AN52701

PSoC 3 および PSoC 5LP - コントローラー エリア ネットワーク (CAN) 入門

著者: Ranjith M
関連プロジェクト: あり
関連製品ファミリ: CAN を内蔵したすべての PSoC 3 と PSoC 5LP 製品
関連サンプル コード: CE95282、CE95283
ソフトウェア バージョン: PSoC Creator™ 4.1

さらにサンプル コードをお求めでしょうか？以下のとおり対応いたします。

PSoC サンプル コードのリストにアクセスするには、サンプル コードのウェブページをご覧ください。PSoC ビデオ ライブラリはここからご覧ください。

本アプリケーション ノートでは、コントローラー エリア ネットワーク (CAN) の基本的な概念を紹介し、デモを通じて PSoC® 3 と PSoC 5LP を使用して CAN バス通信を実装する方法を示します。

目次

1 はじめに 2
1.1 CAN のメリット 2
1.2 PSoC の用途 2
2 CAN の基本 2
2.1 物理層 2
2.2 転送層 3
2.3 バス アービトレーション 4
2.4 CAN のエラー管理 5
3 PSoC における CAN 6
3.1 ハードウェア 6
3.2 CAN コンポーネント 7
3.3 PSoC Creator プロジェクト 8
3.4 ファームウェア 16
4 ハードウェアの実装 19
4.1 CAN アナライザの操作 19
5 サンプル プロジェクト 20
5.1 プロジェクト 1 と 2: シンプレックス通信 20
5.2 プロジェクト 3 と 4: CAN の RTR 機能 22
6 まとめ 24
7 関連リソース 24
Appendix A. データ フレーム 25
Appendix B. PSoC と TJA1050 CAN トランシーバとのインターフェース 26
改訂履歴 27
ワールドワイドな販売と設計サポート 28

1 はじめに

コントローラー エリア ネットワーク (CAN) は、1980 年代初頭に Robert Bosch GmbH によって開発されたシリアル通信プロトコルです。このプロトコルは、最初は中央制御なしでサブシステム間の通信のために車載用途向けに開発されました。CAN は、組み込みシステム (CANOpen) やファクトリー オートメーション (DeviceNet) などの分野でも採用されています。CAN は 2003 年に ISO によって標準化されました (ISO 11898-1:2003)。

本アプリケーション ノートでは、CAN プロトコルの基本的な概念を紹介し、PSoC® 3 と PSoC 5LP を使用して CAN バス通信を実装する方法をデモします。本書には 4 つの例が含まれています。プロジェクト 1 と 2 は、2 つの PSoC 間のシンプレックス通信を示します。プロジェクト 3 と 4 は一緒に、CAN の遠隔送信要求 (RTR) 機能をデモします。

本アプリケーション ノートは、ユーザーが PSoC Creator を使った PSoC 3 または PSoC 5LP のアプリケーションの開発に精通していることを前提としています。PSoC 3 または PSoC 5LP を初めて使用する場合、「AN54181 - Getting Started with PSoC 3」および「AN77759 - Getting Started with PSoC 5LP」にて概要を参照してください。PSoC Creator に慣れていない場合、PSoC Creator のホームページを参照してください。

サイプレスは車載用マイクロコントローラーのリーディング サプライヤーです。FM4 ファミリのマイクロコントローラーで CAN を使用する場合は、FM4 CAN のサンプル コードを参照してください。サイプレスは、PSoC 4 ファミリのデバイスで使用するための Arduino 互換の CAN シールドも提供しています。CAN シールドの詳細は、CY8CKIT-026 のウェブページを参照してください。CY8CKIT-026 シールドを使用して PSoC 4 CAN 通信をデモするサンプル プロジェクトは、PSoC 4 CAN サンプル コードのウェブページに掲載されています。

1.1 CAN のメリット

CAN ネットワークは、データ長が 8 バイトを超えないショート メッセージおよび最大 1Mbps のビット レート向けに設計されています。CAN プロトコルは、他のシリアル通信プロトコルに比べて以下の利点があります。

- CAN はメッセージ ベースのプロトコルです。CAN ネットワークのノードは特定のアドレスが割り当てられません。これにより、他の部分に影響を与えずにネットワークにノードを柔軟に追加／削除できます。その上、いずれかのノードに障害が発生した場合、他のノードは続けて正しく動作して通信します。
- CAN メッセージは優先順位を付けることができます。
- CAN は、信頼性のあるトラフィックとデータの整合性を保証するためにエラー チェックの 5 レベルがあります。
- CAN ネットワークは、システム全体のデータの一貫性を持っています。つまり、メッセージがどれかの受信ノードで破損した場合、そのメッセージはいかなる他の受信ノードでも受容されません。
- バスが再度アイドル状態になると破損したメッセージが自動的に再送されます。

1.2 PSoC の用途

PSoC 3 と PSoC 5LP は、単一のチップ上で CAN 機能を、コンフィギュレーション可能なアナログ、プログラマブルなデジタル、メモリ、および CPU コアと統合します。PSoC Creator は、一般的なタスクを抽象化するためのアプリケーション プログラミング インターフェース (API) を内蔵します。

また、CAN コンポーネントを含むすべての PSoC Creator コンポーネントは、C コードを書くよりも、GUI を使用して簡単に設定きます。CAN コンポーネントのシステム レベルの用途については、「AN70630 - Event Data Recorder with Controller Area Network using PSoC 3 and nvSRAM」アプリケーションノートを参照してください。

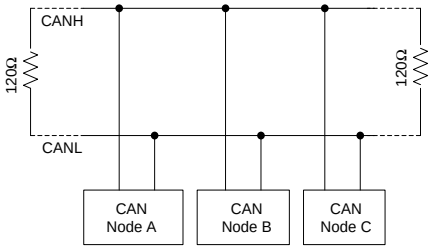
2 CAN の基本

本節では、CAN プロトコルの基本を説明します。プロトコルの詳細は、CAN 仕様で詳しく説明しています。CAN に精通し、CAN を PSoC 3 と PSoC 5LP に実装する方法を理解したいのであれば、「PSoC における CAN」を参照してください。

2.1 物理層

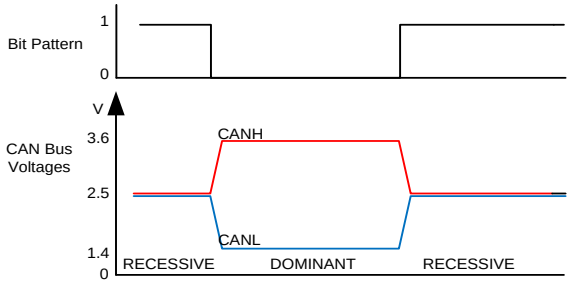
図 1 は、デバイスがどのように CAN バスに接続するかを示します。CAN バスは、CANH と CANL の 2 つの物理的回線で構成されています。CAN バスは、一般的に各端に 120Ω 抵抗で終端されています。

図 1. CAN ネットワークに接続されているノード



CAN バスは、図 2 に示すように差動信号を運びます。バスの両線がほぼ同じ電圧 (通常は 2.5V) であると、「1」が表されます。線間の電圧差が 1.5V～3V であると、「0」が表されます。「1」はリセッシブ ビット (劣性ビット) と呼ばれ、「0」はドミナント ビット (優性ビット) と呼ばれます。

図 2. CAN バス電圧



CAN プロトコルにより、リセッシブ ビットとドミナント ビットが 2 つの異なるノードにより同時に CAN バスに送信された場合、バスはドミナント ビットを持ちます。これは、ワイヤード AND アナロジーに類似しています: すべてのノードがリセッシブ ビットを駆動していない限り、バスはレセッシブ ビットがありません。アイドル状態にある間、リセッシブ ビットがあります。

2.2 転送層

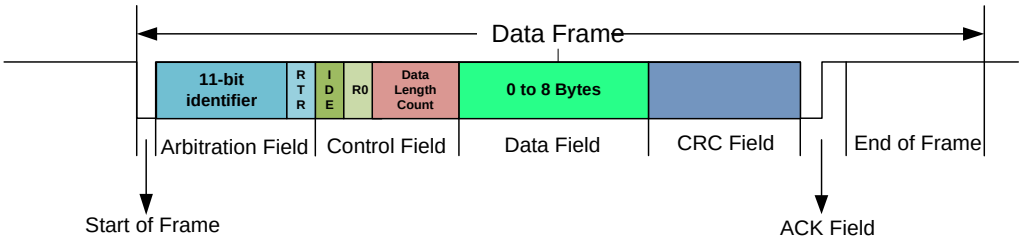
CAN バスがアイドル状態になっているとき、どのノードでもメッセージの送信を開始できます。フレームと呼ばれる固定フォーマットでバスを介して、メッセージは送信されます。CAN は 4 つのフレーム タイプを定義します。

- データ フレーム: トランスミッタからレシーバへデータを伝送します。
- リモート フレーム: データ フレームの送信を要求するために CAN ノードにより送信されます。
- エラー フレーム: バス上のエラーを検出したときに任意の装置により送信されます。
- オーバーロード フレーム: 前後のデータまたはリモート フレームの間に余分な遅延を提供するために使用されます。

これらのフレームのフォーマットの詳細は、付録 A を参照してください。

データ フレームは、図 3 に示すように 7 フィールドで構成されています。

図 3. CAN データ フレーム



アービトレーション フィールド: データ フレームのアービトレーション フィールドは、識別子と遠隔送信要求 (RTR) ビットという 2 つの部分から構成されています。

識別子は、データ フレームで転送されるデータの意味を記述するために使用されます。バス上の各ノードは識別子を確認し、メッセージを受け入れるかどうかを決定します。

識別子フィールドの長さに基づき、CAN メッセージは標準メッセージと拡張メッセージという 2 種が定義されています。

標準 CAN メッセージには 11 ビットの識別子があり、拡張 CAN メッセージには 29 ビットの識別子があります。したがって、標準 CAN データ フレームは 2^{11} のメッセージ タイプをサポートでき、拡張 CAN データフ レームは 2^{29} のメッセージ タイプをサポートできます。

RTR ビットは、特定のフレームがデータ フレームかリモート フレームであるかを示します。RTR ビットは、データ フレームでは「ドミナント」に、リモート フレームでは「リセッシブ」に設定されます。

コントロール フィールド: データ フレームのコントロール フィールドは、データ フィールド内のデータ バイト数を定義します。コントロール フィールドは 6 ビット長で、その中の 4 ビットがデータ長のためで、2 ビットが将来の拡張のために予約されています。

