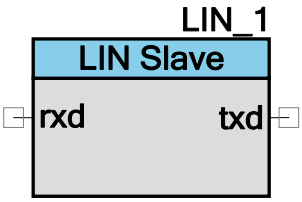
LIN スレーブ
1.0



特長

- 完全な LIN 2.1 または 2.0 スレーブノードの実装
- SAE J2602-1 仕様に準拠
- 自動ボーレート同期機能
- 診断クラス I スレーブノードを完全に実装
- 完全にトランスポート層をサポート
- バスの非アクティブ状態を自動的に検出
- 完全なエラー検出
- 自動コンフィグレーションサービス処理
- すばやく容易に構成するためのカスタマイザー
- *.ncf/*.ldf ファイルのインポートおよび*.ncf ファイルのエクスポート
- 構文チェックを伴う*.ncf/*.ldf ファイル用のエディタ

概要説明

LIN スレーブコンポーネントは、PSoC 3 および PSoC 5 デバイス上に LIN 2.1 スレーブノードを実装します。LIN 2.0 または SAE J2602-1 準拠オプションも利用できます。コンポーネントは、LIN バスでの通信のために必要なハードウェアブロック、およびアプリケーションコードが LIN バスコミュニケーションによって容易に通信するための API から成ります。コンポーネントは LIN 2.1 仕様によって規格化された API に適合する API を提供します。

このコンポーネントは柔軟性と使いやすさを提供します。コンポーネントにはカスタマイザーが提供され、LIN スレーブのすべてのパラメーターを簡単に設定することができます。

定義

このデータシートにある多くの定義は、LIN 2.1 仕様からのものです。この場合、用語を適切に理解するためには、LIN 2.1 仕様の指定されたセクションを参照してください。

入出力接続

このセクションでは、LIN スレーブの入出力接続について説明します。

TXD – 出力

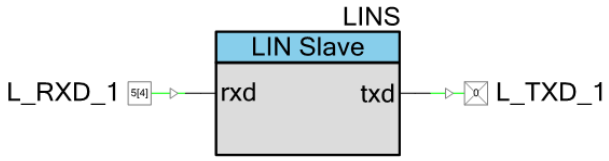
これはデジタル出力端子です。この端子の信号は、このLINノードがLINバスに送信するデータです。

RXD – 入力

これはデジタル入力端子です。この端子の信号は、LINバスの物理層の信号で、CMOSレベルです。この端子は通常、TXD端子から出力される信号も受信することに注意してください。これはLIN物理層トランシーバーが内蔵のループバックがあり、他のLINノードからのものでも自分のLINノードからのものでも、バス上のすべての信号を受信するからです。

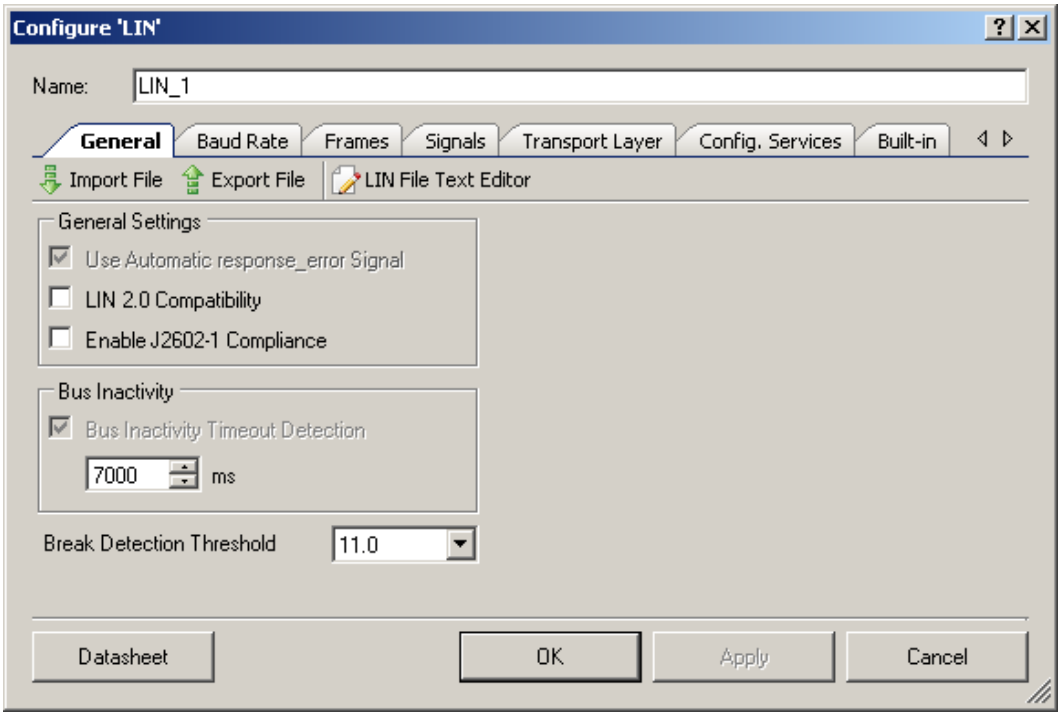
回路図マクロ情報

初期設定で、PSoC Creator Component Catalog には、LIN コンポーネントの回路図マクロが入っています。このマクロは接続および構成済みのピンコンポーネントが含まれています。初期設定状態のコンポーネントの回路図マクロを下に示します。



コンポーネントパラメータ

LIN スレーブコンポーネントを回路図上にドラッグし、ダブルクリックして **Configure LIN** ダイアログを開きます。



General タブ

Use Automatic response_error Signal（自動レスポンスエラー信号）

タブ上のこのチェックボックスは、自動エラー信号の選択設定です。このチェックボックスは常に選択されているので、1 ビットの信号はカスタマイザーの **Signals** タブに自動的に追加されます。この信号のデフォルト名は response_error です。レスポンスエラーが起きると、常に、コンポーネントは自動的にこの信号をセットします。マスターへの送信が成功すると、コンポーネントは自動的にこの信号をクリアします。この信号は LIN 2.1 仕様から要求されているように、LIN マスターにレスポンスエラーを通知します。

LIN 2.0 Compatibility（LIN 2.0 との互換性）

このオプションは、このコンポーネントが LIN 2.0 仕様と互換にするか否かを選択します。チェックボックスのステータスは、カスタマイザーの他の領域に影響を与えます。

Enable J2602-1 Compliance（J2602-1 準拠機能イネーブル）

SAE J2602-1 仕様は LIN 2.x 仕様と同様のものです。それは LIN 2.x 要件に制限を追加します。しかしながら、このコンポーネントによってサポートされるいくつかの特別な機能があり、このコンポーネントを 2602-1 仕様に準拠させます。チェックボックスのステータスは、カスタマイザーの他の領域に影響します。

Bus Inactivity Timeout Detection（バス非アクティブタイムアウト検出）

このオプションは、バス非アクティブ機能を使用や、非アクティブ時間を制御します。バスの非アクティブ状態が指定時間経過した後、対応するステータスビットがセットされます。このビットの値は I_ifc_ioctl()関数の L_IOCTL_READ_STATUS オペレーションによって取得することができます。詳細は関数の詳細を参照してください。

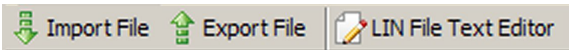
ブレーク検出しきい値

このオプションはスレーブノードブレーク検出しきい値を設定します。デフォルト値は 11 ローカルスレーブビット時間ドミナント状態（訳注：Low レベル）です。ブレーク検出しきい値選択基準の詳細については、LIN 2.1 仕様のセクション 2.3.1.1 を参照してください。

General ツールバー

General タブの上にツールバーがあります。このツールバーはファイルを使う操作へのアクセス手段を提供します。

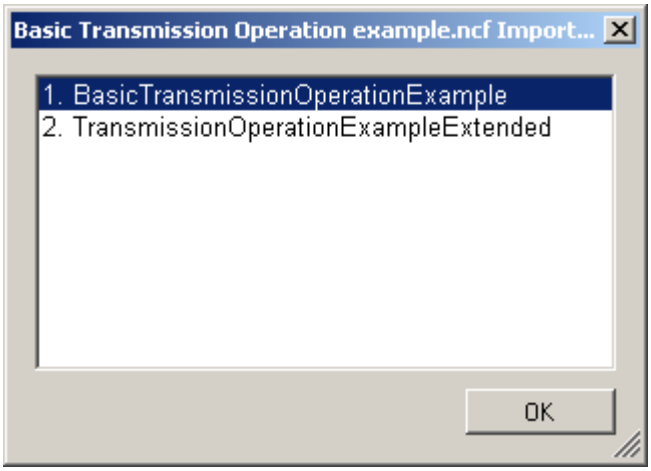
図 1. General ツールバー



Import File（ファイルのインポート）: このボタンをクリックすると、LIN 記述ファイル(LIN Description File:LDF)またはノード機能ファイル(Node Capability File：NCF)のインポートができます。インポートされたファイルはカスタマイザーの設定を、NCF/LDF ファイルの既存ノードのリストから選択されたノードの設定と一致するようにコンフィギュレーションします。

インポートされたファイルの構文が適切な場合、使用可能なノードのリストが表示されます。このリストの一例を図 2 に示します。使用可能なノード機能を 1 つ選択し、インポートしてください。

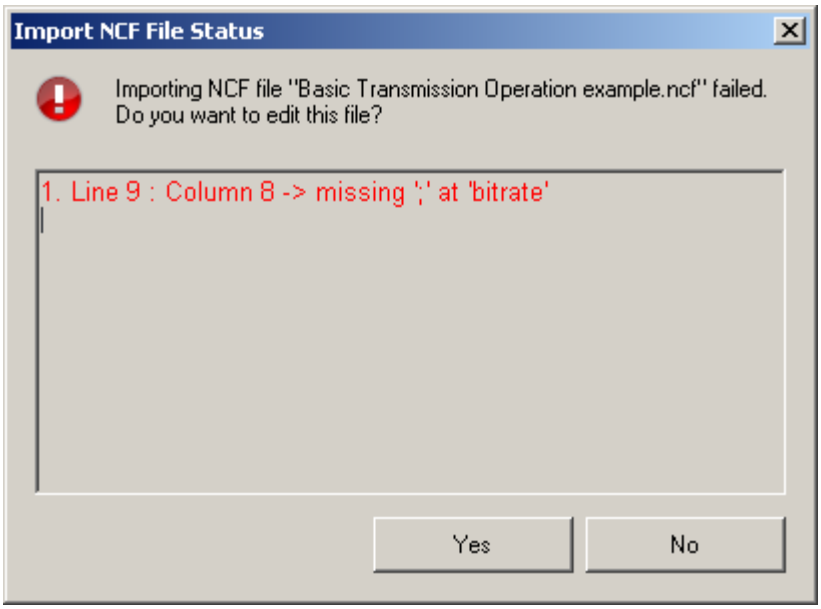
図 2. インポートする NCF ファイルの使用可能なノードのリスト



.ncfと.ldfファイルの構文は、LINノード機能言語仕様(Revision 2.1)およびLINコンフィギュレーション言語仕様(Revision 2.1)のそれぞれに基づいて検証されています。

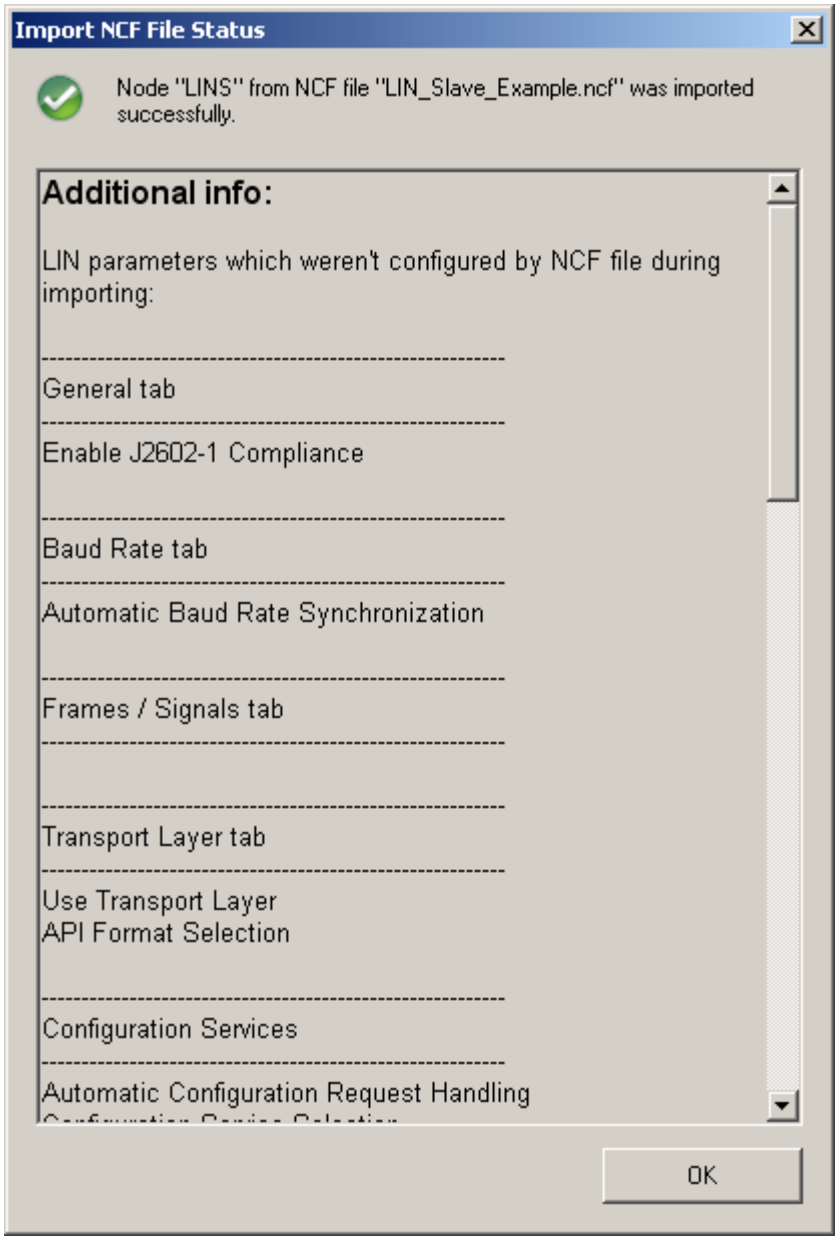
インポートされたファイルにエラーがある場合、図 3 のようなダイアログウィンドウが表示されます。この場合、2 つの対処方法があります。インポートされたファイルを編集し、LIN エンハンスドエディターツール(詳細は LIN ファイルテキストエディタを参照)を使用してエラーを修正するか、または **No** ボタンをクリックしてインポートをキャンセルします。

