CapSense® デルタシグマのデータシート CSD V 1.50

Copyright © 2007-2011 Cypress Semiconductor Corporation. All Rights Reserved.

リソース	PSoC® ブロック			API メモリ (バイト) 標準値		ピン (外部 I/O あたり)
	デジタル	アナログ CT	アナログ SC	Flash	RAM	
CY8C24x94、CY8CLED0xD、CY8CLED0xG. Flash、RAM、ピンの使用は、センサの数と構成によって変わります。						
PRS16 ベースのユーザ モジュール (1 センサ付き)	3	1	0	951	28	2 ー 5
PRS8 ベースのユーザ モジュール (1 センサ付き)	1	1	0	923	26	2 ー 5
プリスケーラ付き PRS8 ベース (1 センサ付き)	2	1	0	935	26	2 ー 5
それぞれの追加 CapSense ボタン	-	-	-	2	10	1
5 素子を持つ静電容量式スライダを使用する場合は、スタティックコードと RAM が増加します。	-	-	-	584	79	-
それぞれの追加スライダ素子	-	-	-	11	2	1
スライダ ダイプレックスを使用する場合は、スタティックコードと RAM が増加します。	-	-	-	14	-	-

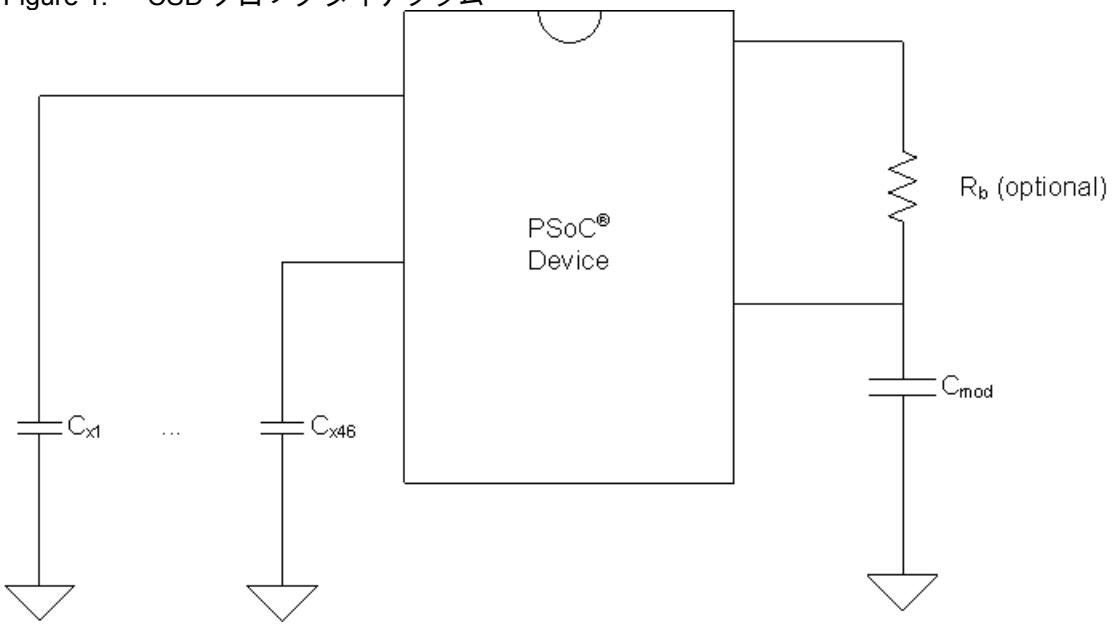
このユーザ モジュールを使用する機能例として完全に設定されたプロジェクトについては、以下を参照してください。www.cypress.com/psocexampleprojects

特徴と概要

- 1 ~ 46 個の静電容量式センサをスキャン。
- ガラスのオーバーレイが最大 15 mm まで検知可能。
- ワイヤベースセンサでは 20 cm の近接検知。
- AC メインノイズ、EMC ノイズ、電源電圧変化に対する高耐性。
- 異なる独立· スライド式静電容量式センサの組み合わせをサポート。
- ダイプレックスを使用したスライドセンサの物理的分解能を 2 倍に。
- 補間法を使用してスライドセンサの分解能を向上。
- 2 つのスライド センサを使ったタッチパッド サポート。
- 高抵抗の導電体材料を使った検知サポート (ITO フィルムなど)。
- 水膜や水滴がある場合にも信頼できる動作を保証するシールド電極サポート。
- CSD ウィザードによるセンサとピン割り当てのサポート。
- 温度、湿度、静電気放電 (ESD) を処理する、統合されたベースライン更新アルゴリズム。
- 調整が簡単な操作パラメータ。
- 生データのモニタリングとリアルタイムのパラメータ最適化を行う PC GUI アプリケーションサポート。

CSD (デルタシグマ変調器を使った静電容量の検知) は、デルタシグマ変調器を備えたスイッチド キャパシタ手法を使って CapSense® 機能を提供し、検知用スイッチド キャパシタの電流をデジタル コードに変換します。

Figure 1. CSD ブロック ダイアグラム



クイックスタート

1. I2C や LCD など、専用ピンを必要とするユーザ モジュールが使用されている場合、それを選択、配置します。必要に応じて、ポートやピンを割り当てます。
2. CSD ユーザ モジュールを選択、配置します。
3. Workspace Explorer で CSD ユーザ モジュールを右クリックし、CSD ウィザードを開きます (ウィザードについては後述します)。
4. 使用するセンサ、スライダ、回転式スライダの数を設定します。
5. 各センサの設定を指定します。
6. ピンおよびグローバル パラメータを設定します。すべてのパラメータの説明を読み、要件とガイドラインにしたがってください。
7. アプリケーションを生成し、Application Editor を開きます。
8. 個別のセンサ、スライド式センサ、またはタッチパッドを実装するために必要なサンプルコードを使用します。
9. I^2C-USB ブリッジをターゲット回路板に接続し、信号を確認します。
10. CSD パラメータを変更して設定を最適化し、アプリケーションをリビルドします。
11. PSoC デバイスをプログラムし、モジュール動作を確認します。CSD パラメータを調整し、*「CapSense の信号対ノイズ比の要件」*で説明されている 5:1 の SNR 要件を満たしてください。AN2403 で説明されている 5:1 の SNR 要件を達成してください。

問題がある場合は、*「付録」*の *「トラブルシューティング」*を参照してください。

機能説明

静電容量式センサのアレイは、独立したセンサ、スライド式センサ、直交スライダ ペアとして実装されたタッチパッドの組み合わせから構成されます。ハイレベルの判断ロジックにより、温度、湿度、電源の電圧変化といった環境要因が補正されます。浮遊容量を軽減するためのセンサーアレイをシールドするシールド エレクトロードが使用できます。これにより、水膜や水滴がある場合でも、安定した動作を実現します。

ハイレベルのソフトウェア関数により、分解能を上げるため物理的な 2 ヵ所で単一の電気センサを使用する、スライダのダイプレックスが可能です。この関数は、物理センサの位置間で、得られたセンサー位置の補間機能も提供します。

