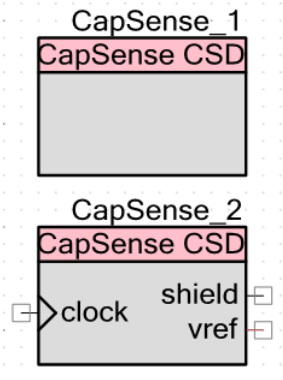
Capacitive Sensing (CapSense® CSD)
2.10

特長



- ユーザー定義のボタン、スライダ、タッチパッドおよび近接容量センサの組み合わせをサポート
- 自動 SmartSense チューニングまたは統合 PC GUI を使ったマニュアル チューニングを提供
- AC 電源供給ラインノイズ、EMC ノイズ、電源電圧変化に対する高いイミュニティ
- センサのスキャンレートを向上するオプションの 2 つのスキャンチャネル（並列同期）
- 水膜や水滴がある場合でも信頼できるオペレーションが可能なシールド電極サポート
- CapSense カスタマイザを使用したセンサと端子の割り当て

概要説明

デルタシグマモジュレータ (CapSense CSD) を用いた静電容量式検知は、タッチセンスボタン、スライダ、タッチパッド、近接検知などのアプリケーションにおいて、静電容量を計測するための柔軟で効率よい方法です。

このデータシートを読んだあとに、次のアプリケーションノートを読まれることを推奨します。アプリケーション ノートは、サイプレス セミコンダクタのウェブサイト www.cypress.com からご覧いただけます：

- CapSense のベスト プラクティス – AN2394
- CapSense アプリケーションでの信号対雑音比要件 – AN2403
- PSoC CapSense アプリケーションの EMC デザインにおける考慮点 – AN2318
- PSoC CapSense のレイアウト ガイドライン – AN2292
- 防水加工の宣伝容量センシング – AN2398

CapSense コンポーネントを使用する場合

静電容量センシングシステムは、従来のボタン、スイッチおよび他のコントロールに代えて、雨や水に曝されるアプリケーションでも使用することができます。このようなアプリケーションには、自動車、アウトドア用具、ATM、公共交通システム、携帯電話や PDA のようなポータブルデバイス、そして台所や浴室のアプリケーションがあります。

入出力接続

このセクションでは、CapSense CSD コンポーネントの様々な入力および出力接続について説明します。I/O リストのアスタリスク (*) は、I/O が、その I/O の説明でリストされている条件において、シンボルに隠れている可能性があることを示します。

clock – 入力 *

CapSense CSD コンポーネントにクロックを供給します。クロック入力は、**Enable clock input** がチェックされている場合にだけ表示されます。

shield – 出力 *

シールド電極信号はこの出力に接続します。シールド電極がイネーブルなときにのみ利用可能です。シールドについての詳細は「Parameters and Setup（パラメーターと設定）」セクションを参照してください。

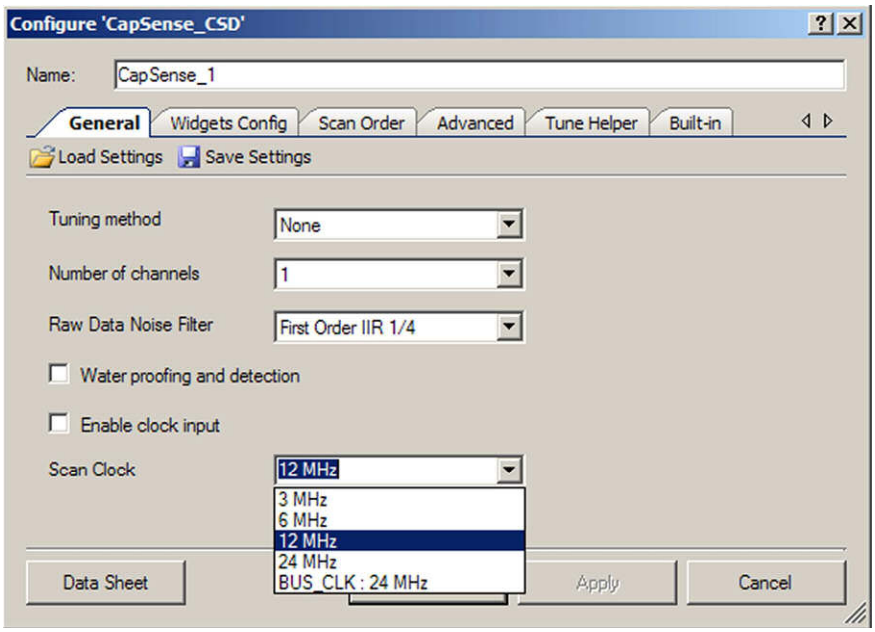
vref – 出力 *

アナログ リファレンス電圧はこの出力に接続します。シールド信号の振幅を調整するために使います。**Shield** オプションが IDAC **Sourcing mode** でイネーブルな時のみ利用可能です。SIO がシールド信号に使用されている場合は、Vref 出力は SIO リファレンスに接続する必要があります。vref 使用の詳細はこのデータシートの「Functional Description（機能詳細）」セクションを参照してください。

パラメータおよびセットアップ

CapSense CSD 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、CSD 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。

[General (全般)] タブ



Load Settings/Save Settings

Save Settings はそのコンポーネント向けに設定された全ての設定およびチューニングデータを保存するために使用します。これを使うと、新しいプロジェクトに素早く複製することができます。**Load Settings** は以前保存された設定をロードするために使用します。

保存した設定は、Tuner GUI に設定やチューニングデータをインポートするためにも使用します。

Tuning method

このパラメーターがチューニング方法を指定します。3 つのオプションがあります:

- Auto (SmartSense) – CapSense CSD コンポーネントの自動チューニングを提供し、全てのデザインに対して推奨されるチューニング方法です。ファームウェアアルゴリズムがランタイムにおける最良のチューニングパラメーターを継続的に決定します。このモードでは、追加の RAM および CPU リソースが必要です。
  Important – SmartSense モードの1つの CapSense_CSD コンポーネントだけがプロジェクト図面上に配置可能です。SmartSense のチューニングを、**Tuner Helper** タブで規定されている EZI2C 通信コンポーネントと共に使用して、ターゲットデバイスから Tuner GUI にデータを転送することができます。

- Manual - Tuner GUI を使って CapSense CSD コンポーネントをマニュアルでチューニングできます。

  GUI を起動するには、記号の上で右クリックして **Launch Tuner** を選択します。Manual チューニングの詳細については、このデータシートの「CapSense Tuner GUI User Guide (CapSense Tuner GUI ユーザーガイド)」セクションを参照してください。マニュアル チューニングでは、ターゲットデバイスから Tuner GUI にデータを転送するためには、**Tuner Helper** タブで規定されている EZI2C 通信コンポーネントが必要です。

  - なし (デフォルト) – チューニングがディスエーブル。全てのチューニング・パラメーターはフラッシュに保存されています。このオプションは、CapSense コンポーネントの全てのパラメーターをチューニングし最終化した後でのみ使用されます。このオプションを使用すると、通信機能は提供されますが、何もしないので、チューナーはこのモードでは動作しません。

Number of channels

このパラメーターは実装されているハードウェア・スキャニング・チャネルの数を規定します。

デフォルト。1 ～ 20 のセンサの使用が最適です。コンポーネントは一度に1つの静電容量スキャンを実行できます。連続して 1 つのセンサがスキャンされます。ハードウェアに単一のチャネルしか実装されていないため、このオプションは結果として最低限のハードウェアリソースしか使用されません。

- AMUX バスが共に接続されます。

  注 全ての静電容量センサがチップの左の片側だけに配置されている場合 (#例えば偶数ポート GPIO :P0[X]、P2[X]、P4[X]) または右 (#例えば奇数ポート GPIO :P1[X]、P3[X]、P5[X]) AMUX バスは共に接続されず、AMUX バスの片側だけが使用されます。

  注 ポートピン P15[0-5] は異なる AMUX バス左および右への接続があります。P12[X] および P15[6-7] は AMUX バスへの接続がありません。選択された部品についての詳細は TRM を参照してください。

- コンポーネントは 1 から (#GPIO – 1) の静電容量センサのスキャンが可能です。
- Cmod の外部コンデンサが 1 つ必要です。

20 以上のセンサの使用が最適です。コンポーネントは同時に 2 つの静電容量スキャンを実行できます。左および右の AMUX バスが両方、各チャネルに対して使用されます。右および左センサは同時に 2 つ(右センサ1つ、左センサ1つ)連続してスキャンされます。一つのチャネルに他のチャネルよりも多くのセンサがある場合、センサ数が最も多いチャネルが、配列中のセンサに対して一度に一つずつスキャンを終了しますが、その間他のチャネルはスキャンを実行しません。2 つのチャネルを使うと、1 つのチャネルよりも使用するリソースが倍になりますが、その結果センサのスキャンレートは倍になります。

- 左の AMUX バスは 1 ～ (#even ports GPIO – 1) の静電容量センサをスキャンすることができます。
- 右の AMUX バスは 1 ～ (#odd ports GPIO – 1) の静電容量センサをスキャンすることができます。
- Cmod の外部コンデンサが、各チャネルに 1 つずつ、合計 2 つ必要です。
- 並列スキャンは同じスキャンレートで実行します。

「Raw Data Noise Filter（生データノイズフィルタ）」

このパラメーターは生データフィルタを選択します。1 つのフィルタを選択して、それを全てのパラメーターに適用します。センサをスキャンしている間のノイズを減らす効果があるので、フィルタの使用を推奨します。フィルタの種類についての詳細はこの文書中の「Functional Description – Filters（機能の説明–フィルタ）」セクションを参照してください。

- なし – フィルタは提供されません。ファームウェアのフィルタまたは SRAM 変数オーバーヘッドは発生しません。
- Median – 直近 3 つのセンサ値を順番にソートし、中央値を返します。
- Averaging – 直近 3 つのセンサ値の単純平均を返します。
- First Order IIR 1/2 – 以前のフィルタ値の 1/2 に最新のセンサ値の 1/2 を加えた値を返します。IIR フィルタは、最も少ないファームウェアと全てのフィルタタイプの中で最小の SRAM オーバーヘッドで動作します。
- First Order IIR 1/4 – デフォルト– 以前のフィルタ値の 3/4 に最新のセンサ値の 1/4 を加えた値を返します。
- First Order IIR 1/8 – 以前のフィルタ値の 7/8 に最新のセンサ値の 1/8 を加えた値を返します。
- First Order IIR 1/16 – 以前のフィルタ値の 15/16 に最新のセンサ値の 1/16 を加えた値を返します。
- Jitter – 最新のセンサ値が最後のセンサ値よりも大きい場合、以前のフィルタ値に 1 を加えますが、以前のフィルタ値が小さい場合は 1 を引きます。

Water proofing and detection

この機能は CapSense CSD を防水をサポートするように設定します。（デフォルトではチェックされていません）。この機能は以下のパラメーターを設定します:

- シールド出力端子をイネーブルにします
- ガードウィジェットを追加します

注 ガードウィジェットの防水機能が望ましくない場合は、**Advanced** タブで解除できます。

「Enable clock input（クロック入力をイネーブルにする）」

このパラメーターは、コンポーネントが内部クロックを使用するか、またはユーザー供給のクロック接続用の入力端子を表示するか、を選択します（デフォルトではチェックされていません）。

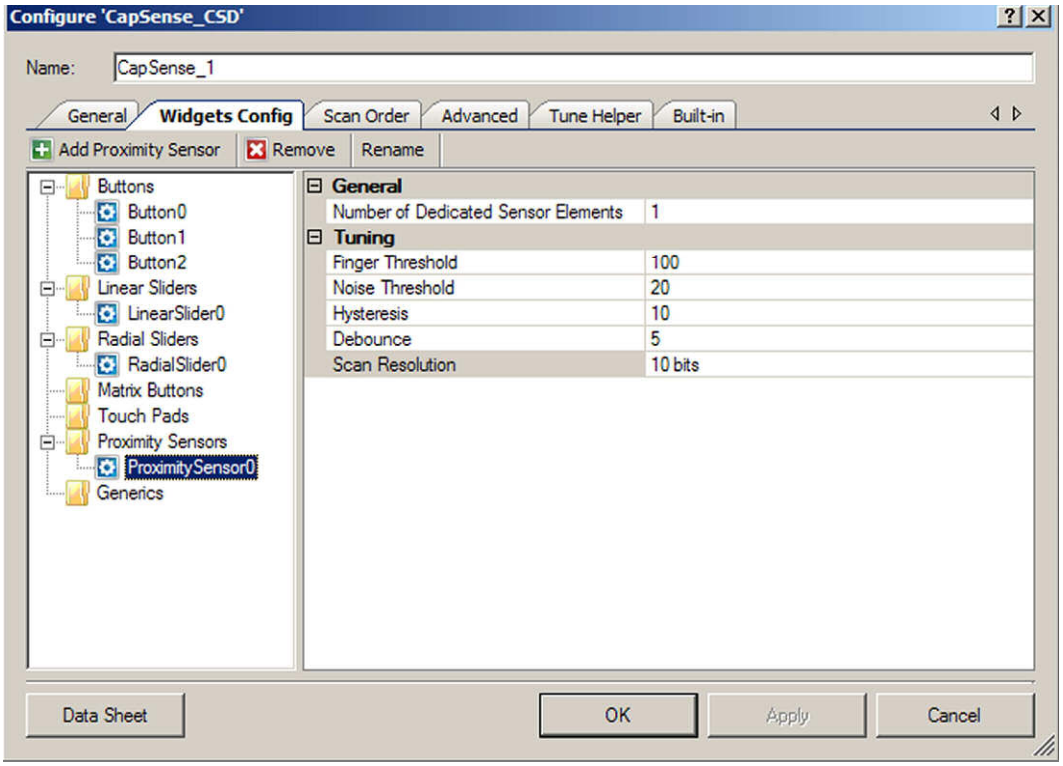
注 カスタマイザは内部データを計算するためにクロック周波数を知る必要があるため、チューニング方法が Auto (SmartSense) である場合はこのオプションはディスエーブルです。

「Scan Clock（スキャンクロック）」

このパラメーターは内部 CapSense コンポーネントクロックの周波数を規定します。値の幅は 3 MHz ～ 24 MHz (12 MHz – デフォルト)。**Enable clock input** がチェックされていると、この機能はディスエーブルです。

注 Analog Switch Drive Source を「FF Timer (Default)」および/または **Digital Implementation** を「FF Timers」に設定すると、BUS_CLK 以下の CapSense CSD クロックをサポートしないため、BUS_CLK を選択してください。

[Widgets Config (ウィジェット設定タブ)] タブ



様々なパラメーターの定義については機能説明セクションを参照してください。

ツールバー

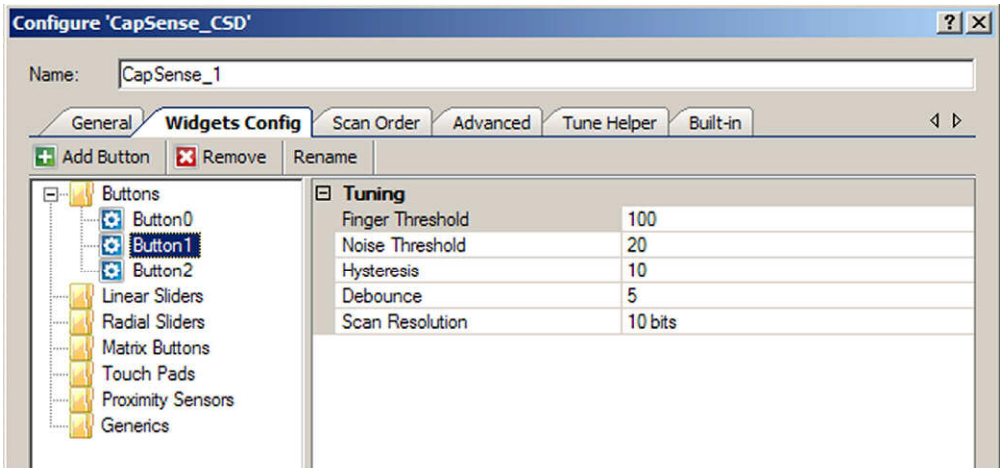
ツールバーには以下のコマンドがあります：

- **Add widget** – 選択した種類のウィジェットをツリーに追加します。ウィジェットの種類：
  - Button – 単一センサ上の指プレスを検知し、単一機械的ボタンの代替を提供します。
  - Linear Slider – リニア スライダ は形状プレスされた、少数センサをインターポレーティングして、整数値を提供します。
  - Radial Slider – ラジアル スライダ はリニア スライダ に似ていますが、センサが円形に配置されています。

- ❑ Matrix Button – マトリックスボタンは行センサと列センサで形成された交点での指プレスを検知します。マトリックスボタンは多数のボタンを効率よくスキャンする方法を提供します。
- ❑ Touch Pad – タッチパッドは、タッチパッド上の形状プレスの X および Y 座標を返します。タッチパッドは複数の行センサおよび列センサから構成されています。
- ❑ Proximity Sensor – 近接センサはセンサから遠く離れた手指またはより大きな物体の存在の検知を最適化し、実際に接触を起こす必要はなくなります。
- ❑ Generic Sensor – ジェネリックセンサは単一センサからの生データを提供し、他のセンサ種類で処理された出力では不可能な、固有または高度なセンサの生成を可能にします。

- **Remove widget** – 選択したウィジェットをツリーから削除します。
- **Rename** – ダイアログを開いて選択したウィジェットの名前を変更します。ウィジェットをダブルクリックしてダイアログを開くこともできます。

Buttons

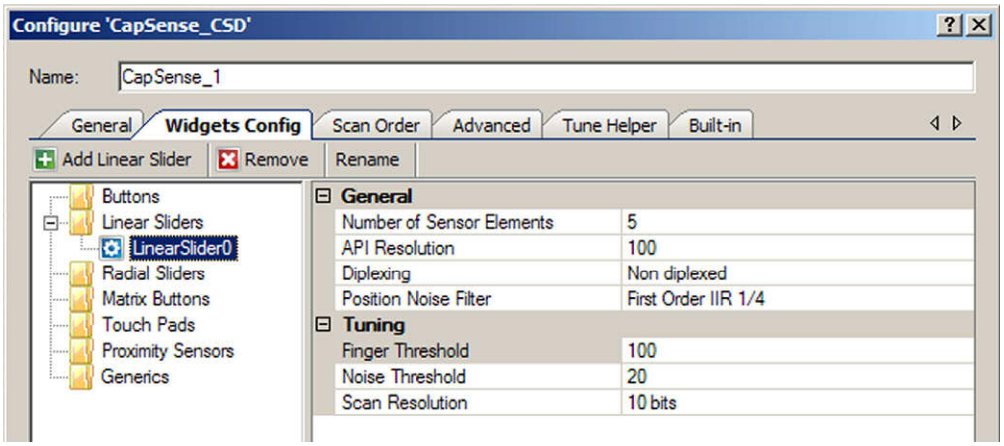


Tuning:

- **Finger Threshold** – センサにアクティブな閾値を定義して、接触に対する感度を高くしたり低くしたりできます。センサスキャン値がこの閾値よりも大きい場合は、ボタンが接触されたとレポートされます。デフォルト値は 100 です。値の有効範囲は [1…255] です。指の閾値 +ヒステリシスは 254 より大きくなることはありません。
- **Noise Threshold** – センサのノイズ閾値を定義します。この閾値を上回るカウント値が観測された場合、ベースラインは更新しません。ノイズ閾値が低すぎると、センサおよびサーマルオフセットは、偽の接触またはミスタッチを生ずる可能性があります。ノイズ閾値が高すぎると、指の接触がノイズと解釈され、人為的に増大したベースラインがミスタッチを生ずることがあります。デフォルト値は 20 です。値の有効範囲は [1…255] です。

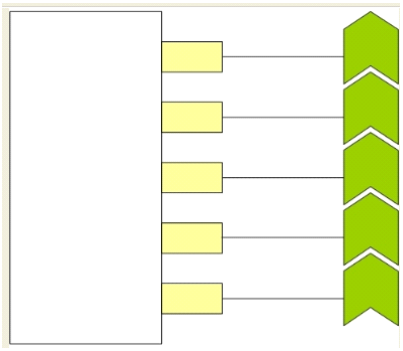
- **Hysteresis** – センサ アクティブ状態の遷移に差のヒステレシスを追加します。センサが アクティブでない場合、差の数は指の閾値＋ヒステリシスを上回る必要があります。センサが アクティブの場合、差の数は指の閾値－ヒステリシスを下回る必要があります。ヒシテリシスは、低振幅のセンサノイズおよびわずかな指の動きが、ボタン状態の循環を引き起こさないように働きます。デフォルト値は 10 です。値の有効範囲は [1…255] です。指の閾値 ＋ヒステリシスは 254 より大きくなることはありません。
- **Debounce** – デバウンスカウンタを追加して、センサ アクティブ状態の遷移を検知します。センサが非アクティブからアクティブへ遷移するためには、指定されたサンプル数に対して、"差の数値"が指閾値＋ヒステリシスを上回る状態を続けなければなりません。デフォルト値は 5 です。高周波数で高振幅のノイズが押されたボタンを誤って検知しないように、デバウンスは確認します。値の有効範囲は [1…255] です。
- **Scan Resolution** – スキャニング分解能を定義します。このパラメーターは、ボタンウィジェット内でのセンサのスキャン時間に影響があります。ビット数が N の場合、スキャニング分解能の最大生カウントは 2^N-1 です。解像度の増大は接触検知の感度と「Signal to Noise Ratio (SNR)（S/N 比-信号対ノイズ比）」を向上しますが、スキャン時間が長くなります。デフォルト値は 10 ビットです。

Linear Sliders

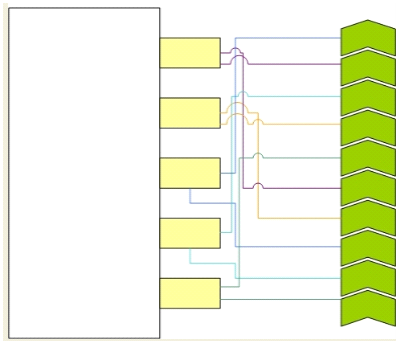


General:

- **Numbers of Sensors Elements** – スライダ内のエレメントの数を定義します。API 分解能対センサ エレメントの好適な比率は 20:1 です。API 分解能対センサエレメントの比率を高くしすぎると、計算した指位置のノイズを高める結果になります。値の有効範囲は [2…32] です。デフォルト値は 5 エレメントです。
- **API Resolution** – スライダの分解能を定義します。ポジションの値はこの範囲で変更できます。値の有効範囲は [1…255] です。
- **Diplexing –** ダイプレクスしていない（デフォルト）またはダイプレクスされている。ダイプレクスすると 2 つのスライダが単一のデバイスピンを共有できるので、所定のスライダセンサが必要とするピンの総数を減らすことができます。



Non Diplexed



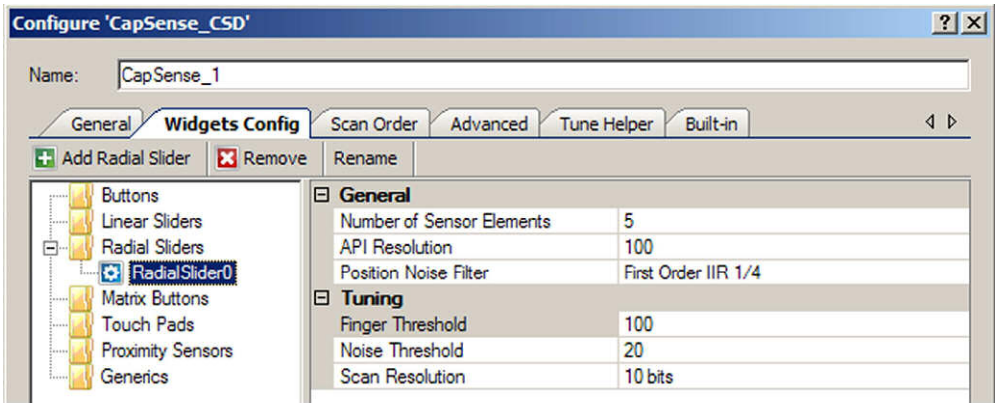
Diplexed

- **Position Noise Filter** – 位置計算で機能するノイズフィルタの種類を選択します。選択されたウィジェットには 1 つのフィルタのみが適用されます。フィルタの種類についての詳細はこの文書中の「Functional Description – Filters（機能の説明-フィルタ）」セクションを参照してください。
  - ❑ なし
  - ❑ Median Filter
  - ❑ Averaging Filter
  - ❑ First Order IIR 1/2
  - ❑ First Order IIR 1/4 – デフォルト
  - ❑ Jitter Filter