図 3. NCF ファイルのインポート失敗例



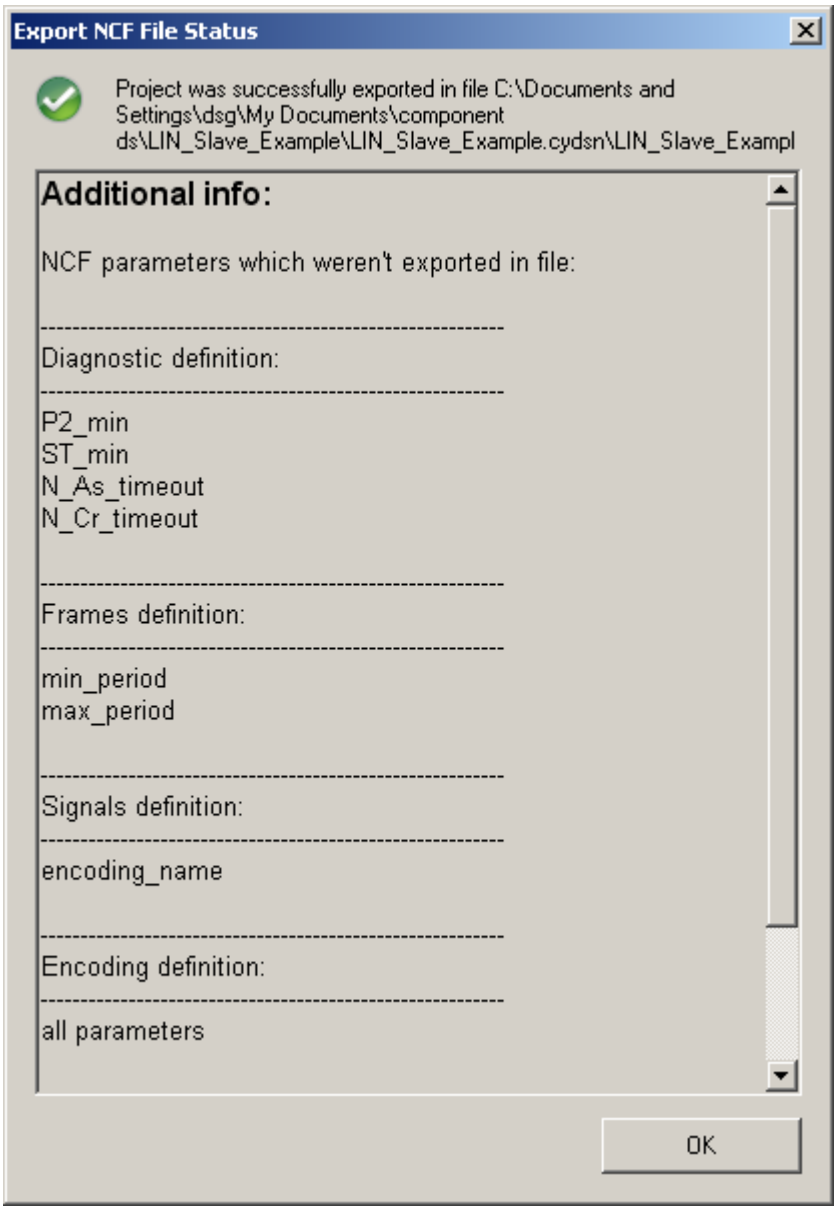
インポートするノードを選択し、カスタマイザーへのインポートが完了したら、インポート結果を説明するダイアログボックスが表示されます（図 4 参照）。インポート結果はインポート中に影響されなかった LIN スレーブコンポーネントパラメータが含まれます。

図 4. NCF ファイルインポート情報



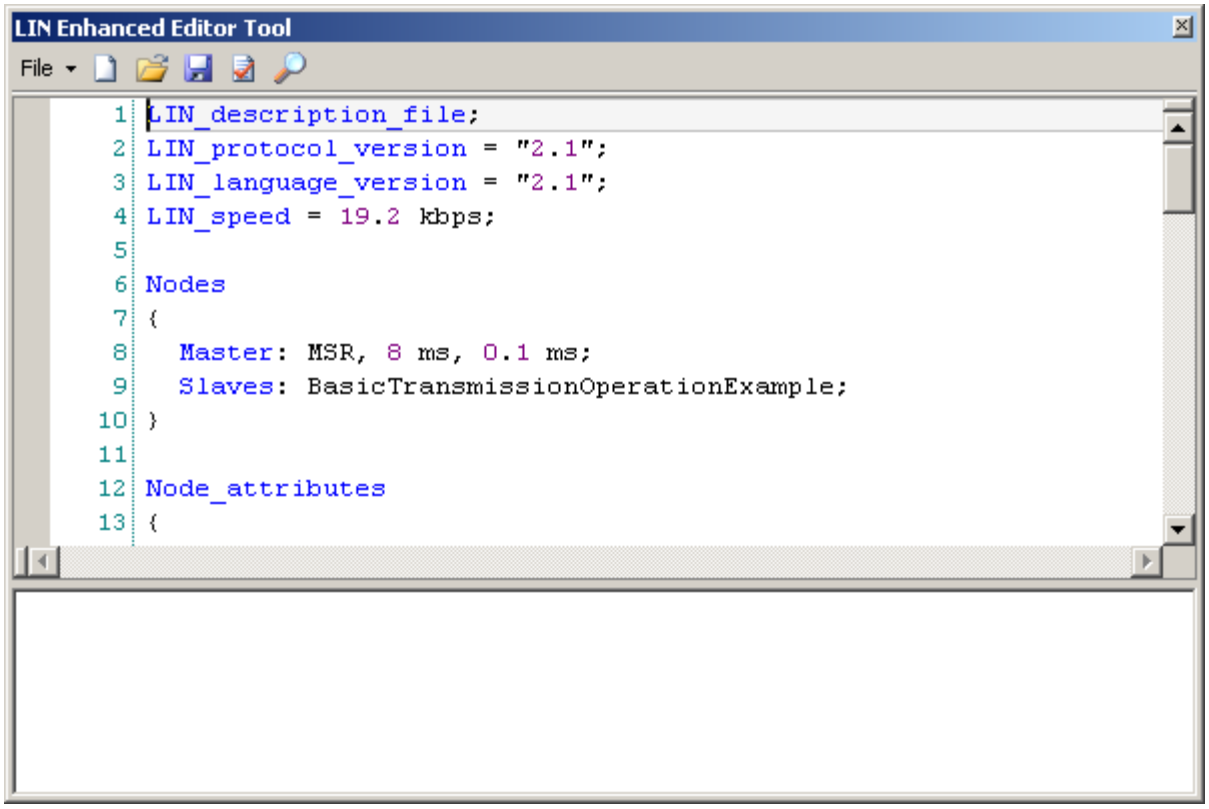
Export File（ファイルのエクスポート）: このツールにより、コンポーネント構成についての情報をノード機能ファイル(NCF)に保存することができます。

図 5. NCF ファイルエクスポート情報



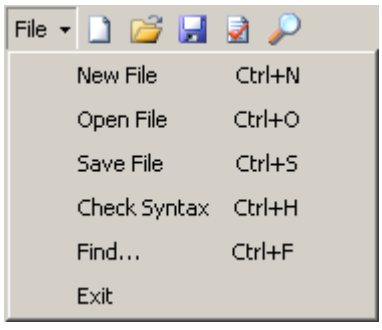
LIN ファイルテキストエディタ: このツールは NCF/LDF ファイルの構文細工や編集をしたり、構文の検証をするために使用されます。*.ncf ファイルの構文は *LIN ノード機能言語仕様* (Revision 2.1)に従って検証されます。*.ldf ファイルの構文は *LIN コンフィグレーション言語仕様* (Revision 2.1)に従い検証されます。

図 6. LIN ファイルテキストエディタツール



LIN Enhanced Editor Tool(**LIN エンハンスドエディタツール**)ウィンドウの上にツールバーがあります(図 7)。

図 7. LIN ファイルテキストエディタツールバー



New File: 選択した LIN ファイルタイプの新規ファイルを作成します。

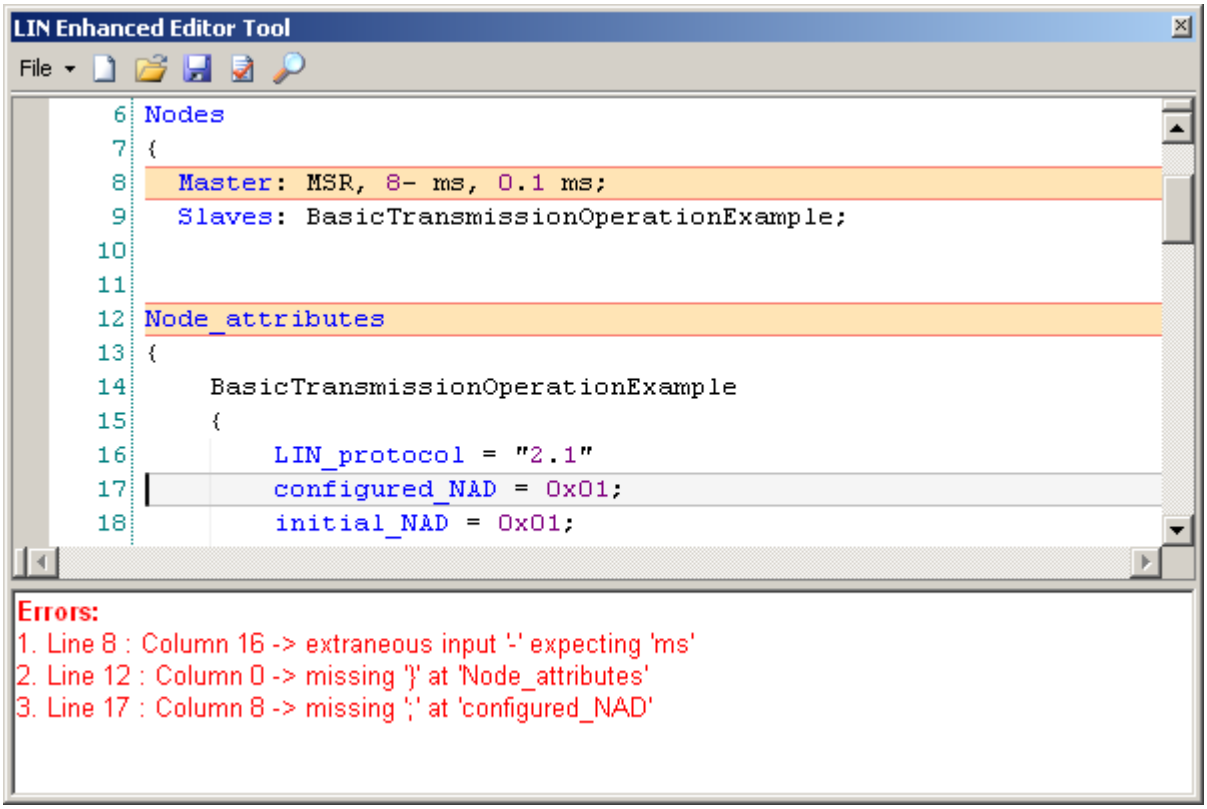
Open File: 指定した既存の LIN ファイルを開きます。

Save File: 作成した LIN ファイルを指定した場所に保存します。

Check Syntax: このコントロールは **.ncf*/**.ldf* ファイルの構文が正しいか否かチェックします。構文エラーがある場合、エラーはエディタウィンドウの出力領域に一覧表示され、行と列の番号が場所を示し、短いエラーの説明があります（図 8）。エラーが含まれるコードの行は赤でハイライトされています。

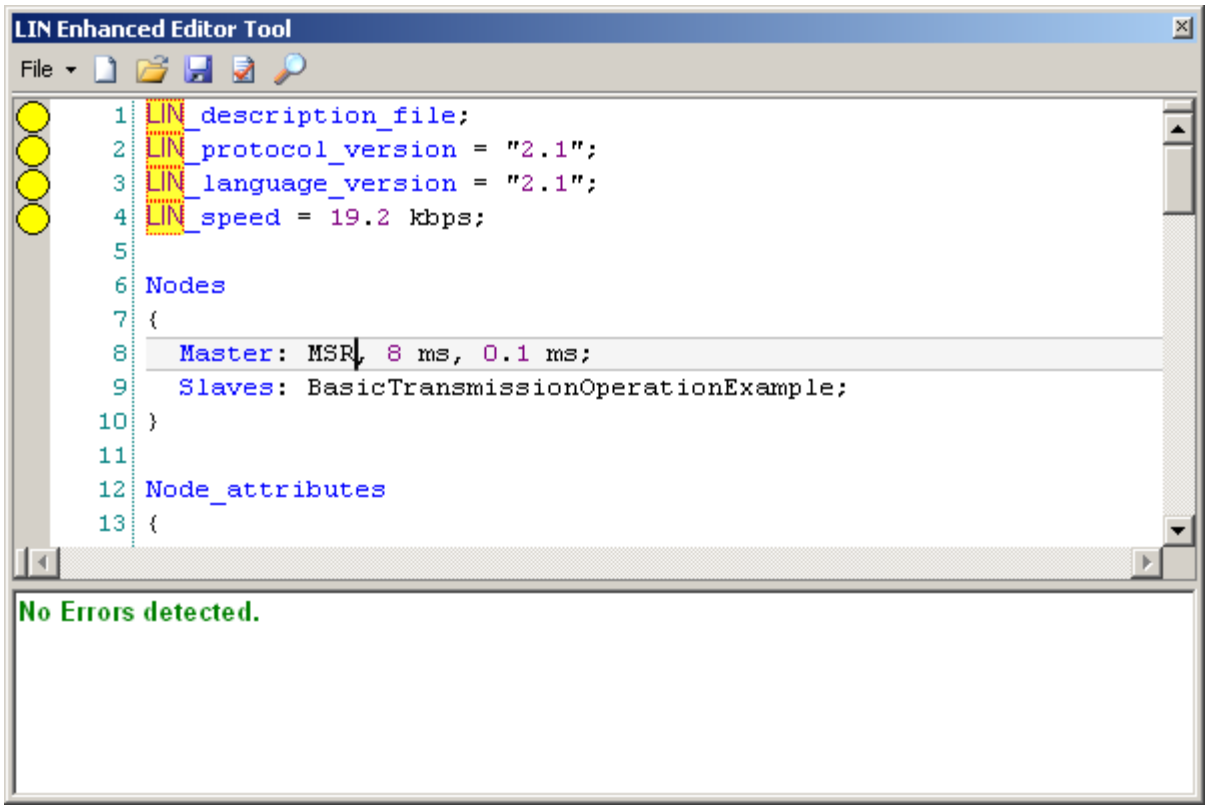
出力領域のエラー行をダブルクリックすると、ファイルのエラーが含まれている行に移動します。

