AN213891

# Traveo™ ファミリ CAN FD の使用方法

著者: Shinichi Takko

関連製品ファミリ: Traveo Family

関連ドキュメント: 完全なリストについては、ここをクリックしてください。

本アプリケーションノートは Traveo ファミリに搭載されている CAN with Flexible Data rate (CAN FD)の使用方法について記載しています。

## 目次



## 1 はじめに

本アプリケーションノートは Cypress Traveo ファミリのユーザを対象に Controller Area Network with Flexible Data rate (CAN FD)の使用方法について説明しています。

CAN は自動車向けの安全で経済的な通信規格であり、長年にわたりデータ通信の標準規格であると考えられています。

CAN FD は CAN を拡張させたものであり、CAN は現在'Classical CAN'と呼ばれています。CAN FD は CAN の制限の 1Mbps を超えるビットレートで最大 64byte のデータフレームを送信できます。データセグメントのバスの最大速度は、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制限されます。

CAN FD コントローラは ISO 11898-1:2015 と ISO16845:2016 に準拠しています。

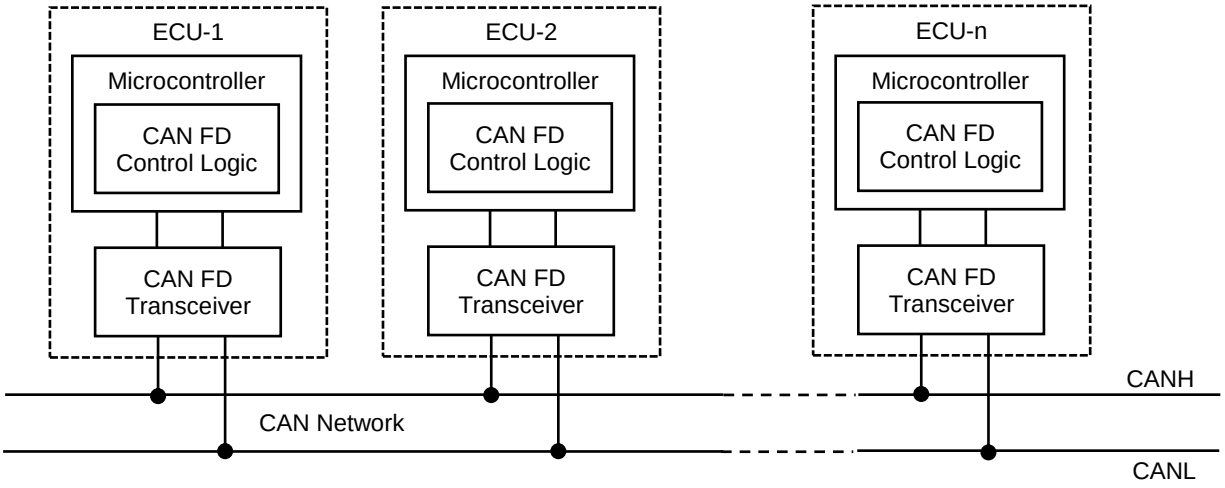
## 2 CAN FD ネットワーク

ここでは CAN FD 通信の動作について説明します。CAN FD ネットワークの例を Figure 1 に示します。

CAN FD ネットワークでは、2 つの通信線（CANH, CANL）を使用してノイズ耐性を持たせています。ネットワークに複数の ECU を接続し、ECU 間でデータを交換します。

受信時は、差動バス電圧を CAN FD トランシーバによってデジタル信号に変換し、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。送信時は、CAN FD 制御ロジックから CAN FD ネットワークの CANH および CANL に差動信号を送信する CAN FD トランシーバにデータが送信されます。

Figure 1. CAN FD ネットワーク




ECU - Electronic Control Unit

CANH - CAN network line High

CANL - CAN network line Low

# 3 CAN FD メッセージ

CAN FD 通信は従来の CAN よりも帯域幅を拡大しています。CAN FD フレームフォーマットを Figure 2 に示します。CAN および CAN FD のメッセージフォーマットは、アービトレーションフィールドと CRC フィールドの後で等しくなります。アービトレーションフェーズより高速送信可能なデータフェーズで差分が生じています。

CAN のデータ長は 8 byte ですが、CAN FD では最大 64 byte に拡張されています。データの通信速度は、CAN の制限値である 1 Mbps を超えることができます。現在 5 Mbps をサポートするトランシーバがありますが、8 Mbps をサポートするアナウンスもあります。

## 3.1 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には、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 フィールドがあります。

アービトレーションフィールドにはメッセージ ID が含まれており、同時に送信しようとしている他のノードからのメッセージ間で優先順位を決定します。

コントロールフィールドには、FD Format（FDF），Bit Rate Switch（BRS），および Error State Indicator（ESI）の 3bit が追加されています。

FDF でフレームタイプ（CAN / CAN FD）を識別します。CAN FD フレームの場合はレセッシブ、CAN フレームの場合はドミナントです。BRS がレセッシブの場合、データフィールドのビットレートは高速側に切り替わり、ドミナントの場合はアービトレーションバス速度を維持します。ESI はエラー状態の識別に使用されます。

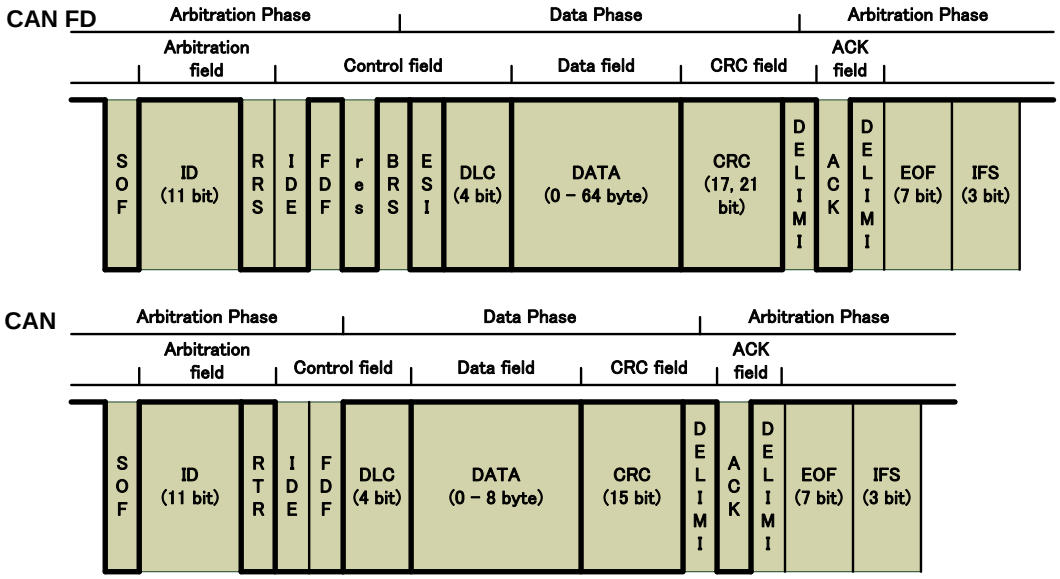
Data Length Code（DLC）は送信されるデータのバイト数を示します。設定可能範囲は CAN の 8 byte から CAN FD では 64 byte に拡張されています。