静電容量式センサは、次の物理、電気、ソフトウェア コンポーネントから成ります。

- 物理コンポーネント
  - 物理センサ。通常、絶縁カバー、軟質膜、またはディスプレイ上の透明なオーバーレイを備え、PSoC に接続された PCB 上に構築される伝導パターンです。
- 電気コンポーネント
  - センサの容量をデジタル形式に変換する部分。変換システムは、検知を担うスイッチド キャパシタ、デルタシグマ変調器、変調器の出力ビット ストリームを読み取り可能なデジタル形式に変換する、カウンタベースのデジタル フィルタから構成されます。
- ソフトウェア
  - 検出および補正ソフトウェア アルゴリズムが、カウント値をセンサの検出結果に変換します。

– 連続的・依存型センサの場合 (スライダやタッチパッドなど)、API はセンサの物理的ピッチよりも高い分解能で位置を補間します。たとえば、10 個のセンサでボリューム スライダを作成し、提供されているファームウェアを使用してボリューム レベルを 100 まで拡張することができます。また、同じ API を使うと、途中から連結しあう 2 つの静電容量式センサを利用して、その間にある伝導性物体 (指など) の位置を特定することもできます。

静電容量を測定する方法は多数ありますが、このユーザ モジュールで使用されている方法は、スイッチド キャパシタとシグマデルタ変調器を組み合わせるものです。

CSD ユーザ モジュールを初めて使う際は、その前に以下の文書を読むことをお勧めします。

- *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
- *CapSense アプリケーションをデバッグするためのチャート作成ツール* – AN2397
- *PSoC CapSense アプリケーションの EMC 設計における考慮点* – AN2318
- *容量検知アプリケーションにおける消費電力とスリープの考慮点* – AN2360
- *PSoC CapSense のレイアウト ガイドライン* – AN2292
- *ユニバーサル非同期トランスミッタのソフトウェア実装* – AN2399
- *耐水静電容量検知* – AN2398

静電容量の測定

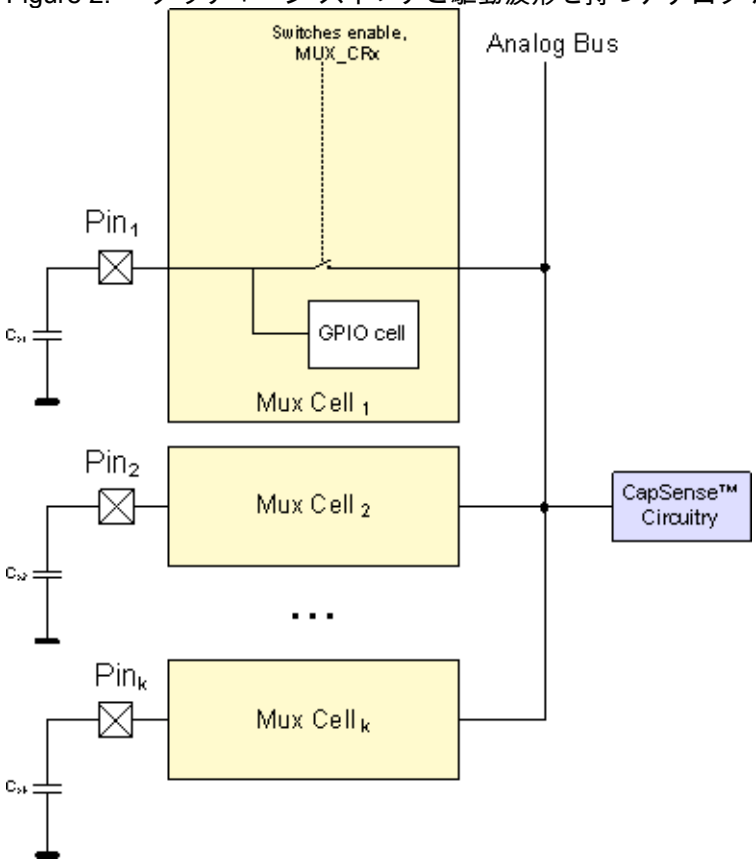
判定ロジックは、ファームウェアに実装されます。ファームウェアは、静電容量測定を分析し、環境要因による容量のゆっくりした変化を追跡します。また、ボタンのタッチを検出し、スライダの位置を計算する判定ロジックを実行します。

センサ アレイのスキャン

CY8C24x94 デバイス ファミリは、2 つのアナログ バスを持ちます。2 つのアナログ バスは互いに接続されており、すべてのピンでセンサのスキャンが可能となります。CSD ユーザ モジュールは、内部のプリチャージ スイッチを使って、クロック信号フェーズ Ph_1 でアクティブなセンサを充電し、フェーズ Ph_2 でアナログ バスをセンサに接続します。デルタシグマ変調器の変調コンデンサとコンパレータの入力は、アナログ バスに永続的に接続されます。

ファームウェアは、MUX_CRx レジスタで該当するビットを設定して、センサのスキャンを実行します。

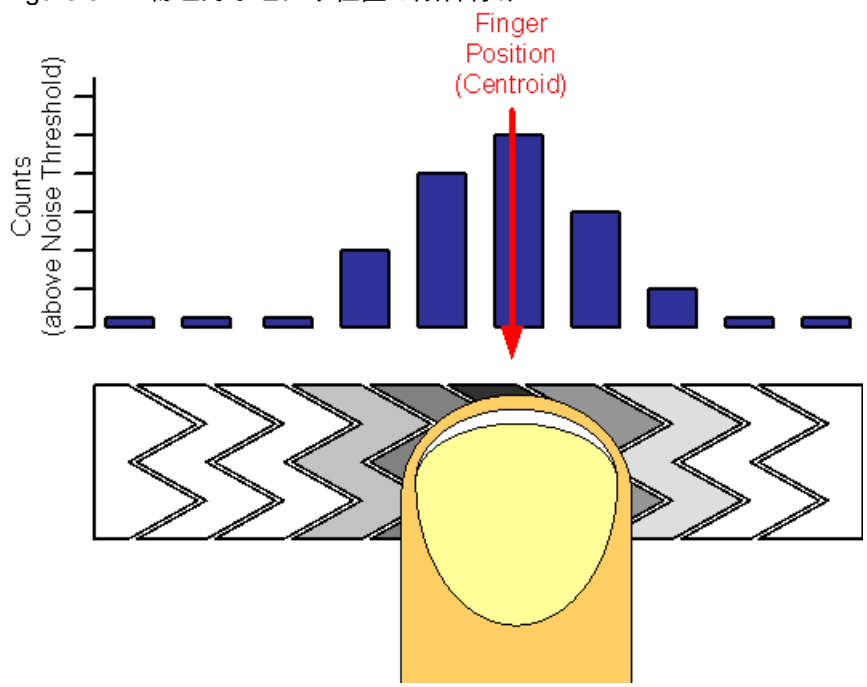
Figure 2.　プリチャージ スイッチと駆動波形を持つアナログ バス



スライダ

スライダは、漸次的調整を必要とする制御に使用します。たとえば、照明管理 (調光装置)、音量管理、グラフィック イコライザ、速度制御などがあります。これらのセンサは互いに、機械的に近接しています。1 つのセンサの作動は、物理的に近接するセンサの部分的な作動につながります。スライダの実際の位置は、作動したセンサ セットの重心位置を計算することによって判断できます。CSD ウィザードは、特定の順番を持つスライダ グループを設定することで、スライダに対応します。センサ スライダの実質的な下限は 5 で、上限は、選択した PSoC デバイスで利用できるセンサ数になります。

