == pstcRxBufferAddr)
    {
        return;
    }
    /* Save received data */
/* XTD : Extended Identifier */
   /*  (8) Read extended identifier flag (XTD)  */
pstcCanFDmsg->idConfig.extended = pstcRxBufferAddr->r0_f.xtd;

/* ID : RxID */
   /*  (9) Read the identifier: 11-bit or 29-bit depending on the XTD flag  */
if (pstcCanFDmsg->idConfig.extended == false)
{
pstcCanFDmsg->idConfig.identifier = pstcRxBufferAddr->r0_f.id >> 18;
}
else
{
pstcCanFDmsg->idConfig.identifier = pstcRxBufferAddr->r0_f.id;
}

/* FDF : Extended Data Length */
   /*  (10) Check Frame type  */
pstcCanFDmsg->canFDFormat = pstcRxBufferAddr->r1_f.fdf;

/* DLC : Data Length Code */
   /*  (11) Read the Data Length Code (DLC)  */
pstcCanFDmsg->dataConfig.dataLengthCode = pstcRxBufferAddr->r1_f.dlc;

/* Copy 0-64 byte of data area */
if (pstcCanFDmsg->dataConfig.dataLengthCode < 8)
{
u16DlcTemp = 0;
}
else
{
   /*  (12) Convert DLC into number of data words and read the data  */
u16DlcTemp = pstcCanFDmsg->dataConfig.dataLengthCode - 8;
}

for (u16Count = 0; u16Count < iDlcInWord[u16DlcTemp]; u16Count++)
{
pstcCanFDmsg->dataConfig.data[u16Count] = pstcRxBufferAddr->data_area_f[u16Count];
}
```