Tuning:

- **Finger Threshold** – センサにアクティブな閾値を定義して、接触に対する感度を高くしたり低くしたりできます。センサスキャン値がこの閾値よりも大きい場合は、ボタンが接触されたとレポートされます。デフォルト値は 100 です。値の有効範囲は [1…255] です。
- **Noise Threshold** – スライダ エレメントのセンサノイズ閾値を定義します。この閾値を上回るカウント値が観測された場合、ベースラインは更新しません。ノイズ閾値が低すぎると、センサおよびサーマルオフセットは、偽の接触またはミスタッチを生ずる可能性があります。ノイズ閾値が高すぎると、指の接触がノイズと解釈され、人為的に増大したベースラインがセントロイドの位置計算に誤りを生ずることがあります。この閾値を下回るカウント値が観測された場合、セントロイドの計算に加えられません。デフォルト値は 20 です。値の有効範囲は [1…255] です。
- **Scan Resolution** – スキャニング分解能を定義します。このパラメーターは、リニア スライダ ウィジェット内の全てのセンサのスキャン時間に影響を与えます。ビット数が N の場合、スキャニング分解能の最大生カウントは 2^N-1 です。分解能を高くすると、検出感度と接触検知の S/N 比が高くなりますが、スキャン時間は増えます。デフォルト値は 10 ビットです。

Radial Slider



General:

- **Numbers of Sensors Elements** – スライダ内のエレメントの数を定義します。API 分解能対センサ エレメントの好適な比率は 20:1 です。API 分解能対センサ エレメントの比率を高くしすぎると、分解能計算のノイズを高める結果になります。値の有効範囲は [2…32] です。デフォルト値は 5 エレメントです。
- **API Resolution** – スライダの分解能を定義します。ポジションの値はこの範囲で変更できます。値の有効範囲は [1…255] です。
- **Position Noise Filter** – 位置計算で機能するノイズフィルタの種類を選択します。選択されたウィジェットには 1 つのフィルタのみが適用されます。フィルタの種類についての詳細はこの文書中の「Functional Description – Filters（機能の説明-フィルタ）」セクションを参照してください。
  - なし
  - Median Filter
  - Averaging Filter
  - First Order IIR 1/2
  - First Order IIR 1/4 – デフォルト
  - Jitter Filter

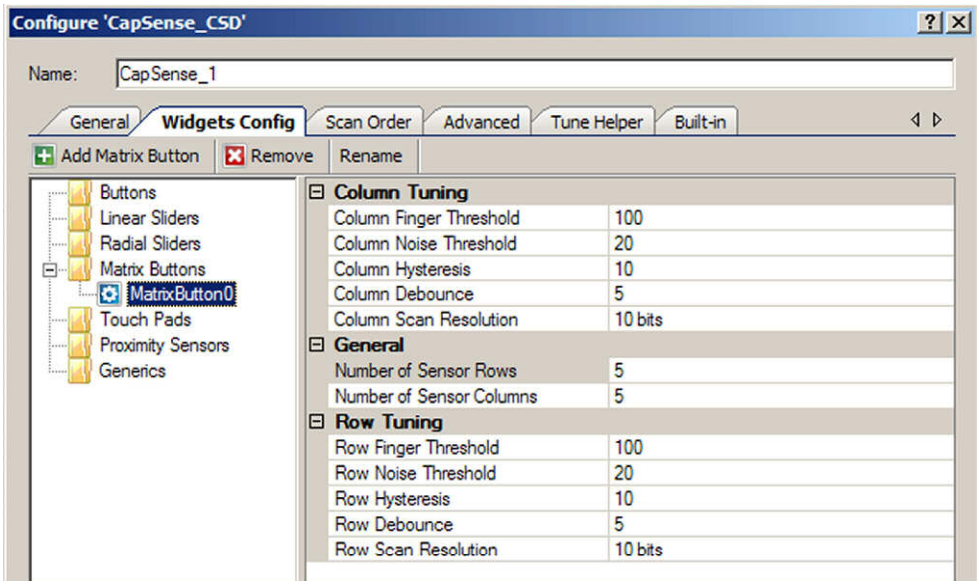
Tuning:

- **Finger Threshold** – センサにアクティブな閾値を定義して、接触に対する感度を高くしたり低くしたりできます。センサスキャン値がこの閾値よりも大きい場合は、ボタンが接触されたとレポートされます。デフォルト値は 100 です。
- **Noise Threshold** – スライダ エレメントのセンサノイズ閾値を定義します。この閾値を上回るカウント値が観測された場合、ベースラインは更新しません。ノイズ閾値が低すぎると、センサおよびサーマルオフセットは、偽の接触またはミスタッチを生ずる可能性があります。ノイズ閾値が高すぎると、指の接触がノイズと解釈され、人為的に増大したベースラインがセントロイドの位置計算に誤りを生ずることがあります。この閾値を下回るカ

ウント値が観測された場合、セントロイドの計算に加えられません。デフォルト値は 20 です。値の有効範囲は [1…255] です。

- **Scan Resolution** – スキャニング分解能を定義します。このパラメーターは、ラジアル スライダ ウィジェット内の全てのセンサのスキャン時間に影響を与えます。ビット数が N の場合、スキャニング分解能の最大生カウントは 2^N-1 です。分解能を高くすると、検出感度と接触検知の S/N 比が高くなりますが、スキャン時間は増えます。デフォルト値は 10 ビットです。

Matrix Buttons



Tuning:

- **Column/Row Finger Threshold** – マトリックスボタンの列と行のセンサにアクティブな閾値を定義して、接触に対する感度を高くしたり低くしたりできます。センサスキャン値がこの閾値よりも大きい場合は、ボタンが接触されたとレポートされます。デフォルト値は 100 です。値の有効範囲は [1…255] です。指の閾値 + ヒステリシスは 254 より大きくなることはありません。

- **Column/Row Noise Threshold** – マトリックスボタンの列と行のセンサノイズ閾値を定義します。この閾値を上回るカウント値が観測された場合、ベースラインは更新しません。ノイズ閾値が低すぎると、センサおよびサーマルオフセットは、偽の接触またはミスタッチを生ずる可能性があります。ノイズ閾値が高すぎると、指の接触がノイズと解釈され、人為的に増大したベースラインがミスタッチを生ずることがあります。デフォルト値は 20 です。値の有効範囲は [1…255] です。

- **Column/Row Hysteresis** – マトリックスボタンの列と行にセンサ アクティブ状態の遷移の差のヒステレシスを追加します。センサが アクティブでない場合、差の数は指の閾値 +ヒステリシスを上回る必要があります。センサが アクティブの場合、差の数は指の閾値 −ヒステリシスを下回る必要があります。ヒシテリシスは、低振幅のセンサノイズおよびわずかな指の動きが、ボタン状態の循環を引き起こさないように働きます。デフォルト

値は 10 です。値の有効範囲は [1…255] です。指の閾値 +ヒステリシスは 254 より大きくなることはありません。

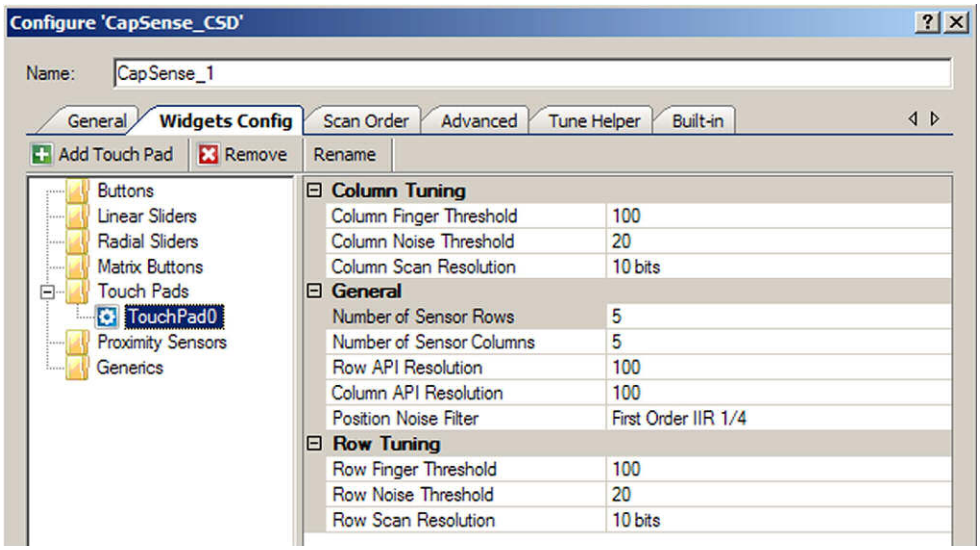
- **Column/Row Debounce** – マトリックスボタンの列と行にデバウンスカウンタを追加して、センサ アクティブ状態の遷移を検知します。センサが非アクティブからアクティブへ遷移するためには、指定されたサンプル数に対して、"差の数値"が指閾値+ヒステリシスを上回る状態を続けなければなりません。デフォルト値は 5 です。高周波数で高振幅のノイズが押されたボタンを誤って検知しないように、デバウンスは確認します。値の有効範囲は [1…255] です。

- **Column/Row Scan Resolution** – マトリックスボタンの列と行のスキャニング分解能を定義します。このパラメーターは、マトリックスボタンウィジェットの列/行内の全てのセンサのスキャン時間に影響を与えます。ビット数が N の場合、スキャニング分解能の最大生カウントは 2^N-1 です。分解能を高くすると、検出感度と接触検知の S/N 比が高くなりますが、スキャン時間は増えます。列と行のスキャニング分解能は、同じ感度レベルを得るために、同じであるべきです。デフォルト値は 10 ビットです。

全般

- **Number of Sensor Columns/Rows** – マトリックスを形成する列と行の数を定義します。値の有効範囲は [2…32] です。デフォルト値は列も行も両方ともに 5 エレメントです。

Touch Pads



Tuning:

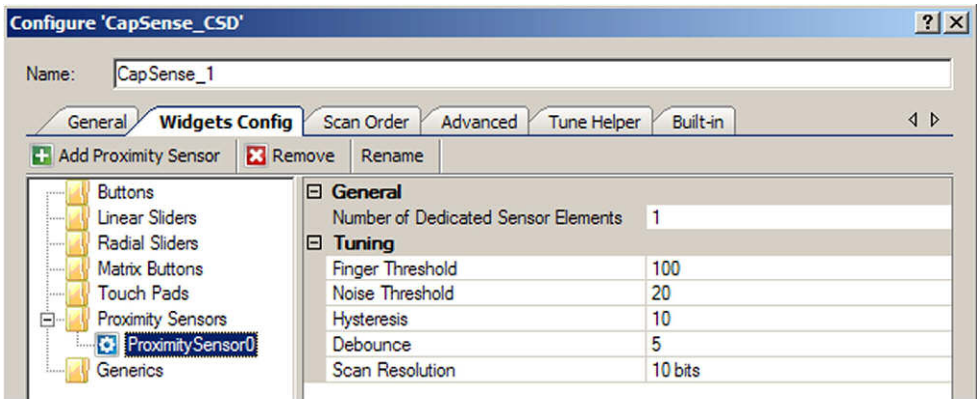
- **Column/Row Finger Threshold** – タッチパッドの列と行のセンサにアクティブな閾値を定義して、接触に対する感度を高くしたり低くしたりできます。センサスキャン値がこの閾値よりも大きい場合は、タッチパッドは接触位置をレポートします。デフォルト値は 100 です。値の有効範囲は [1…255] です。

- **Column/Row Noise Threshold** – タッチパッドの列と行のセンサノイズ閾値を定義します。この閾値を上回るカウント値が観測された場合、ベースラインは更新しません。この閾値を下回るカウント値が観測された場合、セントロイドの計算に加えられません。ノイズ閾値が低すぎると、センサおよびサーマルオフセットは、偽の接触またはミスタッチを生ずる可能性があります。ノイズ閾値が高すぎると、指の接触がノイズと解釈され、人為的に増大したベースラインがセントロイドの計算に誤りを生ずることがあります。デフォルト値は 20 です。値の有効範囲は [1…255] です。
- **Column/Row Scan Resolution** – タッチパッドの列と行のスキャニング分解能を定義します。このパラメーターは、タッチパッドウィジェットの行/列内の全てのセンサのスキャン時間に影響を与えます。ビット数が N の場合、スキャニング分解能の最大生カウントは 2^N-1 です。分解能を高くすると、検出感度と接触検知の S/N 比が高くなりますが、スキャン時間は増えます。列と行のスキャニング分解能は、同じ感度レベルを得るために、同じであるべきです。デフォルト値は 10 ビットです。

全般

- **Number of Sensor Columns/Rows** – タッチパッドを形成する列と行の数を定義します。値の有効範囲は [2…32] です。デフォルト値は列も行も両方ともに 5 エレメントです。
- **API Resolution Colum/Row** – タッチパッドの列と行の分解能を定義します。指位置の値はこの範囲内でレポートされます。値の有効範囲は [1…255] です。
- **Position Noise Filter** – 位置計算にノイズフィルタを追加します。選択されたウィジェットには 1 つのフィルタのみが適用されます。フィルタの種類についての詳細はこの文書中の「Functional Description – Filters（機能の説明-フィルタ）」セクションを参照してください。
  - ❑ なし
  - ❑ Median Filter
  - ❑ Averaging Filter
  - ❑ First Order IIR 1/2
  - ❑ First Order IIR 1/4 – デフォルト
  - ❑ Jitter Filter

Proximity Sensors



全般

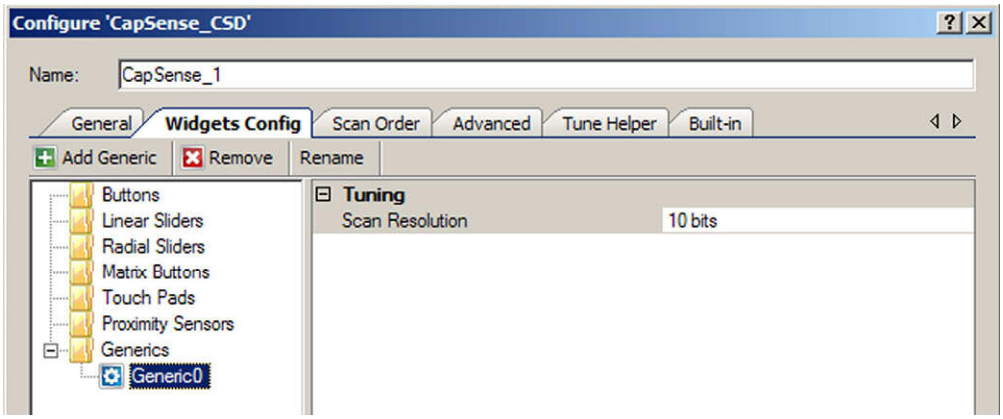
- **Numbers of Dedicated Sensors elements** – 専用の近接センサの数を選択します。これらのセンサ エレメントは、他のウィジェット用にインスタンスを作成した全ての他のセンサに追加されます。任意のウィジェットセンサが単独で、または互いに並列に接続して使用され、近接センサを作り出すことができます。
  - 0 – 近接センサは一つあるいは複数の既存センサだけをスキャンして近接性を判断します。このウィジェットには新しいセンサは割り当てられません。
  - 1 – システム内の専用近接センサの数。– デフォルト。

Tuning:

- **Finger Threshold** – センサにアクティブな閾値を定義して、接触に対する感度を高くしたり低くしたりできます。センサスキャン値がこの閾値よりも大きい場合は、近接センサが接触されたとレポートされます。デフォルト値は 100 です。値の有効範囲は [1…255] です。指の閾値 +ヒステリシスは 254 より大きくなることはありません。
- **Noise Threshold** – センサノイズ閾値を定義します。この閾値を上回るカウント値が観測された場合、ベースラインは更新しません。ノイズ閾値が低すぎると、センサおよびサーマルオフセットは、失敗または偽の接触や近接を生ずる可能性があります。ノイズ閾値が高すぎると、指の接触がノイズと解釈され、人為的に増大したベースラインがミスタッチを生ずることがあります。値の有効範囲は [1…255] です。
- **Hysteresis** – センサ アクティブ状態の遷移に差のヒステリシスを追加します。センサが アクティブでない場合、差の数は指の閾値 +ヒステリシスを上回る必要があります。センサが アクティブの場合、差の数は指の閾値 -ヒステリシスを下回る必要があります。低振幅のセンサノイズおよびわずかな指の動きが、近接センサ状態の循環を引き起こさないことを、ヒステリシスは確認します。値の有効範囲は [1…255] です。
- **Debounce** – デバウンスカウンタを追加して、センサ アクティブ状態の遷移を検知します。センサが非アクティブからアクティブへ遷移するためには、指定されたサンプル数に対して、"差の数値"が指閾値 +ヒステリシスを上回る状態を続けなければなりません。高周波数で高振幅のノイズが近接イベントを誤って検知しないように、デバウンスは確認します。値の有効範囲は [1…255] です。

- **Scan Resolution** – スキャニング分解能を定義します。このパラメーターは、近接ウィジェットのスキャン時間に影響があります。ビット数が N の場合、スキャニング分解能の最大生カウントは 2^N-1 です。分解能を高くすると、検出感度と接触検知の S/N 比が高くなりますが、スキャン時間は増えます。検知範囲を広げるためには、典型的なボタンに使用されるよりも高い分解能を近接検知に使用することを推奨します。デフォルト値は 10 ビットです。

Generics



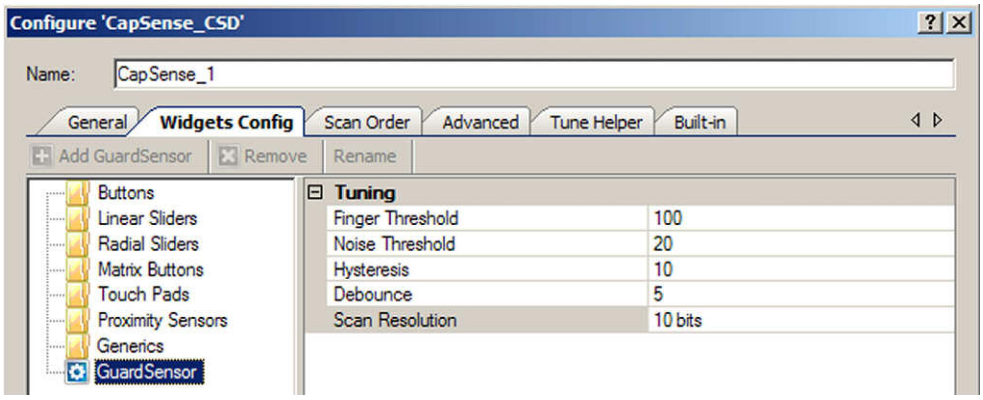
Tuning:

- **Scan Resolution** – スキャニング分解能を定義します。このパラメーターは、ジェネリックウィジェットのスキャン時間に影響があります。ビット数が N の場合、スキャニング分解能の最大生カウントは 2^N-1 です。分解能を高くすると、検出感度と接触検知の S/N 比が高くなりますが、スキャン時間は増えます。デフォルト値は 10 ビットです。

ジェネリックウィジェット利用可能なチューニングオプションは 1 つだけです。なぜなら、前述のウィジェットのどれにも当てはまらない CapSense センサやアルゴリズムをサポートするために、ほかのハイレベルなチューニングはカスタマ向けに残してあるからです。

Guard Sensor

この特別なセンサは **Advanced Tab** を使って追加または削除します。ガードセンサは他のセンサのように指のプレスをレポートしませんが、他のウィジェット付近の無効な状況をレポートして、その更新を妨げます。このセンサタイプとどのような場合に使用すべきかの詳細は、このデータシートの「Functional Description(機能詳細)」セクションにある「Guard Sensor Implementation(ガードセンサの実装)」をご参照ください。

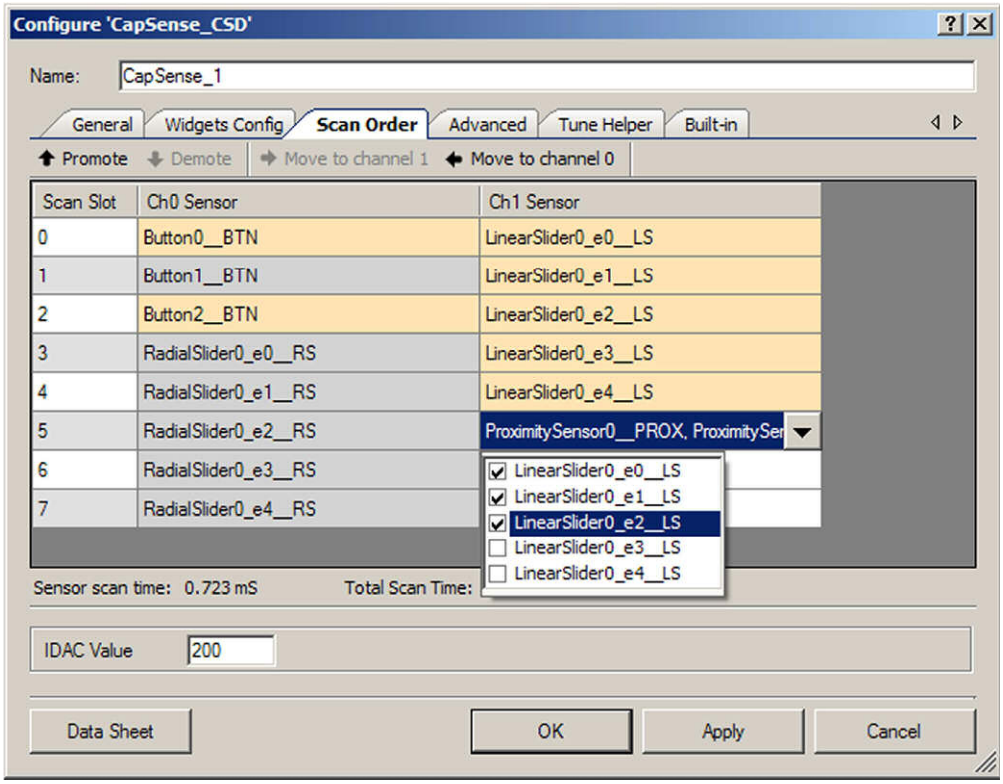


Tuning:

- **Finger Threshold** – センサにアクティブな閾値を定義して、接触に対する感度を高くしたり低くしたりできます。センサスキャン値がこの閾値よりも大きい場合は、ガードセンサが接触されたとレポートされます。デフォルト値は 100 です。値の有効範囲は [1…255] です。指の閾値 +ヒステリシスは 254 より大きくなることはありません。
- **Noise Threshold** – センサノイズ閾値を定義します。この閾値を上回るカウント値が観測された場合、ベースラインは更新しません。ノイズ閾値が低すぎると、センサおよびサーマルオフセットは、偽の接触またはミスタッチを生ずる可能性があります。ノイズ閾値が高すぎると、指の接触がノイズと解釈され、人為的に増大したベースラインがミスタッチを生ずることがあります。デフォルト値は 20 です。有効範囲は [1…255] です。
- **Hysteresis** – センサ アクティブ状態の遷移に差のヒステレシスを追加します。センサが アクティブでない場合、差の数は指の閾値 +ヒステリシスを上回る必要があります。センサが アクティブの場合、差の数は指の閾値 −ヒステリシスを下回る必要があります。ヒシテリシスは、低振幅のセンサノイズおよびわずかな指の動きが、ボタン状態の循環を引き起こさないように働きます。デフォルト値は 10 です。値の有効範囲は [1…255] です。指の閾値 +ヒステリシスは 254 より大きくなることはありません。
- **Debounce** – デバウンスカウンタを追加して、センサ アクティブ状態の遷移を検知します。センサが非アクティブからアクティブへ遷移するためには、指定されたサンプル数に対して、"差の数値"が指閾値 +ヒステリシスを上回る状態を続けなければなりません。デバウンスは、高周波数で高振幅のノイズがガードセンサを誤って検知しないように、機能します。デフォルト値は 5 です。値の有効範囲は [1…255] です。
- **Scan Resolution** – スキャニング分解能を定義します。このパラメーターは、ガードセンサのスキャン時間に影響があります。ビット数が N の場合、スキャニング分解能の最大生カウントは 2^N-1 です。分解能を高くす

ると、検出感度と接触検知の S/N 比が高くなりますが、スキャン時間は増えます。デフォルト値は 10 ビットです。

[Scan Order (スキャン順序)] タブ



ツールバー

ツールバーには以下のコマンドがあります:

- **Promote/Demote** – 選択したウィジェットをデータグリッド中で上下に動かします。一つまたは複数のウィジェット エレメントを選択すると、ウィジェット全体が選択されます。
- **Move to Channel 1/ Channel 0** – 選択したウィジェットを他のチャネルに移動します。このオプションは 2 チャンネルデザインでのみ動作します。一つまたは複数のウィジェット エレメントを選択すると、ウィジェット全体が選択されます。

注 スキャン順が変わったら、ピンを再び割り当てます。

注デフォルトで、近接センサはスキャンプロセスから除かれています。一般的には他のセンサと同時にスキャンされないので、近接スキャンのセンサはマニュアルでランタイムに開始する必要があります。

IDAC Value

選択したセンサの IDAC 値を指定します。このオプションは IDAC が **Current Source** (**[Advanced]** タブの下) として選択されている場合にのみ動作します。有効範囲は 0-255 で、デフォルト値は 200 です。

Sensitivity

Sensitivity はセンサをアクティベートするために必要な Cs (センサ容量) の微小な変化です。有効な価の範囲は [1…4] で、感度レベルに対応しています：0.1、0.2、0.3、および 0.4 pF。デフォルト値は 2 です。
Sensitivity はセンサの全体的な感度を設定し、オーバーレイ素材の様々な厚みに対応します。より厚みのある素材は、より低い感度値を使うべきです。

このオプションは **Tuning method** パラメーターが[Auto (自動)(SmartSense)] に設定されているときにのみ利用可能です。

センサスキャン時間

典型的なシステムでの選択されたセンサに必要なおおよそのスキャン時間を示します。

[Auto (自動)(SmartSense)] が調整方法として選択されている場合、調整されている間にパラメーターが変化するので、表示された値は不正確な場合があります。CapSense CSD コンポーネントの内部クロック周波数が不明な場合は、[Unknown (不明)] が表示されます。

CapSense CSD コンポーネントの以下のパラメーターはセンサのスキャン時間に影響します：

- Scan Speed
- Resolution
- CapSense CSD clock

Total Scan Time

全てのセンサをスキャンするのに必要な総スキャン時間を示します。この値はだいたいのセンサスキャン時間なので、実際の時間よりも若干異なる場合があります。

[Auto (自動)(SmartSense)] が調整方法として選択されている場合、調整されている間にパラメーターが変化するので、表示された値は不正確な場合があります。CapSense CSD コンポーネントの内部クロック周波数が不明な場合は、[Unknown (不明)] が表示されます。

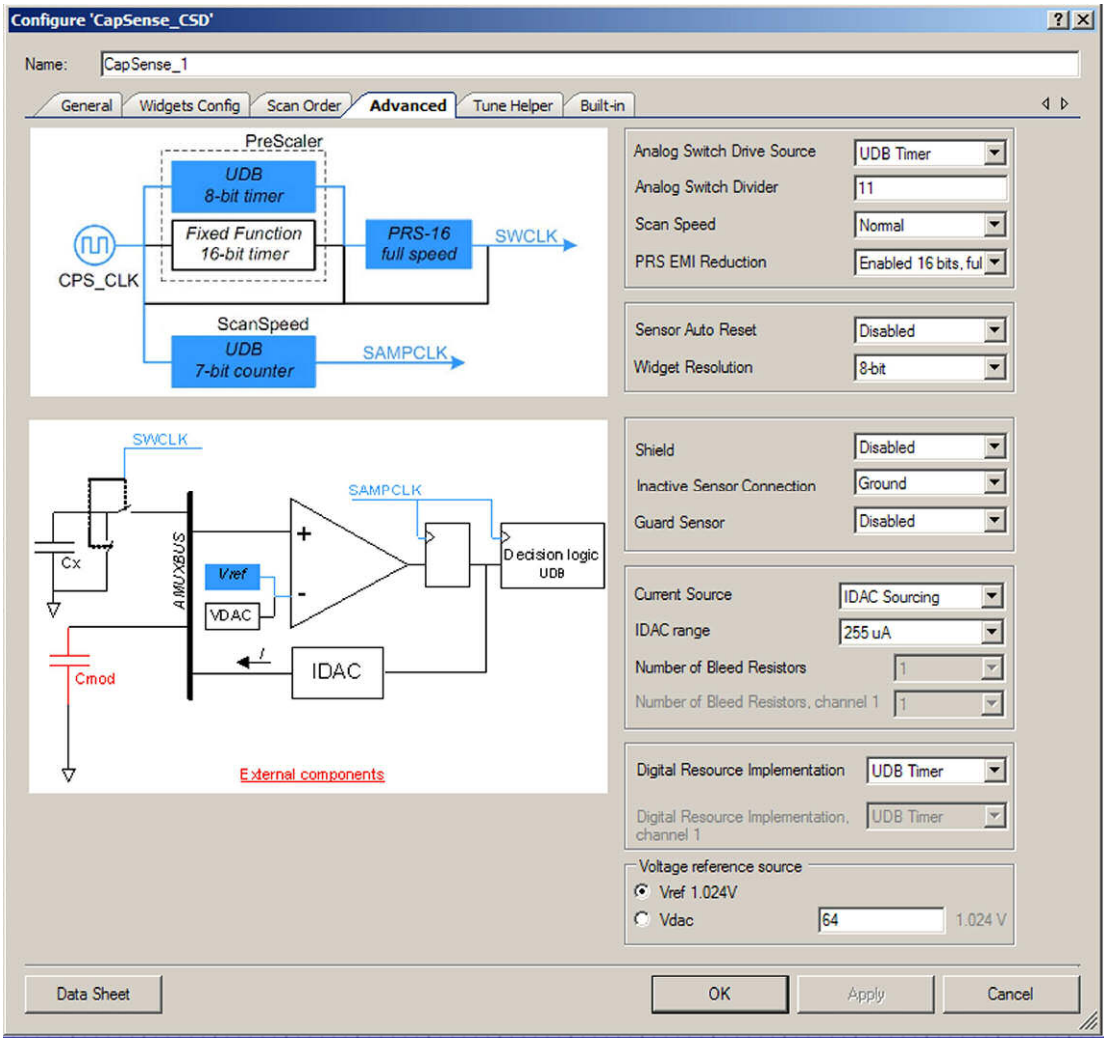
Widget List (ウィジェット一覧)

表中で交差する灰色とオレンジ色の行にウィジェットが一覧されています。一つのウィジェットに関連する全てのセンサは、同じ色を共有し、異なるウィジェットのエレメントを強調しています。

近接スキャンセンサは専用近接センサを使用したり、または専用センサおよび/または他のセンサの組み合わせから近接性を検知することができます。例えば、基板上にボタン配列の周囲に張り巡らされたトレースがあり、近接センサはそのトレースと配列の全てのボタンからなる場合があります。これら全てのセンサは近接性を検知するために同時にスキャンされます。近接スキャンセンサにはドロップダウンリストがあり、近接性を検知するためにスキャンする一つまたは複数のセンサを選択できます。

近接センサと同様に、ジェネリックセンサも複数のセンサで構成されている場合があります。ジェネリックセンサは専用センサ、それ以外の任意かつ既存のセンサ、または複数のセンサからデータを取得できます。ドロップダウンからセンサを選択します。

[Advanced (詳細設定)] タブ



Analog Switch Drive Source

このパラメーターは、センサが変調コンデンサ Cmod へ、または変調コンデンサからスイッチするレートを決定するアナログスイッチ ディバイダのソースを指定します。タイマを「Fixed Function Timer blocks (固定関数タイマブロック) (FF タイマ)」で実装すると、静止時の使用されるリソースが最小に抑えられます

- Direct – FF タイマまたは UDB リソースを使用しませんが、デバイスの最大クロックレートをアナログスイッチレートと同じに制限します。ほとんどのデザインには推奨しません。
- UDB Timer – UDB リソースを使用します
- FF Timer – デフォルト – UDB リソースを使用しません

Analog Switch Divider

このパラメーターは、アナログスイッチ ディバイダの値を特定し、プレチャージスイッチの出力周波数を決定します。値の有効範囲は [1…255] です。デフォルト値は 11 です。

Analog Switch Drive Source が直接に設定されていると、この機能はディスエーブルです。

- センサーはプレチャージクロックの速度で継続的に変調コンデンサ Cmod へスイッチ、または変調コンデンサ Cmod からスイッチします。Analog Switch Divider は CapSense CSD クロックを分割してプレチャージクロックを生成します。ディバイダ値が低下すると、センサは素早くスイッチし、生カウントが増えます。その逆も同じことがいえます。

Scan Speed

このパラメーターは、センサのスキャン時間を決定する、CapSense CSD コンポーネントのデジタル倫理クロック周波数を指定します。スキャニング速度が遅いと時間はかかりますが、S/N 比が向上したり、電源および温度変化へのより良いイミュニティが得られるという利点があります。

- Slow – コンポーネントの入力クロックを 16 で除します
- Normal – コンポーネントの入力クロックを 8 で除します – デフォルト
- Fast – コンポーネントの入力クロックを 4 で除します
- Very Fast – コンポーネントの入力クロックを 2 で除します

表 1。スキャン速度 (単位:µs) と分解能

分解能 (単位:ビット)	スキャニング速度			
	超高速	高速	通常	低速
8	58	80	122	208
9	80	122	208	377
10	122	208	377	718
11	208	377	718	1400
12	377	718	1400	2770
13	718	1400	2770	5500
14	1400	2770	5500	10950

分解能 (単位:ビット)	スキャニング速度			
	超高速	高速	通常	低速
15	2770	5500	10950	21880
16	5500	10950	21880	43720

注 表 1 のスキャン時間は以下の設定を基に予想しています。マスタクロック および CPU クロック = 48 MHz、CapSense CSD クロック = 24 MHz、チャネル数 = 1。スキャン時間は、2 つのセンサスキャンの間の間隔を測定したものです。この時間には、センサのセットアップ時間、サンプル変換間隔、データ処理時間が含まれています。値を直線的に測定して、これらの値を他のクロックレートでのスキャニング速度や追加センサを予想するために使用することができます。

PRS EMI Reduction

このパラメーターは、Psuedo Random Sequence (PRS) ジェネレーターを、アナログプレチャージクロックを生成するために使用するかどうかを指定します。CapSense アナログスイッチング周波数のスペクトルを拡散し、EMI エミッションおよび感度を低下するので、PRS の使用を推奨します。PRS クロック設定は、[Analog Switch Divider (アナログスイッチデバイダ)] 設定から提供されます。PRS EMI リダクションがイネーブルでないと、単一の周波数が使用され、結果として基本周波数および高調波のエミッションが増加します:

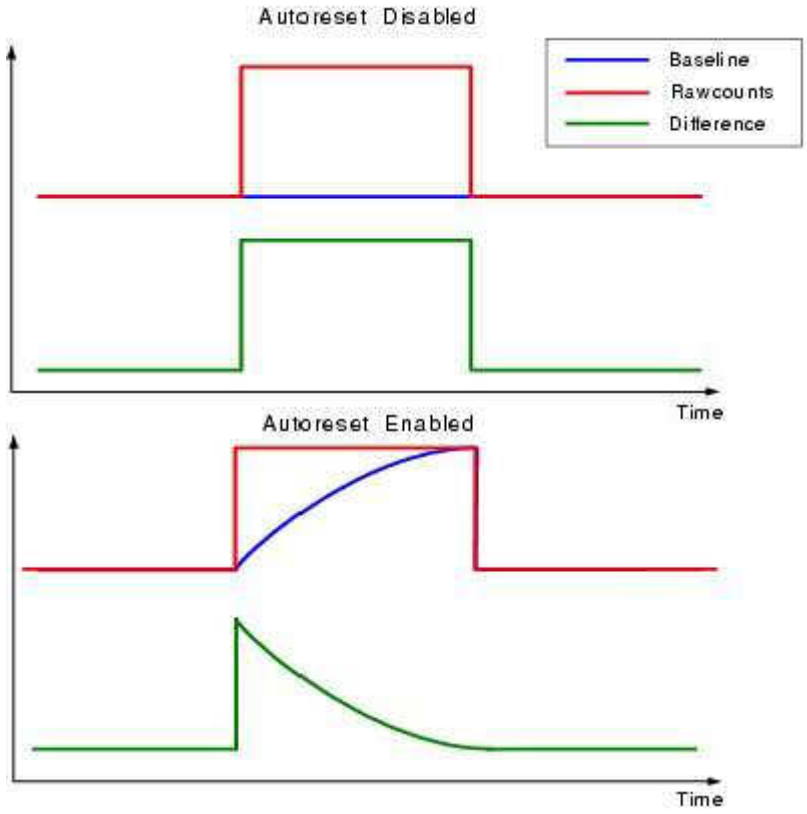
- Disabled (ディスエーブル)
- Enabled 8 bits – 8 ビットはより良い S/N 比をもたらしますが、短い反復期間が EMI を増加します。
- Enabled 16 bits, full speed– デフォルト – 16 ビットはより低い S/N 比をもたらしますが、EMI 減少は優れています。
- Enabled 16 bits, ¼ full speed – 「Enabled 16 bits, full speed」と同じ PRS クロック出力を得るには、4 倍速いクロックが必要です。

Sensor Auto Reset

このパラメーターは、差カウントがノイズ閾値の上下に関わらず、ベースラインを常に更新する自動リセットをイネーブルにします。自動リセットがディスエーブルな場合は、差カウントがノイズ閾値のプラス/マイナス(ノイズ閾値は反映)の範囲内にあるときにだけ、ベースラインを更新します。何もセンサに触れずに 生カウントが突然上がった際に、センサが恒久的にオンになるという問題がない限り、このパラメータは [Disabled] にしておきます。

- Enable – ベースラインは常に更新されていて、ボタンのプレスを見逃したり、ボタンがスタックするのを防ぎますが、ボタンがプレスされているとレポートする時間の最大長を制限するように、自動リセットは確認します。この設定は、センサの最大時間を制限します (標準的な値は 5～10 秒) が、何もセンサに触れずに生カウントが突然上がった際に、センサが恒久的にオンになるのを妨ぐことができます。この突然の上昇の原因には、大幅な電源電圧の変化、高エネルギー RF ノイズ源、非常に速い温度変化があります。

- Disable – デフォルト – 異常なシステム条件によって、ベースラインが継続的にノイズ閾値を超えるために更新を停止することがあり、結果としてボタンのプレスを見逃したり、ボタンがスタックすることがあります。利点としては、ボタンが押されている状態を無制限にレポートできることです。設計者は、スタックしたり反応のないボタンを見つけ出すアプリケーション固有の方法を提供する必要があります。



Widget Resolution

このパラメーターは、ウィジェットがレポートする信号の分解能を指定します。8 ビット (1 バイト) がデフォルトオプションで、ほとんどのアプリケーションに使用されます。ウィジェット値が 8 ビット範囲を超えると、システムの感度が上がりすぎるため、公称値を中間範囲 (～128) に調整します。高い精度が必要なスライダおよびタッチパッドウィジェットは、16 ビットの分解能から恩恵をうけます。8 ビットで起こりうる四捨五入による誤差を避けることで 16 ビットの分解能は線形性を高めますが、1 センサ毎に 2 バイトになる追加の SRAM 使用という負担があります。

- 8 bits (1 バイト) – デフォルト
- 16 bits (2 バイト)

Shield electrode

このパラメーターは、水滴および水膜による影響を取り除くために使用されるシールド電極出力が、イネーブルかディスエーブルかを指定します。シールド電極の使用に関する詳細については **Shield electrode usage and Restrictions** セクションをご参照ください。

- Disable – デフォルト
- Enable

Inactive Sensor Connection

このパラメーターは、アクティブにスキャンされてはいない全てのセンサについてデフォルトのセンサ接続を定義します。

- Ground – デフォルト – アクティブにスキャンされたセンサのノイズを低減するので、ほとんどのアプリケーションに使用できます。
- Hi-Z Analog – 非アクティブなセンサを Hi-Z のままにします。
- Shield – シールド波形をスキャンされていないセンサ全てに提供します。シールド信号の振幅は Vddio に等しくなります。シールド電極と一緒に使用される場合は、防水機能の増加とノイズの低下をもたらします。

Guard Sensor

このパラメーターは、防水が必要なアプリケーションへの水滴を検知する、ガードセンサをイネーブルにします。**Water Proofing and detection** (**[General]** タブの下) がチェックされていると、この機能は自動的にイネーブルになります。ガードセンサの詳細は、このデータシートの「Functional Description(機能詳細)」セクションにある「Guard Sensor Implementation(ガードセンサの実装)」をご参照ください。

- Disable – デフォルト
- Enable

Current Source

CapSense CSD はセンサ上の接触を検知するために正確な電流源が必要です。IDAC シンクおよび IDAC ソースは、PSoC デバイス上でのハードウェア IDAC の使用が必要です。外部レジスタは、ユーザーが供給する抵抗を IDAC ではなく PCB 上で使用し、IDAC 制約アプリケーションで有用です。

- IDAC Sourcing – デフォルト – IDAC は電流を変調コンデンサ CMOD に供給します。アナログスイッチは 変調コンデンサ CMOD と GND を交互に行き来するように設定されていて、電流向けのシンクを提供します。IDAC Sourcing は 3 つの方法の中で最大の信号対ノイズ比をもたらすため、ほとんどのデザインに推奨しますが、他のモードでは必要としない追加の VDAC リソースを Vref レベルに設定することが必要となる場合があります。
- IDAC Sinking – IDAC は変調コンデンサ CMOD からの電流を沈めます。アナログスイッチは 変調コンデンサ CMOD と VDD を交互に行き来するように設定されていて、電流向けのソースを提供します。

S/N 比は一般的に IDAC ソーシングモードほど高くはありませんが、ほとんどのデザインで問題なく動作します。

- External Resistor – IDAC が Rb へ接地するブリーダー抵抗に交換されることを除いて、IDAC シンキング設定と同じように機能します。ブリーダー抵抗は、変調コンデンサ、Cmod および GPIO の間に接続します。GPIO はオープンドレインドライバの Low ドライブモードに設定され、Cmod が Rb から放電されます。このモードは最小のアナログリソースしか必要としないので、リソース制限が必要なときにのみ使用します。このモードは IDAC または VDAC を必要としないので、電源がシステムにとって考慮すべき重要な問題であれば最も低い消費電力にコンポーネントを設定することが可能です。

IDAC Range

このパラメーターは、電流源の IDAC 範囲を指定します。**Current Source** が外部レジスタに設定されていると、このパラメーターはディスエーブルです。デフォルトがほとんど全ての CapSense デザインにとって最良の選択です。より低いまたはより高い電流範囲は、一般的には非接触静電容量センサでのみ使用されます。

- 32uA
- 255uA – デフォルト
- 2.04mA

Number of Bleed Resistors, channel 0/channel 1

このパラメーターは、ブリーダー抵抗の数を指定します。ブリーダー抵抗の最大数はチャネルごとに 3 つです。**Current Source** が IDAC ソースまたは IDAC シンクに設定されていると、この機能はディスエーブルです。複数のブリーダー抵抗は、システム調整に役立つ 3 つまでのグループのセンサに異なる電流を可能にするようにサポートされています。センササイズが類似のほとんどのデザインは、ブリーダー抵抗が 1 つだけ必要です。

Digital Resource Implementation, channel 0/channel 1

このパラメーターは、タイマやカウンタを含む CapSense のデジタル部分を実装するために使用されるリソースの種類を指定します。最大の実装柔軟性が提供されているので、このパラメーターは、ほとんどのデザインで変更するべきではありません。

- UDB Timer – デフォルト – 実装は最も柔軟ですが、高価な UDB リソースが必要です。
- FF Timer – FF Timer の実装には UDB リソースを使用しませんが、スキャンスピード = 高速をサポートしていません。

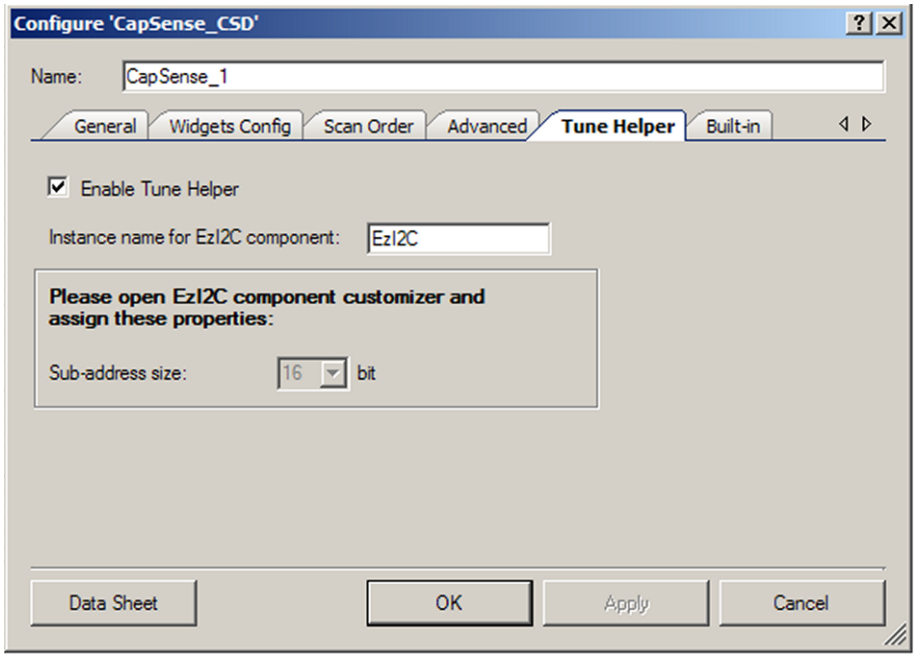
Voltage Reference Source

このパラメーターは、リファレンス ソース電圧の種類とレベルを指定します。IDAC モードでは リファレンス電圧が高いほうがよく、IDAC シンクまたは外部レジスタ電流源モードでは参照電圧が低い方が良いです。

- 1.024V – デフォルト – IDAC シンクモードに最適
- VDAC – IDAC ソースモードに最適。[Voltage DAC (電圧 DAC)] を使って利用可能な範囲を最大化して、リファレンス電圧の調整が可能です。参照ソース VDAC は、**Current Source** が IDAC ソースに設定されていて、VDAC デバイスリソースが必要な場合にのみ、利用可能です。リファレンス電圧が増大するほど、感度も高くなりますが、シールド電極への影響は減少します。