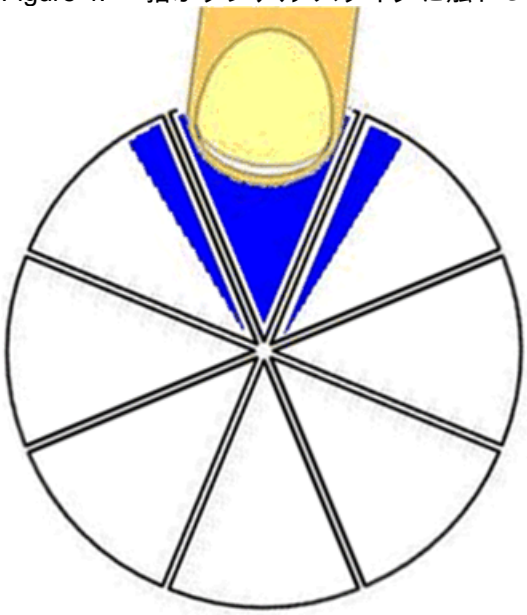
Figure 3. 物理的なセンサ位置の順番付け



スライダの半分で強い信号を近接検知すると、残り半分に同レベルの信号を生じますが、結果は分散してしまいます。検知アルゴリズムは、強い近隣信号セットを検索して、スライダ位置を特定します。

ラジアル スライダ

Figure 4.　指がラジアル スライダに触れる



CSD UM では、リニアおよびラジアルという 2 種類のスライダを利用できます。ラジアル スライダは、リニア スライダと似ています。リニア スライダには始点と終点がありますが、ラジアル スライダにはありません。スライダに触れると、重心計算アルゴリズムが、スイッチのセンサ数を現在のスイッチの左右で考慮します。ラジアル スライダにダイプレックスは対応していません 。

CSD UM には、ラジアル スライダをサポートする 2 つの API 関数があります。最初の関数 CSD_wGetRadiaPos() は重心位置を返し、2 つ目の関数 CSD_wGetRadialInc() が分解能単位で指のシフトを返します。指が時計回り方向に移動すると、正のオフセットとなります。

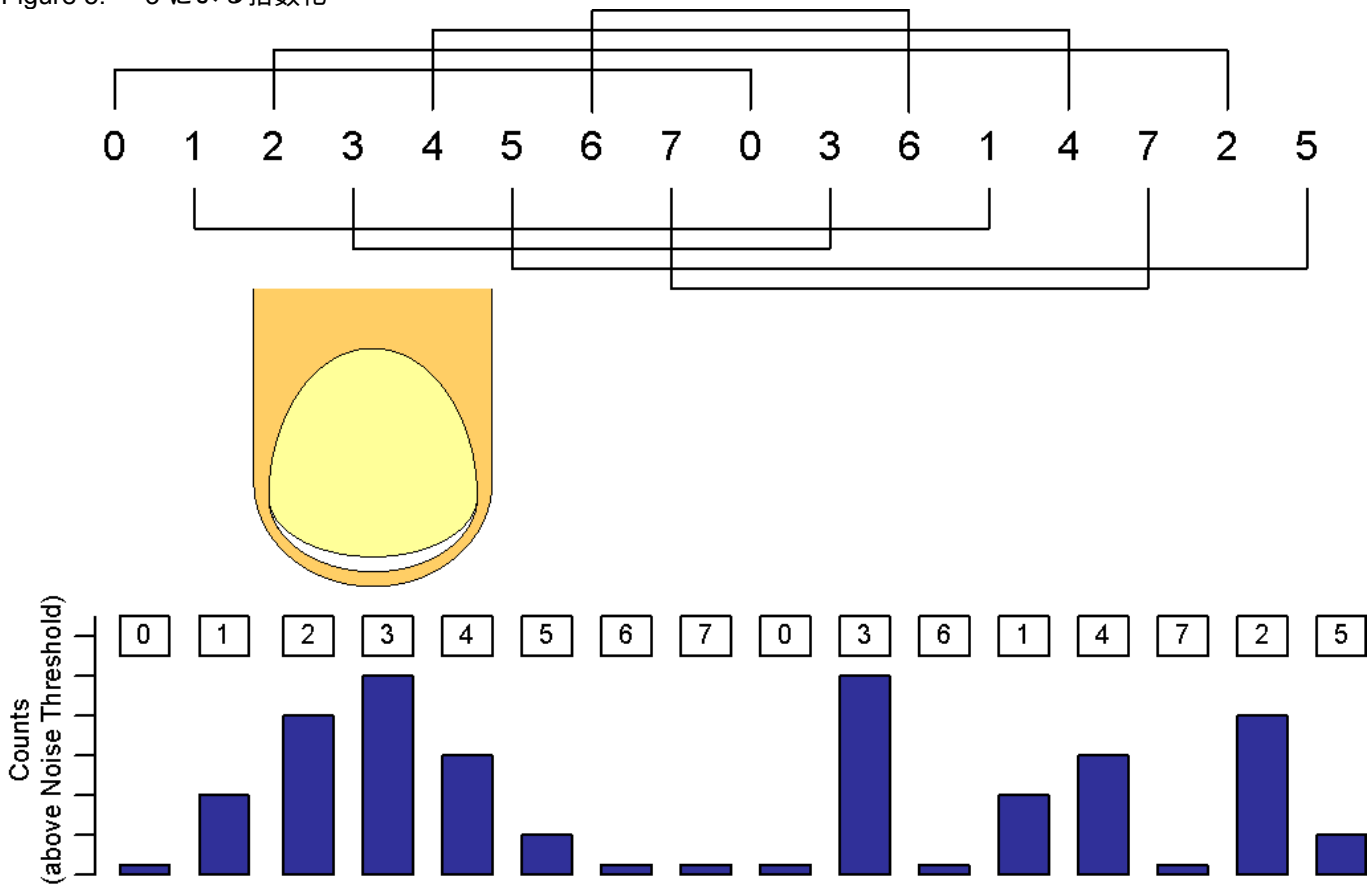
基準点 (0) は、最初のセンサの中央に位置しています。リニア スライダおよびラジアル スライダの分解能は、3000 に制限されています。

ダイプレックス

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 (数字が小さい) 半分は、CSD ウィザード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、ウィザードのアルゴリズムで自動的にマッピングされ、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、プリント回路基板へのマッピングは慎重に行ってください。

物理センサ位置の後の半分に対して順序を決める方法は数多くあります。最も簡単なものは、上半分でセンサを並べ、全ての偶数センサ、全ての奇数センサで指数化する方法です。他の方法では、別の値によってセンサを指数化します。このユーザ モジュールで選択された方法は 3 による指数化です。

Figure 5. 3 による指数化



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 レイアウトによっては、一部のセンサ ペアでルートが長くなる可能性があります。ダイプレックス センサ指数表は、CSD ウィザードのダイプレックスを選択することによって自動的に作成されます。以下の表に、異なるスライダ セグメント カウントのダイプレックス シーケンスを示します。

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

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
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