データフィールドには DLC で設定されたデータ長のメッセージデータが格納されます。

CRC フィールドは CRC シーケンスと CRC デリミタで構成されています。CRC シーケンスは、データ長が 0～16 byte のときは 17 bit、17～64 byte のときは 21 bit です。受信側は受信したメッセージを分析し、CRC と比較して受信メッセージの正常性を確認することができます。

ACK フィールドは ACK スロットと ACK デリミタで構成されています。メッセージ受信が成功した場合、ドミナントが受信したノードから送信されます。

Figure 2. CAN FD Frame

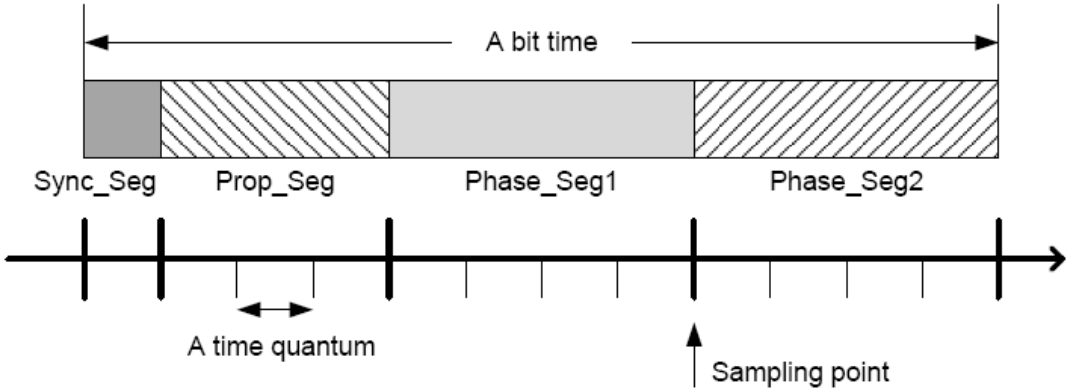


## 3.2 ビットタイミング

CAN FD ではノーマ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存在します。ノーマルビットタイムはアービトレーションフェーズで使用されます。データビットタイムはノーマルビットタイムと比べて、等しいかもしくは短い値で、データフェーズで使用されます

ビットタイムの基本構成は、ノーマル/データビットタイムで共通です。ビットタイムは、同期セグメント(Sync_Seg), 伝搬時間セグメント(Prop_Seg), フェーズバッファセグメント 1(Phase_Seg1), およびフェーズバッファセグメント 2(Phase_Seg2)の 4 つのセグメントに分けられます(Figure 3)。バスのレベルを読み込み、ビットの値を決定するサンプリングポイントは Phase_Seg1 の最後に位置します。

Figure 3. ビットタイム構成




各セグメントは、CAN クロックとプリスケーラで決定される、プログラマブルなタイムクォンタム(TQ)で構成されています。これらを定義するために使用される値は、ノーマルビットタイムとデータビットタイムで異なり、それぞれ Nominal Bit Timing & Prescaler Register (NBTP)と Data Bit Timing & Prescaler Register (DBTP)で設定します。

| パラメータ | 詳細 |
|---|---|
| Time quantum tq (nominal) and tqd (data) | タイムクォンタム。基本単位時間（たとえば CAN クロック周期）にそれぞれのプリスケーラを乗算することによって算出されます。<br>タイムクォンタムは CAN FD コントローラによって次のように設定されます。<br>nominal : tq = (NBTP.NBRP[8:0] +1) × CAN clock period<br>data : tqd = (DBTP.DBRP[4:0] + 1) × CAN clock period |
| Sync_Seg | Sync_Seg は CAN の仕様で定義されているように 1tq に固定されています。(CAN FD コントローラに組み込まれています)<br>nominal : 1 tq<br>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補償するために使用されます。CAN FD コントローラは Prop_Seg と Phase_Seg1 を 1 つのパラメータで設定します。<br>nominal : Prop_Seg + Phase_Seg1 = NBTP.NTSEG1[7:0] + 1<br>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フェーズエラー補償のために使用されます。再同期ジャンプ幅によって長くすることができます。Prop_Seg と Phase_Seg1 の合計は、CAN FD コントローラによって次のように設定されます。<br>nominal : NBTP.NTSEG1[7:0] + 1<br>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フェーズエラー補償のために使用されます。再同期ジャンプ幅によって短くすることができます。Phase_Seg2 は CAN FD コントローラによって次のように設定されます。<br>nominal : NBTP.NTSEG2[6:0] + 1<br>data : DBTP.DTSEG2[3:0] + 1 |
| SJW | 再同期ジャンプ幅。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 の長さを調整するために使用されます。SJW は Phase_Seg1 もしくは Phase_Seg2 より長くはなりません。<br>SJW は CAN FD コントローラによって次のように設定されます。<br>nominal : NBTP.NSJW[6:0] + 1<br>data : DBTP.DSJW[3:0] + 1 |

ノーマル/データビットタイムについて、これらの関係を以下の式で示します。

ノーマルビットタイム

$= [Sync\_Seg + Prop\_Seg + Phase\_Seg1 + Phase\_Seg2] \times tq$

$= [1 + (NBTP.NTSEG1[7:0] + 1) + (NBTP.NTSEG2[6:0] + 1)] \times [(NBTP.NBRP[8:0] + 1) \times CAN\ clock\ period]$

例 (500 kbps)

$= [1 + (13 + 1) + (4 + 1)] \times [(3 + 1) \times (1/40000000)] = 0.000002\ (500\ kbps)$

データビットタイム

$= [1 + (DBTP.DTSEG1[4:0] + 1) + (DBTP.DTSEG2[3:0] + 1)] \times [(DBTP.DBRP[4:0] + 1) \times CAN\ clock\ period]$

例 (5 Mbps)

$= [1 + (3 + 1) + (2 + 1)] \times [(0 + 1) \times (1/40000000)] = 0.0000002\ (5\ Mbps)$

例 (2 Mbps)

$= [1 + (10 + 1) + (7 + 1)] \times [(0 + 1) \times (1/40000000)] = 0.0000005\ (2\ Mbps)$

# 4 CAN FD セッティング

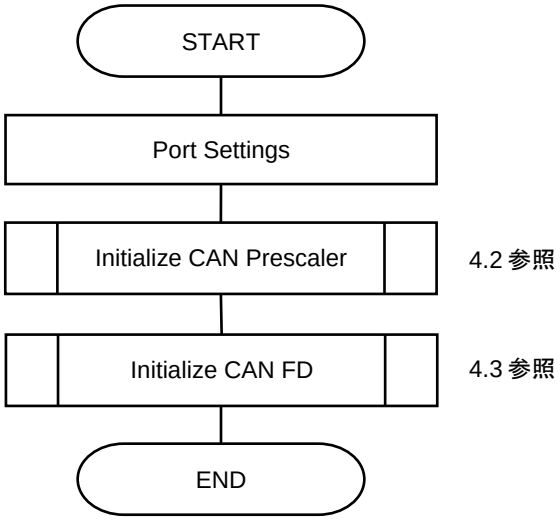
このセクションでは Traveo ファミリの CAN FD 設定について説明します。レジスタの詳細については、対象製品のハードウェアマニュアルの CAN FD の章を参照してください。

