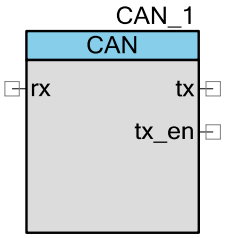
コントローラ・エリア・ネットワーク(CAN)

2.1

特長



- CAN2.0A/B プロトコルの実装、ISO11898-1 準拠
- 8MHz (BUS_CLK)で最大 1Mbps のプログラム可能なビットレート
- 外部トランシーバに対し 2 線または 3 線のインタフェース(Tx、Rx、Enable)
- 1 データバイトと 2 データバイトフィールドをカバーする拡張ハードウェア・メッセージ・フィルター
- プログラム可能な送信プライオリティ:ラウンド・ロビンおよび固定
- PSoC 5 ではサポート対象外

概要説明

コントローラ・エリア・ネットワーク(CAN)コントローラは、Bosch 社の仕様書に規定された CAN2.0A および CAN2.0B 仕様を実現し、ISO-11898-1 規格に準拠しています。

CANの用途

CAN プロトコルは、もともと、高レベルのエラー検出に重点を置いた車載アプリケーションのために設計されたものです。これによって、信頼性の高い通信が低コストで実現されます。CAN は車載アプリケーションで成功したため、動き重視の機械制御ネットワーク(CANOpen)およびファクトリ オートメーション用途(DeviceNet)向けの標準通信プロトコルとして使用されています。CAN コントローラの機能によって、マイクロコントローラの CPU の性能に影響を与えずに高レベルのプロトコルを効率的に実現できます。

入出力接続

このセクションでは、CAN コンポーネントの入出力接続について説明します。I/O リストにアスタリスク(*)が付いている場合、その I/O の説明でリストアップされている条件で、複数の I/O が * シンボル(ワイルドカード)に隠れている可能性があることを示します。

rx – 入力

CAN バス受信信号(外部トランシーバの CAN Rx バスに接続)。

tx– 出力

CAN バス送信信号(外部トランシーバの CAN Tx バスに接続)。

tx_en – 出力 *

外部トランシーバ対応信号。この出力は、**Configure** ダイアログで **Add Transceiver Enable Signal** オプションを選択した場合に表示されます。

interrupt – 出力 *

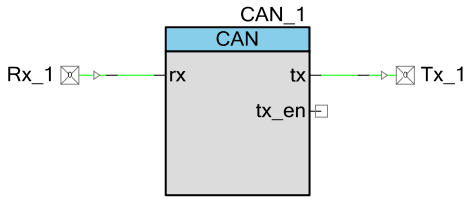
interrupt出力は、CANハードウェアで設定された割り込みソースによって駆動されます。全ソースの OR をとって、最終出力信号が作成されます。割り込みのソース:

- メッセージ送信
- メッセージ受信
- 受信バッファフル
- バス オフ状態
- CRC エラー検出
- メッセージ・フォーマット・エラーを検出
- メッセージ確認、エラーを検出
- ビットスタッフィングエラーを検出
- ビットエラーを検出
- オーバーロードフレームを検出
- アービトレージョン(調停)の不調を検出

この出力は、**Configure** ダイアログの **Interrupt** タブの **Advanced Interrupt Configuration…** ウィンドウで **Enable External Interrupt Line** オプションが選択されている場合に表示されます。

回路図マクロ情報

コンポーネントカタログにあるデフォルトの CAN は、デフォルト設定の CAN コンポーネントを使用した回路図マクロです。CAN コンポーネントは、入力および出力のピン・コンポーネントに接続されます。また、ピン・コンポーネントも、入力ピン・コンポーネントの場合 [Input Synchronized] が false に設定されることを除いて、初期設定で設定されます。



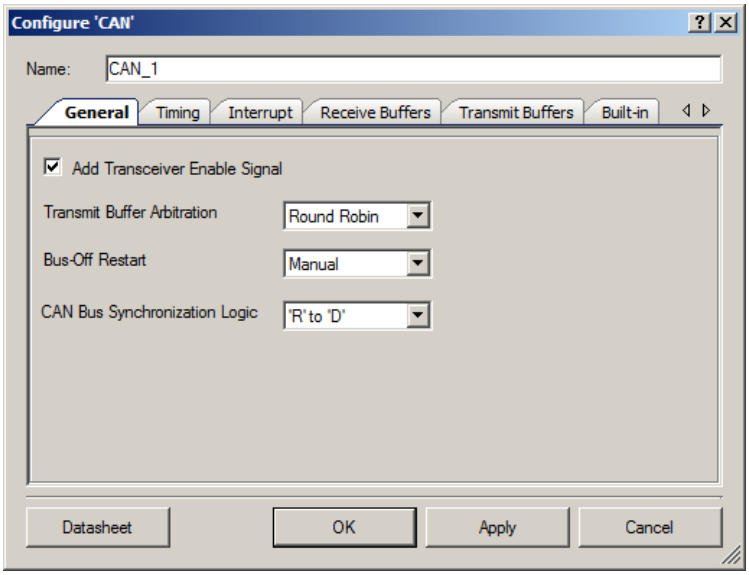
コンポーネント・パラメータ

CAN コンポーネントをデザイン上にドラッグし、ダブルクリックして **Configure** ダイアログを開きます。このダイアログには、CAN コンポーネントのセットアップをガイドする複数のタブがあります。

コンポーネントの更新ノート

以前のバージョンから CAN コンポーネントを更新した場合、多くのパラメータのフォーマットが新しくなっていることに気付くでしょう。これらは変換することが必要です。これを行うには、**Configure** ダイアログを開き、少なくとも 1 つのパラメータ/オプションを変更し、**OK** をクリックして変更を保存します。

General(一般)タブ



General(一般)タブには、以下の設定があります:

Add Transceiver Enable Signal (トランシーバ・イネーブル信号を追加)

外部 CAN トランシーバに対して、tx_en 信号の使用をイネーブルまたはディスエーブルします。初期設定ではイネーブルになっています。

Transmit Buffer Arbitration (送信バッファ調停)

メッセージ転送の調停のスキームを定義:

- **ラウンド・ロビン** (初期設定) – バッファは、定義された順番で使用されます。0-1-2 ... 7-0-1. TxReq フラグが設定されている場合、特定のバッファが選択できるだけです。このスキームは、すべてのバッファが同じ確率でメッセージを送信できることを保証します。
- **固定優先順位** – バッファ 0 が最高の優先順位。こうして、バッファ 0 をエラー・メッセージのバッファとして指定し、必ず最初に送信することを保証できます。

Bus-Off Restart (バス-オフ 再起動):

リセットのタイプを設定:

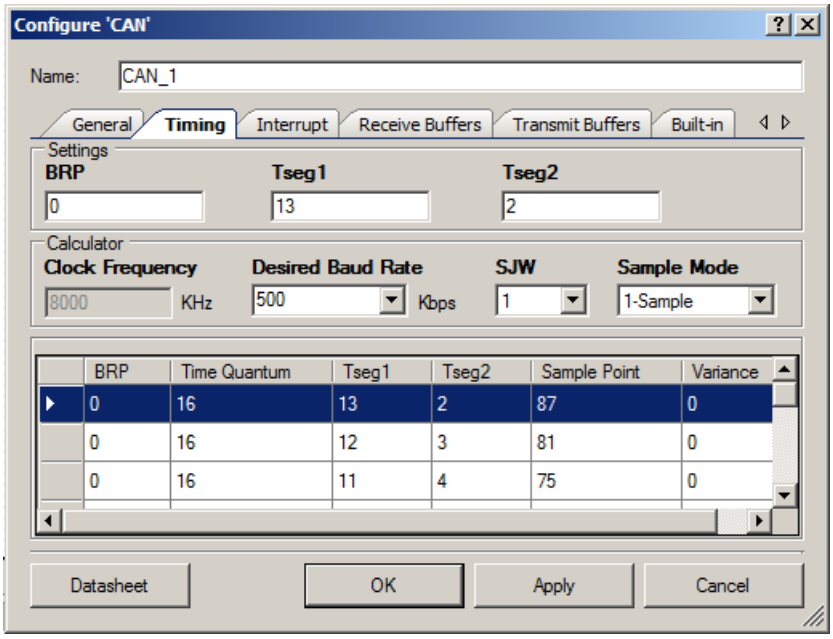
- **Manual** (初期設定) – バスがオフになると、CAN を再起動する必要があります。これは、推奨される設定です。
- **Automatic** – バスがオフになると、128 グループの 11 の recessive(レセシブ)ビットの後に、CAN コントローラが自動的に再起動します。

CAN Bus Synchronization Logic (CAN バス同期ロジック):

エッジの同期を設定:

- **'R' to 'D'** (初期設定) – 'R'(Recessive(劣性))から'D'(Dominant(優性))遷移のエッジのみを同期で使用します。
- **Both edges** – 両方のエッジを、同期で使用します。

Timing(タイミング)タブ

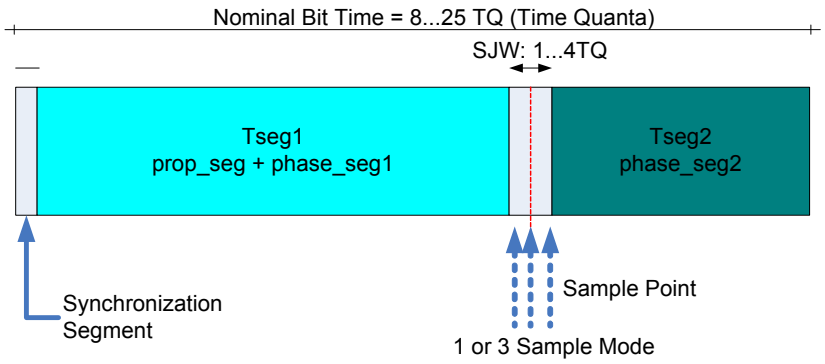


Timing タブには、以下の設定があります:

設定

- **BRP** – ビット・レート・プリスケーラ の値はタイムクォンタムを生成するのに使用します。この値は、ビット・タイミング計算機で計算します。0 は 1 クロックです。7FFFh は 32767 のクロック・サイクルを表します。15 ビットです。
- **Tseg1** – 時間セグメント 1 の値です。
- **Tseg2** – 時間セグメント 2 の値です。値 0 および 1 は許可されません。2 は、**Sample Mode** を直接サンプリング(**1-Sample**)に設定している場合にだけ使用できます。

  以下は、CAN のビット時間表現を示しています:



Calculator(計算機)