VDAC が選択されると、CapSense バッファは使用されません。これは低電圧用に設計されているからです。これにより、スタートアップ時に Cmod は VDAC から Vref へ充電します。Cmod から Vref へ充電するのに必要な時間は、ベースライン初期化の失敗を引き起こすことがあります。一般的には、二重のベースライン初期化が問題を解決します。

[Tune Helper (調整ヘルパー)] タブ



Enable Tune Helper

このパラメーターは、ユーザーによる最小限の努力だけで、Tuner GUI との通信をサポートする機能を追加します。Tuner GUI を使用する場合は、この機能をチェックします。このオプションがチェックされていなくても、通信機能は引き続き提供されますが、何もしません。よって、調整が完了したり、調整方法を変更しても、これらの機能を削除する必要はありません。デフォルト – チェックされています。

Instance name for EZI2C component

このパラメーターは、使用しているデザイン中の EZI2C コンポーネントのインスタンス名を定義し、Tuner GUI との通信で使用できるようにします。

注 実際のインスタンス名が、ここで入力されたインスタンス名と一致するかどうかをリアルタイムで確認する [Design Rule Check (デザインルールチェック)] はありません。それらが一致することを確認してください。名前が一致していないと、名前違いの API によってプロジェクトビルトの最中にビルトエラーが生じます。

Tuner GUI の詳細については、このデータシートの「CapSense Tuner GUI User Guide(CapSense Tuner GUI ユーザーガイド)」セクションをご参照ください。

配置

該当なし

リソース

CapSense CSD アナログおよびピンリソース。

リソース	アナログリソース			ピン (外部入出力ごと)
	VIDAC	コンパレータ	CapSense Buffers	
チャネル:1 現在のモード: 外部抵抗	0	1	1	2 + シールド + センサ番号
チャネル:2 現在のモード: 外部抵抗	0	2	2	4 + シールド + センサ番号
チャネル:1 現在のモード: IDAC シンキング	1	1	1	1 + シールド + センサ番号
チャネル:2 現在のモード: IDAC シンキング	2	2	2	2 + シールド + センサ番号
チャネル:1 現在のモード: IDAC ソーシング Vref: VDAC	2	1	1	1 + シールド + センサ番号
チャネル:2 現在のモード: IDAC ソーシング Vref: VDAC	4	2	2	2 + シールド + センサ番号

CapSense CSD デジタルリソース(スキャニングおよびスリープ API だけが Flash と RAM の使用法に含まれています)。

説明	デジタルリソース						API メモリ (バイト)	
	データパス	マクロセル	ステータスレジスタ	コントロールレジスタ	Counter 7	割り込み	フラッシュ	RAM
チャネル:1 現在のモード: 外部抵抗	4	19	0	1	1	1	1270	11
チャネル:2 現在のモード: 外部抵抗	6	31	0	1	1	2	2262	17
チャネル:1 現在のモード: IDAC シンキング	4	19	0	1	1	1	1345	10
チャネル:2 現在のモード: IDAC シンキング	6	31	0	1	1	2	2446	15
チャネル:1 現在のモード: IDAC ソーシング Vref: VDAC	4	18	0	1	1	1	1452	11
チャネル:2 現在のモード: IDAC ソーシング Vref: VDAC	6	30	0	1	1	2	2656	17

CapSense CSD 高レベル API リソース

プロジェクト詳細	API メモリ (バイト)	
	フラッシュ	RAM
ウィジェット種類:ボタン カウント: 4	1197	22
ウィジェット種類:ダイプレクスしていないリニア スライダ サイズ:5 センサ	1866	25
ウィジェット種類:ダイプレクスしているリニア スライダ サイズ:5 センサ	2304	25
ウィジェット種類:マトリックスボタン サイズ:5x5 センサ	1526	55
ウィジェット種類:ラジアル スライダ サイズ:5 センサ	1704	25
ウィジェット種類:タッチパッド サイズ:5x5 センサ	2289	48

「Tuner GUI User Guide (調整 GUI ユーザーガイド)」

このセクションでは、CapSense Tuner の使用に際し役立つ説明と情報を記載します。

「CapSense Tuner (CapSense 調整)」はマニュアル調整モードで、CapSense コンポーネントをシステムの特定の環境に調整する手助けをします。コンポーネントが「SmartSense」モードの場合は、調整値 (読みだしのみ) と性能を表示することもできます。コンポーネントが非調整モードの場合には調整はサポートされません。全てのパラメーターは Flash に格納されていて、最小の SRAM 使用の読みだしだけが可能だからです。

「CapSense Tuning Process (CapSense 調整プロセス)」

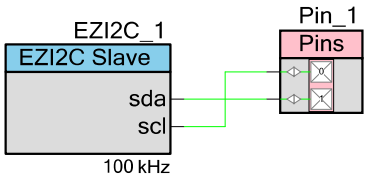
以下は、CapSense コンポーネントを使用し調整する際の典型的なプロセスです。

「PSoC Creator (PSoC Creator) で設計する」

必要に応じて「PSoC Creator Help (PSoC クリエイター ヘルプ)」をご参照ください。

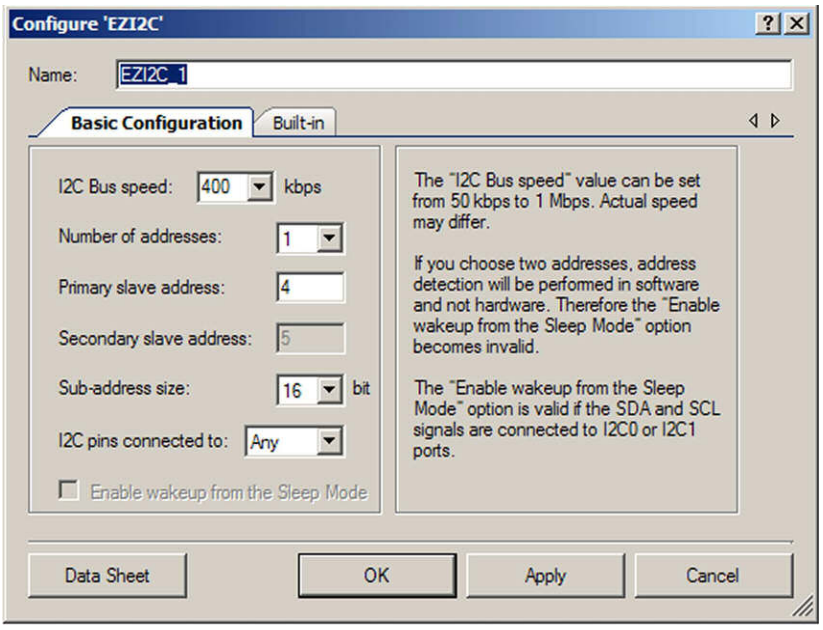
「Place and Configure EZI2C Component (EZI2C コンポーネントを配置、設定する)」

1. コンポーネントカタログから EZI2C をデザインの上にドラッグします。



ダブルクリックして [Configure] ダイアログを開きます。

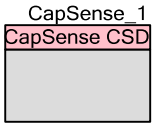
パラメーターを以下のように変更し、**OK** をクリックしてダイアログを閉じます。



- サブアドレスは「16 ビット」である必要があります。
- インスタンス名は、生成された API が機能するために、**[Tune Helper]** タブの下の、CapSense CSD デザイン ダイアログで使用されている名前と一致する必要があります。

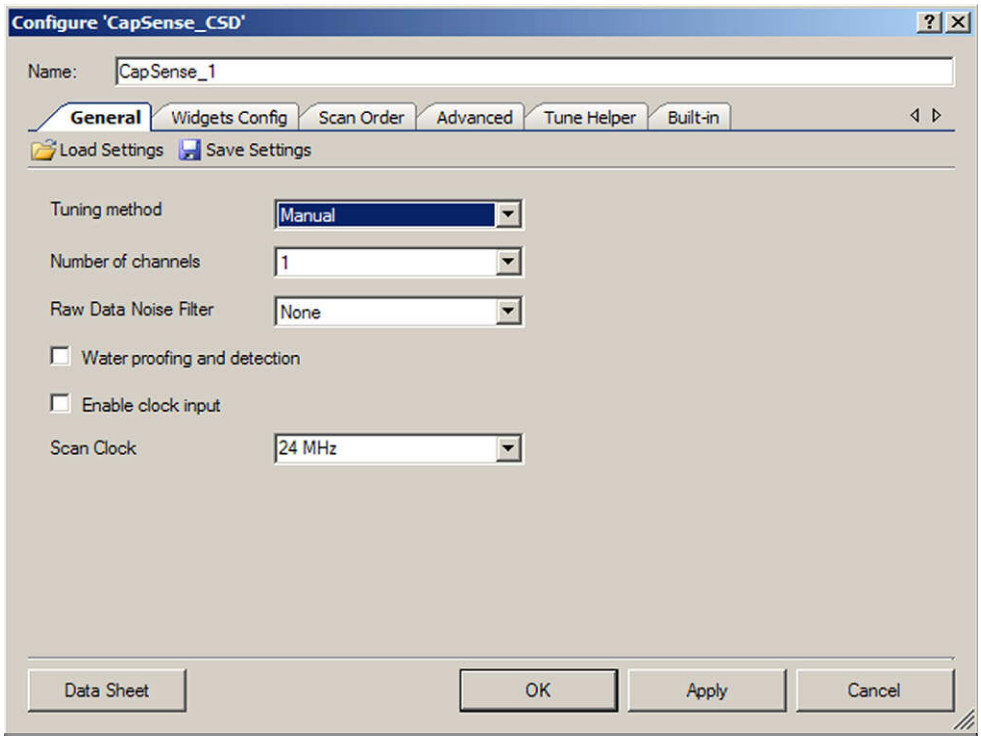
「Place and Configure the CapSense Component (CapSense コンポーネントを配置、設定する)」

1. コンポーネントカタログから CapSense_CSD コンポーネントををデザインの上にドラッグします。



ダブルクリックして [Configure] ダイアログを開きます。

使用しているアプリケーションで必要なように、CapSense CSD パラメーターを変更します。**Tuning method** をマニュアルまたは自動から選んでください (SmartSense)。**OK** をクリックしてダイアログを閉じ、選択したパラメーターを保存します。



「Selecting Auto (自動を選択する)(SmartSense)」

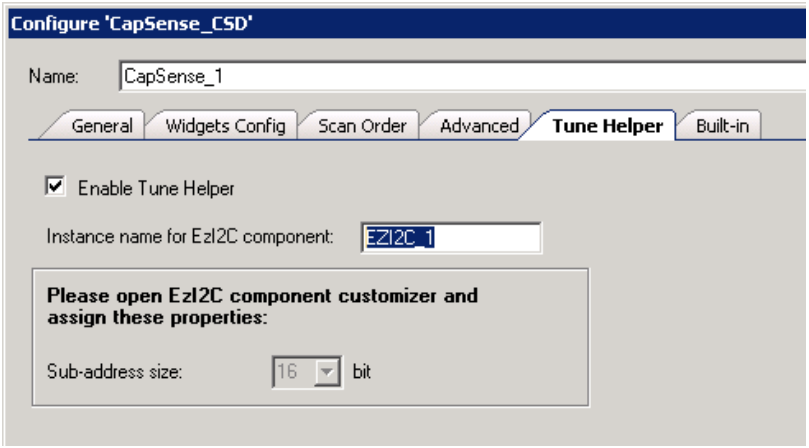
自動 (SmartSense) では、CapSense CSD コンポーネントを、システムの要件に合わせて自動的に調整することができます。CapSense CSD パラメーターは、ランタイム中にファームウェアによって計算されます。このモードでは、追加の RAM および CPU タイムが使用されます。自動 (SmartSense) は CapSense CSD コンポーネント パラメーターをマニュアルで調整する際のエラー傾向や反復プロセスを取り除き、適切なシステム オペレーションを確保します。自動 (SmartSense) 調整を選択すると、以下の CSD パラメーターを調整します:

パラメータ	計算
指の閾値	センサスキャニングの間継続的に計算される。
ノイズ閾値	センサスキャニングの間継続的に計算される。
IDAC Value	CapSense CSD スタートアップ時に一度だけ計算される。
Analog Switch Divider	CapSense CSD スタートアップ時に一度だけ計算される。
スキャン分解能	CapSense CSD スタートアップ時に一度だけ計算される。

自動(SmartSense) 調整方法向けハードウェア パラメーターの制限:

パラメータ	必要な設定
「Scan Clock（スキャンクロック）」	内部クロックが必須 ([General (全般)] タブ の “Enable clock input” (クロック入力をイネーブルにする)を偽に設定)。
Current Source	IDAC ソーシング。
PRS EMI Reduction	16 ビットがイネーブル。
Scan Speed	通常
Vref	1.024 V

「Tune Helper tab (調整ヘルパータブ)」で: EzI2C コンポーネントのインスタンス名が入力されていて、[Enable Tune Helper (調整ヘルパーをイネーブルにする)] チェックボックスがチェックされている必要があります



コードの追加

プロジェクトの *main.c* ファイルに、調整初期化および通信コードを追加します。例 *main.c* ファイル:

```
void main()
{
    CYGlobalIntEnable;
    CapSense_1_TunerStart();
    while(1)
    {
        CapSense_1_TunerComm();
    }
}
```

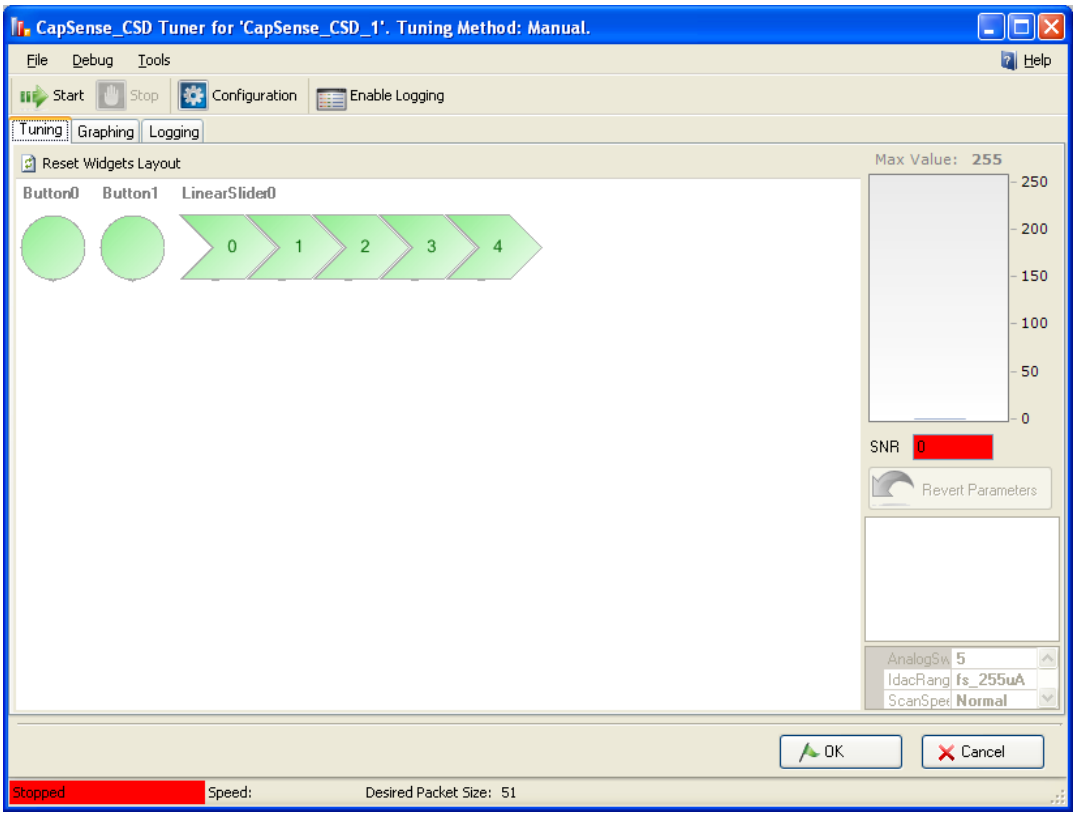
「Build the design and program the PSoC device (デザインをビルドし、PSoC デバイスをプログラムする)」

必要に応じて PSoC Creator のヘルプを参照してください。

調整アプリケーションを起動する

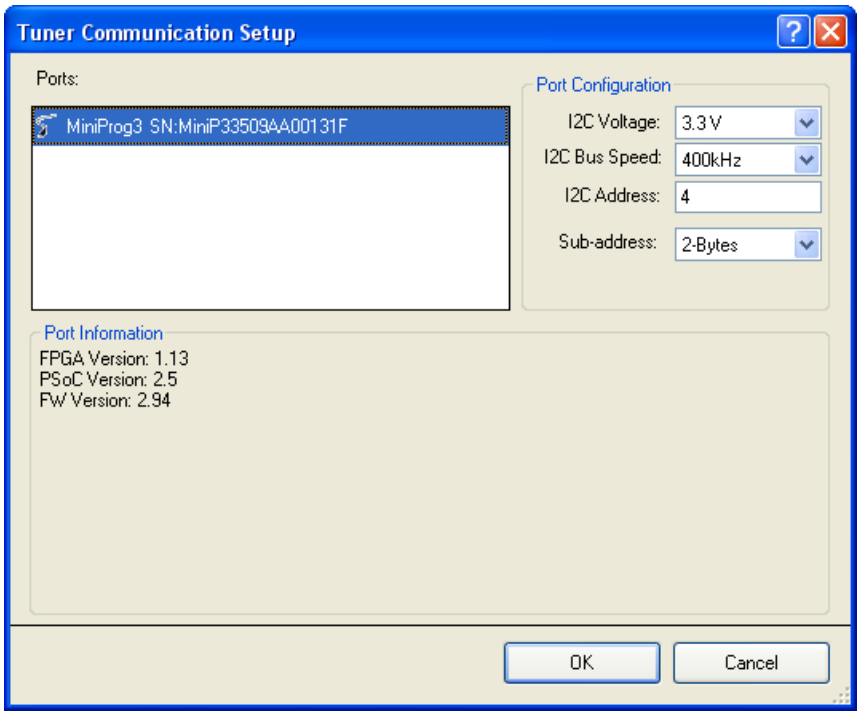
CapSense CSD コンポーネント アイコンを右クリックして、コンテクスト メニューから **Launch Tuner** を選択します。

調整アプリケーションが開きます。



通信パラメーターを設定します

1. [Configuration] をクリックして[Tuner Communication] ダイアログを開きます。



通信パラメーターを設定して OK をクリックします。

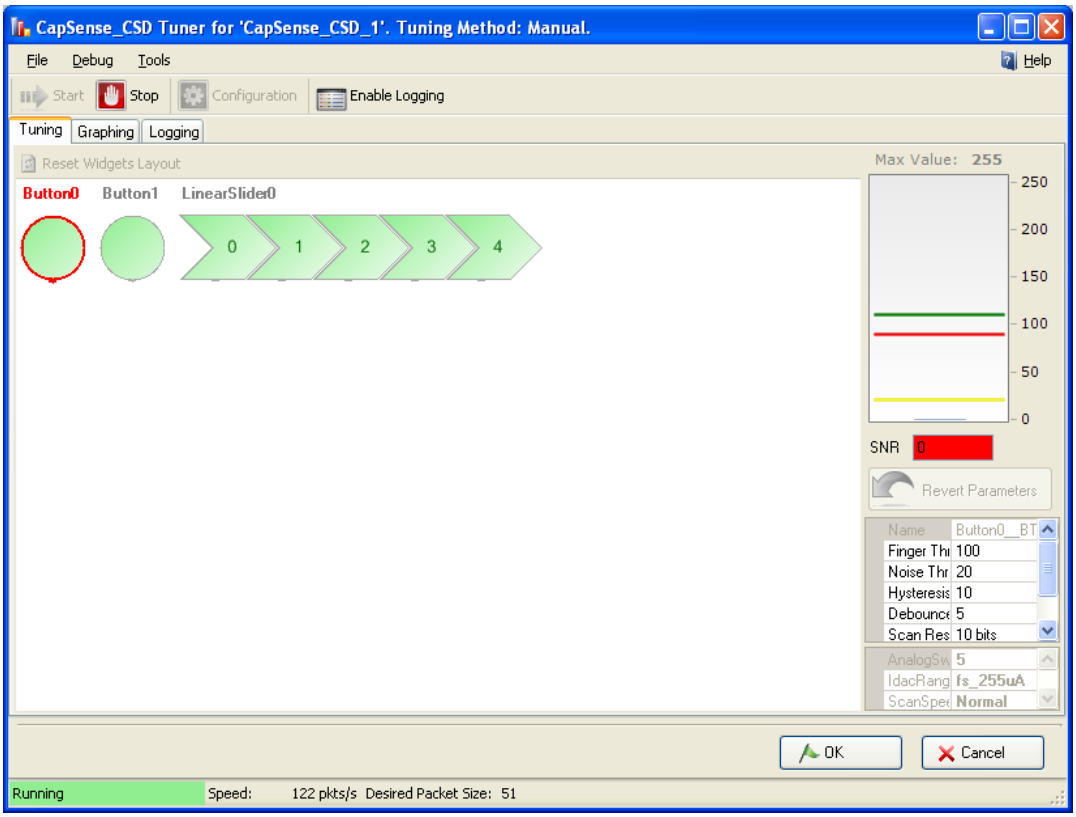
重要: プロパティ—は EZ I^2C コンポーネントのプロパティ—と同一である必要があります：**I2C Bus Speed**、**I2C Address**、**Sub-address** = 2-バイト。

調整の開始

調整 GUI の **Start** をクリックします。CapSense エレメント全てがその値を表示を開始します。

CapSense パラメーター値を編集する

1 つのエレメントの 1 つのパラメーター値を編集すると、「入力」キーを押すか他のオプションに移動した後で、自動的に適用されます。GUI は引き続きスキャニングデータを表示しますが、更新されたパラメーターのアプリケーションに基づいて変更されています。このデータシートの後にでてくる、「調整 GUI インターフェース」のセクションを参照してください。



必要に応じて繰り返します

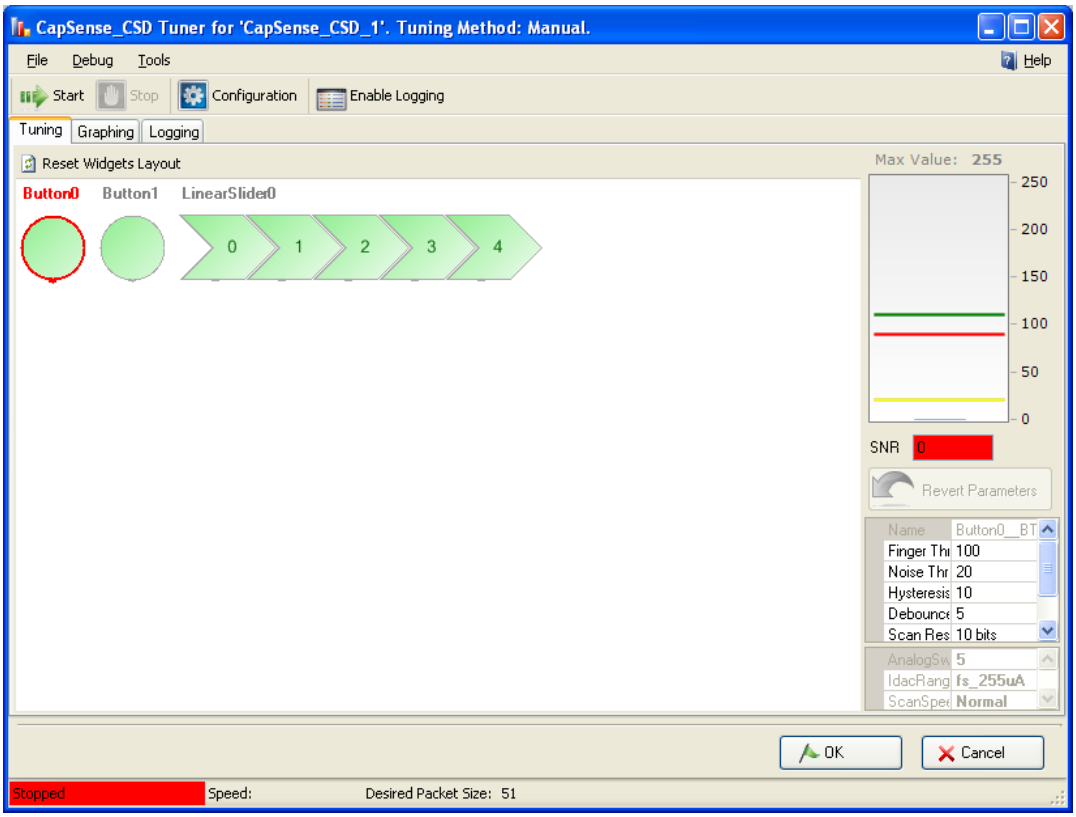
調整が完了し、CapSense コンポーネントが信頼性のあるタッチセンス結果を提供するまで、必要に応じてステップを繰り返します。

