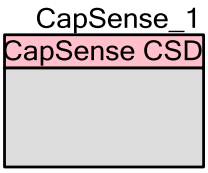
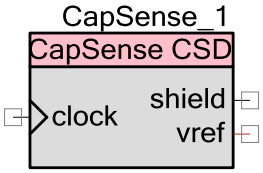
容量センシング(CapSense®CSD)

3.10

特長



- ユーザ定義によるボタン、スライダ、タッチパッドおよび近接静電容量式センサの組み合わせをサポート。
- 内蔵 PCGUI による自動 SmartSense™チューニングまたはマニュアルチューニング。
- AC 電力線ノイズ、EMC ノイズ、電源電圧変化に対するイミュニティー。
- オプションの 2 つのスキャンチャンネル(並列同期)により、センサのスキャンレートを向上。
- 水膜や水滴がある場合にも信頼できるオペレーションのシールド電極サポート。
- CapSense カスタマイザを使用し、センサや端子の割り当てをガイド。



概要

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良い方法です。

このデータシートをお読みいただいた後で以下の文書をお読みください。これらの文書は、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 の導入*
- *耐水静電容量検知* – AN2398

CapSense コンポーネントを使用する時は

従来のボタン、スイッチ、その他の制御の代わりに、静電容量式検知システムを多くのアプリケーションで使用できます。雨や水にさらされるアプリケーションでも使用できます。使用できるアプリケーション例として、自動車、屋外設備、ATM、公共アクセスシステム、携帯電話や PDA などのポータブルデバイス、および台所や浴室でのアプリケーションが挙げられます。

入出力の接続

このセクションでは、CapSense CSD コンポーネントの様々な入出力の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 コンポーネントのためにクロックを提供します。クロック入力は、**Enable clock input** パラメータが選択されている場合にのみ表示されます。

shield – 出力*

シールド電極信号がこの出力に接続されています。これは、シールド電極がイネーブルの場合にのみ使用できます。シールドの使用についての詳細は、「コンポーネントパラメータ 」セクションに記載されています。

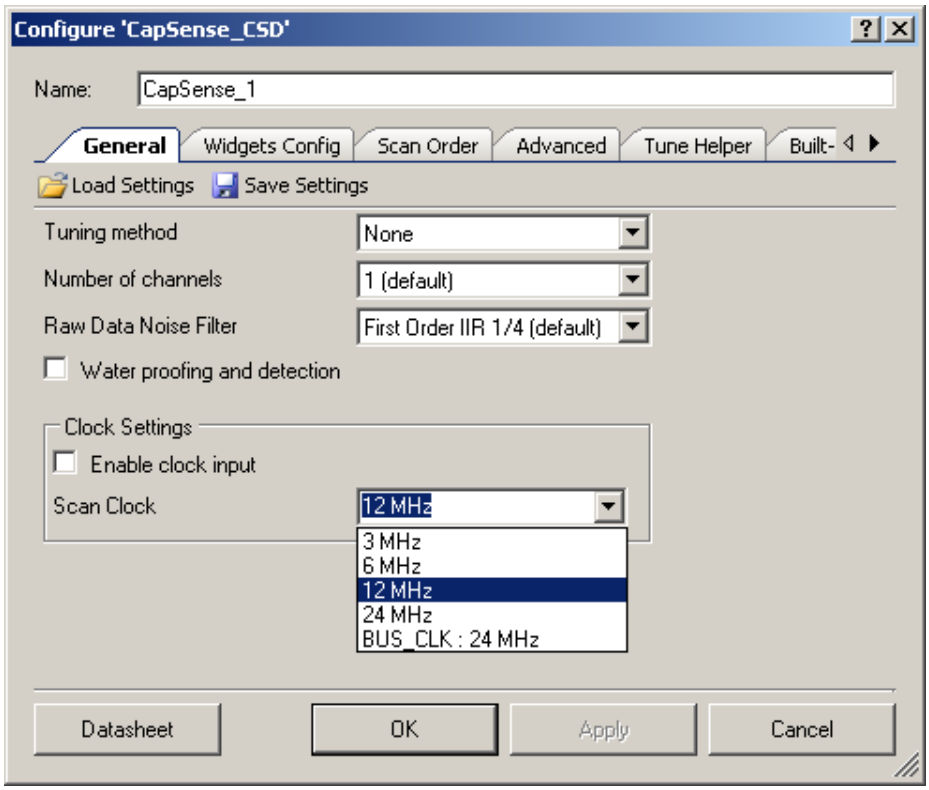
vref – 出力*

アナログリファレンス電圧がこの出力に接続されています。これは、シールド信号の振幅を調整するのに使用できます。IDAC**Sourcing モード**で **Shield** オプションがイネーブルの場合にのみ使用できます。SIO がシールド信号として使用される場合、Vref 出力は SIO リファレンスに接続されている必要があります。Vref の使用についての詳細は、このデータシートの「機能説明」セクションに記載されています。

コンポーネントパラメータ

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、CapSense 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 タブ



Load Settings/Save Settings

Save Settings を使用して、コンポーネントで設定されたチューニングおよび設定データをすべて保存します。これにより、新規プロジェクトで迅速に複製が可能になります。**Load Settings** が以前保存された設定のロードに使用されます。

また、保存された設定を使用して、設定およびチューニングのデータをチューナーGUI にインポートすることができます。

Tuning method

このパラメータでチューニング手法を指定します。以下の 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ます。

  これは、すべての設計で推奨されるチューニング手法です。ファームウェアアルゴリズムは、ランタイムに継続的に最善のチューニングパラメータを決定します。このモードでは追加の RAM リソースおよび CPU リソースが必要になります。

  重要 – プロジェクト回路図には SmartSense モードの CapSense_CSD コンポーネントは 1 つだけ配置できます。SmartSense チューニングは EZI2C 通信コンポーネントと併用することができます。これは、データをターゲットデバイスからチューナーGUI に送信するために **Tuner Helper** タブで指定します。

- **Manual** – チューナーGUI を使用して CapSense CSD コンポーネントをマニュアルチューニングできます。

  GUI を起動するには、シンボルを右クリックして、**Launch Tuner** を選択します。マニュアルチューニングについての詳細情報は、このデータシートの『チューナー GUI ユーザ 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には EZI2C 通信コンポーネントが必要になります。これは、ターゲットデバイスとチューナーGUI の間でデータを送信するために **Tuner Helper** タブで指定します。

- **None**(初期設定)–チューニングをディスエーブルにします。

  すべてのチューニングパラメータはフラッシュに保存されます。CapSense コンポーネントのすべてのパラメータがチューニングされて完了したら、このオプションを使用してください。このオプションを使用する場合、チューナーは読み取り専用モードで動作します。

Number of channels

このパラメータで、実装されるハードウェアスキャンチャンネルの数を指定します。

- **1**(初期設定) – 1～20 センサに最適です。コンポーネントは一度に 1 つの静電容量スキャンを行うことができます。一度に 1 つずつのセンサが順番にスキャンされます。ハードウェアには単一のチャンネルのみが実装されているため、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しません。
  - ❑ AMUX バスは結合しています。

    注すべての静電容量式センサがチップの片側に配置されている場合、左側(偶数ポート GPIO 番号、例：P0[X],P2[X],P4[X])または右側(偶数ポート GPIO 番号、例：P1[X],P3[X],P5[X])AMUX バスは相互に結合していません。AMUX バスの半分が使用されています。

    注ポートピン P15[0-5]は異なる AMUX バス(左側および右側)に接続があります。P12[X]および P15[6-7]は AMUX バスに接続がありません。選択した部分については TRM を参照してください。
  - ❑ コンポーネントは 1～(#GPIO – 1)静電容量式センサをスキャンできます。
  - ❑ C_{MOD} 外部コンデンサ 1 つが必要です。
- **2** – 20 を超えるセンサに最適です。コンポーネントは同時に 2 つの静電容量スキャンを行うことができます。AMUX バスの両方(左側および右側)が各チャンネルごとに使用されます。左右のセンサは一度に連続して 2 つ(右側センサ 1 つ、左側センサ 1 つ)がスキャンされます。1 つのチャンネルに他のチャンネルよりも多くのセ

ンサがある場合、センサが多く搭載されているチャンネルがアレイ内の残りのセンサのスキャンを 1 つずつ完了させます。一方、他のチャンネルはスキャンを行ないません。2 つのチャンネルにより使用されるリソースは 1 つのチャンネルと比較して倍増しますが、センサのスキャンレートも倍増します。

- ❑ 左の AMUX バスは 1～(偶数番号ポートの GPIO – 1)静電容量式センサをスキャンできます。
- ❑ 右の AMUX バスは 1～(奇数番号ポートの GPIO – 1)静電容量式センサをスキャンできます。
- ❑ C_{MOD} 外部コンデンサ 2 つ(各チャンネルに 1 つずつ)が必要です。
- ❑ 並行スキャンは同様のスキャンレートで実行します。

Raw Data Noise Filter

このパラメータで Raw データフィルタを選択します。1 つのフィルタのみを選択でき、すべてのセンサに適用されます。センサのスキャン中に生じるノイズの影響を低減するために、フィルタを使用してください。フィルタの種類についての詳細は、この文書の「機能説明」セクションの「フィルタ」を参照してください。

- **None** –フィルタは提供されません。フィルタファームウェアまたは SRAM 変数オーバーヘッドは発生しません。
- **Median** – 最後の 3 つのセンサ値を順番に並べ替え、中央値を返します。
- **Averaging** – 最後の 3 つのセンサ値の単純平均を返します。
- **First Order IIR 1/2** – 最新のセンサ値の半分に前のフィルタ値の半分を足した値を返します。IIR フィルタは最も低いファームウェアおよびすべてのフィルタの種類の SRAM オーバーヘッドが必要です。
- **First Order IIR 1/4**(初期設定) – 最新のセンサ値の 1/4 に前のフィルタ値の 3/4 を足した値を返します。
- **Jitter** – 最新のセンサ値が最後のセンサ値よりも大きい場合、前のフィルタ値が 1 ずつインクリメントされ、小さい場合は値がデクリメントされます。
- **First Order IIR 1/8** – 最新のセンサ値の 1/8 に前のフィルタ値の 7/8 を足した値を返します。
- **First Order IIR 1/16** – 最新のセンサ値の 1/16 に前のフィルタ値の 15/16 を足した値を返します。

防水性および検知

この機能により CapSense CSD が防水サポートされるよう設定します(初期設定ではディスエーブル)。この機能で、次のパラメータを設定します:

- 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でのご使用を希望しない場合、**Advanced** タブでこれを削除することができます。

Enable clock input

このパラメータで、コンポーネントが内部クロックを使用するか、ユーザ提供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(初期設定ではディスエーブル)。

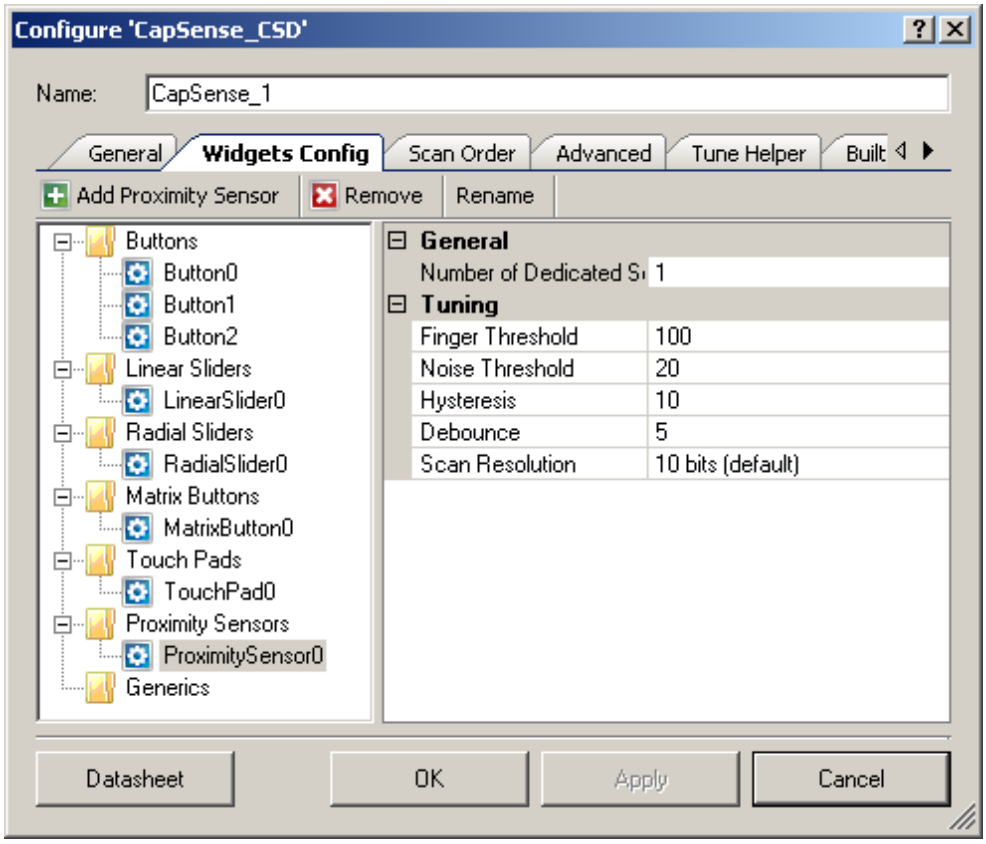
注チューニング手法が **Auto (SmartSense)**の場合、このオプションは使用できません。これは、カスタマイザが内部データの計算にクロック周波数を把握する必要があるためです。

「Scan Clock(スキャンクロック)」

このパラメータで、内部 CapSense コンポーネントのクロック周波数を指定します。値の範囲は 3MHz～24MHz(初期設定:12MHz)です。この機能は **Enable clock input** が選択されている場合は使用できません。

注 Analog Switch Drive Source を **FF Timer** に、**Digital Implementation** を **FF Timer** に、または両方に設定した場合、BUS_CLK 以下の CapSense CSD クロックをサポートしません。そのため BUS_CLK を選択してください。

Widgets Config タブ



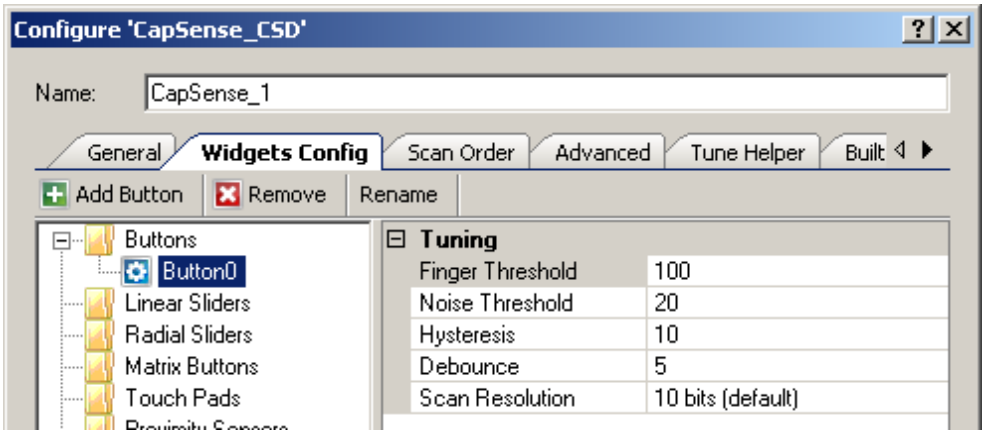
様々なパラメータの定義については「機能説明」セクションに記載されています。

ツールバー

ツールバーには以下のコマンドが含まれています:

- **Add widget** (ホットキー-Insert) – 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:
  - **Buttons** – ボタンは単一のセンサを指で押した場合これを検知し、単一の機械的ボタン置換を提供します。
  - **Linear Sliders** – リニア スライダは、センサ数が少ない場合に指が押した位置の補間に基づいて整数値を提供します。
  - **Radial Sliders** – ラジアル スライダはリニア スライダに類似していますが、センサが円形に配置されている点が異なります。
  - **Matrix Buttons** – マトリックスボタンは、行センサと列センサにより形成される交差部分を指で押したことを検知します。マトリックスボタンは、多数のボタンをスキャンする際に有効な手法です。
  - **Touch Pads** – タッチパッドは、指で押したタッチパッド領域内の X と 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により形成されます。
  - **Proximity Sensors** – 近接検知センサにより、センサから離れた位置にある指、手、またはその他の大きな物体の検出が最適化されます。これにより実際に触れる必要がなくなります。
  - **Generic Sensors** – 汎用センサは、単一センサからの Raw データを提供します。これにより、他のセンサの種類での処理済みの出力で不可能な、独自のセンサまたは高度なセンサを作成することができます。
- **Remove widget** (ホットキー-Delete) – ツリーから選択したウィジェットを削除します。
- **Rename** (ホットキー–F2) – ダイアログを開き、選択したウィジェット名を変更します。ウィジェットをダブルクリックしても、このダイアログを開くことができます。

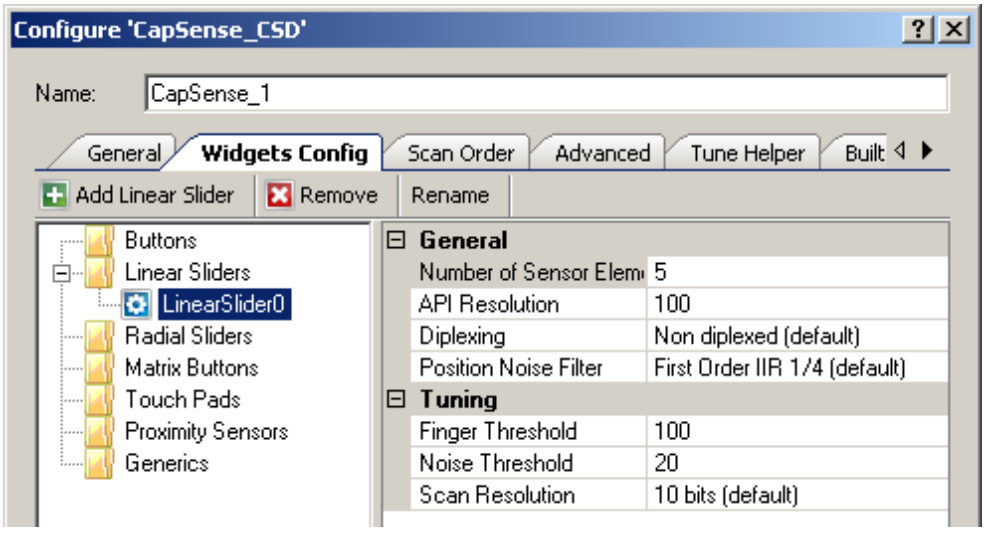
ボタン



チューニング:

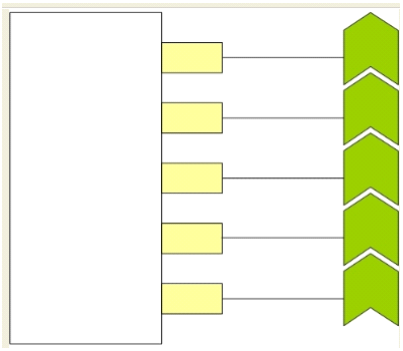
- **Finger Threshold** – センサのアクティブな閾値を定義します。その結果、タッチの検出感度が上昇または低下します。センサのスキャン値がこの閾値よりも大きい場合、ボタンにタッチしたとレポートされます。初期値は **100** です。有効な値の範囲は[1…255]です。**指閾値+ヒステリシス**が 254 を上回ることはできません。
- **Noise Threshold** – センサのノイズ閾値を定義します。この閾値を上回るカウント値が観測された場合、ベースラインは更新しません。ノイズ閾値が低すぎると、センサおよび温度オフセットが含まれない場合があります。これは、タッチが正しく検知されなかったり見つからない原因となる場合があります。ノイズ閾値が高すぎると、指のタッチがノイズとして解釈され、不自然にベースラインが増え、結果として指のタッチが見つからない原因となることがあります。初期値は **20** です。有効な値の範囲は[1…255]です。
- **Hysteresis** – センサのアクティブ状態の遷移で差動ヒステリシスを追加します。センサが非アクティブなの場合、差の数は指閾値＋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なの場合、差の数は指閾値－ヒステリシスを下回る必要があります。ヒステリシスにより、低振幅のセンサノイズおよび小さな指の動きにより、ボタン状態のサイクルを発生させないようにします。初期値は **10** です。有効な値の範囲は[1…255]です。**指閾値+ヒステリシス**が 254 を上回ることはできません。
- **Debounce** – デバウンスカウンタを追加し、センサのアクティブな状態へ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"値が指の閾値+ヒステリシスを上回る状態を維持しなければなりません。初期値は **5** です。デバウンスは、高周波の高振幅ノイズによる、ボタン押しの誤検出を防ぎます。有効な値の範囲は[1…255]です。
- **Scan Resolution** – スキャン分解能を定義します。このパラメータは、ボタンウィジェット内のセンサのスキャン時間に影響を及ぼします。ビット数が N の場合、スキャン分解能の最大生カウントは 2^N–1 です。分解能が上昇すると、感度およびタッチ検知の信号対ノイズ比(SNR)が向上しますが、スキャン時間が長くなります。初期値は **10 ビット**です。

リニア スライダ

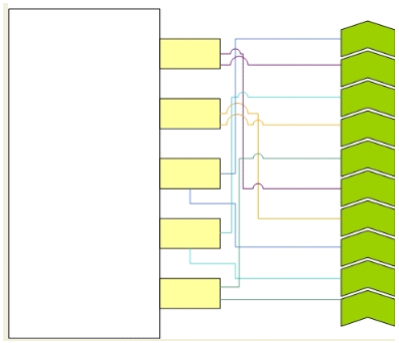


全般:

- **Numbers of Sensor Elements** – スライダ内のエレメント数を定義します。センサエレメントの API 分解能に適した比率は 20:1 です。API 分解能とセンサエレメントの比率を上げすぎると、計算される指位置のノイズが上昇する結果をもたらすことがあります。有効な値の範囲は[2…32]です。初期値は **5** エレメントです。
- **API Resolution** – スライダ分解能を定義します。この範囲で位置の値が変更されます。有効な値の範囲は[1…255]です。
- **Diplexing – Non diplexed**(初期設定)または **Diplexed**。ダイプレックスによって、2 つのスライダセンサが単一のデバイスピンを共有できるようになるため、所与のスライダセンサ数で必要なピンの合計数を減らすことができます。



Non Diplexed



Diplexed

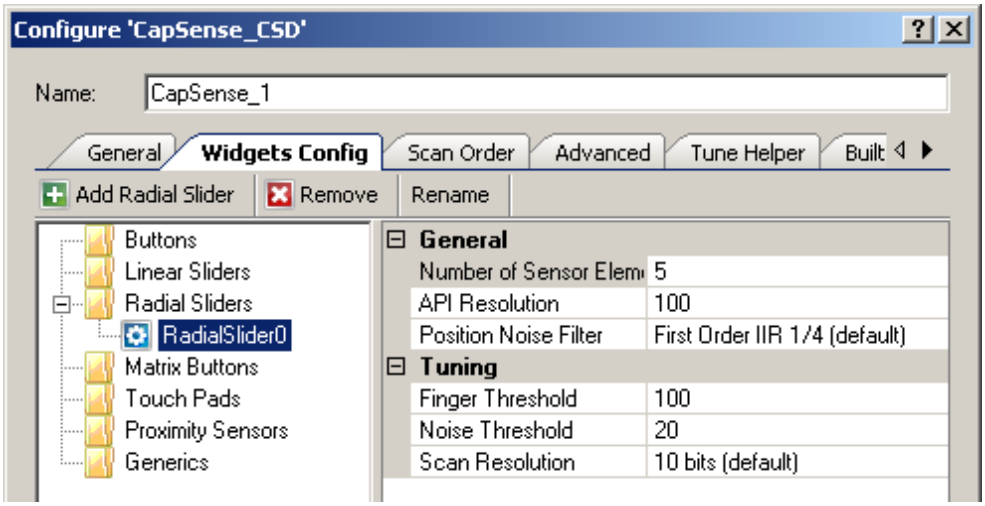
- **Position Noise Filter** – 位置計算を実行するノイズフィルタの種類を選択します。選択したウィジェットに 1 つのフィルタのみが適用できます。フィルタの種類についての詳細は、この文書の「機能説明」セクションの「フィルタ」を参照してください。
  - **None**
  - **Median**
  - **Averaging**
  - **First Order IIR 1/2**
  - **First Order IIR 1/4**(初期設定)
  - **Jitter**

チューニング:

- **Finger Threshold** – センサのアクティブな閾値を定義します。その結果、タッチの検出感度が上昇または低下します。センサのスキャン値がこの閾値よりも大きい場合、ボタンにタッチしたとレポートされます。初期値は **100** です。有効な値の範囲は[1…255]です。

- **Noise Threshold** – スライダ エレメントの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、センサおよび温度オフセットが含まれない場合があります。これは、タッチが正しく検知されなかったり見つからない原因となる場合があります。ノイズ閾値が高すぎると、指のタッチがノイズとして解釈され、不自然にベースラインが増え、結果としてセントロイドの場所の計算エラーの原因となることがあります。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。初期値は **20** です。有効な値の範囲は[1…255]です。
- **Scan Resolution** – スキャン分解能を定義します。このパラメータは、リニア スライダウィジェット内のすべてのセンサのスキャン時間に影響を及ぼします。ビット数が N の場合、スキャン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、検出感度とタッチ検知の S/N 比が高くなりますが、スキャン時間も長くなります。初期値は **10 ビット**です。

ラジアルスライダ



全般:

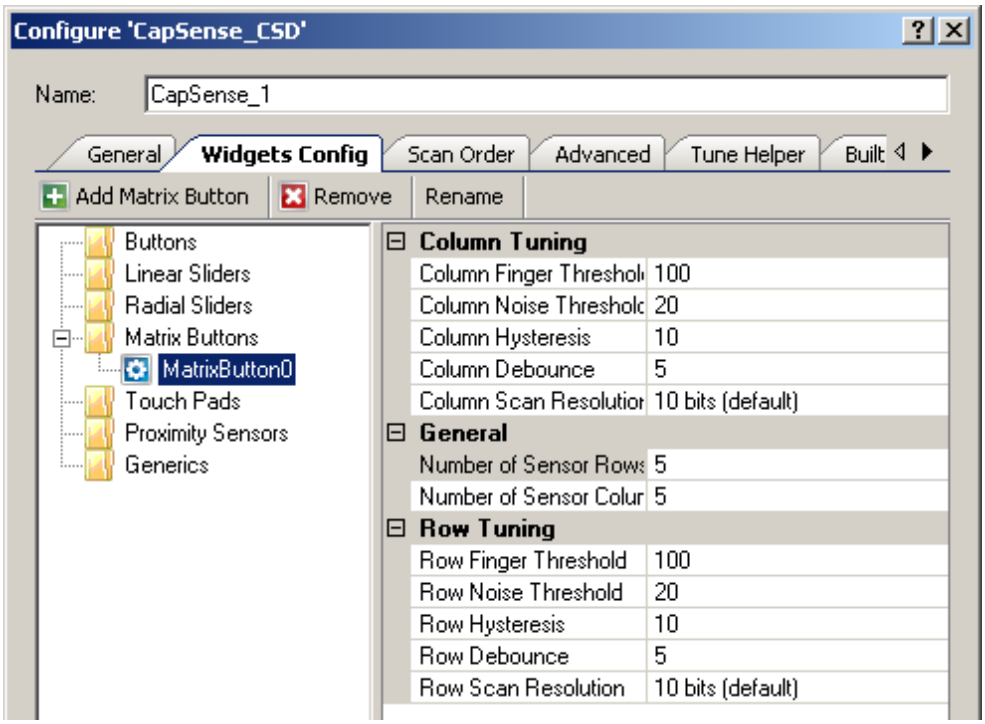
- **Numbers of Sensor Elements** – スライダ内のエレメント数を定義します。センサ エレメントの API 分解能に適した比率は 20:1 です。API 分解能とセンサ エレメントの比率を上昇しすぎると、分解能の計算でのノイズが上昇する結果になることがあります。有効な値の範囲は[2…32]です。初期値は **5** エレメントです。
- **API Resolution** – スライダの分解能を定義します。この範囲で位置の値が変更されます。有効な値の範囲は[1…255]です。
- **Position Noise Filter** – 位置計算を実行するノイズフィルタの種類を選択します。選択したウィジェットに 1 つのフィルタのみが適用できます。フィルタの種類についての詳細は、このデータシートの「機能説明」セクションの「フィルタ」を参照してください。
  - ❑ **None**
  - ❑ **Median**

- **Averaging**
- **First Order IIR 1/2**
- **First Order IIR 1/4**(初期設定)
- **Jitter**

チューニング:

- **Finger Threshold** – センサのアクティブな閾値を定義します。その結果、タッチの検出感度が上昇または低下します。センサのスキャン値がこの閾値よりも大きい場合、ボタンにタッチしたとレポートされます。初期値は **100** です。
- **Noise Threshold** – スライダ エレメントの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、センサおよび温度オフセットが含まれない場合があります。これは、タッチが正しく検知されなかったり見つからない原因となる場合があります。ノイズ閾値が高すぎると、指のタッチがノイズとして解釈され、不自然にベースラインが増え、結果としてセントロイドの場所の計算エラーの原因となることがあります。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。初期値は **20** です。有効な値の範囲は[1…255]です。
- **Scan Resolution** – スキャン分解能を定義します。このパラメータは、ラジアルスライダウィジェット内のすべてのセンサのスキャン時間に影響を及ぼします。ビット数が N の場合、スキャン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、検出感度とタッチ検知の S/N 比が高くなりますが、スキャン時間も長くなります。初期値は **10 ビット**です。

マトリックスボタン



チューニング:

- **Column and Row Finger Threshold** – マトリックスボタンの列と行でのセンサのアクティブな閾値を定義します。その結果、タッチの検出感度が上昇または低下します。センサのスキャン値がこの閾値よりも大きい場合、ボタンにタッチしたとレポートされます。初期値は **100** です。有効な値の範囲は[1…255]です。**指閾値+ヒステリシス**が 254 を上回ることはできません。
- **Column and Row Noise Threshold** – マトリックスボタンの列と行での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、センサおよび温度オフセットが含まれない場合があります。これは、タッチが正しく検知されなかったり見つからない原因となる場合があります。ノイズ閾値が高すぎると、指のタッチがノイズとして解釈され、不自然にベースラインが増えることがあります。これは指のタッチが見つからない原因となる場合があります。初期値は **20** です。有効な値の範囲は[1…255]です。
- **Column and Row Hysteresis** – マトリックスボタンの列と行におけるセンサのアクティブ状態の遷移で、差動ヒステリシスを追加します。センサが非アクティブな場合、差の数は指閾値＋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、差の数は指閾値－ヒステリシスを下回る必要があります。ヒステリシスにより、低振幅のセンサノイズおよび小さな指の動きにより、ボタン状態のサイクルを発生させないようにします。初期値は **10** です。有効な値の範囲は[1…255]です。**指閾値+ヒステリシス**が 254 を上回ることはできません。

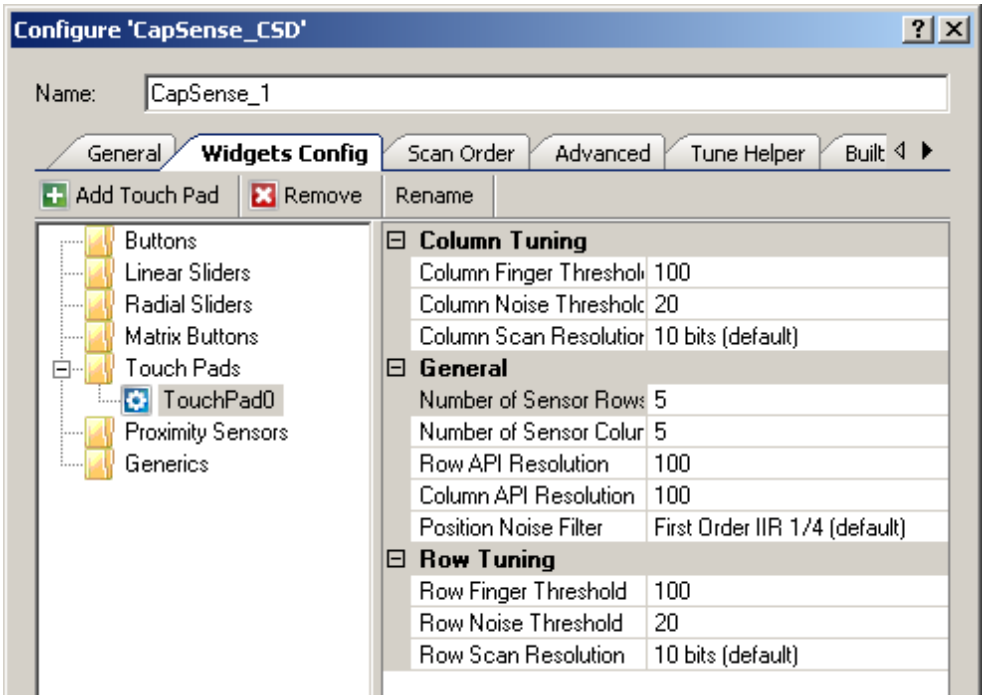
- **Column and Row Debounce** – マトリックスボタンの列と行におけるセンサのアクティブ状態の遷移で、検出用のデバウンスカウンタを追加します。センサが非アクティブからアクティブへ遷移するためには、指定されたサンプル数に対して、"差の数"値が指の閾値+ヒステリシスを上回る状態を維持しなければなりません。初期値は **5** です。デバウンスにより、高周波数の高振幅ノイズにより、ボタンが押されたと誤検出しないようにします。有効な値の範囲は[1…255]です。

- **Column and Row Scan Resolution** – マトリックスボタンの列と行のスキャン分解能を定義します。このパラメータは、マトリックスボタンウィジェット内の列または行にあるすべてのセンサのスキャン時間に影響を及ぼします。ビット数が N の場合、スキャン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、検出感度とタッチ検知の S/N 比が高くなりますが、スキャン時間も長くなります。同じ感度レベルを得るためには、列および行のスキャン分解能は同一でなければなりません。初期値は **10 ビット**です。

全般:

- **Number of Sensor Columns and Rows** – マトリックスを形成する列と行の数を定義します。有効な値の範囲は[2…32]です。列と行の両方で、初期値は **5** エレメントです。

タッチパッド



チューニング:

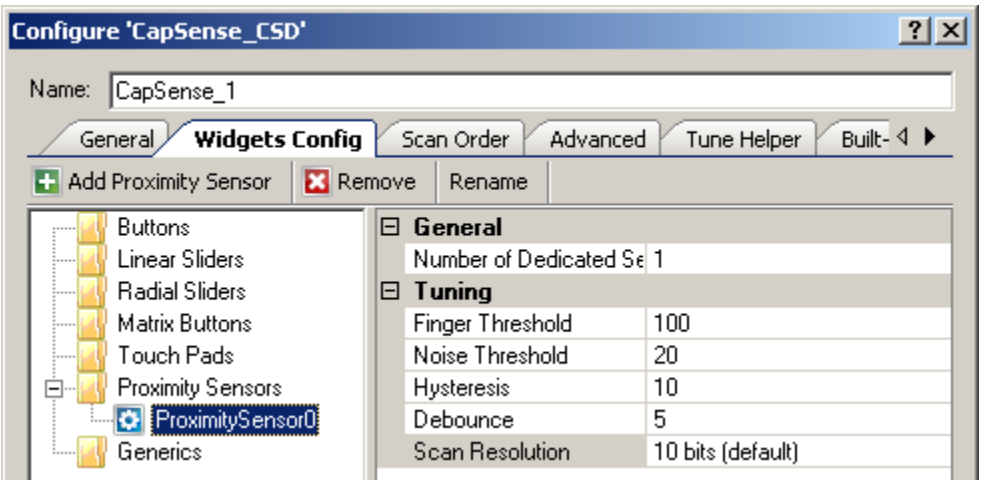
- **Column and Row Finger Threshold** – タッチパッドボタンの列と行でのセンサのアクティブな閾値を定義します。その結果、タッチの検出感度が上昇または低下します。センサのスキャン値がこの閾値よりも大きい場合、タッチパッドはタッチ位置をレポートします。初期値は **100** です。有効な値の範囲は[1…255]です。

- **Column and Row Noise Threshold** – タッチパッドの列と行での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。この閾値を下回るカウント値が観測された場合、セントロイドの位置の計算に加えられません。ノイズ閾値が低すぎると、センサおよび温度オフセットが含まれない場合があります。これは、タッチが正しく検知されなかったり見つからない原因となる場合があります。ノイズ閾値が高すぎると、指のタッチがノイズとして解釈され、不自然にベースラインが増えることがあります。その結果、セントロイドの計算エラーが発生する場合があります。初期値は **20** です。有効な値の範囲は[1…255]です。
- **Column and Row Scan Resolution** – タッチパッドの列と行のスキャン分解能を定義します。このパラメータは、タッチパッドウィジェット内の列または行にあるすべてのセンサのスキャン時間に影響を及ぼします。ビット数が N の場合、スキャン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、検出感度とタッチ検知の S/N 比が高くなりますが、スキャン時間も長くなります。同じ感度レベルを得るためには、列および行のスキャン分解能は同等でなければなりません。初期値は **10 ビット**です。

全般:

- **Number of Sensor Column and Row** – タッチパッドを形成する列と行の数を定義します。有効な値の範囲は[2…32]です。列と行の両方で、初期値は **5** エレメントです。
- **API ResolutionColumn and Row** – タッチパッドの列と行の分解能を定義します。この範囲内の指の位置の値がレポートされます。有効な値の範囲は[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。選択したウィジェットに 1 つのフィルタのみが適用できます。フィルタの種類についての詳細は、このデータシートの「機能説明」セクションの「フィルタ」を参照してください。
  - ❑ **None**
  - ❑ **Median**
  - ❑ **Averaging**
  - ❑ **First Order IIR 1/2**
  - ❑ **First Order IIR 1/4**(初期設定)
  - ❑ **Jitter**

近接検知センサ



全般:

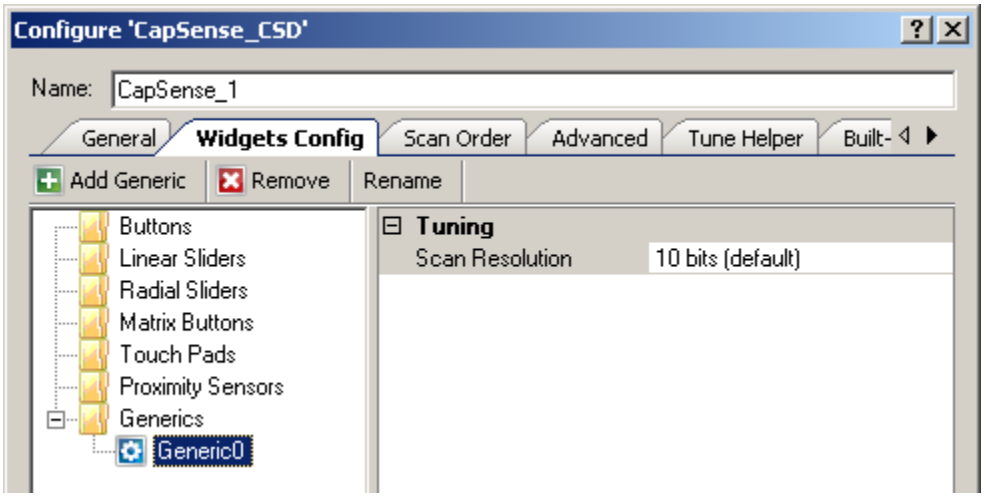
- **Number of Dedicated Sensor Elements** – 専用の近接検知センサの数を選択します。センサ エレメントは、他のウィジェットで使用されるその他すべてのセンサに追加されます。任意のウィジェットセンサを個別に使用したり、並列に接続して近接検知センサを作成することができます。
  - **0** – 近接検知センサは既存の一部のセンサのみをスキャンし、近接検知を行います。このウィジェットには新しいセンサは割り当てられていません。
  - **1**(初期設定) – システム内の専用近接検知センサの数。

チューニング:

- **Finger Threshold** – センサのアクティブな閾値を定義します。その結果、タッチの近接検出感度が上昇または低下します。センサのスキャン値がこの閾値よりも大きい場合、近接検知センサにタッチしたとレポートされます。初期値は **100** です。有効な値の範囲は[1…255]です。**指閾値+ヒステリシス**が 254 を上回ることはできません。
- **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、センサおよび温度オフセットが含まれない場合があります。これは近接検知タッチが正しく検知されなかったり見つからない原因となる場合があります。ノイズ閾値が高すぎると、指のタッチがノイズとして解釈され、不自然にベースラインが増えることがあります。これは指のタッチが見つからない原因となる場合があります。有効な値の範囲は[1…255]です。
- **Hysteresis** – センサのアクティブ状態の遷移で差動ヒステリシスを追加します。センサが非アクティブな場合、差の数は指閾値＋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な場合、差の数は指閾値－ヒステリシスを下回る必要があります。ヒステリシスにより、低振幅のセンサノイズおよび小さな指の動きにより、近接検知センサ状態のサイクルを発生させないようにします。有効な値の範囲は[1…255]です。

- **Debounce** – デバウンスカウンタを追加し、センサのアクティブ状態へ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"値が指の閾値+ヒステリシスを上回る状態を維持しなければなりません。デバウンスにより、高周波数の高振幅ノイズにより、近接検知イベントを誤検出しないようにします。有効な値の範囲は[1…255]です。
- **Scan Resolution** – スキャン分解能を定義します。このパラメータは、近接検知ウィジェットのスキャン時間に影響を及ぼします。ビット数が N の場合、スキャン分解能の最大 Raw カウントは $2^N - 1$ です。分解能を高くすると、検出感度とタッチ検知の S/N 比が高くなりますが、スキャン時間も長くなります。近接検知には、通常のボタンで使用されるよりも高分解能を使用して、検出範囲を広げることが最適です。初期値は **10 ビット**です。

Generics



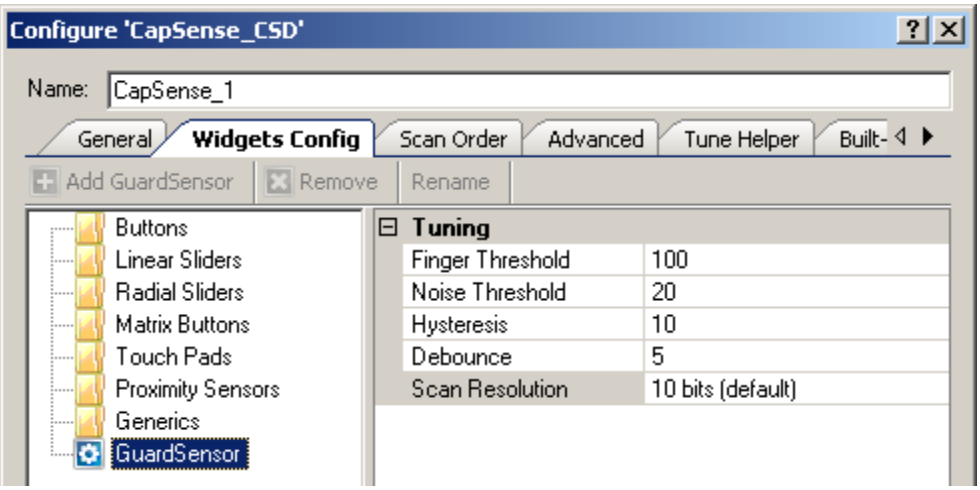
チューニング:

- **Scan Resolution** – スキャン分解能を定義します。このパラメータは、汎用ウィジェットのスキャン時間に影響を及ぼします。ビット数が N の場合、スキャン分解能の最大 Raw カウントは $2^N - 1$ です。分解能を高くすると、検出感度とタッチ検知の S/N 比が高くなりますが、スキャン時間も長くなります。初期値は **10 ビット**です。

汎用ウィジェットでは、チューニングオプション 1 つのみが使用できます。これは、事前定義済みのウィジェットに適合しない CapSense センサやアルゴリズムをサポートする高レベル処理はすべてユーザに任されているためです。

Guard Sensor

この特別なセンサは **Advanced** タブを使用し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で押してもレポートしませんが、他のウィジェット付近の無効な条件をレポートし、更新を抑制します。このセンサの種類についての詳細情報および使用するタイミングは、このデータシートの「機能説明」セクションの「ガードセンサの実装」を参照してください。



チューニング:

- **Finger Threshold** – センサのアクティブな閾値を定義します。その結果、タッチの検出感度が上昇または低下します。センサのスキャン値がこの閾値よりも大きい場合、ガードセンサにタッチしたとレポートされます。初期値は **100** です。有効な値の範囲は[1…255]です。**指閾値**+**ヒステリシス**が 254 を上回ることはできません。

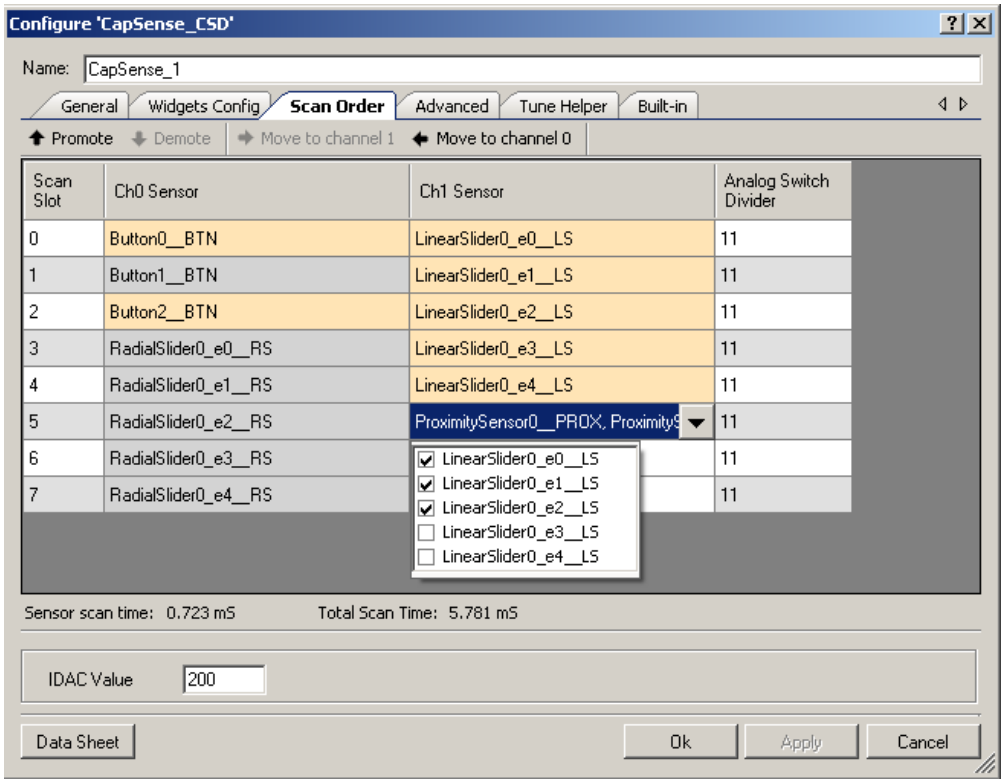
- **Noise Threshold** – センサのノイズ閾値を定義します。この閾値を上回るカウント値が観測された場合、ベースラインは更新しません。ノイズ閾値が低すぎると、センサおよび温度オフセットが含まれない場合があります。これは、タッチが正しく検知されなかったり見つからない原因となる場合があります。ノイズ閾値が高すぎると、指のタッチがノイズとして解釈され、不自然にベースラインが増えることがあります。これは指のタッチが見つからない原因となる場合があります。初期値は **20** です。有効範囲は[1…255]です。

- **Hysteresis** – センサのアクティブ状態の遷移で差動ヒステリシスを追加します。センサが非アクティブな場合、差の数は指閾値＋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な場合、差の数は指閾値－ヒステリシスを下回る必要があります。ヒステリシスにより、低振幅のセンサノイズおよび小さな指の動きにより、ボタン状態のサイクルを発生させないようにします。初期値は **10** です。有効な値の範囲は[1…255]です。**指閾値**+**ヒステリシス**が 254 を上回ることはできません。

- **Debounce** – デバウンスカウンタを追加し、センサのアクティブな状態へ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"値が指の閾値+ヒステリシスを上回る状態を維持しなければなりません。デバウンスにより、高周波数の高振幅ノイズにより、ガードセンサを誤検出しないようにします。初期値は **5** です。有効な値の範囲は[1…255]です。

- **Scan Resolution** – スキャン分解能を定義します。このパラメータは、ガードセンサのスキャン時間に影響を及ぼします。ビット数が N の場合、スキャン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、検出感度とタッチ検知の S/N 比が高くなりますが、スキャン時間も長くなります。初期値は **10 ビット**です。

Scan Order タブ



ツールバー

ツールバーには以下のコマンドが含まれています:

- **Promote/Demote**(ホットキー- +/-) – 選択したウィジェットをデータグリッドで上または下に移動します。ウィジェットのエレメントの一部が選択されている場合、ウィジェット全体が選択されます。
- **Move to Channel 1/Channel 0**(ホットキー-Shift + 1/0) – 選択したウィジェットを別のチャンネルに移動します。このオプションは 2 チャンネル デザインでのみアクティブです。ウィジェットのエレメントの一部が選択されている場合、ウィジェット全体が選択されます。

注スキャン順序が変更された場合、ピンを再割り当てする必要があります。

注近接検知センサは、初期設定ではスキャン処理から除外されます。このスキャンは、通常は他のセンサと同時にスキャンされないため、ランタイムにマニュアルで開始する必要があります。

追加のホットキー

- **Ctrl + A** – すべてのセンサを選択します。
- **Delete** – 複雑なセンサからすべてのセンサを削除します(汎用ウィジェットおよび近接ウィジェットにのみ適用)。

Analog Switch Divider Column

Analog Switch Divider の値を指定し、スキャンスロットのプリチャージスイッチ出力周波数を決定します。有効な値の範囲は[1…255]です。初期値は **11** です。

Analog Switch Drive Source が **Direct** に設定されているか、または **Multiple Analog Switch Divider** がディスエーブルになっている場合(**Advanced** タブ上)この列は表示されません。

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 Sourcing** が **Current Source** として選択されている場合のみアクティブになります(**Advanced** タブ下)。有効範囲は 0～255 です。初期値は **200** です。

Sensitivity

感度は、センサが動作するのに必要となる C(センサの静電容量)の公称(Nominal)値の変化です。値の有効範囲は[1…100]です。これは、以下の感度レベルに対応しています。0.1、0.2、0.3、および 10pF。初期値は **2** です。**Sensitivity** により、センサの全体の感度を設定し、オーバーレイ材質の様々な厚さを含めます。厚い材質では低い感度値を使用してください。

このオプションは、**Tuning method** パラメータが **Auto (SmartSense)**に設定されている場合のみ使用できます。

Sensor Scan Time

標準的なシステムで選択したセンサに必要となるスキャン時間の概数を表示します。

Auto (SmartSense)がチューニング手法として選択されている場合、パラメータがチューニング処理により変更されるため、表示される値は正確でないことがあります。CapSense CSD コンポーネント入力クロック周波数が不明の場合、**Unknown** 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は、センサのスキャン時間に影響を及ぼします:

- スキャン速度
- 分解能
- CapSense CSDclock

注ここで表示されているスキャン時間には、スキャン時間、推定セットアップ時間、前処理時間が含まれています。これは他のデザイン部分、選択したコンパイラ、選択したデバイス(PSoC3 または PSoC5)に依存するため、単独に決まる値ではありません。

Total Scan Time

すべてのセンサのスキャンに必要な合計スキャン時間を表示します。この値はセンサのスキャン時間の概数であり、実際の値とはわずかに異なる場合があります。

Auto (SmartSense)がチューニング手法として選択されている場合、パラメータがチューニング中に変更されるため、表示される値は正確でないことがあります。CapSense CSD コンポーネント入力クロック周波数が不明の場合、**Unknown** が表示されます。