データ フィールド: データ フィールドは、通信するためのデータを含みます。

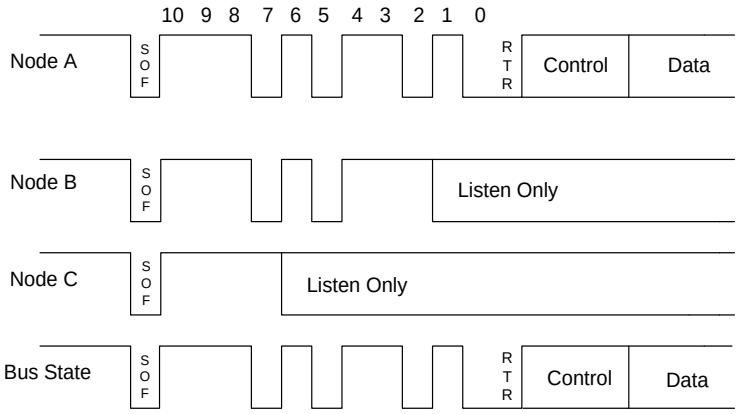
CRC フィールド: 巡回冗長検査 (CRC) フィールドはエラー チェックのためです。このフィールドは、受信したフレームにエラーがあるかどうかを判定するためのビット シーケンスを含みます。ACK フィールドは、フレームが正しく受信されたことを確認するために受信ノードによって使用されます。

受信側として機能するノードは、送信元から特定のメッセージの送信を要求できます。これはリモート フレームにより実施されます。リモート フレームはデータ フィールドを持たないことを除き、データ フレームに似ています。

2.3 バス アービトレーション

複数のノードが同時に送信しようとした場合、バス アービトレーションが図 4 に示すように識別子ビットを使用して行われます。バス上にビットを送信した後、各ノードは、前述したドミナント ビットの特性を利用してバスの状態が送信されたビットと同じかどうかを確認します。同じ場合、ノードは次のビットを送信します。異なっている場合、ノードは送信を停止し、バス上のメッセージの受信を開始します。これは「listen only」 (リッスン オンリー) モードと呼ばれます。

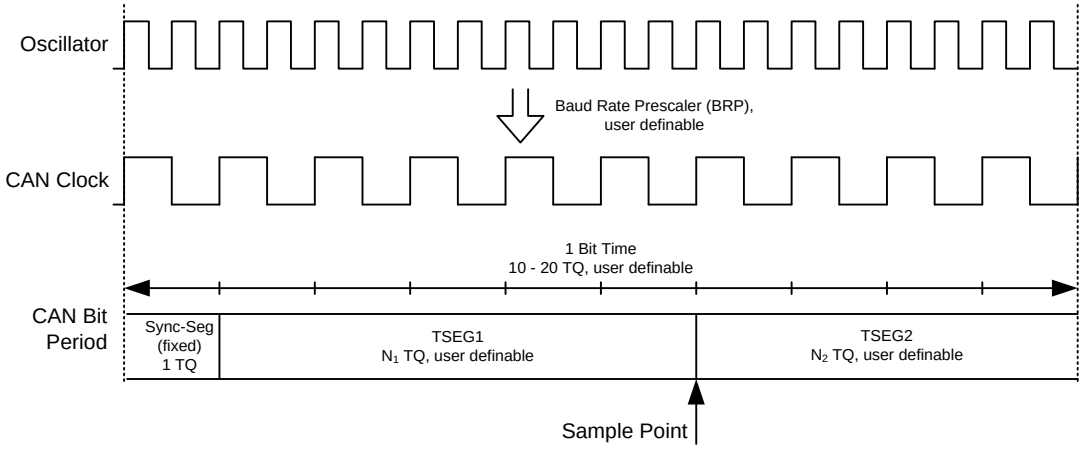
図 4. CAN でのバス アービトレーション



ノードが各ビットを送信した後にバスをリードバックするため、衝突を検出できるようにするために、ビット持続時間はバスの最大伝播遅延よりも大きくなければなりません。したがって、送信されたビットは、伝播遅延を補正するために遅延後にリードバックされます。バスがリードバックされた瞬間は、図 5 に示すように「サンプル ポイント」と呼ばれます。

単一ビットを送信するために要する時間は「ビット時間」と呼ばれます。CAN 仕様では、ビット時間は「タイム クォンタム」 (TQ) で表現されます。タイム クォンタムは、図 5 に示すように発振子の周期から派生する時間の固定単位です。CAN クロック周波数を得るために、発振周波数がボー レート プリスケーラ (BRP) と呼ばれる係数で分周されます。

図 5. 発振子からビット時間の派生



同期セグメント (最初の TQ にある) の開始時に、トランスミッタはバス上にビットを駆動し始めます。トランスミッタはビット時間を通じてバスを駆動し続け、定めた時間後、衝突検出のためにバスをサンプリングします。この時間は、パラメーターTSEG1 と TSEG2 を設定することで確定されます。15 ページの図 18 を参照してください。一般に、サンプル ポイントはビット時間の 60%～80%です。

2.4 CAN のエラー管理

CAN ノードは、以下の 5 つのエラー タイプを検出して処理します。

- ビット エラー: トランスミッタが送信されたビットとバス上のビットが同じでないことを発見したとき、ビット エラーがトランスミッタによって検出されます。
- フォーム エラー: フォーム エラーはメッセージの固定形式に偏差があるときに検出されます。8 バイトを超えるデータ長カウント (DLC) もフォーム エラーと見なされます。
- スタッフ エラー: トランスミッタはビット ストリーム内で同じ値の 5 連続ビットを検出すると、自動的にストリームに相補ビットを挿入します。これはビット スタッフィングと呼ばれます。スタッフ エラーは、同じ値の 6 連続ビットが発見されると検出されます。
- CRC エラー: フレームの CRC フィールドにより示される値がフレームの予想された CRC 値と一致しないときに、CRC エラーが検出されます。
- アクノリッジ エラー: 確認応答 (ドミナント ビット) がフレームのアクノリッジ フィールド中に得られない場合、アクノリッジ エラーがトランスミッタによって検出されます。

ビット エラーとアクノリッジ エラーはトランスミッタによって検出され、スタッフ エラー、CRC エラーおよびフォーム エラーはレシーバによって検出されます。メッセージがエラー検出方法のいずれかに失敗した場合、そのメッセージは受容されず、受信ノードはエラー フレームを生成します。その後、受信ノードはメッセージが正常に受信されるまでメッセージを再送信します。

障害のあるノードが連続してエラー フレームを再送信することでバスをハングアップした場合、エラーの数がエラー上限数に達した後、そのノードの送信機能が除去されます。各ノードは、送信と受信エラーのためのエラー カウンターを維持し、これらは対応するエラーを検出するたびにインクリメントします。エラー カウンター値によって CAN モジュールは 3 つの異なる状態になります。

- エラー アクティブ状態: 送信または受信エラー カウンターの値が 127 以下の場合、ノードは「エラー アクティブ」状態になります。エラー アクティブ状態のノードは通常のバス通信を行えます。
- エラー パッシブ状態: 送信または受信エラー カウンターの値が 128 以上の場合、ノードは「エラー パッシブ」状態になります。エラー パッシブ状態のノードは通常のバス通信を行えます。
- バス オフ状態: 送信エラー カウンターが 256 以上の場合、ノードは「バス オフ」状態になります。バス オフ状態のノードはバス通信を行いません。バスには影響を与えません。

3 PSoC における CAN

3.1 ハードウェア

6 ページの図 6 に示す PSoC 3 と PSoC 5LP の CAN ブロックは CAN 2.0a と 2.0b 規格に準拠します。ただし、出力電圧のレベル シフトを行い、CAN プロトコルと互換性を持たせるために外部トランシーバが必要です。NXP 製の TJA1050 または TI 製の SN65HVD1050-EP を外部トランシーバとして使用できます。これらのデバイスは、3 ページの図 2 に示すように、ビット パターンとバス電圧間を変換します。

本アプリケーション ノートでは、サイプレス キット CY8CKIT-017 は 7 ページの図 7 に示すように、TJA1050 を外部トランシーバとして使用します。

図 6. CAN ブロック図

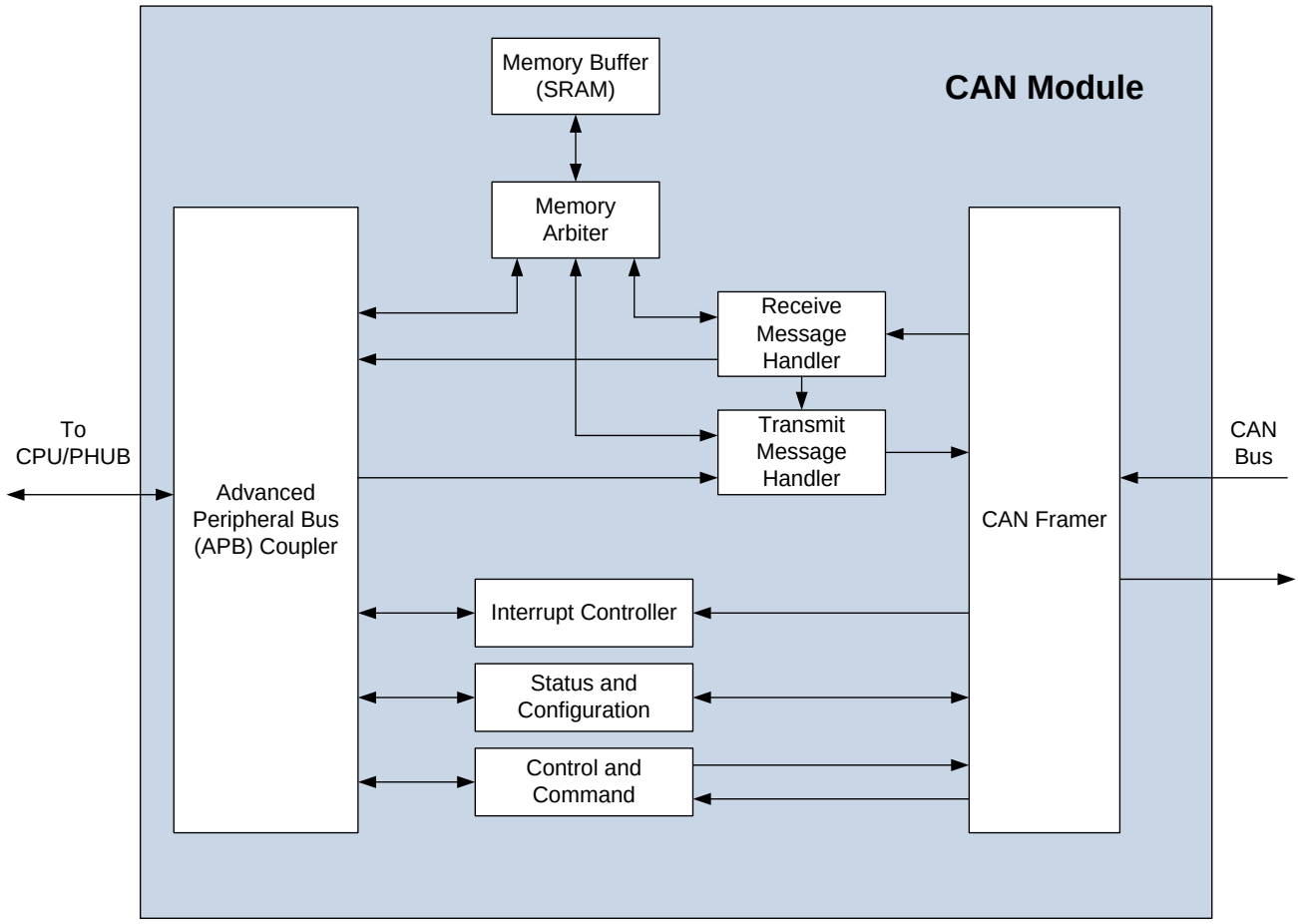
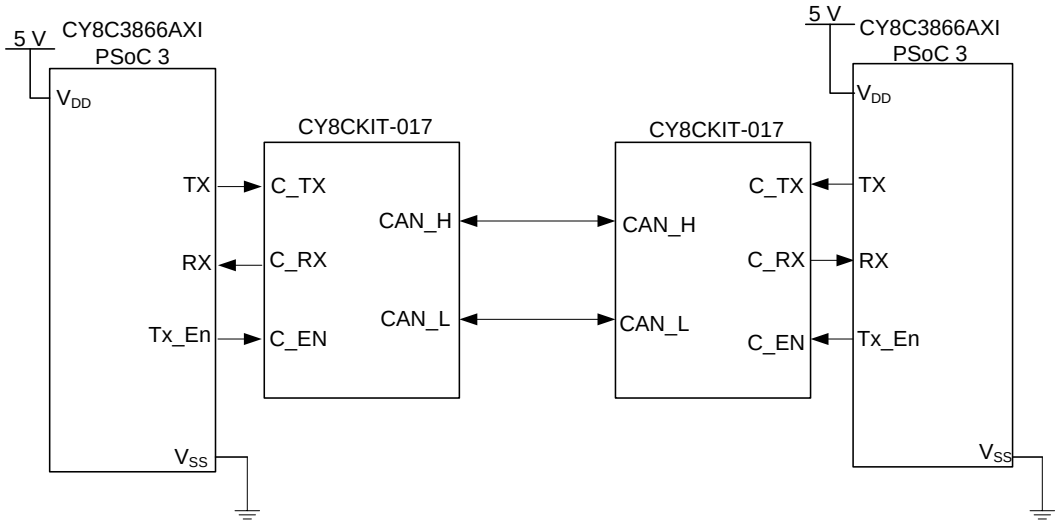