- **クロック周波数**(単位 MHz) – BUS_CLK に等しいシステム・クロックの周波数。
- **望ましいボー・レート** (単位 Kbps) – 利用できるオプション:**10**、**20**、**62.5**、**125**、**250**、**500**、**800**、または **1000**。
- **SJW** – 同期ジャンプ幅(2ビット)の設定。この値は Tseg1 以下、かつ Tseg2 以下です。利用できるオプション:**1**、**2**、**3**、または **4**。
- **Sample Mode** – サンプリング・モードの設定。利用できるオプション:**1 サンプル**または **3 サンプル**。

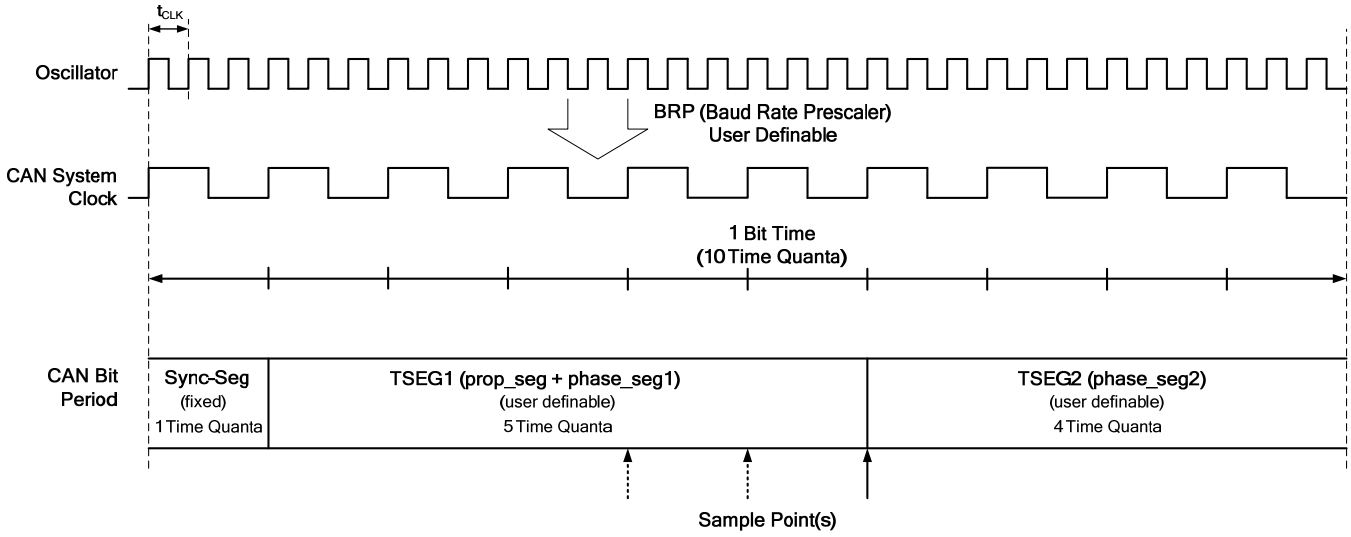
Table(表)

ビット時間を計算した後、提示された時間セグメント(Tseg1 と Tseg2)、BRP レジスタ設定がパラメータ・テーブルに表示されます。該当する行をダブルクリックして、ロードする値を選択できます。選択した値は、上の **Settings** 入力ボックスに表示されます。

また、提示される入力ボックスで、Tseg1、Tseg2、BRP の値を手動で選択することもできます。

注 ビット時間の設定が正しくないと、CAN コントローラはエラー状態のままになります。

次の図は、発振器からどのようにしてすべてのタイミングが導かれるかの例を示しています。



ビット時間セグメント

SYNC SEG (同期セグメント)

ビット時間のこの部分は、バスのさまざまなノードと同期します。エッジは、セグメント内になることが予想されます。

PROP SEG (伝播時間セグメント)

ビット時間のこの部分は、ネットワークでの物理的遅延時間を補正します。バス・ラインでの信号の伝播時間、入力コンパレータの遅延、出力ドライバーの遅延の総和の 2 倍になります。

PHASE SEG1、PHASE SEG2 (フェーズ・バッファ・セグメント 1/2)

これらのフェーズ・バッファ・セグメントは、エッジ・フェーズのエラーを補正します。これらのセグメントは、再同期によって延長または短縮されます。

サンプルポイント

サンプルポイントとは、バスのレベルが読み取られ、それぞれのビットの値として解釈される一瞬です。PHASE_SEG1 の終わりに位置します。

情報処理時間

情報処理時間は、以後のビットのレベルを計算するために予約されているサンプルポイントで開始する時間セグメントです。

タイムクォンタム

タイムクォンタムは、発振器の周期から導かれる時間の固定単位です。積分値が 1～32767 の範囲である、プログラム可能プリスケーラ(BRP)があります。最小タイムクォンタムで始まり、タイムクォンタムは次の長さになります:

TIME QUANTUM = m × MINIMUM TIME QUANTUM

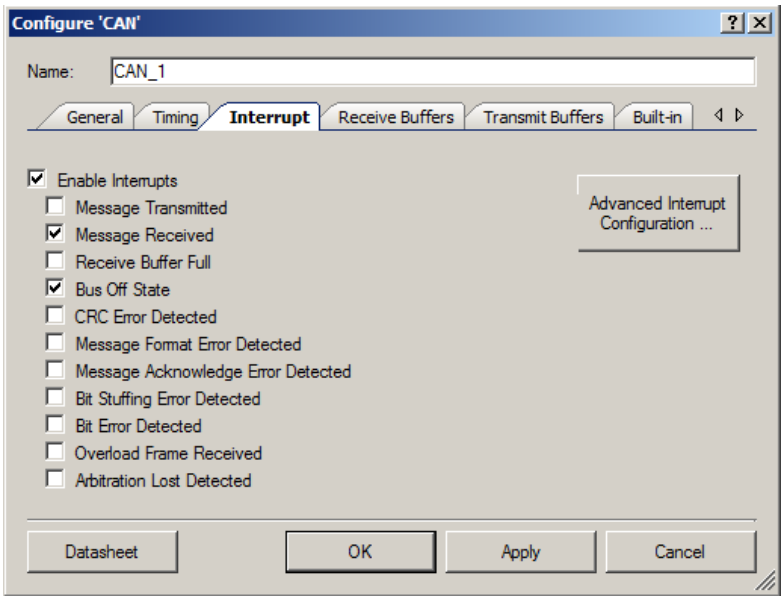
ここで、m はプリスケーラの値です。

時間セグメントの長さ

- SYNC_SEG は、1 単位のタイムクォンタムの長さです。
- PROP_SEG は、タイムクォンタムの長さの 1、2、…、8 倍にプログラム可能です。
- PHASE_SEG1 は、タイムクォンタムの長さの 1、2、…、8 倍にプログラム可能です。
- PHASE_SEG2 は、PHASE_SEG1 および情報処理時間の最大値です。

割り込みタブ

基本割り込み設定



Basic Interrupt Configuration タブには、以下の設定があります:

Enable Interrupts

CAN コントローラからのグローバル割り込みをイネーブルまたはディスエーブルにします。初期設定ではイネーブルになっています。

Enabled – CAN コンポーネントが CAN_Start()を使用して起動されている場合、グローバル割り込みはイネーブルになります。

Disabled – CAN コンポーネントが CAN_Start()を使用して起動されている場合、グローバル割り込みはイネーブルになりません。グローバル割り込みのイネーブル・ビットが設定されるまで、CAN ISR は入力されません。CAN_GlobalIntEnable()または CAN_GlobalIntDisable()を使用して、メインコードでグローバル割り込みをイネーブルまたはディスエーブルにするのはユーザーの責任です。

メッセージ送信

メッセージが送信されたことを示すメッセージ転送割り込みをイネーブルまたはディスエーブルにします。初期設定ではディスエーブルになっています。メッセージ転送割り込みのオプションがディスエーブルになっていれば、CAN は以下のメッセージを表示します: **すべてのメッセージ転送割り込みをディスエーブルにしますか**?

- **Yes** – **Message Transmitted** チェックボックスの選択を解除し、**Transmit Buffers** タブのすべての個々の転送バッファ割り込みの選択を解除します。

- **No** (初期設定) – **Message Transmitted** チェックボックスの選択を解除し、**Transmit Buffers** タブのすべての個々の転送バッファ割り込みの選択をそのままにします。
- **Cancel** – 変更なし。

メッセージ受信

メッセージを受信したことを示すメッセージ受信割り込みをイネーブルまたはディスエーブルにします。初期設定ではイネーブルになっています。メッセージ受信割り込みのオプションをディスエーブルにすると、CAN は以下のメッセージを表示します : **すべてのメッセージ受信割り込みをディスエーブルにしますか?**

- **Yes** (初期設定) – **Message Received** チェックボックスの選択を解除し、**Receive Buffers** タブのすべての個々の受信バッファ割り込みの選択を解除します。
- **No** (削除) – **Message Received** チェックボックスの選択を解除し、**Receive Buffers** タブのすべての個々の受信バッファ割り込みの選択をそのままにします。
- **Cancel** – 変更なし。

受信バッファフル

前のメッセージが確認されていない時に、CAN コントローラが新しいメッセージを受信したことを示す、メッセージ消失割り込みをイネーブルまたはディスエーブにします。初期設定ではディスエーブルになっています。

バス オフ状態

CAN ノードが[BUS Off]状態に達したことを示す、バス・オフ割り込みをイネーブルまたはディスエーブにします。初期設定ではイネーブルになっています。

CRCエラー検出

CAN コントローラが CAN CRC エラーを検出したことを示す、CAN エラー割り込みをイネーブルまたはディスエーブにします。初期設定ではディスエーブルになっています。

メッセージ・フォーマット・エラー検出

CAN コントローラが CAN メッセージ・フォーマット・エラーを検出したことを示す、メッセージ・フォーマット・エラー割り込みをイネーブルまたはディスエーブルにします。初期設定ではディスエーブルになっています。

メッセージ確認エラー検出

CAN コントローラが CAN メッセージ確認エラーを検出したことを示す、メッセージ確認エラーを割り込みイネーブルまたはディスエーブルにします。初期設定ではディスエーブルになっています。

ビットスタッフィングエラー検出

CAN コントローラがビットスタッフィング・エラーを検出したことを示す、ビットスタッフィング・エラーを割り込みイネーブルまたはディスエーブルにします。初期設定ではディスエーブルになっています。

ビットエラー検出

CAN コントローラがビット・エラーを検出したことを示す、ビット・エラーの割り込みをイネーブルまたはディスエーブルにします。初期設定ではディスエーブルになっています。