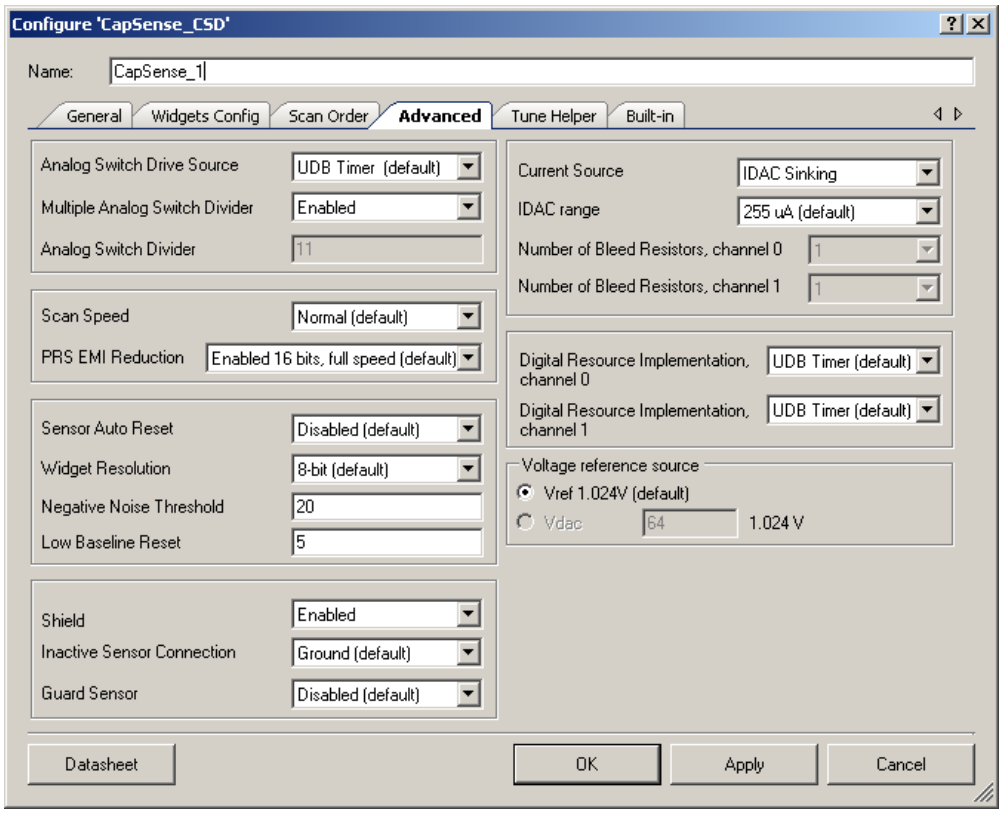
Widget List

表に交互にグレーとオレンジでウィジェットが一覧表示されます。ウィジェットに関連付けられているすべてのセンサが同じ色を共有し、異なるウィジェット エレメントを強調表示します。

近接検知スキャンセンサは、専用近接検知センサを使用するか、または専用センサと他のセンサの両方を組み合わせて近接検知することができます。たとえば、ボタンのアレイ全体に至るトレースがあるボードで、近接検知センサはトレースおよびアレイのすべてのボタンで構成することができます。近接検知と同時に、これらすべてのセンサがスキャンされます。近接検知スキャンセンサではドロップダウンを使用して、1 つまたは複数のセンサを選択し、近接検知することができます。

近接検知センサと同様に、汎用センサも複数のセンサで構成することができます。汎用センサは専用センサ、その他の既存のセンサ、あるいは複数のセンサからデータを取得できます。ドロップダウンでセンサを選択します。

Advanced タブ



Analog Switch DriveSource

このパラメータは、アナログスイッチドライバのソースを指定します。これにより、センサと変調コンデンサ C_{MOD} の間の切り替えレートを決定します。固定機能タイマブロック(FF タイマ)にタイマを実装すると、UDB リソースの使用が最小限に抑えられます。

- **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一に制限します。ほとんどのデザインへのご利用はお勧めできません。
- **UDB Timer**(初期設定) – UDB リソースを使用します
- **FF Timer** – UDB リソースを使用しません

Multiple Analog Switch Divider

このパラメータはアナログスイッチ ディバイダの使用を定義します。イネーブルにした場合、各スキャンスロットで専用のアナログスイッチ ディバイダの値が使用されます。それ以外の場合、センサはアナログスイッチ ディバイダの値のみを使用します。

この機能は、**Analog Switch Drive Source** が **Direct** に設定されている場合は使用できません。

Analog Switch Divider

このパラメータは、アナログスイッチディバイダの値を指定し、プリチャージスイッチ出力周波数を決定します。有効な値の範囲は[1…255]です。初期値は **11** です。

この機能は、**Analog Switch Drive Source** が **Direct** に設定されているか、または **Multiple Analog Switch Divider** が **Enabled** の場合は使用できません。

センサと変調コンデンサ C_{MOD} 間は、プリチャージクロックの速度で継続的に切り替えられます。**Analog Switch Divider** は CapSense CSD クロックを分割して、プリチャージクロックを生成します。ディバイダの値が低減すると、センサの切り替え速度が速くなり、Raw カウントが上昇します(またはその反対)。

スキャン速度

このパラメータは CapSense CSD コンポーネントデジタルロジッククロック周波数を指定します。これは、センサのスキャン時間を決定します。スキャン速度が遅いと時間がかかりますが、S/N 比が改善され、電源や温度の変化による影響を受けにくくなるなどのメリットがあります。

- **Slow** – コンポーネント入力クロックを 16 で分割します
- **Normal**(初期設定) – コンポーネント入力クロックを 8 で分割します
- **Fast** – コンポーネント入力クロックを 4 で分割します
- **Very Fast** – コンポーネント入力クロックを 2 で分割します

表 1. スキャン時間(単位:µs)対スキャン速度と分解能

分解能 (単位:ビット)	スキャン速度			
	Very Fast	Fast	Normal	Slow
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
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に基づく推定です。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、チャンネル数= 1。スキャン時間は、いずれかのセンサスキャンの間隔を測

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。これらの値を使用して、リニアにスケーリングすることで、他のクロックレートや追加のセンサのスキャンニング速度を推定することができます。

ここで表示されている値は、カスタマイザスキャン時間により推定される値とは異なる場合があります。これは、カスタマイザが行なったセットアップおよび前処理時間の近似値によるものです。

PRS EMI Reduction

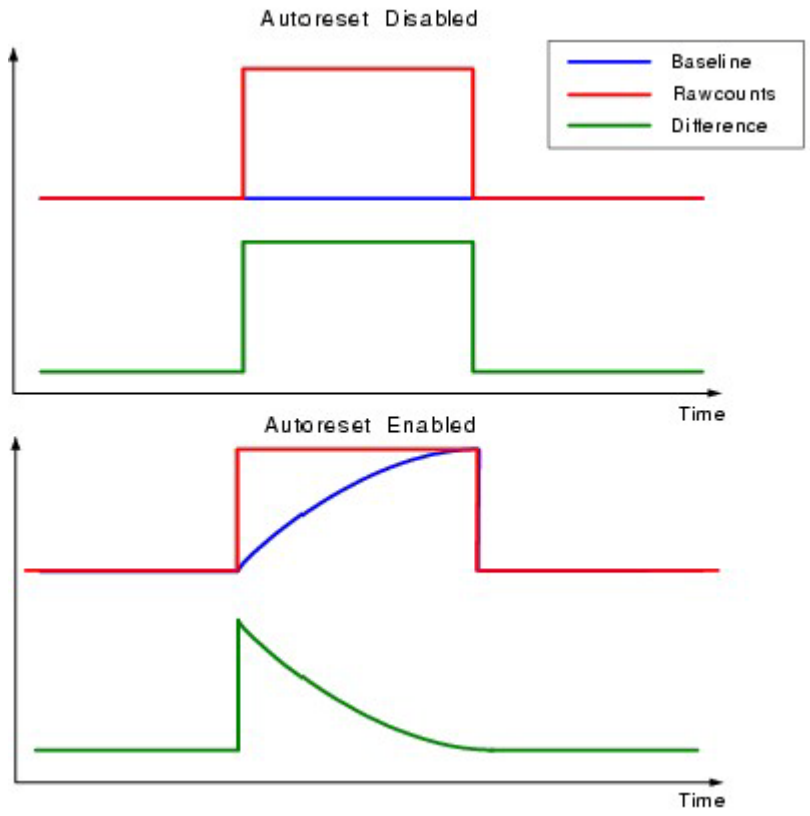
このパラメータで、アナログプリチャージクロックの生成に擬似ランダム系列(Psuedo Random Sequence、PRS)発生器が使用されているかどうかを指定します。PRS の使用を推奨します。これにより CapSense アナログスイッチ周波数のスペクトルを拡散し、EMI エミッションと感度を低減します。PRS クロックソースは **Analog Switch Divider** 設定により提供されます。PRS EMI の低減がイネーブルになっていない場合、単一周波数が使用され、基本周波数と高調波のエミッションが増加する原因となります。

- **Disabled**
- **Enabled 8 bits** – 8 ビットでは S/N 比が改善しますが、繰り返し期間が短くなると、EMI が増加します。
- **Enabled 16 bits, full speed**(初期設定) – 16 ビットでは S/N 比が低下しますが、EMI がより下がります。
- **Enabled 16 bits, 1/4 full speed** – **Enable 16 bits, full speed** より 4 倍速い PRS クロック出力が必要です。

Sensor Auto Reset

このパラメータは、差の数がノイズ閾値を上回っていても下回っていても、自動リセットをイネーブルにし、ベースラインが常に更新されるようにします。自動リセットがディスエーブルになっている場合、"差の数"がノイズ閾値のプラス/マイナス範囲内(ノイズ閾値はミラーリングされます)の場合のみベースラインが更新されます。何もセンサに触れずにRaw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。

- **Enabled** – 自動リセットにより、ベースラインが常に更新され、ボタンを押したのに検出されなかったり、ボタンがスタックするのを防げますが、ボタンが押されたとレポートされる最大時間の長さに制限されます。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 Raw 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。
- **Disabled**(初期設定) – 異常なシステム条件により、ノイズ閾値を継続的に超過して、ベースラインが更新を停止する場合があります。この結果、ボタンを押したのに検出されなかったり、ボタンがスタックすることがあります。メリットとして、ボタンが押された状態が無制限にレポートされ続けます。ボタンのスタックや無応答を検出するには、アプリケーション依存型手法を使用する必要があります。



Widget Resolution

このパラメータで、ウィジェットがレポートする信号分解能を指定します。8 ビット(1 バイト)が初期設定オプションです。ほとんどのアプリケーションではこの値を使用してください。ウィジェットの値が 8 ビットの範囲を超える場合、システムの感度が高すぎるため、公称(Nominal)値がミッドレンジ辺り(~128)に移動するようにチューニングが必要です。高精度が要求されるスライダウィジェットおよびタッチパッドウィジェットは 16 ビット分解能を使用できます。16 ビット分解能では、8 ビットで発生する可能性のある四捨五入エラーが防げるため、リニアリティが向上します。一方、センサごとに 2 バイトの SRAM 使用が追加で必要になります。

- **8 ビット**(1 バイト) – 初期設定
- **16 ビット**(2 バイト)

Negative Noise Threshold

このパラメータで Raw カウントと、ベースラインを Raw カウントレベルにリセットするベースラインレベルの負の差を指定します。これは **Sensor Auto Reset** パラメータが **Enabled** になっている際に使用します。

Low Baseline Reset

このパラメータは、ベースラインを Raw カウントレベルにする際に必要なベースラインを引いた Raw カウントのサンプル数を定義します。

Shield

このパラメータで、水滴や水膜の影響を除くために使用す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ついての詳細情報は、「シールド電極の使用および制限」セクションを参照してください。

- **Disabled**(初期設定)
- **Enabled**

Inactive Sensor Connection

このパラメータは、アクティブにスキャンされていないセンサすべての初期設定のセンサ接続を定義します。

- **Ground**(初期設定) – アクティブにスキャンされているセンサのノイズを低減するため、ほとんどのアプリケーションには、これを使用してください。
- **Hi-Z Analog** – 非アクティブなセンサを Hi-Z の状態にします。
- **Shield** – スキャンされていないセンサすべてにシールド波形を提供します。シールド信号の振幅は V_{DDIO}と同等になります。シールド電極と併用すると、防水性が向上し、ノイズが低減します。

Guard Sensor

このパラメータでガードセンサをイネーブルにし、防水性が必要なアプリケーションで水滴を検出できるようにします。**Water Proofing and detection** (**General** タブ下)が選択されていると、この機能は自動的にイネーブルになります。ガードセンサについての詳細情報は、このデータシートの「機能説明」セクションの「ガードセンサの実装」を参照してください。

- **Disabled**(初期設定)
- **Enabled**

Current Source

センサのタッチを検出するために、CapSense CSD は高精度な電流源が必要です。**IDAC Sinking** および **IDAC Sourcing** では、PSoC デバイスのハードウェア IDAC の使用が必要になります。**External Resistor** は IDAC よりも PCB のユーザ提供の抵抗を使用し、IDAC 制約のあるアプリケーションで役立ちます。

- **IDAC Sourcing**(初期設定) – IDAC は電流源を変調コンデンサ C_{MOD} にします。アナログスイッチは変調コンデンサ C_{MOD} および GND を代替するように設定され、電流シンキングを提供します。**IDAC Sourcing** は、3 つの手法の中で最高の信号対ノイズ比を実現するため、ほとんどのデザインで推奨されます。ただし、Vref レベルを設定するために、他のモードでは要求されない追加の VDAC リソースが必要となる場合があります。

- **IDAC Sinking** – IDAC は変調コンデンサ C_{MOD} からの電流をシンキングします。アナログスイッチは V_{DD} および変調コンデンサ C_{MOD} を代替するように設定され、電流源を提供します。これはほとんどのデザインで適切に動作しますが、S/N 比は一般的に **IDAC Sourcing** モードほど高くなりません。
- **External Resistor** – これは IDAC シンキング設定と同様に機能しますが、IDAC の代わりにグランド Rb のブリード抵抗になります。ブリード抵抗は変調コンデンサ C_{MOD} および GPIO の間に接続されます。GPIO は Open-Drain Drives Low drive モードに設定され、C_{MOD} が Rb を通じて放電されるようにします。このモードで要求されるリソースは最小限になるため、リソースの制約があるために必要な場合にのみ使用してください。このモードでは IDAC または VDAC は要求されないため、結果としてコンポーネントの最も低い電力設定となります。これは、電源がシステムの検討材料として非常に重要な場合に役立ちます。

IDACRange

このパラメータは **Current Source** の IDAC 範囲を指定します。**Current Source** が **External Resistor** に設定されている場合、このパラメータはディスエーブルになっています。ほぼすべての CapSense 設計で、初期設定が最善の選択です。高い電流範囲、または低い電流範囲は、一般的に非接触静電容量ベースのセンサでのみ使用されます。

- **32 uA**
- **255 uA**(初期設定)
- **2.04 mA**

Number of Bleed Resistors, channel 0/channel 1

このパラメータで、ブリード抵抗の数を指定します。各チャンネルのブリード抵抗の最大数は 3 つです。**Current Source** が **IDAC Source** または **IDAC Sink** に設定されている場合、この機能は使用できません。様々な電流を使用できるように、最大 3 グループのセンサに複数のブリード抵抗がサポートされており、システムチューニングに役立ちます。類似したセンサのサイズを使用したほとんどのデザインでは、ブリード抵抗は 1 つしか必要ありません。

Digital Resource Implementation, channel 0/channel 1

このパラメータで CapSense のデジタル部分の実装に使用するリソースの種類を指定します。これには、タイマやカウンタが含まれます。初期設定のパラメータは実装の柔軟性を最大限に保つよう設計されているため、ほとんどのデザインでは、このパラメータを変更しないでください。

- **UDB Timer(初期設定)** – 最も柔軟な実装ですが、貴重な UDB リソースを使用します
- **FF Timer** – FF タイマを実装することで UDB リソースが解放されますが、**Scan Speed** = **Very Fast** はサポートされません。

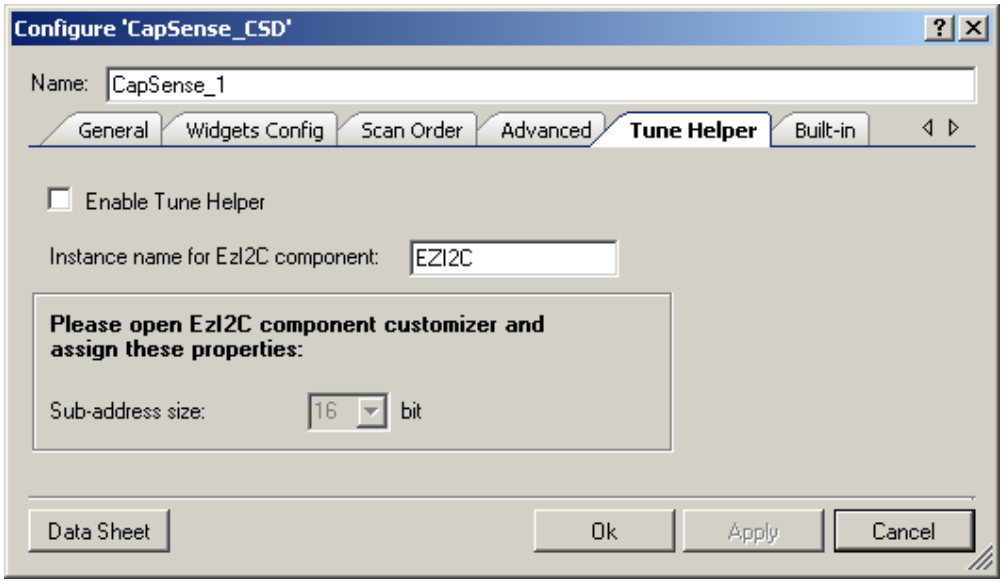
Voltage Reference Source

このパラメータで、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 Sourcing** モードではリファレンス電圧をできるだけ高く、**IDAC Sinking** モード、または **External Resistor Current Source** モードではリファレンス電圧をできるだけ低くすることが望まれます。

- **Vref 1.024V**(初期設定) – IDAC シンキングモードに最適です。
- **Vdac** – IDAC ソーシングモードに最適です。電圧 DAC を使用してリファレンス電圧を調整することで、使用可能な範囲を最大化できます。リファレンスソース VDAC は、**Current Source** が **IDAC Sourcing** に設定されており、VDAC デバイスリソースを必要とする場合にのみ使用できます。リファレンス電圧が上昇すると、感度も上昇しますが、シールド電極の影響は低減します。

VDAC が選択されていると、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ためです。そのために C_{MOD} が起動時に VDAC から Vref に充電されます。C_{MOD} から Vref に充電するために必要な時間により、ベースライン初期化が失敗する原因となります。通常は、2 度のベースライン初期化により問題が解消されます。

Tune Helper タブ



Enable Tune Helper

このパラメータは、チューナーGUI との通信を容易にサポートするために機能を追加します。チューナーGUI を使用する場合、この機能を使用してください。このオプションが選択されていない場合、通信機能は提供されますが、実際には何も起こりません。そのため、チューニングが完了した後、あるいはチューニング手法を変更した場合、これらの機能を削除する必要はありません。初期設定ではディスエーブルになっています。

Instance name for EZI2C component

このパラメータは、デザインに含まれる EZI2C コンポーネントで、チューナーGUI で通信用に使用するもののインスタンス名を定義します

注実際のインスタンス名が、ここで入力したインスタンス名に一致するかどうかを確認するリアルタイムのデザイン ルールチェックはありません。ユーザが一致することを確認してください。名前が一致しない場合、誤った名前の API が原因でプロジェクトビルドの途中でビルドエラーが生成されます。

チューナーGUI についての詳細情報は、このデータシートの『『チューナー GUI ユーザ ガイド』』セクションを参照してください。

リソース

以下の表は CapSense CSD アナログリソースおよびピンリソースを表示しています。

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 バッファ	
チャンネル:1 電流モード:外部抵抗	0	1	1	2 + Shield + SensorsNumber
チャンネル:2 電流モード:外部抵抗	0	2	2	4 + Shield + SensorsNumber
チャンネル:1 電流モード:IDACシンキング	1	1	1	1 + Shield + SensorsNumber
チャンネル:2 電流モード:IDACシンキング	2	2	2	2 + Shield + SensorsNumber
チャンネル:1 電流モード:IDACソーシング Vref: VDAC	2	1	1	1 + Shield + SensorsNumber
チャンネル:2 電流モード:IDACソーシング Vref: VDAC	4	2	2	2 + Shield + SensorsNumber

以下の表では CapSense CSD デジタルリソースを表示しています(Flash および RAM の使用にはスキャン API とスリープ API のみが含まれています)。

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
チャンネル:1 電流モード: 外部抵抗	4	19	0	1	1	1	1270	11
チャンネル:2 電流モード: 外部抵抗	6	31	0	1	1	2	2262	17
チャンネル:1 電流モード: IDACシンキング	4	19	0	1	1	1	1345	10
チャンネル:2 電流モード: IDACシンキング	6	31	0	1	1	2	2446	15
チャンネル:1 電流モード: IDAC ソーシング Vref: VDAC	4	18	0	1	1	1	1452	11
チャンネル:2 電流モード: IDAC ソーシング Vref: VDAC	6	30	0	1	1	2	2656	17

以下の表は CapSense CSD 高レベル API リソースを表示しています。

プロジェクトの詳細	APIメモリ(バイト)	
	フラッシュ	RAM
ウィジェットの種類:ボタン カウント:4	1197	22
ウィジェットの種類:非ダイプレックス リニア スライダ サイズ:5 センサ	1866	25
ウィジェットの種類:ダイプレックス リニア スライダ サイズ:5 センサ	2304	25

プロジェクトの詳細	APIメモリ(バイト)	
	フラッシュ	RAM
ウィジェットの種類:マトリックス ボタン サイズ:5x5 センサ	1526	55
ウィジェットの種類:ラジアル スライダ サイズ:5 センサ	1704	25
ウィジェットの種類:タッチパッド サイズ:5x5 センサ	2289	48

『チューナー GUI ユーザ ガイド』

このセクションには、CapSense チューナーの使用に役立つ情報および手順を記載しています。

CapSense チューナーは、マニュアルチューニング モードの際に、CapSense コンポーネントを特定のシステムの環境向けにチューニングするのに役立ちます。また、コンポーネントが SmartSense モードの際には、チューニング値(読み取り専用)およびパフォーマンスを表示できます。コンポーネントがチューニング モードのいずれでもない場合は、チューニングはサポートされません。これは、あらゆるパラメータがフラッシュに保存され、最小限の SRAM 使用では読み取り専用になるためです。

CapSense チューニングのプロセス

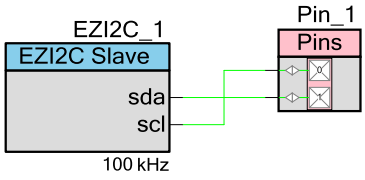
以下は CapSense コンポーネントの使用およびチューニングのための標準的なプロセスです:

PSoC Creator でのデザインの作成

PSoC Creator ヘルプを必要に応じて参照してください。

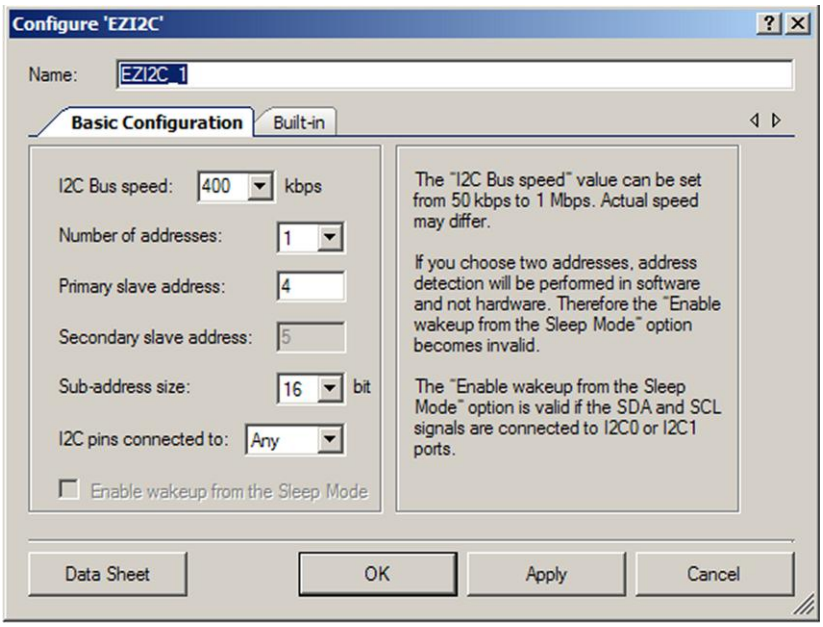
EZI2C コンポーネントの配置および設定

1. EZI2C コンポーネントを Component Catalog からデザインにドラッグします。



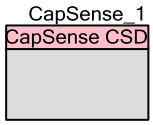
2. これをダブルクリックして **Configure** ダイアログを開きます。
3. 以下のようにパラメータを変更し、**OK** をクリックしてダイアログを閉じます。
   - ❑ サブアドレスのサイズは 16 ビットである必要があります。

- 生成された API が機能するためには、インスタンス名は **CapSense CSD Configure** ダイアログ (**Tune Helper** タブ下)で使用した名前と一致する必要があります。