```
}
```

4.4.2 Rx FIFO 0/1 のメッセージ受信

受信メッセージが受入れフィルタリングを経由してメッセージ RAM 領域の Rx FIFO 0/1 に格納されると、Rx FIFO 割込みがイネーブルの場合、割込みが発生します。受信メッセージは、Rx FIFO Put Index によって示されるバッファ位置の Rx FIFO に格納され、割込みレジスタ (IR.RF0N/RF1N) の対応するビットがセットされます。FIFO 内のメッセージは、Rx FIFO Get Index が指し示す位置から常に読み出されます。図 8 は 8 つの FIFO 要素を持つ FIFO の例を示します。

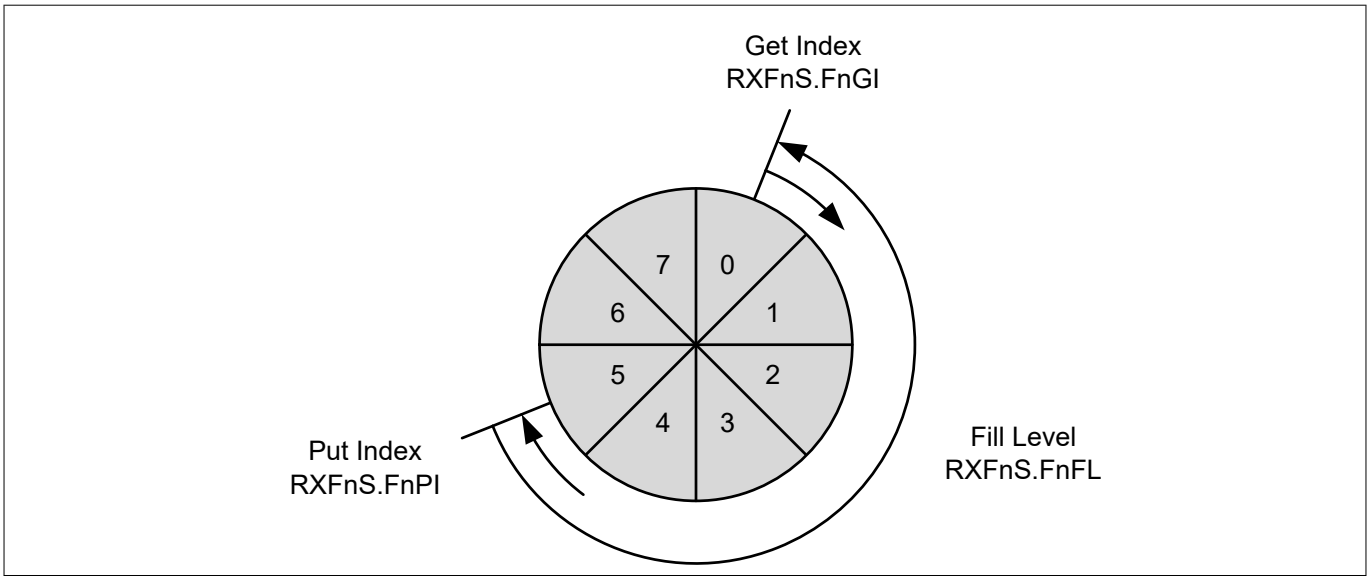

図 8 8 つの要素を持つ Rx FIFO の例

従来の Rx FIFO メッセージ処理方法は、以下の 3 つのステップを含みます。

1. Get Index 位置でのバッファの絶対アドレスを計算する
2. 受信したメッセージ情報を読み出す
3. Get Index の位置でメッセージを確認する

この方法はソフトウェアオーバヘッドの欠点があります。このオーバヘッドを排除するため、TRAVEO™ T2G は Rx FIFO の上にハードウェアロジックを実装しています。Rx FIFO のトップポインタロジックは、Get Index の位置からメッセージの内容を読み出すための単一ソースレジスタ (RXFTOPn_DATA) を提供し、絶対アドレス計算の必要性を排除します。また、メッセージのすべてのワードが RXFTOPn_DATA レジスタを介して読み出されると、トップポインタロジックが Get Index の位置でメッセージを確認する処理を行います。