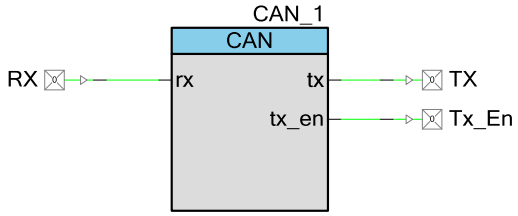
図 7. CAN ネットワークの実装用のハードウェア接続



3.2 CAN コンポーネント

図 8 に、CAN コンポーネントがどのように PSoC Creator 回路図上で表示されるかを示します。

図 8. PSoC Creator の CAN コンポーネント



PSoC は、Full (フル) CAN と Basic (基本) CAN という 2 つの CAN 通信モードを提供しています。以下は、フル CAN のメッセージと基本 CAN のメッセージの重要な違いです。

- フル CAN 通信は、GUI を使用して非常に限られたプログラミングで容易にセットアップできます。基本 CAN 通信は、すべてのパラメーターをファームウェアで設定することを必要とします。
- フル CAN は、メッセージ フィルタリングにはハードウェアを使用します。基本 CAN は、メッセージが受容されるかどうかを判断するために、メッセージ受信のたびに CPU を中断させることを必要とします。
- フル CAN は、メールボックス[1]ごとに単一のメッセージ タイプのみを受け取れる一方で、基本 CAN はメールボックスごとに広範な識別子のメッセージを受け取れます。

次の節では、CAN コンポーネントをフル CAN 通信に設定する方法を説明します。以下の手順は、2 つの PSoC デバイスをそれぞれ別々の開発キット (DVK) で設定する方法を説明します。1 つの PSoC の CAN コンポーネントはトランスミッタに設定され、もう 1 つの PSoC の CAN コンポーネントはレシーバに設定されます。

注: 本アプリケーション ノートに関連するオプションのみが記載されています。CAN コンポーネントに対応するすべてのオプションの詳細は、CAN コンポーネント データシートを参照してください。

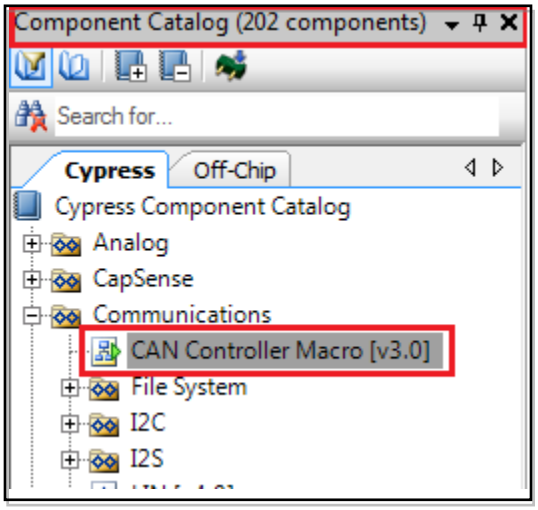
[1] メールボックスは、CAN メッセージを送受信するための入力／出力バッファの一式です。

3.3 PSoC Creator プロジェクト

基本 CAN をデモするためのプロジェクトのビルドは、次の手順を実行してください。

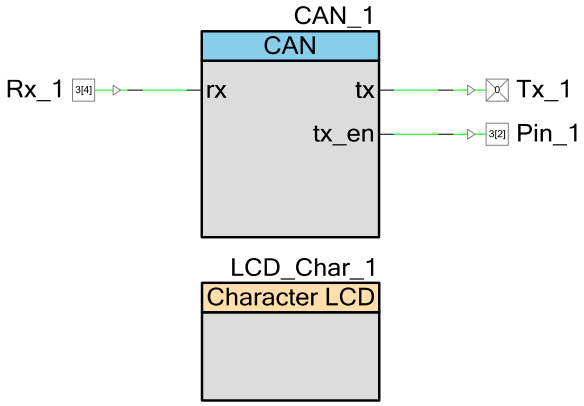
1. PSoC Creator を起動し、新規プロジェクトを作成します (**File** > **New** > **Project**)。プロジェクト名を「Receiver」とします。
2. 図 9 に示すように、CAN コントローラー マクロをコンポーネント カタログから TopDesign 回路図にドラッグ &ドロップしてください。

図 9. コンポーネント カタログの CAN コントローラー マクロ



3. キャラクタ LCD コンポーネントをコンポーネント カタログから TopDesign 回路図にドラッグ &ドロップしてください。これは「Display」フォルダの下にあります。図 10 は完成された回路図を示します。

図 10. 完成された TopDesign

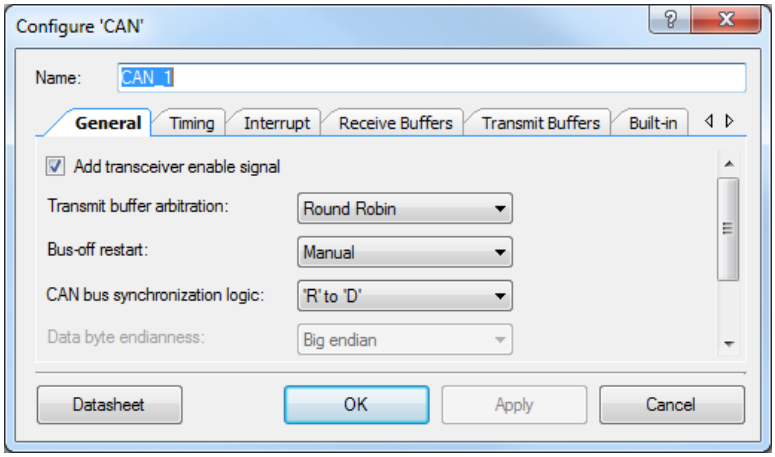


4. TopDesign にある CAN コンポーネントをダブルクリックして設定ウィンドウを開いてください。

3.3.1 CAN の一般設定

図 11 に、PSoC Creator の CAN コンポーネントの一般設定タブを示します。

図 11. CAN コンポーネントの一般設定タブ



5. 「Add Transceiver Signal」チェックボックスにチェックが入っていることを確認します。これは初期設定で有効になっています。

   一般設定タブ内の「Add Transceiver Signal」オプションでは、CAN コンポーネントの TX_EN 信号を有効／無効にします。この信号は、外部トランシーバのイネーブル ピンに接続するために使用されます。

6. 「Transmit Buffer Arbitration」を「Round-Robin」に設定してください。

   「Round Robin」オプションは、すべての送信メールボックスが均等な送信の機会を与えられることを保証することに対し、「Fixed priority」オプションはどのメッセージが送信されるかに応じてメールボックスに優先順位を割り当てます。

7. 「Bus-Off Restart」の方式を「Manual」に設定してください。

   バス オフ再起動は、手動または自動で送信エラーの数を監視することで行います。

8. 「CAN Bus Synchronization Logic」に「'R' to 'D'」を選択してください。

   CAN ネットワークではクロック信号は送信されません。代わりに、すべての受信ノードのクロックは、トランスミッタによって送信されたフレームの開始 (SOF) ビットの立ち下がりエッジで同期化されます。後続のエッジは、異なるノード間のクロックの小さなドリフトに適合するためにクロックを同期化するために使用されます。

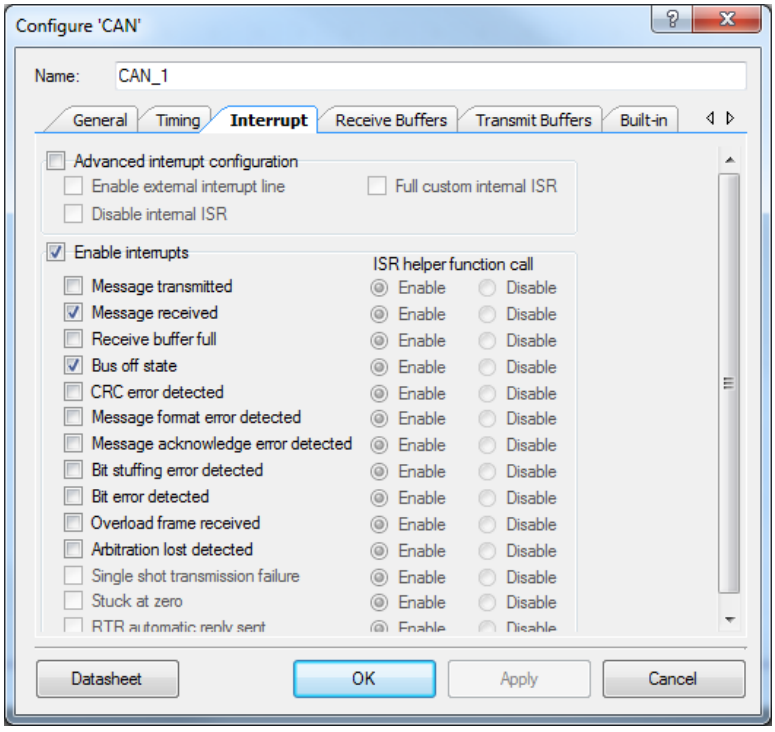
   同期化は、リセッシブからドミナントへ ('R' to 'D') の遷移、または両方のエッジ (リセッシブからドミナントへ、ドミナントからリセッシブへ) で行えます。