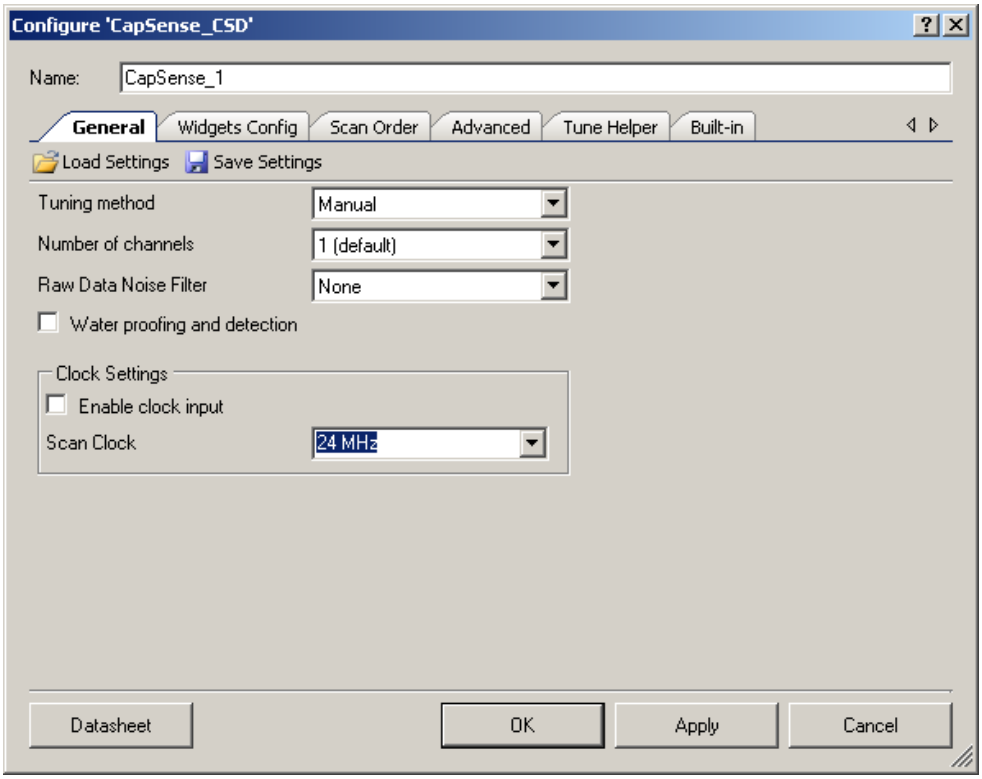


CapSense コンポーネントの配置および設定

1. CapSense_CSD コンポーネントを Component Catalog からデザインにドラッグします。



2. これをダブルクリックして **Configure** ダイアログを開きます。
3. アプリケーションの要件に応じて、CapSense CSD パラメータを変更します。**Tuning method** に **Manual** または **Auto (SmartSense)** を選択します。**OK** をクリックしてダイアログを閉じ、パラメータを保存します。



Auto (SmartSense)の選択

Auto (SmartSense)により CapSense CSD コンポーネントをシステムの詳細にあわせて自動的にチューニングすることができます。CapSense CSD パラメータは、ファームウェアによりランタイムに計算されます。このモードでは追加の RAM および CPU 時間が使用されます。Auto (SmartSense)はエラー発生の原因となりやすく繰り返しが多い CapSense CSD コンポーネント パラメータのマニュアルチューニングを不要にし、適切なシステムオペレーションができるようにします。Auto (SmartSense)を選択すると、以下の CSD パラメータがチューニングされます。

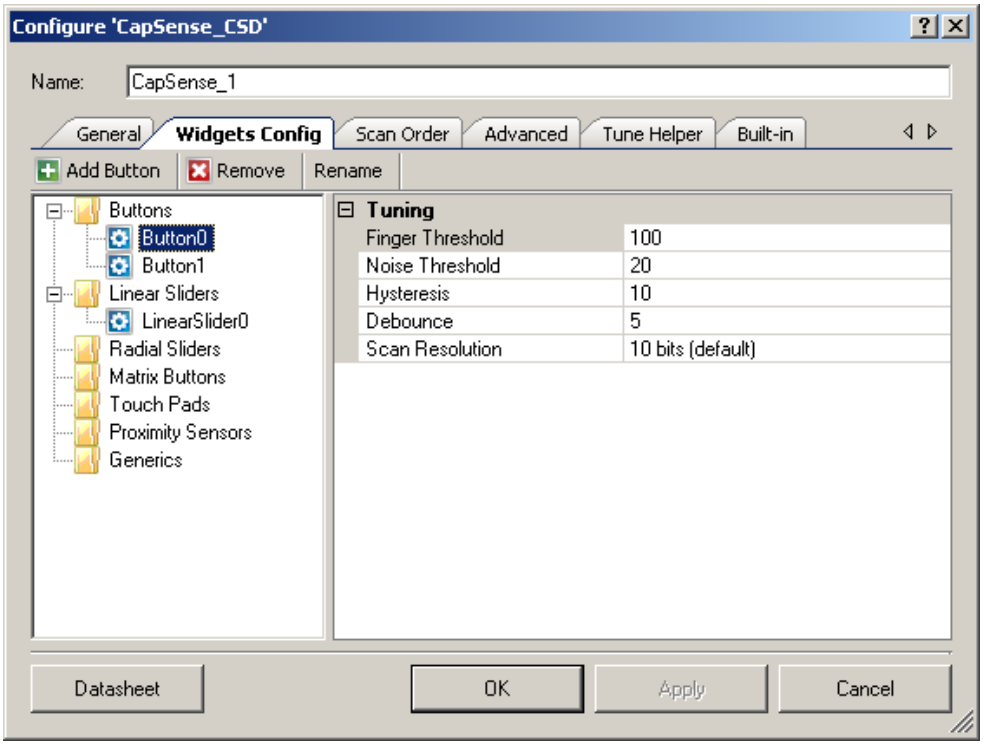
パラメータ	計算
Finger Threshold	センサのスキャン中、継続的に計算されます。
Noise Threshold	センサのスキャン中、継続的に計算されます。
IDAC Value	CapSense CSD起動時に、一度計算されます。
Analog Switch Divider	CapSense CSD起動時に、一度計算されます。
Scan Resolution	CapSense CSD起動時に、一度計算されます。

以下は、Auto (SmartSense)チューニングメソッドでのハードウェアパラメータの制限です。

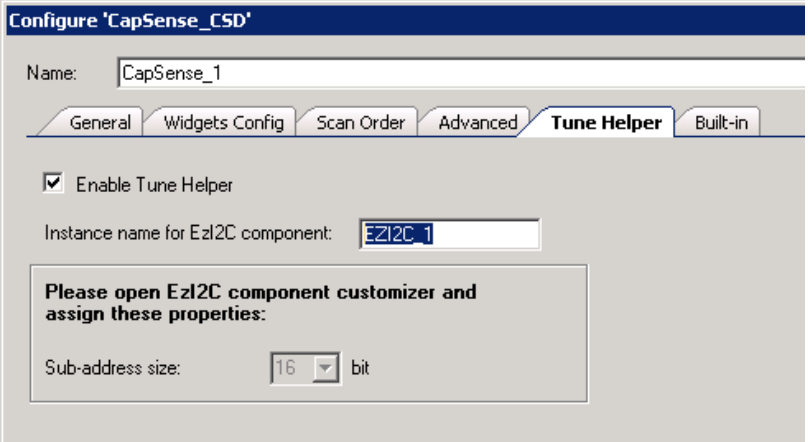
パラメータ	必要な設定
「Scan Clock（スキャンクロック）」	クロックは内部にする必要があります(**General**タブの**Enable clock input**をクリア)。
Current Source	IDACソーシング。
PRS EMI Reduction	イネーブルな16ビット。
スキャン速度	Normal
Vref	1.024 V

CapSense コンポーネントの設定

1. **Widgets Config** タブにウィジェットを追加し、設定します。



2. **Tune Helper** タブで、EZI2C コンポーネントのインスタンス名を入力し、**Enable Tune Helper** チェックボックスを選択する必要があります。



コードの追加

- チューナーの初期化およびプロジェクトの通信コードを *main.c* ファイルに追加します。*main.c* ファイルの例:

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

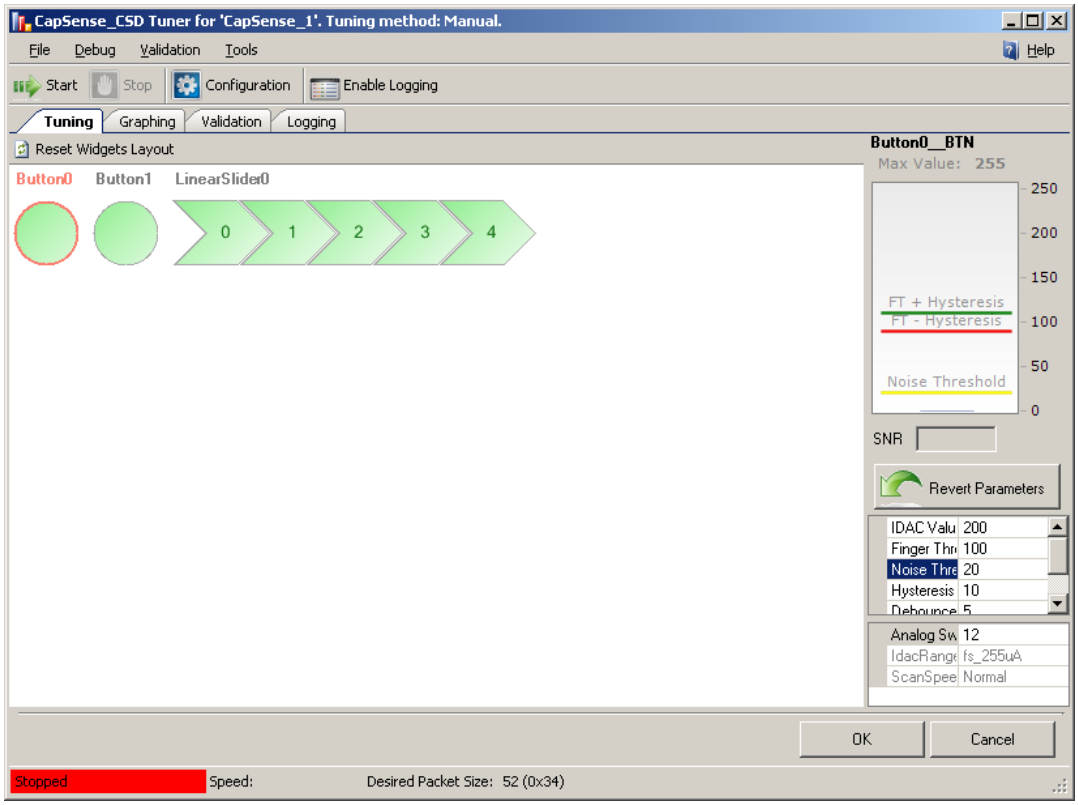
プロジェクトを準備して、PSoC デバイスのプログラミングを実施してください

PSoC Creator ヘルプを必要に応じて参照してください。

チューナー アプリケーションの起動

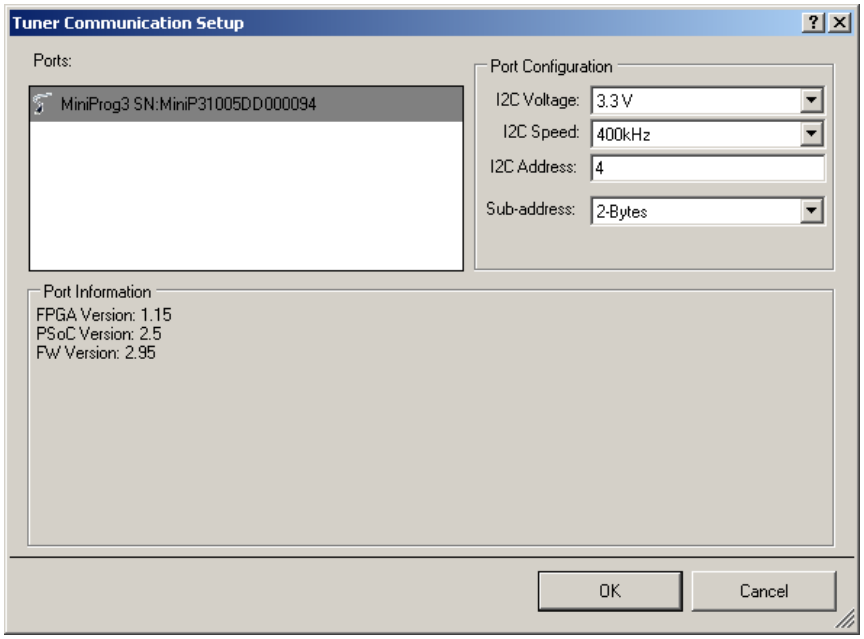
- CapSense CSD コンポーネント アイコンを右クリックし、コンテキスト メニューから **Launch Tuner** を選択します。

  チューナー アプリケーションが開きます。



通信パラメータの設定

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



2. 通信パラメータを設定して、**OK** をクリックします。

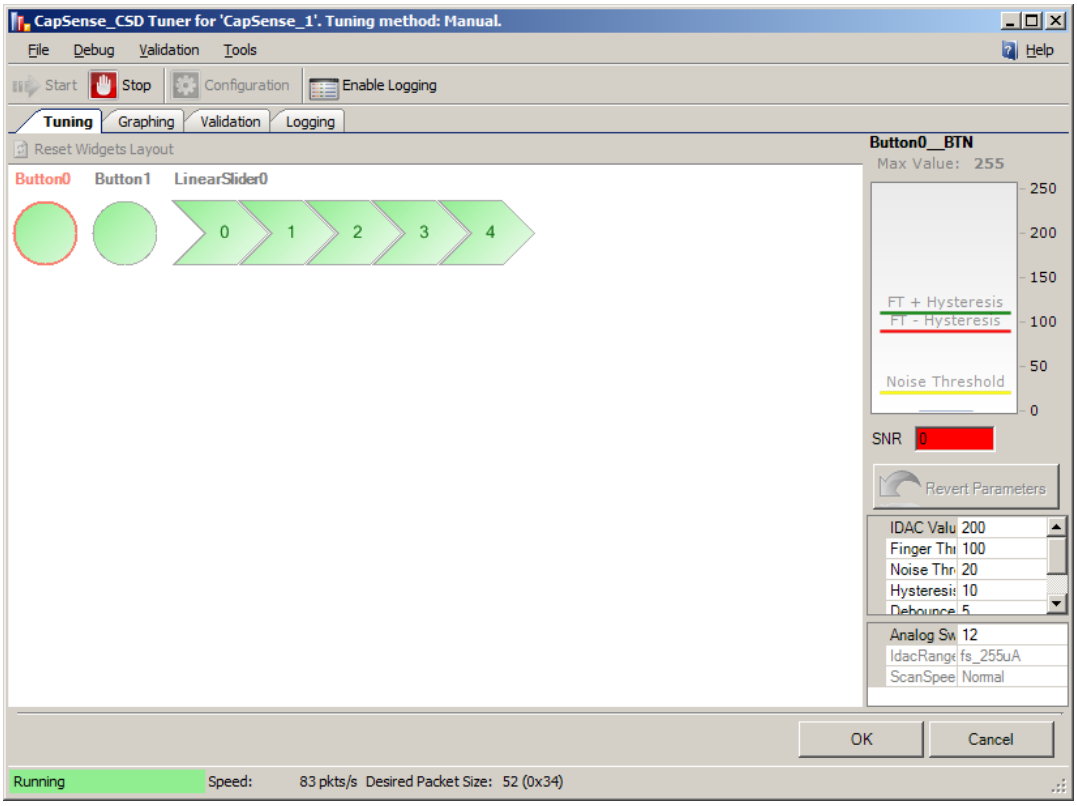
重要:プロパティは EZI2C コンポーネントと同一のプロパティにする必要があります。**I2C Bus Speed**、**I2C Address**、**Sub-address** = 2 バイト。

チューニングの開始

- チューニング GUI の **Start** をクリックします。CapSense エレメントすべてが開始し、値が表示されます。

CapSense パラメータ値の編集

- エレメントの 1 つについてパラメータ値を編集して **[Enter]** キーを押すか、別のオプションに移動すると、パラメータ値が自動的に適用されます。GUI は引き続きスキャン データを表示しますが、更新したパラメータを使用したアプリケーションに基づいて変更されます。このデータシートの後半の「チューナー GUI インタフェース」セクションを参照してください。



必要に応じて繰り返し

- チューニングが終了し、CapSense コンポーネントで信頼性の高いタッチ センサ結果を得られるまで、必要に応じてステップを繰り返します。

チューナー アプリケーションを閉じます

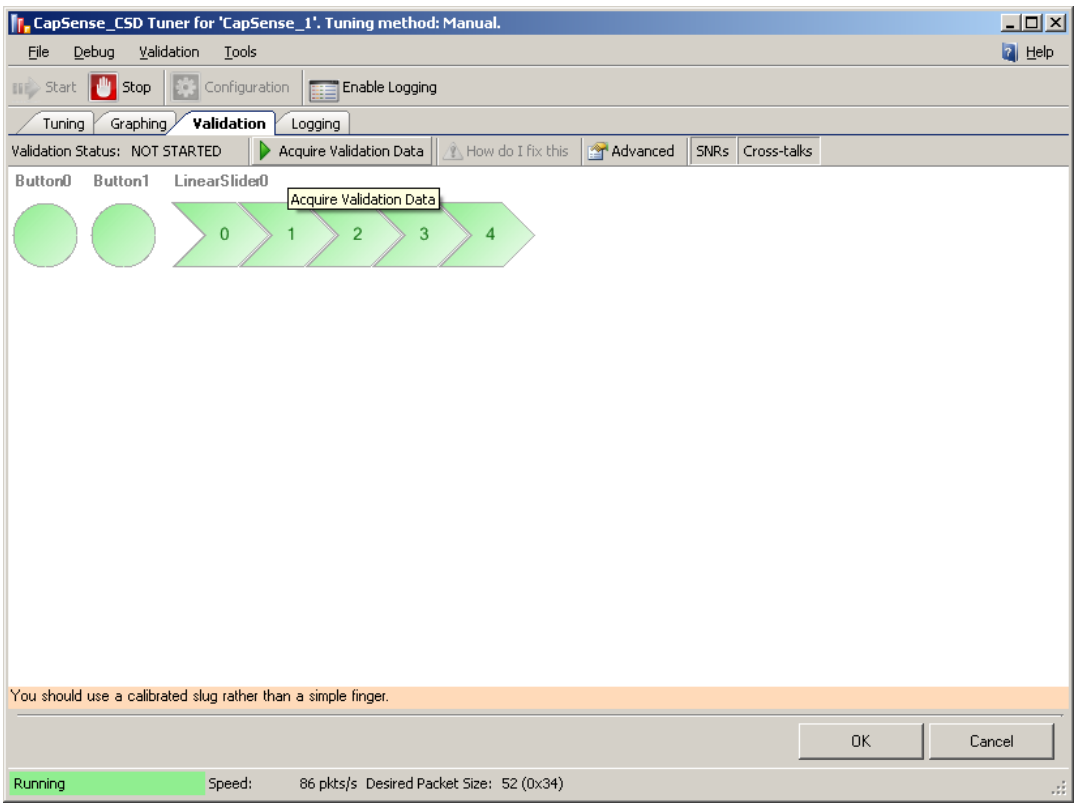
- **OK** をクリックすると、パラメータが CapSense_CSD インスタンスに書き込まれます。チューナーアプリケーションダイアログが閉じます。

CapSense バリデーション プロセス

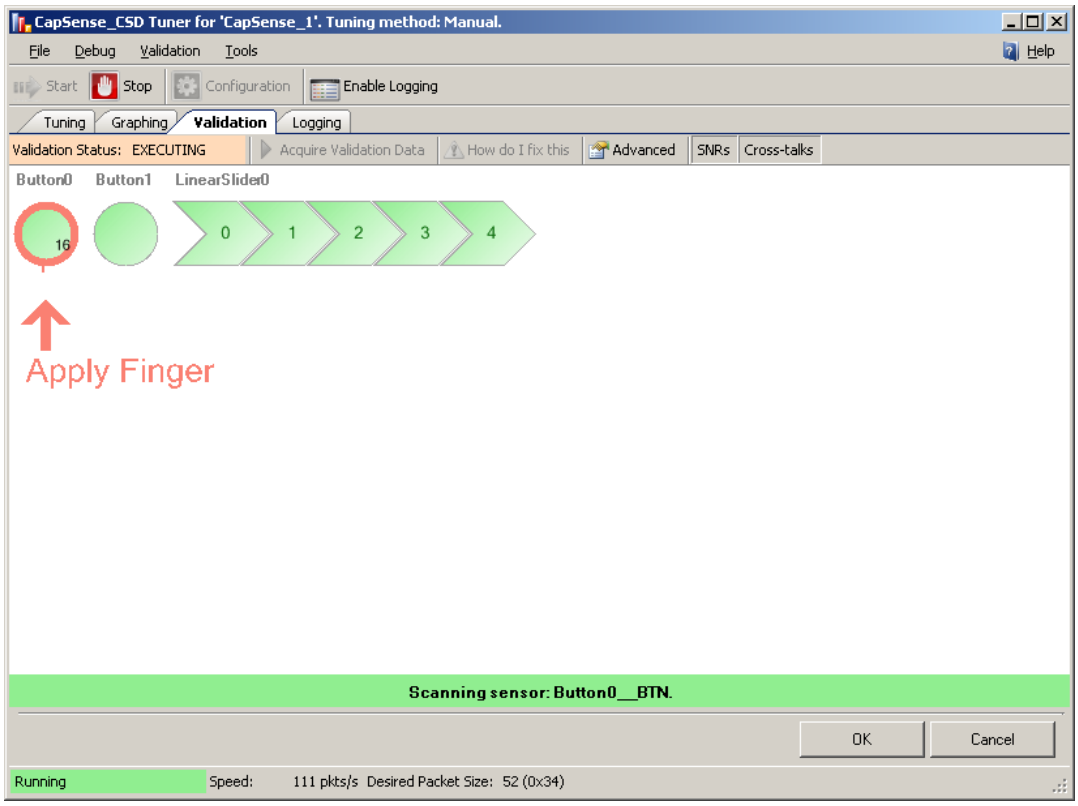
バリデーション メカニズムにより、ボードが十分にチューニングされたかどうかを決定します。以下は、チューナーバリデーション機能を使用して CapSense デザインの妥当性を確認する標準的なプロセスです。

バリデーションの開始

1. スキャン プロセスを開始する前に、チューナーおよびハードウェアの準備ができている必要があります。前の セクション、CapSense チューニングのプロセス、を参照して、システム スキャンの準備をします。
2. **Validation** タブで、**Acquire Validation Data** をクリックします。すべての CapSense エレメントについて値の表示が開始されます。



スティミュレーション センサ

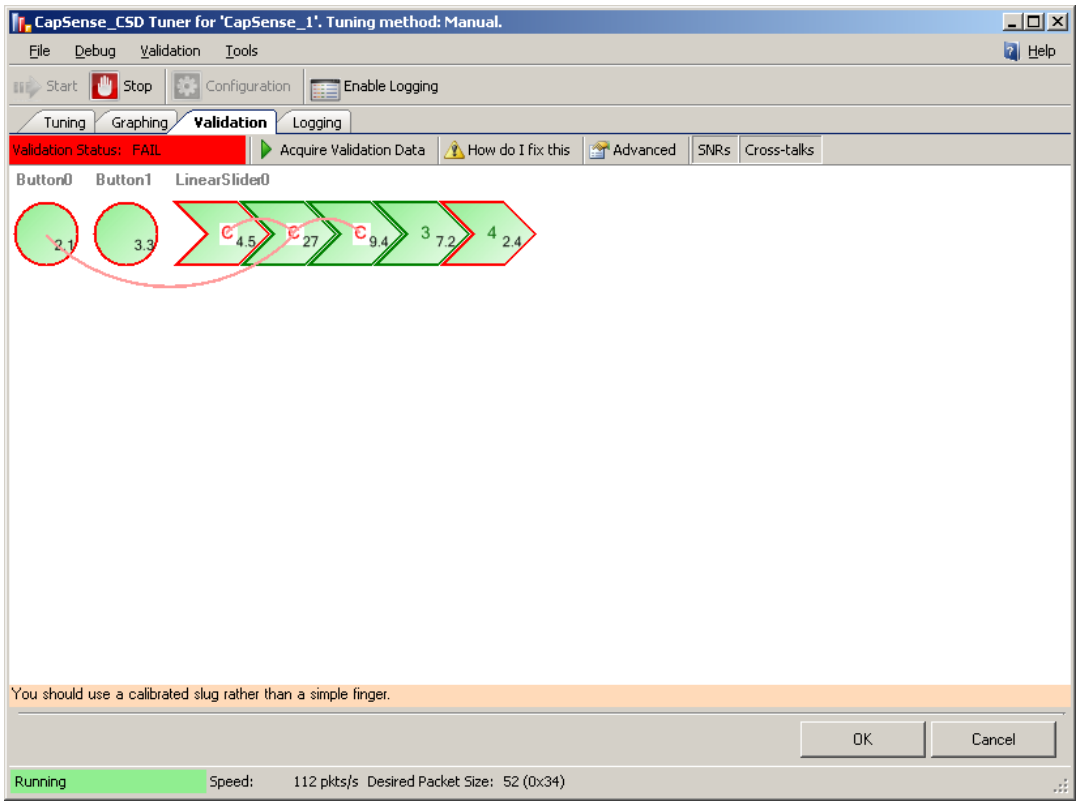


各センサに指を当てるよう、プロンプト表示されます。CapSense エレメントを押すようにプロンプト表示されるたびに、赤い矢印が点滅して、**PRESS HERE** というテキストがレイアウト上のターゲットを示します。テキストはチューナー下に表示され、バリデーション プロセスをガイドします。

- 現在のセンサのスキャンを開始するには、キーボードの任意のキーを押します。

  センサを刺激するには、指で押す代わりに、校正されたスラグの使用を推奨します。

バリデーションの表示



SNR 警告が以下のように表示されます:

- **赤で点滅**、S/N 比が **Sufficient Value** 未満の CapSense センサの周囲が強調表示されます。
- **黄色で点滅**、S/N 比が **Sufficient Value** および **Optimal Value** の間である CapSense センサの周囲が強調表示されます。
- **緑で点灯**、S/N 比が **Optimal Value** を上回っている CapSense センサの周囲が強調表示されます。