たとえば、Rx FIFO エレメントサイズが 18 ワードに設定されている場合、RXFTOPn_DATA レジスタは完全なメッセージを読み出すために 18 回読み出す必要があります。18 回目の読出し後、Get Index のメッセージが自動的に確認されます。

図 9 は、Rx FIFO と Rx 割込みを使用するメッセージ受信フローの例を示します。この例では、Rx FIFO New Message 割込みのみを使用します。

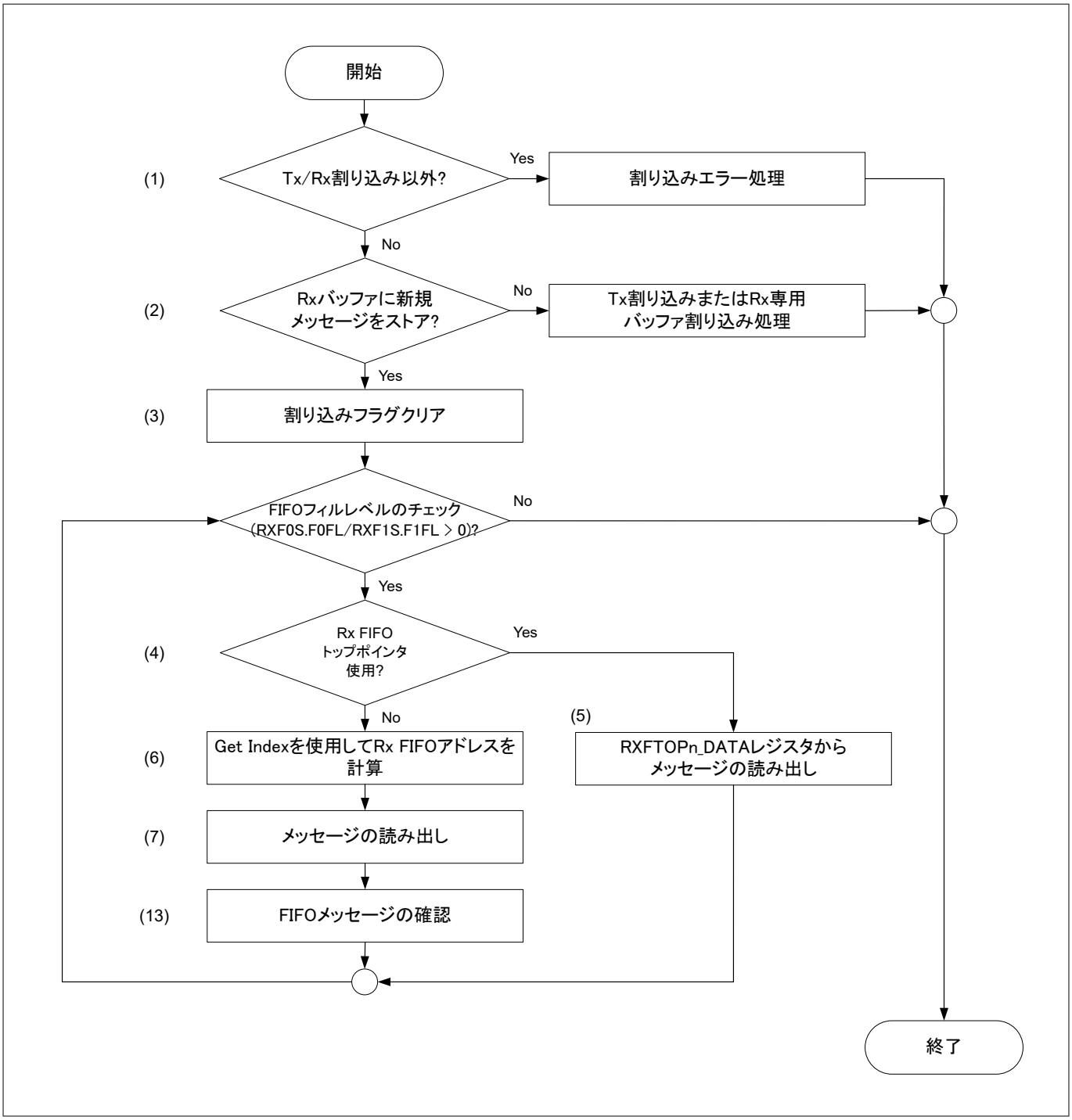

図 9　　Rx FIFO でのメッセージ受信フローの例

4.4.2.1　ユースケース

ここでは、4.2.1 ユースケースで説明したユースケースを使用した Rx FIFO でのメッセージ受信の例について説明します。

4.4.2.2　構成

ここでは、CAN FD コントローラの構成で説明した構成を使用した Rx FIFO でのメッセージ受信の例について説明します。

4.4.2.3 Rx FIFO でのメッセージ受信のプログラム例

Code Listing 10 は、Rx FIFO でのメッセージ受信のサンプルプログラムを示します。このプログラムは、Rx FIFO 0 と Rx 割込みを使用します。

Code Listing 10 Rx FIFO でのメッセージ受信の例