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
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
58	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,0,3,6,9,12,15,18,21,2, 27,1,4,7,10,13,16,19,22,25,28,2,5,8,11,14,17,20,23,26
60	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,0,3,6,9,12,15,18,21,24,27,1,4,7,10,13,16,19,22,25,28,2,5,8,11,14,17,20,23,26,29
62	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,0,3,6,9,12,15,18,21,24,27,30,1,4,7,10,13,16,19,22,25,28,2,5,8,11,14,17,20,23,26,29
64	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,0,3,6,9,12,15,18,21,24,27,30,1,4,7,10,13,16,19,22,25,28,31,2,5,8,11,14,17,20,23,26,29
66	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,0,3,6,9,12,15,18,21,24,27,30,1,4,7,10,13,16,19,22,25,28,31,2,5,8,11,14,17,20,23,26,29,32
68	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,0,3,6, 9,12,15,18,21,24,27,30,33,1,4,7,10,13,16,19,22,25,28,31,2,5,8,11,14,17,20,23,26,29,32
70	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,0,3,6,9,12,15,18,21,24,27,30,33,1,4,7,10,13,16,19,22,25,28,31,34,2,5,8,11,14,17,20,23,26,29,32
72	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,0,3,6,9,12,15,18,21,24,27,30,33,1,4,7,10,13,16,19,22,25,28,31,34,2,5,8,11,14,17,20,23,26,29,32,35
74	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,0,3,6,9,12,15,18,21,24,27,30,33,36,1,4,7,10,13,16,19,22,25,28,31,34,2,5,8,11,14,17,20,23,26,29,32,35
76	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,0,3,6,9,12,15,18,21,24,27,30,33,36,1,4,7,10,13,16,19,22,25,28,31,34,37,2,5,8,11,14,17,20,23,26,29,32,35
78	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,0,3,6,9,12,15,18,21,24,27,30,33,36,1,4,7,10,13,16,19,22,25,28,31,34,37,2,5,8,11,14,17,20,23,26,29,32,35,38
80	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,0,3,6,9,12,15,18,21,24,27,30,33,36,39,1,4,7,10,13,16,19,22,25,28,31,34,37,2,5,8,11,14,17,20,23,26,29,32,35,38
82	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,0,3,6,9,12,15,18,21,24,27,30,33,36,39,1,4,7,10,13,16,19,22,25,28,31,34,37,40,2,5,8,11,14,17,20,23,26,29,32,35,38
84	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,0,3,6,9,12,15,18,21,24,27,30,33,36,39,1,4,7,10,13,16,19,22,25,28,31,34,37,40,2, 5,8,11,14,17,20,23,26,29,32,35,38,41

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
86	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,0,3,6,9,12,15,18,21,24,27,30,33,36,39,42,1,4,7,10,13,16,19,22,25,28,31,34,37,40,2,5,8,11,14,17,20,23,26,29,32,35,38,41
88	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,0,3,6,9,12,15,18,21,24,27,30,33,36,39,42,1,4,7,10,13,16,19,22,25,28,31,34,37,40,43,2,5,8,11,14,17,20,23,26,29,32,35,38,41
90	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,0,3,6,9,12,15,18,21,24,27,30,33,36,39,42,1,4,7,10,13,16,19,22,25,28,31,34,37,40,43,2,5,8,11,14,17,20,23,26,29,32,35,38,41,44
92	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,0,3,6,9,12,15,18,21,24,27,30,33,36,39,42,45,1,4,7,10,13,16,19,22,25,28,31,34,37,40,43,2,5,8,11,14,17,20,23,26,29,32,35,38,41,44

補間とスケーリング

多くの場合、スライド式センサとタッチパッド用のアプリケーションでは、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工エリアは、しばしば 1 個のセンサより大きくなっています。

重心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、センサの位置が有効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、ノイズ閾値より大きい近隣信号を使用して重心を算出します。重心は (通常)、2 つから 8 つのセンサを使用して、次の数式で計算されます。

Equation 1

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

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、重心を特定の分解能の形で報告するには、計算されたスカラー量を重心に掛けます。1 つの計算で補間とスケーリングを組み合わせ、その結果を直接、希望のスケールで報告する方が効率的です。これは高レベル API で行う処理です。

スライダセンサ数と分解能は、CSD ウィザードで設定します。スケーリング値は、ウィザードで計算され、整数ではない値として保存されます。

重心の分解能の乗数は、3 バイトに含まれ、それぞれのビット定義は以下になります。

分解能乗数 MSB								
ビット	7	6	5	4	3	2	1	0
乗数	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}
分解能乗数 ISB								
乗数	128	64	32	18	16	8	4	2
分解能乗数 LSB								
乗数	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256

分解能はこの数式を用いて算出されます。

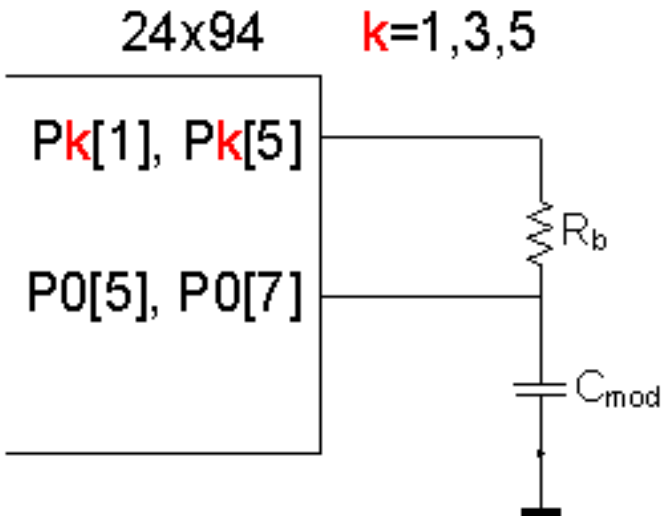
分解能 = (センサ数 – 1) x 乗数

重心は 24-bit 符号無し整数で保持され、その分解能はセンサ数と乗数の関数です。

フィードバックコンポーネント選択のガイドライン

ユーザ モジュールには、外付け変調コンデンサ C_{mod} とモジュレータ フィードバック レジスタ R_b が必要です。コンデンサは、P0[5]、P0[7] ポート ピンと Vss アースに接続できます。フィードバック レジスタ R_b は、ポート ピン P1[1]、P1[5]、P3[1]、P3[5]、P5[1]、P5[5] およびコンデンサ ピンに接続できます。ピンは、ユーザ モジュールのパラメータ設定により選択されます。変調器コンポーネントの接続用に選択されているピンは、これ以外の目的で使用しないでください。

Figure 6. 外部コンポーネントの接続



変調コンデンサ用に推奨されている値は、4.7 ~ 47.0 nF です。最適な静電容量は、最大 SNR を得るための実験を行うことで決定することができます。ほとんどの場合、5.6 ~ 10.0 nF の値で良い結果が得られます。フィードバック レジスタを選択した後で、最良の SNR を得るためには、いくつかのコンデンサ値で実験してみるとよいでしょう。セラミックのコンデンサを使用するよう強く推奨します。温度静電容量係数は重要ではありません。レジスタ値は、総センサ静電容量 C_s によって異なります。レジスタ値は、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モニタする。
- 選択されたスキャン分解能で、フルスケール値より約 30% 小さい値が最大値になるように抵抗値を選択する。抵抗値が増えると、Raw カウントは増えます。