クロストーク効果の警告は以下のように表示されます。

- **Individual Crosstalk Check**。バリデーション プロセスの間、刺激するように指示が出された以外のすべてのエレメントが、ソフトウェアにより監視されます。あるエレメントが異なるカウントで、**Crosstalk Threshold Percentage**(直接刺激されなかった場合)を超える値になった場合、クロストーク警告が生成されます。望ましくないカウントを示すエレメントと刺激されたエレメントの間の線が点滅表示されます。
- **Worst Case Crosstalk Check**。個別のクロストーク確認がなされると、ソフトウェアは異なるカウント測定値を記録します。プロセスが終了すると、最悪の場合のクロストーク予測がなされます。

  各センサについて要約が表示されます。これは、**Worst Case Crosstalk Sensor Count** と等しいクロストーク効果の数です。最大のクロストーク値は、要約の最初の値であり、2 つ目に大きい値は 2 つ目に表示

されます。例：以下のクロストークカウント(1,5,3,2,4,1,1,0)があり、**Worst Case Crosstalk Sensor Count** が 2 の場合、**Worst Case Crosstalk** の計算は(5 + 4 = 9)となります。

この値が **Worst Case Crosstalk Threshold** を超える場合、センサ表示の中央に**「C」の文字が点滅**します。

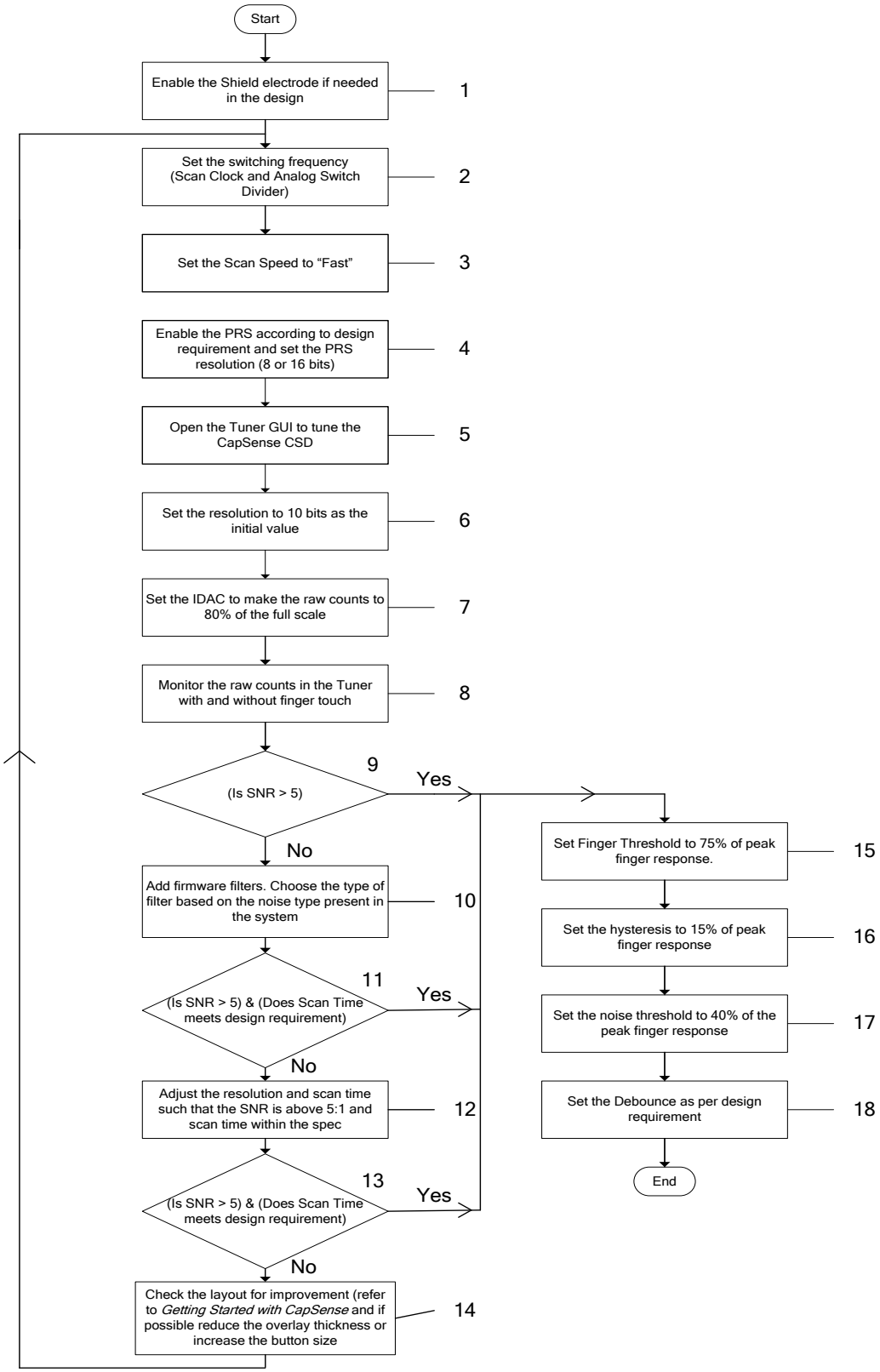
バリデーションの結果

バリデーションで障害がカバーできない場合、**Validation Report** が表示されます。このレポートには以下の情報が含まれます。

- **Optimal Value** 未満の SNR 値
- **Sufficient Value** 未満の SNR 値
- 最悪のケースのクロストーク障害のシグナル。およびその場合にはクロストーク数

Validation Report を開くには、**Validation** タブの **How do I fix this** ボタンをクリックすることもできます。

マニュアルチューニングのプロセス



1. 設計要件により、シールドはイネーブルまたはディスエーブルになっています。センサのオーバーレイが濡れる可能性があるアプリケーションの場合(AN2398 を参照)、EMI に対するシールド、または過剰に高い Cp の緩和には、シールドが役立ちます。電流モードが **IDAC Sourcing** に設定されており、リファレンス電圧が **Vref 1.024V** の場合、シールド信号は SIO ピンに接続する必要があります。それ以外の場合は、シールドを任意のピンにルーティングできます。詳細情報については、この文書の後半の「シールド電極」セクションを参照してください。

2. 静電容量式センサがいずれも完全に充電および放電されるように、切り替え周波数を設定してください。**Scan Clock** および **Analog Switch Divider** により、センサコンデンサが切り替えられる周波数を決定します。**Scan Clock** は CapSense コンポーネントの 1 つ目のクロック源です。**Analog Switch Divider**(プリスケーラ)は、スキャンクロックを分割し、切り替えクロックを作成します。これを使用して各センサの充電および放電が行なわれます。センサが完全に充電および放電されない場合は、**Analog Switch Divider** を上昇させて、切り替え周波数を低下させます。

   センサが完全に充電および放電しているかテストするには、各センサのピンをプローブします。注、センサコンデンサの電圧をオシロスコーププローブで観測している間に、センサの寄生容量にプローブ静電容量が追加されます。プローブを 10 倍モードで使用すると、プローブの静電容量が低下します。使用可能な場合は、FET 入力プローブを使用してください。センサが完全に充電および放電されていることを確認します。完全に充電および放電されていない場合は、コンポーネント設定で **Analog Switch Divider** 値を上昇します。このパラメータはチューナー GUI で変更できないため、コンポーネント設定を行なう必要があります。そのため、この値がコンポーネントの **Configure** ダイアログで変更されると、デバイスでプロジェクトを再びビルドし、プログラムする必要があります。

3. **Scan Speed** の初期値は **Fast** に設定されています。信号対ノイズ比(SNR-S/N 比)またはスキャン時間の要件に適合しない場合は、この値を事後に変更することができます。

4. CapSense の外部 EMI の影響を低減し、センサのスキャン エミッションを低減するために、PRS は初期設定でイネーブルになっています。EMI の影響を受けやすい設計ではイネーブルにします。PRS の分解能は 8 ビットまたは 16 ビットです。これはスキャン時間に依存します。スキャン時間が長い場合 PRS 16 を、スキャン時間が短い場合は PRS 8 を使用します。

5. CapSense チューナー GUI を開きます。

6. 分解能を 10 に設定します。

   分解能およびスキャン速度を上昇すると、感度が向上しますが、スキャン時間も上昇します。そのため、スキャン時間と感度はトレードオフの関係にあります。

   分解能 10 はチューニングプロセスの開始値として適切ですが、1 mm 未満の薄いオーバーレイを使用した設計では、分解能 8 および 9 などの低めの値も初期値として使用できます。

7. GUI の IDAC 値を変更して、Raw カウントがフルスケール値の 80 % に到達するようにします。フルスケール値は 2^ 分解能です。IDAC 値を低減すると Raw カウントが上昇し、逆の場合もまた同様になる点に注意してください。IDAC 値を変えても 80 % を達成することが不可能であれば、コンポーネント設定で IDAC 範囲を変更してください。IDAC 範囲はチューナー GUI では変更できません。コンポーネントの **Configure**

ダイアログで変更してください。値が **Configure** ダイアログで変更されたら、プロジェクトを再びビルドして、デバイスにプログラムしてください。

8. 指の存在がある場合とない場合で、Raw カウントを監視します。ピークツーピーク ノイズとピークの指の応答に注意します。SNR(S/N 比) を次のように計算します。

$$SNR = \frac{\text{Peak Finger Response}}{\text{Peak to Peak Noise when finger is not present}}$$

9. CapSense を適切に設計するには、S/N 比は 5 を超える値にしてください。合計スキャン時間が設計に適しているか確認します。S/N 比 要件に適合しない場合、ファームウェア フィルタを追加します。フィルタ セクションを参照し、システムに存在しているノイズに適したフィルタの種類を選択します。ほとんどのデザインでは、First Order IIR 1/4 フィルタから開始します。これは、最小限の SRAM で高速の応答が得られるためです。

10. ステップ 8 と同様、S/N 比を確認します。また、スキャン時間が設計要件に適合しているか確認します。GUI 内でチューニングした値は、GUI で **OK** ボタンをクリックすると更新されます。パラメータの設定に基づいて、コンポーネントによりスキャン時間の概数が計算されます。スキャン時間は、スキャン順序のタブに表示されます。高分解能でスキャン速度が低速のセンサがデザインに多数含まれる場合、すべてのセンサの合計スキャン時間は、結果として、センサ間のスキャン間隔が長くなります。

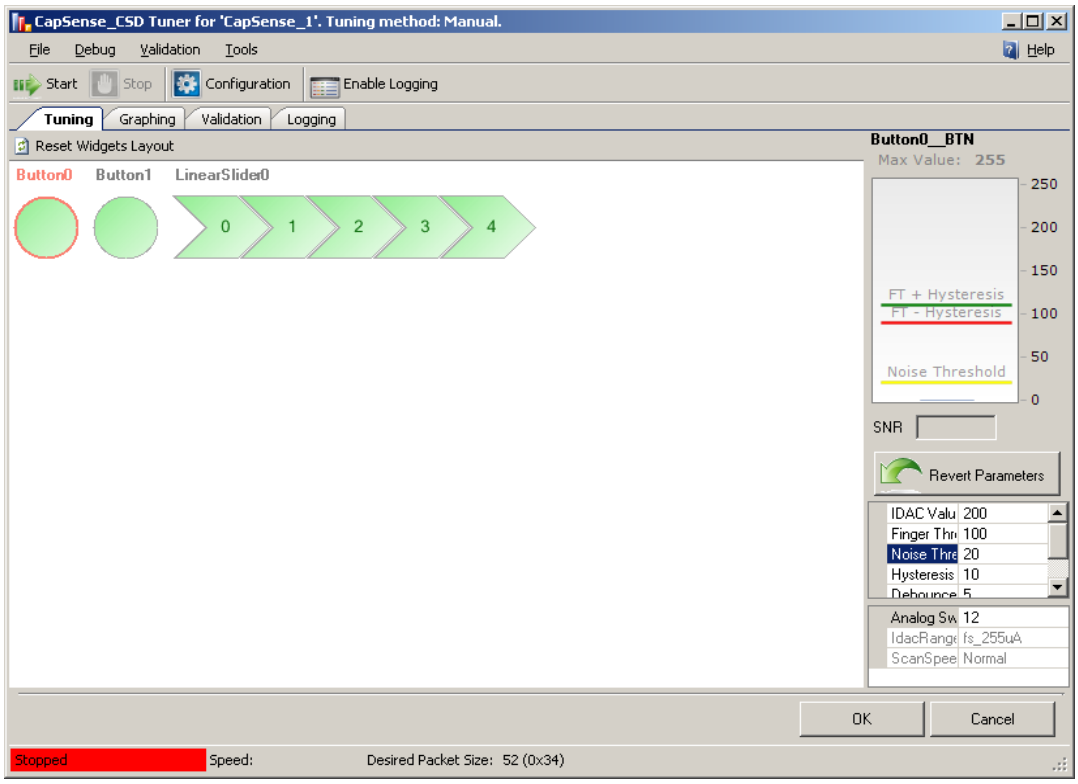
    S/N 比が 5 未満の場合、分解能、スキャン速度、あるいは両方を上昇します。これによってスキャン時間が上昇します。そのため、S/N 比が 5 を超える値にし、スキャン時間を設計仕様未満に保つには、分解能とスキャン時間の両方をチューニングしてください。S/N 比およびスキャンを再確認します。S/N 比を 5:1 にし、スキャン時間を設計仕様以内に保てない場合、PCB レイアウトまたはオーバーレイ設計の改善を試みてください。「*CapSense の導入*」で PCB デザインのガイドラインを参照してください。また、オーバーレイの厚さを低減するか、ボタンの直径を大きくすることで、感度を上昇させることができます。

11. S/N 比が 5:1 に達したら、以下のファームウェアパラメータを設定します。

    - ❑ Finger Threshold は、センサがアクティブかどうかを決定するために、ファームウェアが閾値として使用するパラメータです。このパラメータをピークの指の応答の 75% にセットします。
    - ❑ ヒステリシスをピークの指の応答の 15% にセットします。
    - ❑ ノイズ閾値を指の応答の 40% にセットします。
    - ❑ デバウンスは、ESD イベントのように高周波数、高い振幅のノイズでも、ボタンが起動されないようにします。デバウンスの値は 1 または 2 などの小さい数にしてください。これは、スパイクや高周波数ノイズによりボタンのタッチが誤ってトリガされるが、2 つのスキャンの長さほど広くないためです。高速スキャン設計では、デバウンスの値は 5 などの高い値に設定してください。

チューナー GUI インタフェース

汎用インタフェース



トップ パネルボタンは以下の通りです:

- **Start**(またはメインメニュー項目の **Debug > Start**) – チップからのデータの読み込みと表示が開始されます。設定されている場合、グラフおよびログも開始されます。
- **Stop**(またはメインメニュー項目の **Debug > Stop**) – チップからのデータの読み込みと表示が停止します。
- **Configuration**(またはメインメニュー項目の **Debug > Configuration**) – **Communication Configuration** ダイアログが開きます。
- **Enable Logging** (またはメインメニュー項目の **Debug > Start**) – デバイスから受信したデータのログ 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 チューニングファイルから設定をインポートし、チューナーにすべてのデータをロードします。
- **File > Help** – ヘルプファイルが開きます。

その他の項目により、トップおよびボトムのパネルボタンの機能が 2 倍になります。

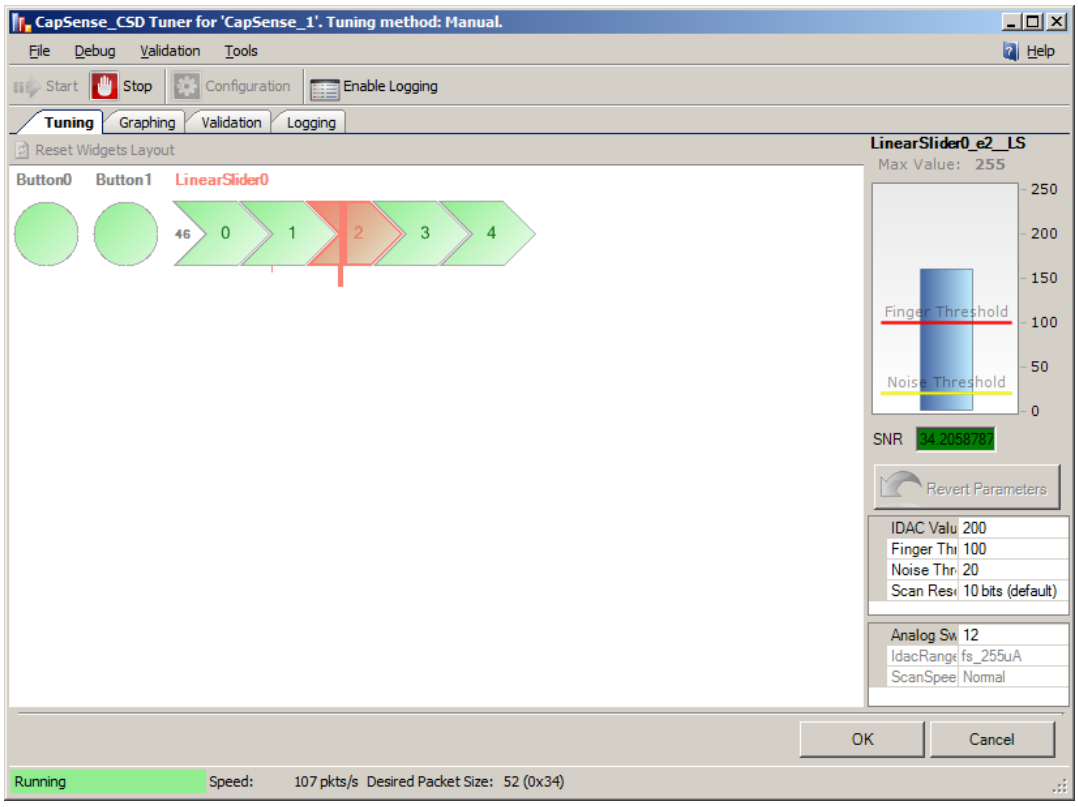
タブ:

- **Tuning** – コンポーネントウィジェットすべてがワークスペースで構成した通りに表示されます。これにより、物理的 PCB またはエンクロージャと同様にウィジェットを配列することができます。このタブは、ウィジェットパラメータのチューニングおよびウィジェットのデータと状態の表示に使用します。
- **Graphing** – 個別のウィジェットデータを図で詳細表示します。
- **Logging** – ログデータ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項目の **File > Apply Changes and Close**) – パラメータの現在の値を CapSense コンポーネントインスタンスにコミットし、GUI を閉じます。
- **Cancel** (またはメインメニュー項目の **File > Exit**) – パラメータの値をコンポーネントインスタンスにコミットせず、GUI を閉じます。

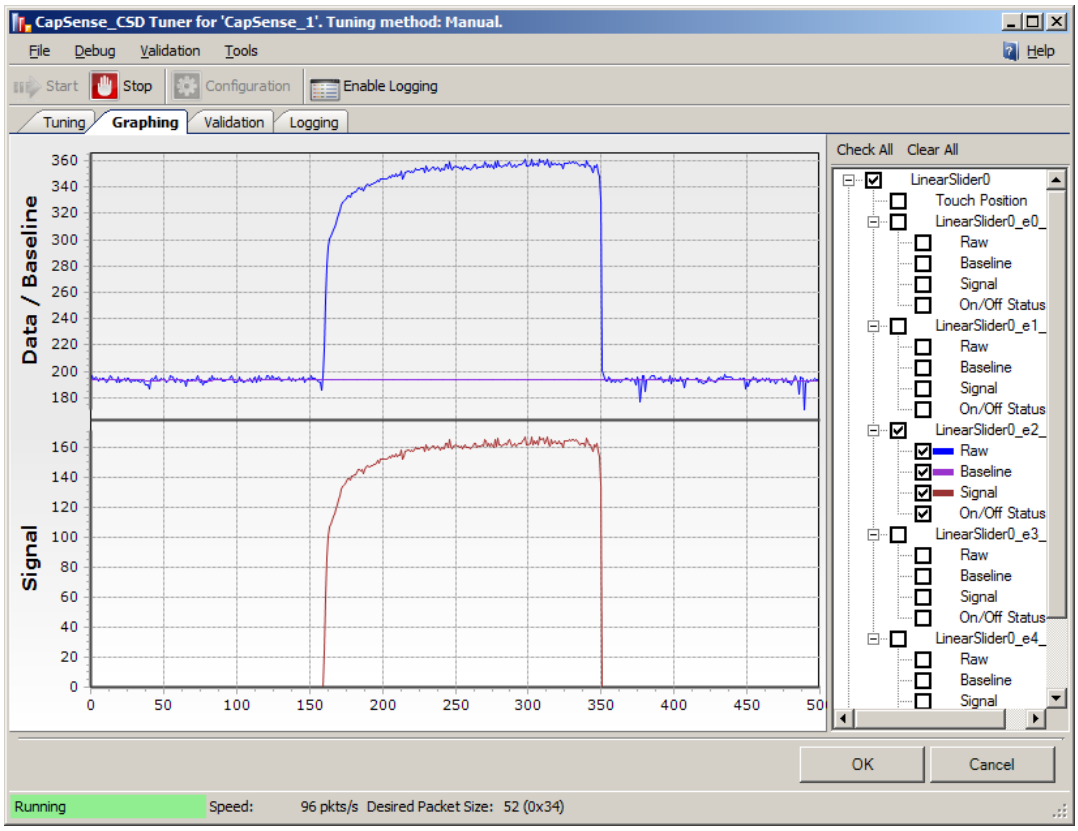
Tuning タブ



- **Widgets schematic** – 設定されたウィジェットすべてのグラフ表示が含まれます。ウィジェットが 2 つ以上のセンサで構成される場合、個別のセンサを選択して、詳細分析することができます。回路図で各ウィジェットを移動できます。

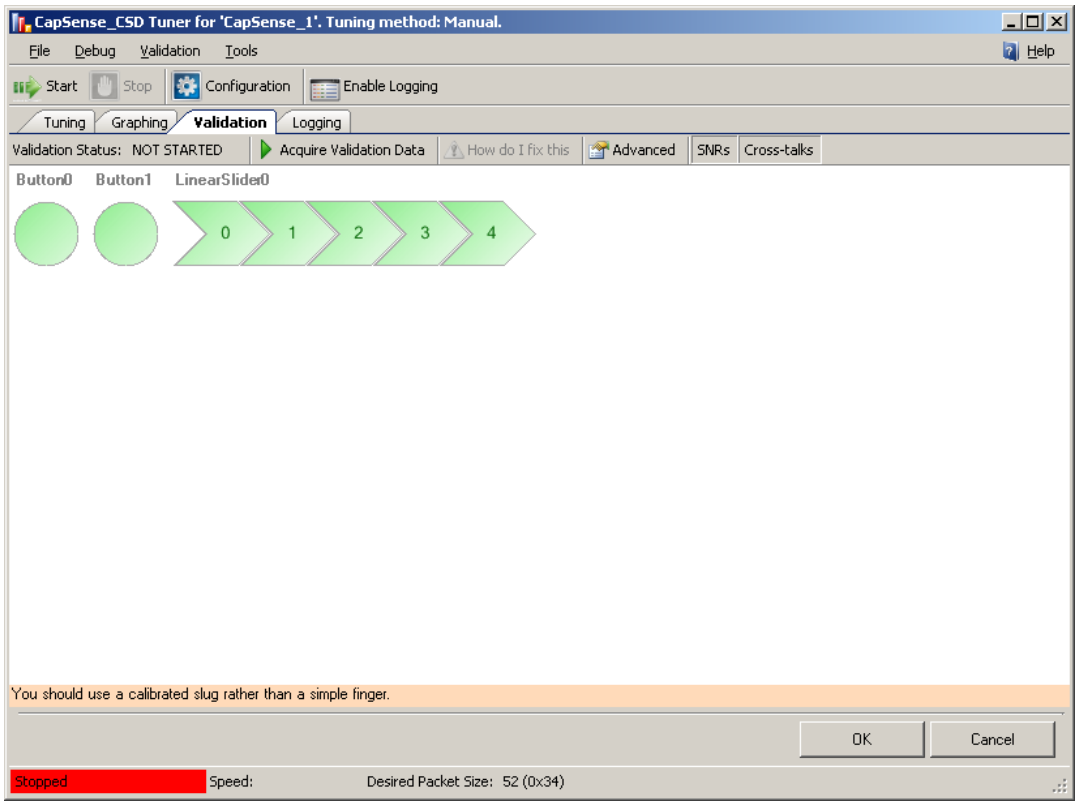
- **Reset Widget Layout** ボタン – ウィジェットを回路図内の初期設定位置に移動します。
- **Bar graph** – 選択したセンサの信号値を表示します。
  - 詳細表示した棒グラフの最大スケールは Max Value ラベルをダブルクリックして、調整することができます。有効な範囲は 1～255 で、初期設定は **255** です。
  - 現在の指がオンにする閾値は、棒グラフに**緑色の線**で表示されます。
  - 現在の指がオフにする閾値は、棒グラフに**赤色の線**で表示されます。
  - 現在のノイズ閾値は、棒グラフに**黄色の線**で表示されます。
- **SNR** (S/N 比)-選択したセンサについて、信号対ノイズ比がリアルタイムで計算されます。S/N 比の値が 5 未満では不適切で、赤色表示されます。5～10 は下限値で黄色、10 を超える値は適切であり、緑色表示されます。SNR(S/N 比)値は、以前に受信したデータに基づいて計算されます。
- **Revert Parameters** ボタン – パラメータを初期値にリセットし、これらの値をチップに送ります。初期値は GUI を起動したときに表示されていた値です。
- **Sensor properties** – ウィジェットの種類に基づき、選択したセンサのプロパティを表示します。右側のサイド パネルに配置されています。
- **General CapSense properties**(読み取り専用) – ランタイム中に変更できない 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。これらは参照目的のみです。この情報は右側のパネルの下部に表示されます。
- **Widget controls context menu**(この機能は GUI のウィジェット制御にのみ適用されます):
  - **Send To Back** – ウィジェット制御を表示の背面に送ります。
  - **Bring To Front** – ウィジェット制御を表示の正面に配置します。
  - **Rotate Clockwise 90** – ウィジェット制御を時計回りに 90 度回転します。(リニア スライダのみ)
  - **Rotate Counter Clockwise 90** – ウィジェット制御を反時計回りに 90 度回転します。(リニア スライダのみ)
  - **Flip Sensors** – センサの順序を反対にします。(リニア スライダおよびラジアル スライダのみ)
  - **Flip Column Sensors** – 列センサの順序を反対にします。(タッチパッドおよびマトリックス ボタンのみ)
  - **Flip Row Sensors** – 行センサの順序を反対にします。(タッチパッドおよびマトリックス ボタンのみ)
  - **Exchange Columns and Rows** – 列センサが行になり、行センサが列になります。(タッチパッドおよびマトリックス ボタンのみ)