3.3.2 CAN 割り込み設定

図 12 に、CAN コンポーネントの割り込み設定タブを示します。このタブを使用していくつかのイベントが発生したときの割り込みを有効／無効にします。割り込みを有効にした場合、PSoC はイベントが発生するときに割り込みサービス ルーチン (ISR) を実行します。

Message Received (メッセージ受信) と Bus Off State (バスオフ状態) の割り込みは初期設定では選択されています。Message Receive 割り込みは自動的に *CAN_TX_RX_func.c* 内の `ReceiveMsgx()` 関数を呼び出します。

図 12. CAN コンポーネントの割り込み設定タブ



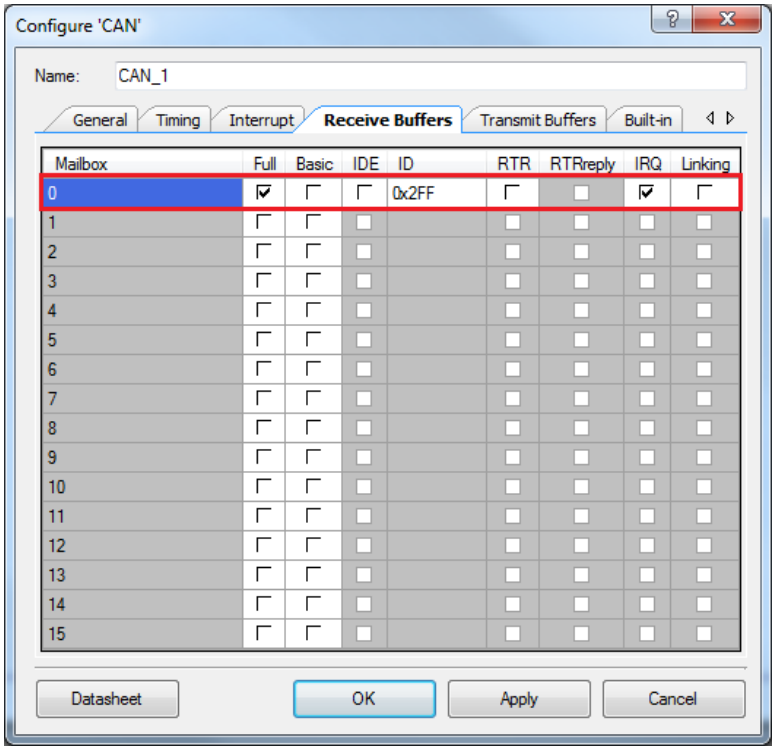
「Enable Interrupts」、「Message Received」および「Bus Off State」の割り込みを有効にするために、これらのチェックボックスが必ずチェックされていることを確認してください。他のすべてのボックスをチェックしないでください。

3.3.3 CAN 受信バッファ

CAN コンポーネントは、メッセージを受信するために 16 の入力バッファ (メールボックス) を持ちます。したがって、CAN コンポーネントは最大 16 の異なる CAN メッセージ タイプを受信できます。

メールボックスは、初期設定では 0～15 の番号が付けられ、お好みの名前に置き換えられます。このために、図 13 に示すように「Full」モードを選択してください。Full モードを選択すると、その行の他の設定可能なオプションが有効になります。

図 13. CAN コンポーネントの受信バッファ設定



1. 0x2FF の ID を持つフル メールボックスを有効にするために、図 13 に示すようにチェックボックスを選択してください。ID フィールドは、該当するメールボックスで受信されるメッセージの識別子を定義し、任意の 11 ビット値に設定できます。
2. メッセージがメールボックスで受信されると割り込みをトリガーするために、IRQ ボックスにチェックを入れてください。

   これで、レシーバ側の CAN コンポーネントの設定が完了します。この場合、送信ノードがメッセージを送信する必要がないため、送信バッファを設定する必要はありません。

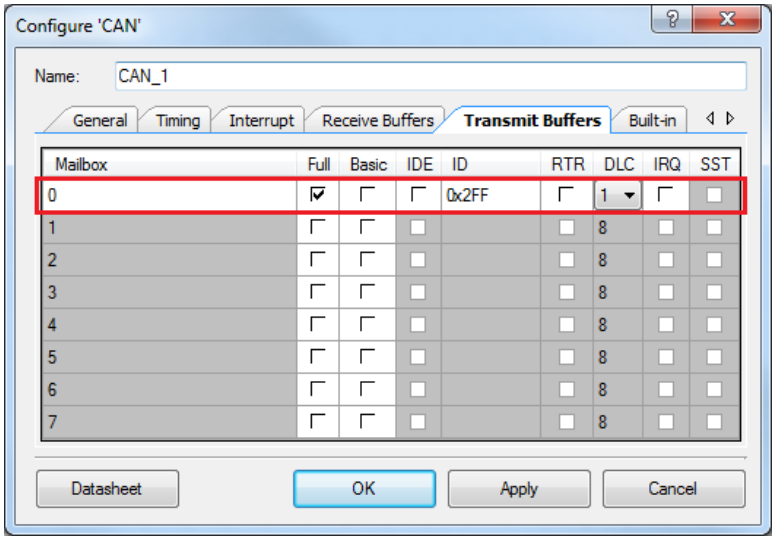
3.3.4 CAN の送信バッファ

次に、トランスミッタ側を設定しましょう。別の PSoC に対して行うため、その PSoC 向けの別のプロジェクトを作成する必要があります。

1. ワークスペースに別のプロジェクトを追加するために、Workspace Explorer ウィンドウ内のワークスペース名を右クリックし、**Add** > **New Project** を選択してください。プロジェクト名を「Transmitter」とします。
2. CAN コントローラー マクロをこのプロジェクトの TopDesign にドラッグ &ドロップしてください。レシーバ側と同様に、3.3.1～3.3.3 で説明したように CAN コンポーネントを設定してください。

「Transmit Buffers」タブは、図 14 示すように送信メールボックスを設定するために使用します。送信メールボックスは Full (フル) または Basic (基本) のいずれかに設定できます。フル モードを選択すると、その行の他の設定可能なオプションが有効になります。

図 14. CAN コンポーネントの受信バッファ設定



3. レシーバ側と同様に、0x2FF の ID を持つフル メールボックスを有効にするために、図 14 に示すようにチェックボックスを選択してください。

   1 バイトのみを送信するため、データ長カウント (DLC) フィールドは 1 です。

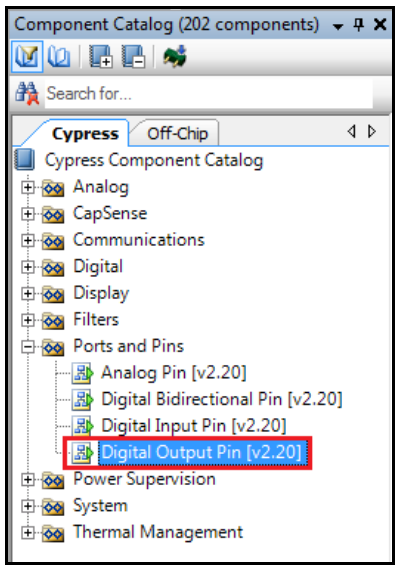
3.3.5 ピン配置

本節の手順は、「Receiver」と「Transmitter」両方のプロジェクトに適用します。

これらのプロジェクトにピンを追加して設定しましょう。Tx_1 と Rx_1 ピンはステップ 2 で CAN マクロの一部として含まれています。図 10 に示すように第 3 のピンを追加する必要があります。

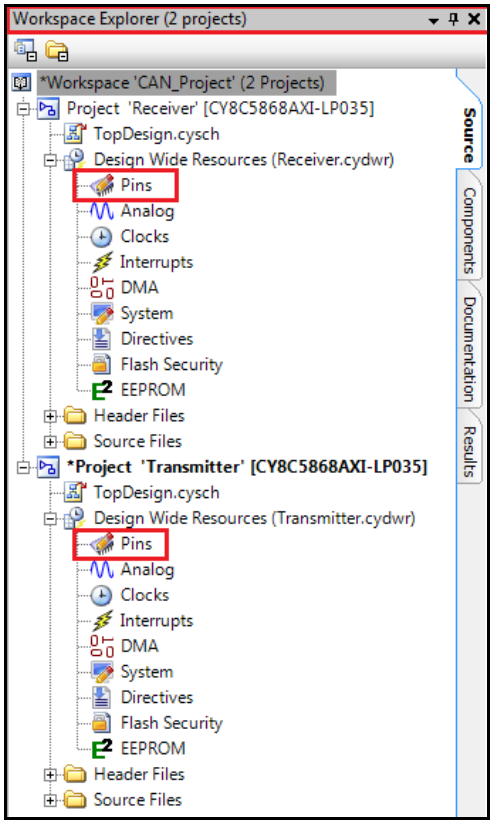
1. 図 10 と図 15 に示すように両プロジェクトの TopDesign 回路図にデジタル出力ピンをドラッグ &ドロップしてください。このピンを CAN コンポーネントの tx_en 出力に接続してください。このピンは、外部トランシーバを有効にするために使用されます。

図 15. デジタル出力ピンを探す



3. 図 16 に示すように、Workspace Explorer で各プロジェクトのデザイン ワイド リソース (*.cydwr*) ファイルの「Pins」タブを開いてください。

図 16. .cydwr ファイルを探す



4. 表 1 に示すように、各ピンにポートを割り当てます。

表 1. ピンの割り当て

ピン	CY8CKIT-001	CY8CKIT-030／CY8CKIT-050
Rx_1	P3[4]	P3[4]
Tx_1	P3[3]	P3[3]
Pin_1	P3[2]	P3[2]
LCD_Char_1	P2[6:0]	P2[6:0]