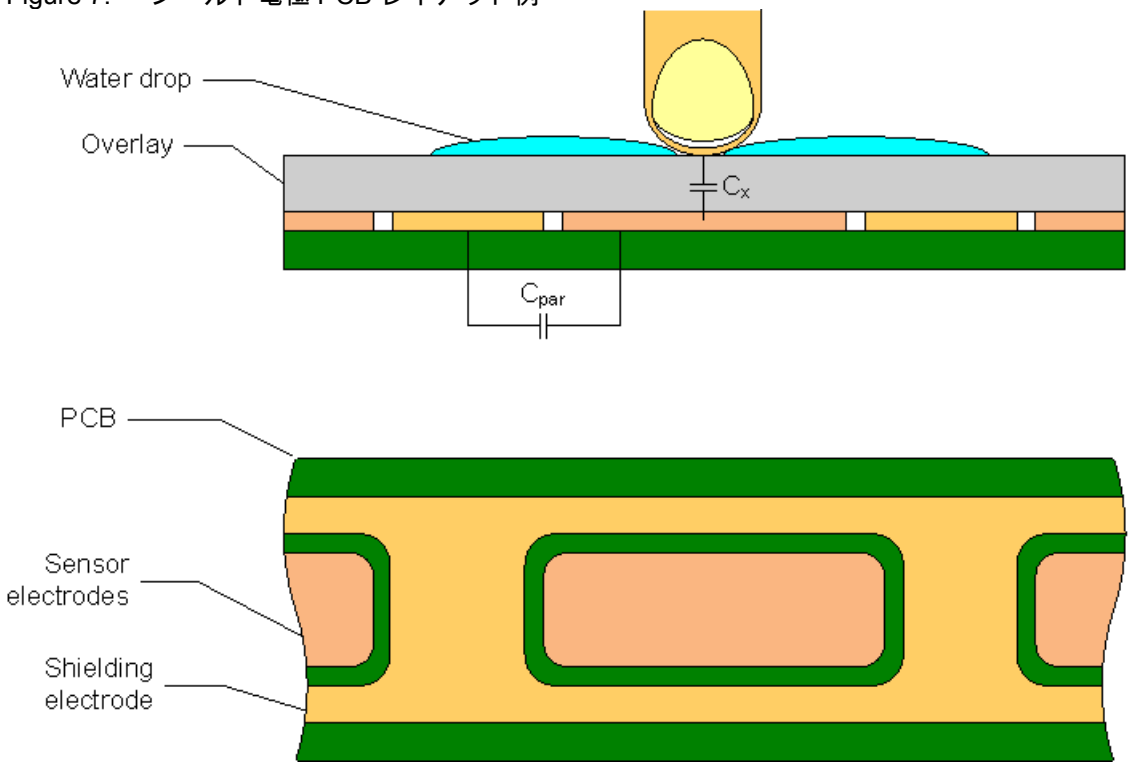
標準値は 500Ω ー 10 kΩ で、センサの静電容量とプリチャージ スイッチの動作周波数によって変わります。CY3214 評価ボードを使用している場合は、2.0kΩ から開始できます。

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動作の信頼性が要求されます。ホワイト ノイズ、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、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。水膜がデバイスの絶縁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、寄生静電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、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逆を発生させるため、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は有効です。これにより、ハイレベルソフトウェアの API 作業が簡単になります。CSD ユーザ モジュールは、別のシールド電極出力に対応しています。

Figure 7.　シールド電極 PCB レイアウト例



前の図は、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の影響を阻止し、同時に浮遊静電容量を低減するため、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、C_{par} が増え、変調器の電流が低減することがあります。実際のテストでは、変調器の基準電圧は API によって増加でき、水滴による生カウントはゼロに近いか、わずかにマイナスとなっています。これは、適切な変調器基準値を選択することによって達成できます。

シールド電極は、空いている Row 出力バスのいずれにも接続できます。Row LUT では、**A** を選択してください。

シールド電極は、迂回可能ないずれの PSoC ピンにも接続できます。駆動モードを **Strong Slow** に設定し、グランド ノイズと輻射を軽減します。また、スルー制限レジスタも、PSoC デバイスとシールド電極の間に接続できます。

クロック源

クロック源は、検知コンデンサ上でスイッチの制御に使用されます。ユーザ モジュールは、プリチャージ スイッチのクロック源として、次の 3 つの選択オプションをサポートしています。

- 16-bit 擬似ランダム系列発生器 (PRS16)
- 8-bit PRS 源
- プリスケーラを持つ 8-bit PRS 源

必要な構成は、ユーザ モジュールを最初に選択する際に選択してください。この選択を後で変更するには、[Interconnect View (相互接続ビュー)] で CSD ユーザ モジュールのアイコンを右クリックし、**[User Module Selection Options (ユーザ モジュール選択オプション)]** を選択します。

PRS16 構成は、クロック源として PRS16 モジュールを使用します。PRS16 源は、スペクトル拡散を実現し、外部ノイズ源に対する耐性を提供します。さらに、拡散スペクトルクロックを使用した設計では、電磁放射レベルが低くなります。アプリケーションが EMC/EMI テスト合格を目的としている場合、または厳しい環境で信頼性の高い動作が要求される場合には、PRS16 構成を推奨します。以下の表で、3 つの構成を比較します。

構成	動作周波数	EMC 耐ノイズ性
PRS16	拡散スペクトル。平均は $F_{IMO}/4$、ピークは $F_{IMO}/2$。	高。高感度ポイントは、PRS シーケンスの繰り返し周期の倍数と、PRS 基本周波数 F_{IMO}。
PRS8	拡散スペクトル。平均は $F_{IMO}/4$、ピークは $F_{IMO}/2$。	中。PRS 繰り返し期間が短くなるため、より多くのポイントで影響を受ける。
プリスケーラを持つ PRS8	調整可能な拡散スペクトル、平均は $F_{IMO}/8$ ～ $F_{IMO}/1024$、ピークは $F_{IMO}/4$ ～ $F_{IMO}/512$	中。PRS 繰り返し期間が短くなるため、より多くのポイントで影響を受ける。

変調器のリファレンス源

コンパレータのリファレンス源は、コンパレータのリファレンス電圧を形成するために使用されます。リファレンス電圧値は、検出感度を決定します。変調器のリファレンスは、連続時間ブロック内にある抵抗分圧器を使って形成されます。リファレンス値は、UM パラメータ、または API の呼び出しで変更することができます。

DC および AC の電気的特性

Table 1. 電源電圧

パラメータ	最小値	標準値	最大値	単位	テスト条件とコメント
値	2.7	5.0	5.25	V	

Table 2. ノイズ

パラメータ[a]	最小値	標準値	最大値	単位	テスト条件 (Vdd = 3.3V、SysClk = 24 MHz、CPU クロック = 6 MHz、ベースライン値 >= 分解能最大値の 70%)
ノイズ値 ピーク - ピーク		0.2		% (ノイズ値)/ (ベースライン値)	分解能 = 16
ノイズ値、 ピーク - ピーク		1		% (ノイズ値)/ (ベースライン値)	分解能 = 14
ノイズ値、 ピーク - ピーク		10		% (ノイズ値)/ (ベースライン値)	分解能 = 10

a. SNR は、スキャン速度が落ち、ベースライン値が増加するにしたがって増えます。

Table 3. 消費電力

電源電圧	最小値	標準値	最大値	単位	テスト条件とコメント
アクティブ電流		10		mA	スキャン中の平均電流、8 センサ
スタンバイ電流		250		µA	スキャン速度 = 超高速、分解能 = 9、100 ms レポートレート、8 センサ
		1.6		mA	スキャン速度 = 高速、分解能 = 12、100 ms レポートレート、8 センサ
スリープ / ウェイク電流		10		µA	1 s レポート レート、1 センサ