```
void Cy_CANFD_IrqHandler( cy_pstc_canfd_type_t pstcCanFD )
{
    uint32_t*                       pu32Adrs = 0;
    uint8_t                         u8MessageBufferNumber ;
    cy_stc_canfd_msg_t              stcCanFDmsg;
    volatile stc_CAN_CH_M_TTCAN_t*  pstcCanFDChMTTCAN;
    cy_stc_canfd_context_t*         pstcCanFDContext;
    uint8_t                         u8BufferSizeTemp = 0;
    uint32_t                        au32RxBuf[18];

    /* Get pointer to internal data structure */
    pstcCanFDContext = Cy_CANFD_GetContext(pstcCanFD);

    /* Get the pointer to M_TTCAN of the CAN FD channel */
    pstcCanFDChMTTCAN = &pstcCanFD->M_TTCAN;

    /* Other than Tx/Rx interrupt occurred */
    /* (1) Check for interrupts other than Tx/Rx */
    if (pstcCanFDChMTTCAN->unIR.u32Register & 0x3ff7E0EE)
    {
        Cy_CANFD_ErrorHandling(pstcCanFD);
    }
    /* (2) Check if New Message stored in Rx FIFO0 */
    else if(pstcCanFDChMTTCAN->unIR.stcField.u1RF0N == 1)   // New message stored into RxFIFO 0
    {
        /* (3) Clear Interrupt flag */
        pstcCanFDChMTTCAN->unIR.stcField.u1RF0N = 1;        // Clear the new message interrupt
flag
while(pstcCanFDChMTTCAN->unRXF0S.stcField.u7F0FL > 0)
{
            /* (4) Check if Rx FIFO 0 Top pointer logic is used */
            if(pstcCanFD->unRXFTOP_CTL.stcField.u1F0TPE == 1)   // RxFifo Top pointer logic is
used
            {
                u8BufferSizeTemp = 2 + dataBufferSizeInWord[pstcCanFD-
>M_TTCAN.unRXESC.stcField.u3F0DS];