3.3.6 クロック設定

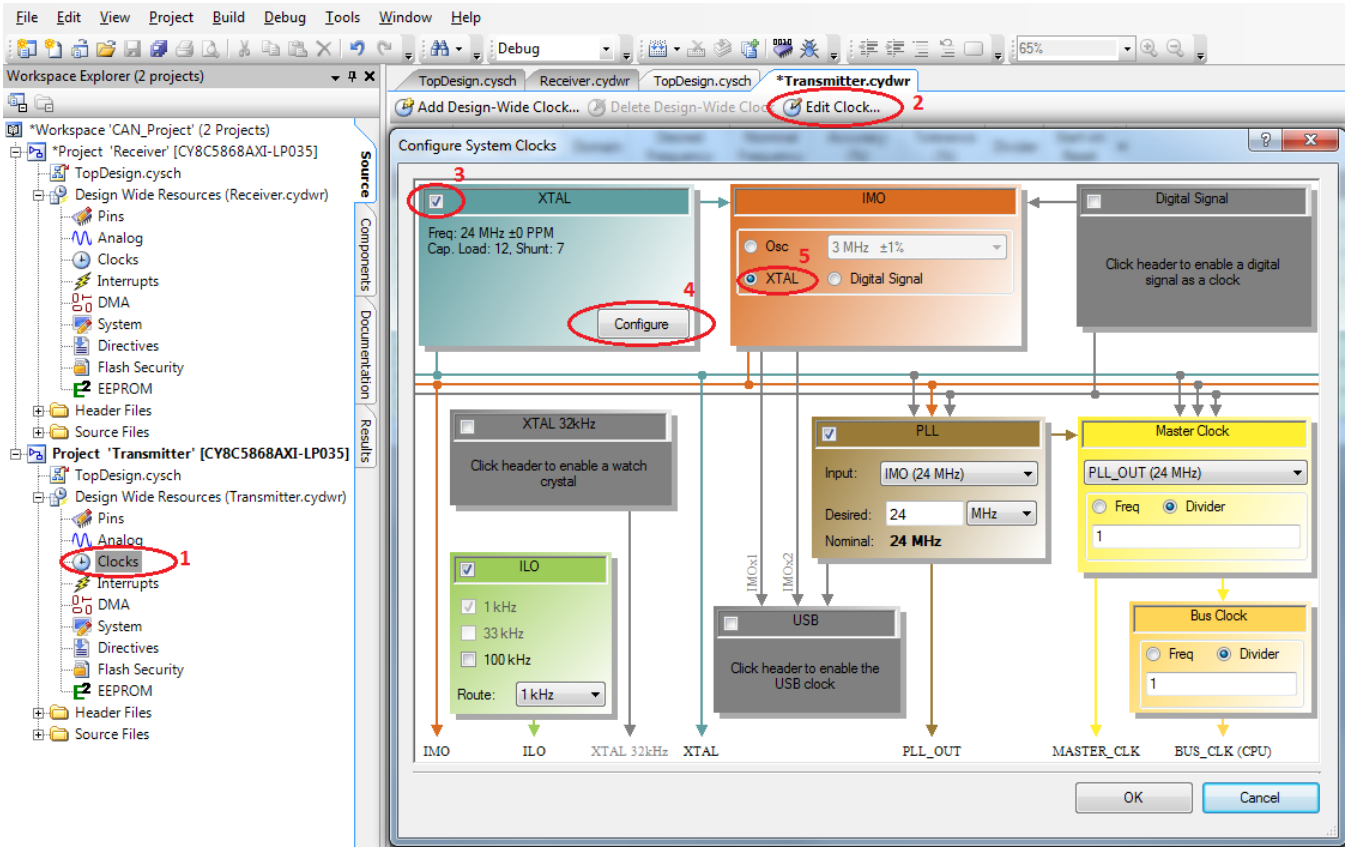
本節の手順は、「Receiver」と「Transmitter」両方のプロジェクトに適用します。

CAN プロトコルはビットを同期させるためのクロックを送信しません。5 ページの図 5 に示したように、ノード間の同期化は同期セグメント中に送信されたビットごとに行われます。このためには、125Kbps より高いボー レートを得るために高精度な発振子を使用する必要があります。

CAN プロトコルでは、クロック精度は 0.5%以下であることが規定されています。外部水晶発振子を使用することで 0.1%未満の誤差は PSoC で得られます。デザイン ワイド リソース ファイル (拡張子.cydwr を持つファイル) 内のクロック ツリー ダイアログがこれを設定するために使用されます。

1. 図 17 に示すように、デザイン ワイド リソース ファイルを開いてください。「Clocks」タブで、**Edit Clocks** をクリックします。クロック ツリー ダイアログが開きます。
2. 水晶振動子を有効にするために、クロック ツリーの左上にあるチェックボックスをクリックします。**Configure** ボタンをクリックして、水晶振動子周波数に「24 MHz」を入力してください。
3. IMO ソースとして XTAL を選択してください。他の設定を変更しないでください。
4. ご使用のキットが XTAL を備えていない場合、24MHz 水晶振動子と 2 個のコンデンサをピン P15[0]と P15[1]に接続します。詳細については、PSoC 3 または PSoC 5LP のデータシートを参照してください。

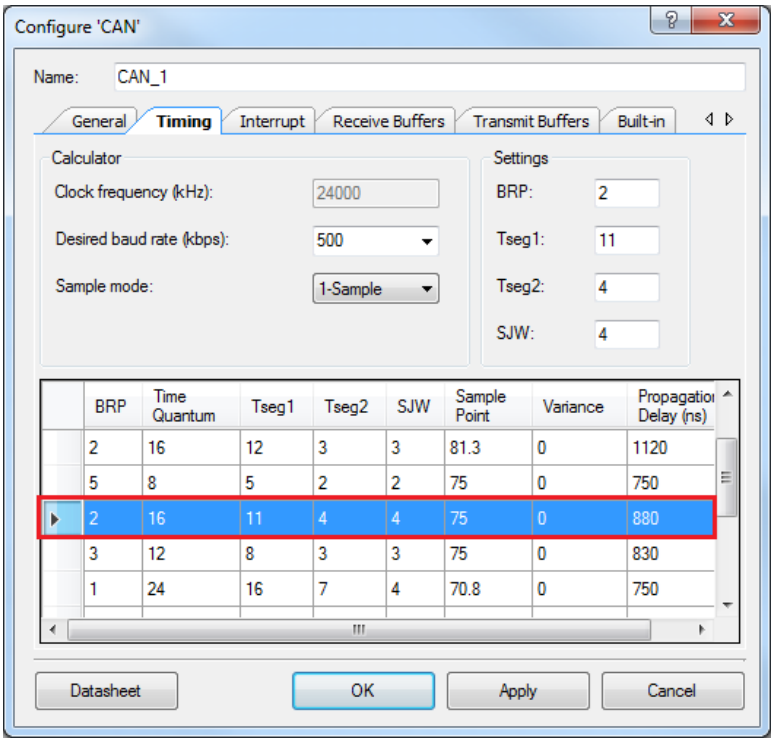
図 17. クロック ツリー設定



3.3.7 CAN タイミング設定

図 18 に、PSoC Creator の CAN コンポーネントのタイミング設定タブを示します。

図 18. CAN コンポーネントのタイミング設定タブ



1. 「Baud-rate」を「500 kbps」に設定してください。これにより、図 18 に示すように値が自動的に変更されます。

   ボーレートは、デバイス間の通信速度を決定します。この速度を最大 1Mbps まで設定できます。バス上のすべての CAN ノードが同じボー レートで動作する必要があります。

   注: ボー レートの選択により、表内にあり得るタイミング パラメータの一覧のみが表示されます (図 18 を参照してください)。表内で行をダブルクリックしてそれぞれのフィールドのパラメーターを更新する必要があります。

2. BRP = 2、Time Quantum = 16、Sample Point = 75、SWJ = 4 の行 (図 18 を参照) をダブルクリックして、これらの値を「Settings」に追加してください。

   最高性能を得るためには、「Sample Point」が 60～80 で、「Variance」が 0 の行を選択してください。

   同期ジャンプ幅 (SJW) は、サンプル ポイントがその平均位置から外れられるタイム クォンタムの数です。この値は、5 ページの図 5 に示される TSEG1 と TSEG2 の両方の以下でなければなりません。

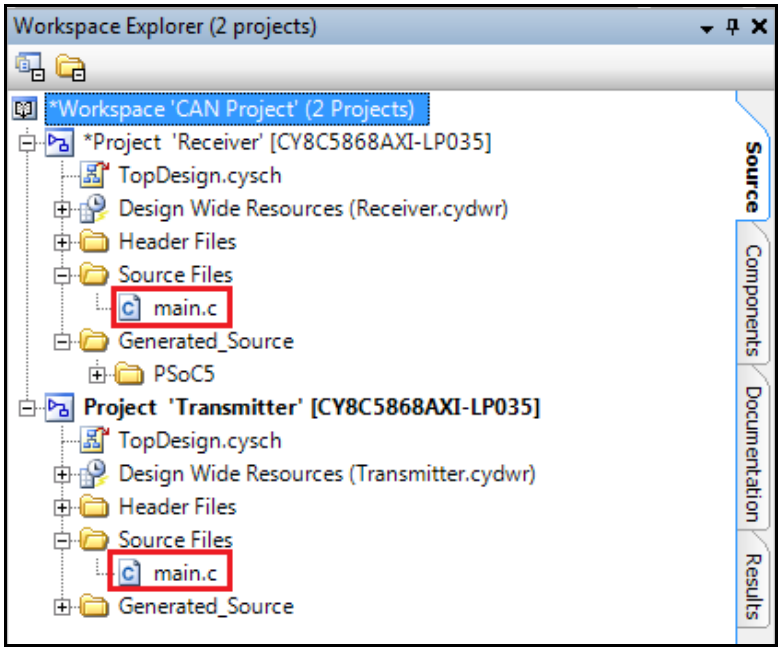
   サンプル モードは、バスの状態を判定するために取られるサンプルの数です。シングル サンプル モードまたは 3 サンプル モードを選択できます。

3.4 ファームウェア

本節の手順は、「Receiver」と「Transmitter」両方のプロジェクトに適用します。各プロジェクトに少量のファームウェア コードを追加する必要があります。

1. ***Build*** > ***Build All Projects*** メニュー項目を選択し、各プロジェクトをビルドしてください。CAN コンポーネントおよびその他の API ファイルが自動的に生成されます。
2. Workspace Explorer 内で「Transmitter」プロジェクトの *main.c* ファイル名をダブルクリックしてファイルを開きます (図 19 を参照してください)。コード 1 からのコードを *main.c* ファイルに追加してください。

図 19.*main.c* ファイルを探す

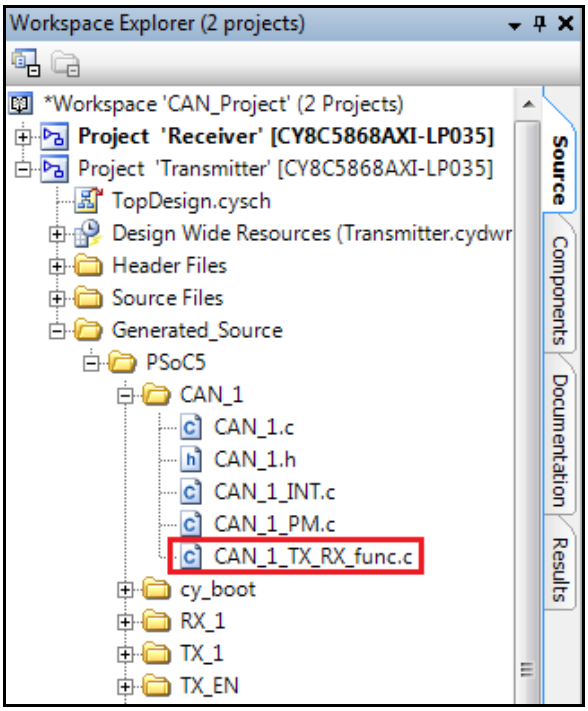


コード 1.「Transmitter」の *main.c* コード

```
#include <device.h>

uint8 Tx_Data = 0;

int main()
{
  CAN_1_Start();
  LCD_Char_1_Start();
  CyGlobalIntEnable;

  for(;;) /* do forever */
  {
    LCD_Char_1_ClearDisplay();
    LCD_Char_1_Position(0,0);
    LCD_Char_1_PrintNumber(Tx_Data);
    CAN_1_SendMsg0();
    Tx_Data++;
    CyDelay(500);
  }
}
```