調整アプリケーションを閉じます

OK ボタンをクリックすると、パラメーターは CapSense_CSD インスタンスにライトバックされ、調整アプリケーションダイアログを閉じます。

調整 GUI インターフェース

全般インターフェース



トップパネルボタン:

- **Start** (またはメインメニューアイテム **Debug > Start**) – チップからのデータの読み込みと表示を開始します。設定されていれば、グラフ化やログ化も開始します。
- **Stop** (またはメインメニューアイテム **Debug > Stop**) – チップからのデータの読み込みと表示を停止します。
- **Configuration** (またはメインメニューアイテム **Debug > Configuration**) – 通信設定ダイアログを開きます。
- **Enable Logging** (またはメインメニューアイテム **Debug > Start**) – デバイスからのデータのログファイルへの記録をイネーブルにします。

メインメニュー:

- **File > Settings > Load Settings from File** – XML 調整ファイルから設定をインポートして、全てのデータをチューナーに読み込みます。
- **File > Help** – ヘルプファイルを開きます。

- ❑ 他のアイテムはトップおよびボトムパネルボタンと機能が重複しています。

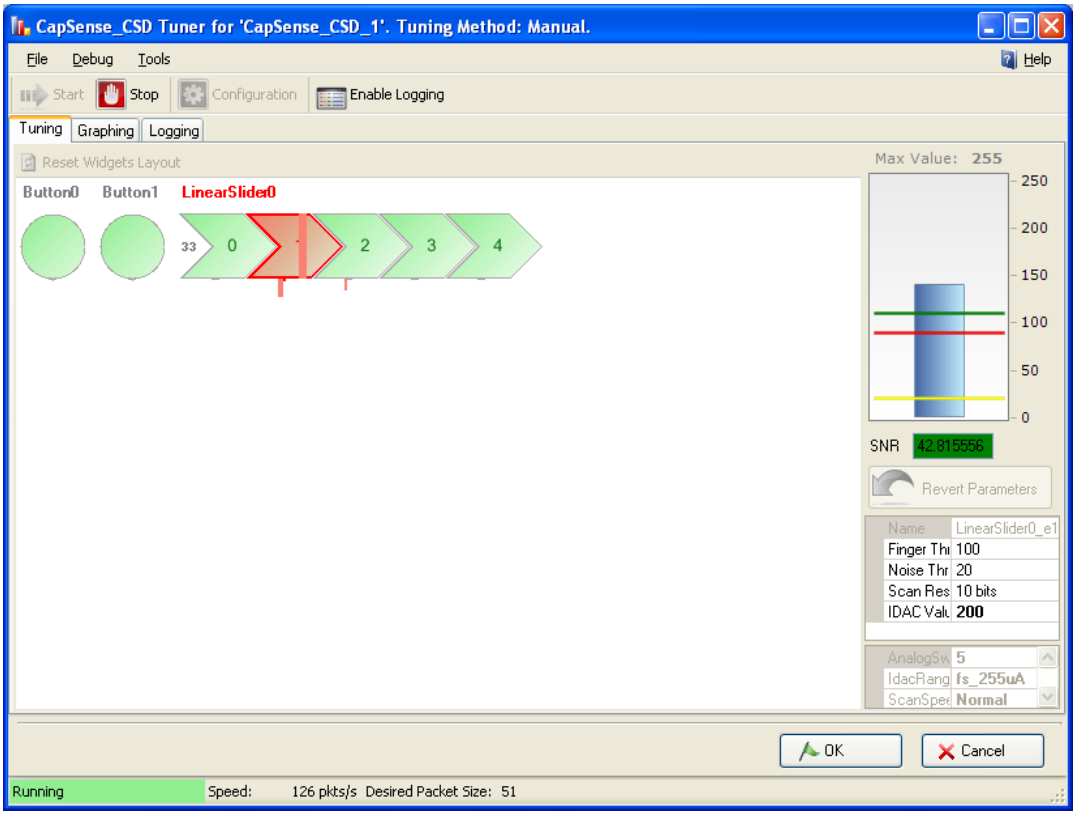
タブ

- **[Tuning] タブ** – 全てのコンポーネント ウィジットをワークスペースで設定されているように表示します。これにより、物理的な PCB または筐体のなかに存在するのと同じように、ウィジットを配置できます。このタブはウィジットのパラメーター調整と、ウィジットデータと状態の可視化に使われます。
- **Graphing Tab** – 個々の詳細なウィジットデータをチャート上に表示します。
- **Logging Tab** – データのログ機能と、デバッグ機能を提供します。

ボトムパネルボタン:

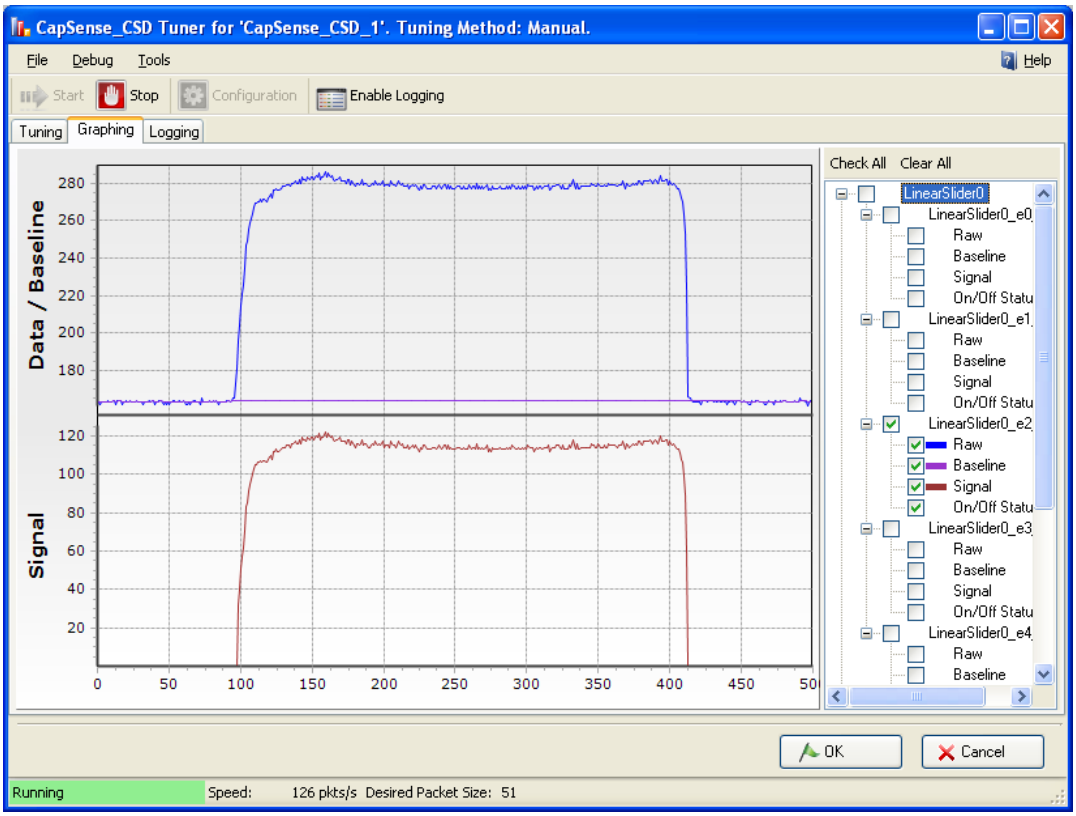
- **OK** (またはメインメニューアイテム **File > Apply Changes and Close**) – パラメーターの現在の値を CapSense コンポーネント インスタンスに適用し、GUI を終了する。
- **Cancel** (またはメインメニューアイテム **File > Exit**) – パラメーターの値をコンポーネントのインスタンスに適用せずに GUI を終了する。

[Tuning (調整タブ)] タブ



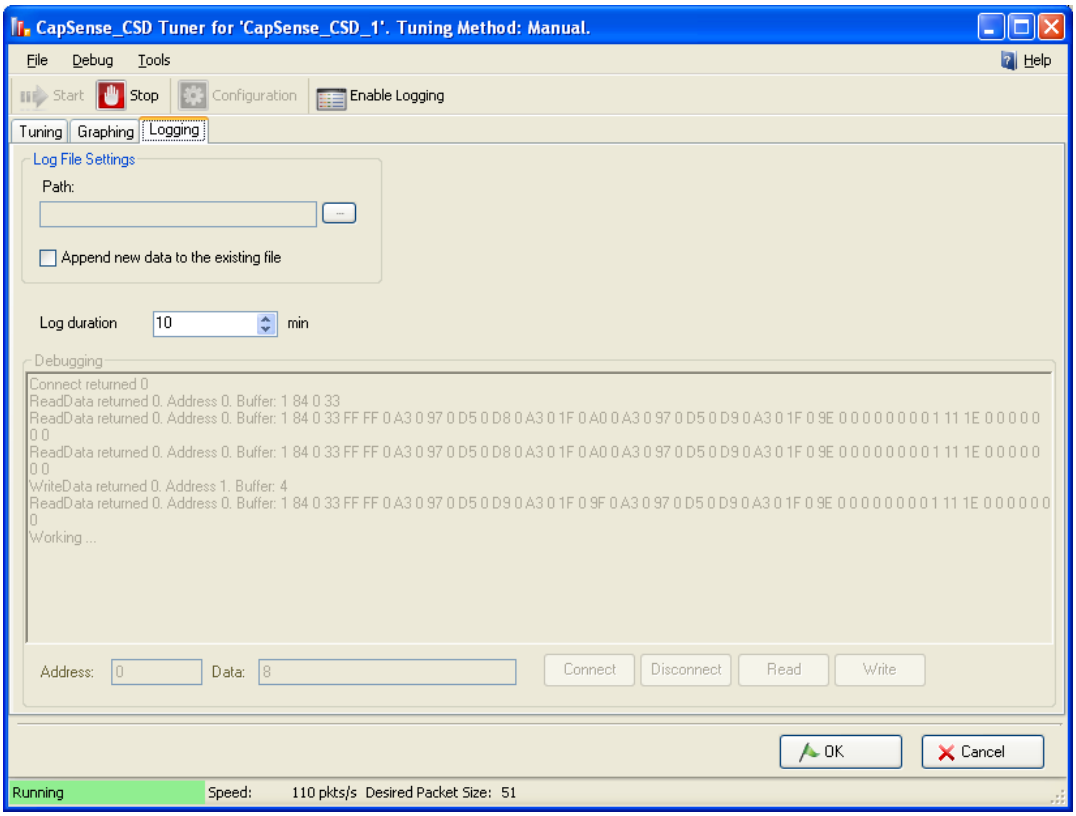
- **Widgets schematic** – 設定済のウィジット全てのグラフィック表示を補完します。ウィジットが複数のセンサから構成されている場合は、個々のセンサを選択して詳細分析を行います。
- **Bar graph** – 選択したセンサの信号値を表示します。
  - 詳細ビューバーグラフの最大メモリは、「Max Value(最大値)」ラベルをダブルクリックして調整可能です。値の範囲は 1 から 255 の間ですが、デフォルト値は 255 です。
  - 電流指ターンオン閾値はグラフ上に **green line** として表示されます。
  - 電流指ターンオフ閾値はグラフ上に **red line** として表示されます。
  - 電流ノイズ閾値はグラフ上に **yellow line** として表示されます。
- **SNR** – S/N 比(信号対ノイズ比)は選択したセンサについてリアルタイムで計算されます。5 以下の SNR(S/N 比)値は 品質が悪く赤色で表示され、5 ～ 10 は最低限で黄色で表示され、10 以上は良好で緑色で表示されます。SNR(S/N 比)値は以前に受け取ったデータに基づいて計算されます。
- **Revert Parameters** ボタン – パラメーターを初期値にリセットし、その値をチップに送ります。初期値とは、GUI が起動した際に表示された値です。
- **Sensor properties** – ウィジット種類に基づいて選択したセンサのプロパティを表示します。パネルの右側にあります。
- **General CapSense properties** (読みだしのみ) – CapSense CSD コンポーネントのグローバルプロパティを表示します。ランタイム中は変更できません。参照用途のみです。パネルの右側下部にあります。
- **Widget controls context menu** (この機能は GUI 中のウィジットコントロールのレイアウトにのみ適用できます):
  - 背後に送る – ウィジットコントロールをビューの背後に送る。
  - 前面に持ってくる – ウィジットコントロールをビューの前面に持ってくる。
  - 時計回りに 90 度回転 – ウィジットコントロールを時計回りに 90 度回転します。(リニア スライダ のみ)
  - 反時計回りに 90 度回転 – ウィジットコントロールを反時計回りに 90 度回転します。(リニア スライダ のみ)
  - センサの反転 – センサの順番を逆にする。(線形およびラジアル スライダ のみ)
  - コラムセンサの反転 – コラムセンサの順番を逆にする。(タッチパッドおよびマトリックスボタンのみ)
  - 行センサの反転 – 行センサの順番を逆にする。(タッチパッドおよびマトリックスボタンのみ)
  - 列と行を交換する – 列センサが行センサになり、行センサが列センサになります。(タッチパッドおよびマトリックスボタンのみ)

[Graphing (グラフタブ)] タブ



- **Chart area** – ツリービューから選択したアイテムのチャートを表示する。
  - [Chart area (チャートエリア)] でコンテクストメニューアイテムを右クリックすると、**Export to .jpg** がチャートエリアのスクリーンショットを作成し、.jpg ピクチャとして保存します。
- **Tree view** – ウィジットおよびセンサ向けのデータの組み合わせを全て提供し、ログ化機能がイネーブルになっていれば、ファイルに記録します。On/Off 状況値は記録されるだけで、チャート上には表示されません。

[Logging (ログ)] タブ

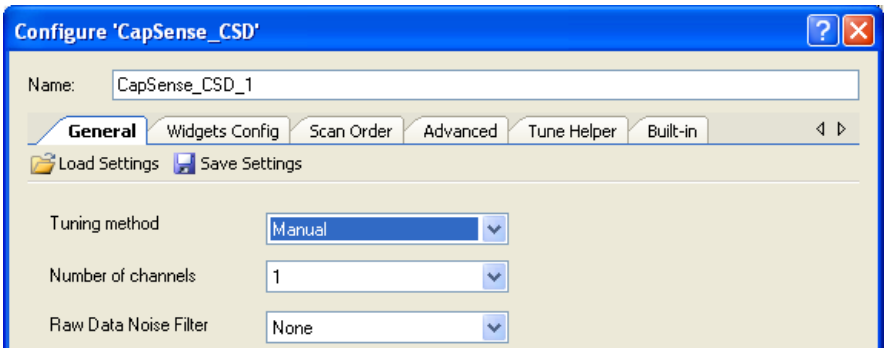


- グラフタブのツリービューにあるチェックボックスにチェックを入れ、記録されたデータを選択します。
- パス – ログファイルへのパスを定義します(ファイルの拡張子は .csv)。
- 新しいデータを既存のファイルチェックボックスに追加します – チェックされていると、新しいデータが既存のファイルに追加されます。チェックされていないと、古いデータがファイルから消去され、新しいデータに置き換わります。
- ログ期間 – ログ期間を分単位で定義します。
- 接続 – PSoC デバイスに接続します。
- 切断 – PSoC デバイスとの接続を切断します。
- 読みだし – PSoC デバイスからデータを読み出します。アドレス フィールドは、バッファ内のアドレスを定義します。データ フィールドは、読み出されるバイト数を定義します。
- 書き込み – PSoC デバイスにデータを書き込みます。アドレス フィールドは、バッファ内のアドレスを定義します。データ フィールドは、書き込まれるデータを定義します。

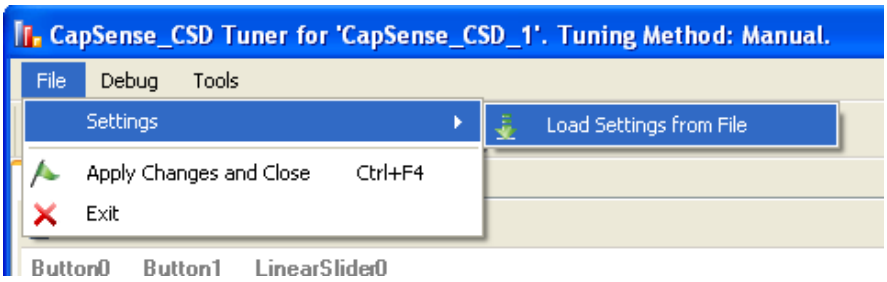
保存/読み込み 設定機能

チューナー GUI はスタンドアローン アプリケーションとしても開くことができます。この場合、ユーザーは CapSense CSD コンポーネント Tuner GUI の保存および読み込み設定機能を使用する必要があります。

1. カスタマイザの「Save Settings(保存設定)」ボタンをクリックします。



2. [Save File (ファイルを保存)] ウインドウで、ファイル名と保存先を指定します。
3. [Tuner (調整)] ウィンドウを開いて、「File > Settings > Load Settings (ファイル＞設定＞設定を読み込み)」をクリックします。



4. [File Open (ファイルを開く)] ダイアログで、そのコンポーネント設定で以前に保存されたファイルをクリックします。設定が自動的に Tuner に読みこまれます。

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。以下の表は各関数の概要を示します。その次のセクションでは、各関数について詳しく説明します。

デフォルトでは、PSoC Creator はインスタンス名「CapSense_1」をデザイン上のコンポーネントの最初のインスタンスに割り当てます。コンポーネントのインスタンス名は、識別子の文法ルールに従って固有の名前に変更できます。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「CapSense」というインスタンス名を使用しています。

全般 API

これらは、コンポーネントをオペレーション状態にしたり、停止したりする、一般的な CapSense API 関数です:

関数	説明
CapSense_Start	コンポーネントを開始する望ましい方法です。レジスタを初期化し、CapSense 内で使用されたサブコンポーネントのアクティブ モードの電源テンプレートビットをイネーブルにします。
CapSense_Stop	コンポーネントの割り込みをディスエーブルにし、全てのセンサを非アクティブ状態にするために CapSense_ClearSensors() を呼び出します。
CapSense_Sleep	低電力モードに入ったデバイス向けのコンポーネントを用意します。CapSense 内で使用されたサブコンポーネントのアクティブ モードの電源テンプレートビットをディスエーブルにし、ノン-リテンション レジスタを保存し、全てのセンサを非アクティブ状態にリセットします。
CapSense_WakeUp	デバイスが低電力スリープモードからウェイクした後、CapSense 設定とノン-リテンション レジスタ値を回復します。
CapSense_Init	カスタマイズで、デフォルト CapSense 設定を初期化します。
CapSense_Enable	CapSense 内で使用されたサブコンポーネントのアクティブ モードの電源テンプレートビットをイネーブルにします。
CapSense_SaveConfig	CapSense ノン-リテンション レジスタの設定を保存します。全てのセンサを非アクティブ状態にリセットします。
CapSense_RestoreConfig	CapSense 設定とノン-リテンション レジスタ値を回復します。

void CapSense_Start (void)

説明: これは、コンポーネントのオペレーションを開始する際に推奨される方法です。CapSense_Start() は CapSense_Init() 関数を呼び出してから、CapSense_Enable() 関数を呼び出します。レジスタを初期化して、CapSense コンポーネントの CSDメソッドを開始します。全てのセンサを非アクティブ状態にリセットします。センサスキャニングの割り込みをイネーブルにします。SmartSense 調整モードが選択されている場合は、調整手順を全てのセンサに適用します。CapSense_Start() ルーチンは、他の API ルーチンよりも先に呼び出される必要があります。

パラメータ: なし

戻り値: なし

副作用: なし

void CapSense_Stop (void)

説明: センサスキャニングを停止し、コンポーネントの割り込みをディスエーブルにし、全てのセンサを非アクティブ状態にリセットします。CapSense 内で使用されたサブコンポーネントのアクティブ モードの電源テンプレートビットをディスエーブルにします。

パラメータ: なし

戻り値: なし

副作用: スキャニングを全て完了した後で、この関数を呼び出します。

void CapSense_Sleep(void)

説明: これは、低電力モードのデバイス向けにコンポーネントを準備する際の望ましい方法です。CapSense 内で使用されたサブコンポーネントのアクティブ モードの電源テンプレートビットをディスエーブルにします。CapSense_SaveConfig() 関数を呼び出して CapSense ノン-リテンション レジスタのカスタマ設定を保存し、全てのセンサを非アクティブ状態にリセットします。

パラメータ: なし

戻り値: なし

副作用: スキャニングを全て完了した後で、この関数を呼び出します。

この関数は、CapSense コンポーネントで使用されたピンを最低の消費電力状態にしません。ピンのドライブモードを変更するには、「Pins APIs」セクションに記載の関数を使用します。

void CapSense_WakeUp(void)

説明: CapSense 設定とノン-リテンション レジスタ値を回復します。CapSense 内で使用されたサブコンポーネントのアクティブ モードの電源テンプレートビットを設定することで、コンポーネントのイネーブル状態を回復します。

パラメータ: なし

戻り値: なし

副作用: この関数は、CapSense コンポーネントで使用されたピンを、以前の状態に回復しません。

void CapSense_Init(void)

説明: カスタマイザから提供されたコンポーネントのオペレーションを決定する CapSense デフォルト設定を初期化して、全てのセンサを非アクティブ状態にリセットします。

パラメータ: なし

戻り値: なし

副作用: なし

void CapSense_Enable(void)

説明: CapSense 内で使用されたサブコンポーネントのアクティブ モードの電源テンプレートビットをイネーブルにします。

パラメータ: なし

戻り値: なし

副作用: なし

void CapSense_SaveConfig(void)

説明: CapSense ノン-リテンション レジスタの設定を保存します。全てのセンサを非アクティブ状態にリセットします。

パラメータ: なし

戻り値: なし

副作用: スキャニングを完了した後で、この関数を呼び出します。
この関数は、CapSense コンポーネントで使用されたピンを最低の消費電力状態にしません。ピンのドライブモードを変更するには、「Pins APIs」セクションに記載の関数を使用します。

void CapSense_RestoreConfig(void)

説明: CapSense 設定とノン-リテンション レジスタ値を回復します。

パラメータ: なし

戻り値: なし

副作用: スキャニングを完了した後で、この関数を呼び出します。
この関数は、CapSense コンポーネントで使用されたピンを、以前の状態に回復しません。

「Scanning Specific APIs (特定の API をスキャニングする)」

これらの API 関数は、CapSense センサスキャニングを実装するために使用されます。

関数	説明
CapSense_ScanSensor	スキャン設定を設定し、各チャネルの1つのセンサまたは複数のセンサグループのスキャニングを開始します。
CapSense_ScanEnabledWidgets	望ましいスキャニング方法です。イネーブルなウィジットを全てスキャンします。
CapSense_IsBusy	センサスキャニングの状態を返します。
CapSense_SetScanSlotSettings	選択したスキャンスロット (センサまたは対のセンサ) のスキャン設定を設定します。
CapSense_ClearSensors	全てのセンサを非サンプリング状態にリセットします。
CapSense_EnableSensor	次のスキャニングサイクルでスキャンするために選択されたセンサを構成します。