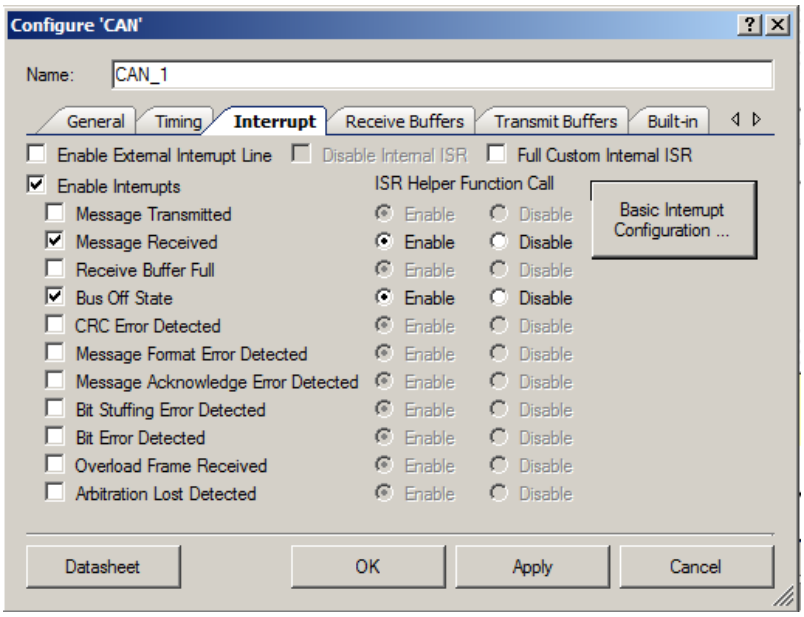
オーバーロードフレーム検出

CAN コントローラがオーバーロードフレームを受信したことを示す、オーバーロード割り込みをイネーブルまたはディスエーブルにします。初期設定ではディスエーブルになっています。

アービトレージョン(調停)不調検出

キューに蓄えるメッセージのアービトレーションおよびキャンセルの管理をイネーブルまたはディスエーブルにします。この割り込みは、メッセージの送信中にアービトレーションが失われたことを示しています。初期設定ではディスエーブルになっています。

高度な割り込み設定



Advanced Interrupt Configuration タブには、以下の設定があります:

Enable External Interrupt Line（外部割込みラインのイネーブル）

CAN ブロック割り込みラインの外部可視性および接続性をイネーブルにします。初期設定ではディスエーブルです (外部割込みラインは、CAN コンポーネントのシンボルのインスタンスでは表示されません)。

Disable Internal ISR（内部ISRのディスエーブル）

内部 ISR コンポーネントをディスエーブルにする、あるいはバイパスします。内部 ISR がディスエーブルであれば、関連する CAN API は ISR 起動/停止プロセスを管理しません。初期設定ではディスエーブル (内部 ISR がイネーブル)。チェックボックスは **Enable External Interrupt Line** が選択されている場合にだけ利用可能(グレイアウトされない)です。内部 ISR をディスエーブルにできるのは、割り込みを処理できる代替手段(外部割込みライン)がある場合だけです。

Full Custom Internal ISR(フルカスタム内部ISR)

イネーブルなら完全なカスタムコードによる内部 ISR を使用します。このオプションを選択すると、CAN_ISR にはオプションの PSoC 3 ES1/ES2 ISR パッチ以外のコードは含まれません。次のラインの間に、カスタムコードを配置します:

```
/* Place your Interrupt code here. */
/* `#START CAN_ISR` */

/* `#END` */
```

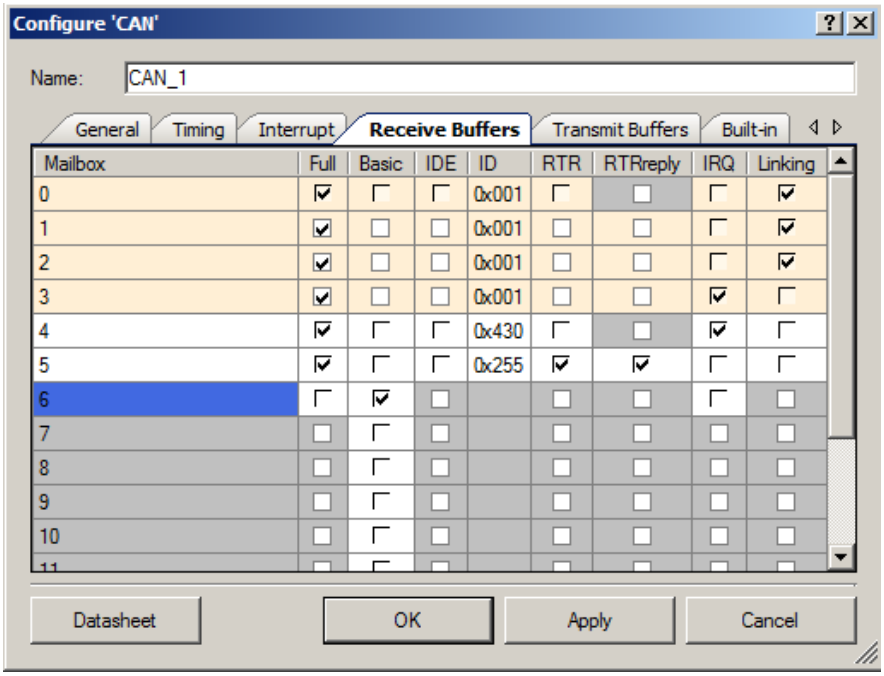
初期設定はディスエーブル (初期設定は CAN v1.50 ISR 処理)です。**Disable Internal ISR** が選択されている場合、チェックボックスは使用できません(グレーアウトされます)。

ISR Helper Function Call（ISRヘルパーファンクションコール)

基本割り込み設定だけ(たとえば、CAN v1.50)を選択すると、CAN ISR は、割り込みが発生した場合に、割り込みのイネーブル状態にもとづて、関連するユーザーがカスタマイズした関数(ISR helpers) を呼び出します。

これらのオプションによって、特定の割り込みのカスタム処理をハードウェアおよびファームウェアの両方で実装できるように、ISR helper の呼び出しをイネーブルまたはディスエーブルにできます。初期設定は**イネーブル**です。これらのオプションを利用できるのは、関連する割り込みがイネーブルであり、**Full Custom Internal ISR** が選択されておらず、**Disable Internal ISR** も選択されていない場合だけです。

Receive Buffers(レシーブバッファ)タブ



Receive Buffers タブには、以下の設定があります:

Mailbox(メールボックス)

Full(完全版)または **Basic(基本版)**を選択するまでは、受信メールボックスはディスエーブルです。すべてのディスエーブルであるメールボックスでは、**IDE**、**ID**、**RTR**,、**RTRreply**、および **IRQ** のフィールドはロックされます。

完全版メールボックスでは、**Mailbox** フィールドは編集可能であり、固有のメールボックス名を入力できます。各メールボックスを管理するために提供される API には、メールボックスの文字列が追加されます。使用可能なシンボル:A–Z, a–z, 0–9, および _. 間違った名前を入力すると、エラーメッセージが表示され、**Mailbox** フィールドは初期設定値に戻ります。

Full(完全版)

Full(完全版)を選択すると、**Mailbox**、**IDE**、**ID**、**RTR**、**RTRreply**、**IRQ**、および **Linking** フィールドを変更できます。フィールドは、以下の初期設定値で初期化されます:

- **Mailbox** = メールボックス番号 0～15
- **IDE** = クリア
- **ID** = 0x001
- **RTR** = クリア

- **RTRreply** = クリアおよびロック (**RTR** が選択されている場合だけイネーブル)
- **IRQ** = 選択、**Message Received** (**Interrupt** タブ)割り込みが選択されている場合だけ使用可能
- **Linking** = クリア

Basic(基本版)

Basic(基本版)を選択した場合、オプションの **IDE**、**ID**、**RTR**、および **RTRreply** は使用できません。フィールドは、以下の初期設定値で初期化されます:

- **IDE** = クリア (使用不可)
- **ID** = <All> (使用不可)
- **RTR** = クリア (使用不可)
- **RTRreply** = クリア (使用不可)
- **IRQ** = クリア、**Message Received** 割り込みを選択した場合だけ使用可能
- **Linking** = クリア

IDE

IDE チェックボックスをクリアすると、識別子は 11 ビット(0x001～0x7FE)に限定されます。**IDE** を選択した場合は、識別子は 29 ビット(0x00000001～0x1FFFFFFE)になります。

RTR - リモート送信要求

全 CAN メッセージ受信対応するメールボックスの設定でのみ使用可能です。選択すると、RTR ビットが設定されているメッセージだけを受信するように受入れフィルターを設定します。

RTRreply - リモート転送要求自動応答

RTR ビットが設定されているメッセージだけを受信するようにメールボックスを設定する場合だけ使用可能です。選択すると、受信バッファのコンテンツで RTR 要求に自動的に応答します。

IRQ(割り込み要求)

メールボックスに対して IRQ をイネーブルにすると、**Interrupt** タブの **Message Received Interrupt** がクリアされ、以下のメッセージが表示されます:**グローバル"メッセージ受信割り込み"はディスエーブルです。イネーブルにしますか?**

- **Yes** –**Interrupt** タブの **IRQ** チェックボックスを選択し、**Message Received Interrupt** チェックボックスを選択します。
- **No** または **Cancel** – **Interrupt** タブの **IRQ** チェックボックスを選択し、**Message Received Interrupt** チェックボックスはそのままにします。

リンク

Linking チェックボックスにより、複数の連続する受信メールボックスをリンクし、受信メールボックスのアレイを作成します。このアレイは受信 FIFO と似た動作をします。同じアレイのすべてのメールボックスには、同じメッセージフィルタ設定が必要になります。すなわち受入れマスクレジスタ (AMR) および受入れコードレジスタ (ACR) は同一になります。

- アレイの最後のメールボックスには、リンクのフラグを設定できません。
- 最後のメールボックス 15 には、リンクのフラグを設定できません。
- リンクされているすべてのメールボックスは同じ色で強調表示されます。
- リンクされているアレイの最初のメールボックスのみが編集可能です。同じアレイ内のすべてのリンクされているメールボックスに、すべてのパラメータが自動的に適用されます。
- リンクされているすべてのメールボックス用に 1 つの関数が生成されます。

受信メッセージ関数

すべての Full(完全版) RX メールボックスには事前定義された API があります。関数のリストは *CAN_1_TX_RX_func.c* プロジェクトファイルにあります。これらの関数は、受信メールボックス設定により、条件付きでコンパイルされます。Full(完全版) に定義されているメールボックスでのみ、それぞれの関数がコンパイルされます。