## 4.1 CAN FD セットアップ

Figure 4 は CAN FD セットアップフローの例です。まず、CAN FD 通信に使用する I/O ポートを有効にします。その後、CAN FD コントローラユニットに供給されるクロックを設定し、CAN FD コントローラの設定を行います。

次のフローのサンプルコードを Section 4.1.1 に示します。

Figure 4. CAN FD セットアップの例




### 4.1.1 CAN FD セットアップのサンプルコード

CAN FD セットアップのサンプルコードを Code 1 に示します。

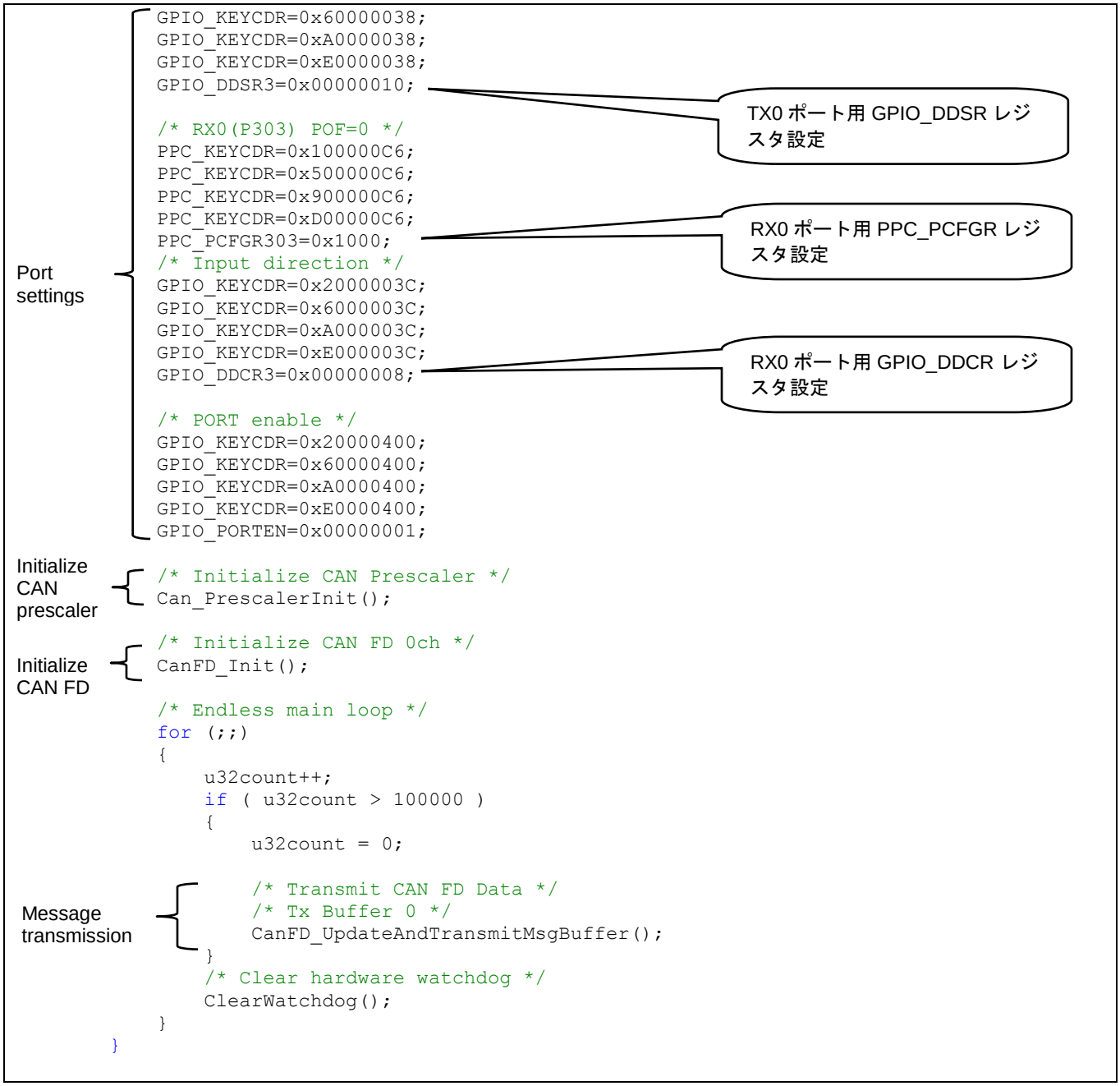
Code 1. CAN FD セットアップ

```
int main(void)
{
    uint32_t    u32count = 0;

    /* Finalize initialization to default settings. */
    /* (this will do IRQ and NMI initialization and global IRQ/NMI enable) */
    Start_Init();

    /* CAN FD transceiver device Initialize */
    CanFD_Device_Init();

    /* PORT - port pin configuration */
    /* TX0(P304) POF=2 */
    PPC_KEYCDR=0x100000C8;
    PPC_KEYCDR=0x500000C8;
    PPC_KEYCDR=0x900000C8;
    PPC_KEYCDR=0xD00000C8;
    PPC_PCFGR304=0x0002;
    /* Output direction */
    GPIO_KEYCDR=0x20000038;
```

Port settings

Keycode レジスタ設定

TX0 ポート用 PPC_PCFGR レジスタ設定



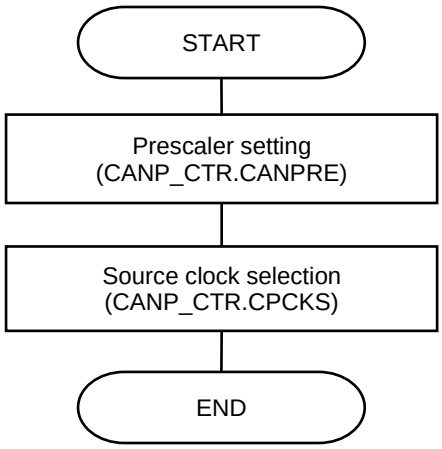

```
    GPIO_KEYCDR=0x60000038;
    GPIO_KEYCDR=0xA0000038;
    GPIO_KEYCDR=0xE0000038;
    GPIO_DDSR3=0x00000010;

    /* RX0(P303) POF=0 */
    PPC_KEYCDR=0x100000C6;
    PPC_KEYCDR=0x500000C6;
    PPC_KEYCDR=0x900000C6;
    PPC_KEYCDR=0xD00000C6;
    PPC_PCFGR303=0x1000;
    /* Input direction */
    GPIO_KEYCDR=0x2000003C;
    GPIO_KEYCDR=0x6000003C;
    GPIO_KEYCDR=0xA000003C;
    GPIO_KEYCDR=0xE000003C;
    GPIO_DDCR3=0x00000008;

    /* PORT enable */
    GPIO_KEYCDR=0x20000400;
    GPIO_KEYCDR=0x60000400;
    GPIO_KEYCDR=0xA0000400;
    GPIO_KEYCDR=0xE0000400;
    GPIO_PORTEN=0x00000001;

    /* Initialize CAN Prescaler */
    Can_PrescalerInit();

    /* Initialize CAN FD 0ch */
    CanFD_Init();

    /* Endless main loop */
    for (;;)
    {
        u32count++;
        if ( u32count > 100000 )
        {
            u32count = 0;

            /* Transmit CAN FD Data */
            /* Tx Buffer 0 */
            CanFD_UpdateAndTransmitMsgBuffer();
        }
        /* Clear hardware watchdog */
        ClearWatchdog();
    }
}
```