図 8. LIN ファイル構文チェック



Find: このツールは LIN ファイルの検索フィールドで指定された用語を検索します。**Find Next** ボタンは次に一致するものを探します。ツールの **Mark Line** チェックボックスが選択されている場合、必要な用語が含まれている行は **Find All** ボタンをクリックしてから、黄色の円でラベル付けされます。**Style found token** チェック ボックスは **Find All** ボタンをクリックしてから、検索された黄色のトークンのハイライトをイネーブルまたはディスエーブルします。図 9 参照。**Clear** ボタンはハイライトされているすべてのトークンを削除します。

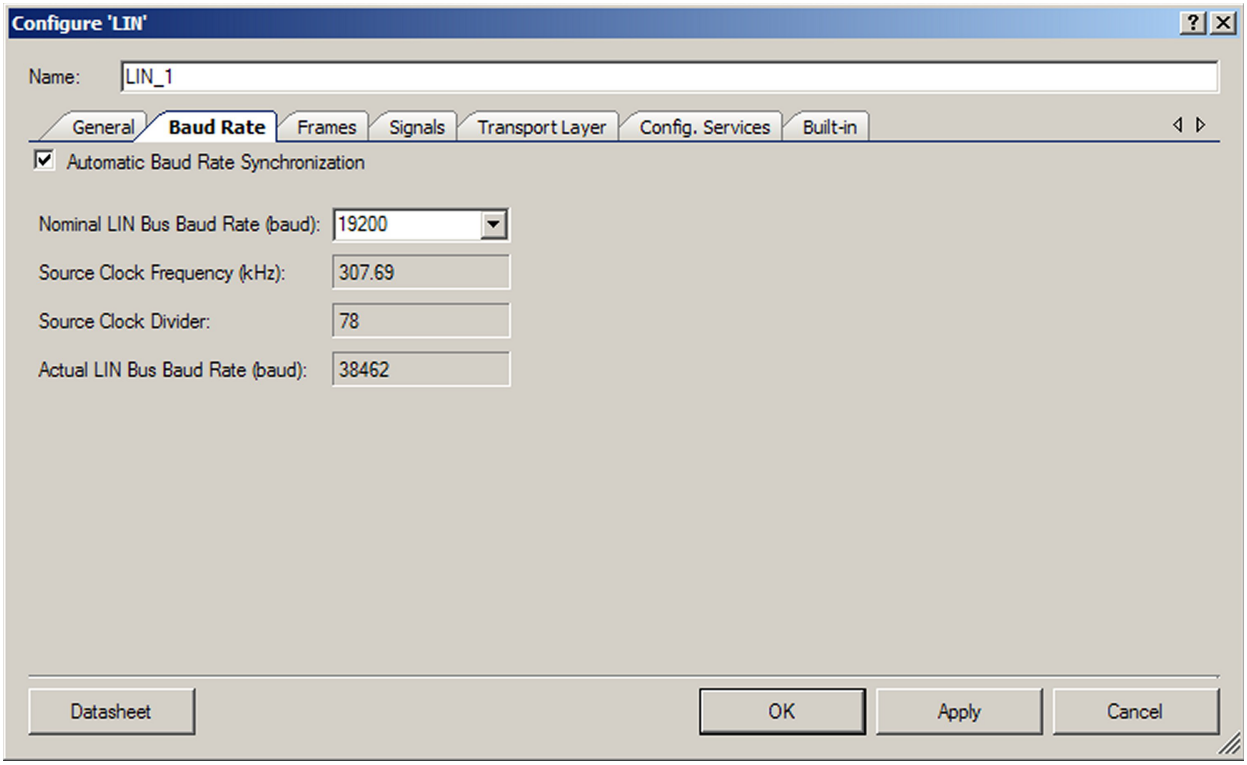
図 9. LIN ファイル検索結果



すべてのツールは LIN エンハンスドエディタの **File** メニュー（図 6 参照）および適切なツールバーコマンドから使用可能です。

Baud Rate タブ

図 10. LIN ダイアログ、ボーレートタブの構成



Automatic Baud Rate Synchronization（自動ボーレート同期機能）

このオプションは自動ボーレート同期機能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、このオプションはイネーブルです。

このオプションがイネーブルの場合、コンポーネントは各 LIN フレームヘッダーの同期バイトフィールドからバスの正確なボーレートを測定します。

このオプションがディスエーブルの場合、コンポーネントは同期バイトフィールドからのボーレートを測定しません。代わりに、同期バイトフィールドを 0x55 データバイトとして受け取ります。

LIN 2.1 仕様で要求されているように、周波数偏差が±1.5%以下の LIN スレーブノードは、各フレームの同期バイトフィールドを測定する、自動ボーレート同期機能を使用する必要がありません。しかしながら、LIN スレーブノードの周波数偏差が±1.5%を超えている場合、スレーブノードは各フレームの同期バイトフィールドを測定するために、自動ボーレート同期機能を使用する必要があります。

このため、周波数偏差の仕様は、BusClk に供給されるクロックソース（これは通常内部メイン振動子(IMO)です）により、チェックされなくてはなりません。

公称 LIN バスボーレート

この LIN スレーブノードが作動する必要がある公称 LIN バスボーレートを入力します。最大値は 20000 ボーで、最小値は 1000 ボーです。カスタマイザーはこの範囲外のボーレートの選択を許可しません。ドロップダウンリストにある値は 19200、10417、9600 および 2400 です。しかしながら、コンボボックスには 1000～20000 の間の任意の値を入力することができます。**公称 LIN バスボーレート**が修正された場合、**Apply** ボタンを押して **Source Clock Frequency（ソースクロック周波数）**、**Source Clock Divider（ソースクロック分周器）**および **Actual LIN Bus Baud Rate（実際の LIN バスボーレート）**フィールドのために新しい値を取得します。

Source Clock Frequency（ソースクロック周波数）

これは、データ送信に使用される、8 倍オーバーサンプルされているクロック周波数です。

Source Clock Divider（ソースクロック分周器）

これは**ソースクロック周波数**で指定されているクロック周波数を BusClk から取得するために使用されるクロック分周器の値です。

Actual LIN Bus Baud Rate（実際の LIN バスボーレート）

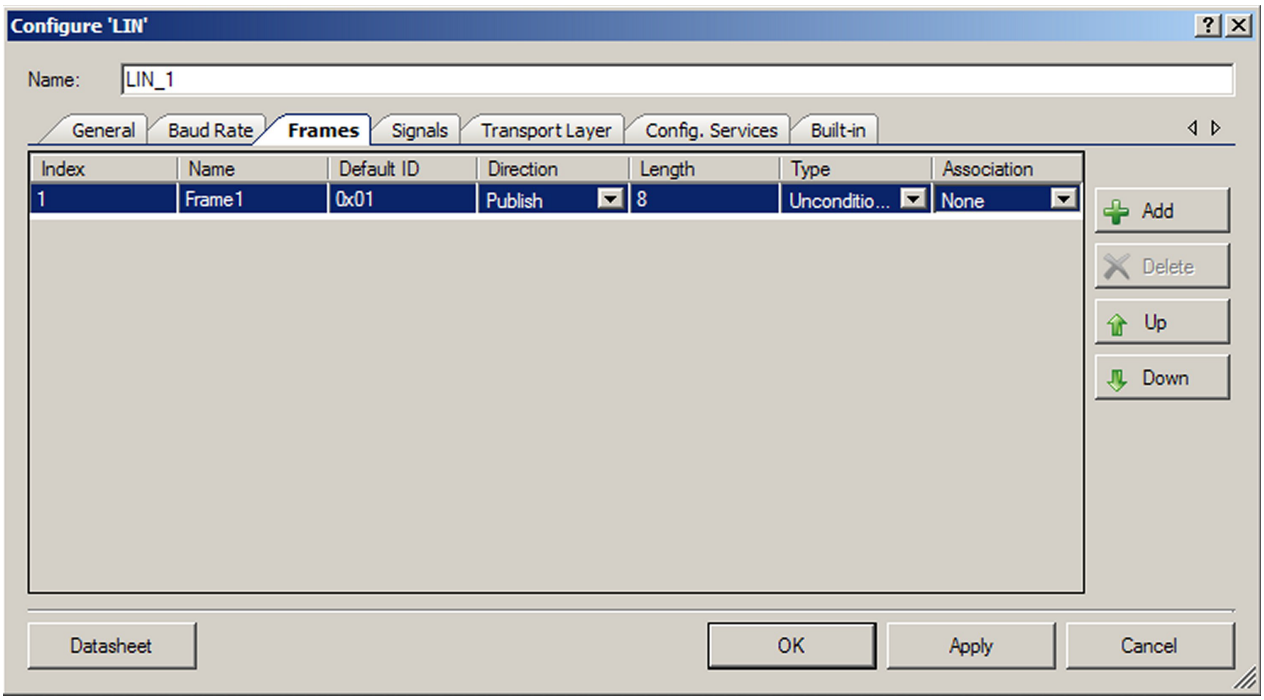
ここにバスボーレートの実際の値が表示されています。LIN スレーブはこのボーレートで動作します。BusClk 値を修正して、**公称 LIN バスボーレート**を**実際の LIN バスボーレート**と等しくすることができます。

Frame（フレーム）タブ

このタブは、バス上にマスターが送信した PID 値に対する LIN スレーブの応答方法をコンフィギュレーションするために使用されています。

このタブ上で構成されている設定は、コンポーネント API および ISR コードを適切に発生するために使用されます。作動中、LIN スレーブはフレーム ID を持つ PID を受け取り、それが LIN スレーブ（コンポーネント）の応答方法を決定します。

図 11. LIN ダイアログ、フレームの構成タブ



フレームコンフィギュレーションテーブル

コンフィギュレーションテーブルはこのタブの中央にあります。表には行と列があります。

各行が 1 つの LIN フレームに対応します。このタブは「ユーザー」LIN フレームのみ表示します。MRF と SRF フレームはこのコンポーネントによりサポートされていますが、この表には表示されません。

データフィールドには 8 つの可能性のある列があります。

Index 列のフィールドは、使用されている各フレームの順番を示します。これらの数字は直接変更することができません。

Name 列のフィールドには、各フレームの名称を入力します。C のプログラム中で有効な任意の文字列を入力することができます。各フレームの番号は一意である必要があります。

Default ID 列のフィールドは、マスターによるコンフィギュレーション要求の前にフレームが使用するフレーム ID を定義するために使用されます。これらのフレーム ID はダイナミックである（訳注：動作中に変化することがある）ことに注意してください。言い換えると、LIN マスターはランタイム中にフレーム ID を再設定することができます。これらのセルに 0x00～0x3B の値を入力する必要があります。これらの値は 16 進法または 10 進法で入力することができます。

Message ID 列は図 11 に表示されていません。これは通常非表示であるためです。この列はカスタマイザーの **General** タブの **LIN 2.0 Compatibility** チェックボックスが選択された場合にのみ使用できます。任意の 16 ビットの値を入力することができます。これらの値は 16 進法または 10 進法で入力することができます。全てのメッセージ ID は一意である必要があります。また、この表に入力されたメッセージ ID は、LIN クラスタ全体に対して一意

である必要があります。例えば、他の LIN スレーブのメッセージ ID が 0x000F のフレームがある場合、このコンポーネントはメッセージ ID が 0x000F のフレームがあってはなりません。

Direction 列のフィールドは、フレームのデータがどの方向で送信されるか（このスレーブに対して）定義します。**Publish** はデータ送信を意味し、**Subscribe** はデータの受信を意味します。

Length 列のフィールドは、各フレームに対して送受信されるバイト数を定義します。1～8 の値が有効です。

Type 列のフィールドは、LIN フレームの種類を定義するために使用されます。LIN スレーブデバイスに対して、2 種類のフレームがあります。**Unconditional（無条件）**と **Event-Triggere（イベントトリガ）**です。フレームが subscribe フレームの場合、イベントトリガタイプを選択することができません。この場合、セルは修正できません。セルを **Event-Triggere（イベントトリガ）**から **Unconditional（無条件）**に変更する場合、フレームの名前がその列の任意のセルに表示されるとき、このフレームの名前を **Association** 列において **None** に変更する必要があります。

Association 列のフィールドは、無条件フレームをイベントトリガーされるフレームに関連付けするために使用されます。LIN 仕様に準拠するには、イベントトリガーされるフレームは、関連付けされている無条件フレームが 1 つ以上必要です。そのため、**Association** 設定はイベントトリガーフレームとまだ関連付けされていない任意の無条件フレームのフレーム名の選択を可能にします。この設定の有効な値は、既存の関連付けされていない無条件フレームの名前です。イベントトリガーフレームは 1 つの無条件フレームとのみ関連付けできます。その結果、これらのセルの 1 つが無条件フレームの名前の場合、この無条件フレームの名前は他の行では使用できません。無条件フレームと関連付けされているイベントトリガーフレームは、関連付けされている無条件フレームと同じ長さと方向でなければなりません。そのため、これらの基準が適用される無条件フレームの行にのみ、イベントトリガーフレームの名前が表示されます。カスタマイザーの **OK** ボタンをクリックする、または他のタブをクリックしてこのタブを終了すると、カスタマイザーは無条件フレームに関連付けされていないイベントトリガーフレームがないかどうかチェックします。