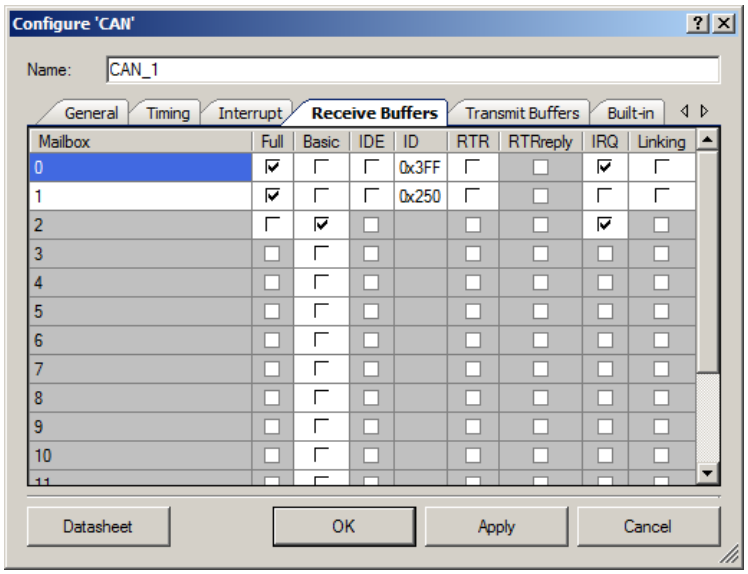
マクロ識別子 CAN_1_RX*x*_FUNC_ENABLE により、関数がコンパイルされるかどうかを定義します。定義は *CAN_1.h* プロジェクトファイルにリストされています。

- メッセージ受信の割り込みが発生すると、CAN_MsgRXIsr() 関数が呼び出されます。この関数はすべての受信メールボックスを巡回し、それぞれの「メッセージ可能フラグ」(MsgAv – Read: 0 使用できる新しいメッセージなし、1 使用できる新しいメッセージあり) を確認します。さらに「割り込みイネーブル」(Receive Interrupt Enable: 0 割り込み生成が無効、1 割り込み生成が有効) を確認し、CAN メッセージが正常に受信できるようにします。
- **Message Receive** 割り込みが有効の場合、メッセージを受信すると CAN_ReceiveMsg*X*() 関数が呼び出されます。ここで *X* は Full(完全版) CAN メールボックス番号またはユーザ定義名を示します。

- すべての割り込みベースの Basic(基本版) CAN メールボックスで CAN_ReceiveMsg(uint8 rxMailbox) 関数が呼び出されます。ここで rxMailbox パラメータはメッセージを受信したメールボックスの番号を示します。

受信バッファ構成

以下は、受信メッセージ API の使用を説明するための一例です。



CAN_1.h file:

```
...
#define CAN_1_RX0_FUNC_ENABLE 1
#define CAN_1_RX1_FUNC_ENABLE 0
#define CAN_1_RX2_FUNC_ENABLE 1
...
```

CAN_1_TX_RX_func.c file:

```
#if(CAN_1_RX0_FUNC_ENABLE)
void CAN_1_ReceiveMsg0(void)
{
    /* `#START MESSAGE_0_RECEIVED` */


    /* `#END` */

    CAN_1_RX[0].rxcmd.byte[0] |= CAN_1_RX_ACK_MSG;

}
#endif

#if(CAN_1_RX1_FUNC_ENABLE)
void CAN_ReceiveMsg1(void)
```

```
{
    /* `#START MESSAGE_1_RECEIVED` */

    /* `#END` */
CAN_1_RX[1].rxcmd.byte[0] |= CAN_1_RX_ACK_MSG;
}
#endif
```

以下の関数が CAN Receive Message 2 で呼び出され Basic CAN として割り込みイネーブルに設定されます。

```
void CAN_ReceiveMsg(uint8 rxMailbox)
{
    if (CAN_1_RX[rxMailbox].rxcmd.byte[0] & CAN_1_RX_ACK_MSG)
    {
        /* `#START MESSAGE_BASIC_RECEIVED` */


        /* `#END` */
        CAN_1_RX[rxMailbox].rxcmd.byte[0] |= CAN_1_RX_ACK_MSG;
    }
}
```

CAN_1_INT.c file

```
void CAN_1_MsgRXIsr(void)
{
…
/* RX Full mailboxes handler */
    switch(i)
    {
        case 0 : CAN_ReceiveMsg0();
      break;
        case 1 : CAN_ReceiveMsg1();
      break;
        default:
      break;
    }
…
}
```

リンクが実装されている場合、メールボックスに IRQ フラグが設定され、条件付きコンパイルが適用されます。すべての受信関数は、Message Available (MsgAv) フラグをクリアすることでメッセージを確認します。

ID の範囲を受入れるように AMR および ACR を設定する方法

以下は ACR/AMR レジスタの説明です。

```
[31:3] - Identifier(ID[31:21] - identifier when IDE = 0, ID[31:3] - identifier when
IDE = 1), [2] - IDE, [1] - RTR, [0] - N/A;
```

受入れマスクレジスタ (AMR) は、受入れコードレジスタ (ACR) に対して着信ビットを確認するかどうかを定義します。

AMR:　‘0’　それぞれの ACR に対して着信ビットを確認します。着信ビットが対応する ACR ビットに一致しない場合、メッセージは受容されません。

‘1’　着信ビットを確認しません。

たとえば、ID 範囲 0x180–187、IDE = 0 (クリア)、RTR = 0 (クリア)、メールボックス 5 を受信するメールボックスを設定するには、以下の操作を追加で実行します。

ID の低い範囲 (例：0x180) および IDE と RTR をそれぞれ取ります。この ID、IDE、および RTR の値では、AMR および ACR レジスタが以下の値に設定されます。

- ACR[31:21] = 0x180　　AMR[31:21] = 0x0
- ACR[20:3] = 0x0　　AMR[20:3] = 0x3FFFF (全部 1)
- ACR[2] = 0　　AMR[2] = 0
- ACR[1] = 0　　AMR[1] = 0
- ACR[0] = 0　　AMR[0] = 0

範囲 ID の共通部分の定義 (11 ビット):

- 0x180 = 0`b001 1000 0000
- 0x187 = 0`b001 1000 0111
- Mask = 0`b001 1000 0XXX
- ACR[31:21] = 0’b000 0000 0111

“XXX” の代わりに幾つか 1 を置き、AMR レジスタの共通ビットの代わりにヌルを置いてください。これにより、次の値は以下となります。

- ACR[31:21] = 0x180　　AMR[31:21] = 0x7;
- ACR[20:3] = 0x0　　AMR[20:3] = 0x3FFFF - 全部 1;
- ACR[2] = 0　　AMR[2] = 0
- ACR[1] = 0　　AMR[1] = 0
- ACR[0] = 0　　AMR[0] = 0

CAN_RXRegisterInit() 関数を使用してメールボックス番号 5 の AMR レジスタに書き込みます。

```
uint8 result = FAIL;
uint16 temp_amr;
uint16 temp_acr;

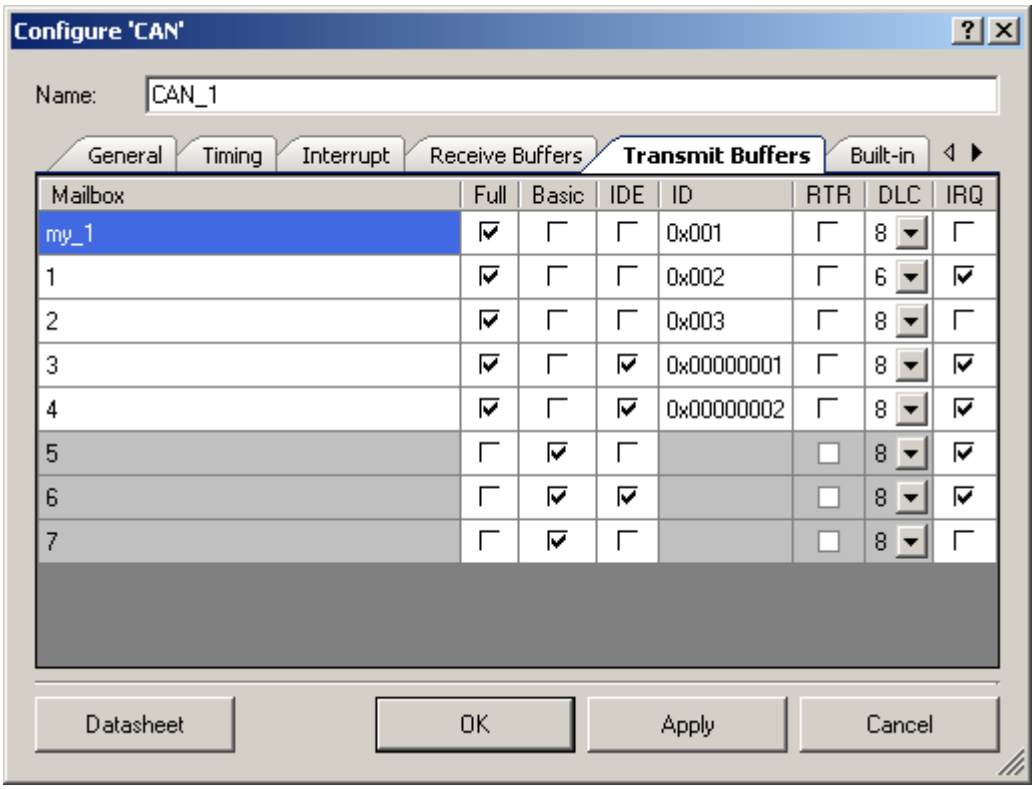
/* Upper address value, so address is shifted */
temp_amr = (0x7 << 21);       /* obtain necessary value to put in AMR */
```

```
temp_acr = (0x180 << 21);      /* obtain necessary value to put in ACR */

if (CAN_RXRegisterInit(&CAN_1_RX[5].rxamr, temp_amr))
{
    if ((CAN_RXRegisterInit(&CAN_1_RX[5].rxacr, temp_acr))
    result = SUCCESS;
}
return result;  /* error - if return no zero value; */
```

AMRおよびACR構成の詳細情報は、『*PSoC® 3 および PSoC 5® Technical Reference Manual*』のコントローラエリアネットワーク (CAN) の章を参照してください。

Transmit Buffers タブ



Transmit Buffers タブには以下の設定が含まれます。

Mailbox(メールボックス)

完全版メールボックスでは、**Mailbox** フィールドは編集可能であり、固有のメールボックス名を入力できます。各メールボックスの管理用に提供される関数には、メールボックス文字列を付記しています。受入れられる文字: A–Z、a–z、0–9、および _ 。正しくない名前を入力すると、**Mailbox** フィールドがデフォルト値に戻ります。

Full(完全版)

Full(完全版)を選択すると、**Mailbox**、**IDE**、**ID**、**RTR**、**RTRreply**、**IRQ**、および **Linking** フィールドを変更できます。フィールドは以下のデフォルト値で初期化されます。

- **Mailbox** = 数値 0 ～ 7
- **IDE** = クリア
- **ID** = 0x01
- **RTR** = クリア
- **DLC** = 8
- **IRQ** = クリア

Basic(基本版)