## 4.2 CAN プリスケーラ初期化

Figure 5 は CAN プリスケーラの初期化フローの例です。プリスケーラ設定では、CAN FD コントローラに供給されるクロックの分周比を設定します(CANP_CTR.CANPRE)。ソースクロックには PLL クロックとメインクロックのどちらかを選択できます(CANP_CTR.CPCKS)。

次のフローのサンプルコードを 4.2.1 に示します。

Figure 5. CAN プリスケーラ初期化フローの例




### 4.2.1 CAN プリスケーラ初期化のサンプルコード

CAN プリスケーラ設定のサンプルコードを Code 2 に示します。

Code 2. CAN プリスケーラ設定

```
void Can_PrescalerInit(void)
{
    /* CAN prescaler division setting : 11 (Divided by 12) */
    CANP_CTR_CANPRE = 11;  /* CAN clock = 40MHz (PLL clock = 480MHz)
    while( CANP_STR_BUSY == 1 )
    {
    }
    /* Select the source clock of CAN Prescaler */
    CANP_CTR_CPCKS = 0;    /* Select the PLL clock */
}
```

プリスケーラ設定

ソースクロック選択

## 4.3 CAN FD 初期化

Figure 6 は CAN FD 初期化フローの例です。このセクションでは初期化の一般的な流れについて説明します。

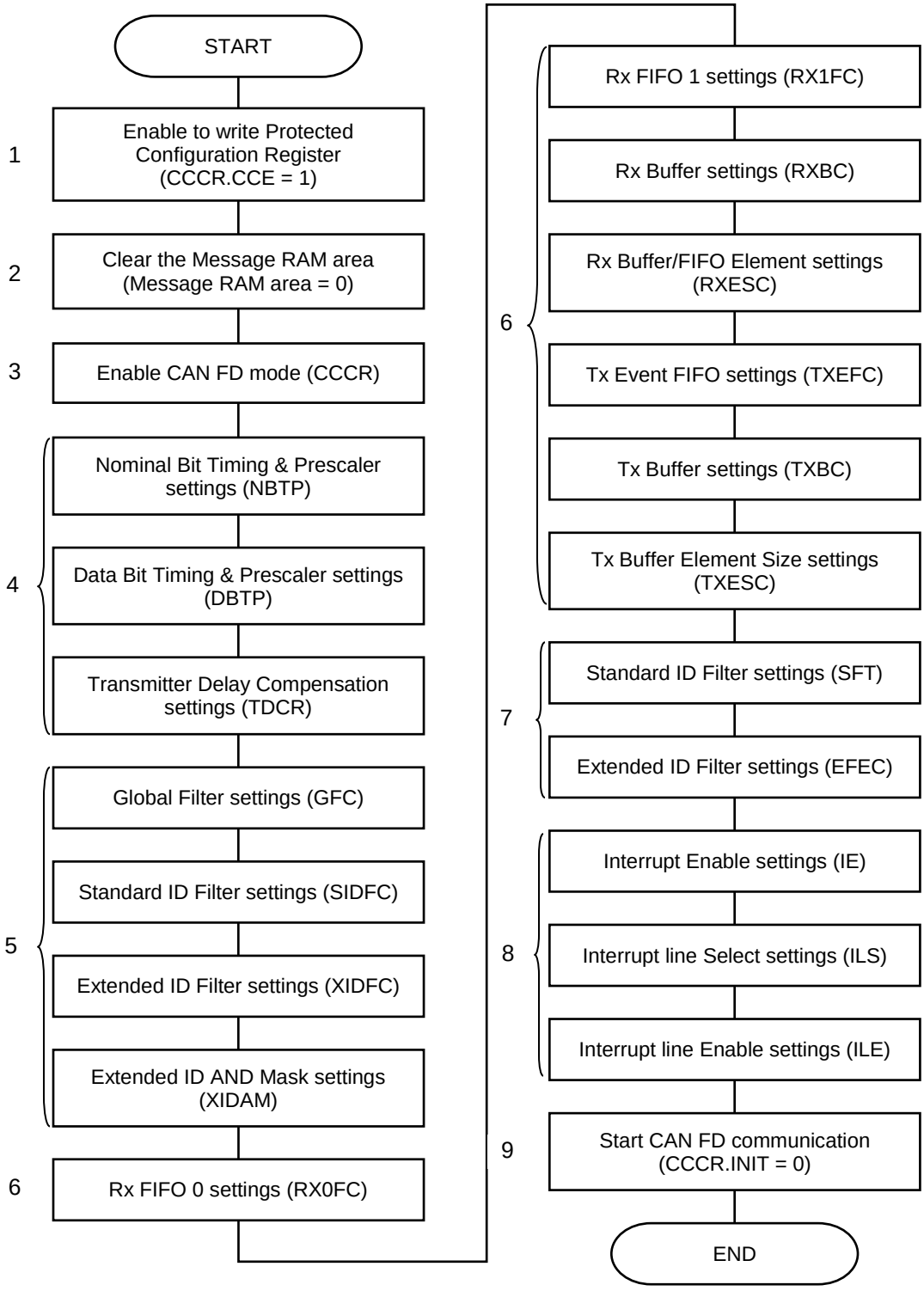
1. Configuration Change Enable register (CCCR.CCE)を有効にすることによって、書込み保護の CAN FD コンフィグレーションレジスタに書込み可能にします。
2. Rx/Tx メッセージやフィルタ設定のエリアであるメッセージ RAM をクリアします。
3. CC Control Register (CCCR)の Bit Rate Switch (CCCR.BRSE)と CAN FD mode (CCCR.FDOE)を設定します。
4. ビットタイミングは、アービトレーションフェーズで使用される Nominal Bit Timing & Prescaler Register (NBTP)と、ビットレートスイッチが有効な場合にデータフェーズで使用される Data Bit Timing & Prescaler Register (DBTP)で設定します。
5. メッセージフィルタについて、Global Filter Configuration Register (GFC)は、どのフィルタとも一致しないメッセージ ID を持つ受信フレームの扱いを決定します。メッセージフィルタ数とメッセージ RAM 内の開始アドレスオフセットは、Standard ID Filter Configuration register (SIDFC)と Extended ID Filter Configuration register (XIDFC)で設定します。
6. RX/Tx メッセージについて、Rx FIFO のエレメントサイズとメッセージ RAM の開始アドレスオフセットを Rx FIFO 0 Configuration (RXF0C)と Rx FIFO 1 Configuration (RXF1C)に設定します。