注: フレームの合計数は 60 を超えることはできません。すべてのフレームの合計サイズは 256 バイトまでに制限されています。

Tab ボタン

このタブでは 4 つのボタンが使用できます。

Add ボタンは表に新しいフレームを追加します。

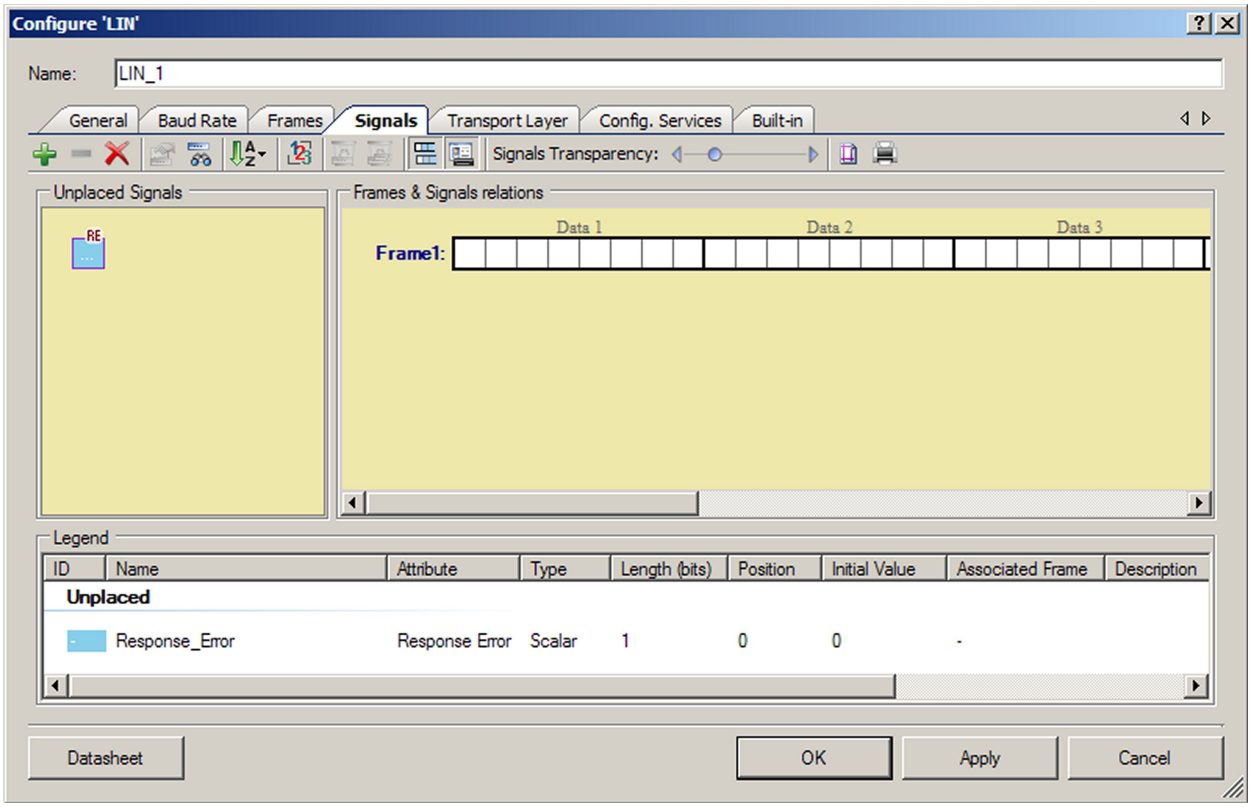
Delete ボタンは表から現在選択されているフレームを削除します。インデックス番号フィールドは適宜に変更されます。フレームがこのタブ上で削除された場合、その中にある任意の信号（**Signals** タブで設定されたもの）は、**Unplaced Signals** 領域に移動されます（Signals タブセクションの Sort Signals ボタンを参照してください）。

Up および **Down** ボタンを使用して、各フレームの**インデックス**番号を変更することができます。

Signals タブ

カスタマイザーのタブは、LIN フレームに入れられた「信号」を定義するために使用されます。

図 12. Configure LIN Dialog, Signals（LIN ダイアログ、Signals の構成）タブ



フレームと信号の関係

Signals タブのグラフィック領域は、カスタマイザーで定義したフレームと信号のインタラクティブな図を表示します。

フレームのグラフィック: 1 つのフレームグラフィックはカスタマイザーの **Frames** タブで定義されているそれぞれのフレームを示しています。

信号のグラフィック: 各信号グラフィックが LIN スレーブに対して定義されている 1 つの信号を代表します。信号のグラフィックはソリッドバーで表示されます（図 12 参照）。信号はドラッグアンドドロップして、フレーム上に配置することがができます。これらの信号はフレームのビットまたはバイトを占有します。

信号をクリックすると、その信号が選択されます。信号の上にマウスカーソルをもってくると、その信号に該当する情報がツールチップに表示されます。

Unplaced Signals（未配置信号）

このグラフィック領域は、追加された後、まだ配置されていない信号が保存されるテンポラリ領域領域です。信号は **Unplaced Signals（未配置信号）**領域と **Frames & Signals relations（フレームと信号の関係）**領域の間で行き来させることができます。

注：Frames タブ上でフレームが削除されると、中にある任意の信号（**Signals** タブで設定）は **Unplaced Signals（未配置信号）**領域に移動します。

response_error

1 ビットの response_error 信号はカスタマイザーの **Signals** タブに自動的に追加されます。response_error 信号の名前を変更することはできますが、**Signals** タブから削除することはできません。

response_error 信号のインスタンスは 1 つだけ可能で、その名前はこのコンポーネントに対して一意である必要があります。response_error エラー信号はブール信号（Boolean signal）で、LIN スレーブによって発行されるフレームの任意の場所に置くことができます。

この信号の目的は、LIN マスターにステータス情報を報告することです。

この信号に関する追加情報は、LIN 2.1 仕様のセクション 2.7.3「クラスタへの報告」を参照してください。

信号ツールバー

Signals タブの上にツールバーがあります。このツールバーはタブ上の信号を管理する簡単な方法を提供します。

図 13. Signals Toolbar（信号ツールバー）



1. **Add/Delete** ボタン

   Add Signal ボタンは **Unplaced Signals（未配置信号）**領域に信号を追加します。**Delete Signal** ボタンはコンポーネントから選択した信号を削除します。**Delete All Signals** ボタンはすべての既存の信号を削除します。

2. **Signal Properties** ボタン

   このコントロールは、選択した信号に対して **Signal Properties（信号プロパティ）**ウィンドウを開きます。このウィンドウは信号のプロパティを変更するために使用できます。信号のプロパティウィンドウは、信号をダブルクリックしてアクセスすることもできます。

3. **Find Signal** ボタン

   このボタンは、特定の信号を検索することを可能にします。

4. **Sort Signals** ボタン

   このボタンは **Unplaced Signals**（**未配置信号**）領域の信号を並べ替えします。信号は名前、長さまたは種類で並べ替えできます。

5. **Renumber Signals** ボタン

   このボタンは、昇順で信号インデックス値の番号をつけ直します。

6. **Move** ボタン

   Unplace Signal ボタンは選択した信号を **Frames & Signals relations**（**フレームと信号関係**）領域から **Unplaced Signals**（**未配置信号**）領域へと移動します。

   Unplace All Signals ボタンは、すべての信号を **Unplaced Signals**（**未配置信号**）領域に移動します。

7. **Show/Hide Event-triggered frames** ボタン

   このボタンは、**Frames & Signals relations**（**フレームと信号関係**）領域のイベントトリガーフレームに対応するフレームグラフィックを表示/非表示することができます。

8. **Show/Hide Legend** ボタン

   このボタンは、信号のプロパティを説明する説明文領域を表示/非表示することができます。

9. **Signals Transparency**（**信号透明度**）スライダー

   このスライダーは、信号グラフィックの透明度を設定します。

10. **Print** ボタン

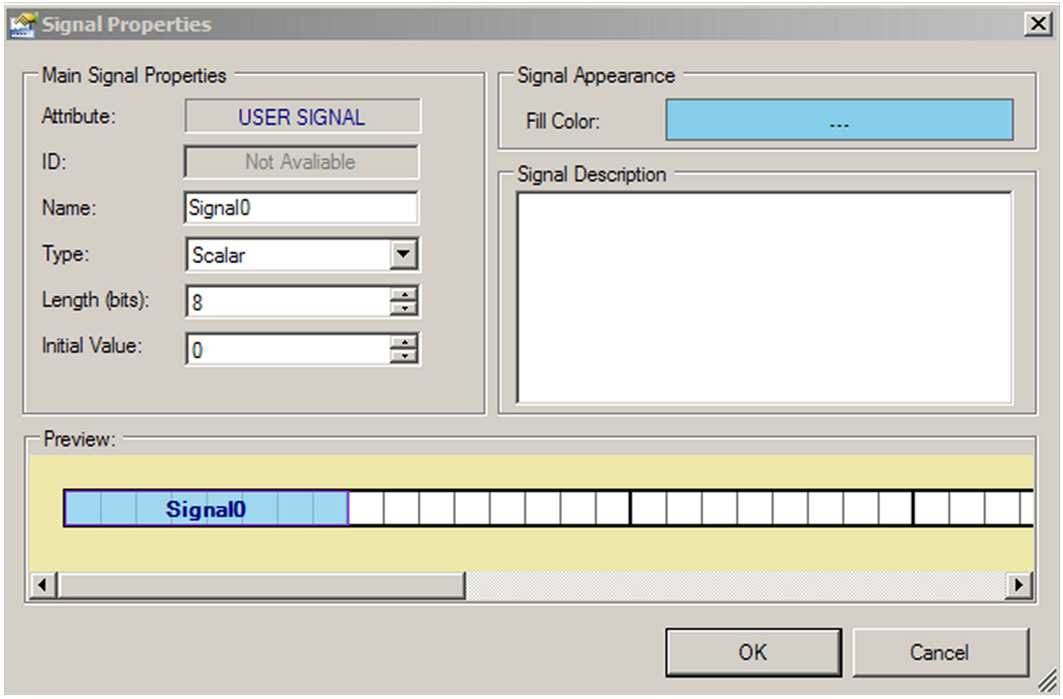
    これらのボタンは **Frames & Signals relations**（**フレームと信号関係**）領域を印刷します。

信号プロパティウィンドウ

信号の追加

ツールバーには **Add Signal** ボタンがあります。このボタンは、信号プロパティオプションと共に新しいウィンドウを表示させ、設定することができます（図 14 を参照）。プロパティが設定された後、新しい信号が追加されます。このウィンドウで構成できる様々な信号プロパティは、このセクションで説明されています。

図 14. 信号プロパティウィンドウ



Name（名前）

Name プロパティは、信号の名前をつけるために使用されます。デフォルト信号名は Signalx で、「x」は信号のインデックス番号と等しくなります。信号用として入力した名前は、C のプログラム中で使用できるシンボル名である必要があります。

Type（種類）

このプロパティは、信号のタイプを選択するために使用されます。LIN 2.1 仕様には、2種類の信号が定義されています。**スカラ**信号の長さは 1～16 ビットで、**ByteArray** 信号の長さは 1～8 バイトです。

Length (長さ)

このプロパティは、信号の長さを選択するために使用されます。スカラ信号の長さは 1～16 ビットです。
ByteArray 信号の長さは 1～8 バイトです。

Initial Value（初期値）

このプロパティは、信号の初期値を選択するために使用されます。この値は 10 進法で入力する必要があります。

Fill Color（塗りつぶし色）

このコントロールは、信号グラフィックの色を選択するのに使用されます。

Signal Description（信号の説明）

このプロパティは、任意の該当する説明または信号に関係するその他の情報を入力するために使用できます。

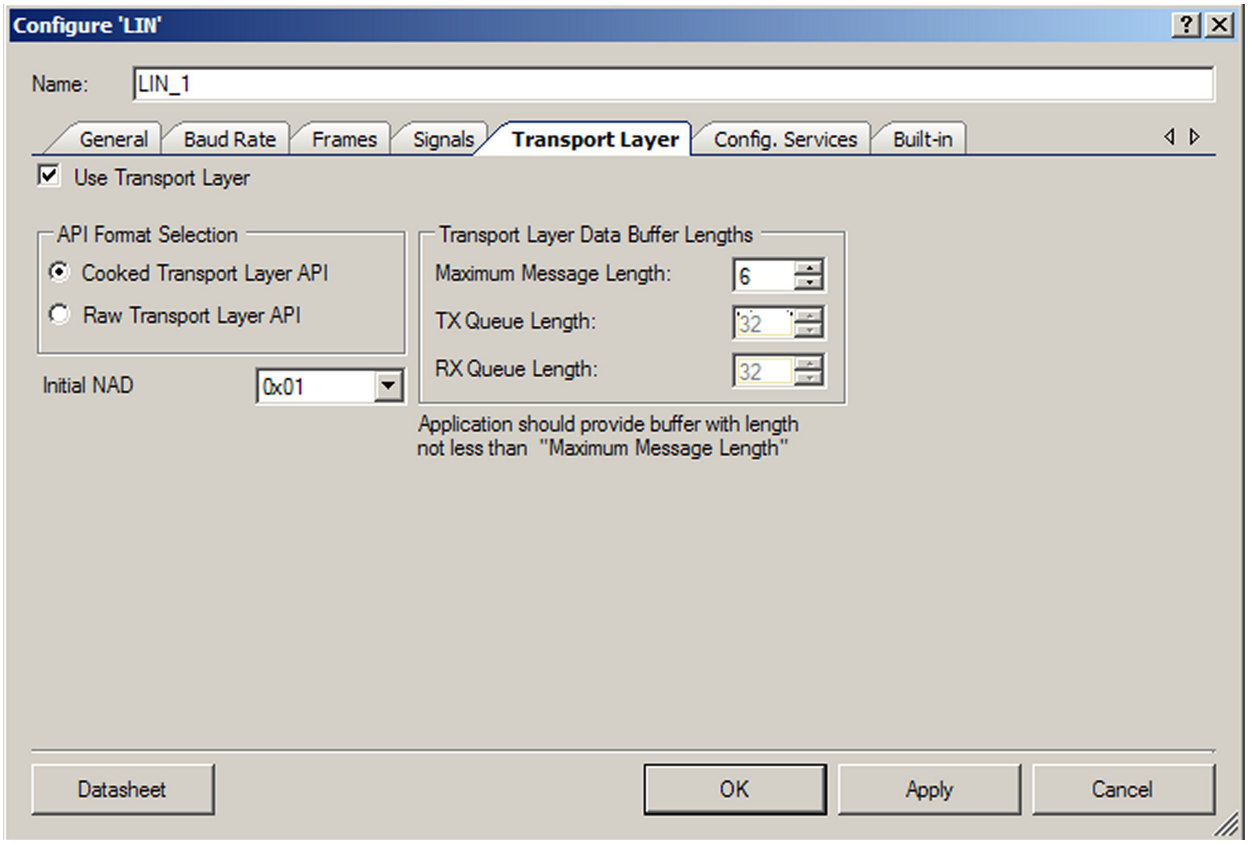
Preview（プレビュー）

このグラフィック領域は、信号が追加されたときにどのように見えるか示します。

Transport Layer（トランスポート層）タブ

Transport Layer（トランスポート層）タブ表示が図 15 に表示されています。

図 15. Configure LIN Dialog, Transport Layer（LIN ダイアログ、トランスポート層の構成）タブ



Use Transport Layer（トランスポート層を使用）

Use Transport Layer チェックボックスが選択されていない場合、スレーブノードはトランスポート層をサポートしません。選択されている場合、スレーブノードコンポーネントはトランスポート層をサポートします。トランスポート層の詳細については、LIN 2.1 仕様を参照してください。