3. 図 20 に示すように、Workspace Explorer 内で「Transmitter」プロジェクトの *CAN_1_TX_RX_func.c* ファイル名をダブルクリックしてファイルを開いてください。このファイルは、プロジェクトがビルドされた後に生成されます。

図 20.*CAN_1_TX_RX_func.c* を探す



4. 以下の行に移動します。

```
/* `#START TX_RX_FUNCTION` */

/* `#END` */
```

5. 上記の行の間に次のコード行を入力します。

```
extern uint8 Tx_Data;
```

6. 同じファイル内に関数 CAN_1_SendMsg0() を配置してください。以下のコード行に移動します。

```
/* `#START MESSAGE_0_TRASMITTED` */

/* `#END` */
```

7. 上記の行の間に次のコード行を入力します。

```
CAN_1_TX_DATA_BYTE1(0) = Tx_Data;
```

8. Workspace Explorer 内で「Receiver」プロジェクトの *main.c* ファイル名をダブルクリックしてファイルを開きます。コード 2 からのコードを *main.c* ファイルに追加してください。

コード 2.「Receiver」の main.c コード

```
#include <device.h>

uint8 Rx_Data;

int main()
{
  CAN_1_Start();
  LCD_Char_1_Start();
  CyGlobalIntEnable;

  for(;;) /* do forever */
  {
    LCD_Char_1_ClearDisplay();
    LCD_Char_1_Position(0,0);
    LCD_Char_1_PrintNumber(Rx_Data);
  }
}
```

9. Workspace Explorer 内で「Receiver」プロジェクトの *CAN_1_TX_RX_func.c* ファイル名をダブルクリックしてファイルを開いてください。このファイルは、プロジェクトがビルドされた後に生成されます。

10. 以下の行に移動します。

```
/* `#START TX_RX_FUNCTION` */
/* `#END` */
```

11. 上記の行の間に次のコード行を入力します。

```
extern uint8 Rx_Data;
```

12. 同じファイル内に関数 CAN_1_ReceiveMsg0() を配置します。以下のコード行に移動します。

```
/* `#START MESSAGE_0_RECEIVED` */
/* `#END` */
```

13. 上記の行の間に次のコード行を入力します。

```
Rx_Data = CAN_1_RX_DATA_BYTE1(0);
```

14. 両方のプロジェクトをビルドし、それぞれを 2 つのキットの PSoC デバイスにプログラムしてください。プログラム オプションは、PSoC Creator の Debug メニューにあります (**Debug** > **Program**)。

3.4.1 その他のファームウェアの注意事項

CAN コンポーネントのファームウェアを書く際には、次の点に注意してください。

- CAN コンポーネントは、使用する前に初期化して *main.c* ファイルで開始する必要があります。
- グローバル割り込みは、CAN コンポーネントからの割り込み要求を処理するために有効にする必要があります。
- 特定の送信メールボックスからのフル CAN メッセージを送信するために、`CAN_TX_SendMsgx()` 関数を呼び出します (x はメールボックス番号であり、CAN_TX は PSoC Creator の TopDesign 回路図内で CAN コンポーネントに与えられた名前です)。
- フル CAN メッセージの送受信に必要な関数は *CAN_TX_TX_RX_func.c* ファイルで提供されています。このファイルは、プロジェクトがビルドされた後に生成されます。

PSoC で CAN を設定するためにファームウェアを書く方法を理解するには、本アプリケーション ノートで提供されたサンプル プロジェクトを参照してください。

4 ハードウェアの実装

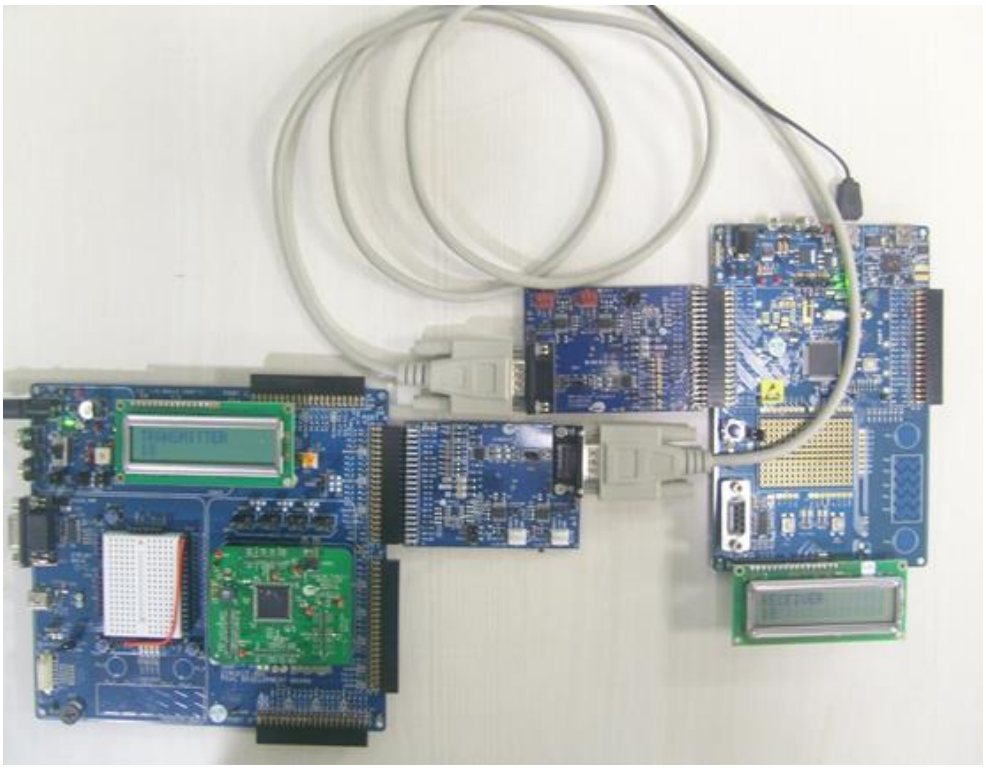
CAN コンポーネントからの PSoC 出力は TX と RX です。CANH および CANL 信号を取得するには、これらの出力のレベルを変換する必要があります。CY8CKIT-017 拡張基板キットは、本アプリケーション ノートで提供されたサンプル プロジェクトにおいてレベル変換を行う外部トランシーバとして使用されます。このキットで使用されている CAN トランシーバ IC は TJA1050 です。トランシーバは、21 ページの図 25 に示すように最小限の接続で実装することがあります。その図では、CY8CKIT-017 は TJA1050 に置き換えられます。詳細な図は、付録 B に示します。

CAN バスには 2 本のワイヤしか必要としません。図 21 に示すように CY8CKIT-017 では、標準オスオス DB9 コネクタは 2 つの CAN ノード間を接続するために使用できます。CAN ネットワーク内の任意の 2 CAN ノード間のケーブル長を表 2 に示す値に制限するべきです。これらの値は CAN 仕様には記載されていませんが、設計で使用される標準値です。

表 2. 異なるボー レートに対応する標準ケーブル長

ボー レート (ビット/秒)	標準ケーブル長 (メートル)
1Mbps	40
500kbps	100
250kbps	200
125kbps	500
10kbps	6000

図 21. CAN の物理接続

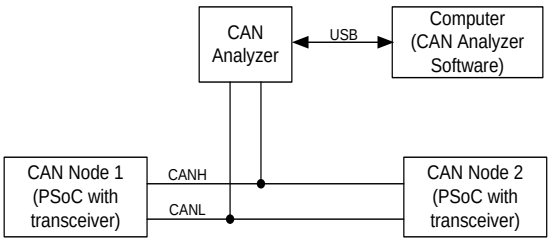


4.1 CAN アナライザの操作

CAN アナライザは、デバッグ目的のために CAN バス上でデータ トラフィックを監視するために使用されるデバイスです。これらのデバイスは Microchip や Peak-system などのメーカーから入手できます。図 22 は、CAN バスへの USB CAN アナライザの接続を示すブロック図です。

CAN アナライザは、PCAN View などのソフトウェアが、CAN アナライザが接続したコンピュータ システム上にインストールされることも必要とします。このソフトウェアは、CAN アナライザが受信したデータを解釈して、アプリケーションのインターフェースに表示します。

図 22. CAN バスに接続されている USB CAN アナライザ



5 サンプル プロジェクト

本アプリケーション ノートには 4 つのサンプル プロジェクトが含まれています。

5.1 プロジェクト 1 と 2: シンプレックス通信

21 ページの図 25 に示すプロジェクト 1 と 2 は、2 つの CAN ノード間のシンプレックス通信をデモします。

プロジェクト 1 ではトランスミッタを実装します。スイッチ上のキーの押下を監視し、CAN を介してキーの押下回数を通信します。プロジェクト 2 ではレシーバを実装します。データを受信してキャラクタ LCD ディスプレイに表示します。21 ページの図 23 と図 24 に示すフローチャートは、これらのプロジェクトのプログラム フローを示します。

サンプル プロジェクト 1 と 2 をセットアップするには、次の手順を行ってください。

1. 21 ページの図 25 に示すようにシステムを構築してください。2 つの CY8CKIT-001 DVK と 2 つの CY8CKIT-017 EBK を使用できます。CY8CKIT-001 DVK の代わりに CY8CKIT-030 または CY8CKIT-050 DVK を使用できます。これらのプロジェクトをテストするために、PSoC 5LP FreeSoC2 開発基板と CY8CKIT-026 を使用することもできます。

   注: FreeSoC2 開発基板には、デフォルトで水晶振動子がありません。FreeSoC2 開発キットでこのプロジェクトをテストするためには、FreeSoC2 基板に 24MHz の水晶振動子を装着する必要があります。
2. プロジェクト ファイルを開き、***.cydwr*** ファイルで正しいピン割り当てを行ってください。ピン割り当ては、表 3 と表 4 に示します。

表 3. CAN シンプレックス トランスミッタのピン使用

ピン	CY8CKIT-001	CY8CKIT-030／CY8CKIT-050	FreeSoC2 キット
LCD_Tx	P2[6:0]	P2[6:0]	該当なし
Data_In	P0[0]	P6[1]	P1[2]
RX	P3[4]	P3[4]	P3[4]
TX	P3[3]	P3[3]	P3[3]
Tx_En	P3[2]	P3[2]	P3[2]
UART_Tx	該当なし	該当なし	P4[0]

表 4. CAN シンプレックス レシーバのピン使用

ピン	CY8CKIT-001	CY8CKIT-030／CY8CKIT-050	FreeSoC2
LCD_Rx	P2[6:0]	P2[6:0]	該当なし
RX	P3[4]	P3[4]	P3[4]
TX	P3[3]	P3[3]	P3[3]
Tx_En	P3[2]	P3[2]	P3[2]
UART_Tx	該当なし	該当なし	P4[0]

3. 両方のプロジェクトをビルドしてください。CAN_SimplexCommunication_Tx を第 1 の PSoC に、CAN_SimplexCommunication_Rx を第 2 の PSoC にプログラムしてください。