   Rx バッファは Rx Buffer Configuration (RXBC)で設定します。Rx FIFO または Rx buffer に格納されるデータフィールドサイズは Rx Buffer/FIFO Element Size Configuration (RXESC)で設定します。

   メッセージ RAM の Tx バッファ数と開始アドレスオフセットは Tx Buffer Configuration (TXBC)で設定します。Tx バッファのデータフィールドサイズは Tx Buffer Element Size Configuration (TXESC)で設定します。
7. メッセージフィルタは、メッセージ RAM の開始アドレスに SIDFC/XIDFC で設定した開始アドレスオフセットを加えたアドレスに設定されます。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。
8. 各割込みは Interrupt Enable (IE)で設定します。CAN FD コントローラは 2 つの割込みラインを持っており、Interrupt Line Select (ILS)によって割込みラインが決定します。Interrupt Line Enable (ILE)で割込みラインを有効にします。
9. CAN FD の動作を開始するために、Initialization register (CCCR.INIT)を 0 に設定します。

次のフローのサンプルコードを 4.3.1 に示します。

Figure 6. CAN FD 初期化フローの例



### 4.3.1 CAN FD 初期化のサンプルコード

CAN FD 初期化設定のサンプルコードを Code 3 に示します。

Code 3. 初期化設定

```
void CanFD_Init(void)
{
    stc_id_filter_t*         pstcIdFilter;
    stc_extid_filter_t*      pstcExtIdFilter;
    uint32_t*                pulAdrs;
    uint16_t                 u16count;

    /* Shadow data to avoid RMW and speed up HW access */
    un_cpg_canfdn_sidfc_t unSIDFC = { 0 };
    un_cpg_canfdn_xidfc_t unXIDFC = { 0 };
    un_cpg_canfdn_xidam_t unXIDAM = { 0 };
    un_cpg_canfdn_rxf0c_t unRXF0C = { 0 };
    un_cpg_canfdn_rxf1c_t unRXF1C = { 0 };
    un_cpg_canfdn_rxbc_t  unRXBC  = { 0 };
    un_cpg_canfdn_txefc_t unTXEFC = { 0 };
    un_cpg_canfdn_txbc_t  unTXBC  = { 0 };
    un_cpg_canfdn_cccr_t  unCCCR  = { 0 };
    un_cpg_canfdn_nbtp_t  unNBTP  = { 0 };
    un_cpg_canfdn_dbtp_t  unDBTP  = { 0 };
    un_cpg_canfdn_tdcr_t  unTDCR  = { 0 };
    un_cpg_canfdn_gfc_t   unGFC   = { 0 };
    un_cpg_canfdn_rxesc_t unRXESC = { 0 };
    un_cpg_canfdn_txesc_t unTXESC = { 0 };
    un_cpg_canfdn_ie_t    unIE    = { 0 };
    un_cpg_canfdn_ils_t   unILS   = { 0 };
    un_cpg_canfdn_ile_t   unILE   = { 0 };

    /* Set CCCR.INIT to 1 and wait until it will be updated. */
    unCCCR.stcField.u1INIT = 1;
    CPG_CANFD0_CCCR = unCCCR.u32Register;

    while ( CPG_CANFD0_CCCR_INIT != 1 )
    {
    }

    /* Cancel protection */
    unCCCR.stcField.u1CCE = 1;
    CPG_CANFD0_CCCR = unCCCR.u32Register;

    /* Clear the message RAM area */
    pulAdrs = (uint32_t *)((uint32_t)&CPG_CANFD0 + (uint32_t)0x00008000);
    for (u16count = 0; u16count < 4096; u16count++)
    {
        *pulAdrs++ = 0;
    }

    /* Configuration of CAN bus */

    /* CCCR register */
    unCCCR.stcField.u1TXP  = 0; /* Transmit pause disabled. */
    unCCCR.stcField.u1BRSE = 1; /* Bit rate switching for transmissions enabled. */
    unCCCR.stcField.u1FDOE = 1; /* FD operation enabled. */
    unCCCR.stcField.u1TEST = 0; /* Normal operation */
    unCCCR.stcField.u1DAR  = 0; /* Automatic retransmission enabled. */
```

CAN FD 通信停止

書込み保護のレジスタへの書込み許可設定

メッセージ RAM をクリア

```
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. */
CPG_CANFD0_CCCR = unCCCR.u32Register;

/* Normal Bit Timing & Prescaler Register */
unNBTP.stcField.u9NBRP   = 3;
unNBTP.stcField.u8NTSEG1 = 13;
unNBTP.stcField.u7NTSEG2 = 4;
unNBTP.stcField.u7NSJW   = 4;
CPG_CANFD0_NBTP = unNBTP.u32Register;

/* Data Bit Timing & Prescaler */
unDBTP.stcField.u1TDC    = 1; /* Transceiver Delay Compensation enabled. */
unDBTP.stcField.u5DBRP   = 0;
unDBTP.stcField.u5DTSEG1 = 3;
unDBTP.stcField.u4DTSEG2 = 2;
unDBTP.stcField.u4DSJW   = 2;
CPG_CANFD0_DBTP = unDBTP.u32Register;

/* Transmitter Delay Compensation */
unTDCR.stcField.u7TDCO   = 4; /* Transmitter Delay Compensation Offset */
unTDCR.stcField.u7TDCF   = 0; /* Transmitter Delay Compensation Filter Window Length */
CPG_CANFD0_TDCR = unTDCR.u32Register;

/* Configuration of Message RAM */

/* Configuration of ID Filter List */
/* Configuration of Global Filter */
unGFC.stcField.u2ANFS = 2; /* Reject when unmatch id */
unGFC.stcField.u2ANFE = 2; /* Reject when unmatch id */
unGFC.stcField.u1RRFS = 1; /* Reject all remote frame */
unGFC.stcField.u1RRFE = 1; /* Reject all remote frame */
CPG_CANFD0_GFC = unGFC.u32Register;

/* Standard ID filter */
unSIDFC.stcField.u8LSS = 1; /* Number of standard Message ID filter elements = 1 */
unSIDFC.stcField.u14FLSSA = 0x00000000; /* offset(word) */
CPG_CANFD0_SIDFC = unSIDFC.u32Register;

/* Extended ID filter */
unXIDFC.stcField.u7LSE = 1; /* Number of extended Message ID filter elements = 1 */
unXIDFC.stcField.u14FLESA = 0x00000004; /* offset(word) */
CPG_CANFD0_XIDFC = unXIDFC.u32Register;

unXIDAM.stcField.u29EIDM = 0x1fffffff;   /* not filtering(initial value) */
CPG_CANFD0_XIDAM = unXIDAM.u32Register;

/* Configuration of Rx Buffer and Rx FIFO */
/* Rx FIFO 0 (not use) */
unRXF0C.stcField.u1F0OM = 0;    /* Rx FIFO 0 blocking mode */
unRXF0C.stcField.u7F0WM = 0;    /* Watermark interrupt disabled */
unRXF0C.stcField.u7F0S = 0;     /* FIFO Element Number = 0 */
unRXF0C.stcField.u14F0SA = 0x00000010; /* offset(word) */
CPG_CANFD0_RXF0C = unRXF0C.u32Register;

/* Rx FIFO 1 (not use) */
unRXF1C.stcField.u1F1OM = 0;    /* Rx FIFO 1 blocking mode */
unRXF1C.stcField.u7F1WM = 0;    /* Watermark interrupt disabled */
unRXF1C.stcField.u7F1S = 0;     /* FIFO Element Number = 0 */
unRXF1C.stcField.u14F1SA = 0x00000020; /* offset(word) */
```