特性グラフ

テスト条件：分解能は 12 ビットに設定され、データは、CY3214 リビジョン A 評価ボードを使って収集されました。詳しくは、注を参照してください。

Figure 8. 異なるスキャン速度、PRS16 構成での電源電圧に対する Raw カウント

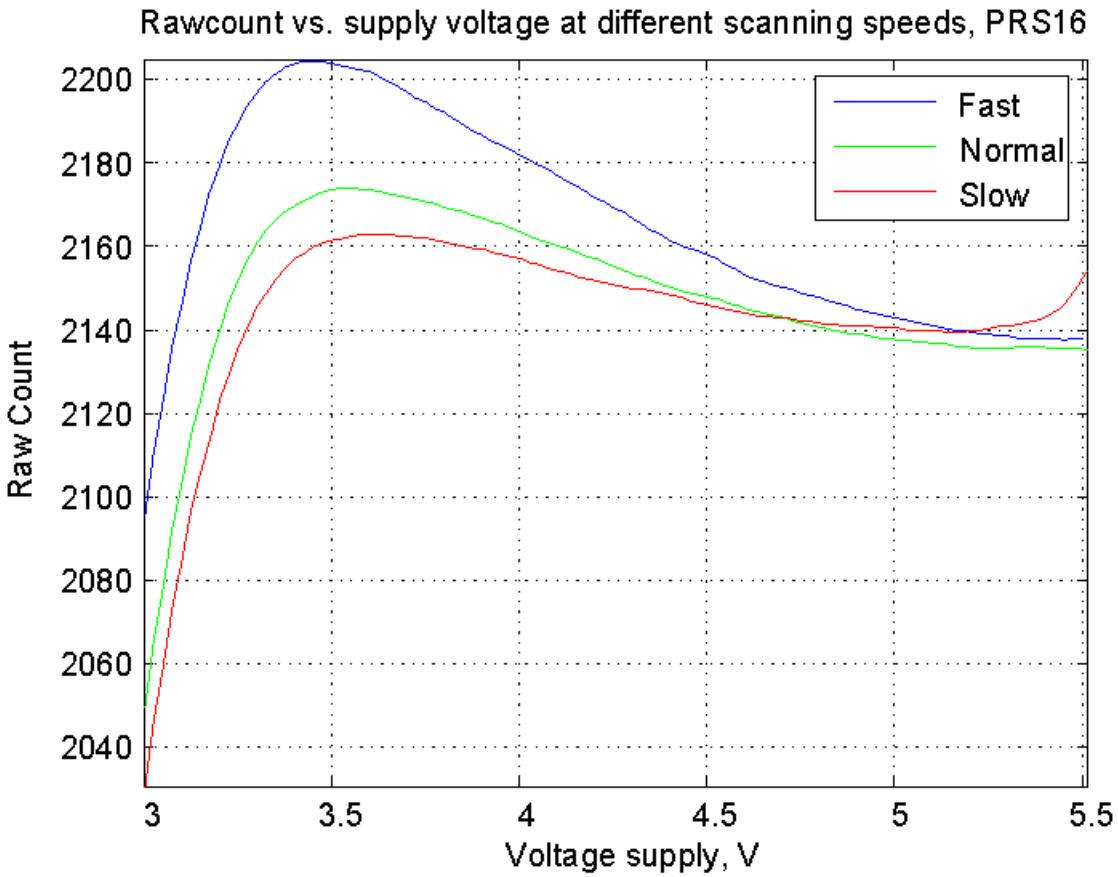
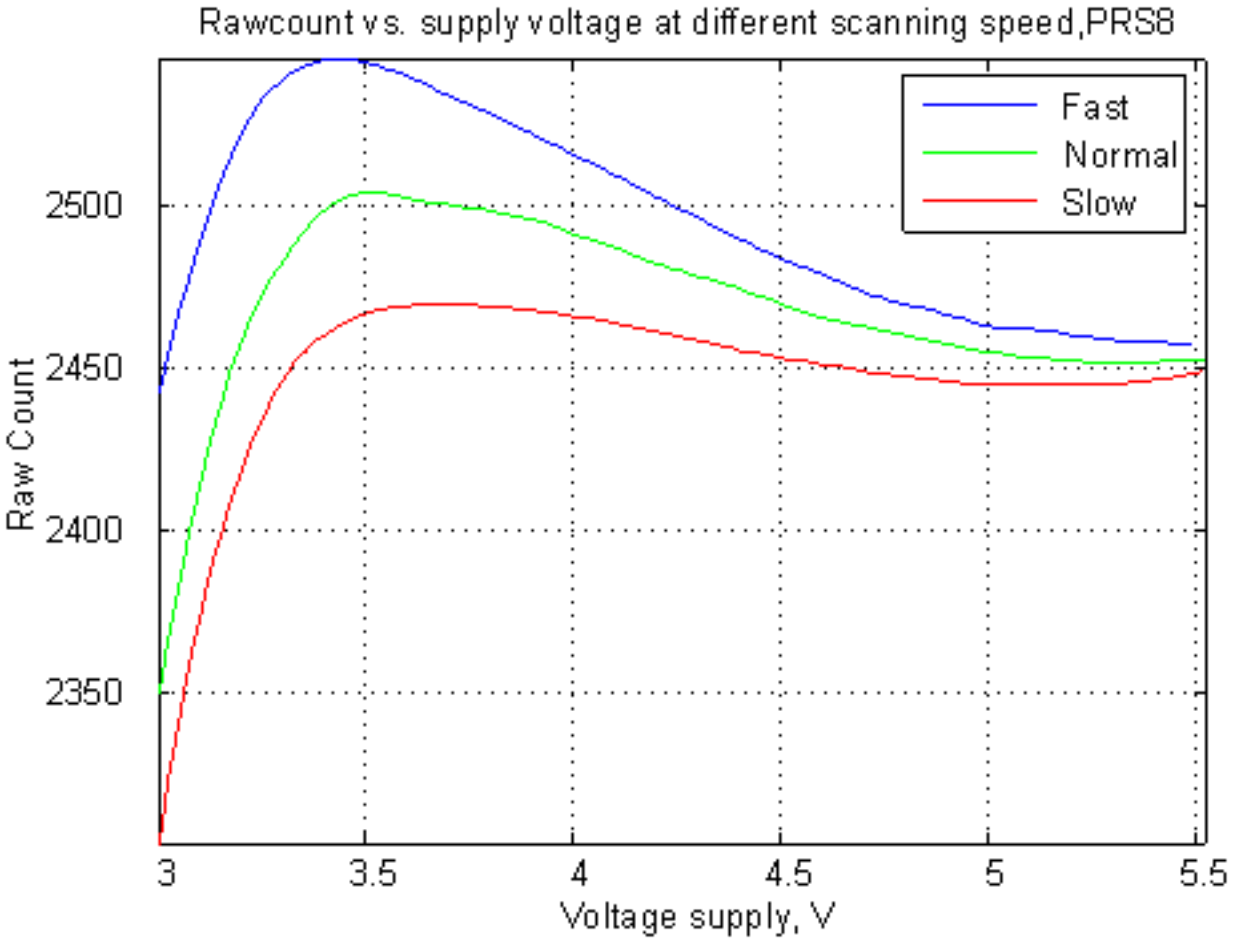


Figure 9.　異なるスキャン速度、PRS8 構成での電源電圧に対する Raw カウント



注 曲線を比較すると、スキャン速度が遅い場合に、Raw カウントの変化が小さくなることがわかります。幅広い電源電圧で安定した動作が必要な用途では、スキャン速度を遅くしてください。