API Format Selection（API フォーマットの選択）

このコントロールは、トランスポート層 API 関数のフォーマットを選択するために使用されます。**Cooked Transport Layer API** オプションと **Raw Transport Layer API** オプションがあります。通常、LIN スレーブアプリケーションには cooked 形式が推奨されます。cooked 形式は各メッセージに対して 1 つの API だけを利用して、トランスポート層メッセージの送受信をするために使用されます。raw 形式は、各フレームに対して 1 つの API 関数呼び出しを使用して、トランスポート層メッセージを成す各フレームを送受信するために使用されます。

トランスポート層 API の 2 つの形式は、LIN 2.1 仕様書のセクション 7.4 で定義されています。

Initial NAD（初期 NAD）

このフィールドは、スレーブノードのネットワークアドレス(NAD)を選択するために使用されます。NAD は MRF および SRF フレームで使用され、クラスタの 1 つの特定のスレーブノードに対応します。このフィールドはノードの初期 NAD を選択するために使用されます。スレーブノードの NAD は動作中に変更することができます。

初期設定では、初期 NAD の値は 0x01～0xFF の範囲です。0x00 の NAD 値は、「Go To Sleep（スリープ状態に移行）」コマンドのために予約されています。NAD 値 0x7E は、診断サービスに使用される「Functional（機能）NAD」として確保されています。NAD 値 0x7F は、「wildcard（ワイルドカード）」NAD として確保されています。そのため、カスタマイザーはこのフィールドに 0x00、0x7E または 0x7F を入力することができません。

J2602-1 Compliance チェックボックスがチェックされている場合、トランスポート層タブの初期 NAD 値は 0x60～0x6F に限られています。デフォルト値は 0x60 です。初期値の範囲はカスタマイザーの **Frames** タブ上で使用されているフレーム数に基づいて、さらに制限されています。詳細については表 1 を参照してください。

表 1. スレーブノードで使用されているフレーム数に基づいた、初期 NAD の制約

フレーム数	使用可能な初期NAD値
1～4	0x60～0x6F
5～8	0x60、0x62、0x64、0x66、0x68、0x6A、0x6C、0x6E、0x6F
9～16	0x60、0x64、0x68、0x6E、0x6F
17以上	0x6E、0x6F

最大メッセージ長

このプロパティは、スレーブノードがサポートする最大トランスポート層メッセージ長を選択するために使用します。フレームを1つしか使用しないメッセージ中に最大6バイトのトランスポートレイヤメッセージデータバイトがあるので、最小値は 6 です。このコンポーネントは長さが 4095 バイトまでのトランスポート層メッセージのみをサポートします。実際のトランスポート層メッセージバッファは、ノードのアプリケーションコードにあることに注意してください。

TX Queue Length/RX Queue Length（TX キューの長さ/RX キューの長さ）

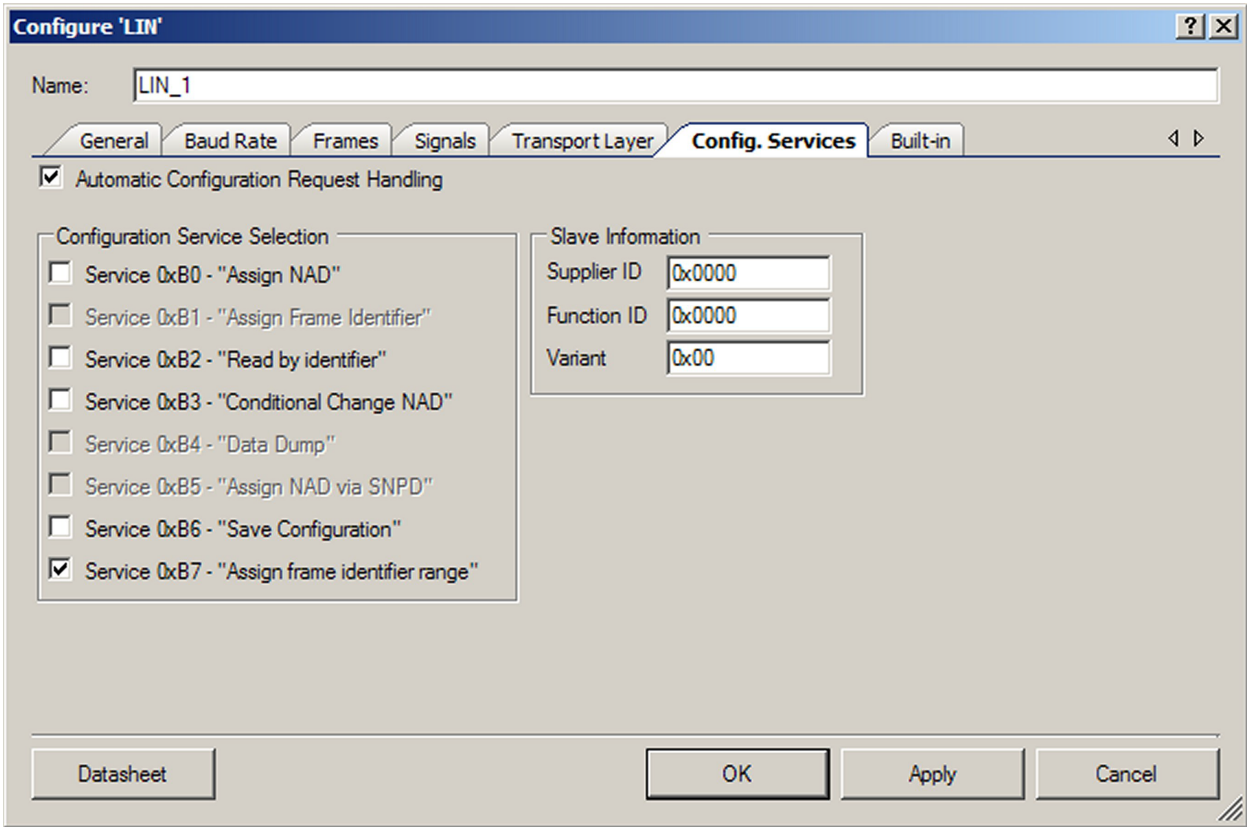
これらのプロパティは **Raw Transport Layer API** 形式が選択されたときのみ使用できます。Raw API フォーマットを使用する際は、送受信するフレーム応答データをバッファするメッセージ「キュー（queue）」が存在します。スレーブがキューを迅速に更新できない場合には、キューの長さを大きくしてください。スレーブがキューを非常に迅速に更新できる場合には、キューを短くして RAM の使用率を低くすることができます。コンポーネントは、8 バイト刻みの 8~2048 バイト長のキューをサポートします。各キューのデフォルト サイズは 32 バイトです。

Config. Services(設定 サービス)タブ

LIN 2.1 仕様書はスレーブがサポートしなければならない設定 サービス要求を定義します（LIN 2.1 仕様書によれば一部は必須であり、一部はオプションです）。このコンポーネントはすべての必須要求と一部のオプションサービス要求をサポートします。

合計 8 個の設定 サービス要求があります（0xB0 ～ 0xB7）。LIN 2.1 仕様書の表 4.6 にこれらのサービスのリストがあります。このコンポーネントはその一部をサポートします。サポートされるサービスを個々にディスエーブルまたはイネーブルとする選択ができます。設定 サービス要求は、LIN 2.1 仕様書のセクション 4.2.5 に説明されています。

図 16. LIN ダイアログ、設定 サービス タブの構成



設定 サービス自動処理

このコンポーネントは設定 サービス要求を自動的に処理するように設計されています。言い換えるなら、マスターからのこれらの要求を実行するために、何らかの API やアプリケーション コードを使用する必要は無いということです。ただし、この自動処理をディスエーブルとして、これらの要求を自身のカスタム アプリケーション コードによって処理することができます。

この選択が簡単にできるように、このタブに **Automatic Configuration Request Handling** チェックボックスがあります。このボックスにチェックが付けられた場合は、このタブのすべてのオプションが使用可能になります。このボックスにチェックを付けなければ、タブの他のすべてのオプションはディスエーブルになります。

このタブの中でイネーブルとしたサービスはいずれも、このコンポーネントにより自動的に処理されます。自動的に処理されるこれらの要求のいずれかが LIN バス オペレーションの間に発生した場合は常に、それに対応する MRF および SRF フレームはトランスポート層 API を通したアプリケーションでは使用できなくなります。サービス要求が自動的に処理されない（すなわち、タブ上でイネーブルとされていない）場合は、設定 サービス要求の、対応する MRF および SRF フレームはトランスポート層 API を使用するアプリケーションにより受信または送信される必要があります。

Configuration Service Selection（コンフィグレーションサービス選択）

サポートされる各々の設定 サービス要求は、チェックボックス付きのタブにリストされます。自動的に処理するサービスを個別に選択することができます。

- サービス 0xB0 –「Assign NAD」

  これは LIN 2.1 仕様書中のオプションサービスです。

  これは、スレーブ ノードに NAD 値が新たに割り当てられる場合のサービス要求です。

  このサービス要求は、PSoC デバイスの高度なプログラマブル特性により、このコンポーネントで必要がないと考えられます。PSoC は起動の後にその NAD を望ましい値に簡単に設定することができ、おそらく、LIN マスターが NAD 変更を要求する必要はありません。

- サービス 0xB1 –「Assign Frame Identifier」

  これは LIN 2.1 仕様書では廃止となったサービスです。**LIN 2.0 Compatibility** チェックボックスがカストマイザの **General** タブ中でチェックを付けられた場合のみに使用可能となります。

  この設定 サービス要求は、スレーブ ノードが応答するフレームのフレーム ID 値を変更するために使用します。

  このサービスは LIN 2.1 仕様書に記述されていません。LIN 2.0 仕様書のセクション 2.5.1. に記述されているだけです。このサービスは、このコンポーネントで下位互換性の目的で使用可能です。

- サービス 0xB2 –「Read by identifier」

この設定 サービス 要求は LIN 2.1 仕様書に従えば必須です。この要求は LIN マスターがスレーブの識別情報（サプライヤ ID、ファンクション ID、バリエーション）を読み取ることができるように使用されます。このコンポーネントはこの要求の LIN 製品識別バージョンのみをサポートします。

- サービス 0xB3 –「Conditional Change NAD」

  これは LIN 2.1 仕様書中のオプションサービスです。

  Assign NAD 設定 サービスによく似ています。大きな違いのひとつは、このサービスがスレーブの初期の NAD（不揮発性）ではなく現行の NAD（揮発性）を使用することです。この要求が発生すると、スレーブはマスターから受け取ったデータ バイトにいくつかの論理演算処理を行い、処理結果がゼロである場合にその現行の NAD（揮発性）のみを更新します。

- サービス 0xB4 –「Data Dump」

  このサービス要求は LIN 2.1 仕様書ではオプションであり、このコンポーネントではサポートしておりません。

- サービス 0xB5 –「Assign NAD via SNPD」

  このサービスは LIN 2.1 仕様書ではサポートされていませんが、J2602-1 仕様書ではサポートされています。このサービスはこのコンポーネントではサポートされていません。

  Targeted Reset 設定 （0xB5）サービス 要求は J2602-1 仕様書に定義されています。したがって、0xB5 サービスはカストマイザの General タブにある Enable J2602-1 Compliance チェックボックスにチェックが付けられた場合のみに（ターゲットを定めたリセットとして）使用可能です。Targeted Reset がこのスレーブにより処理される場合、l_ifc_ioctl() 関数の L_IOCTL_READ_STATUS オペレーションにフラグがセットされ、アプリケーションに Targeted Reset が発生することを知らせます。詳細は機能説明をご覧ください。

- サービス 0xB6 –「Save Configuration」

  このサービス要求は LIN 2.1 仕様書ではオプションサービスです。

  スレーブ デバイスはその設定 データ（NAD 値と PID 値）を不揮発性メモリ（フラッシュ）に保存することができます。ただし、アプリケーション コードには実際のフラッシュ書き込み操作が実装される必要があります。

  この設定 サービス 要求が発生すると、l_ifc_read_status() API 関数により返される状態のコンフィギュレーションの保存フラグがセットされます。このことにより、アプリケーションに現行の LIN スレーブ ノード 設定情報を、不揮発性メモリ（フラッシュ）に保存しなければならないことを知らせます。