関数	説明
CapSense_DisableSensor	選択したセンサをディスエーブルにし、次のスキャニングサイクルでスキャンされないようにします。
CapSense_ReadSensorRaw	CapSense_SensorResult[ ] 配列からセンサの生データを返します。
CapSense_SetRBleed	複数のブリーダー抵抗が使用されている場合に、ピンをブリーダー抵抗(Rb)で使用するように設定します。

void CapSense_ScanSensor (uint8 sensor)

説明: スキャン設定を設定し、各チャネルの 1 つのセンサまたは 1 組みのセンサのスキャニングを開始します。2 つのチャネルが設定されていると、同時に 2 つのセンサをスキャンすることもあります。スキャニングが完了すると、isr が測定したセンサ生データをグローバル Raw センサ配列にコピーします。isr を使用すると、この関数がブロックされていないことを確認できます。各センサは、センサアレイ内で独自の番号を持っています。この番号は、CapSense カスタマイザが順番に割り当てます。

パラメータ: (uint8) センサ: センサ番号

戻り値: なし

副作用: なし

void CapSense_ScanEnabledWidgets (void)

説明: イネーブルなウィジットを全てスキャンする望ましい方法です。イネーブルなウィジット内の 1 つのセンサまたは 1 組みのセンサのスキャニングを開始します。全てのイネーブルなウィジットがスキャンされるまで、isr はスキャニングセンサを継続します。isr を使用すると、この関数がブロックされていないことを確認できます。近接ウィジット以外のウィジットは全てデフォルトでイネーブルになっています。近接ウィジットは、スキャン時間が長く他のウィジット種類が望む素早いレスポンスと互換性がないため、マニュアルでイネーブルにしてください。

パラメータ: なし

戻り値: なし

副作用: イネーブルなウィジットがないと、関数の呼び出しは何の影響も持ちません。

uint8 CapSense_IsBusy (void)

説明: センサスキャニングの状態を返します。

パラメータ: なし

戻り値: (uint8) スキャニングの状態を返します。'1' – スキャニングが進行中 '0' – スキャニングが完了。

副作用: なし

void CapSense_SetScanSlotSettings (uint8 slot)

説明: カスタマイザ内のスキャン設定、または選択したスキャンスロットのウィザードを設定します(2 チャネルのデザインにはセンサまたは1 組のセンサ) 。スキャン設定は全てのセンサに分解能と同様に IDAC 値 (IDAC 設定向け) も提供します。ウィジット内の全てのセンサで分解能は同じです。

パラメータ: (uint8) スロット: スロット番号をスキャンします

戻り値: なし

副作用: なし

void CapSense_ClearSensors (void)

説明: 全てのセンサの Analog MUX バスとの接続を順番に切断し非サンプリング状態にリセットして、全てのセンサを非アクティブ状態に接続する。

パラメータ: なし

戻り値: なし

副作用: なし

void CapSense_EnableSensor(uint8 sensor)

説明: 次の測定サイクルでスキャンするために選択されたセンサを構成します。対応するピンは Analog High-Z モードに設定されていて、Analog Mux バスに接続しています。コンパレータ出力にも影響します。

パラメータ: (uint8) センサ: センサ番号

戻り値: なし

副作用: なし

「void CapSense_DisableSensor (uint8 sensor)」

説明: 選択したセンサをディスエーブルにします。対応するピンは Analog Mux Bus との切断を遮断し、非アクティブ状態におかれます。

パラメータ: (uint8) センサ: センサ番号

戻り値: なし

副作用: なし

「uint16 CapSense_ReadSensorRaw (uint8 sensor)」

説明: グローバル CapSense_SensorResult[ ] 配列からセンサの生データを返します。各スキャンセンサは、センサアレイ内で独自の番号を持っています。この番号は、CapSense カスタマイザが順番に割り当てます。生データは CapSense が提供したフレームワークの外での計算を実行するために使用されることがあります。

パラメータ: (uint8) センサ: センサ番号

戻り値: (uint16) 現在の生データ値。

副作用: なし

「void CapSense_SetRBleed(uint8 rbleed)」

説明: ピンをブリーダー抵抗 (Rb) との接続に使えるように設定します。この関数は、カスタマイザで定義されている現在の Rb ピン設定を選択するために、ランタイム中に呼び出されることがあります。関数は、コンポーネントのパラメータ設定を上書きします。この関数は、[Current Source] が「External Resistor(外部レジスタ)」に設定されているときにのみ利用可能です。

複数のセンサを異なるブリーダー抵抗の値でスキャンする必要がある場合に、この機能は効果的です。例えば、通常のボタンは値の低いブリーダー抵抗でスキャンすることができます。近接検知器はより大きなスキャン間隔で、近接検知距離を最大化するために大きなブリード抵抗値でスキャンします。この関数は CapSense_ScanSensor() 関数とともに使用できます。

パラメータ: (uint8) rbleed: ブリーダー抵抗の順番は CapSense カスタマイザに定義されています。

戻り値: なし

副作用: ブリーダー抵抗の数は 3 に制限されています。この関数は範囲外の数をチェックしません。

ハイレベル API

これらの API 関数はセンサウィジェットの生データと一緒に使用されます。生データをスキャンしたセンサから受け取り、ボタン向けにはオン／オフに、スライダ向けには位置に、タッチパッド向けには X および Y 座標として変換します。

関数	説明
CapSense_InitializeSensorBaseline	選択されたセンサをスキャンして、初期値を伴う CapSense_SensorBaseline[sensor] アレイ エレメントを読み込みます。
CapSense_InitializeAllBaselines	それぞれのセンサをスキャンして、CapSense_SensorBaseline[] アレイに初期値を読み込みます。
CapSense_UpdateSensorBaseline	この経時的カウント値は、センサ別に独立して計算され、センサのベースラインと呼ばれます。このベースラインは、k = 256 の ローパス フィルターを使って更新されます。
CapSense_UpdateEnabledBaselines	CapSense_SensorEnableMask[] アレイをチェックし、CapSense_UpdateSensorBaseline 関数を呼び出して、イネーブルなセンサのベースラインを更新します。
CapSense_EnableWidget	スキャニングプロセスの間、ウィジェット内の全てのセンサ エレメントをイネーブルにします。
CapSense_DisableWidget	スキャニングプロセスから、ウィジェット内の全てのセンサ エレメントをディスエーブルにします。

関数	説明
CapSense_CheckIsWidgetActive	選択したウィジェットと CapSense_Signal[] アレイとを比較し、指プレスがあったかどうかを判断します。
CapSense_CheckIsAnyWidgetActive	CapSense_CheckIsWidgetActive() 関数を使って、CapSense CSD コンポーネントのウィジェットがアクティブ状態かどうかを確認します。
CapSense_GetCentroidPos	リニア スライダ における CapSense_SensorSignal[] アレイの指プレスをチェックし、位置を返します。
CapSense_GetRadialCentroidPos	ラジアル スライダ における CapSense_SensorSignal[] アレイの指プレスをチェックし、位置を返します。
CapSense_GetTouchCentroidPos	指があると、この関数はタッチパッドのセントロイドを計算することで、指の X および Y 位置を計算します。

void CapSense_InitializeSensorBaseline (uint8 sensor)

説明: 選択したセンサ(1 チャネル デザイン)または1 対のセンサ(2 チャネル デザイン)をスキャニングして、初期値を伴う CapSense_SensorBaseline[sensor] アレイ エレメントを読み込みます。生カウント値は、それぞれのセンサのベースラインのアレイにコピーされます。イネーブルになると、生データフィルタは初期化されます。

パラメータ: (uint8) センサ: センサ番号

戻り値: なし

副作用: なし

void CapSense_InitializeAllBaselines(void)

説明: それぞれのセンサをスキャンして、CapSense_SensorBaseline[ ] アレイに初期値を読み込むために、CapSense_InitializeSensorBaseline 関数を使います。生のカウント値は、それぞれのセンサのベースラインのアレイにコピーされます。イネーブルになると、生データフィルタは初期化されます。

パラメータ: なし

戻り値: なし

副作用: なし

void CapSense_UpdateSensorBaseline (uint8 sensor)

説明: センサのベースラインは経時的カウント値は、センサ別に独立して計算されます。k = 256 の ローパス フィルターを使って CapSense_SensorBaseline[sensor] アレイ エレメントを更新します。この関数は、電流生カウント値から以前のベースラインを減算して差カウントを計算し、CapSense_SensorSignal[sensor] に格納します。

自動リセットオプションがイネーブルである場合、ベースラインの更新はノイズ閾値と無関係です。

自動リセットオプションがディスエーブルである場合、信号がノイズ閾値より大きい場合はベースラインは更新を停止し、信号がマイナスノイズ閾値よりも小さい場合はベースラインはリセットします。

ベースライン計算の前にイネーブルになっていれば、生データフィルタが値に適用されます。

パラメータ: (uint8) センサ: センサ番号

戻り値: なし

副作用: なし

void CapSense_UpdateEnabledBaselines(void)

説明: CapSense_SensorEnableMask[] アレイをチェックし、CapSense_UpdateSensorBaseline 関数を呼び出して、イネーブルなセンサのベースラインを更新します。

パラメータ: なし

戻り値: なし

副作用: なし

void CapSense_EnableWidget (uint8 widget)

説明: 選択したウィジェットセンサをイネーブルにして、スキャニングプロセスの一部にする。

パラメータ: (uint8) widget: ウィジェット番号 全てのウィジェットには、以下の形式の定義があります:

```
#define CapSense_"widget_name"__"widget type"          5
```

例:

```
#define CapSense_MY_VOLUME1__LS                        5
#define CapSense_MY_UP__BNT                            6
```

ウィジェット名は全て大文字です。

戻り値: なし

副作用: なし

void CapSense_DisableWidget (uint8 widget)

説明: 選択したウィジェットセンサをスキャニングプロセスからディスエーブルにする。

パラメータ: (uint8) widget: ウィジェット番号 全てのウィジェットには、以下の形式の定義があります:

```
#define CapSense_"widget_name"__"widget type"          5
```

例:

```
#define CapSense_MY_VOLUME1__RS                        5
#define CapSense_MY_UP__MB                             6
```

ウィジェット名は全て大文字です。

戻り値: なし

副作用: なし

uint8 CapSense_CheckIsWidgetActive(uint8 widget)

説明: 選択したセンサである CapSense_Signal[ ] アレイ値を指閾値と比較します。Hysteresis と Debounce デバウンスを考慮にいれます。センサがアクティブである場合、ヒステレシス量によって閾値が下がります。非アクティブである場合、ヒステレシス量によって閾値が上がります。アクティブな閾値を満たすと、この API がウィジェットをアクティブと設定する、センサのアクティブ遷移に達するまで、Debounce カウンタが1ずつ増加します。この関数は、CapSense_SensorOnMask[ ] アレイでセンサのビットを更新します。

ウィジェットのアクティブ状態を返すために、タッチパッドおよびマトリックスボタンには列と行にアクティブセンサが必要です。

パラメータ: (uint8) widget: ウィジェット番号 全てのウィジェットには、以下の形式の定義があります:

```
#define CapSense_"widget_name"__"widget type"            5
```

例:

```
#define CapSense_MY_VOLUME1__LS                          5
```

ウィジェット名は全て大文字です。

戻り値: (uint8) ウィジェットセンサ状態 ウィジェット内の 1 つ以上のセンサがアクティブな場合は 1 、ウィジェット内の全てのセンサが非アクティブな場合は 0。

副作用: この関数はまた、CapSense_SensorOnMask[ ] にあるこのウィジェット内のセンサ全ての値を更新します。アクティブ状態への遷移がある場合は、デバウンスカウンタも呼び出し毎に修正されます。

uint8 CapSense_CheckIsAnyWidgetActive(void)

説明: CapSense_Signal[ ] アレイの全センサをその指閾値と比較します。各ウィジェットについて Capsense_CheckIsWidgetActive() を呼び出し、CapSense_SensorOnMask[ ] アレイが関数呼び出し後に最新の状態であるようにします。

パラメータ: なし

戻り値: (uint8) アクティブなウィジェットがある場合は 1 、ない場合は 0 。

副作用: CapSense_CheckIsWidgetActive() 関数と同じ副作用がありますが、全センサに対してです。

uint16 CapSense_GetCentroidPos(uint8 widget)

説明: CapSense_Signal[ ] アレイをチェックして、リニア スライダ 内で指プレスがあったかどうかを確認します。指位置は、CapSense カスタマイザで指定した API 分解能を計算します。イネーブルな場合は、位置フィルタが結果に適用されます。この関数は、リニア スライダ ウィジェットが CapSense カスタマイザで定義されている場合のみ利用できます。

パラメータ: (uint8) widget: ウィジェット番号 全てのリニア スライダ ウィジェットには、以下の形式の定義があります:

```
#define CapSense_"widget_name"__LS                    5
```

例:

```
#define CapSense_MY_VOLUME1__LS                       5
```

ウィジェット名は全て大文字です。

戻り値: (uint16) リニア スライダ の位置値。

副作用: スライダウィジェット内のセンサが1つでもアクティブな場合、この関数は 0 から CapSense カスタマイザで設定されている API 分解能の間の値を返します。センサが1つもアクティブでない場合、関数は 0xFFFF を返します。セントロイド/ダイプレックスアルゴリズムの実行中にエラーが発生した場合、関数は 0xFFFF を返します。

この関数に備わっているウィジェットの規則についてはチェックがありません。不正確なウィジェット値は、予想外の位置計算を引き起こします。

注 スライダセグメント上のノイズカウントがノイズ閾値より大きい場合、このサブルーチンは正しくない指プレス結果を示すことがあります。ノイズが偽の指プレス結果を生じないように、ノイズ閾値は慎重に設定します (ノイズレベルを超える高さ)。

uint16 CapSense_GetRadialCentroidPos(uint8 widget)

説明: CapSense_Signal[ ] アレイをチェックして、ラジアル スライダ 内で指プレスがあったかどうかを確認します。指位置は、CapSense カスタマイザで指定した API 分解能を計算します。イネーブルな場合は、位置フィルタが結果に適用されます。この関数は、ラジアル スライダ ウィジェットが CapSense カスタマイザで定義されている場合のみ利用できます。

パラメータ: (uint8) widget: ウィジェット番号 全てのラジアル スライダ ウィジェットには、以下の形式の定義があります:

```
#define CapSense_"widget_name"__RS                    5
```

例:

```
#define CapSense_MY_VOLUME2__RS                       5
```

ウィジェット名は全て大文字です。

戻り値: (uint16) ラジアル スライダ の位置値。

副作用: スライダウィジェット内のセンサが1つでもアクティブな場合、この関数は 0 から CapSense カスタマイザで設定されている API 分解能の間の値を返します。センサが1つもアクティブでない場合、関数は 0xFFFF を返します。

この関数に備わっているウィジェット種類の規則についてはチェックがありません。不正確なウィジェット値は、予想外の位置計算を引き起こします。

注 スライダセグメント上のノイズカウントがノイズ閾値より大きい場合、このサブルーチンは正しくない指プレス結果を示すことがあります。ノイズが偽の指プレス結果を生じないように、ノイズ閾値は慎重に設定します (ノイズレベルを超える高さ)。

uint8 CapSense_GetTouchCentroidPos (uint8 widget)

説明: タッチパッド上に指があると、この関数はタッチパッドセンサのセントロイドを計算することで、指の X および Y 位置を計算します。X および Y 位置は、CapSense カスタマイザで指定した API 分解能を計算します。指がタッチパッド上にあると、「1」を返します。イネーブルな場合は、位置フィルタが結果に適用されます。この関数は、タッチパッドが CapSense カスタマイザで定義されている場合のみ利用できます。

パラメータ: (uint8) widget: ウィジェット番号 全てのタッチパッドウィジェットには、以下の形式の定義があります:

```
#define CapSense_"widget_name"__TP                    5
```

例:

```
#define CapSense_MY_TOUCH1__TP                        5
```

ウィジェット名は全て大文字です。

戻り値: (uint8) 指がタッチパッド上にあると 1 、無い場合は 0 。

副作用: X および Y 位置の計算結果は、グローバルアレイに格納されます。アレイ名と位置は:

```
CapSense_position[widget]    - X の位置
CapSense_position[widget + 1]    - Y の位置
```

この関数に備わっているウィジェットの値についてはチェックがありません。不正確なウィジェット値は、予想外の位置計算を引き起こします。

「Tuner Helper APIs」

API 関数は「Tuner GUI」と共に用いられます。

関数	説明
CapSense_TunerStart	CapSense CSD および EZI2C コントロールを初期化し、ベースラインを初期化してセンサスキャニングループを開始します。
CapSense_TunerComm	「Tuner GUI」間の通信を実行します。

void CapSense_TunerStart (void)

説明: CapSense CSD コンポーネントおよび EZI2C コンポーネントを初期化します。ベースラインを初期化して、現在イネーブルなセンサでセンサスキャニングループを開始します。

パラメータ: なし

戻り値: なし

副作用: なし

void CapSense_TunerComm (void)

説明: Tuner GUI で通信関数を実行します。

- マニュアル モード: CapSense CSD コンポーネントから Tuner GUI へ、センサスキャニングとウィジェット処理の結果を転送します。Tuner GUI から新しいパラメーターを読みだして、CapSense CSD コンポーネントに適用します。
- 自動(SmartSense): Tuner GUI で通信関数を実行します。Tuner GUI へ、センサスキャニングとウィジェット処理の結果を転送します。自動調整パラメーターも、Tuner GUI に転送します。Tuner GUI パラメーターは、CapSense CSD コンポーネントに返送されません。

Tuner GUI が CapSense CSD コンポーネントバッファを新しいデータ向けに修正している間、この関数はブロックして待機します。

パラメータ: なし

戻り値: なし

副作用: なし

ピン API

これらの API 関数は、CapSense コンポーネントで使用されるピンのドライブ モードを変更するために使用します。これらの API は主に、デバイスが低電力モードのときに、リーク電流を最小化するために、CapSense CSD コンポーネントのピンを強力ドライブモードにするために使用されます。

関数	説明
CapSense_SetAllSensorsDriveMode	CapSense コンポーネントの静電容量センサで使用されているピン全てをドライブモードに設定します。
CapSense_SetAllCmodsDriveMode	CapSense コンポーネントの Cmod コンデンサで使用されているピン全てをドライブモードに設定します。

CapSense_SetAllRbsDriveMode	CapSense コンポーネントの ブリーダー抵抗で使用されているピン全てをドライブモードに設定します。[Current Source (電流源)] が「External Resistor(外部抵抗)」に設定されているときにのみ利用可能です。

void CapSense_SetAllSensorsDriveMode(uint8 mode)

説明: CapSense コンポーネントの静電容量センサで使用されているピン全てをドライブモードに設定します。

パラメータ: (uint8) mode: 使用するドライブ モード。ドライブ モードの詳細は、ピン コンポーネントのデータシートを参照してください。

戻り値: なし

副作用: なし

void CapSense_SetAllCmodsDriveMode(uint8 mode)

説明: CapSense コンポーネントの Cmod コンデンサで使用されているピン全てをドライブモードに設定します。

パラメータ: (uint8) mode: 使用するドライブ モード。ドライブ モードの詳細は、ピン コンポーネントのデータシートを参照してください。

戻り値: なし

副作用: なし

void CapSense_SetAllRbsDriveMode(uint8 mode)

説明: CapSense コンポーネントの ブリーダー抵抗で使用されているピン全てをドライブモードに設定します。[Current Source (電流源)] が「External Resistor(外部抵抗)」に設定されているときにのみ利用可能です。

パラメータ: (uint8) mode: 使用するドライブ モード。ドライブ モードの詳細は、ピン コンポーネントのデータシートを参照してください。

戻り値: なし

副作用: なし

データ構造

API 関数はセンサとウィジェットのデータを処理するためにいくつかのグローバルアレイを使用します。そのため、これらのアレイをマニュアルで変更してはなりません。これらの値はデバッグおよびチューニングプロセスで閲覧可能です。たとえば、チャート作成ツールを使用して、アレイの内容を表示することができます。グローバルアレイは:

- CapSense_SensorRaw [ ]
- CapSense_SensorEnableMask [ ]
- CapSense_portTable[ ] and CapSense_maskTable[ ]
- CapSense_SensorBaseline [ ]
- CapSense_SensorBaselineLow[ ]
- CapSense_SensorSignal [ ]
- CapSense_SensorOnMask[ ]

CapSense_SensorRaw [ ]

このアレイには各センサの生データがあります。アレイサイズはセンサの総数と同等です (CapSense_TOTAL_SENSOR_COUNT)。CapSense_SensorRaw[ ] データは以下の関数により更新されます:

- CapSense_ScanSensor()
- CapSense_ScanEnabledWidgets()
- CapSense_InitializeSensorBaseline()
- CapSense_InitializeAllBaselines()
- CapSense_UpdateEnabledBaselines()

CapSense_SensorEnableMask [ ]

これはセンサスキャニング状態を保持するバイトアレイです。CapSense_SensorEnableMask[0] にはセンサ 0 から 7 までの マスクされたビットがあります(センサ 0 はビット 0、センサ 1 はビット 1)。CapSense_SensorEnableMask[1] にはセンサ 8 から 15 (必要な場合)までのマスクされたビットがあります。このバイトアレイは、センサの総数を含むのに必要な数のエレメントを保持しています。ビット値は、センサが CapSense_ScanEnabledWidgets() 関数の呼び出しでスキャンされたかどうかを特定します: 1 – センサはスキャンされた、0 – センサはスキャンされていない。CapSense_SensorEnableMask[ ] データは以下の関数により更新されます:

- CapSense_EnabledWidget()
- CapSense_DisableWidget()

CapSense_SensorEnableMask[ ] データは以下の関数により使用されます:

- CapSense_ScanEnabledWidgets()

CapSense_portTable[ ] and CapSense_maskTable[ ]

これらのアレイには、全てのセンサのポートおよびピンマスクがあり、どのピンにセンサが接続しているかを特定します。

- ポート – ピンが属するポート番号を定義します。
- マスク – ポート内のピン番号を定義します。

CapSense_SensorBaselineLow[ ]

このアレイは、ベースライン更新の際のローパス フィルタに使われる、各センサのベースラインデータのフラクショナルバイトを保持しています。アレイサイズはセンサの総数と同等です。CapSense_SensorBaselineLow[ ] アレイは以下の関数により更新されます:

- CapSense_InitializeSensorBaseline()
- CapSense_InitializeAllBaselines()
- CapSense_UpdateSensorBaseline()
- CapSense_UpdateEnabledBaselines()

CapSense_SensorBaseline[ ]

このアレイは各センサのベースライン データを保持しています。アレイサイズはセンサの総数と同等です。CapSense_SensorBaseline[ ] アレイは以下の関数により更新されます:

- CapSense_InitializeSensorBaseline()
- CapSense_InitializeAllBaselines()
- CapSense_UpdateSensorBaseline()
- CapSense_UpdateEnabledBaselines()

CapSense_SensorSignal [ ]

このアレイは、各センサの現在の生カウントから以前のベースラインを減算して算出したセンサ信号カウントを保持しています。アレイサイズはセンサの総数と同等です。**Widget Resolution** はこのアレイの分解能を **1 バイト** または **2 バイト**として定義します。CapSense_SensorSignal[ ] アレイは以下の関数により更新されます:

- CapSense_InitializeSensorBaseline()
- CapSense_InitializeAllBaselines()
- CapSense_UpdateSensorBaseline()
- CapSense_UpdateEnabledBaselines()