Graphing タブ



- **図領域** – ツリー表示から選択した項目について、図を表示します。メニュー項目 **Export to .jpg** を右クリックすると、図領域のスクリーンショットが生成され、.jpg ファイルとして保存されます。
- **Tree view** – ウィジェットとセンサのデータの組み合わせすべてを図で表示し、ログ機能がイネーブルになっている場合はファイルに記録します。On/Off Status データ値はログされるだけで、図には表示されません。

Validation タブ



Validation は診断専用です。このタブにはウィジェット レイアウト表示が含まれますが、レイアウトを編集することはできません。このレイアウト部分は表示専用として使用されます。

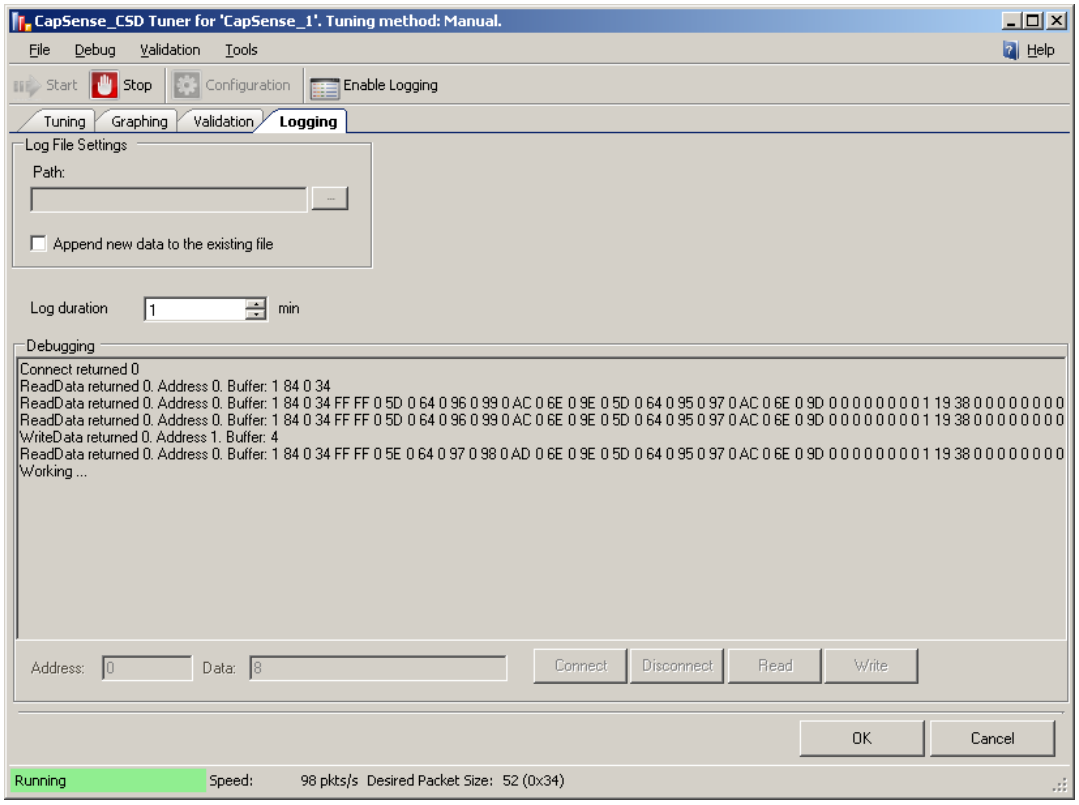
- **Widgets schematic** – 設定されたウィジェットすべてのグラフ表示が含まれます。

トップ パネル コントロール:

- **Validation Status** ラベル –バリデーションの状況を表示します。以下のメッセージがあります。
  - **VALIDATION NOT STARTED** – デザインが最後に変更されて以来、バリデーション プロセスは実行されていません。
  - **PASS** – 完全なバリデーション プロセスが障害なく終了しました。
  - **FAIL** – バリデーションでカバーできない障害がありました。バリデーション レポートが表示されます。
- **Acquire Validation Data** ボタン(またはメイン メニュー項目の **Validation > Acquire Validation Data**) – バリデーション プロセスを開始します。このプロセスによりオペレーション順序がガイドされ、各センサに順番に指を当てるようにプロンプト表示されます。
- **How do I fix this** ボタン – バリデーションに合格しなかったセンサに推奨される修正方法を含むレポートが開きます。このボタンは、バリデーション プロセスが過去に完了しており、デザイン エラーが検出された場合にのみ使用できます。

- **Advanced ボタン**(またはメインメニュー項目の **Validation > Validation Advanced properties**)– バリデーション プロパティのプロパティ ウィンドウを開きます(詳細情報は「バリデーションの高度なプロパティ」を参照してください)。
- **SNR** ボタン – ウィジェット回路図で、S/N 比 表示をオンまたはオフにします(詳細情報は「バリデーションの表示」を参照してください)。
- **Crosstalks** ボタン – ウィジェットで、回路図のクロストーク表示をオンまたはオフにします(詳細情報は「バリデーションの表示」を参照してください)。

Logging タブ



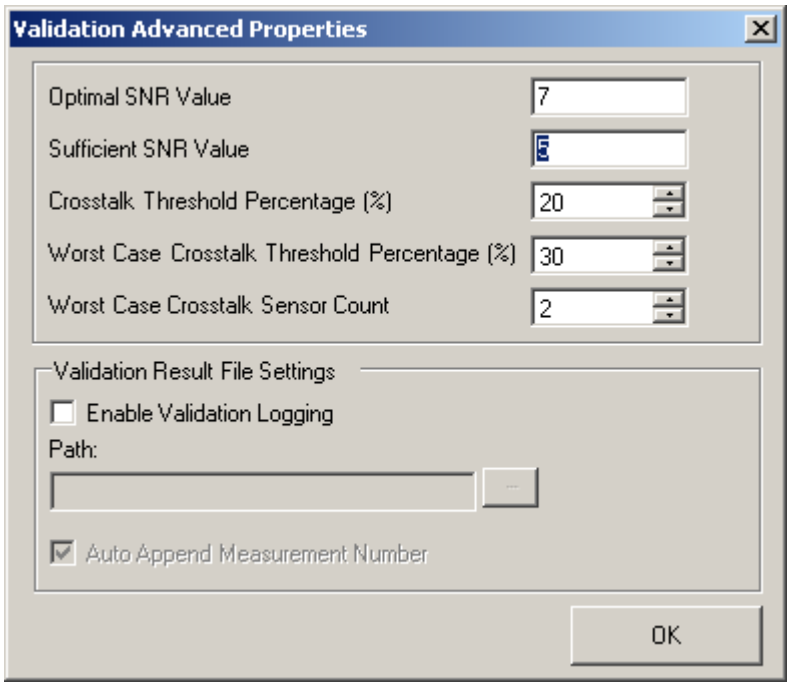
- **Graphing** タブのツリー表示のチェックボックスを選択することで、ログに記録されるデータが示されます。
- **Path** – ログファイルのパスを定義します(ファイル拡張子 .csv)。
- **Append new data to existing file** チェックボックス – 選択すると、既存のファイルに新しいデータを付記します。選択しないと、ファイルから古いデータが消去され、新しいデータに置き換えられます。
- **Log duration** – ログの長さを分単位で定義します。有効な範囲は 1～480 で、初期値は **255** です。

デバッググループ

この機能はデバッグ目的でのみ存在します。この機能により、チューナーの通信エラーの調査を手助けします。

- デバッグログウィンドウ – チューナーが実行する通信コマンドを表示します。すべての通信エラーはここに記録されます。チューナーが正常に開始すると、最初の通信コマンド数行のみが記録されます。
- **Connect** – PSoC デバイスに接続します。
- **Disconnect** – PSoC デバイスから切断します。
- **Address** – PSoC デバイスのアドレスを指定します。
- **Read** – PSoC デバイスからデータを読み取ります。アドレスフィールドはバッファのアドレスを定義します。データフィールドは読み取られるバイト数を定義します。
- **Write** – 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のアドレスを定義します。データ フィールドで書き込むデータを定義します。

バリデーションの高度なプロパティ



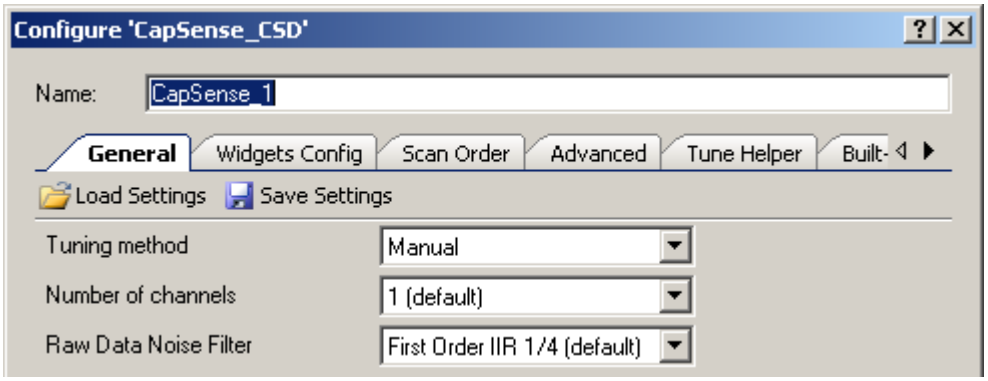
- **Optimal SNR Value** – 最適な SNR(S/N 比)値を定義します。有効な範囲は 0～100 で、初期値は **7** です。
- **Sufficient SNR Value** – 十分な SNR(S/N 比)値を定義します。有効な範囲は 0～100 で、初期値は **5** です。

- **Crosstalk Threshold Percentage (%)**– クロストーク閾値を、各センサについて指の閾値のパーセントで定義します。有効な範囲は 0～100 パーセントで、初期値は **20** です。
- **Worst Case Crosstalk Threshold Percentage (%)** – 最悪のケースのクロストーク閾値を、最悪のケースのクロストークのパーセントで表示します。有効な範囲は 0～100 パーセントで、初期値は **30** です。
- **Worst Case Crosstalk Sensor Count** – 最悪のケースのクロストークを計算するのに使用するセンサの数を定義します。有効な範囲は 0～100 で、初期値は **2** です。
- **Enable Validation Logging** – バリデーション データのログをイネーブルにします。
- **Path** – バリデーション データのログ ファイルのパスを定義します(ファイル名の拡張子 .csv)。
- **Auto Append Measurement Number** チェックボックス – 選択した場合、バリデーション プロセスが開始するごとに、ログファイル名がインクリメントし(例 : “validation001.csv”)、データは新しいファイルに保存されます。

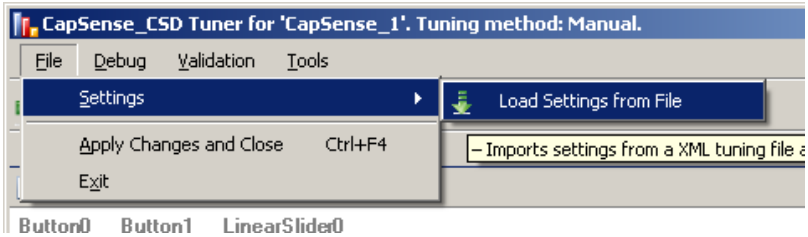
保存/ロード設定機能

チューナー GUI は、スタンドアロンのアプリケーションとしても開けます。その場合は、CapSense CSD コンポーネントチューナー GUI の保存およびロード設定を使用してください。

1. カスタマイザの **Save Settings** ボタンをクリックします。



2. **Save File** ダイアログボックスで、ファイル名と保存する場所を指定します。
3. チューナーウィンドウを開き、**File > Settings > Load Settings from File** をクリックします。



4. **File Open** ダイアログボックスで、コンポーネント設定を前に保存したファイルを示します。設定が自動的にチューナーにロードされます。

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。以下の表では、各関数の概要を説明しています。続くセクションでは、各関数について詳しく説明します。

初期設定では、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。コンポーネントの名称は、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。理解しやすいように、次の表では、インスタンス名「CapSense」を使用しています。

汎用 API

これらは汎用の CapSense API 関数で、コンポーネントのオペレーションを開始したり停止します:

関数	説明
CapSense_Start()	コンポーネントを起動する際の推奨される手法です。レジスタを初期化し、CapSense 内で使用するサブコンポーネントのアクティブモード電源テンプレートのビットをイネーブルにします。
CapSense_Stop()	コンポーネントの中断をディスエーブルにし、CapSense_ClearSensors()を呼び出して、すべてのセンサを非アクティブ状態にリセットします。
CapSense_Sleep()	デバイスが低電力モードに入れるように、コンポーネントを準備します。CapSense 内で使用する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の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、すべてのセンサを非アクティブ状態にリセットします。
CapSense_Wakeup()	デバイスが低電力モード、スリープモードからウェイクした後、CapSense 設定およびノンリテンション レジスタの値を復元します。
CapSense_Init()	カスタマイザで提供されている初期設定の CapSense 設定を初期化します。
CapSense_Enable()	CapSense 内で使用するサブコンポーネントのアクティブモード電源テンプレートの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。すべ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 設定およびノンリテンション レジスタ値を復元します。

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()がCapSense_Init()関数を呼び出し、次に CapSense_Enable()関数を呼び出します。レジスタを初期化し、CapSense コンポーネントの CSD 手法を開始します。すべてのセンサを非アクティブ状態にリセットします。センサのスキャン中断をイネーブルにします。SmartSense チューニングモードが選択されている場合、このチューニング手順がすべてのセンサに適用されます。他の API ルーチンよりも前に CapSense_Start()ルーチンを呼び出す必要があります。

引数: なし

戻り値: なし

副作用: なし

void CapSense_Stop(void)

説明: センサのスキャンを停止し、コンポーネントの中断をディスエーブルにし、すべてのセンサを非アクティブ状態にリセットします。CapSense 内で使用するサブコンポーネントのアクティブモード電源テンプレートのビットをディスエーブルにします。

引数: なし

戻り値: なし

副作用: この関数は、すべてのスキャンが完了した後で呼び出してください。

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ための推奨手法です。CapSense 内で使用するサブコンポーネントのアクティブモード電源テンプレートのビットをディスエーブルにします。Calls CapSense_SaveConfig()関数でカスタマ 設定の CapSense ノンリテンション レジスタを保存し、すべてのセンサを非アクティブ状態にリセットします。

引数: なし

戻り値: なし

副作用: この関数は、スキャンが完了した後で呼び出してください。

この関数では CapSense コンポーネントで使用したピンを最低消費電力状態にはできません。ピンのドライブモードを変更するには、「ピンAPI」セクションに記載されている関数を使用します。

void CapSense_Wakeup(void)

説明: CapSense 設定およびノンリテンション レジスタ値を復元します。CapSense 内で使用するサブコンポーネント用にアクティブモード電源テンプレートのビットを設定して、イネーブル状態を復元します。

引数: なし

戻り値: なし

副作用: この関数では、CapSense コンポーネントで使用するピンは元の状態には復元されません。

void CapSense_Init(void)

説明: カスタマイザで提供され、コンポーネント オペレーションを定義する初期設定の CapSense 設定を初期化します。すべてのセンサを非アクティブ状態にリセットします。

引数: なし

戻り値: なし

副作用: なし

void CapSense_Enable(void)

説明: CapSense 内で使用するサブコンポーネントのアクティブモード電源テンプレートのビットをイネーブルにします。

引数: 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。すべてのセンサを非アクティブ状態にリセットします。

引数: なし

戻り値: なし

副作用: この関数は、スキャンが完了した後で呼び出してください。

この関数では CapSense コンポーネントで使用したピンを最低消費電力状態にはできません。ピンのドライブモードを変更するには、「ピンAPI」セクションに記載されている関数を使用します。

void CapSense_RestoreConfig(void)

説明: CapSense 設定およびノンリテンション レジスタを復元します。

引数: なし

戻り値: なし

副作用: この関数は、スキャンが完了した後で呼び出してください。
この関数では、CapSense コンポーネントで使用するピンは元の状態には復元されません。

スキャン固有の API

これらの API 関数は CapSense センサのスキャンを実装するのに使用します。

関数	説明
CapSense_ScanSensor()	スキャン設定を行い、センサ 1 つまたは各チャンネルのセンサを組み合わせたグループのスキャンを開始します。
CapSense_ScanEnabledWidgets()	推奨されるスキャン手法です。イネーブルになっているウィジェットすべてをスキャンします。
CapSense_IsBusy()	センサスキャンの状態を返します。
CapSense_SetScanSlotSettings()	選択したスキャンスロットのスキャン設定を行ないます(センサまたはセンサペア)。
CapSense_ClearSensors()	すべてのセンサをノンサンプリング状態にリセットします。
CapSense_EnableSensor()	次のスキャン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。
CapSense_DisableSensor()	選択したセンサをディスエーブルにし、次のスキャンサイクルでスキャンされないようにします。
CapSense_ReadSensorRaw()	センサの Raw データを CapSense_SensorResult[ ] アレイから返します。
CapSense_SetRBleed()	複数のブリード抵抗が使用されている場合、ピンを設定してブリード抵抗(Rb)接続で使用できるようにします。

void CapSense_ScanSensor(uint8 sensor)

説明: スキャン設定を行い、センサ 1 つまたは各チャンネルのペアセンサのスキャンを開始します。2 つのチャンネルで構成されている場合、2 つのセンサを同時にスキャンできます。スキャンが完了したら、ISR は測定したセンサの Raw データをグローバル Raw センサアレイにコピーします。IRS の使用により、この関数はブロックされなくなります。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 カスタマイザによって順番に割り当てられるものです。

引数: uint8 sensor: センサ番号

戻り値: なし

副作用: なし

void CapSense_ScanEnabledWidgets(void)

説明: これは、イネーブルになっているウィジェットすべてをスキャンするのに推奨される手法です。イネーブルになっているウィジェット内で、センサ 1 つまたはペアセンサのスキャンを開始します。ISR は、イネーブルになっているウィジェットすべてがスキャンされるまで、センサのスキャンを続行します。IRS の使用により、この関数はブロックされなくなります。近接検知ウィジェット以外、すべてのウィジェットが初期設定ではイネーブルになっています。近接検知ウィジェットはマニュアルでイネーブルにしなければなりません。これは、長いスキャン時間が他のウィジェットの種類で必要になる高速応答に対応できないためです。

引数: なし

戻り値: なし

副作用: イネーブルになっているウィジェットがない場合、関数呼び出しの効果は発生しません。

uint8 CapSense_IsBusy(void)

説明: センサスキャンの状態を返します。

引数: なし

戻り値: uint8: スキャンの状態を返します。'1' – スキャンが進行中、'0' – スキャンが完了。

副作用: なし

void CapSense_SetScanSlotSettings(uint8 slot)

説明: カスタマイザまたは選択したスキャンスロットのウィザードにより、スキャン設定を行ないます(2 チャンネル デザインのセンサまたはセンサペア)。スキャン設定は、各センサについて IDAC 値(IDAC 構成用)および分解能を提供します。分解能はウィジェット内のすべてのセンサで同一です。

引数: uint8 slot: スキャンスロット数

戻り値: なし

副作用: なし

void CapSense_ClearSensors(void)

説明: すべてのセンサをアナログマルチプレクサバスから順番に切断して、非アクティブ状態に接続することで、すべてのセンサをノンサンプリング状態にリセットします。

引数: 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 HI-Z モードに設定され、アナログマルチプレクサバスに接続されます。また、これはコンパレータ出力にも影響します。

引数: uint8 sensor: 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、アナログマルチプレクサバスから切断され、非アクティブ状態になります。

引数: uint8 sensor: センサ番号

戻り値: なし

副作用: なし

uint16 CapSense_ReadSensorRaw(uint8 sensor)

説明: センサの Raw データを CapSense_SensorResult[ ] アレイから返します。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 カスタマイザによって順番に割り当てられるものです。Raw データは CapSense が提供するフレームワークの外側で計算を実行するのに使用します。

引数: uint8 sensor: センサ番号

戻り値: uint16: 現在の Raw データ値

副作用: なし

void CapSense_SetRBleed(uint8 rbleed)

説明: ブリード抵抗(Rb)接続用にピンを設定します。カスタマイザで定義されたものから現在の Rb ピン設定を選択すると同時に、この関数を呼び出すことができます。関数は、コンポーネントパラメータ設定を上書きします。**Current Source** が **External Resistor** に設定されている場合のみ、この関数が使用できます。

この関数は、センサの一部で異なるブリード抵抗値でのスキャンが必要な場合にイネーブルです。たとえば、通常のボタンはブリード抵抗の低い値でスキャンできます。近接検出器では、より大きなブリード抵抗で間隔を空けてスキャンすることで、近接検知の距離を最大化することができます。この関数は CapSense_ScanSensor()関数とともに使用できます。

引数: uint8 rbleed: CapSense カスタマイザで定義されるブリード抵抗用の序数です。

戻り値: なし

副作用: ブリード抵抗の数は 3 つに制限されます。範囲外の数は、関数により確認されません。

高レベル APIs

これらの API 関数は、センサウィジェット用の Raw データでの作業に使用します。スキャンしたセンサから取得した Raw データは、ボタンではオン/オフ、スライダでは位置、タッチパッドでは X と Y 座標に変換されます。

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_InitializeEnabledBaselines()	イネーブルになっているセンサのみをスキャンして、初期値を伴う CapSense_SensorBaseline[] アレイを読み込みます。 この関数は 2 チャンネル デザインでのみ使用できます。
CapSense_InitializeAllBaselines()	すべてのセンサをスキャンして、初期値を伴う CapSense_SensorBaseline[] アレイを読み込みます。
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値と呼ばれます。このベースラインは更新に k = 256 の低パスフィルタを使用します。
CapSense_UpdateEnabledBaselines	CapSense_SensorEnableMask[] アレイを確認し、CapSense_UpdateSensorBaseline()関数を呼び出し、イネーブルになっているセンサのベースラインを更新します。
CapSense_EnableWidget()	ウィジェットのすべてのセンサ エレメントをスキャンプロセスでイネーブルにします。
CapSense_DisableWidget()	ウィジェットのすべてのセンサ エレメントをスキャンプロセスからディスエーブルにします。
CapSense_CheckIsWidgetActive()	選択したウィジェットを CapSense_Signal[] アレイと比較して、指で押されたかどうかを決定します。
CapSense_CheckIsAnyWidgetActive()	CapSense_CheckIsWidgetActive()関数を使用して、CapSense CSD コンポーネントがアクティブ状態なウィジェットがあるかどうかを検出します。
CapSense_GetCentroidPos()	CapSense_SensorSignal[] アレイで、リニア スライダが指で押されたことをを確認し、位置を返します。
CapSense_GetRadialCentroidPos()	CapSense_SensorSignal[] アレイで、ラジアルスライダウィジェットが指で押されたことをを確認し、位置を返します。
CapSense_GetTouchCentroidPos()	指が存在する場合、この関数はタッチパッド内のセントロイドの計算をすることで、指の X と Y 位置を計算します。
CapSense_GetMatrixButtonPos()	指が存在する場合、この関数はマトリックスボタンでの指の行と列の位置を計算します。

void CapSense_InitializeSensorBaseline(uint8 sensor)

説明: 選択されたセンサ(1 チャンネル デザイン)またはペアセンサ(2 チャンネル デザイン)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。Raw 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。Raw データフィルタがイネーブルの場合は初期化されます。

引数: uint8 sensor: センサ番号

戻り値: なし

副作用: なし

void CapSense_InitializeEnabledBaselines(void)

説明: イネーブルになっているウィジェットすべてをスキャンします。スキャンプロセスでイネーブルにしたすべてのセンサについて、Raw カウント値は CapSense_SensorBaseline[ ] アレイにコピーされます。スキャンプロセスでディスエーブルにしたセンサについて、CapSense_SensorBaseline[ ] をゼロ値で初期化します。Raw データフィルタがイネーブルの場合は初期化されます。

この関数は 2 チャンネル デザインでのみ使用できます。

引数: なし

戻り値: なし

副作用: なし

void CapSense_InitializeAllBaselines(void)

説明: CapSense_InitializeSensorBaseline()機能を使用して、すべてのセンサをスキャンして、初期値を伴う CapSense_SensorBaseline[] アレイを読み込みます。RAW 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。Raw データフィルタがイネーブルの場合は初期化されます。

引数: 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値は経時的カウント値で、センサ別に独立して計算されます。CapSense_SensorBaseline[sensor] アレイ エレメントの更新に k = 256 のローパスフィルタを使用します。この関数は現在の Raw カウント値から過去のベースラインを差し引いて、差の数を計算し、CapSense_SensorSignal[sensor] に保存します。

自動リセットオプションがイネーブルになっている場合、ベースラインはノイズ閾値とは別個に更新されます。

自動リセットオプションがディスエーブルになっている場合、信号がノイズ閾値を上回ると、ベースラインは更新を停止し、信号がノイズ閾値を下回ると、ベースラインをリセットします。

ベースライン計算の前にイネーブルになった場合、Rawデータフィルタが値に適用されます。

引数: uint8 sensor: センサ番号

戻り値: なし

副作用: なし

voidCapSense_UpdateEnabledBaselines(void)

説明: CapSense_SensorEnableMask[ ]アレイを確認し、CapSense_UpdateSensorBaseline()関数を呼び出し、イネーブルになっているセンサのベースラインを更新します。

引数: 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のセンサをスキャンプロセスの一部としてイネーブルにします。

引数: uint8 widget: ウィジェットの数。各ウィジェットについて、次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
```

ウィジェット名はすべて大文字です。

戻り値: なし

副作用: なし

void CapSense_DisableWidget(uint8 widget)

説明: スキャンプロセスから選択したウィジェットセンサをディスエーブルにします。

引数: uint8 widget: ウィジェットの数。各ウィジェットについて、次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
```