Figure 10. 異なるスキャン速度、PRS16 構成での温度に対する Raw カウント

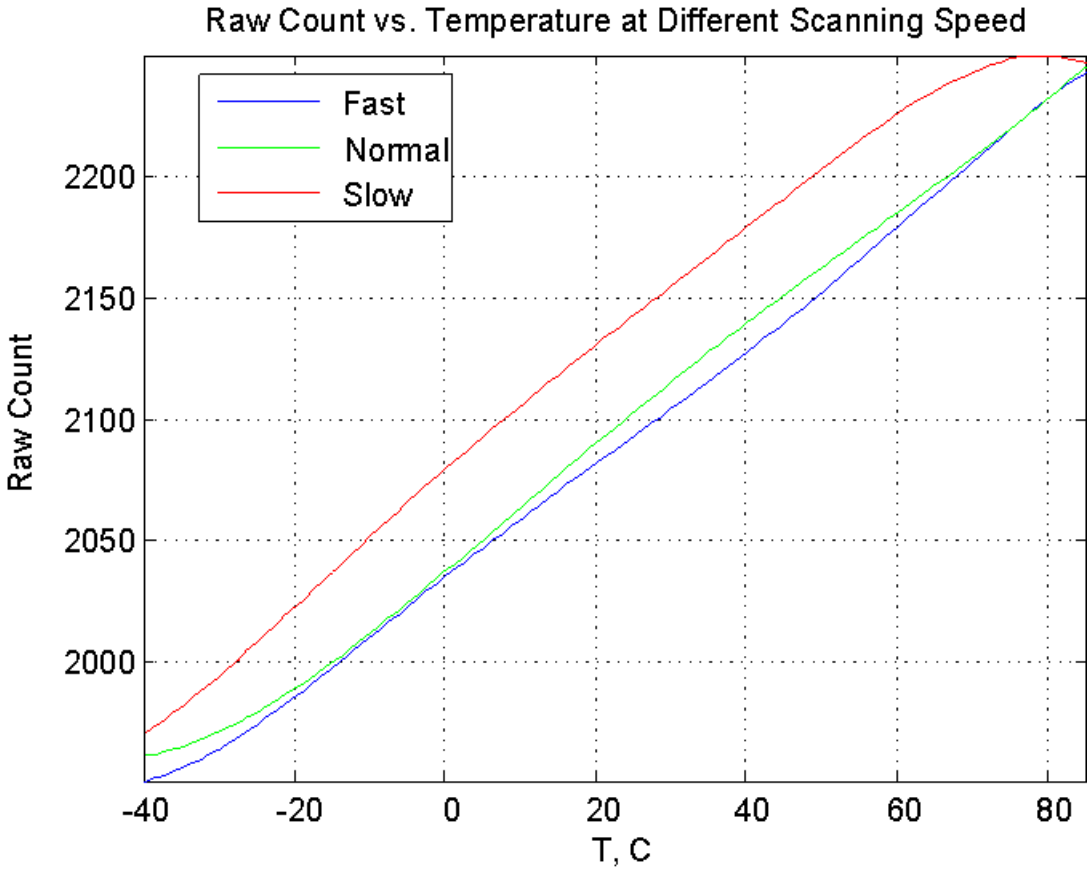
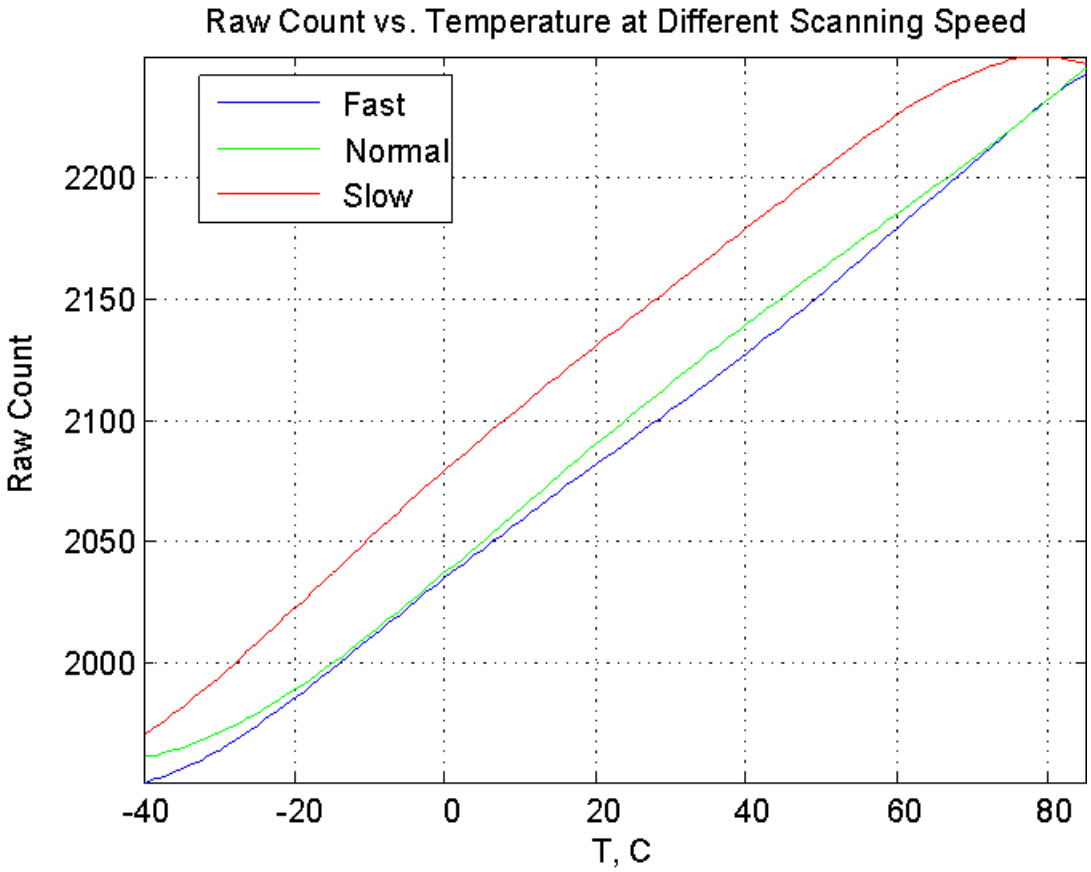
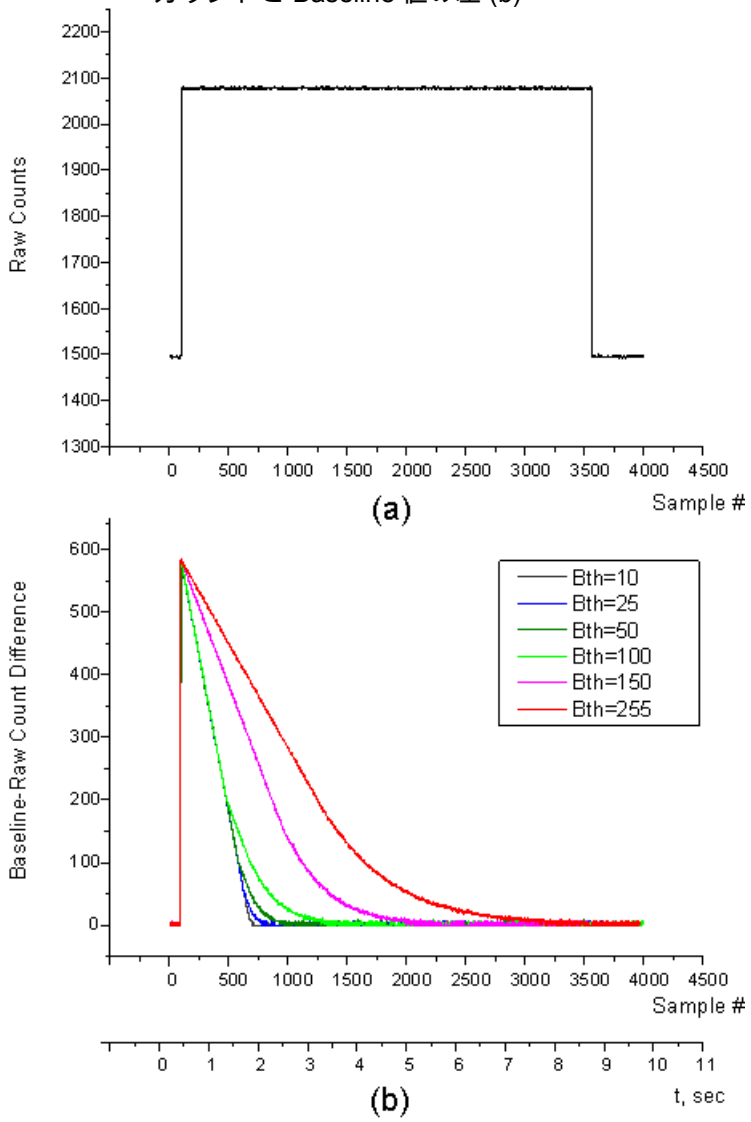