                // Now read the RX FIFO Top Data register to copy the content of received
message
                for(uint8_t u8LoopVar = 0u; u8LoopVar < u8BufferSizeTemp; u8LoopVar++)
                {
                /* (5) Read the message content directly from RXFTOP0_DATA register */
                    au32RxBuf[u8LoopVar] = pstcCanFD->unRXFTOP0_DATA.u32Register;
                }

                /* CAN-FD message received, check if there is a callback function */
                /* Call callback function if it was set previously. */
                if (pstcCanFDContext-
>canFDInterruptHandling.canFDRxWithTopPtrInterruptFunction != NULL)
                {
                /* Message handling by application */
```

```
                    pstcCanFDContext-
>canFDInterruptHandling.canFDRxWithTopPtrInterruptFunction(CY_CANFD_RX_FIFO0, u8BufferSizeTemp,
&au32RxBuf[0]);
                }
            }
            else        // RxFifo Top pointer logic is not used
            {
                un_CAN_CH_RXF0S_t unRXF0S;
               /* (6) When Rx FIFO 0 Top pointer logic is not used, get the Rx FIFO address
holding the message at FIFO 0 get index position with a calculation function (See Code Listing
11).*/
                unRXF0S.u32Register = pstcCanFDChMTTCAN->unRXF0S.u32Register;

                pu32Adrs = Cy_CANFD_CalcRxFifoAdrs(pstcCanFD, CY_CANFD_RX_FIFO0,
unRXF0S.stcField.u6F0GI);

                if(pu32Adrs)
                {
                    // Extract the received message from Buffer
                    /* (7) Read the message at get index position (See Code Listing 9) */
                    Cy_CANFD_ExtractMsgFromRXBuffer((cy_stc_canfd_rx_buffer_t *) pu32Adrs,
&stcCanFDmsg);

                    // Acknowledge the FIFO message
                   /* (13) Acknowledge the FIFO 0 message at get index position */
                    pstcCanFDChMTTCAN->unRXF0A.stcField.u6F0AI = unRXF0S.stcField.u6F0GI;

                    /* CAN-FD message received, check if there is a callback function */
                    /* Call callback function if it was set previously. */
                    if (pstcCanFDContext->canFDInterruptHandling.canFDRxInterruptFunction !=
NULL)
                    {
                       /* Message handling by application */
                        pstcCanFDContext->canFDInterruptHandling.canFDRxInterruptFunction(true,
CY_CANFD_RX_FIFO0, &stcCanFDmsg);
                    }
                }
            }
        }
    }
}
```

Code Listing 11 は、Rx FIFO アドレス計算のプログラム例を示します。

Code Listing 11 Rx FIFO アドレス計算の例

```
static uint32_t* Cy_CANFD_CalcRxFifoAdrs(cy_pstc_canfd_type_t pstcCanFD, uint8_t u8FifoNumber,
uint32_t u32GetIndex)
{
    uint32_t* pu32Adrs;

    if(u8FifoNumber > 1)
    {
        /* Set 0 to the return value if the FIFO number is invalid */
        pu32Adrs = NULL;
    }
    else
    {
        /*   Calculate the Rx FIFO address  */
        /* Set the message buffer address to the return value if the index is available */
        pu32Adrs = (uint32_t*)(((uint32_t)pstcCanFD & 0xFFFF0000ul) +
(uint32_t)CY_CANFD_MSGRAM_START);
        pu32Adrs += (u8FifoNumber == 0) ? pstcCanFD->M_TTCAN.unRXF0C.stcField.u14F0SA :
pstcCanFD->M_TTCAN.unRXF1C.stcField.u14F1SA;
        pu32Adrs += u32GetIndex * (2 + dataBufferSizeInWord[(u8FifoNumber == 0) ? pstcCanFD-
>M_TTCAN.unRXESC.stcField.u3F0DS : pstcCanFD->M_TTCAN.unRXESC.stcField.u3F1DS]);
    }