デフォルトでは、すべてのメールボックスに **Basic** (基本版)チェックボックスが選択されています。**Basic** が選択されていると、オプション **ID**、**RTR**、および **DLC** は使用できません。**Basic** が選択されている場合、必要な CAN メッセージフィールドにコードを使用して入力します。フィールドは以下のデフォルト値で初期化されます。

- **IDE** = クリア
- **ID** = なし (使用不可)
- **RTR** = クリア (使用不可)
- **DLC** = 8 (使用不可)
- **IRQ** = クリア

IDE

IDE チェックボックスがクリアされた場合、識別子を 11 ビットより大きくすることはできません (0x001 ～ 0x7FE)。IDE が選択されている場合、29 ビットの識別子が使用できます (0x00000001 ～ 0x1FFFFFFE)。識別子 0x000 または 0x7FF (11 ビット) あるいは 0x1FFFFFFF (29 ビット) は選択できません。

ID

メッセージの識別子。

RTR

メッセージは、返信リクエストのメッセージです。

DLC

メッセージに含まれるバイト数。

IRQ

IRQ ビットは **Message Transmitted** (**Interrupt** タブ) により異なります。

Message Transmitted チェックボックスがクリアされている場合、**IRQ** を選択すると、メッセージが表示されます。**グローバル “Message Transmitted Interrupt” がディスエーブルになっています。イネーブルにしますか?**

- **Yes** – **Interrupt** タブの **IRQ** および **Message Transmitted Interrupt** チェックボックスを選択します。
- **No** または **Cancel** – **IRQ** を選択します。**Interrupt** タブの **Message Transmitted Interrupt** チェックボックスはクリアされたままになります。

CAN TX 関数

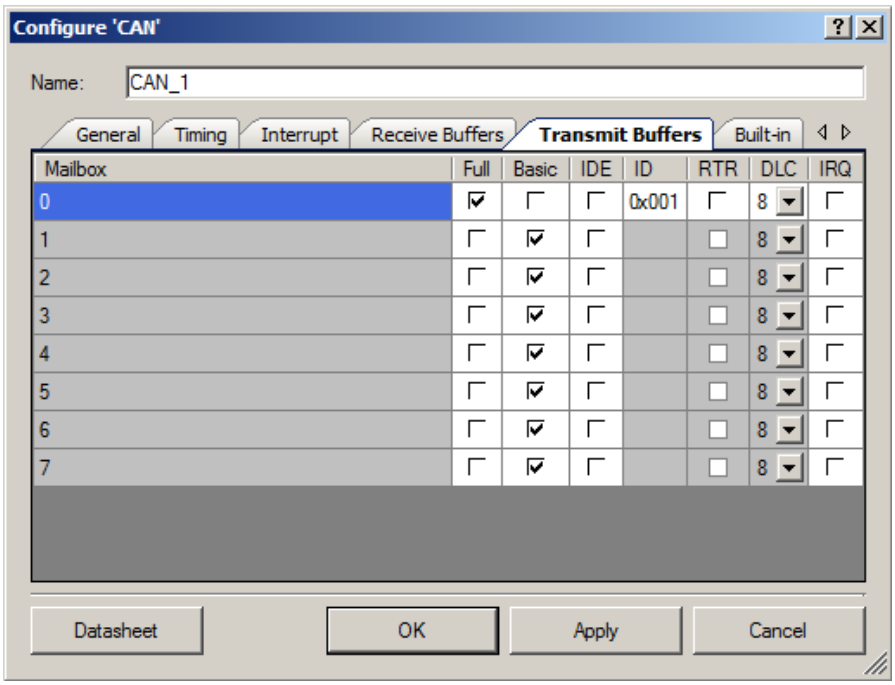
すべての Full(完全版) TX メールボックスには事前定義された API があります。関数のリストは *CAN_1_TX_RX_func.c* プロジェクトファイルにあります。これらの関数は、送信メールボックス設定により、条件付きでコンパイルされます。Full に定義されているメールボックスのみ、それぞれの関数がコンパイルされます。

マクロ識別子 CAN_1_TX*x*_FUNC_ENABLE により、関数がコンパイルされるかどうかを定義します。定義は *CAN_1.h* プロジェクトファイルにリストされています。

CAN_SendMsg*X*() 関数が、Full に設定されたすべての TX メールボックスに提供されます。ここで *X* は Full CAN メールボックス番号またはユーザ定義名を示します。

送信バッファ構成

以下は、送信 API の使用を説明するための一例です。



CAN_1.h file

```
#define CAN_1_TX0_FUNC_ENABLE 1
#define CAN_1_TX1_FUNC_ENABLE 0
```

CAN_1_TX_RX_func.c file

```
#if(CAN_1_TX0_FUNC_ENABLE)
uint8 CAN_1_SendMsg0(void)
{
uint8 result = SUCCESS;
    if (CAN_1_TX[0u].txcmd.byte[0u] & CAN_1_TX_REQUEST_PENDING) ==
        CAN_1_TX_REQUEST_PENDING)
    {
        result = CAN_FAIL;
    }
    else
    {
        /* `#START MESSAGE_0_TRANSMITTED` */

        /* `#END` */

        CY_SET_REG32((reg32 *) & CAN_1_TX[0u] |= txcmd CAN_SEND_MESSAGE);
    }

return result;
}
#endif
```

すべての Basic（基本版) Transmit メールボックス用に提供される共通の関数：

```
uint8 CAN_SendMsg(CAN_TX_MSG *message)
```

CAN 送信メッセージで必要なデータのアセンブリに使用するアプリケーション用に汎用構造体が定義されます。

- ID – ID スロットに以下が含まれる場合の制限:
  - 標準メッセージ (IDE = 0) では、識別子は 11 ビットに制限されます (0x001 ～ 0x7FE)。
  - 拡張メッセージ (IDE = 1) では、識別子は 29 ビットに制限されます (0x00000001 ～ 0x1FFFFFFE)。
- RTR (0 – 標準メッセージ、1 – 0xFF: メッセージ中に RTR ビット設定)
- IDE (0 – 標準メッセージ、1 – 0xFF: 拡張メッセージ)
- DLC (データのバイト数定義 0 ～ 8、9 ～ 0xFF は 8 と同等)
- IRQ (0 – IRQ イネーブル, 1 – 0xFF: IRQ ディスエーブル)
- DATA_BYTES (送信データを示す 8 バイトの構造体へのポインタ)

呼び出されると、CAN_SendMsg() 関数が、Basic CAN 専用の送信メッセージのメールボックスを巡回し、使用可能な最初のメールボックスを検索します。

- 空き容量のある Basic CAN メールボックスが検出されると、CAN_TX_MSG を通過したデータが適切な CAN 送信メールボックスにコピーされます。メッセージが送信キューに置かれると、アプリケーションに “SUCCESS” の表示が返されます。
- 空き容量のある Basic メールボックスが検出されない場合、関数によりあと 3 回繰り返し試行されます。3 回の試行に失敗すると、アプリケーションに “FAIL” の表示が返されます。

CAN_TX_MSG 構造体には、メッセージ送信に必要なすべての情報が含まれます。

```
typedef struct _CAN_TX_MSG
{
    uint32 id;
    uint8 rtr;
    uint8 ide;
    uint8 dlc;
    uint8 irq;
    CAN_DATA_BYTES_MSG *msg;
} CAN_TX_MSG;
```

CAN_DATA_BYTES 構造体には、メッセージに 8 バイトのデータが含まれます。

```
typedef struct CAN_DATA_BYTES_MSG
{
    uint8 byte[8u];
} CAN_DATA_BYTES_MSG;
```

Clock Select（クロック選択）

CAN コンポーネントを BUS_CLK クロック信号に接続できます。標準 CAN ボーレートを最大 1 Mbps まで対応するために、最低でも 8MHz が必要です。

配置

CAN コンポーネントは固定関数 CAN ブロックに配置されます。

リソース

説明	デジタルリソース		API メモリ(バイト)		ピン(外部入出力ごと)
	CAN IP Block	割り込み	フラッシュ	RAM	
tx_en と外部割込み信号がディスエーブルの場合	1*	1	4156	18	2
tx_en 信号が有効、外部割込み信号はディスエーブルの場合	1*	1	4156	18	3
tx_en および外部割り込み信号がイネーブルの場合	1*	1	4156	18	4

*CAN コンポーネントは、シリコンで専用の CAN ハードウェアブロックを使用します。

アプリケーション・プログラミング・インタフェース

アプリケーション・プログラミング・インターフェース(API)ルーチンにより、ソフトウェアを使用してコンポーネントを設定できます。次の表は、各関数へのインターフェースとその説明を示しています。続くセクションでは、各関数について詳しく説明します。

デフォルトでは、PSoC Creator は、与えられたユーザの回路図に最初に配置した制御レジスタ・インスタンス名として「CAN_1」を割り当てます。インスタンス名は、識別子の構文ルールに従った固有の値に変更できます。インスタンス名は、すべてのグローバル関数名、変数名、定数名のプリフィックスになります。読みやすいように、下表では「CAN」というインスタンス名を使用しています。

関数	説明
CAN_Start()	initVar 変数を設定し、CAN_Init() 関数を呼び出し、それからCAN_Enable() 関数を呼び出します。
CAN_Stop()	CAN コンポーネントをディスエーブルにします。
CAN_GlobalIntEnable()	CAN コンポーネントからのグローバル割り込みをイネーブルにします。
CAN_GlobalIntDisable()	CAN コンポーネントからのグローバル割り込みをディスエーブルにします。
CAN_SetPreScaler()	BUS_CLK からタイムクォンタム(最小時間単位)を生成するプリスケーラを設定します。
CAN_SetArbiter()	送信バッファ用のアービトレーションの種類を設定します。
CAN_SetTsegSample()	次の設定をします:Time segment 1、Time segment 2、Synchronization Jump Width(同期ジャンプ幅)、Sampling Mode(サンプリングモード)。
CAN_SetRestartType()	リセットの種類を設定します。
CAN_SetEdgeMode()	エッジモードを設定します。