CapSense_SensorOnMask[ ]

これはセンサのオン／オフ状態を保持しているバイトアレイです。

CapSense_SensorEnableMask[0] にはセンサ 0 から 7 までの マスクされたビットがあります(センサ 0 はビット 0、センサ 1 はビット 1)。CapSense_SensorEnableMask[1] にはセンサ 8 から 15 (必要な場合)までのマスクされたビットがあります。このバイトアレイは、センサの総数を含むのに必要な数のエレメントを保持しています。センサがオン(アクティブ)であればビット数は 1 で、センサがオフ(非アクティブ)であれば 0 です。
CapSense_SensorOnMask[ ] データは以下の関数により更新されます:

- CapSense_CheckIsWidgetActive()
- CapSense_CheckIsAnyWidgetActive()

定数

以下の定数が定義されています。条件付きで定義されていて、現在の設定に必要な場合に限って、表示される定数もあります。

- CapSense_TOTAL_SENSOR_COUNT – CapSense CSD コンポーネント内のセンサの総数を定義します。

2 チャネル デザインの場合、1 つのセンサに属するセンサの数は以下のように定義されます:

- CapSense_TOTAL_SENSOR_COUNT__CH0 – チャネル 0 に属するセンサの総数を定義します。
- CapSense_TOTAL_SENSOR_COUNT__CH1 – チャネル 1 に属するセンサの総数を定義します 。
- CapSense_CSD_TOTAL_SCANSLOT_COUNT – チャネル 0 またはチャネル 1 どちらかの最大センサカウントを定義します。

センサ定数

各センサに定数が用意されています。これらの定数は以下の関数ではパラメーターとして使用されます:

- CapSense_EnableSensor

- CapSense_DisableSensor

定数名は以下から構成されています:

インスタンス名 + "_SENSOR" + ウィジェット名 +エレメント + "#エレメント番号" + "__" + Widget 種類

例えば、

```
#define CapSense_SENSOR_TP1_ROW0__TP    0
#define CapSense_SENSOR_TP1_ROW1__TP    1
#define CapSense_SENSOR_TP1_COL0__TP    2
#define CapSense_SENSOR_TP1_COL0__TP    3
#define CapSense_SENSOR_LS0_E0__LS      5
#define CapSense_SENSOR_LS0_E1__LS      6
#define CapSense_SENSOR_PROX1__PROX     7
```

- ウィジェット名 – ユーザー設定のウィジェット名 (有効な C スタイルの識別子であること)。ウィジェット名は CapSense CSD コンポーネント内で固有である必要があります。ウィジェット名は全て大文字です。
- エレメント番号 – エレメント番号は、ラジアル スライダ のような複数のエレメントがあるウィジェットだけに存在します。タッチパッドおよびマトリックスボタンでは、エレメント番号は 'Col' または 'Row' という単語とその番号で構成されています。(例えば: Col0, Col1, Row0, Row1)。線形およびラジアル スライダ では、エレメント エレメント番号は 'e' という文字と、その番号で構成されています。(例えば: e0, e1, e2, e3)。
- ウィジェット種類 – いくつかのウィジェット種類があります:

エイリアス	説明
BTN	Buttons
LS	Linear Sliders
RS	ラジアル スライダ
TP	Touch Pads
MB	Matrix Buttons
PROX	Proximity Sensors
GEN	一般的センサ
GRD	Guard Sensor

ウィジェット定数

定数は各ウィジェットに対して提供されます。これらの定数は以下の関数ではパラメーターとして使用されます:

- CapSense_CheckIsWidgetActive()
- CapSense_EnableWidget() および CapSense_DisableWidget()
- CapSense_GetCentroidPos()
- CapSense_GetRadialCentroidPos()
- CapSense_TouchPos()

定数は以下で構成されます:

インスタンス名 + ウィジェット名 + ウィジェットタイプ

例えば:

```
#define CapSense_UP__BTN          0
#define CapSense_DOWN__BTN        1
#define CapSense_VOLUME__SL       2
#define CapSense_TOUCHPAD__TP     3
```

ファームウェア ソースコードの例

PSoC Creator は、[Find Example Project (プロジェクト例を検索)] ダイアログに数多くのプロジェクト例を提供しており、そこには回路図およびコード例が含まれています。コンポーネント固有の例を見るには、[Component Catalog (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 (プロジェクト例を検索)」を参照してください。

ピンの割り当て

CapSense カスタマイザにより、CapSense センサおよびサポート信号それぞれのピン アライアス名が生成されます。これらのアライアスは、センサおよび信号をデバイスの物理的ピンに割り当てるために、使用します。CapSense CSD コンポーネントのセンサおよび信号を、Design Wide Resources ファイル ビューの [Pin Editor (ピン エディタ)] タブのピンに割り当てます。

サイド

PSoC デバイス内のアナログ配線マトリックスは、左と右の 2 つに分割されます。偶数ポート番号のピンはデバイスの左側にあり、奇数ポート番号のピンは右側にあります。

シリアル センシング アプリケーションでは、センサ ピンをデバイスのどちら側にでも割り当てることができます。アプリケーションに使用されるセンサが少数の場合、すべてのセンサ信号をデバイスの片側に割り当てると、アナログ リソースの配線が効率的になり、アナログ リソースが他のコンポーネント用に解放されます。

パラレル センシング アプリケーションでは、CapSense コンポーネントによって、2 つの独立したハードウェア セットで 2 つのスキャンを同時に実行できます。2 つのパラレル回路のそれぞれに、別個の Cmod および Rb (該当する場合)、および独自のセンサ ピン セットがあります。1 組がデバイスの右側を、もう 1 組が左側を占有します。信号名アライアスに、信号がどちら側に関連するかが示されています。

センサ ピン – CapSense_cPort – ピンの割り当て

アライアスは、センサ名を CapSense カスタマイザのウィジェット タイプおよびウィジェット名と関連付けるために準備されています。

センサのアライアスは、以下のとおりです:

ウィジェット名 + エレメント番号 + "__" + ウィジェット タイプ

注 2 チャンネル デザインでは、1 つのチャンネルに属するウィジェットエレメントのみを、そのチャンネルの Cmod と同じ側のチップに接続できます。**ピン エディタでは、正しいピン割り当ての、デザイン ルール チェックによる検証は行われません**。ピン配置エラーは、ビルド プロセス中にフラグされます。

注 Opamp 出力 P0[0]、P0[1]、P3[6] および P3[7] の寄生静電容量は、他のピンより大きくなっています。これにより、CapSense アプリケーションでの P0[0]、P0[1]、P3[6] および P3[7] からの指の応答が低減し、可能であれば、これを回避する必要があります。これを使用する必要がある場合、静電容量差がスライダおよびタッチパッドの位置エラーの原因にならない個々のボタンに、使用する必要があります。

CapSense_cCmod_Port – ピン割り当て

外部変調コンデンサ (Cmod) の片側を物理的ピンに、他方を GND に接続する必要があります。2 チャンネル デザインでは 2 つの Cmod コンデンサが必要で、1 つはデバイスの左側用、もう 1 つは右側用です。Cmod は **any pin** に接続できますが、最も効率的なアナログ配線には、以下のピンで直接接続できます。

- 左側 : P2[0]、P2[4]、P6[0]、P6[4]、P15[4]
- 右側 : P1[0]、P1[4]、P5[0]、P5[4]

Cmod コンデンサのエイリアスは以下のとおりです:

エイリアス	説明
CmodCH0	Cmod、チャンネル 0 用

エイリアス	説明
CmodCH1	Cmod、チャンネル 1 用。2 チャンネル デザインでのみ使用可能。

変調コンデンサに推奨されている値は、4.7 ～ 47 nF です。最適な静電容量は、そのアプリケーションで最大の S/N 比を得るための実験を行うことで選択できます。ほとんどの場合、5.6～10 nF の値で良い結果が得られます。

セラミック コンデンサを使用する必要があります。コンデンサの温度係数は重要ではありません。

Current Source が [External Resistor (外部抵抗)] に設定されている場合は、最適 Cmod 値を決定するまえに、外部 Rb フィードバック抵抗値を選択する必要があります。

CapSense_cRb_Ports – ピン割り当て

Current Source が [External Resistor (外部抵抗)] に設定されている場合、外部ブリーダ抵抗が必要です。外部ブリーダ抵抗 (Rb) は、物理的ピンおよび変調コンデンサ (Cmod) の非接地接続部に接続する必要があります。

1 チャンネル当たり最大 3 つのブリーダ抵抗が、サポートされています。ブリーダ抵抗には、3 つのピン cRb0、cRb1 および cRb2 を割り当てることができます。

外部ブリーダ抵抗のエイリアスは、以下のとおりです:

エイリアス	説明
Rb0CH0、Rb1CH0、Rb2CH0	外部抵抗、チャンネル 0 用。
Rb0CH1、Rb1CH1、Rb2CH1	外部抵抗、チャンネル 1 用。2 チャンネル デザインでのみ使用可能。

抵抗値は、総センサ静電容量によって異なります。抵抗値は、次の条件で選択します:

- 異なるセンサの接触の生カウントをモニターする。
- 選択したスキャン分解能で、フルスケール読み値より約 30% 小さい最大読み値を提供する抵抗値を選択します。抵抗値が増加すると、生のカウント値が増加します。

一般的なブリーダ抵抗値は 500 Ω ～ 10 kΩ で、センサ静電容量に依存します。

割り込みサービス ルーチン

CapSense コンポーネントでは割り込みが使用され、これによって各センサ スキャン後にトリガされます。スタブ ルーチンが提供され、そこでは、必要に応じて独自のコードを追加できます。スタブ ルーチンは、初めてプロジェクトが構築されたときに、*CapSense_INT.c* ファイルで生成されます。割り込みの回数は、チャンネルの数に応じた、

チャンネル当たり 1 つの、CapSense モードの選択に依存します。使用するコードは、ビルド間に保存されるように、提供されているコメント タグの間に追加する必要があります。

2 つのチャンネル モード ISR の優先順位設定

CapSense CSD コンポーネントの ISR ルーチンは、再入可能ではありません。これにより、2 チャンネル デザインに対して設定されている ISR の優先順位に制約が発生します。チャンネル ISR ルーチンが再入可能になることを防止するため、2 つのチャンネルの ISR 優先順位は同じである必要があります。

CapSense_CSD_IsrCH1	Default <7>	☐	15
CapSense_CSD_IsrCH0	Default <7>	☐	19

機能説明

定義

センサ

1 つのピンを経由して PSoC に接続された 1 つの CapSense 素子。センサは、基板上の導電性の素子です。センサの例には、以下があります：FR4 上のカッパー(銅)、Flex 上のカッパー、PET 上のシルバーインク、ガラス上の ITO。

スキャン時間

スキャン時間は、CapSense モジュールが 1 つ以上の静電容量式センサをスキャンしている期間です。複数のセンサを所定のスキャン センサに組み合わせて、近接センシングなどのモードを可能にすることができます。

CapSense ウィジェット

CapSense ウィジェットは、さらに高いレベルの機能を提供するために、1 つ以上のスキャン センサから構築されます。CapSense ウィジェットの例には、ボタン、スライダ、ラジアル スライダ、タッチ パッド、マトリックス ボタン、近接検知センサがあります。

FingerThreshold

この値は、センサに触れているかどうかを判断するために使用します。

NoiseThreshold

静電容量スキャンのノイズのレベルが決定されます。センサの基準値での電圧および温度の変動を追跡するため、ベースライン アルゴリズムによってノイズをフィルタリングします。

Debounce (デバウンス)

デバウンスカウンタをセンサのアクティブ遷移に追加します。センサが非アクティブからアクティブに遷移するには、高振幅および高頻度のノイズを除去するために指定されたサンプル数に対して、差のカウント値が指閾値＋ヒステリシスを上回る状態を続けなければなりません。

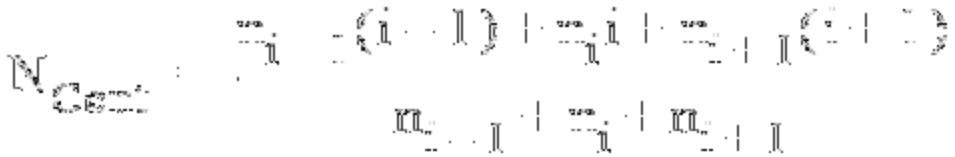
Hysteresis

指閾値とともに使用されるヒステリシス値を設定します。ヒステリシスが必要な場合、センサは、カウント値が指閾値 + ヒステリシス値を超えるまで、「オン」または「アクティブ」とは見なされません。センサは、測定されたカウント値が指閾値 - ヒステリシス値を下回るまで、「オフ」または「非アクティブ」とみなされません。

API 分解能 – 補間およびスケーリング

スライダ センサおよびタッチパッドでは、多くの場合個々のセンサのネイティブ ピッチよりも高い分解能を得られるように指 (またはその他の静電容量性物体) の位置を特定する必要があります。スライド式センサやタッチパッドで指が触れるエリアは、しばしば 1 個のセンサより大きくなっています。

セントロイド計算を使用して補間位置を計算するため、先ずアレイをスキャンして、所定のセンサ位置が有効であることを、確認します。特定の数の近隣センサ信号がノイズ閾値を超えていることが必要です。最も強い信号が見つかったら、その信号と、ノイズ閾値より大きい近隣信号を使用して、セントロイドを算出します。セントロイドの計算には、最少 2 個、最多で 8 個のセンサが使用されます。

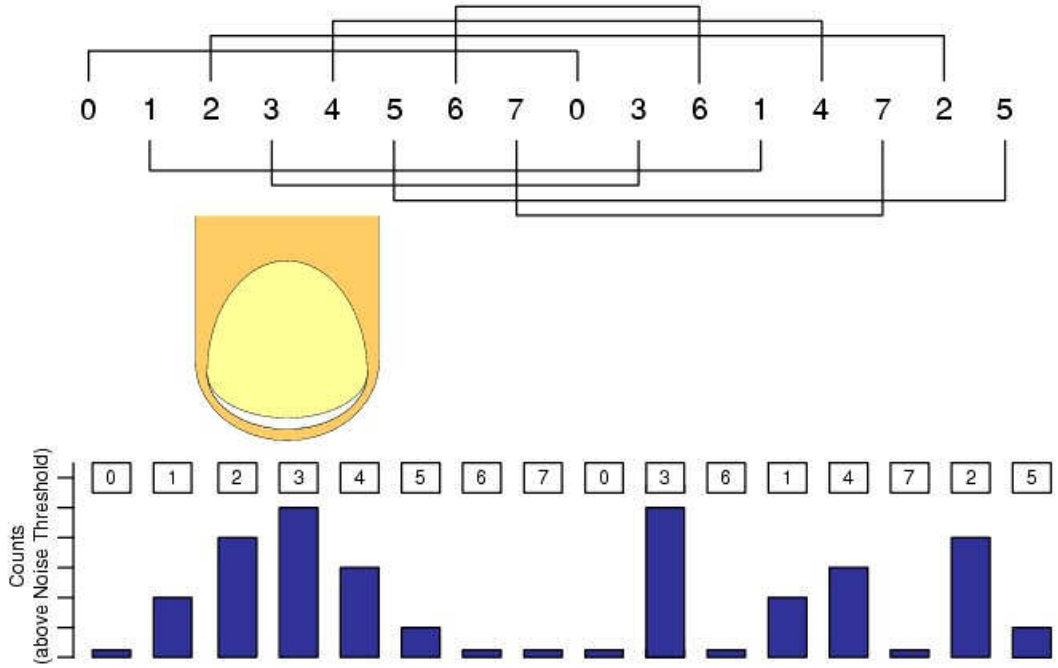


通常、計算結果は分数です。たとえば 12 個のセンサに対して範囲が 0 ～ 100 の場合、セントロイドを特定の分解能でレポートするには、セントロイド値にスカラ量を掛けます。1 つの計算で補間とスケーリングのオペレーションを組み合わせ、その結果を直接、希望のスケールでレポートする方が効率的です。これは高レベル API で扱われます。スライダ センサのカウントおよび分解能は、CapSense CSD カスタマイザで設定されます。

ダイプレックス

ダイプレックス スライダでは、スライダの個々の PSoC センサ接続部は、スライダ センサのアレイの 2 つの物理的位置にマップされます。物理的位置の最初の (数字が小さい) 半分は、CapSense カスタマイザを使用して設計者が割り当てたポート ピンを使用して、ベース割り当てセンサに連続的にマップされます。物理的センサ位置の後の (数字が大きい) 半分は、カスタマイザのアルゴリズムによって自動的にマップされて、include ファイルにリストされます。この順序は、半分内における近隣センサ起動が別の半分の近隣センサ起動を引き起こさないように設定されます。この順序の決定と、PCB 基板へのマッピングは慎重に行ってください。

図 1。ダイプレックス



スライダのセンサ静電容量は、均衡がとれている必要があります。センサや PCB レイアウトによって、一部のセンサペアではセンサー配線が長くなる可能性があります。ダイプレックス センサ インデックスは、ダイプレックスを選択すると CapSense カスタマイザによって自動的に作成され、参照用に以下に含まれます。

異なるスライダセグメントカウントのダイプレックスシーケンス

合計スライダセグメント Count (カウント)	セグメントシーケンス
10	0,1,2,3,4,0,3,1,4,2

合計スライダセグメント Count (カウント)	セグメントシーケンス
12	0,1,2,3,4,5,0,3,1,4,2,5
14	0,1,2,3,4,5,6,0,3,6,1,4,2,5
16	0,1,2,3,4,5,6,7,0,3,6,1,4,7,2,5
18	0,1,2,3,4,5,6,7,8,0,3,6,1,4,7,2,5,8
20	0,1,2,3,4,5,6,7,8,9,0,3,6,9,1,4,7,2,5,8
22	0,1,2,3,4,5,6,7,8,9,10,0,3,6,9,1,4,7,10,2,5,8
24	0,1,2,3,4,5,6,7,8,9,10,11,0,3,6,9,1,4,7,10,2,5,8,11
26	0,1,2,3,4,5,6,7,8,9,10,11,12,0,3,6,9,12,1,4,7,10,2,5,8,11
28	0,1,2,3,4,5,6,7,8,9,10,11,12,13,0,3,6,9,12,1,4,7,10,13,2,5,8,11
30	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,0,3,6,9,12,1,4,7,10,13,2,5,8,11,14
32	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0,3,6,9,12,15,1,4,7,10,13,2,5,8,11,14
34	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,0,3,6,9,12,15,1,4,7,10,13,16,2,5,8,11,14
36	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,0,3,6,9,12,15,1,4,7,10,13,16,2,5,8,11,14,17
38	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,0,3,6,9,12,15,18,1,4,7,10,13,16,2,5,8,11,14,17
40	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,0,3,6,9,12,15,18,1,4,7,10,13,16,19,2,5,8,11,14,17
42	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,0,3,6,9,12,15,18,1,4,7,10,13,16,19,2,5,8,11, 14,17,20
44	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,0,3,6,9,12,15,18,21,1,4,7,10,13,16,19,2,5, 8,11,14,17,20
46	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,0,3,6,9,12,15,18,21,1,4,7,10,13,16,19, 22,2,5,8,11,14,17,20
48	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,0,3,6,9,12,15,18,21,1,4,7,10,13,16, 19,22,2,5,8,11,14,17,20,23

合計スライダセグメント Count (カウント)	セグメントシーケンス
50	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,0,3,6,9,12,15,18,21,24,1,4,7,10,13,16,19,22,2,5,8,11,14,17,20,23
52	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,0,3,6,9,12,15,18,21,24,1,4,7,10,13,16,19,22,25,2,5,8,11,14,17,20,23
54	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,0,3,6,9,12,15,18,21,24,1,4,7,10,13,16,19,22,25,2,5,8,11,14,17,20,23,26
56	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,0,3,6,9,12,15,18,21,24,27,1,4,7,10,13,16,19,22,25,2,5,8,11,14,17,20,23,26

フィルタ

CapSense コンポーネントでは、複数のフィルタ (メジアン、アベレージ、一次 IIR 、およびジッタ) が、提供されます。このフィルタは、生センサ データによってセンサ ノイズを低減することにも、スライダおよびタッチパッドの位置データによって位置ノイズを低減することにも、使用できます。

Median Filter

メジアン フィルタは最新の 3 つのサンプルを調べて、メジアン値をレポートします。メジアンは、3 つのサンプルを並べ替えて、その中央値を取ることによって得られます。このフィルタを使用することにより、短いノイズ スパイクが除去され、サンプル 1 つの遅延が発生します。このフィルタは、遅延の発生および RAM 使用量により、一般的にはお勧めしません。このフィルタをイネーブルにすると、各センサ (raw) およびウィジェット (位置) に対して RAM の 4 バイトが消費されます。これは、デフォルトではディスエーブルになっています。

Averaging Filter

アベレージ フィルタでは、位置の最新 3 サンプルを調べて、単純平均値がレポートされます。これを使用することによって短いノイズ スパイクが除去され、サンプル 1 つの遅延が発生します。このフィルタは、遅延の発生および RAM 使用量により、一般的にはお勧めしません。このフィルタをイネーブルにすると、各センサ (raw) およびウィジェット (位置) に対して RAM の 4 バイトが消費されます。これは、デフォルトではディスエーブルになっています。

一次 IIR フィルタ

一次 IIR フィルタを、生フィルタおよびセンサ フィルタの両方に推奨します。その理由は、SRAM の必要量が最少で、応答が速いからです。IIR フィルタによって最新のセンサ データまたは位置データが拡大または縮小され、

そのデータが前のフィルタ出力のスケール調整版に追加されます。このフィルタをイネーブルにすると、各センサ (raw) およびウィジェット (位置) に対して RAM の 2 バイトが消費されます。デフォルトでは、生フィルタおよび位置フィルタの両方に対して IIR1/4 がイネーブルです。

1 次 IIR フィルタ :

$$IIR1/2 = 1/2\ previous + 1/2\ current$$

$$IIR1/4 = 3/4\ previous + 1/4\ current$$

$$IIR1/8 = 7/8\ previous + 1/8\ current$$

$$IIR1/16 = 15/16\ previous + 1/16\ current$$

Jitter Filter

このフィルタにより、生センサ データまたは位置データのノイズが除去されます。この 2 種類の値は切り替わります (ジッタ)。最新のセンサ値が前回のセンサ値より大きい場合、前のフィルタ値を 1 だけ増やし、前回より小さい場合は 1 だけ減らします。ピーク-ピーク 4 LSB 以下のノイズが含まれているデータに適用する場合、および一部の位置センサに有用な低速応答が許容される場合に、もっとも効率的です。このフィルタをイネーブルにすると、各センサ (raw) およびウィジェット (位置) に対して RAM 2 バイトが消費されます。これは、デフォルトではディスエーブルになっています。

水の CapSense システムへの影響

CapSense への水滴および指による影響は、同じです。ただし、水滴は指による影響とは異なり、検知領域面全体に影響します。

CapSense 表面への水滴の影響には、いくつかの種類があります :

- デバイス表面での水の薄い縞または流れの形成。
- 分散した水滴。
- デバイスを洗浄するときまたは水に浸けるときの、デバイス表面全体または大部分を覆う水の流れ。

水に含まれている塩分または金属により、導電性になります。さらに、濃度が高いほど、水の導電性が高くなります。石鹸を含んだ水、海水、およびミネラル ウォーターは、液体として CapSense に悪影響を与えます。これらの液体はデバイス表面への指による接触と類似の効果があり、これによってデバイスが誤動作することがあります。

防水および検出

- この機能により、CapSense システムへの水の影響を抑制するように、CapSense CSD コンポーネントが設定されます。この機能は以下のパラメーターを設定します :
- シールド電極を使用して、センサへの水滴の影響をハードウェア レベルで補償するようにすることができます。