- サービス 0xB7 –「Assign frame identifier range」

  この設定 サービス要求は LIN 2.1 仕様書では必須です。

  このサービスにより、LIN マスターはスレーブ フレームの揮発性（volatile）フレーム PID 値を変更することができるようになります。

スレーブ情報

Automatic Configuration Request Handling チェックボックスにチェックを付けた場合は、3 つのフィールドが使用可能になります。

使用可能なフィールドは **Supplier ID**、**Function ID**、そして **Variant** です。Supplier ID は 16 ビット値ですが、有効な範囲は 0x0000 ～ 0x7FFE です。Function ID も 16 ビット値で、有効な範囲は 0x0000 ～ 0xFFFE です。Variant は 8 ビットで、有効な範囲は 0x00 ～ 0xFF です。

これらの値は、LIN クラスタ中の異なるスレーブノードを区別するために、,コンフィグレーションサービス要求で使用されます。このため、スレーブのアドレスの一種のように作用するときがあります。

クロック選択

PSoC Creator は要求される周波数とクロックのソースを計算して、実装に必要なリソースを生成します。

クロックの公差は **Automatic Baud Rate Synchronization** オプションがディスエーブルになっている場合は ±1.5 パーセント、なっていなければ ±14 パーセントです。クロックがこの制限以内で生成できない場合には警告が表示されます。その場合には、DWR 内のマスター クロック ソースを変更する必要があります。

配置

LIN コンポーネントは、UDB アレイ全体に配置され、すべての配置情報は、*cyfitter.h* ファイルを通して API に提供されます。各デザイン中にコンポーネント インスタンスを一つだけ配置することができます。

リソース

モード	デジタルブロック				API メモリ（バイト）		ピン
	データパス セル	PLD	ステータス セル	Control/Count7 セル	フラッシュ	RAM	
LIN_Slave_Example プロジェクト	4	12	3	3	4321	171	2

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、規定された設計のコンポーネントの最初のインスタンスに「LIN_1」のインスタンス名を割り当てます。インスタンス名は、識別子の構文ルールに従った独自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「LIN」というインスタンス名を使用しています。

コア API 関数

初期化サブグループ

関数	説明
l_sys_init()	LIN コアを初期化します。

信号インタラクション 関数 サブグループ

関数	説明
l_bool_rd()	1 ビットシグナルの現在のシグナル値を読み取り、返します。
l_u8_rd()	2~8 ビットシグナルの現在のシグナル値を読み取り、返します。
l_u16_rd()	9~16 ビットシグナルの現在のシグナル値を読み取り、返します。
l_bytes_rd()	シグナルの中から選択されたバイトの現在の値を読み取り、返します。
l_bool_w()	1 ビットシグナルの現在のシグナル値を v に設定します。
l_u8_wr()	2~8 ビットシグナルの現在のシグナル値を設定します。
l_u16_wr()	9~16 ビットシグナルの現在のシグナル値を設定します。
l_bytes_wr()	信号の中から選択されたバイトの現在のシグナル値を設定します。

通知関数 サブグループ

関数	説明
l_flg_tst()	現在のフラグの状態を示すブール値を返します。
l_flg_clr()	フラグの現在の値をゼロに設定します。

インターフェース管理関数*サブグループ

関数	説明
l_ifc_init()	LIN スレーブ コンポーネントを初期化します。
l_ifc_wake_up()	ウェークアップシグナルを送信します。
l_ifc_ioctl()	仕様書外の制御関数。
l_ifc_rx()	LIN スレーブはこの API ルーチンを自動的に呼び出します。
l_ifc_tx()	LIN スレーブはこの API ルーチンを自動的に呼び出します。
l_ifc_aux()	LIN スレーブはこの API ルーチンを自動的に呼び出します。
l_ifc_read_status()	指定された LIN インターフェースの状態を返します。

ユーザー規定コールアウト サブグループ

関数	説明
l_sys_irq_disable()	コンポーネントのすべての割り込みをディスエーブルにします。
l_sys_irq_restore()	コンポーネントのすべての割り込みを復元します。

ノード 設定関数

関数	説明
ld_read_configuration()	現行の設定情報をシリアライズし、アプリケーションから提供された領域（データ ポインタ）にコピーします。
ld_set_configuration()	入力パラメータにより指定された設定に基づいて NAD と PID をコンフィグレーションします。
ld_read_by_id_callout()	マスターノードが、ユーザーが定義エリア中のIDによるread by identifier requestを送信する際に使用します。

トランスポート層の関数

初期化サブグループ

関数	説明
ld_init()	RAW、または、COOKEDレイヤーを初期化もしくは再初期化します。トランスポート層のすべてのバッファは初期化されます。バス上でCOOKEDまたはRAWメッセージを送信している処理作業中の診断フレームがある場合には、中断はされません。

トランスポート層の RAW API の関数のサブグループ

関数	説明
ld_put_raw()	呼び出しは、データの 8 バイトをひとつのフレームで送信することを要求します。
ld_get_raw()	受け取った最も古い診断フレーム データを入力パラメータにより指定されたメモリにコピーします。
ld_raw_tx_status()	RAWフレーム送信関数の状態を返します。
ld_raw_rx_status()	RAWフレーム受信関数の状態を返します。

COOKED トランスポート層 API 関数サブグループ

関数	説明
ld_send_message()	データと長さにより指定された情報を、ひとつまたは複数の診断フレームにパックします。フレームはアドレス NAD と合わせてマスター ノードに送信されます。
ld_receive_message()	LIN 診断モジュールを用意し、ひとつのメッセージを受信しデータによって指定されたバッファに保存できるようにします。呼び出し時に、lengthは許容される最大長さを指定します。受信が完了すると、lengthは実際のデータ長に変更され、NAD はメッセージ中の NAD に変更されます。
ld_tx_status()	ld_send_message() に対して最後に行われた呼び出しのステータスを返します。
ld_rx_status()	ld_receive_message() に対して最後に行われた呼び出しのステータスを返します。

Non-LIN-Specified API

関数	説明
LIN_Start()	コンポーネント動作の開始。
LIN_Stop()	コンポーネント動作の停止。

コア API 関数 初期化

l_bool l_sys_init()

説明: LIN コアを初期化します。この関数は何も行わず、常にゼロを返します。

スタティック プロトタイプ: l_bool l_sys_init(void)

ダイナミック プロトタイプ なし

パラメータ: なし

返り値: 常にゼロを返します。

副作用: なし

コア API 関数 シグナルの相互作用

すべてのスタティック シグナル API コールでは、「sss」に続くもの、たとえば l_u8_rd_EngineSpeed()、がシグナルの名前となります。ダイナミック シグナル API コールでは、アプリケーションプログラミングインタフェースに定義された通り、「sss」に続くものがシグナル処理になります。

l_bool_rd()

説明: 1 ビットシグナルの現在のシグナル値を読み取り、返します。無効なシグナル処理がこの関数に渡された場合には何もアクションは取られません。

スタティック プロトタイプ: l_bool l_bool_rd_sss(void)

ダイナミック プロトタイプ l_bool l_bool_rd(l_signal_handle sss)

パラメータ: sss: 読込みシグナルのシグナル処理。

返り値: シグナルの現在の値を返します。

副作用: なし

l_u8_rd()

説明: シグナルの現在の値を読み込んで返します。無効なシグナル処理がこの関数に渡された場合には何もアクションは取られません。

スタティック プロトタイプ: l_u8 l_u8_rd_sss(void)

ダイナミック プロトタイプ l_u8 l_u8_rd(l_signal_handle sss)

パラメータ: sss: 読込みシグナルのシグナル処理

返り値: シグナルの現在の値を返します。

副作用: なし

l_u16_rd()

説明: シグナルの現在の値を読み込んで返します。無効なシグナル処理がこの関数に渡された場合には何もアクションは取られません。

スタティック プロトタイプ: l_u16 l_u16_rd_sss(void)

ダイナミック プロトタイプ l_u16 l_u16_rd(l_signal_handle sss)

パラメータ: Sss: 読込みシグナルのシグナル処理

返り値: シグナルの現在の値を返します。

副作用: この関数は、読み込まれたデータ バイトが不可分であることを保証しません。データバイトが不可分であることが必要な場合は、アプリケーションによりそのようであることを保証する必要があります。

l_bytes_rd()

説明: 信号の中から選択されたバイトの現在のシグナル値を読み取り、返します。**start** と **count** パラメータの合計は、決してバイト アレイの長さより大きくてはなりません。**start** と **count** の合計が信号バイト アレイの長さより大きい場合は、誤った**データ**が読み込まれることに注意してください。

無効なシグナル処理がこの関数に渡された場合には何もアクションは取られません。

バイト アレイが 8 バイト長であり、0~7 の番号が取られていることを確認してください。ユーザが選択したアレイから読み込まれる 2 から 6 までのバイトは、**start** が 2（0 と 1 のバイトはスキップ）で、**count** が 5 でなければなりません。この場合は、バイト 2 は user_selected_array[0] に書き込まれ、それに続くすべてのバイトは user_selected_array に昇順に書き込まれます。

スタティック プロトタイプ: void l_bytes_rd_sss(l_u8 start, l_u8 count, l_u8* const data)

ダイナミック プロトタイプ void l_bytes_rd(l_signal_handle sss, l_u8 start, l_u8 count, l_u8* const data)

パラメータ: sss: 読込みシグナルのシグナル処理

start: 読込みを始める最初のバイト

count: 読み込むバイト数

data: シグナルから読み込まれるデータが保存されるアレイへのポインタ

返り値: なし

副作用: この関数は、読み込まれたデータ バイトが不可分であることを保証しません。データバイトが不可分であることが必要な場合は、アプリケーションによりそのようであることを保証する必要があります。

l_bool_wr()

説明: シグナルに値 v を書き込みます。無効なシグナル処理がこの関数に渡された場合には何もアクションは取られません。

スタティック プロトタイプ: void l_bool_wr_sss(l_bool v)

ダイナミック プロトタイプ void l_bool_wr(l_signal_handle sss, l_bool v)

パラメータ: sss: 書き込みシグナルのシグナル処理

v: 設定されるシグナル値

返り値: なし

副作用: なし

l_u8_wr()

説明: シグナルに値 v を書き込みます。無効なシグナル処理がこの関数に渡された場合には何もアクションは取られません。

スタティック プロトタイプ: void l_u8_wr_sss(l_u8 v)

ダイナミック プロトタイプ void l_u8_wr(l_signal_handle sss, l_u8 v)

パラメータ: sss: 書き込みシグナルのシグナル処理
v: 設定されるシグナル値

返り値: なし

副作用: なし

l_u16_wr()

説明: シグナルに値 v を書き込みます。無効なシグナル処理がこの関数に渡された場合には何もアクションは取られません。

スタティック プロトタイプ: void l_u16_wr_sss(l_u16 v)

ダイナミック プロトタイプ void l_u16_wr(l_signal_handle sss, l_u16 v)

パラメータ: sss: 書き込みシグナルのシグナル処理
v: 設定されるシグナル値

返り値: なし

副作用: この関数は、書き込まれるデータ バイトが LIN マスターによって不可分に読み取られることを保証しません。データバイトが不可分であることが必要な場合は、アプリケーションによりそのようであることを保証する必要があります。

l_bytes_wr()

説明: 選択されたバイトの現在の値を sss の名前で指定されたシグナルに書き込みます。スタートとカウントの合計は、バイト アレイの長さより決して長くてはなりません。ただし、ランタイムにデバイス ドライバがそれを強制しないようにすることも選択できます。スタートとカウントの合計が信号バイト アレイの長さより大きい場合は、誤ったデータが読み込まれることに注意してください。

無効なシグナル処理がこの関数に渡された場合には何もアクションは取られません。

バイト アレイ信号が 8 バイト長であり、0~7 の番号が取られていることを確認してください。このアレイのバイト 3 と 4 の書き込みには、開始が 3（バイト 0、1、2 はスキップ）、カウントが 2 であることが必要です。この場合、バイト アレイ 信号のバイト 3 は user_selected_array[0] から、バイト 4 は user_selected_array[1] から書き込まれます。

スタティック プロトタイプ: void l_bytes_wr_sss(l_u8 start, l_u8 count, const l_u8* const data)

ダイナミック プロトタイプ void l_bytes_wr(l_signal_handle sss, l_u8 start, l_u8 count, const l_u8* const data)

パラメータ: sss: 書き込みシグナルのシグナル処理

start: 書き込まれる最初のバイト

count: 書き込むバイト数

data: LIN マスタに送信されるデータが配置されるアレイへのポインタ

返り値: なし

副作用: この関数は、書き込まれるデータ バイトが LIN マスターによって不可分に読み取られることを保証しません。データバイトが不可分であることが必要な場合は、アプリケーションによりそのようであることを保証する必要があります。

コア API 関数: 通知

アプリケーション プログラムを LIN コアに同期させるために通知フラグが使用されます。フラグは LIN コアによって自動的にセットされ、アプリケーション プログラムからはテストまたはクリアしかできません。通知フラグは、信号、特定のフレームの信号（同じ信号が複数のフレームにパックされる場合）、または、フレームに対応することができます。フラグは、対応する信号またはフレームが適正に送信または受信された際に、コンポーネントによって設定されます。

以下のフラグ API ルーチンでは、「fff」がフラグの名前になります。たとえば、l_flg_tst_RxEngineSpeed()。ダイナミック フラグ API ルーチンでは「fff」は、前に アプリケーションプログラミングインタフェース に定義された通り、シグナル処理です。

l_flg_tst()

説明: この関数は「fff」の名前により指定されたフラグの現在の状態を返します。フラグがクリアされているかtrueでない限り、falseを返します。このルーチンが「true」の値を返す場合は、対応する信号またはフレームが適正に送信または受信されたことを示します。

スタティック プロトタイプ: l_bool l_flg_tst_fff(void)

ダイナミック プロトタイプ l_bool l_flg_tst(l_flag_handle fff)

パラメータ: fff: フラグ処理の名前

返り値: 「fff」の名前により指定されたフラグの現在の状態を示す C言語のブール値を返します。

false: フラグはクリアされます

true: フラグはクリアされません

副作用: なし

l_flg_clr()

説明: 「fff」の名前により指定されたフラグをクリアします。このルーチンはフラグをテストした後で（l_flg_tst() API の後）クリアするために使用してください。コンポーネントは通知フラグを自動的にクリアしません。このルーチンは通知フラグをクリアする唯一の方法です。