関数	説明
CAN_RXRegisterInit()	受信 CAN レジスタにのみ書き込みます。
CAN_SetOpMode()	動作モードを設定します。
CAN_GetTXErrorflag()	送信エラーの数が 0x60 を超えているかどうかを示すフラグを返します。
CAN_GetRXErrorflag()	受信エラーの数が 0x60 を超えているかどうかを示すフラグを返します。
CAN_GetTXErrorCount()	送信エラーの数を返します。
CAN_GetRXErrorCount()	受信エラーの数を返します。
CAN_GetErrorState()	CAN コンポーネントのエラーステータスを返します。
CAN_SetIrqMask()	特定の割り込みソースをイネーブルまたはディスエーブルに設定します。
CAN_ArbLostIsr()	Arbitration Lost 割り込みフラグをクリアします。
CAN_OvrLdErrrorIsr()	Overload Error 割り込みフラグをクリアします。
CAN_BitErrorIsr()	Bit Error 割り込みフラグをクリアします。
CAN_BitStuffErrorIsr()	Bit Stuff Error 割り込みフラグをクリアします。
CAN_AckErrorIsr()	Acknowledge Error 割り込みフラグをクリアします。
CAN_MsgErrorIsr()	Form Error 割り込みフラグをクリアします。
CAN_CrcErrorIsr()	CRC Error 割り込みフラグをクリアします。
CAN_BusOffIsr()	Bus Off（バス オフ）割り込みフラグをクリアします。CAN コンポーネントを停止モードにします。
CAN_MsgLostIsr()	Message Lost（メッセージロスト）割り込みフラグをクリアします。
CAN_MsgTXIsr()	Transmit Message（メッセージ送信）割り込みフラグをクリアします。
CAN_MsgRXIsr()	Receive Message（メッセージ受信）割り込みフラグをクリアし、Basic（基本版）およびFull（完全版）割り込みベースのメールボックスに適切なハンドラを呼び出します。
CAN_RxBufConfig()	特定のメールボックスのすべての受信レジスタを設定します。
CAN_TxBufConfig()	特定のメールボックスのすべての送信レジスタを設定します。
CAN_SendMsg()	Basic（基本版）メールボックスの 1 つから、メッセージを送信します。
CAN_SendMsg0-7()	メールボックス 0 ~ 7 にアービトレーション待ちの未送信メッセージがないか確認します。
CAN_TxCancel()	送信キューに入っているメッセージの送信を取消します。
CAN_ReceiveMsg0-15()	新しいメッセージの受信を確認します。
CAN_ReceiveMsg()	特定のReceive Message（メッセージ受信)割り込みフラグをクリアします。
CAN_Sleep()	CAN コンポーネントがスリープモードに入る準備をします
CAN_Wakeup()	CAN コンポーネントのウェイクアップの準備をします

関数	説明
CAN_Init()	Configure ダイアログ設定に従って、CAN コンポーネントを初期化または復元します。
CAN_Enable()	CAN コンポーネントをイネーブルにします。
CAN_SaveConfig()	現在の設定を保存します。
CAN_RestoreConfig()	設定を復元します。

関数の実行状態を返す値で、0 は “SUCCESS” 、1 は“FAIL” 、2 は “OUT_OF_RANGE” です。

グローバル変数

変数	説明
CAN_initVar	CAN が初期化されているかどうかを示します。変数は 0 に初期化され、CAN_Start() が初めて呼び出されたときに 1 に設定されます。これにより、CAN_Start() ルーチンへの最初の呼び出し後、再初期化することなくコンポーネントを再起動することができます。 コンポーネントの再初期化が必要な場合は、CAN_Init() 関数を CAN_Start() または CAN_Enable() 関数の前に呼び出すことができます。

uint8 CAN_Start(void)

説明: この関数は、initVar 変数を設定し、CAN_Init() 関数を呼び出し、それから CAN_Enable() 関数を呼び出します。この関数は、CAN コンポーネントを実行モードに切り替え、ポーリング メールボックスが使用できる場合はカウントを開始します。

パラメータ: なし

戻り値: uint8: レジスタが書き込まれ、検証されているかを示します。

副作用: initVar 変数がすでに設定されている場合、この関数は CAN_Enable() 関数のみを呼び出します。

uint8 CAN_Stop(void)

説明: この関数は、CAN コンポーネントを停止モードに切り替え、ポーリング メールボックスが使用できる場合は、カウンタを停止します。

パラメータ: なし

戻り値: uint8: レジスタが書き込まれ、検証されているかを示します。

副作用: なし

uint8 CAN_GlobalIntEnable(void)

説明: この関数は、CAN コンポーネントからのグローバル割り込みをイネーブルにします。

パラメータ: なし

戻り値: uint8: レジスタが書き込まれ、検証されているかを示します。

副作用: なし

uint8 CAN_GlobalIntDisable(void)

説明: この関数は、CAN コンポーネントからのグローバル割り込みをディスエーブルにします。

パラメータ: なし

戻り値: uint8: レジスタが書き込まれ、検証されているかを示します。

副作用: なし

uint8 CAN_SetPreScaler(uint16 bitrate)

説明: この関数は、BUS_CLK からのタイムクォンタム（最小時間単位）を生成するプリスケーラを設定します。0x0 ~ 0x7FFF の値が有効です。

パラメータ: uint16 bitrate: プリスケーラの値

戻り値: uint8: レジスタが書き込まれ、検証されているかを示します。

副作用: なし

uint8 CAN_SetArbiter(uint8 arbiter)

説明: この関数は、送信バッファのアービトレーションのタイプを設定します。アービタのタイプは、Round Robin (総当り) と Fixed (固定) 優先順位です。値 0 または 1 が有効です。

パラメータ: uint8 arbiter: アービタのタイプ

戻り値: uint8: レジスタが書き込まれ、検証されているかを示します。

副作用: なし

uint8 CAN_SetTsegSample(uint8 cfgTseg1, uint8 cfgTseg2, uint8 sjw, uint8 sm)

説明: この関数は以下を設定します: Time segment 1、Time segment 2、Synchronization Jump Width(同期ジャンプ幅)、Sampling Mode(サンプリングモード)。

パラメータ: uint8 cfgTseg1: 時間セグメント1、0x2 ~ 0xF の値が有効

uint8 cfgTseg2: 時間セグメント2、0x1 ~ 0x7 の値が有効

uint8 sjw: 同期ジャンプ幅、0x0 ~ 0x3 の値が有効

uint8 sm: サンプリング モード、1 つまたは 3 つのサンプリング点を使用

戻り値: uint8: レジスタが書き込まれ、検証されているかを示します。

副作用: なし

uint8 CAN_SetRestartType(uint8 reset)

説明: この関数はリセットタイプを設定します。リセットのタイプは Automatic または Manual です。Manual リセットが推奨設定です。値0または1が有効です。

パラメータ: uint8 reset: リセットタイプ

戻り値: uint8: レジスタが書き込まれ、検証されているかを示します。

副作用: なし

uint8 CAN_SetEdgeMode(uint8 edge)

説明: この関数はエッジモードを設定します。モードは 'R' ~ 'D' (Recessive (劣性) から Dominant (優性) 片エッジ) と両エッジが使用されます。値0または1が有効です。

パラメータ: uint8 edge: エッジ モード

戻り値: uint8: レジスタが書き込まれ、検証されているかを示します。

副作用: なし

uint8 CAN_RXRegisterInit(uint32 *regAddr, uint32 config)

説明: この関数は CAN 受信レジスタのみに書き込みます。

パラメータ: uint32 * regAddr: CAN 受信レジスタへのポインタ

uint32 設定: レジスタに書き込まれる値

戻り値: uint8: レジスタが書き込まれ、検証されているかを示します。

副作用: なし

uint8 CAN_SetOpMode(uint8 opMode)

説明: この関数は操作モードを設定します。操作モードは「Active」または「Listen Only」です。値 0 または 1 が有効です。

パラメータ: uint8 opMode: 操作モード値

戻り値: uint8: レジスタが書き込まれ、検証されているかを示します。

副作用: なし

uint8 CAN_GetTXErrorflag(void)

説明: この関数は、送信エラー数が 0x60 を超えたかどうかを示すフラグを返します。

パラメータ: なし

戻り値: uint8: 送信エラーの数が 0x60 を超えたかどうかを示します。

副作用: なし

uint8 CAN_GetRXErrorflag(void)

説明: この関数は、受信エラー数が 0x60 を超えたかどうかを示すフラグを返します。

パラメータ: なし

戻り値: uint8: 受信エラーの数が 0x60 を超えたかどうかを示します。

副作用: なし

uint8 CAN_GetTXErrorCount(void)

説明: この関数は送信エラーの数を返します。

パラメータ: なし

戻り値: uint8: 送信エラーの数

副作用: なし

uint8 CAN_GetRXErrorCount(void)

説明: この関数は、受信エラーの数を返します。

パラメータ: なし

戻り値: uint8: 受信エラー数

副作用: なし

uint8 CAN_GetErrorState(void)

説明: この関数は、CAN コンポーネントのエラー ステータスを返します。

パラメータ: なし

戻り値: uint8: エラーステータス

副作用: なし

uint8 CAN_SetIrqMask(uint16 mask)

説明: この関数は特定の割り込みソースをイネーブル/ディスエーブルにします。割り込みマスクは、CAN 割り込みイネーブルのレジスタに直接書き込みます。

パラメータ: uint8 request. 割り込みイネーブルまたはディスエーブル要求。割り込みソース当たり 1 ビット

戻り値: uint8: レジスタが書き込まれ、検証されているかを示します。

副作用: なし

void CAN_ArbLostIsr(void)

説明: この関数は、Arbitration Lost(アービトレーションロスト)割り込みのエントリ ポイントです。Arbitration Lost(アービトレーションロスト)割り込みフラグをクリアします。Arbitration Lost(アービトレーションロスト)割り込みパラメータがイネーブルな場合のみ生成されます。

パラメータ: なし

戻り値: なし

副作用: なし

void CAN_OvrLdErrrorIsr(void)

説明: この関数は、Overload Error(過負荷エラー)割り込みのエントリ ポイントです。Overload Error(過負荷エラー)割り込みフラグをクリアします。Overload Error(過負荷エラー)割り込みパラメータがイネーブルな場合のみ生成されます。

パラメータ: なし

戻り値: なし

副作用: なし

void CAN_BitErrorIsr(void)

説明: この関数は、Bit Error(ビットエラー)割り込みのエントリ ポイントです。Bit Error(ビットエラー)割り込みフラグをクリアします。Bit Error(ビットエラー)割り込みパラメータがイネーブルな場合のみ生成されます。

パラメータ: なし

戻り値: なし

副作用: なし

void CAN_BitStuffErrorIsr(void)

説明: この関数は、Bit Stuff Error(ビットスタッフエラー)割り込みのエントリ ポイントです。Bit Stuff Error(ビットスタッフエラー)割り込みフラグをクリアします。Bit Stuff Error(ビットスタッフエラー)割り込みパラメータがイネーブルな場合のみ生成されます。

パラメータ: なし

戻り値: なし

副作用: なし

void CAN_AckErrorIsr(void)

説明: この関数は、Acknowledge Error（確認エラー）割り込みのエントリ ポイントです。Acknowledge Error（確認エラー）割り込みフラグをクリアします。Acknowledge Error（確認エラー）割り込みパラメータがイネーブルな場合のみ生成されます。

パラメータ: なし

戻り値: なし

副作用: なし

void CAN_MsgErrorIsr(void)