    return pu32Adrs;
}
```

5 用語集

表 5 用語集

用語	説明
ACK	Acknowledgement (応答)
BRS	Bit Rate Switch (ビットレートスイッチ)
CAN	Controller Area Network
CAN FD	Controller Area Network with Flexible Data rate
CANH	CAN Network Line High
CANL	CAN Network Line Low
CRC	Cyclic Redundancy Check (巡回冗長検査)
DLC	Data Length Code (データ長コード)
ECC	Error Correction Code (誤り訂正符号)
ECU	Electronic Control Unit (電子制御ユニット)
EOF	End of Frame (エンドオブフレーム)
ESI	Error State Indicator (エラーステートインジケータ)
FDF	FD Format indicator (FD フォーマットインジケータ)
FIFO	First in First out
ID	Identifier (識別子)
IDE	Identifier Extension
MMIO	Memory Mapped I/O (メモリマップド I/O)
RAM	Random Access Memory
RTR	Remote Transmission Request
SOF	Start of Frame
SPI	Serial Peripheral Interface

6 関連ドキュメント

以下は TRAVEO™ T2G ファミリのデータシートおよびテクニカルリファレンスマニュアルです。これらのドキュメントの入手についてはテクニカルサポートに連絡してください。

- デバイスデータシート
  - CYT2B6 datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
  - CYT2B7 datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
  - CYT2B9 datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
  - CYT2BL datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
  - CYT3BB/4BB datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
  - CYT4BF datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
  - CYT6BJ datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family (Doc No. 002-33466)
  - CYT3DL datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
  - CYT4DN datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
  - CYT4EN datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family (Doc No. 002-30842)
  - CYT2CL datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
- Body controller entry ファミリ
  - TRAVEO™ T2G automotive body controller entry family architecture technical reference manual (TRM)
  - TRAVEO™ T2G automotive body controller entry registers technical reference manual (TRM) for CYT2B7
  - TRAVEO™ T2G automotive body controller entry registers technical reference manual (TRM) for CYT2B9
  - TRAVEO™ T2G automotive body controller entry registers technical reference manual (TRM) for CYT2BL (Doc No. 002-29852)
- Body controller high ファミリ
  - TRAVEO™ T2G automotive body controller high family architecture technical reference manual (TRM)
  - TRAVEO™ T2G automotive body controller high registers technical reference manual (TRM) for CYT3BB/4BB
  - TRAVEO™ T2G automotive body controller high registers technical reference manual (TRM) for CYT4BF
  - TRAVEO™ T2G automotive body controller high registers technical reference manual (TRM) for CYT6BJ (Doc No. 002-36068)
- Cluster 2D ファミリ
  - TRAVEO™ T2G automotive cluster 2D architecture technical reference manual (TRM)
  - TRAVEO™ T2G automotive cluster 2D registers technical reference manual (TRM) for CYT3DL
  - TRAVEO™ T2G automotive cluster 2D registers technical reference manual (TRM) for CYT4DN
  - TRAVEO™ T2G automotive cluster 2D registers technical reference manual (TRM) for CYT4EN (Doc No. 002-35181)
- Cluster Entry ファミリ
  - TRAVEO™ T2G automotive cluster entry architecture technical reference manual (TRM)
  - TRAVEO™ T2G automotive cluster entry registers technical reference manual (TRM) for CYT2CL
- アプリケーションノート
  - AN219755 – TRAVEO™ T2G 車載マイクロコントローラーでの SAR ADC の使用
  - AN219842 – TRAVEO™ T2G の割込みの使用方法
  - AN225401 – TRAVEO™ T2G ファミリのシリアル通信ブロック (SCB) 使用方法
  - AN220224–TRAVEO™ T2G ファミリのタイマ, カウンタ, および PWM (TCPWM) の設定方法

7 参考資料

インフィニオンは、さまざまな周辺機器にアクセスするためのサンプルソフトウェアとして、スタートアップを含む Sample Driver Library (SDL) を提供しています。SDL は、公式の AUTOSAR 製品でカバーされていないドライバの顧客へのリファレンスとしても機能します。SDL は自動車規格に適合していないため、製造目的で使用できません。このアプリケーションノートのプログラムコードは SDL の一部です。SDL の入手については、テクニカルサポートに連絡してください。

改訂履歴

版数	発行日	変更内容
**	2018-12-06	これは英語版 002-20278 Rev. **を翻訳した日本語版 002-25671 Rev. **です。英語版の改訂内容: New application note.
*A	2019-04-25	これは英語版 002-20278 Rev. *A を翻訳した日本語版 002-25671 Rev. *A です。英語版の改訂内容: Updated Associated Part Family as “TRAVEO™ T2G family CYT2B/CYT4B Series”. Added target part numbers “CYT4B Series” related information in all instances across the document. [Section 1, 2.2] - Change to bps from bits/s. [Section 2.2] - Added the explanation of DATA FRAME. - Updated the figure 2 according to the specifications. [Section 4.2] - Updated the flow of Figure 5. - Added the CCCR.INIT setting to (1). - Updated the contents for (2), (3), (7). [Section 4.2.1] - Added new [Section 4.3] - Updated the flow of Figure 6. [Section 4.3.1] - Updated the example code.
英語版*B	-	この版は英語版のみです。英語版の改訂内容: Updated Associated Part Family as “TRAVEO™ T2G family CYT2B/CYT4B/CYT4D Series”. Added target part numbers “CYT4D Series” related information in all instances across the document. [Section 3] - Updated the link of device datasheet [Section 4] - Updated the link of Architecture TRM - Added the link of device datasheet [Section 6] - Changed to new format and added the CYT4D series documents

改訂履歴

版数	発行日	変更内容
*B	2020-09-23	これは英語版 002-20278 Rev. *C を翻訳した日本語版 002-25671 Rev. *B です。英語版の改訂内容: Updated Associated Part Family as “TRAVEO™ T2G family CYT2/CYT3/CYT4 Series”. Changed target part numbers from “CYT2B/CYT4B/CYT4D Series” to “CYT2/CYT4 Series” in all instances across the document. Added target part numbers “CYT3 Series” in all instances across the document. [Section 3] - Updated the Figure 4 [Section 4] - Updated the flow and code to align [Section 6] - Updated the Related Documents [Section 7] - Added the information of the Sample Driver Library
*C	2021-08-04	テンプレートの変更を実施。これは英語版 002-20278 Rev. *D を翻訳した日本語版 Rev. *C です。英語版の改訂内容: Updated to Infineon template.
*D	2024-05-30	これは英語版 002-20278 Rev. *E を翻訳した日本語版 Rev. *D です。英語版の改訂内容: Updated the document title. Template update; no content update.
*E	2025-01-16	これは英語版 002-20278 Rev. *F を翻訳した日本語版 002-25671 Rev. *E です。英語版の改訂内容: Updated 関連ドキュメント and added CYT6 series

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2025-01-16

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-pvu1681443277092

重要事項

本手引書に記載された情報は、本製品の使用に関する手引きとして提供されるものであり、いかなる場合も、本製品における特定の機能性能や品質について保証するものではありません。本製品の使用の前に、当該手引書の受領者は実際の使用環境の下であらゆる本製品の機能及びその他本手引書に記された一切の技術的情報について確認する義務が有ります。インフィニオンテクノロジーズはここに当該手引書内で記される情報につき、第三者の知的所有権の不侵害の保証を含むがこれに限らず、あらゆる種類の一切の保証および責任を否定いたします。

本文書に含まれるデータは、技術的訓練を受けた従業員のみを対象としています。本製品の対象用途への適合性、およびこれら用途に関連して本文書に記載された製品情報の完全性についての評価は、お客様の技術部門の責任にて実施してください。

警告事項

技術的要件に伴い、製品には危険物質が含まれる可能性があります。当該種別の詳細については、インフィニオンの最寄りの営業所までお問い合わせください。

インフィニオンの正式代表者が署名した書面を通じ、インフィニオンによる明示の承認が存在する場合を除き、インフィニオンの製品は、当該製品の障害またはその使用に関する一切の結果が、合理的に人的傷害を招く恐れのある一切の用途に使用することはできないこと予めご了承ください。