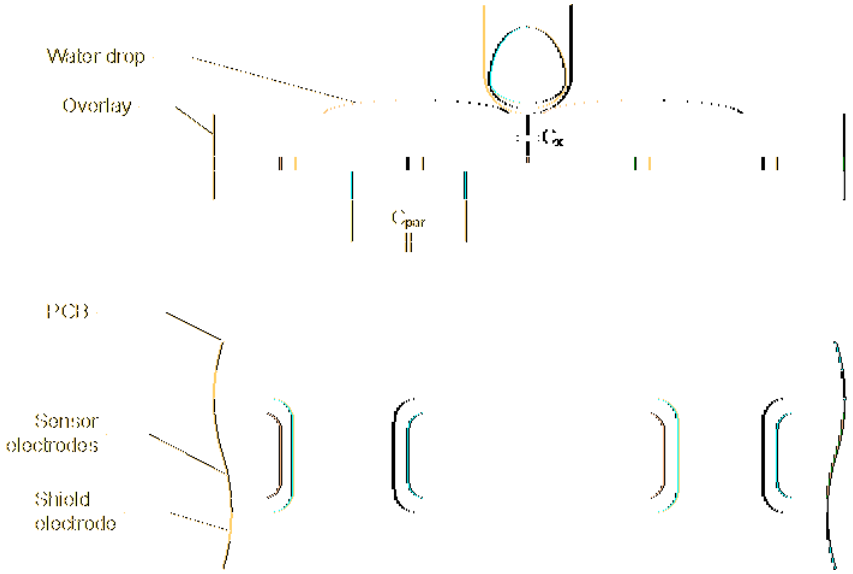
- ガード センサが追加されます。ガード センサはすべてのセンサを囲み、実際の検知ウィジェットがカバーされる場合は、ガード センサの配置によって水からのカバーが保証されるようにする必要があります。ウィジェットのステータスの CapSense 出力は、ガード センサがトリガしたとき、プログラムによってブロックされる必要があります。

シールド電極

アプリケーションによっては、水膜や水滴の存在下で、信頼性のあるオペレーションが必要です。白物家電、車載アプリケーション、種々の産業用アプリケーション、その他では、水、氷、凝縮をもたらす湿度変化によって間違ってトリガすることがない、静電容量式センサが必要です。この場合、別個のシールド電極を使用することができます。この電極は、検知電極の背部または周囲にあります。水膜がデバイスのオーバーレイ表面にある場合、シールドと検知電極のカップリングが増加します。シールド電極により、寄生静電容量の影響を軽減することができ、これにより、検知静電容量の変化を処理するダイナミック レンジが広がります。

アプリケーションによっては、シールド電極の信号、およびシールド電極の検知電極に対する相対的配置を選択することが、有用です。これらの電極間のカップリングが水分によって増加して、検知電極の静電容量測定にネガティブな接触の変化をもたらします。これによって、水分による接触の誤認が抑制されることにより、高レベルのソフトウェア API 動作が単純化されます。CapSense CSD コンポーネントにより、シールド電極への個別の出力がサポートされ、PCB 配線が単純化されます。

図 2。可能なシールド電極 PCB レイアウト



前図に、ボタンのシールド電極の可能なレイアウト構成の一例が、示されています。シールド電極は、透明 ITO タッチパッド デバイスに、特に有用です。この場合、LCD ドライブ電極のノイズが阻止され、これと同時に浮遊静電容量が低減されます。

この例では、ボタンはシールド電極平面で覆われています。代替案として、ボタンの下のプレーンなど、PCB の反対側のレイヤに置くことも可能です。この場合、充填率約 30～40% で、ハッチパターンを使用することが推奨されます。ここでは、グランド プレーンを追加する必要はありません。

水滴がシールド電極と検知電極の間にある場合、寄生静電容量 (Cpar) が増加し、変調器電流が減少することがあります。

シールド電極は、任意のピンに接続できます。駆動モードを Strong Slow に設定し、グランド ノイズと放射性エミッションを軽減します。また、スリュー制限抵抗も、PSoC デバイスとシールド電極の間に接続できます。

シールド電極の使用と制約

CapSense CSD コンポーネントにより、シールド電極の使用に対して、以下のモードが提供されます。

電流モード IDAC ソース

このモードでは、センサが GND と Vref = 1.024 V の間で交互に切り替わるため、制約があります。シールド電極信号が、GND と Vddio (通常は電源電圧と同じ) の間で交互に切り替わります。その差は著しく、シールド信号でセンサからの信号が完全に相殺されることがあります。可能な解決策は以下のとおりです:

- 高 Vref を使用して、差を最小値まで解消します。リファレンスとして VDAC を、この目的に使用できます。
- SIO ピンをシールドとして使用して、Vref に等しい出力を供給します。CapSense CSD 出力の Vref 端子を、Vref を SIO ピンに配線するために使用できます。これが推奨法です。このモードでは、シールドへのセンサ接続を使用しないでください。これを使用すると、出力が Vddio に等しくなります。Vref = 1.024 V 設定には配線上の制限があり、ピンに配線することができません。

電流モード IDAC シンクおよび外部抵抗

これらのモードには、シールドおよび非アクティブ センサ モードの使用に制約はありません。その理由は、センサが Vddio と Vref = 1.024 V の間を交互に切り替わるからです。シールド電極信号は、GND と Vddio (通常は電源電圧に同じ) の間で交互に切り替わります。この場合、この差は、問題を起こすほど大きくはありません。

ガード センサの搭載

ガード センサは防水アプリケーションに一般的に使用され、表面上の水を検知します。

[Advanced (詳細設定)] タブ オプションは、ガード センサを追加するためのものです。このセンサには特別なレイアウトが必要であり、通常は感知領域表面の周囲に配置されます。ガード センサの表面に水があると、ウィジェットがアクティブになります。ウィジェット アクティブ検知ファームウェア CapSense_1_IsWidgetActive() は、ガード センサの状態を定義するために、使用できます。

CapSense ウィジェットでの検知により、ガード センサによってトリガされたとき、一定の期間、プログラムによりユーザー コードで阻止する必要があります。ガード センサによってトリガされると、水があり、他のセンサの検知は信頼できません。

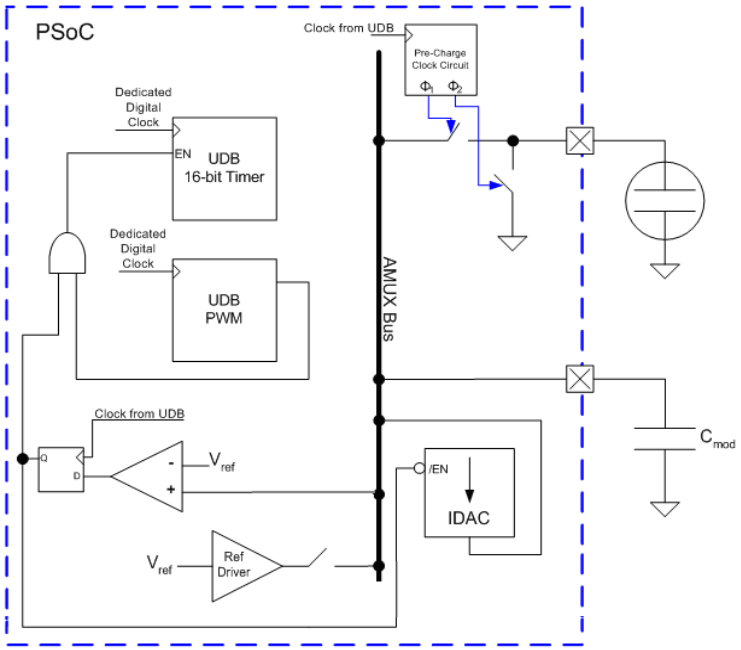
ガード センサのサイズを考えれば分かるように、その信号は、他のセンサの信号とは違います。このことは、その表面には、標準センサの表面上よりも多量の水が存在する可能性があることを、意味します。このため、水滴が存在することによって受信した信号は、指の接触によって発生した信号より、はるかに強力です。これにより、ガード センサへの指の接触が影響しないように、トリガ閾値およびフィルタを設定することができます。ガード センサは、特別なオプションなしでスキャンします。シールド電極は、ガード センサのスキャン中、ディスエーブルにはなりません。2 チャンネル デザインのガード センサは、常に最後に自動的にスキャンします。

ブロック ダイアグラムとコンフィギュレーション

シグマ デルタ(CSD) 変調器を使用した静電容量検知により、スイッチト キャパシタ アナログ技法およびデジタル デルタ-シグマ変調器を使用した静電容量検知が提供され、センスされたスイッチト キャパシタ電流がデジタル コードに変換されます。これにより、ボタン、スライダ、近接検知器、タッチ パッド、および導電性センサのアレイを使用したタッチスクリーンを、実装できます。高レベルのソフトウェア ルーチンにより、ダイプレックスを使用したスライダ分解能の増強、および物理的および使用環境的センサの種別に対する補正が、可能になります。基本的 CSD メソッドで可能なアナログ ハードウェアは 3 種類あり、詳細は以下のとおりです。

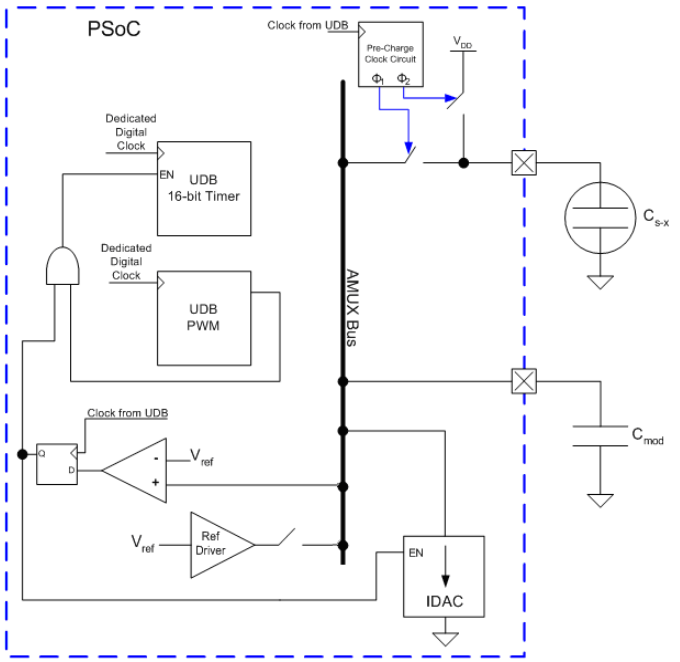
IDAC ソース

センサの切り替えは、GND と、変調コンデンサに接続する AMUX バスとを、交互に切り替えるように設定されます。この構成では、IDAC はセンサへのソース電流に合わせて設定されます。



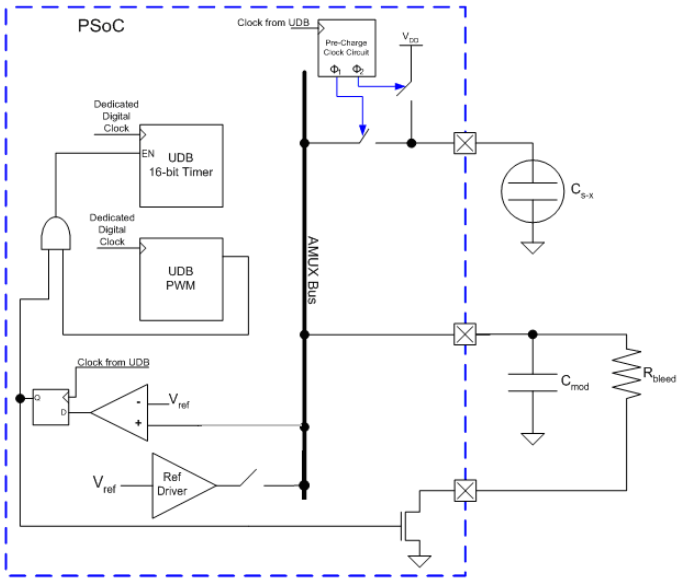
IDAC シンク

センサの切り替えは、Vdd および変調コンデンサに接続する AMUX バスとの間で交互に切り替えるように設定されます。jこの構成では、IDAC は、センサからのシンク電流に合わせて設定されます。



IDAC がディスエーブル、外部 R_b を使用

外部ブリーダ抵抗 Rb は、IDAC がグランドへの抵抗 R_b によって置き換えられている以外、IDAC シンク構成と同様に機能します。ブリーダ抵抗は、物理的に C_{mod} と GPIO の間に接続されます。GPIO は、"オープンドレイン ドライブ低" ドライブ モードで設定されます。このモードでは、C_{mod} を、R_b を通して放電することができます。



DC 電気的特性と AC 電気的特性

5.0V/3.3V DC 電気的特性と AC 電気的特性

電源電圧

パラメータ	テスト条件とコメント	Min	Typ	Max	単位
値	--	2.7	5.0	5.5	V

ノイズ

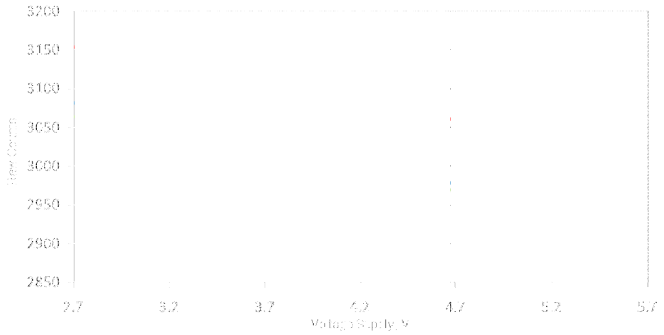
パラメータ	テスト条件とコメント	Min	Typ	Max	単位
ノイズ カウント、ピーク-ピーク (ノイズ カウント / ベースライン値カウント)	分解能 = 16 (ノイズ カウント / ベースライン値カウント)	--	0.2	--	%
	分解能 = 14	--	0.3	--	%
	分解能 = 10	--	0.4	--	%

消費電力

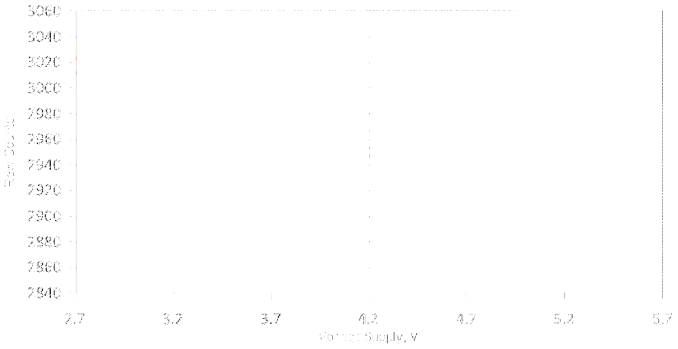
パラメータ	テスト条件とコメント	Min	Typ	Max	単位
アクティブ電流	Vdd = 3.3 V、CPU クロック = 24 MHz、CapSense スキャン クロック = 24 MHz、スキャン中の平均電流、センサ 8 個	--	9.42	--	mA
スタンバイ電流	Vdd = 3.3 V、CPU クロック = 24 MHz、CapSense スキャン クロック = 24 MHz、スキャン速度 = 超高速、分解能 = 9 100 ms レポート速度、センサ 8 個	--	92	--	uA
	Vdd = 3.3 V、CPU クロック = 24 MHz、CapSense スキャン クロック = 24 MHz、スキャン速度 = 高速、分解能 = 12 100 ms レポート速度、センサ 8 個	--	574	--	uA
スリープ/ウェイク電流	Vdd=3.3 V、CPU クロック = 24 MHz、CapSense スキャン クロック = 24 MHz、1 秒レポート速度、スキャン速度 = 高速、分解能 = 12、センサ 1 個	--	8	--	uA

図

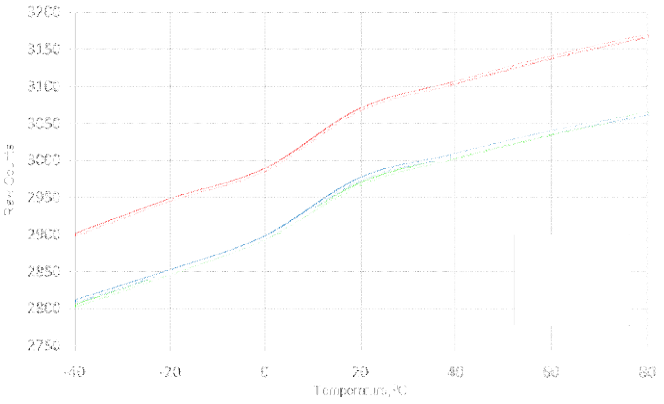
種々のスキャン速度での Raw カウント対電源電圧、PRS 16 フルスピード



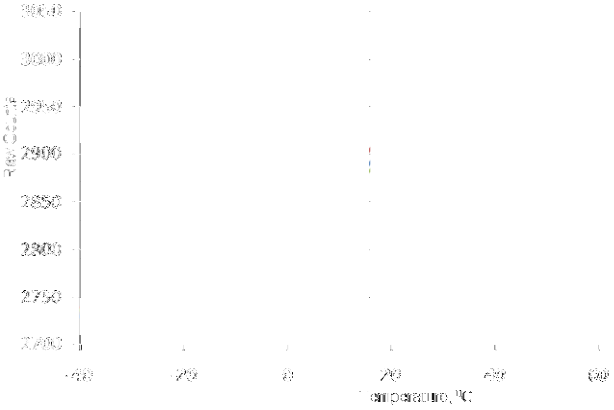
種々のスキャン速度での Raw カウント対電源電圧、PRS 8



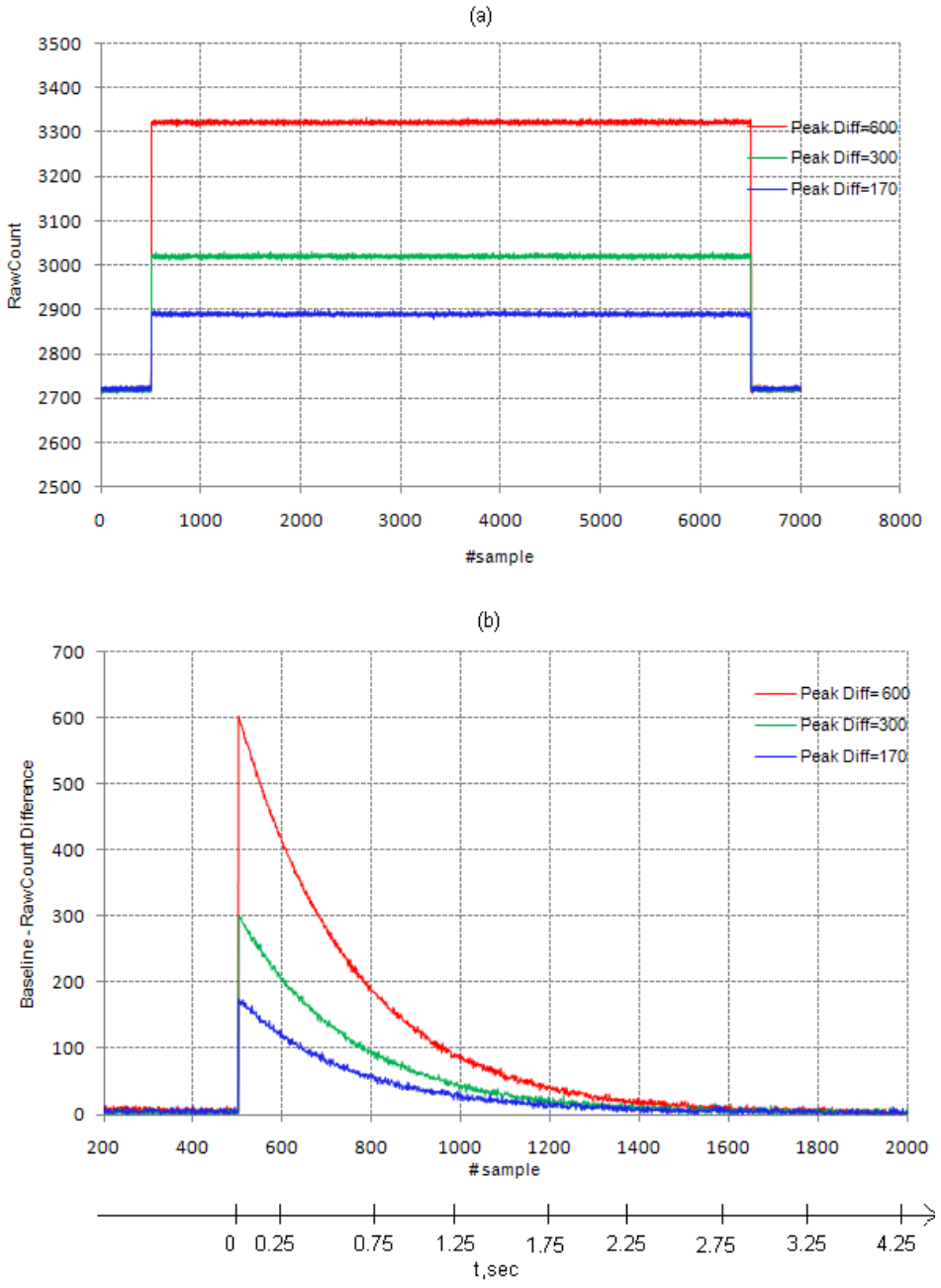
種々のスキャン速度での Raw カウント対温度、PRS 16 フルスピード



種々の速度での Raw カウント対温度、PRS 8



種々の 生カウント ステップ変更値での、基準値の時間的変動
(a) 種々のステップ値に対する、Raw カウント ステップ変更
(b) Raw カウント間の差。



コンポーネントの変更

バージョン	変更の説明	変更の理由 / 影響
2.10	Vdac を参照として使用する場合、CapSense バッファは削除されます。この操作は、1.024 V が基準として使用されているデザインに、影響します。	Cmod での過渡プロセスは、Vdac が参照として使用される場合に予測できます。
	チューナー GUI 通信とスキャンは、分離されます。	EZI2C 通信割り込みは、スキャン プロセスに影響しません。
	ユーザー セクションが CapSenseCSD_PreScan() 機能に追加されます。	割り込みがスキャン結果に影響する場合、スキャン開始時に割り込みをイネーブルおよびディスエーブルすることが、可能になります。
	カスタマ インターフェースの CapSense_InitializeEnabledBaselines() に追加します。	イネーブルなウィジェットに対してのみ、ベースライン値の初期化が可能になります。
	ベクトル ベースの画像をカスタマイザで追加します。	カスタマイザの画像品質が改善されます
	[Redesign Advanced (再設計詳細設定)] タブ	コントロールのレイアウトの改善
	AutoSense モードのパケット サイズを最適化します。	パケット サイズの低減と通信速度の増加
	カスタマイザとチューナーの間のインポートおよびエクスポートのメカニズムの再設計	使いやすくする
	チューナーの GUI の更新 : 追加メイン メニュー、色スキームの更新。	使いやすくする
	セントロイド ウィジェットのセンサ詳細ビューで、ヒステリシス関連の線を削除する	セントロイド ウィジェットのセンサ詳細ビューを明確にする
	デザインが遅い場合に、SNR(S/N比)計算速度を改善します。	
	チューナーの AutoSense 手順時の、ヒステリシス パラメータ変更の可視化を、追加します。	カスタマが現在のヒステリシス値をレビューできるようになります
	チューナーの "ログ期間" 機能を固定します。	

バージョン	変更の説明	変更の理由 / 影響
	センサの ScanResolution、IdacValue およびプリスケーラを、AutoSense モードの最初の読み取り時にのみ読み取るメカニズムを実装します。これは、これらのデータは最初のトランザクション後に変化しないからです。	チューナー GUI に転送されるデータ サイズを下げる
	データシートに特性データを追加	
	データシート再書き込み合計数	

CapSense CSD コンポーネント バージョン 2.10 は、CapSense CSD コンポーネント バージョン 2.00 の改良版です。