ウィジェット名はすべて大文字です。

戻り値: なし

副作用: なし

uint8 CapSense_CheckIsWidgetActive(uint8 widget)

説明: 選択したセンサCapSense_Signal[ ]アレイの値をその指の閾値と比較します。ヒステリシスとデバウンスが考慮されます。センサがアクティブな場合、ヒステリシス量により閾値は低くなります。センサが非アクティブな場合、ヒステリシス量により閾値は高くなります。アクティブ閾値が一致したら、デバウンスカウンタがセンサのアクティブ状態遷移に到達するまで1ずつインクリメントします。この時点で、このAPIがウィジェットをアクティブに設定します。また、この関数は、CapSense_SensorOnMask[ ]アレイでセンサのビットを更新します。

タッチパッドおよびマトリックスボタンウィジェットがウィジェットのアクティブな状態を返すには、列および行の内部にアクティブなセンサが必要です。

引数: uint8 widget: ウィジェットの数。各ウィジェットについて、次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
```

ウィジェット名はすべて大文字です。

戻り値: uint8: ウィジェットセンサの状態。1ウィジェット内の1つまたは複数のセンサがアクティブな場合、0ウィジェット内のすべてのセンサが非アクティブな場合。

副作用: また、このウィジェットに属しているすべてのセンサについて、この関数によりCapSense_SensorOnMask[ ]の値が更新されます。デバウンスカウンタも、アクティブ状態への遷移があると、呼び出しのたびに修正されます。

uint8 CapSense_CheckIsAnyWidgetActive(void)

説明: CapSense_Signal[ ]アレイのすべてのセンサを指閾値と比較します。各センサについて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。

引数: なし

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

副作用: CapSense_CheckIsWidgetActive()関数として、すべてのセンサに対して同じ副作用があります。

uint16 CapSense_GetCentroidPos(uint8 widget)

説明: リニア スライダ内を指で押されたか、CapSense_Signal[ ]アレイを確認します。指の位置はCapSenseカスタマイザで指定したAPI分解能まで計算されます。イネーブルになっている場合、位置フィルタが適用されます。この関数は、リニア スライダ ウィジェットがCapSenseカスタマイザで指定されている場合のみ利用できます。

引数: uint8 widget: ウィジェットの数。各リニア スライダ ウィジェットについて、次の形式の定義があります:

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

例:

```
#define CapSense_MY_VOLUME1__LS               5
```

ウィジェット名はすべて大文字です。

戻り値: uint16: リニア スライダの位置の値。

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、関数はゼロからCapSenseカスタマイザで設定されたAPI分解能の値を返します。アクティブなセンサがない場合、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。

この関数に提供されるウィジェットの引数は確認されません。不適切なウィジェットの値により、予期しない位置計算が発生することがあります。

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高い場合、このサブルーチンは正しくない指圧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(ノイズレベルを超える高さに)設定します。

uint16CapSense_GetRadialCentroidPos(uint8 widget)

説明: ラジアルスライダ内を指で押されたか、CapSense_Signal[ ]アレイを確認します。指の位置はCapSenseカスタマイザで指定したAPI分解能まで計算されます。イネーブルになっている場合、位置フィルタが適用されます。この関数は、ラジアルスライダがCapSenseカスタマイザで指定されている場合のみ利用できます。

引数: uint8 widget: ウィジェットの数。各ラジアルスライダウィジェットについて、次の形式の定義があります:

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

例:

```
#define CapSense_MY_VOLUME2__RS               5
```

ウィジェット名はすべて大文字です。

戻り値: uint16: ラジアルスライダの位置の値です。

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、関数はゼロからCapSenseカスタマイザで設定されたAPI分解能の値を返します。アクティブなセンサがない場合、関数は0xFFFFを返します。

この関数に提供されるウィジェットの種類の引数は確認されません。不適切なウィジェットの値により、予期しない位置計算が発生することがあります。

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高い場合、このサブルーチンは正しくない指の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(ノイズレベルを超える高さに)設定します。

uint8 CapSense_GetTouchCentroidPos(uint8 widget, uint16* pos)

説明: 指が存在する場合、この関数はタッチパッド内のセントロイドを計算することで、指のXとY位置を計算します。XとY位置はCapSenseカスタマイザで設定したAPI分解能まで計算されます。指がタッチパッドにある場合'1'が返されます。イネーブルになっている場合、位置フィルタが適用されます。この関数は、タッチパッドがCapSenseカスタマイザで指定されている場合のみ利用できます。

引数: uint8 widget: ウィジェットの数。各タッチパッドウィジェットについて、次の形式の定義があります:

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

例:

```
#define CapSense_MY_TOUCH1__TP                5
```

ウィジェット名はすべて大文字です。

`(uint16* pos)`: タッチ位置が保存される、2つの`uint16`のアレイをポイントします:

```
pos[0] - X position;
pos[1] - Y position.
```

戻り値: uint8: 指がタッチパッドにある場合は1、指がタッチパッドにない場合は0。

副作用:

uint8 CapSense_GetMatrixButtonPos(uint8 widget, uint8* pos)

説明: マトリックスボタンに指が存在する場合、この関数は指の行と列の位置を計算します。指がマトリックスボタンにある場合'1'が返されます。この関数は、マトリックスボタンがCapSenseカスタマイザで指定されている場合のみ利用できます。

引数: uint8 widget: ウィジェットの数。各マトリックスボタンについて、次の形式の定義があります:

```
#define CapSense_"widget_name"__MB            5
```

例:

```
#define CapSense_MY_TOUCH1__MB                5
```

ウィジェット名はすべて大文字です。

(uint8* pos): タッチ位置が保存される、2つのunit8のアレイをポイント:

```
    pos[0] - column position;
    pos[1] - row position.
```

戻り値: uint8: 指がタッチパッドにある場合は1、指がタッチパッドにない場合は0。

副作用:

チューナーヘルパーAPI

これらの API 関数はチューナーGUI での作業に使用します。

関数	説明
CapSense_TunerStart()	CapSense CSDおよびEZI2Cコンポーネントを初期化し、ベースラインを初期化し、センサスキャンループを開始します。
CapSense_TunerComm()	チューナーGUIとの間の通信を実行します。

void CapSense_TunerStart(void)

説明: CapSense CSDコンポーネントとEZI2Cコンポーネントを初期化します。また、ベースラインを初期化し、現在イネーブルになっているセンサでセンサスキャンループを開始します。

引数: なし

戻り値: なし

副作用: なし

void CapSense_TunerComm(void)

説明: チューナーGUIとの通信のための関数を実行します。

- マニュアルモード:CapSense CSD コンポーネントから、センサスキャンおよびウィジェットプロセス結果をチューナーGUI に送信します。チューナーGUI から新しいパラメータを読み取り、CapSense CSD コンポーネントに適用します。
- Auto (SmartSense): チューナーGUI との通信のための関数を実行します。センサスキャンおよびウィジェットプロセス結果をチューナーGUI に送信します。自動チューニングのパラメータもチューナーGUI に送信されます。チューナーGUI パラメータは CapSense CSD コンポーネントに返信されません。

新しいデータを許可するためにチューナーGUIがCapSense CSDコンポーネントバッファを変更している間、この関数はブロックして待機します。

引数: 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ストロング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コンポーネント内の静電容量式センサで使用されるすべてのピンについて、ドライブモードを設定します。
CapSense_SetAllCmodsDriveMode()	CapSenseコンポーネント内のC_{MOD}コンデンサで使用されるすべてのピンについて、ドライブモードを設定します。
CapSense_SetAllRbsDriveMode()	CapSenseコンポーネント内のブリード抵抗(Rb)で使用されるすべてのピンについて、ドライブモードを設定します。**Current Source**が**External Resistor**に設定されている場合のみ使用できます。

void CapSense_SetAllSensorsDriveMode(uint8 mode)

説明: CapSenseコンポーネント内の静電容量式センサで使用されるすべてのピンについて、ドライブモードを設定します。

引数: 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コンポーネント内のC_{MOD}コンデンサで使用されるすべてのピンについて、ドライブモードを設定します。

引数: 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コンポーネント内のブリード抵抗(Rb)で使用されるすべてのピンについて、ドライブモードを設定します。**Current Source**が**External Resistor**に設定されている場合のみ使用できます。

引数: 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 関数では、センサおよびウィジェットデータのプロセスに複数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、デバッグやチューニングのために表示できます。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は可能です。グローバルアレイとは、以下を指します:

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

CapSense_SensorRaw [ ]

これには、各センサの Raw データが含まれます。アレイのサイズは、センサの数と同じです (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[ ]

これはバイトアレイで、センサのスキャン状態 CapSense_SensorEnableMask[0]にセンサ 0～7(センサ 0 はビット 0、センサ 1 はビット 1)のマスクされたビットが含まれます。CapSense_SensorEnableMask[1]にはセンサのマスクされたビット 8～15(必要な場合)が含まれ、以降も同様となります。このバイトアレイには、センサの合計数に必要な多くのエレメントが含まれます。ビットの値により、センサが CapSense_ScanEnabledWidgets() 関数呼び出しによりスキャンされたかどうかを指定します。1 – センサがスキャンされました。0 – センサはスキャンされていません。CapSense_SensorEnableMask[ ]データは、以下の関数により変更されます。

- 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[ ]と CapSense_maskTable[ ]

これらのアレイには、各センサ用のポートおよびピンマスクが含まれ、センサが接続されているピンを指定します。

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

CapSense_SensorBaselineLow[ ]

このアレイには、各センサのベースラインデータの分数値でのバイトを含みます。これは、ベースライン更新でローパスフィルタに使用されます。アレイのサイズは、センサの数と同じです。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[ ]

このアレイには、各センサのベースラインデータを含みます。アレイのサイズは、センサの数と同じです。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[ ]

このアレイには、各千差の現在の Raw カウントから過去のベースラインを引き算することで計算されたセンサ信号カウントが含まれます。アレイのサイズは、センサの数と同じです。**WidgetResolution** パラメータで、このアレイの分解能を **1 バイト**または **2 バイト**に定義します。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[ ]

これはバイトアレイで、センサのオン/オフ状態を含みます。

CapSense_SensorOnMask[0]はセンサ 0～7(センサ 0 はビット 0、センサ 1 はビット 1)のマスクされたビットが含まれます。CapSense_SensorOnMask[1]にはセンサのマスクされたビット 8～15(必要な場合)が含まれ、

以降も同様となります。このバイトアレイには、センサの合計数に必要な多くのエレメントが含まれます。センサがオン(アクティブ) の場合、ビットの値は 1 になります。センサがオフ(非アクティブ)の場合、ビットの値は 0 になります。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。定数の一部は条件により定義され、現在の設定で必要な場合にのみ存在します。

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

2 チャンネル デザインでは、1 つのチャンネルに属するセンサ数は以下のように定義されます。

- CapSense_TOTAL_SENSOR_COUNT__CH0 – チャンネル 0 に属する合計数を定義します。
- CapSense_TOTAL_SENSOR_COUNT__CH1 – チャンネル 1 に属する合計数を定義します。
- CapSense_CSD_TOTAL_SCANSLOT_COUNT – チャンネル 0 または 1 のいずれかでの最大のセンサカウントを定義します。

センサの定数

各センサに定数が与えられています。これらの定数は、以下の関数でパラメータとして使用できます:

- CapSense_EnableSensor()
- CapSense_DisableSensor()

定数名は以下で構成されます:

インスタンス名+"_SENSOR"+ウィジェット名+エレメント+"#elementnumber"+"__"+ウィジェットの種類

例:

```
#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
```

- Widget Name – ウィジェットのユーザ定義名(イネーブルになっている C スタイル識別子でなければなりません)。CapSense CSD コンポーネント内で、ウィジェット名は固有でなければなりません。ウィジェット名はすべて大文字です。
- Element Number – エレメント数は、ラジアルスライダ等、複数のエレメントを持つウィジェットにのみ存在します。タッチパッドやマトリックスボタンでは、エレメント数は単語'Col'または'Row'およびそれぞれの数で構成されます(例:Col0、Col1、Row0、Row1)。リニア スライダおよびラジアル スライダでは、エレメント数は文字'e'とその数で構成されます(例:e0、e1、e2、e3)。
- ウィジェットの種類 – 以下はウィジェットの種類です:

エイリアス	説明
BTN	ボタン
LS	リニア スライダ
RS	ラジアルスライダ
TP	タッチパッド
MB	マトリックスボタン
PROX	近接検知センサ
GEN	汎用センサ
GRD	Guard Sensor

ウィジェットの定数

各ウィジェットに定数が与えられています。これらの定数は、以下の関数でパラメータとして使用できます:

- CapSense_CheckIsWidgetActive()
- CapSense_EnableWidget()andCapSense_DisableWidget()
- CapSense_GetCentroidPos()
- CapSense_GetRadialCentroidPos()
- CapSense_GetTouchCentroidPos()

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+ウィジェットの種類

例:

```
#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サンプル

PSoC Creator は、Find Example Project ダイアログに数多く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コード例が含まれています。コンポーネント固有のサンプルを見るには、Component Catalog または回路図に置いたコンポーネントのインスタンスからダイアログを開きます。一般的なサンプルについては、Start Page または **File** メニューからダイアログを開きます。必要に応じてダイアログにある **Filter Options** を使用し、選択できるプロジェクトのリストを絞り込みます。

詳しくは、PSoC Creator ヘルプの「Find Example Project」トピックを参照してください。

ピンの割り当て

CapSense カスタマイザは、各 CapSense センサとサポート信号用に、ピンのエイリアス名を生成します。これらのエイリアスを使用して、デバイスの物理的ピンにセンサや信号を割り当てます。Design Wide Resources ファイル表示の Pin Editor タブで CapSense CSD コンポーネントセンサおよび信号をピンに割り当てます。

両側

PSoC デバイス内のアナログルーティングマトリックスは、左右 2 つに分かれています。デバイス左側には偶数のポート番号ピンが、右側には奇数のポート番号ピンがあります。

シリアル検知アプリケーションでは、センサピンをデバイスのいずれの側にも割り当てられます。アプリケーションで使用されるセンサが少数の場合、デバイスの片側にすべてのセンサ信号を割り当てると、アナログリソースのルーティングの効率が良くなり、アナログリソースがその他のコンポーネントに解放されます。

パラレル検知アプリケーションでは、CapSense コンポーネントは、2 つの独立したハードウェアで 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いずれにも個別の C_{MOD} および Rb(該当の場合)があり、それぞれにセンサピン一式があります。デバイス右側に一式が収められ、左側にもう一式が収められます。信号名のエイリアスにより、信号が関連付けられる側が指示されます。

センサピン – CapSense_cPort – ピンの割り当て

CapSense カスタマイザのウィジェットの種類およびウィジェット名にセンサ名を関連付けるために、エイリアスがあります。

センサのエイリアスは以下となります :

ウィジェット名+エレメント数+"__"+ウィジェットの種類

注 2 チャンネル デザインで、1 つのチャンネルに属するウィジェット エレメントは、チャンネルの CMOD のチップと同じ側にのみ接続できます。**PinEditor は、デザイン ルールのチェックでピンの割り当てが適切かどうかを確認しません。**. ビルドプロセス中、ピンの配置エラーはフラグ表示されます。

注オペアンプ出力 P0[0]、P0[1]、P3[6]、P3[7]には、他のピンよりも大きな寄生容量があります。これにより、CapSense アプリケーションの P0[0]、P0[1]、P3[6]、および P3[7]の指の応答が小さくなるため、可能であれば避けることが望まれます。使用を余儀なくされる場合、静電容量の差がスライダやタッチパッドの位置エラーに変換されないよう、個別のボタンに使用してください。

CapSense_cCmod_Port – ピンの割り当て

外部変調コンデンサ(C_{MOD})の片側が物理的ピンおよび他の GND に接続しなければなりません。2 チャンネル デザインでは 2 つの C_{MOD} コンデンサが必要であり、1 つはデバイスの左側、もう 1 つは右側に使用します。C_{MOD} は**任意のピン**に接続できますが、最も効率の良いアナログルーティングでは、以下のピンで直接接続することができます:

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

C_{MOD} コンデンサのエイリアスは次の通りです。

エイリアス	説明
CmodCH0	C_{MOD}(チャンネル 0 用) 。
CmodCH1	C_{MOD}(チャンネル 1 用)。2 チャンネル デザインでのみ使用できます。

C_{MOD} の理想値は、センサのスキャン電圧での電圧スイングにより異なります。電圧スイングが高いほど、C_{MOD} 値も高くなるものとします。センサの電圧スイングは IDAC モードおよびリファレンス電圧の設定(Vref)により異なります。推奨される C_{MOD} 値では次の式を使用します。

IDAC ソーシング モード用:

C_{MOD} = 2.2 nF × Vref

IDAC シンキング モード用:

C_{MOD} = 2.2 nF ×(V_{DD} – Vref

セラミック コンデンサを使用してください。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** に設定されている場合、外部 Rb フィードバック抵抗の値を、最適の C_{MOD} 値を決定する前に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** に設定されている場合、外部ブリード抵抗(Rb)が必要になります。外部ブリード抵抗(Rb)を物理的ピンと変調コンデンサ(C_{MOD})のグランドされていない接続部に接続する必要があります。

チャンネルごとに最大 3 つのブリード抵抗がサポートされます。3 つのピンはブリード抵抗 cRb0、cRb1 および cRb2 に割り当てられます。

外部ブリード抵抗のエイリアス:

エイリアス	説明
Rb0CH0、Rb1CH0、Rb2CH0	チャンネル 0 用外部抵抗。
Rb0CH1、Rb1CH1、Rb2CH1	チャンネル 1 用外部抵抗。2 チャンネル デザインでのみ使用できます。

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:

- 異なるセンサのタッチの生カウントを監視する。
- 選択され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する。抵抗値が減ると、生カウントは増えます。

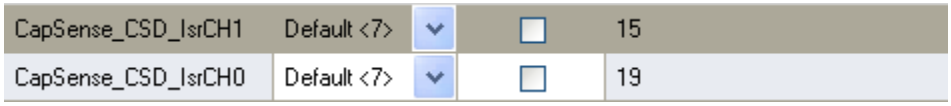
一般的なブリード抵抗値は 500 Ω～10 kΩ の間で、センサの静電容量にに依存します。

割り込みサービスルーチン

CapSense コンポーネントでは、各センサスキャン終了後にトリガされる割り込みを使用します。必要に応じてユーザ自身のコードが追加できる場合、スタブルーチンが提供されます。スタブルーチンは、プロジェクトが初めてビルトされた際に *CapSense_INT.c* ファイルで生成されます。割り込み数は、チャンネル数に基づく CapSense モード選択により異なり、チャンネルごとに 1 つです。ビルドの間に保存するには、コメントタグの間にユーザのコードを追加する必要があります。

2 チャンネルモード ISR の優先順位の設定

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 デザインで設定した ISR の優先順位に制限が発生します。チャンネル ISR ルーチンが再入可能になるのを防止するには、2 つのチャンネルが同一でなければなりません。



機能説明

定義

センサ

CapSense エレメント 1 つが 1 つのピンで PSoC に接続されます。センサは基板上の導電性エレメントです。センサの例には、以下が含まれます。Copper on FR4、Copper on Flex、Silver ink on PET、ITO on glass。

スキャン時間

スキャン時間とは CapSense モジュールが 1 つまたは複数の静電容量式センサをスキャンする一定時間です。所与のスキャンセンサに複数のセンサを組み合わせて、近接検知などのモードをイネーブル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 1 つまたは複数のスキャンセンサでビルドされ、より高いレベルの機能を実現します。CapSense ウィジェットの例として、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、近接検知センサが上げられます。

FingerThreshold

この値は、センサ上に指が存在するかどうかを決定するために使用します。

NoiseThreshold

静電容量スキャンでノイズのレベルを決定します。ベースラインアルゴリズムにより、ノイズがフィルタリングされ、センサのベースライン値で電圧や温度の変化を追跡します。

デバウンス

センサのアクティブ状態への遷移のためのデバウンス カウンタを設定します。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"値が指の閾値+ヒステリシスを上回る状態を維持しなければなりません。高い振幅および周波数のノイズをフィルタリングするために必要です。

ヒステリシス

指の閾値で使用するヒステリシス値を設定します。ヒステリシスが望ましい場合、カウント値が指閾値とヒステリシス値の合計を超えるまで、センサは "On" または "Active" とみなされません。測定したカウント値が指閾値からヒステリシス値を引いた値を下回るまで、センサは "Off" または "Inactive" とみなされません。

API 分解能 – 補間とスケーリング

多くの場合、スライダセンサおよびタッチパッドでは、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。

重みづけを使用した補間位置の計算では、まずアレイをスキャンして、センサの位置がイネーブルであることを確認します。ここでは、近隣センサ信号のある番号がノイズ閾値を超えていることが要件となります。最も強い信号が見つかったら、その信号と、ノイズ閾値より高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小で 2 つ、最大で 8 つのセンサが使用されます。

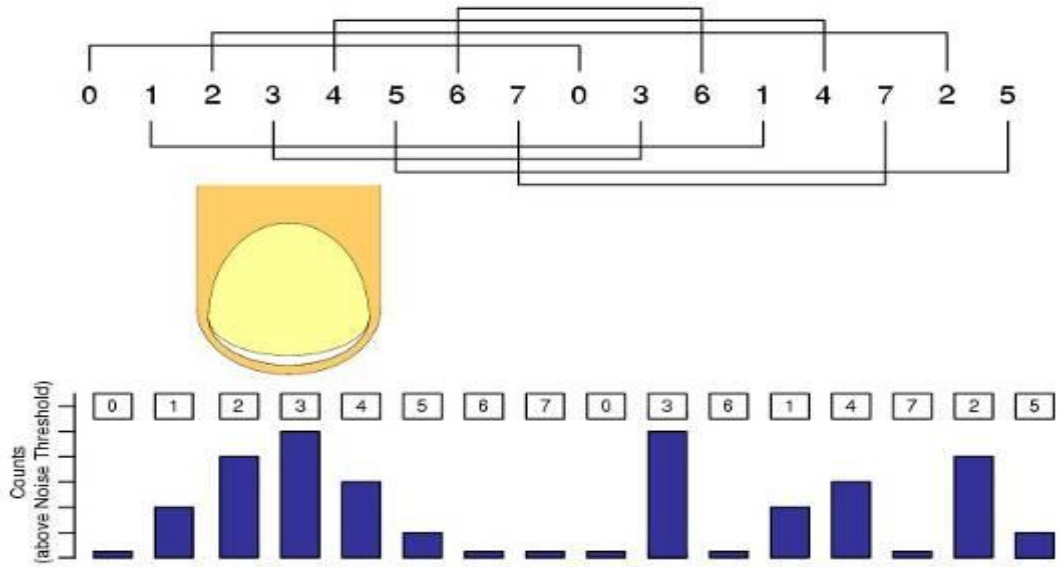
$$N_{Cent} = \frac{n_{i-1}(i-1) + n_i i + n_{i+1}(i+1)}{n_{i-1} + n_1 + n_{i+1}}$$

通常、計算結果は整数ではありません。たとえば 12 個のセンサに対して 0～100 という範囲である場合、セントロイドを特定の分解能の形で報告するには、スカラ量をセントロイドに掛けます。1 つの計算で補間とスケーリング 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 でハンドルされます。スライダセンサカウントと分解能は、CapSense CSD カスタマイザで設定します。

ダイプレックス

ダイプレックススライダでは、スライダの各 PSoC センサ接続は、スライダセンサのアレイにある 2 つの物理的な位置にマッピングされます。物理的位置の最初の(もしくは数字が小さい)半分は、CapSense カスタマイザで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ピングされます。物理的位置の後の(数字が大きい)半分は、カスタマイザのアルゴリズムで自動的にマッピングされ、取り込みファイルに一覧表示されます。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



スライダ中のセンサ静電容量は均衡がとれていなければなりません。センサや PCB のレイアウトによって、一部のセンサペアではセンサ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 インデックス表は、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、以下の参照用の表に含まれます。

表 2. 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

スライダセグメントの総カウント	セグメントシーケンス
10	0,1,2,3,4,0,3,1,4,2
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
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

スライダセグメントの総カウント	セグメントシーケンス
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、ジッタ)が提供されています。フィルタを Raw センサのデータと一緒に使用して、センサのノイズを低減したり、スライダとタッチパッドの位置データと一緒に使用して、位置ノイズを低減することができます。

Median フィルタ

メジアン フィルタは最新の 3 つのサンプルを確認し、メディアン値をレポートします。3 つのサンプルを並べ替えて、中央値を取ることで、メディアンが計算されます。このフィルタは、短いノイズスパイクを除去し、1 つのサンプルの遅延を生成するために使用されます。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、センサ(Raw)およびウィジェット(位置)ごとに RAM が 4 バイト消費されます。これは、初期設定ではディスエーブルになっています。

Averaging フィルタ

アベレージ フィルタは位置の最新の 3 つのサンプルを確認し、単純な平均値をレポートします。これは、短いノイズスパイクを除去し、1 つのサンプルの遅延を生成するために使用されます。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、センサ(Raw)およびウィジェット(位置)ごとに RAM が 4 バイト消費されます。これは、初期設定ではディスエーブルになっています。

First Order IIR フィルタ

一次 IIR フィルタは、Raw およびセンサフィルタの両方で推奨されるフィルタです。これは、最小量の SRAM しか必要なく、高速応答を提供するためです。IIR フィルタは最新のセンサまたは位置データをスケールし、過去のフィルタ出力をスケールしたバージョンに加算します。このフィルタをイネーブルにすると、センサ(Raw)およびウィジェット(位置)ごとに RAM が 2 バイト消費されます。IIR1/4 は、Raw および位置フィルタの両方で、初期設定でイネーブルになっています。