スタティック プロトタイプ: void l_flg_clr_fff(void)

ダイナミック プロトタイプ void l_flg_clr(l_flag_handle fff)

パラメータ: fff: フラグ処理の名前

返り値: なし

副作用: なし

インターフェース管理関数

これらのコールは特定のインターフェース（バスへの論理チャンネル）を管理します。各々のインターフェースはそのインターフェース名で識別され、l_ifc_init_MyLinIfc() のような各 API コールでは、「iii」拡張子で示されます。このコンポーネントでは、インターフェース名はコンポーネントのインスタンス名と同じです。このコンポーネントは最大でひとつのインターフェースをサポートします。したがって、「iii」の有効な識別子は一つを超えることはありません。

l_ifc_init()

説明: l_ifc_init() は「iii」の名前により指定された、LIN スレーブコンポーネント インスタンスを初期化します。これはボーレートのなどの内部関数を設定し、LIN スレーブコンポーネントにより使用されるディジタル ブロックを起動します。これは、インターフェースに関連する他の LIN スレーブ API 関数を使用する前に実行が必要な最初のコールです。

スタティック プロトタイプ: l_bool l_ifc_init_iii(void)

ダイナミック プロトタイプ l_bool l_ifc_init(l_ifc_handle iii)

パラメータ: iii: インターフェース ハンドルの名前

返り値: この関数は、初期化が成功した場合にはゼロを返し、失敗した場合にはゼロ以外の値を返します。

副作用: なし

l_ifc_wake_up()

説明: この関数はひとつのウェークアップシグナルを送信します。ウェークアップシグナルはこの関数が呼び出されると直接送信されます。この API 関数を呼び出すと、アプリケーションはウェークアップシグナルが LIN バス上で送信されるまでブロックされます。CyDelayUs() 関数はタイミング ソースとして使用されます。遅延は、PSoC Creatorに入力されたクロック設定により計算されます。

スタティック プロトタイプ: void l_ifc_wake_up_iii(void)

ダイナミック プロトタイプ void l_ifc_wake_up(l_ifc_handle iii)

パラメータ: iii: インターフェース ハンドルの名前

返り値: なし

副作用: なし

l_ifc_ioctl()

説明: この API は他の API コールにより処理されない機能を制御します。この関数は、このコンポーネントをデバイスに特化した方法で制御します。

この関数によりサポートされる動作に関しては コンポーネントパラメータ のセクションをご覧ください。

スタティック プロトタイプ: l_u16 l_ifc_ioctl_iii(l_ioctl_op op, void* pv)

ダイナミック プロトタイプ l_u16 l_ifc_ioctl(l_ifc_handle iii, l_ioctl_op op, void* pv)

パラメータ: iii: op で定義された動作が適用されるインターフェース ハンドルの名前

op: 動作を指定するために使用されるパラメータ

pv: 関数に対して提供する必要がある、指定された動作のための一連のオプションパラメータへのポインタ

次の表は可能性がある動作と l_ifc_ioctl API ファンクションによりサポートされるコード値を記述します。表のパラメータ リストは、パラメータがいくつあり、どのデータ タイプを持っているかを示します。

“op”動作 (記号名)	値	「pv」パラメータ リスト	説明
L_IOCTL_READ_STATUS	0x00u	なし	オプションのステータスインジケータ
L_IOCTL_SET_BAUD_RATE	0x01u	l_u16 l_u16	ボー レートを変更します
L_IOCTL_SLEEP	0x02u	なし	デバイスのローパワーモードに遷移する準備をします
L_IOCTL_WAKEUP	0x03u	なし	ウェークアップ後のコンポーネントの状態を復元します。
L_IOCTL_SYNC_COUNTS	0x04u	なし	同期フィールド タイマー カウントの現在の数を返します
L_IOCTL_SET_SERIAL_NUMBER	0x05u	l_u8*	シリアル番号のポインターを更新します

返り値: 選択された動作に返されるエラー コード値はありません。これは、関数に渡される値が適正であることを確証しなければならないことを意味します。

L_IOCTL_READ_STATUS 動作

このバイトの最初のビットは一定の経過時間内にバス上に信号がなかったことを示します（**Bus Inactivity Timeout Detection** オプションを使用可能にした場合に使用できます）。経過時間が一定の閾値を超えると、このフラグが設定されます。この API を呼び出すとすべての状態ビットが返された後クリアされます。2番目のビットはTargeted Reset service request (0xB5) が受信されたことを示します（J2602-1 準拠がイネーブルの場合）。

記号名	値	説明
LIN_IOCTL_STS_BUS_INACTIVITY	0x0001u	一定の経過時間内にバス上に信号が検知されませんでした。
LIN_IOCTL_STS_TARGET_RESET	0x0002u	Targeted Reset service request(0xB5) が受信されました

L_IOCTL_SET_BAUD_RATE 動作

操作が成功すると0 を返し、不正な動作パラメータが関数に渡されると 1 を返します。

L_IOCTL_SLEEP 動作

操作が成功すると 0 を返し、不正な動作パラメータが関数に渡されると 1 を返します。

L_IOCTL_WAKEUP 動作

操作が成功すると0を返し、不正な動作パラメータが関数に渡されると 1 を返します。

L_IOCTL_SYNC_COUNTS 動作

同期フィールド バイト8のための同期フィールド タイマーカウントの現在の数を返します。

L_IOCTL_SET_SERIAL_NUMBER 動作

操作が成功すると0を返し、不正な動作パラメータが関数に渡されると 1 を返します。

副作用: なし

l_ifc_rx()

説明: LIN スレーブコンポーネントはこの API ルーチンを自動的に呼び出します。したがって、この API ルーチンはアプリケーション コードによって呼び出されてはなりません。LIN 仕様書に準拠していることを示すためだけにここにリストされています。

スタティック プロトタイプ: void l_ifc_rx_iii(void)

ダイナミック プロトタイプ void l_ifc_rx(l_ifc_handle iii)

パラメータ: iii: インターフェース ハンドルの名前

返り値: なし

副作用: なし

l_ifc_tx()

説明: LIN スレーブコンポーネントはこの APIルーチンを自動的に呼び出します。したがって、この APIルーチンはアプリケーション コードによって呼び出されてはなりません。LIN 仕様書に準拠していることを示すためだけにここにリストされています。

スタティック プロトタイプ: void l_ifc_tx_iii(void)

ダイナミック プロトタイプ void l_ifc_tx(l_ifc_handle iii)

パラメータ: iii: インターフェース ハンドルの名前

返り値: なし

副作用: なし

l_ifc_aux()

説明: LIN スレーブコンポーネントはこの API ルーチンを自動的に呼び出します。したがって、この API ルーチンはアプリケーション コードによって呼び出されてはなりません。LIN 仕様書に準拠していることを示すためだけにここにリストされています。

スタティック プロトタイプ: void l_ifc_aux_iii(void)

ダイナミック プロトタイプ void l_ifc_aux(l_ifc_handle iii)

パラメータ: iii: インターフェース ハンドルの名前

返り値: なし

副作用: なし

l_ifc_read_status()

説明: この関数は以前の通信の状態を返します。LIN スレーブ ステータス ワードの各々の状態情報フィールドに関する詳細な情報は LIN 2.1 仕様書を参照してください。

スタティック プロトタイプ: l_u16 l_ifc_read_status_iii(void)

ダイナミック プロトタイプ l_u16 l_ifc_read_status(l_ifc_handle iii)

パラメータ: iii: インターフェース ハンドルの名前

返り値: コールは次の表に示されたステータス ワード(16 ビット値)を返します:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
最後のフレームの PID								0	設定の保存	イベントによりトリガされたフレームの競合	バス動作	スリープに入ります	オーバラン	適正な送信	応答のエラー

ステータス ワードはノードにより送信または受信されたフレームに基づいてのみセットされます(bus activityを除く)。ステータス ワードは API が呼び出されるとクリアされます。

副作用: なし

ユーザが規定したコールアウト

l_sys_irq_disable()

説明: この関数はコンポーネントのすべての割り込みをディスエーブルします。割り込みマスクビットがある状態のマスクを返します。この関数はほとんどのコンポーネントの DisableInt API と本質的に同等です。

これらの関数とは異なり、返された値を保存し、後で割り込み状態を適正に復元するために l_sys_irq_restore() 関数と合わせて使用する必要があります。この API ルーチンを使用する際には十分に注意を払うことを強くお勧めします。このコンポーネントの割り込みをあまり長い時間ディスエーブルにすると、LIN 通信のエラーが起こりやすくなります。

このルーチンはアプリケーションにより提供されることが想定されています。ただし、LIN スレーブ コンポーネントはこのルーチンを自動的に実装します。必要に応じてルーチン中のコードを変更することができます。

スタティック プロトタイプ: なし

ダイナミック プロトタイプ l_irqmask l_sys_irq_disable(void)

パラメータ:

返り値: 割り込みがディスエーブルされるディジタル ブロックを定義する割り込みレジスタ マスクを返します。

副作用: なし

l_sys_irq_restore()

説明: この関数はコンポーネントの割り込みを復元します。l_sys_irq_disable() と合わせて使用してください。この関数は殆どのコンポーネントの EnableInt API と本質的に同等ですが、コンポーネントを起動するときに呼び出さなければなりません。

このルーチンはアプリケーションにより提供されることが想定されています。ただし、LIN スレーブ コンポーネントはこのルーチンを自動的に実装します。必要に応じてルーチン中のコードを変更することができます。

スタティック プロトタイプ: なし

ダイナミック プロトタイプ void l_sys_irq_restore(l_irqmask previous)

パラメータ: previous: 割り込みが使用可能になるディジタル ブロックを定義する割り込みマスク。

返り値: なし

副作用: なし

ノード 設定関数

ld_read_configuration()

説明: この関数は揮発性メモリから NAD と PID 値を読み取るために使用します。この関数は現在の設定データを読み出し、このデータを不揮発性（フラッシュ）メモリに保存するために使用することができます。アプリケーションは、LIN 状態レジスタ（l_ifc_read_status() により返される）に「Save Configuration（設定の保存）」ビットがセットされている場合には設定 データをフラッシュに保存しなければなりません。

読み取られるコンフィギュレーション データはバイト列です。1 番目のバイトはスレーブの現在の NAD です。

次のバイトはスレーブが応答する先のフレームの現在の PID 値です。PID 値は LDF または NCF ファイルに現れるフレームの順番になっています。

スタティック プロトタイプ: なし

ダイナミック プロトタイプ l_u8 ld_read_configuration(l_ifc_handle iii, l_u8* const data, l_u8* length)

パラメータ: iii: インターフェース ハンドルの名前

data: コンフィギュレーション データが読み込まれるアレイ

length: コンフィギュレーション データのバイト サイズ。長さポインタ パラメータにポイントされる値は、コンフィギュレーション データの実際の長さに設定されます。

返り値: この関数は次の表にリストされた値を返します。

記号名	説明
LD_READ_OK	コンフィギュレーション データの読み込みが成功した場合に返されます
LD_LENGTH_TOO_SHORT	長さポインタ パラメータによりポイントされる値が、コンフィギュレーション データの実際の長さより小さい場合に返されます

副作用: なし

ld_set_configuration()

説明: この関数はスレーブ ノードの揮発性 NAD および PID 値を設定するために使用します。NAD および PID 値をラン タイムで変更するために使用できます。通常は、起動直後またはマスターがそれを要求した後のみに実行しなければなりません。さもなければ、スレーブが NAD または PID あるいは両方の値を変更すると、マスターはスレーブとの通信ができなくなる場合があります。

コンフィギュレーション データに何が含まれ、どのように保管されるかに関する情報は、ld_read_configuration function() をご覧ください。

スタティック プロトタイプ: なし

ダイナミック プロトタイプ l_u8 ld_set_configuration(l_ifc_handle iii, const l_u8* const data, l_u16 length)

パラメータ: iii: インターフェース ハンドルの名前

data: スレーブ ノードに適用されるコンフィギュレーション データの配列

length: コンフィギュレーション データのバイト サイズ

返り値: 関数の戻り値は次の表にリストされています。

記号名	説明
LD_SET_OK	コンフィギュレーション データが適正に設定された場合に返されます
LD_LENGTH_NOT_CORRECT	長さパラメータの値がスレーブ ノードのコンフィギュレーション データの値と等しくない場合に返されます
LD_DATA_ERROR	コンフィギュレーション データが正しく設定されなかった場合に返されます

副作用: なし

ld_read_by_id_callout()

説明: このコールアウトはマスターノードが、ユーザーが定義した領域中の識別子の識別子要求による読み取りを送信する際に使用されます。このような要求が受信されると、スレーブ ノード アプリケーションがドライバから呼び出されます。

スタティック プロトタイプ: なし

ダイナミック プロトタイプ l_u8 ld_read_by_id_callout (l_ifc_handle iii, l_u8 id, l_u8* data)

パラメータ: iii: インターフェースハンドルの名前

id: read by identifierコンフィグレーション要求からの、ユーザー定義領域（32~63）の識別子

frameData: 5 バイトのデータ領域をポイントします。この領域はアプリケーションが正の応答を設定するために使用します。

返り値: 関数の戻り値は次の表にリストされています。

記号名	説明
LD_NEGATIVE_RESPONSE	デフォルトでは API の状態を返します。API を変更せず、これを他の状態に再割り当てすると、必ず返されます。
LD_NO_RESPONSE	この状態は手動で設定することができます。設定すると、サービスに応答がされないように指定します。
LD_POSITIVE_RESPONSE	この状態は手動で設定することができます。設定すると、サービスに応答がされるように指定します。応答は frameData パラメータによりポイントされます。

副作用: なし

トランスポート層の関数

トランスポート層は LIN ネットワーク スタックの上位層です。この層はアプリケーションがデータを「frame」フォーマットではなく「message」フォーマットで送受信できるようにします。メッセージは多バイトが可能で、複数のフレームで送受信されます。トランスポート層は設定 サービス、診断サービス、または、カスタムのユーザ定義実装に使用されます。