CAN FD モード設定

ノーマルビットタイム、プリスケーラ設定
(500kbps)

データビットタイム、プリスケーラ設定
(5Mbps)

転送遅延補償設定

グローバルフィルタ設定

標準 ID フィルタ設定

拡張 ID フィルタ設定

拡張 ID、マスク設定

Rx FIFO 0 設定

```
CPG_CANFD0_RXF1C = unRXF1C.u32Register;

/* Rx buffer */
unRXBC.stcField.u14RBSA = 0x00000030; /* offset(word) */
CPG_CANFD0_RXBC = unRXBC.u32Register;

/* Configuration of Rx Buffer and Rx FIFO */
unRXESC.stcField.u3RBDS = 7; /* 64 byte data field. */
unRXESC.stcField.u3F1DS = 7; /* FIFO1 64 byte data field. */
unRXESC.stcField.u3F0DS = 7; /* FIFO0 64 byte data field. */
CPG_CANFD0_RXESC = unRXESC.u32Register;

/* Configuration of Tx Buffer and Tx FIFO/Queue */
/* Tx Event FIFO (not use) */
unTXEFC.stcField.u6EFWM = 0; /* Watermark interrupt disabled. */
unTXEFC.stcField.u6EFS = 0;  /* Tx Event FIFO disabled. */
unTXEFC.stcField.u14EFSA = 0x00000100; /* offset(word) */
CPG_CANFD0_TXEFC = unTXEFC.u32Register;

/* Tx buffer */
unTXBC.stcField.u1TFQM = 0;  /* Tx FIFO operation */
unTXBC.stcField.u6TFQS = 0;  /* No Tx FIFO/Queue */
unTXBC.stcField.u6NDTB = 1;  /* Number of Dedicated Tx Buffers = 1 */
unTXBC.stcField.u14TBSA = 0x00000200; /* offset(word) */
CPG_CANFD0_TXBC = unTXBC.u32Register;

/* Configuration of Tx Buffer Element Size */
unTXESC.stcField.u3TBDS = 7; /* 64 byte data field. */
CPG_CANFD0_TXESC = unTXESC.u32Register;

/* Configuration of ID Filter */
/* Standard Message ID Filter */
pstcIdFilter = (stc_id_filter_t*)((uint32_t*)((uint32_t)&CPG_CANFD0 +
               (uint32_t)0x00008000) + CPG_CANFD0_SIDFC_FLSSA);
pstcIdFilter->SFT   = 2;        /* Standard Filter Type : Classic filter */
pstcIdFilter->SFEC  = 7;        /* Store into dedicated Rx Buffer,    */
                                /* configuration of SFT[1:0] ignored. */
pstcIdFilter->SFID1 = 0x010;    /* Filter ID : 0x10(16) */
pstcIdFilter->SFID2 = (0 << 9)  /* Store message into a dedicated Rx Buffer */
                    | 0;        /* Buffer index : 0 */

/* Extended Message ID Filter */
pstcExtIdFilter = (stc_extid_filter_t*)((uint32_t*)((uint32_t)&CPG_CANFD0 +
                  (uint32_t)0x00008000) + CPG_CANFD0_XIDFC_FLESA);
pstcExtIdFilter->F1_f.EFT   = 2;        /* Extended Filter Type : Classic filter */
pstcExtIdFilter->F0_f.EFEC  = 7;        /* Store into dedicated Rx Buffer,    */
                                        /* configuration of EFT[1:0] ignored. */
pstcExtIdFilter->F0_f.EFID1 = 0x10001;  /* Filter ID : 0x10001(65537) */
pstcExtIdFilter->F1_f.EFID2 = (0 << 9)  /* Store message into a dedicated Rx Buffer */
                            | 1;        /* Buffer index : 1 */

/* Configuration of Interrupt */

/* Interrupt Enable */
unIE.stcField.u1DRXE  = 1; /* Message stored to Dedicated Rx Buffer */
CPG_CANFD0_IE = unIE.u32Register;

/* Interrupt Line Select */
unILS.u32Register = 0;  /* Interrupt line canfd_int0 */
CPG_CANFD0_ILS = unILS.u32Register;
```

Rx FIFO 1 設定

Rx バッファ設定

Rx バッファ/FIFO エレメント設定

Tx イベント FIFO 設定

Tx バッファ設定

Tx バッファエレメントサイズ設定

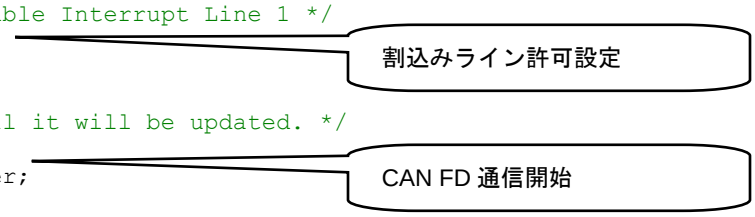
標準 ID フィルタ設定

拡張 ID フィルタ設定

割込み許可設定

割込みライン選択設定

```
    /* Interrupt Line Enable */
    unILE.stcField.u1EINT0 = 1; /* Enable Interrupt Line 0 */
    unILE.stcField.u1EINT1 = 0; /* Disable Interrupt Line 1 */
    CPG_CANFD0_ILE = unILE.u32Register;

    /* CAN FD operation start */
    /* Set CCCR.INIT to 0 and wait until it will be updated. */
    unCCCR.stcField.u1INIT = 0;
    CPG_CANFD0_CCCR = unCCCR.u32Register;

    while ( CPG_CANFD0_CCCR_INIT != 0 )
    {
    }

}
```



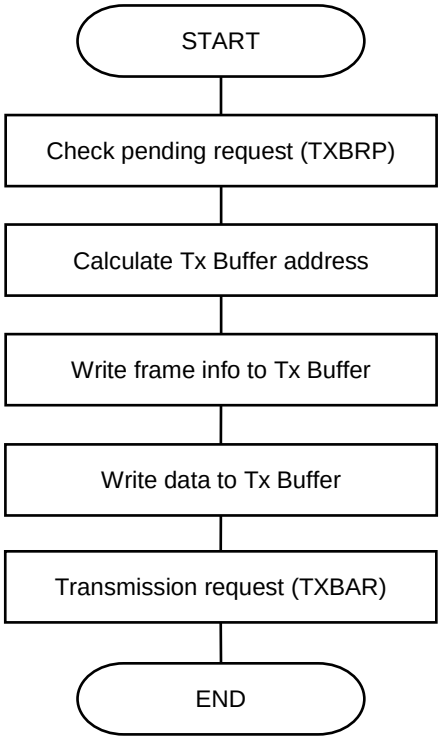
## 4.4 メッセージ送信

Figure 7 はメッセージ送信フローの例です。この例では Tx 割込みを使用していません。

メッセージは、メッセージ RAM の Tx バッファを介して送信されます。保留中の要求(TXBRP)がなければ、メッセージ RAM の Tx バッファのアドレスを算出し、CAN FD コントローラが送信するフレームの情報とデータを書き込みます。Tx Buffer Add Request (TXBAR)に書き込むことにより、メッセージ送信が開始されます。