First Order IIR フィルタ:

$$\text{IIR } 1/2 = 1/2\,\text{previous} + 1/2\,\text{current}$$

$$\text{IIR } 1/4 = 3/4\,\text{previous} + 1/4\,\text{current}$$

$$\text{IIR } 1/8 = 7/8\,\text{previous} + 1/8\,\text{current}$$

$$\text{IIR } 1/16 = 15/16\,\text{previous} + 1/16\,\text{current}$$

Jitter フィルタ

このフィルタは、2 つの値(ジッタ)の間をトグルする Raw センサまたは位置データからのノイズを除去します。最新のセンサ値が最後のセンサ値よりも大きい場合、前のフィルタ値が 1 ずつインクリメントされ、小さい場合は値がデクリメントされます。4 つの LSB ピークツーピークまたはそれ以下のノイズがあるデータに適用すると、最も効果が高くなります。また、低速応答で問題がない場合、位置センサの一部で役立ちます。このフィルタをイネーブルにすると、センサ(Raw)およびウィジェット(位置)ごとに RAM が 2 バイト消費されます。これは、初期設定ではディスエーブルになっています。

CapSense システムでの水の影響

水滴と指が CapSense に与える影響は類似しています。ただし、水滴は検知領域の表面全体に影響を与え、指の影響とは異なります。

CapSense 表面での水の影響には、複数のパターンがあります:

- デバイス表面に、水の細い筋または流れが形成される。
- 独立した水滴。
- デバイス表面の大部分または全体にわたる水の流れ。デバイスが洗浄または浸漬された場合。

水に含まれる塩またはミネラルにより、水は導電性になります。さらに、濃度が高いほど水の導電性が高くなります。石鹸水、海水、ミネラルウォーターなどは CapSense に悪影響を与える液体です。これらの液体がデバイス表面の指のタッチをエミュレートし、デバイスの誤動作の原因となります。

防水性および検知

この機能により CapSense CSD コンポーネントを設定し、CapSense への水の影響を防ぎます。この機能で、次のパラメータを設定します:

- シールド電極をイネーブルにし、センサのハードウェアレベルでの水滴の影響を補正するのに使用します。
- ガードセンサを追加します。ガードセンサですべてのセンサを囲みます。実際に静電検知ウィジェットを覆う場合は、ガードセンサにより水がかからないように配置する必要があります。ウィジェット状態の CapSense 出力は、ガードセンサがトリガされた際に、プログラムによりブロックされなければなりません。

シールド電極

一部のアプリケーションでは、水膜や水滴がある場合でもオペレーションの信頼性が要求されます。白物家電、車載アプリケーション、様々な産業用アプリケーションなどでは、凝結の原因となる水、氷、湿度変化があっても誤動作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外部に装備します。水膜がデバイスのオーバーレイの表面にある場合、シールドと検知電極のカップリングが増えます。シールド電極は、寄生容量の影響を低減し、検知静電容量の変化の処理をするダイナミックレンジを広げます。

一部のアプリケーションでは、湿気により電極間のカップリングを増やして、検知電極の静電容量測定値のタッチ変化の負の影響を発生させるため、シールド電極信号と検知電極に対するその相対的位置を適切に選ぶことが有効です。これにより、ハイレベルソフトウェア API で、湿気により生じるタッチの誤認を防ぎ、操作を簡素化します。CapSense CSD コンポーネントはシールド電極用の個別の出力をサポートし、PCB ルーティングを簡素化します。

図 2. シールド電極 PCB レイアウト例

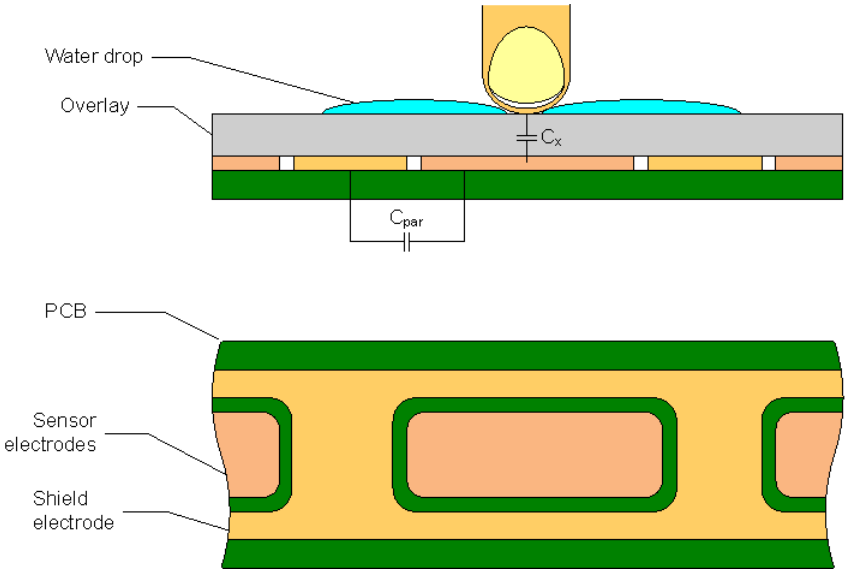


図 2 は、ボタンのシールド電極のレイアウト構成例を示しています。シールド電極は、LCD ドライブ電極のノイズを阻止し、同時に浮遊容量を低下するため、透明な ITO タッチパッドデバイスでは特に有用です。

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。

水滴がシールドと検知電極の間にある場合、寄生容量(C_{PAR})が増え、変調器の電流が低下することがあります。

シールド電極は、いずれのピンにも接続できます。ドライブモードを 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および制限

CapSense CSD コンポーネントでは、以下のモードがシールド電極で使用できます。

電流モード IDAC ソーシング

このモードでは、センサが GND および Vref = 1.024 V を代替するため、一定の制限があります。シールド電極信号は GND および Vddio(通常は電源と同等)の間で代替します。この差は大きく、シールド信号はセンサからの信号を完全にオフセットします。これに対する解決法として、以下が考えられます:

- 高い Vref を使用して最小値の差を排除します。この目的のためには、VDAC をリファレンスとして使用することができます。
- SIO ピンをシールドとして使用し、出力を Vref と同一にします。CapSense CSD 出力 Vref 端子を使用して、Vref を SIO ピンにルーティングすることができます。これは推奨される手法です。このモードでは、シールドへのセンサの接続は使用しないでください。これは、出力が Vddio と同一になるためです。Vref = 1.024 V 設定にはルーティングの制限があり、ピンにルーティングすることができません。

電流モード IDAC シンキングおよび外部レジスタ

これらのモードでは、センサが Vdddio および Vref = 1.024 V を代替するため、シールドおよび非アクティブセンサモードの使用に制限はありません。シールド電極信号は GND および Vddio(通常は電源と同等)の間で代替します。この場合の差はあまり大きくなく、問題は発生しません。

ガードセンサの実装

ガードセンサは防水アプリケーションで一般的に使用され、表面の水を検出します。

ガードセンサを追加するために、**Advanced** タブオプションがあります。一般的に、検知領域表面の境界線に配置するなど、このセンサには特別なレイアウトが必要になります。ガードセンサ表面に水が存在すると、ウィジェットがアクティブになります。ウィジェットのアクティブ検出ファームウェア CapSense_1_IsWidgetActive()を使用して、ガードセンサの状態を定義できます。

CapSense ウィジェットの検出は、ガードセンサがトリガされる一定の期間、ユーザコードでプログラムによりブロックする必要があります。ガードセンサがトリガされると、水が存在し、その他のセンサの信頼性が低下します。

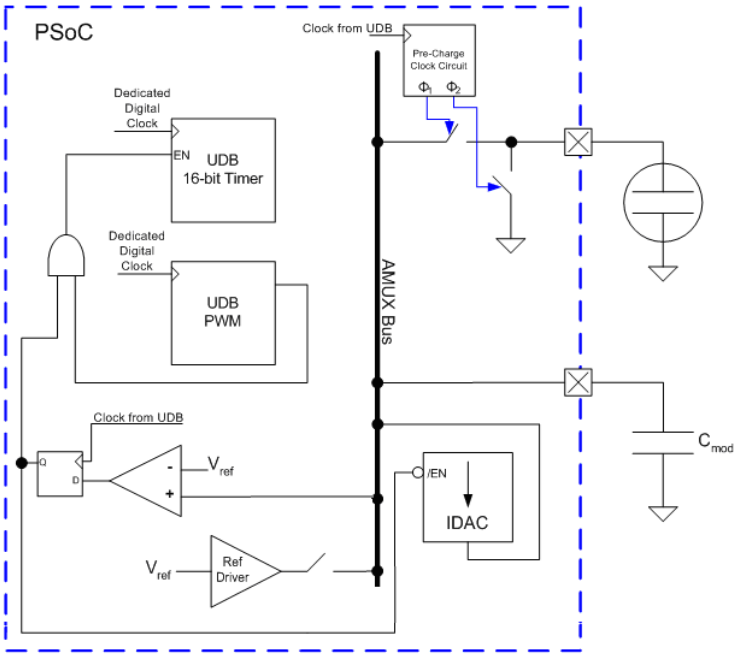
ガードセンサのサイズを考慮すると、ガードセンサと他のセンサの信号は異なります。すなわち、標準的なセンサ表面よりも多くの水が存在する可能性があります。そのため、水滴の存在により受信した信号は、指でタッチしたことで生じた信号よりもはるかに強くなります。これに基づいてトリガ閾値およびフィルタを設定することで、ガードセンサでの指のタッチの影響を受けないようにします。ガードセンサは、特別なオプションなしでスキャンします。ガードセンサのスキャン中、シールド電極はディスエーブルになっていません。2 チャンネル デザインのガードセンサは、常に自身で最後にスキャンされます。

ブロックダイアグラムと設定

シグマデルタ(CSD)変調器を使用した静電容量式検知は、スイッチト キャパシタ アナログ手法およびデルタシグマ変調器を使用して検知したスイッチト キャパシタの電流をデジタルコードに変換することで、静電容量を検出します。これにより、導電性センサのアレイを使用したボタン、スライダ、近接検知器、タッチパッド、タッチスクリーンが実装できるようになります。ハイ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が向上します。また、環境および物理センサの差を補正します。基本の CSD メソッドで 3 つのアナログハードウェアがあります。これらについては、以下のセクションで詳細を記載しています。

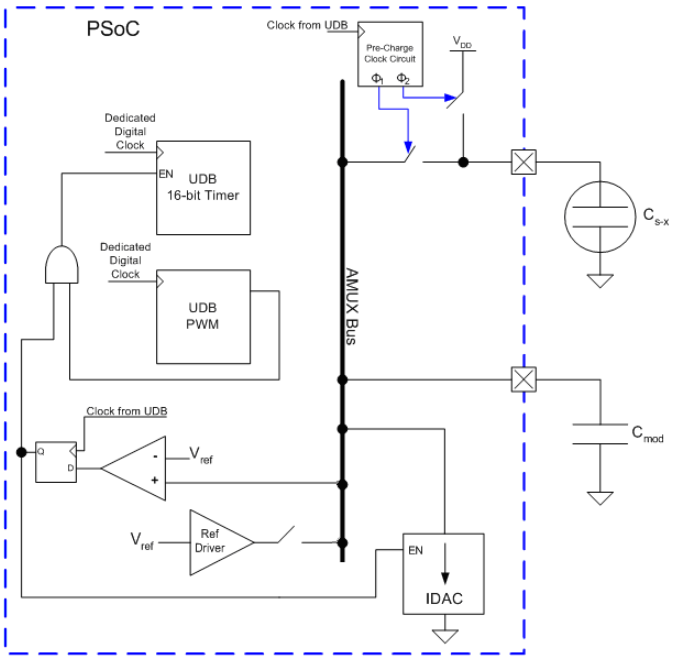
IDAC ソーシング

センサのスイッチ段階は、GND および変調コンデンサに接続する AMUX バスの間を代替するように設定されています。この設定では IDAC はセンサのソース電流として設定されています。



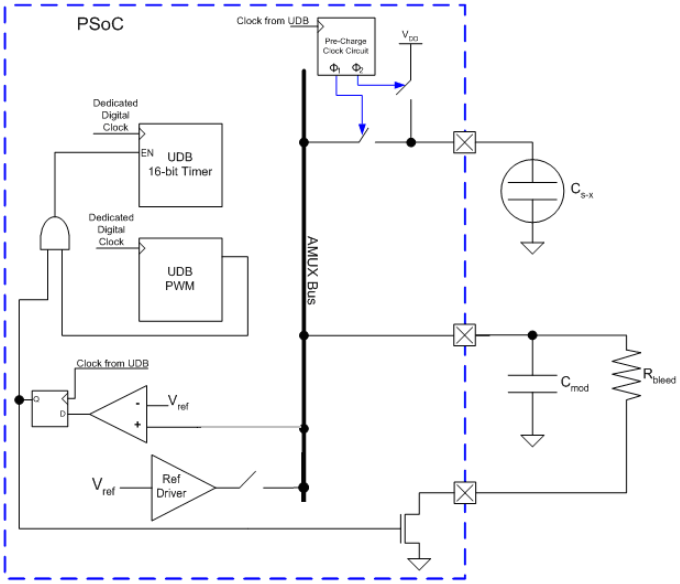
IDAC シンキング

センサのスイッチ段階は、V_{DD} および変調コンデンサに接続する AMUX バスの間を代替するように設定されています。この設定では IDAC はセンサからのシンク電流として設定されています。



IDAC ディスエーブル、外部 Rb を使用

外部ブリード抵抗の Rb を使用すると、IDAC シンキング設定と同様に機能しますが、IDAC の代わりにグランド Rb の抵抗になります。ブリード抵抗は C_{MOD} および GPIO の間に物理的に接続されています。GPIO は "Open-Drain Drives Low" ドライブモードで設定されます。このモードにより C_{MOD} は Rb で放電することができます。



DC 電気的特性と AC 電気的特性

5.0-V/3.3-V 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	*Units*
Value	--	2.7	5.0	5.5	V

ノイズ

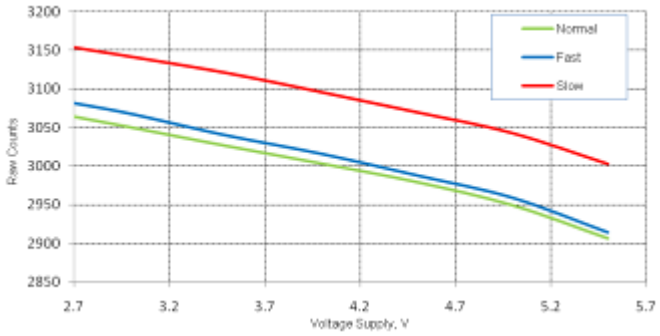
パラメータ	テスト条件とコメント	Min	Typ	Max	Units
ノイズカウント、ピークツーピーク(ノイズカウント /(ベースラインカウント)	分解能 = 16(ノイズカウント /(ベースラインカウント)	–	0.2	–	%
	分解能 = 14	–	0.3	–	%
	分解能 = 10	–	1.0	–	%

消費電力

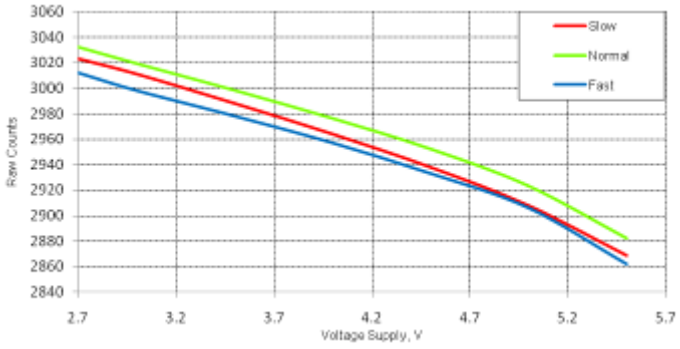
パラメータ	テスト条件とコメント	Min	Typ	Max	Units
アクティブ電流	V_{DD} = 3.3 V、CPU クロック = 24 MHz、CapSense スキャンクロック = 24 MHz、スキャン中の平均電流、8 センサ	–	8	–	mA
スリープ/ウェイク電流、100 ms レポートレート	V_{DD} = 3.3 V、CPU クロック = 24 MHz、CapSense スキャンクロック = 24 MHz、スキャン速度 = 超高速、分解能 = 9、8 センサ	–	78.9	–	µA
	V_{DD} = 3.3 V、CPU クロック = 24 MHz、CapSense スキャンクロック = 24 MHz、スキャン速度 = 高速、分解能 = 12、8 センサ	–	484	–	µA
スリープ/ウェイク電流、1s レポートレート	V_{DD} = 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、1 センサ	–	8.2	–	µA

図 – Raw カウント対電源電圧

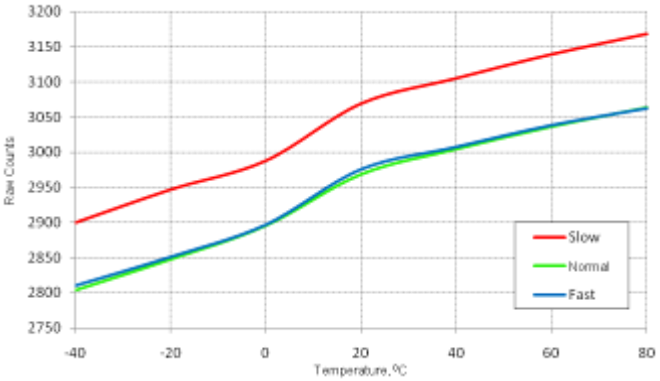
異なるスキャン速度での Raw カウント対電源電圧、PRS 16 フルスピード



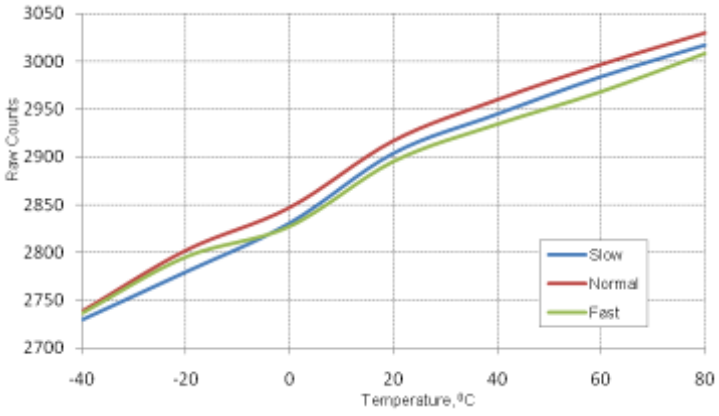
異なるスキャン速度での Raw カウント対電源電圧、PRS 8



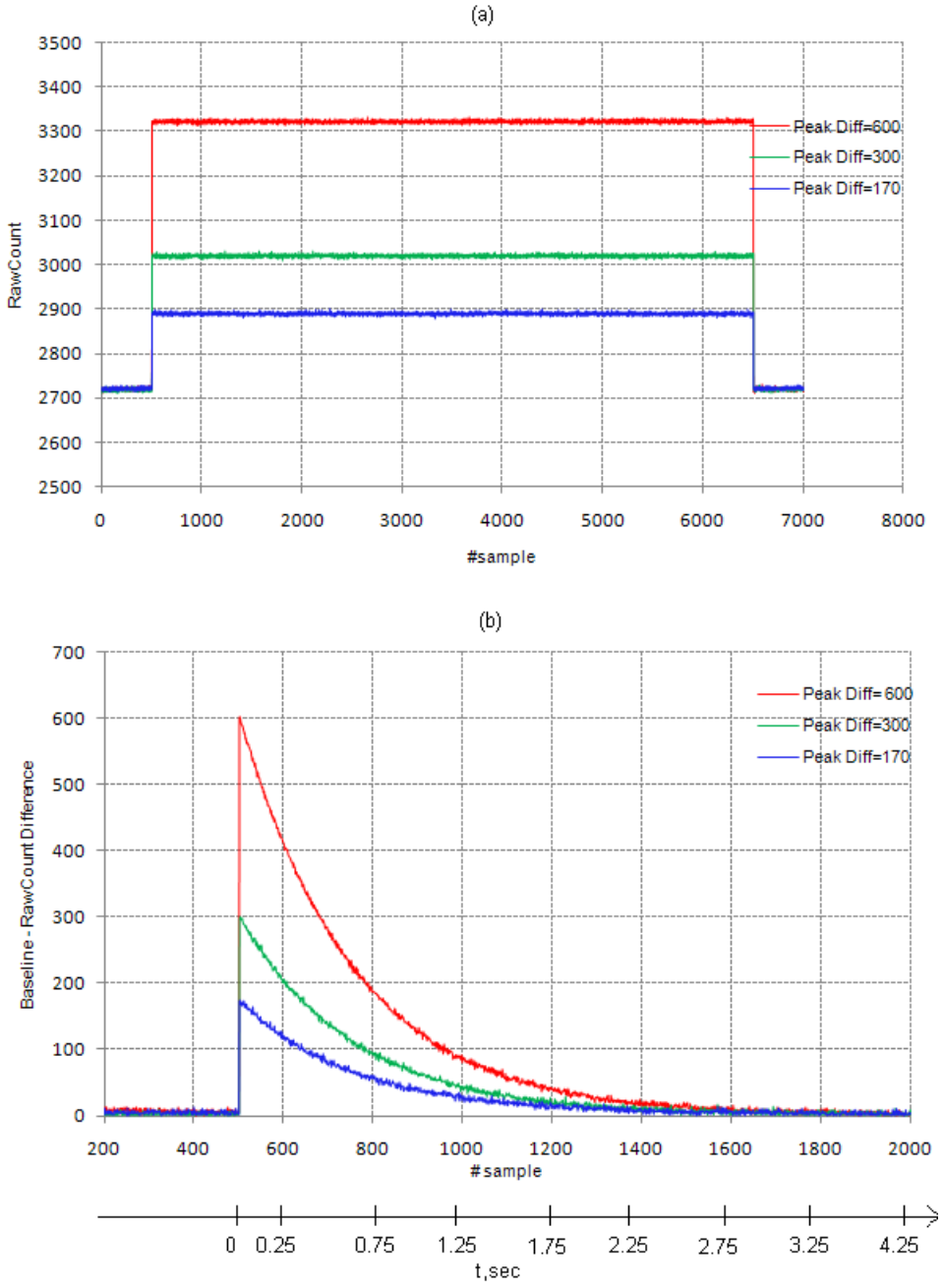
異なるスキャン速度での Raw カウント対温度、PRS 16 フルスピード



異なるスキャン速度での Raw カウント対温度、PRS 8



異なる Raw カウントステップの変更値での時間によるベースラインの変動
(a)異なるステップ値での RawCounts ステップの変化
(b)Raw カウント間の差。



コンポーネントの変更履歴

バージョン	変更内容	変更理由 / 影響
3.10	一部の変数をインスタンス名にプリフィックスし、コンポーネントとインスタンスごとに固有になるようにしました。	複数の CapSense CSD コンポーネントがデザインに含まれている場合、変数の競合を防止します。このような競合は不適切な動作の原因となります。この修正はPSoC 3 ES2とPSoC 5デバイスに固有のものです。
	更新された CapSense_EnableSensor()および CapSense_DisableSensor()関数でポート 15 のピンを正しくハンドルします。	ポート 15 のピンを CapSense で使用することができます。修正前のポート 15 ピンは有効ではなく、これらのポート 15 ピンの関数がメモリ破損の原因となる場合がありました。
3.0	CapSense_GetMatrixButtonPos()API 関数がマトリックスボタンのタッチ位置計算に追加されました。	過去には、マトリックスボタンのタッチ位置は、ユーザが計算していました。新しい関数でこの問題が簡単になります。
	マトリックス ボタンのタッチ位置は、チューナーに送信され、適切な GUI ウィジェットに表示されます。	マトリックスボタンのタッチ位置計算用に新しい関数が追加されたため、この位置はチューナーに送信されます。過去には、タッチ位置はチューナーにより API とは別個に計算されていました。
	Multiple Analog switch dividers オプションが追加されました。これによって個別のアナログスイッチ ディバイダを各スキャンスロットに設定できるようになりました。コンポーネントは、単一のアナログスイッチ ディバイダ(過去のコンポーネントバージョンと同様)または複数のアナログスイッチ ディバイダで動作することができます。	センサごとに異なるアナログスイッチ ディバイダの値を使用することで、個別のスキャンスロットで柔軟なチューニングが可能になります。
	アナログスイッチ ディバイダは、チューナーに送信され、GUI で表示されます。さらに、アナログスイッチ ディバイダはチューナーから変更できます。	ユーザはセンサのアナログスイッチ ディバイダを確認しながら、変更することができます。
	自動チューニング手順の機能が拡張され、異なる IDAC モード(ソーシングおよびシンキング)と Vref 値(1.024VおよびVDAC)が使用できるようになりました。追加の計算がカスタマイザおよびファームウェア部分に追加されました。	IDAC シンキングモードでの自動チューニングがイネーブルになり、リファレンス電圧の生成にVDAC が使用できるようになりました。
	チューナーがイネーブルになっている場合の、データの準備と送信の手順が変更されました。	過去にはデータ送信の準備および送信は並行して処理されており、バイトのスワップが実行される際に問題が発生していました。
	自動リセットがイネーブルになっている場合の、ベースライン更新アルゴリズムが変更されました。	過去には、自動リセット関数がイネーブルになっている場合、ベースライン未満に Raw データのカウントが多数ジャンプしても、ベースラインは即時に Raw データに調整で戻されませんでした。これは、ユーザが長時間ノード上に指を置いたままにすると発生する可能性があり、ON 条件の間、ベースラインが徐々に Raw データに近づく原因となっていました。上段に “stair stepped” するのと同様に下段に “stair stepped” していました。これは、PSoC 1 CapSense 機能とは異なっていました。

バージョン	変更内容	変更理由 / 影響
	CapSense_GetTouchCentroidPos()API 関数が変更されて、グローバルアレイの使用を排除し、タッチ位置計算を保存します。	過去のバージョンの関数では、タッチパッドのタッチ位置保存にグローバルアレイを使用していました。これは追加のメモリが必要になり、複数のタッチパッドを使用した場合に、アレイインデックスを計算していたユーザには便利になりました。新しいバージョンでは、タッチ位置保存のためのアレイのポインタが、ローカルパラメータとして関数に送信されます。そのアレイの保存用のスタックが使用できるようになり、複数のタッチパッドを使用した場合に、インデックス計算が不要になります。

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

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

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

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

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