トランスポート層のメッセージを送受信する API 関数にはふたつの異なるフォーマットがあります。COOKED フォーマットと RAW フォーマットです。このコンポーネントは、トランスポート層 API 関数のひとつのフォーマットの使用をサポートします。API フォーマットはコンポーネント カスタマイザの **Transport Layer** タブで選択します。

注 LIN のトランスポート層 API 関数を使用するには、トランスポート層を LIN スレーブ コンポーネント カストマイザの **Transport Layer** タブで使用可能にする必要があります。

ld_init()

説明: このルーチンはスレーブ ノードのトランスポート層を初期化または再初期化します。この API はいずれのトランスポート層 API 関数も使用される前に呼び出されなければなりません。スレーブ ノードがトランスポート層通信が行える前に呼び出されなければなりません。API が実行中の診断フレームがCOOKEDまたはRAWメッセージをバス上で送信中に呼び出されると、メッセージは中断され、代わりに、API はメッセージの完了を待ちます。

スタティック プロトタイプ: なし

ダイナミック プロトタイプ void ld_init(l_ifc_handle iii)

パラメータ: iii: インターフェース ハンドルの名前

返り値: なし

副作用: なし

トランスポート層の RAW API 関数

ld_put_raw()

説明: この関数はアプリケーション コードがトランスポート層を使用してデータを送信できるようにするために使用します。本質的には、ユーザ アプリケーション アレイからフレーム バッファ アレイにいくつかのデータをコピーします。この関数は 1 回にひとつの完全なトランスポート層メッセージのフレームを送信するために使用されます。したがって、複数フレームのトランスポート層メッセージにはこの API 関数への複数のコールが必要です。この API を呼び出す前にバッファにフレームの位置があることを必ず確認してください。

スタティック プロトタイプ: なし

ダイナミック プロトタイプ void ld_put_raw(l_ifc_handle iii, const l_u8* const data)

パラメータ: iii: インターフェース ハンドルの名前

data: 送信するデータのアレイ

返り値: なし

副作用: なし

ld_get_raw()

説明: この関数はアプリケーション コードがトランスポート層を使用してデータを受信できるようにするために使用します。本質的には、フレーム バッファ アレイからユーザ アプリケーション アレイにいくつかのデータをコピーします。この関数は 1 回にひとつの完全なトランスポート層メッセージのフレームを受信するために使用されます。したがって、複数フレームのトランスポート層メッセージにはこの API 関数への複数のコールが必要です。受信キューが空の場合には、データはコピーされません。この API を呼び出す前にバッファにフレームの位置があることを必ず確認してください。

スタティック プロトタイプ: なし

ダイナミック プロトタイプ void ld_get_raw(l_ifc_handle iii, l_u8* const data)

パラメータ: iii: インターフェース ハンドルの名前

data: 受信した最も古い診断フレームがコピーされるアレイ

返り値: なし

副作用: なし

ld_raw_tx_status()

説明: このコールは、RAW APIが使用される場合に、最後に実行されたバス上のフレーム送信の状態を返します。

スタティック プロトタイプ: なし

ダイナミック プロトタイプ l_u8 ld_raw_tx_status(l_ifc_handle iii)

パラメータ: iii: インターフェース ハンドルの名前

返り値:

記号名	説明
LD_QUEUE_EMPTY	送信キューは空です。ld_put_raw() に対するコールが前に行われていれば、キューのすべてのフレームは送信されています。
LD_QUEUE_AVAILABLE	この送信キューにはエントリが含まれますが、一杯ではありません。
LD_QUEUE_FULL	送信キューは一杯でこれ以上フレームを受け入れられません。
LD_TRANSMIT_ERROR	送信中に LIN プロトコル エラーが発生しました。初期化して再度送信してください。

副作用: なし

ld_raw_rx_status()

説明: このコールは、RAW APIが使用される場合に、最後に実行されたバス上のフレーム受信の状態を返します。

スタティック プロトタイプ: なし

ダイナミック プロトタイプ l_u8 ld_raw_rx_status(l_ifc_handle iii)

パラメータ: iii: インターフェース ハンドルの名前。

返り値:

記号名	説明
LD_NO_DATA	受信キューは空です。
LD_DATA_AVAILABLE	受信キューには読み取り可能なデータが含まれています。
LD_RECEIVE_ERROR	送信中に LIN プロトコル エラーが発生しました。初期化して再度送信してください。

副作用: なし

COOKED トランスポート層 API 関数

ld_send_message()

説明: この関数はアプリケーション コードがトランスポート層を使用してデータを送信できるようにします。複数の SRF フレームを経由して自動的に送信されるデータのキューを作成する責任を負います。この関数は 完全なトランスポート層メッセージを送信するために使用されます。したがって、複数フレームのトランスポート層メッセージでもこの API 関数への 1 回のコールのみで済みます。長さ値は、6 から 4095 バイトの間で設定する必要があります。

進行中のメッセージがある場合には、動作なしでコールが返ります。

スタティック プロトタイプ: なし

ダイナミック プロトタイプ void ld_send_message(l_ifc_handle iii, l_u16 length, l_u8 nad, const l_u8* const data)

パラメータ: iii: インターフェース ハンドルの名前

length: 送信するデータのバイト サイズ

nad: データ送信先のスレーブ ノードのアドレス

data : 送信するデータのアレイ RSID の値はデータ領域の 1 番目のバイトです。

返り値: なし

副作用: 呼び出しは非同期です。すなわち、メッセージが送信されるまでサスペンドしません。また、バッファは、ld_tx_status() へのコールが LD_IN_PROGRESS を返す限り、アプリケーションにより変更されません。

ld_receive_message()

説明: この関数はアプリケーション コードがトランスポート層を使用してデータを受信できるようにします。複数の MRF フレームを受信し、メッセージのすべてのデータをユーザ アプリケーション バッファ アレイにコピーする責任を負います。この関数は 完全なトランスポート層メッセージを受信するために使用されます。したがって、複数フレームのトランスポート層メッセージでもこの API 関数への 1 回のコールのみで済みます。長さ値は、6 から 4095 バイトの間で設定する必要があります。

スタティック プロトタイプ: なし

ダイナミック プロトタイプ void ld_receive_message(l_ifc_handle iii, l_u16* const length, l_u8* const nad, l_u8* const data)

パラメータ: iii: インターフェース ハンドルの名前

length: 受信するデータのバイト サイズ

nad: 受信データ元のスレーブ ノードのアドレス

data: 受信するデータのアレイ SID の値はデータ領域の 1 番目のバイトです。

返り値: なし

副作用: コールは非同期です。すなわち、メッセージが受信されるまでサスペンドしません。また、バッファは ld_tx_status() へのコールが LD_IN_PROGRESS を返す限り、アプリケーションにより変更されません。

ld_tx_status()

説明: この関数は、ld_send_message() に対して行われた最後のコールと、最後のバス上のトランスポート層データ送信の状態を返します。

スタティック プロトタイプ: なし

ダイナミック プロトタイプ l_u8 ld_tx_status(l_ifc_handle iii)

パラメータ: iii: インターフェース ハンドルの名前。

返り値: 次の値が戻されます。

記号名	説明
LD_IN_PROGRESS	送信はまだ完了していません。
LD_COMPLETED	送信は適正に完了しました（新しい ld_send_message call() を発行することができます）。この値はトランスポート層が初期化された後にも返されます。
LD_FAILED	送信はエラーを起こして終了しました。データは部分的にしか送られていません。さらにメッセージを処理する前にトランスポート層を再初期化する必要があります。送信がエラーとなった理由を知るためには、状態管理関数 l_read_status() をチェックしてください。
LD_N_AS_TIMEOUT	送信は N_As のタイムアウトのためにエラーになりました。また、現在のメッセージ送信は中断されました。LIN 2.1 仕様書のセクション 3.2.5 をご覧ください。

副作用: なし

ld_rx_status()

説明: この関数は、ld_receive_message() に対して行われた最後のコールと、最後のバス上のトランスポート層データ受信の状態を返します。

スタティック プロトタイプ: なし

ダイナミック プロトタイプ l_u8 ld_rx_status(l_ifc_handle iii)

パラメータ: lii: インターフェース ハンドルの名前

返り値: 次の値が戻されます。

記号名	説明
LD_IN_PROGRESS	受信はまだ完了していません。
LD_COMPLETED	受信は適正に完了し、すべての情報(長さ、NAD、データ)が使用可能です。また、新しい ld_receive_message() コールを発行できます。この値はトランスポート層が初期化された後にも返されます。
LD_FAILED	受信はエラーにより終了しました。データは部分的にしか受信されていないので、信頼できません。さらにトランスポート層メッセージを処理する前に初期化してください。受信がエラーとなった理由を知るためには、状態管理関数 l_read_status() をチェックしてください。
LD_N_CR_TIMEOUT	受信は N_Cr タイムアウトによりエラーになりました。また、現在のメッセージ受信は中断されます。LIN 2.1 仕様書のセクション 3.2.5 をご覧ください。
LD_WRONG_SN	予期しないシーケンス番号のため受信はエラーになりました。

副作用: なし

Non-LIN-Specified API

LIN_Start()

説明: コンポーネント動作の開始 この関数は必要ありません。

スタティック プロトタイプ: なし

ダイナミック プロトタイプ l_bool LIN_Start()

パラメータ: なし

返り値: Zero: 正常に初期化されました。
Nonzero: 初期化はエラーとなりました。

副作用: なし

LIN_Stop()

説明:	コンポーネント動作の停止 この関数は必要ありません。
スタティック プロトタイプ:	なし
ダイナミック プロトタイプ	l_bool LIN_Stop()
パラメータ:	なし
返り値:	なし
副作用:	なし

ファームウェアソースコードのサンプル

PSoC Creator は、[Find Example Project (サンプルプロジェクトを検索)] 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、コンポーネントカタログ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 (スタートページ)」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)」を参照してください。

PSoC 3 Reentrancy サポート

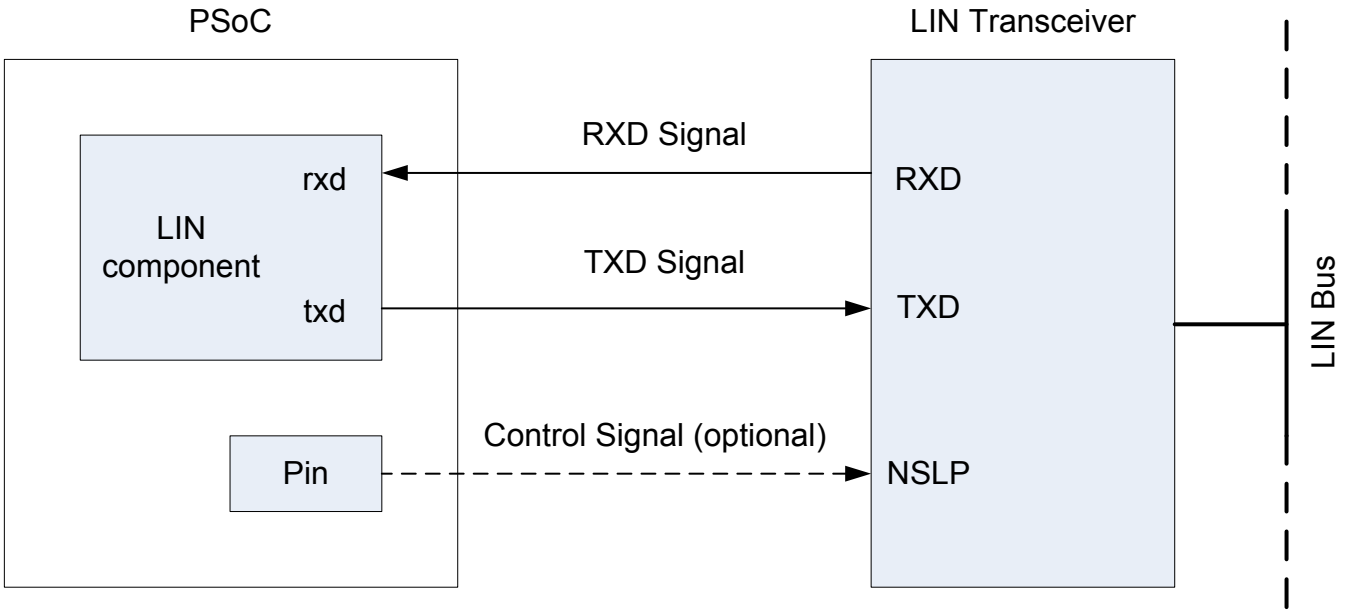
CyIntClearPending() 関数は、LIN コンポーネント内部のふたつの別々の割り込みから呼び出しができるため、同時呼び出しが可能です。デフォルトではリエントラントはできませんが、コンパイル時に「関数への複数コール」警告を排除するために、リエントラント性をサポートできるようにすることができます。詳細は、PSoC Creator ヘルプの「PSoC 3 のリエントラント コード」のトピックをご覧ください。また、コンポーネント プロジェクト例 (example project) にも、リエントラント性サポートが追加されました。

PSoC および LIN バス ハードウェア インターフェース

PSoC LIN スレーブ ノードが LIN バスに直接接続される場合には LIN 物理層トランシーバデバイスが必要です。この場合には、LIN コンポーネントの TxD ピンがトランシーバの TXD ピンに接続され、RxD ピンがトランシーバの RXD ピンに接続されます。LIN トランシーバ デバイスは、PSoC の電気信号レベルが LIN バスの電気信号と一致しないために必要となります。

一部の LIN トランシーバ デバイスにはデバイスの動作状況を制御するために使用する「イネーブル」または「スリープ」の入力信号もあります。LIN コンポーネントはこの制御信号を提供しません。その代わりに、信号が必要な場合は、LIN トランシーバ デバイスに必要な信号を出力するピンを使用します。

図 17. PSoC と LIN Bus の間のハードウェア インターフェース



DC 電気的特性と AC 電気的特性

DC および AC の電気特性の情報は、*LIN 2.1 仕様書* の「LIN 物理層仕様」のチャプタをご覧ください。

コンポーネントの変更

バージョン 1.0 は LIN スレーブ コンポーネントの最初のリリースです。

バージョン	変更の説明	変更の理由 / 影響
1.0.a	データシートのマイナーな編集と更新	

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使用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提供することは、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提供することは、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。