   CAN_SimplexCommunication_Tx プロジェクトは LCD ディスプレイの 1 行目に「TRANSMITTER」を表示します。2 行目は、Data_in ピンに登録されたキーの押下回数を示します。

   CAN_SimplexCommunication_Rx プロジェクトは LCD ディスプレイの 1 行目に「RECEIVER」を表示します。2 行目は、第 1 の PSoC により CAN を介して送信されたデータを表示します。これは、CAN_SimplexCommunication_Tx プロジェクトの 2 行目に表示されているキーの押下回数と同じです。

図 23. プロジェクト 1 のフローチャート (トランスミッタ)

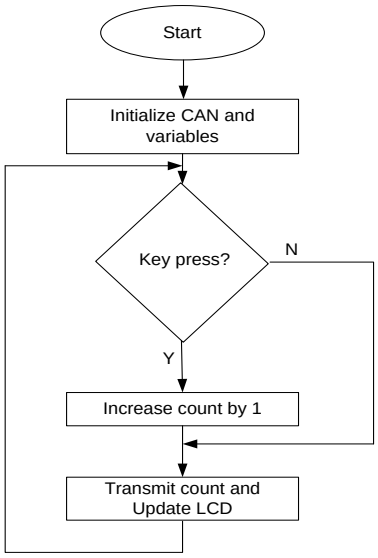


図 24. プロジェクト 2 のフローチャート (レシーバ)

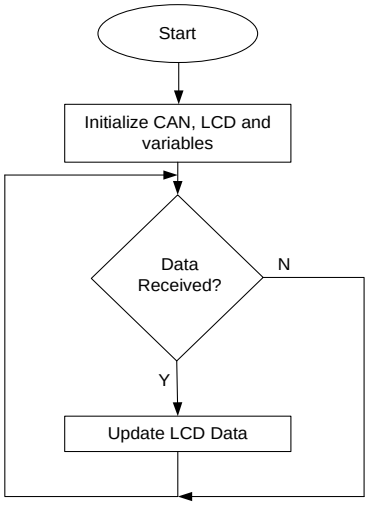
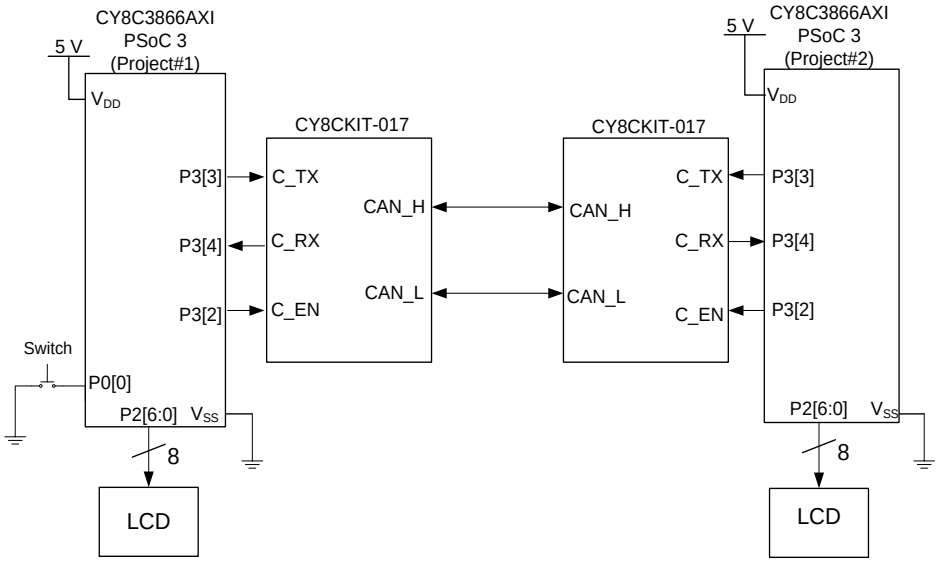


図 25. プロジェクト 1 と 2 の物理構成 (CY8CKIT-030)



5.2 プロジェクト 3 と 4: CAN の RTR 機能

23 ページの図 28 に示すプロジェクト 3 と 4 は、CAN の RTR 機能をデモします。

プロジェクト 3 はノード 1、プロジェクト 4 はノード 2 と名付けます。ノード 1 は ADC データ入力を監視し、ADC 値を LCD ディスプレイに表示します。ノード 2 からの RTR 要求が発生した場合、現時点の ADC 値はノード 2 に送信されます。ノード 2 はスイッチ上のキーの押下があるかを連続して確認し、キーが押されたときにノード 1 に RTR 要求を発行します。ノード 1 によって送信された ADC データ値は、ノード 2 によって受信され、LCD ディスプレイに表示されます。23 ページの図 26 と図 27 に示すフローチャートは、これらのプロジェクトのプログラム フローを示します。

サンプル プロジェクト 3 と 4 をセットアップするには、次の手順を行ってください。

1. 23 ページの図 28 に示すようにシステムを構築してください。2 つの CY8CKIT-001 DVK と 2 つの CY8CKIT-017 EBK を使用できます。CY8CKIT-001 DVK の代わりに CY8CKIT-030 または CY8CKIT-050 DVK を使用できます。
2. プロジェクト ファイルを開き、*.cydwr* ファイルで正しいピン割り当てを行ってください。ピン割り当ては、表 5 と表 6 に示します。

表 5. CAN_RTR_Node1 のピン使用

ピン	CY8CKIT-001	CY8CKIT-030／CY8CKIT-050
LCD	P2[6:0]	P2[6:0]
ADC_In	P0[0]	P6[5]
RX	P3[4]	P3[4]
TX	P3[3]	P3[3]
Tx_En	P3[2]	P3[2]

表 6. CAN_RTR_Node2 のピンの使用

ピン	CY8CKIT-001	CY8CKIT-030／CY8CKIT-050
LCD	P2[6:0]	P2[6:0]
RTR_In	P0[0]	P6[1]
RX	P3[4]	P3[4]
TX	P3[3]	P3[3]
Tx_En	P3[2]	P3[2]

3. 両方のプロジェクトをビルドします。CAN_RTR_Node1 を第 1 の PSoC に、CAN_RTR_Node2 を第 2 の PSoC にプログラムしてください。

   CAN_RTR_Node1 プロジェクトは LCD ディスプレイの 1 行目に「Node1」を表示し、2 行目に ADC 出力の現時点の値を表示します。

   CAN_RTR_Node2 プロジェクトは LCD ディスプレイの 1 行目に「Node2」を表示します。RTR_In キーが押されたとき、1 行目は「RTR Sent」(RTR 送信済み) を表示し、2 行目は RTR に応答してノード 1 から受信した ADC 出力の値を表示します。

図 26. プロジェクト 3 (ノード 1) のフローチャート

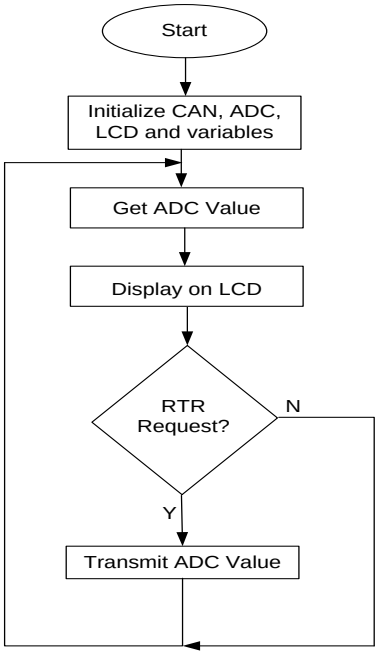


図 27. プロジェクト 4 (ノード 2) のフローチャート

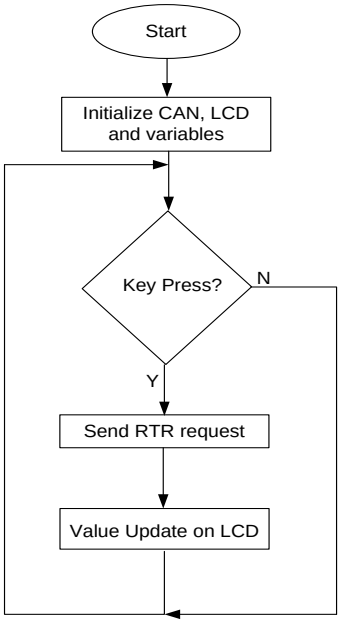
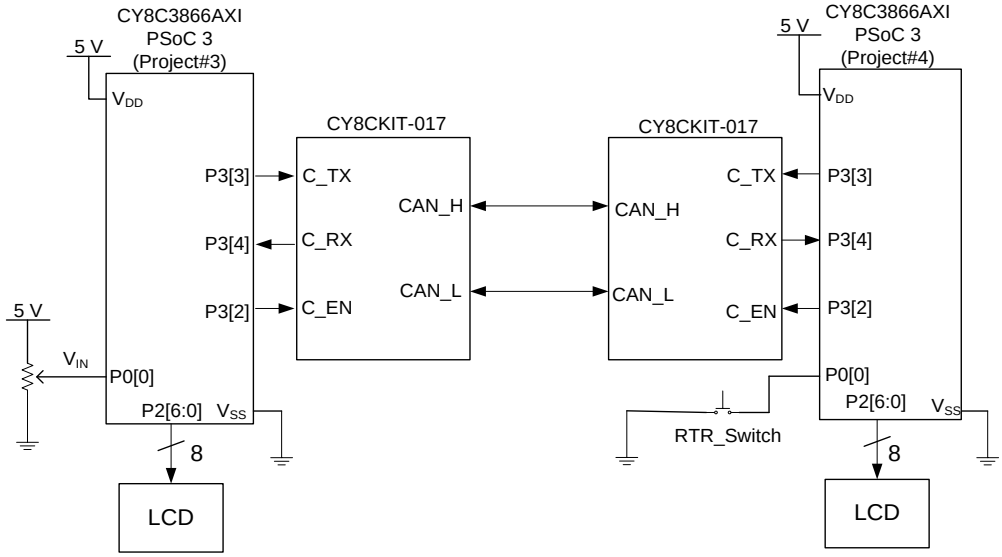


図 28. プロジェクト 3 と 4 の物理構成



これらのプロジェクトが正常に動作するために、CY8CKIT-017 のジャンパ JP2 が装着されていることを確認してください。また、CY8CKIT-017 のジャンパ JP6 は V5_0 と V_{DD} の間を接続する必要があります。標準オスオス DB9 コネクタを 2 つの CAN ノード間を接続するために使用できます。

PSoC Creator サンプル プロジェクトのセクションには、もう 2 つのサンプル プロジェクトが提供されています。

6 まとめ

CAN は主に車載アプリケーションに使用されている信頼性の高いシリアル通信プロトコルです。このプロトコルは、デバイス間の双方向通信を可能にし、フレキシブルなネットワークを提供しています。

サイプレスの PSoC 3 と PSoC 5LP は PSoC Creator とともに、CAN 2.0a と CAN 2.0b 仕様に準拠しており、CAN ネットワークを容易にセットアップするためのユーザーフレンドリーなインターフェースと API 一式を提供しています。本アプリケーション ノートは、PSoC で CAN を円滑に実装できるように案内します。