次のフローのサンプルコードを 4.4.1 に示します。

Figure 7. メッセージ送信フローの例



### 4.4.1 メッセージ送信のサンプルコード

CAN FD メッセージ送信のサンプルコードを Code 4 に示します。このサンプルコードでは Tx 割込みを使用していません。

Code 4. メッセージ送信

```
void CanFD_UpdateAndTransmitMsgBuffer(void)
{
    uint8_t u8DataLengthWord;
    stc_canfd_tx_buffer_t* pstcCanFDTxBuffer;
    uint16_t u16count;
    uint16_t u16dlcTmp;
    uint16_t u16MessageBufferNumber ;
    uint8_t  u8DataLengthCode;
    uint32_t au32Data[16] = { 0 };

    /* Message Buffer 0 */
    u16MessageBufferNumber = 0;

    /* Check whether Tx buffer is empty or not */
    while ( 0 != (CPG_CANFD0_TXBRP & 0x00000001 )) /* Check the TRP0 */
    {
    }

    /* Get Tx Buffer address */
    pstcCanFDTxBuffer = (stc_canfd_tx_buffer_t *)((uint32_t *)((uint32_t)&CPG_CANFD0 +
                        (uint32_t)0x00008000) + CPG_CANFD0_TXBC_TBSA +
                        (2 + iDlcInWord[CPG_CANFD0_TXESC_TBDS]) * u16MessageBufferNumber) ;

    /* Set data to Tx buffer */
    pstcCanFDTxBuffer->T0_f.RTR = 0; /* Transmit data frame. */
    pstcCanFDTxBuffer->T0_f.XTD = 0; /* 11-bit ID */
    pstcCanFDTxBuffer->T0_f.ID  = 0x200 << 18; /* Identifier */
    pstcCanFDTxBuffer->T1_f.EFC = 0; /* Not store Tx event FIFO */
    pstcCanFDTxBuffer->T1_f.MM  = 0; /* Not used */
    pstcCanFDTxBuffer->T1_f.DLC = 15; /* DataLengthCode *
    pstcCanFDTxBuffer->T1_f.FDF = 1; /* CAN FD format */
    pstcCanFDTxBuffer->T1_f.BRS = 1; /* Transmitted with bit rate switching */

    /* Convert the DLC to word data type */
    u8DataLengthCode = pstcCanFDTxBuffer->T1_f.DLC;
    u16dlcTmp = u8DataLengthCode - 8;
    u8DataLengthWord = iDlcInWord[u16dlcTmp];

    /* Data set */
    au32Data[0]              = 0x12345678;    /* Data of CAN FD message */
    au32Data[1]              = 0x9abcdef9;    /* Data of CAN FD message */
    au32Data[2]              = 0x12345677;    /* Data of CAN FD message */
    au32Data[15]             = 0x87654321;    /* Data of CAN FD message */

    for ( u16count = 0; u16count < u8DataLengthWord; u16count++ )
    {
        pstcCanFDTxBuffer->DATA_AREA_f[u16count] = au32Data[u16count];
    }

    /* Transmission request */
    CPG_CANFD0_TXBAR = 0x00000001; /* request for buffer 0 */
}
```

Tx Buffer 0

保留中のリクエストをチェック

Tx Buffer 0 のアドレスを算出

フレーム情報を Tx Buffer 0 に書き込む

DLC を word データ型へ変換

データを TxBuffer 0 へ書き込む

送信要求

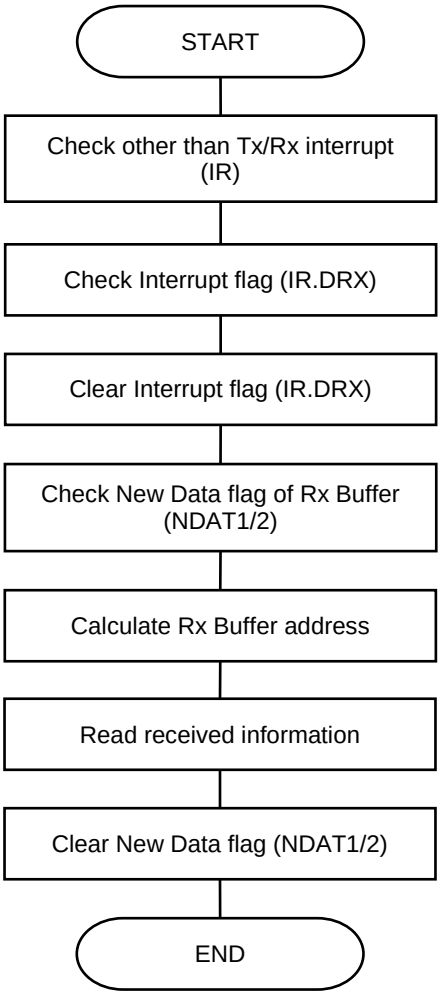
## 4.5 メッセージ受信

Figure 8 は、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です。

受信したメッセージはメッセージフィルタによってフィルタリングされ、メッセージ RAM の Rx バッファに格納されます。このイベントで割込みが発生します。メッセージが専用の Rx バッファに格納されると、Interrupt Register (IR.DRX)と New Data 1/2 (NDAT 1/2)の対応するビットがセットされます。そして、Rx バッファの開始アドレスを算出し、算出されたアドレスから受信したメッセージ情報を読み出します。