説明: この関数は、Form Error（フォームエラー）割り込みのエントリ ポイントです。Form Error（フォームエラー）割り込みフラグをクリアします。Form Error（フォームエラー）割り込みパラメータがイネーブルな場合のみ生成されます。

パラメータ: なし

戻り値: なし

副作用: なし

void CAN_CrcErrorIsr(void)

説明: この関数は、CRC Error（CRCエラー）割り込みのエントリポイントです。CRC Error（CRCエラー）割り込みフラグをクリアします。CRC Error（CRC エラー）割り込みパラメータがイネーブルな場合のみ生成されます。

パラメータ: なし

戻り値: なし

副作用: なし

void CAN_BusOffIsr(void)

説明: この関数は、Bus Off（バスオフ）割り込みのエントリ ポイントです。CAN コンポーネントを停止モードに切り替えます。Bus Off（バスオフ）割り込みパラメータがイネーブルな場合のみ生成されます。この割り込みをイネーブルにすることをお勧めします。

パラメータ: なし

戻り値: なし

副作用: CAN コンポーネントの動作を停止します。

void CAN_MsgLostIsr(void)

説明: この関数は、Message Lost（メッセージロスト）割り込みのエントリポイントです。Message Lost（メッセージロスト）割り込みフラグをクリアします。Message Lost（メッセージロスト）割り込みパラメータがイネーブルな場合のみ生成されます。

パラメータ: なし

戻り値: なし

副作用: なし

void CAN_MsgTXIsr(void)

説明: この関数は、Transmit Message（送信メッセージ）割り込みのエントリポイントです。Transmit Message（送信メッセージ）割り込みフラグをクリアします。Transmit Message（送信メッセージ）割り込みパラメータがイネーブルな場合のみ生成されます。

パラメータ: なし

戻り値: なし

副作用: なし

void CAN_MsgRXIsr(void)

説明: この関数は、Receive Message（受信メッセージ）割り込みのエントリ ポイントです。Receive Message（受信メッセージ）割り込みフラグをクリアし、Basic（基本版）またはFull（完全版）割り込みベースのメールボックスに適切なハンドラを呼び出します。Receive Message（受信メッセージ）割り込みパラメータがイネーブルな場合のみ生成されます。この割り込みをイネーブルにすることをお勧めします。

パラメータ: なし

戻り値: なし

副作用: なし

uint8 CAN_RxBufConfig(CAN_RX_CFG *rxConfig)

説明: この関数は、特定のメールボックスのすべての受信レジスタを設定します。メールボックス番号には CAN_RX_CFG 構造体が含まれています。

パラメータ: CAN_RX_CFG * rxConfig: 特定のメールボックスの受信レジスタをすべて設定するために必要なすべての値が含まれる構造体へのポインタ

戻り値: uint8: 特定の受信レジスタ設定が承認されたか拒否されたかを示します。

副作用: なし

uint8 CAN_TxBufConfig(CAN_TX_CFG *txConfig)

説明: この関数は特定のメールボックスの送信レジスタをすべて設定します。メールボックス番号には CAN_TX_CFG 構造体が含まれています。

パラメータ: CAN_TX_CFG * txConfig: 特定のメールボックスの送信レジスタをすべて設定するために必要なすべての値が含まれている構造体へのポインタ

戻り値: uint8: 特定の送信レジスタ設定が承認されたか拒否されたかを示します。

副作用: なし

uint8 CAN_SendMsg(CANTXMsg *message)

説明: この関数は、Basic (基本版)CANメールボックスとして指定されている送信メッセージ メールボックスを検索します。最初に空いているメールボックスを検索し、そこから送信します。空いているBasic（基本版）メールボックスがない場合は、最高 3 回まで検索を続けます。

パラメータ: CAN_TX_MSG * message: メッセージの送信に必要なデータを含む構造体へのポインタ

戻り値: uint8: メッセージが送信されたかを示します。

副作用: なし

uint8 CAN_SendMsg0-7(void)

説明: この関数は、Transmit Message 0-7（送信メッセージ 0 ~ 7）のエントリ ポイントです。メールボックス 07 に、アービトレーションを待つ未送信のメッセージがすでに存在しないか検索します。ある場合は、メッセージの送信を開始します。この関数は、Full（完全版）と指定されている送信メールボックスのためにのみ生成されます。

パラメータ: なし

戻り値: uint8: メッセージが送信されたかを示します。

副作用: なし

void CAN_TxCancel(uint8 bufferId)

説明: この関数は、送信キューにあるメッセージの送信を取り消します。0 ~ 15 の値が有効です。

パラメータ: uint8 bufferId: Tx メールボックスの番号

戻り値: なし

副作用: なし

void CAN_ReceiveMsg0-15(void)

説明: この関数は、Receive Message 0-15（受信メッセージ 0 ~ 15 ）割り込みのエントリ ポイントです。Receive Message 0-15（受信メッセージ 0 ~ 15 ） 割り込みフラグをクリアします。Full（完全版）割り込みベースと指定された受信メールボックス用にのみ生成されます。

パラメータ: なし

戻り値: なし

副作用: なし

void CAN_ReceiveMsg(uint8 rxMailbox)

説明: この関数は、Basic（基本版）メールボックスのReceive Message（メッセージ受信）割り込みのエントリ ポイントです。特定のReceive Message（メッセージ受信）割り込みフラグをクリアします。Receive（受信）メールボックスがBasic（基本版）と指定されている場合のみ生成されます。

パラメータ: uint8 rxMailbox: Receive Message（メッセージ受信）割り込みをトリガするメールボックス番号

戻り値: なし

副作用: なし

void CAN_Sleep(void)

説明: これは、コンポーネントをスリープさせる準備をするのに推奨されるルーチンです。CAN_Sleep() ルーチンは現在のコンポーネントの状態を保存します。次に、CAN_Stop() 関数と CAN_SaveConfig() 関数を呼び出してハードウェア設定を保存します。

CyPmSleep() または CyPmHibernate() 関数を呼び出す前に CAN_Sleep() 関数を呼び出します。電源管理関数については、*『PSoC Creator システム・リファレンスガイド』*を参照してください。

パラメータ: なし

戻り値: なし

副作用: なし

void CAN_Wakeup(void)

説明: これは、CAN_Sleep() が呼び出されたときの状態にコンポーネントを復元するのに推奨されるルーチンです。CAN_Wakeup() 関数は CAN_RestoreConfig() 関数を呼び出して設定を復元します。CAN_Sleep() 関数が呼び出される前にコンポーネントがイネーブルにされた場合は、CAN_Wakeup() 関数はコンポーネントの再有効化も行います。

パラメータ: なし

戻り値: なし

副作用: CAN_Sleep() または CAN_SaveConfig() 関数を呼び出す前に CAN_Wakeup() 関数を呼び出すと、予期しない動作を引き起こすことがあります。

uint8 CAN_Init(void)

説明: この関数はカスタマイザの [Configure] (設定) ダイアログの設定に従って、コンポーネントを初期化または復元します。CAN_Start() ルーチンが CAN_Init() 関数を呼び出すので、この関数を呼び出す必要はありません。これはコンポーネントの動作を開始する際に推奨される方法です。

パラメータ: なし

戻り値: uint8: 設定が承認されたか拒否されたかを示します。

副作用: すべてのレジスタが初期値にリセットされます。これにより、次の例外を除いて、コンポーネントが再初期化されます。メールボックスは例外で、データは消去されません。

CAN コアへの電源供給をイネーブルにします。

uint8 CAN_Enable(void)

説明: この関数はハードウェアを作動させ、コンポーネントの動作を開始します。CAN_Start() ルーチンが CAN_Enable() を呼び出すため、この関数を呼び出す必要はありません。これはコンポーネントの動作を開始する際に推奨される方法です。

パラメータ: なし

戻り値: なし

副作用: なし

void CAN_SaveConfig(void)

説明: この関数は、コンポーネントの設定と一時保持レジスタを保存します。この関数は、[Configure] ダイアログで定義されている、または該当する API で変更される、現在のコンポーネント・パラメータ値も保存します。この関数は CAN_Sleep() 関数によって呼び出されます。

パラメータ: なし

戻り値: なし

副作用: なし

void CAN_RestoreConfig(void)

説明: この関数は、コンポーネントの設定と一時保持レジスタを復元します。また、CAN_Sleep() 関数を呼び出す前のコンポーネント パラメータ値を復元します。

パラメータ: なし

戻り値: なし

副作用: CAN_Sleep() または CAN_SaveConfig() 関数を呼び出す前にこの関数を呼び出すと予期せぬ動作が引き起こされることがあります。以下のレジスタはデフォルト値に戻ります: CAN_INT_SR、CAN_INT_EN、CAN_CMD、CAN_CFG。

ファームウェア・ソースコードのサンプル

PSoC Creator は、[Find Example Project] ダイアログに多数のサンプルプロジェクトを提供しており、そこには回路図およびサンプル コードが含まれています。コンポーネント固有の例を見るには、[Component Catalog] または回路図に置いたコンポーネント インスタンスからダイアログを開きます。一般例については、[Start Page] または **[File]** メニューからダイアログを開きます。必要に応じてダイアログにある **Filter Options** を使用し、選択できるプロジェクトのリストを絞り込みます。

詳しくは、PSoC Creator ヘルプの「Find Example Project（サンプルプロジェクトを検索）」トピックを参照してください。

割り込みサービスルーチン

複数の CAN コンポーネント割り込みソースがあります:

- Arbitration Lost Detection（アービトレーションロスト検知） – メッセージの送信中にアービトレーションが失われました。
- Overload Error（過負荷エラー） – CAN コントローラが過負荷フレームを受信しました。

- Bit Error（ビット エラー） – CAN コントローラがビット エラーを検知しました。
- Bit Stuff Error（ビットスタッフエラー） – CAN コントローラが ビットスタッフエラーを検知しました。
- Acknowledge Error（確認エラー） – CAN コントローラが CAN メッセージ確認エラーを検知しました。
- Form Error（フォーム エラー） – CAN コントローラが CAN メッセージ フォーマット エラーを検知しました。
- CRC Error（CRC エラー） – CAN コントローラが CAN CRC エラーを検知しました。
- Bus Off（バス オフ） – CAN コントローラがバス オフ状態になりました
- Message Lost（メッセージロスト） – 新しいメッセージを受信しましたが保存する場所がありません。
- Transmit Message（メッセージ送信） – キューの中のメッセージを送信しました。
- Receive Message（メッセージ受信） – メッセージを受信しました。

これらの割り込みソースのすべてにエントリ ポイント (関数) があるため、コードを挿入できます。この関数は、カスタマイザの設定によって条件付きでコンパイルされます。

Receive Message（メッセージ受信）割り込みは Full（完全版）および Basic（基本版）メールボックスのために適切な関数を呼び出す特殊なハンドラーがあります。