7 関連リソース

- CE95282 - CAN as Control Node with PSoC 3/5LP

  CAN コンフィギュレーション、フル メッセージの送受信、および Tx と Rx エラーの処理方法を示します。

- CE95283 - CAN as Remove Node with PSoC 3/5LP

  CAN コンフィギュレーション、フル メッセージの送受信、および Tx と Rx エラーの処理方法を示します。

- CE97311 - PSoC® 4 M: CAN Simplex Communication with CapSense®

  CAN バスを介してデータを送受信する方法を示します。CapSense®ボタンのステータスは、CAN レシーバの LED を制御するために CAN 経由で送信されます。

- CE211027 - CAN Communication Between FM4-S6E2Gx Series and PSoC® 4 M-Series Using CY8CKIT- 026 CAN and LIN Shield Kit

  CY8CKIT-026 CAN および LIN シールド キットを使用した FM4-S6E2Gx シリーズと PSoC®4 M シリーズ間の CAN 通信をデモします。

著者について

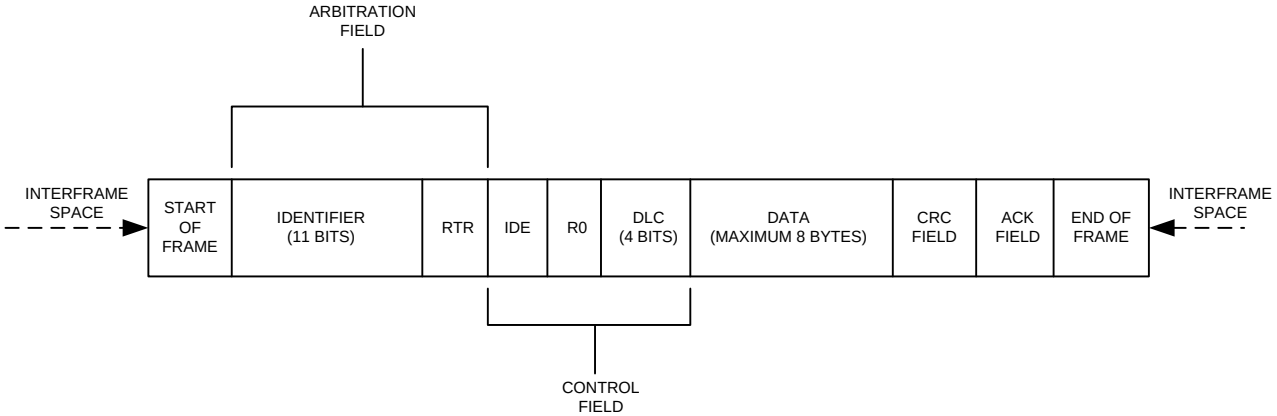
氏名: Ranjith M

役職: アプリケーション エンジニア

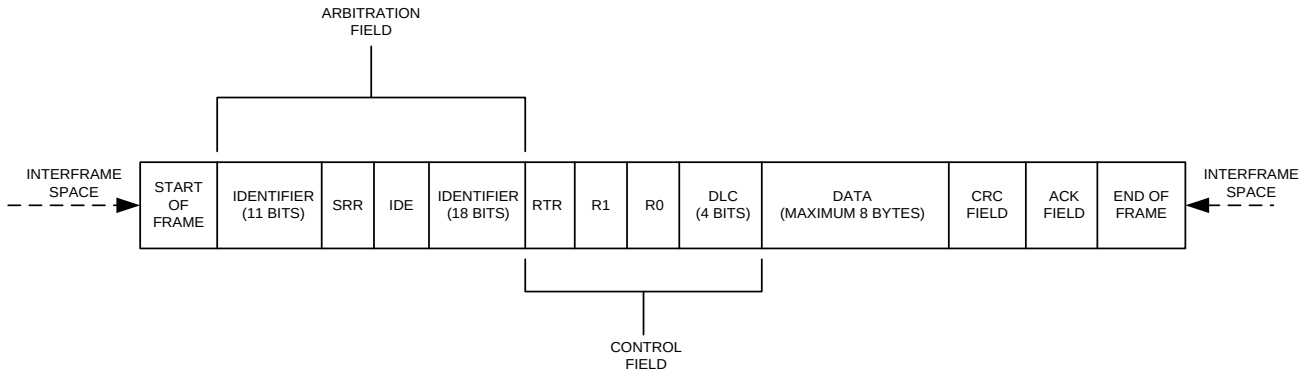
経歴: Ranjith は Government Engineering College (トリチュール、インド) を卒業し、電子通信工学の学士号を取得しました。

Appendix A. データ フレーム

STANDARD DATA FRAME

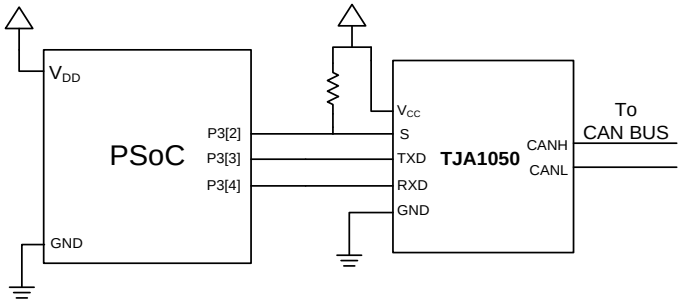


EXTENDED DATA FRAME



Appendix B. PSoC と TJA1050 CAN トランシーバとのインターフェース

Interfacing with TJA1050



改訂履歴

文書名: AN52701 – PSoC 3 および PSoC 5LP - コントローラー エリア ネットワーク (CAN) 入門

文書番号: 001-96361

版	ECN	発行日	変更内容
**	4669754	03/26/2015	これは英語版 001-52701 Rev. *I を翻訳した日本語版 001-96361 Rev.**です。
*A	6655981	01/08/2020	これは英語版 001-52701 Rev. *L を翻訳した日本語版 001-96361 Rev.*A 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
メモリ	cypress.com/memory
マイクロコントローラ	cypress.com/mcu
PSoC	cypress.com/psoc
電源用 IC	cypress.com/pmic
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	cypress.com/wireless

PSoC®ソリューション

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

サイプレス開発者コミュニティ

コミュニティ | Projects | ビデオ | ブログ | トレーニング | Components

テクニカル サポート

cypress.com/support

本書で言及するその他すべての商標または登録商標は、それぞれの所有者に帰属します。

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

© Cypress Semiconductor Corporation, 2009-2020. 本書面は, Cypress Semiconductor Corporation 及び Spansion LLC を含むその子会社 (以下「Cypress」という。) に帰属する財産である。本書面 (本書面に含まれ又は言及されているあらゆるソフトウェア若しくはファームウェア (以下「本ソフトウェア」という。) を含む) は, アメリカ合衆国及び世界のその他の国における知的財産法令及び条約に基づき Cypress が所有する。Cypress はこれらの法令及び条約に基づく全ての権利を留保し, 本段落で特に記載されているものを除き, その特許権, 著作権, 商標権又はその他の知的財産権のライセンスを一切許諾しない。本ソフトウェアにライセンス契約書が伴っておらず, かつ Cypress との間で別途本ソフトウェアの使用方法を定める書面による合意がない場合, Cypress は, (1) 本ソフトウェアの著作権に基づき, (a) ソースコード形式で提供されている本ソフトウェアについて, Cypress ハードウェア製品と共に用いるためにのみ, かつ組織内部でのみ, 本ソフトウェアの修正及び複製を行うこと, 並びに (b) Cypress のハードウェア製品ユニットに用いるためにのみ, (直接又は再販売者及び販売代理店を介して間接のいずれかで) 本ソフトウェアをバイナリーコード形式で外部エンドユーザーに配布すること, 並びに (2) 本ソフトウェア (Cypress により提供され, 修正がなされていないもの) が抵触する Cypress の特許権のクレームに基づき, Cypress ハードウェア製品と共に用いるためにのみ, 本ソフトウェアの作成, 利用, 配布及び輸入を行うことについての非独占的で譲渡不能な一身専属的ライセンス (サブライセンスの権利を除く) を付与する。本ソフトウェアのその他の使用, 複製, 修正, 変換又はコンパイルを禁止する。

適用される法律により許される範囲内で, Cypress は, 本書面又はいかなる本ソフトウェア若しくはこれに伴うハードウェアに関しても, 明示又は黙示をとわず, いかなる保証 (商品性及び特定の目的への適合性の黙示の保証を含むがこれらに限られない) も行わない。いかなるコンピューティングデバイスも絶対に安全ということはない。従って, Cypress のハードウェアまたはソフトウェア製品に講じられたセキュリティ対策にもかかわらず, Cypress は, Cypress 製品への権限のないアクセスまたは使用といったセキュリティ違反から生じる一切の責任を負わない。加えて, 本書面に記載された製品には, エラッタと呼ばれる設計上の欠陥またはエラーが含まれている可能性があり, 公表された仕様とは異なる動作をする場合がある。適用される法律により許される範囲内で, Cypress は, 別途通知することなく, 本書面を変更する権利を留保する。Cypress は, 本書面に記載のある, いかなる製品若しくは回路の適用又は使用から生じる一切の責任を負わない。本書面で提供されたあらゆる情報 (あらゆるサンプルデザイン情報又はプログラムコードを含む) は, 参照目的のためのみに提供されたものである。この情報で構成するあらゆるアプリケーション及びその結果としてのあらゆる製品の機能性及び安全性を適切に設計, プログラム, かつテストすることは, 本書面のユーザーの責任において行われるものとする。Cypress 製品は, 兵器, 兵器システム, 原子力施設, 生命維持装置若しくは生命維持システム, 蘇生用の設備及び外科的移植を含むその他の医療機器若しくは医療システム, 汚染管理若しくは有害物質管理の運用のために設計され若しくは意図されたシステムの重要な構成部分としての使用, 又は装置若しくはシステムの不具合が人身傷害, 死亡若しくは物的損害を生じさせるようなその他の使用 (以下「本目的外使用」という。) のためには設計, 意図又は承認されていない。重要な構成部分とは, それの不具合が装置若しくはシステムの不具合を生じさせるか又はその安全性若しくは実効性に影響すると合理的に予想できるような装置若しくはシステムのあらゆる構成部分をいう。Cypress 製品のあらゆる本目的外使用から生じ, 若しくは本目的外使用に関連するいかなる請求, 損害又はその他の責任についても, Cypress はその全部又は一部をとわず一切の責任を負わず, かつ Cypress はそれら一切から本書により免除される。Cypress は Cypress 製品の本目的外使用から生じ又は本目的外使用に関連するあらゆる請求, 費用, 損害及びその他の責任 (人身傷害又は死亡に基づく請求を含む) から免責補償される。

Cypress, Cypress のロゴ, Spansion, Spansion のロゴ及びこれらの組み合わせ, WICED, PSoC, CapSense, EZ-USB, F-RAM, 及び Traveo は, 米国及びその他の国における Cypress の商標又は登録商標である。Cypress のより完全な商標のリストは, cypress.com を参照すること。その他の名称及びブランドは, それぞれの権利者の財産として権利主張がなされている可能性がある。