次のフローのサンプルコードを 4.5.1 に示します。

Figure 8. メッセージ受信フローの例



### 4.5.1 メッセージ受信のサンプルコード

CAN FD メッセージ受信のサンプルコードを Code 5 に示します。このサンプルコードでは Rx バッファと Rx 割込みを使用しています。

Code 5. メッセージ受信

```
/* CAN FD Reception interrupt routine */
FN_IRQ_DEFINE_BEGIN(CanFD_Isr_CanFD0, INTERRUPTS_IRQ_NUMBER_56)
{
    uint32_t*    pulAdrs = 0;
    uint16_t     u16count = 0;
    uint16_t     u16dlcTmp = 0;
    uint16_t     u16MessageBufferNumber ;
    stc_canfd_msg_t stcCanFDmsg;

    /* Other than Tx/Rx interrupt occured */
    if ( CPG_CANFD0_IR & 0x3ff7E0EE )
    {
        /* User handling */
    }

    /* Received a data frame */
    if ( CPG_CANFD0_IR_DRX == 1 ) /* At least one received message stored */
                                  /* into an Rx Buffer */
    {
        /* Clear the Message stored to Dedicated Rx Buffer flag */
        CPG_CANFD0_IR_DRX = 1;

        /* New data is exist */
        if (CPG_CANFD0_NDAT1 & 0x00000001)      /* Check the message buffer 0 */
        {
            /* Message Buffer 0 */
            u16MessageBufferNumber = 0;
            /* Rx Buffer address */
            pulAdrs = (uint32_t *)((uint32_t)&CPG_CANFD0 + (uint32_t)0x00008000) +
                      CPG_CANFD0_RXBC_RBSA + (2 + iDlcInWord[CPG_CANFD0_RXESC_RBDS]) *
                      u16MessageBufferNumber;

            /* Clear NDAT1 register */
            CPG_CANFD0_NDAT1 = 0x00000001;
        }
        else if (CPG_CANFD0_NDAT1 & 0x00000002)   /* Check the mess
        {
            /* Message Buffer 1 */
            u16MessageBufferNumber = 1;
            /* Rx Buffer address */
            pulAdrs = (uint32_t *)((uint32_t)&CPG_CANFD0 + (uint32_t)0x00008000) +
                      CPG_CANFD0_RXBC_RBSA + (2 + iDlcInWord[CPG_CANFD0_RXESC_RBDS]) *
                      u16MessageBufferNumber;

            /* Clear NDAT1 register */
            CPG_CANFD0_NDAT1 = 0x00000002;
        }

        if (pulAdrs)
        {
            /* Save received data */
            /* XTD : Extended Identifier */
            stcCanFDmsg.stcIdentifier.bExtended = ((stc_canfd_rx_buffer_t *) pulAdrs)
                                                  ->R0_f.XTD;
```

Tx/Rx 割込み以外をチェック

IR.DRX フラグチェック

割込みフラグクリア

Rx Buffer 0 の New Data フラグをチェック

Rx Buffer 0 アドレスを算出

New Data フラグクリア

Rx Buffer 1 の New Data フラグをチェック

Rx Buffer 1 アドレスを算出

New Data フラグクリア

拡張 ID ビット読出し

```
            /* ID : RxID */
            if ( stcCanFDmsg.stcIdentifier.bExtended == FALSE )
            {

                stcCanFDmsg.stcIdentifier.u32Identifier =
                                    ((stc_canfd_rx_buffer_t *) pulAdrs)->R0_f.ID >> 18;
            }
            else
            {
                stcCanFDmsg.stcIdentifier.u32Identifier =
                                        ((stc_canfd_rx_buffer_t *) pulAdrs)->R0_f.ID;
            }

            /* FDF : Extended Data Length */
            stcCanFDmsg.bCanFDFormat = ((stc_canfd_rx_buffer_t *) pulAdrs)->R1_f.FDF;


            /* DLC : Data Length Code */
            stcCanFDmsg.stcData.u8DataLengthCode =
                                        ((stc_canfd_rx_buffer_t *) pulAdrs)->R1_f.DLC;

            /* Copy 0-64 byte of data area */
            if ( stcCanFDmsg.stcData.u8DataLengthCode < 8 )
            {
                u16dlcTmp = 0;
            }
            else
            {
                u16dlcTmp = stcCanFDmsg.stcData.u8DataLengthCode - 8;
            }

            for ( u16count = 0; u16count < iDlcInWord[u16dlcTmp]; u16count++ )
            {
                stcCanFDmsg.stcData.au32Data[u16count] =
                            ((stc_canfd_rx_buffer_t *) pulAdrs)->DATA_AREA_f[u16count];
            }
        }
    }
    else if ( CPG_CANFD0_IR_TC == 1 ) /* Transmission completed */
    {
    }
}
FN_IRQ_DEFINE_END()
```

ID 読出し

拡張データ長ビット読出し

DLC 読出し

データ読出し

# 5 関連ドキュメント

- S6J311E/D/C/B Series 32-bit Microcontroller Traveo Family Datasheet
- S6J311A/9/8 Series 32-bit Microcontroller Traveo Family Datasheet
- S6J3110 Series 32-bit Microcontroller Traveo Family Hardware Manual
- S6J3120 Series 32-bit Microcontroller Traveo Family Datasheet
- S6J3120 Series 32-bit Microcontroller Traveo Family Hardware Manual
- S6J3200 Series 32-bit Microcontroller Traveo Family Datasheet
- S6J3200 Series 32-bit Microcontroller Traveo Family Hardware Manual
- S6J32E/F/G Series 32-bit Microcontroller Traveo Family Datasheet
- S6J32E/F/G Series 32-bit Microcontroller Traveo Family Hardware Manual
- Traveo Family Hardware Manual Platform Part for S6J3200 Series
- S6J3300 Series 32-bit Microcontroller Traveo Family Datasheet
- S6J3350 Series 32-bit Microcontroller Traveo Family Datasheet
- S6J3300/S6J3350 Series 32-bit Microcontroller Traveo Family Hardware Manual
- S6J3400 Series 32-bit Microcontroller Traveo Family Datasheet
- S6J3400 Series 32-bit Microcontroller Traveo Family Hardware Manual
- Traveo Family Hardware Manual Platform Part for S6J3300/S6J3350/S6J3400 Series
- S6J3360/70 Series Datasheet (Doc.No.002-03359)
- S6J3360/70 Series Hardware Manual (Doc.No.002-18302)
- Traveo Family HardwareManual Platform Part for S6J3360/3370 Series (Doc.No.002-07884)
- S6J3510 Series Datasheet (Doc.No.002-18647)
- S6J3510 Series Hardware Manual (Doc.No.002-18642)
- Traveo Family Hardware Manual Platform Part for S6J3510 Series (Doc.No.002-07884)

# 6 まとめ

このアプリケーションノートでは、主に車載アプリケーションで使用される CAN FD の使用方法ついて説明しました。

# 改定履歴

ドキュメント名: AN213891 - Traveo™ ファミリ CAN FD の使用方法

ドキュメント番号: 002-20669

| Revision | ECN | 変更者 | 変更日 | 変更内容 |
|---|---|---|---|---|
| ** | 5874294 | SITO | 09/06/2017 | このドキュメントは英語版 002-13891 Rev *A を翻訳した日本語 002-20669 Rev ** です。 |

## ワールドワイドな販売と設計サポート

サイプレスは、事業所、ソリューションセンター、メーカー代理店、および販売代理店の世界的なネットワークを保持しています。お客様の最寄りのオフィスについては、サイプレスのロケーション ページをご覧ください。

### 製品

| | |
|---|---|
| ARM® Cortex® Microcontrollers | cypress.com/arm |
| 車載用 | cypress.com/automotive |
| クロック&バッファ | cypress.com/clocks |
| インターフェース | cypress.com/interface |
| IoT (モノのインターネット) | cypress.com/iot |
| 照明&電力制御 | cypress.com/powerpsoc |
| メモリ | cypress.com/memory |
| PSoC | cypress.com/psoc |
| タッチ センシング | cypress.com/touch |
| USB コントローラー | cypress.com/usb |
| ワイヤレス／RF | cypress.com/wireless |

### PSoC® ソリューション

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

### サイプレス開発者コミュニティ

コミュニティ | フォーラム | ブログ | ビデオ | トレーニング | Components

### テクニカルサポート

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709