割り込み出力のユースケース

次の例は、CAN コンポーネントのハードウェア割り込み出力ラインの使用例です:

割り込みイベントのハードウェア制御論理

ハードウェア割り込みラインは、CAN バスの負荷予想など、簡単なタスクを実行するために使用することができます。CAN コンポーネントカスタマイザーで Message Transmitted（メッセージ送信済み）と Message Received（メッセージ受信済み）割り込みを有効にし、割り込みラインをカウンタに接続することにより、特定の時間間隔においてバス上にあるメッセージ数を評価することができます。また、メッセージ率が特定の値を超えている場合、ハードウェアにおいて直接アクションをとることができます。

DMAによる割り込み出力とのインタラクション

CAN コンポーネントは内部において DMA 操作をサポートしませんが、DMA コンポーネントを外部割り込みラインに接続することができます（イネーブルの場合）。DMA の構成と操作はユーザの責任となります。また、CAN 割り込みを適切に取り扱うためには、いくつかのメンテナンスタスク（メッセージを確認し、割り込みフラグをクリアするなど）をコードで管理する必要があることを念頭においてください。

ハードウェア DMA トリガーを用いて、CPU でファームウェアを実行することなく、Message Received（メッセージ受信）割り込みが生じたとき、レジスタを処理し、データを転送することができます。これは RTR メッセージを取り扱うときにも役に立ちます。Message Transmitted（メッセージ送信済み）割り込みは、DMA 転送をトリガーす

るために使用することができ、CPU を介することなく、メッセージバッファに新しいデータを再度読み込むことができます。

カスタム外部割り込みサービスルーチン

カスタム外部 ISR を内部 ISR に追加あるいは代用として使用することができます。内部 ISR に加えて外部 ISR が使用されているとき、割り込みの優先順位をどの ISR が最初に実行されるべきか（内部または外部）設定することができ、そのため、内部 ISR コードの前か後にアクションを強制することができます。外部 ISR が内部 ISR に代わりに使用されるとき、CAN レジスタおよびイベントの適切な取扱いの全責任はユーザにあります。

割り込みサブシステムによる割り込み出力のインタラクション

CAN コンポーネント割り込み出力設定でできること:

- 外部割り込みラインをイネーブルまたはディスエーブルできます（カスタマイザーオプション）
- 内部 ISR をディスエーブルまたはバイパスできます（カスタマイザーオプション）
- 内部 ISR を完全にカスタマイズできます（カスタマイザーオプション）
- 該当するイベント割り込みがイネーブルの場合（カスタマイザーオプション）、内部 ISR の指定割り込み処理関数呼び出しをイネーブルまたはディスエーブルできます。個々の割り込み（メッセージ送信、メッセージ受信、受信バッファーがフル、バスがオフ状態など）を CAN コンポーネントカスタマイザーにおいてイネーブルまたはディスエーブルすることができます。イネーブルになると、該当する関数の呼び出しが内部 CAN_ISR で実行されます。これにより、このような関数の呼び出しをディスエーブル（削除）することができます。

外部割り込みラインはカスタマイザーでイネーブルの場合にのみ表示できます。

外部割り込みコンポーネントが接続された場合、外部割り込みコンポーネントは CAN_Start() API の一部として起動されず、そのルーチン外で起動する必要があります。

外部割り込みコンポーネントが接続され、内部 ISR がディスエーブルまたはバイパスされていない場合、2 つの割り込みコンポーネントは同じラインに接続されています。この場合、2 つの別々な割り込みコンポーネントがあり、これらが同じ割り込みイベントを取り扱います。これは特別なものであり、多くの場合、好ましくない状況です。

内部 ISR が（カスタマイザーオプションを使用し）ディスエーブルまたはバイパスされている場合、内部割り込みコンポーネントはビルドプロセス中削除されます。

内部割り込みルーチンにおいて、個々の割り込み関数の呼び出しをディスエーブルにする場合（割り込みイベントのイネーブルにはカスタマイザーオプションを使用）、CAN ブロック割り込みは（該当するイベントが生じたとき）トリガーしますが、内部 CAN_ISR ルーチンにおいて内部関数呼び出しは実行されません。使用例として、特定のイベント（メッセージ受信など）を、標準ユーザー関数呼び出し機能（DMA を通してなど）以外の異なるパスを通して取り扱いたい場合があります。

内部 ISR を（カスタマイザーオプションを使用して）完全にカスタマイズしたい場合、CAN_ISR 関数にはオプションの PSoC 3 ES1/ES2 ISR パッチ以外の関数呼び出しは含まれません。

機能の説明

完全な機能については、*PSoC® 3 およびPSoC® 5 テクニカルリファレンスマニュアル*のコントローラーエリアネットワーク(CAN)の章をご覧ください。

ブロックダイアグラムと設定

完全なブロック図および構成情報については、*PSoC® 3 およびPSoC® 5 テクニカルリファレンスマニュアル*のコントローラーエリアネットワーク(CAN)の章をご覧ください。

参考資料

1. *ISO-11898: 道路車両* -- コントローラーエリアネットワーク(CAN):
    - ❑ パート 1:データリンク層および物理的信号
    - ❑ パート 2:高速メディアアクセスユニット
    - ❑ パート 3:低速、フォールトトレラント、媒体依存のインターフェース
    - ❑ パート 4:タイムトリガ通信
    - ❑ パート 5:低電力モードの高速メディアアクセスユニット
2. *CAN 仕様バージョン 2 BOSCH*
3. *Inicore CANmodule-III-AHB データシート*

DC電気的特性とAC電気的特性

以下の値は、期待される性能を示しており、初期特性データを基にしています。

CANのDC仕様

パラメータ	説明	条件	Min	Typ	Max	単位
	ブロックの電流消費	500 kbps	--	--	285	µA
		1 Mbps	--	--	330	µA

CANのAC仕様

パラメータ	説明	条件	Min	Typ	Max	単位
	ビット レート	最低8 MHzクロック	--	--	1	Mbit

コンポーネントの変更

ここでは、過去のバージョンからコンポーネントに加えられた主な変更を示します。

現在のバージョン	変更の説明	変更の理由 / 影響
2.1	.cyre ファイルのすべての CAN API に CYREENTRANT キーワードを追加	すべての API が真に再入可能とは限りません。コンポーネント API ソースファイルに含まれているコメントは、どの関数が候補であるかを示しています。 この変更では、安全な方法（フラグにより並行コール禁止またはクリティカル セクションとして保護）で使用される再入可能ではない関数のコンパイラの警告をなくす必要があります。
2.0.a	タイミング図を置き換え、データシートに機能テキストを追加	
	データシートのマイナーな編集と更新	
2.0	コンポーネント記号に割り込み出力を追加	PSoC 3 CANコンポーネントのマーケティング要件文書を更新しました。
	ConnectExtInterruptLine、IntISRDisable、FullCastomIntISR、AdvancedInterruptTabパラメーターおよびISRヘルパーパラメーターをコンポーネント記号に追加	PSoC 3 CANコンポーネントのマーケティング要件文書MRDを更新しました。
	CAN_Init()関数の初めにストップ文を追加	CANが初期化されたときに停止することを確実にするために（追加しました。）
	割り込みハンドラー関数を更新 すべての場合、割り込みフラグはユーザーコードの前にクリア	同じ方法で割り込みフラグをクリアする(ように更新しました。)
	廃止された定義を削除	
1.50.a	データシートに特性データを追加	
	データシートテキストの編集	
1.50	スリープ/ウェークアップAPIを追加	これらのAPIは低電力モードのサポートを提供します。
	CAN_Init()とCAN_Enable() APIを追加	社内標準に準拠し、コンポーネントを起動せずに初期化、復元できるAPIを提供するため（追加しました）。

Copyright © 2005-2012 Cypress Semiconductor Corporation 本文書に記載される情報は、予告なく変更される場合があります。Cypress Semiconductor Corporation は、サイプレス製品に組み込まれた回路以外のいかなる回路を使用することに対しても一切の責任を負いません。特許又はその他の権限下で、ライセンスを譲渡又は暗示することもありません。サイプレス製品は、サイプレスとの書面による合意に基づくものでない限り、医療、生命維持、救命、重要な管理、又は安全の用途のために仕様することを保証するものではなく、また使用することを意図したものでもありません。さらにサイプレスは、誤動作や故障によって使用者に重大な傷害をもたらすことを合理的に予想される、生命維持システムの重要なコンポーネンツとしてサイプレス製品を使用することを許可していません。生命維持システムの用途にサイプレス製品を供することは、製造者がそのような使用におけるあらゆるリスクを負うことを意味し、その結果サイプレスはあらゆる責任を免除されることを意味します。

PSoC Designer™及び Programmable System-on-Chip™は、Cypress Semiconductor Corp.の商標、PSoC®は同社の登録商標です。本文書で言及するその他全ての商標又は登録商標は各社の所有物です。

全てのソースコード(ソフトウェア及び/又はファームウェア)は Cypress Semiconductor Corporation (以下「サイプレス」)が所有し、全世界(米国及びその他の国)の特許権保護、米国の著作権法並びに国際協定の条項により保護され、かつそれらに従います。サイプレスが本書面によるライセンシーに付与するライセンスは、個人的、非独占的かつ譲渡不能のライセンスであって、適用される契約で指定されたサイプレスの集積回路と併用されるライセンシーの製品のみをサポートするカスタムソフトウェア及び/又はカスタムファームウェアを作成する目的に限って、サイプレスのソースコードの派生著作物を複製、使用、変更、そして作成するためのライセンス、並びにサイプレスのソースコード及び派生著作物をコンパイルするためのライセンスです。上記で指定された場合を除き、サイプレスの書面による明示的な許可なくして本ソースコードを複製、変更、変換、コンパイル、又は表示することは全て禁止されます。

免責条項:サイプレスは、明示的又は黙示的を問わず、本資料に関するいかなる種類の保証も行いません。これには、商品性又は特定目的への適合性の黙示的な保証が含まれますが、これに限定されません。サイプレスは、本文書に記載される資料に対して今後予告なく変更を加える権利を留保します。サイプレスは、本文書に記載されるいかなる製品又は回路を適用又は使用したことによって生ずるいかなる責任も負いません。サイプレスは、誤動作や故障によって使用者に重大な傷害をもたらすことが合理的に予想される生命維持システムの重要なコンポーネンツとしてサイプレス製品を使用することを許可していません。生命維持システムの用途にサイプレス製品を供することは、製造者がそのような使用におけるあらゆるリスクを負うことを意味し、その結果サイプレスはあらゆる責任を免除されることを意味します。

ソフトウェアの使用は、適用されるサイプレスソフトウェアライセンス契約によって制限され、かつ制約される場合があります。