Figure 11. 異なるスキャン速度、PRS8 構成での温度に対する Raw カウント



テスト条件：12-bit 分解能、電源電圧 5.0 V。センサを持つ完成ボード (CY3214 PSoC 評価ボード) が、テスト中に温度調整ルームに配置されました。

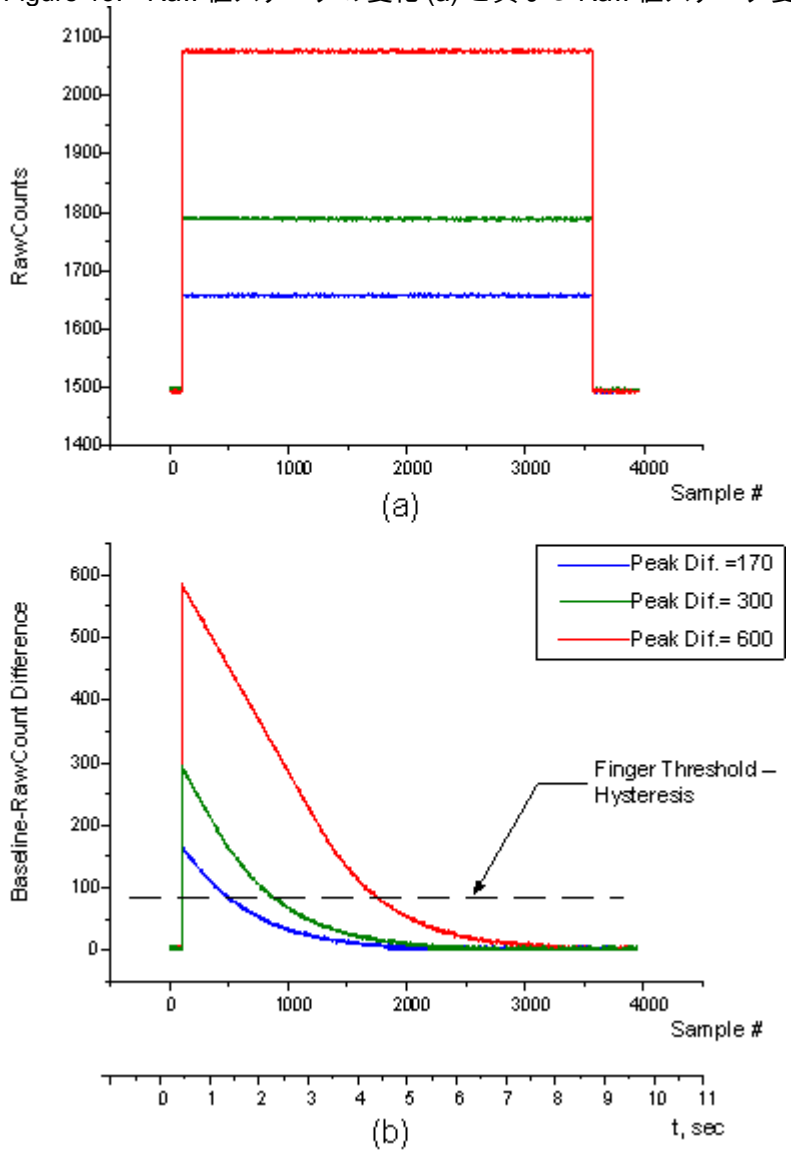
Figure 12. Raw カウントステップの変化 (a) と異なる BaselineUpdate 閾値 （ Bth ）パラメータ値 に対する Raw カウントと Baseline 値の差 (b)



テスト条件 : 12-bit 分解能、SensorsAutoreset = 有効、センサアレイスキャンとデータ転送時間の合計は約 2.5 ms。

注 BaselineUpdate 閾値を上げると、差の減衰速度が下がり、最大ボタンタッチ検知時間が長くなります。

Figure 13. Raw 値ステップの変化 (a) と異なる Raw 値ステップ変化の値に対する Raw 値と Baseline 値の差 (b) 。



テスト条件 : 12-bit 分解能、SensorsAutoreset = 有効、センサアレイスキャンとデータ転送時間の合計は約 2.5 ms。パラメータの BaselineUpdate 閾値は 255 に設定。

注 Raw 値ステップ値が大きいと、FingerThreshold-Hysteresis 値以下の差分を小さくする時間が長くかかるため、センサのオートリセットの間隔が長くなります。

[+] Feedback

配置

ユーザ モジュールのブロックは、ユーザ モジュールがインスタンス化されると自動的に配置されます。他の配置は利用できません。変調器コンパレータは、ACB01 連続時間ブロックに配置されます。異なる UM 構成では、1 ～ 3 個のデジタル ブロックが使用されます。以下の表に、使用されるデジタルリソースを示します。

構成	使用されるデジタル ブロック
PRS16	DBB00、DBB01、DCB0
PRS8	DBB00
プリスケーラを持つ PRS8	DBB00。DBB01

使用されないアナログ ブロックおよびデジタル ブロックは、ほかの目的で利用できます。すべての UM 構成では、ハードウェア デシメータが使用されます。

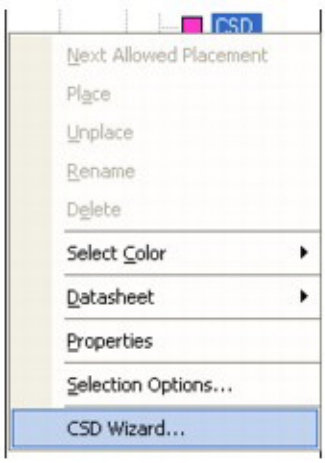
LCD や I2CHW などの特定のピンを必要とするユーザ モジュールは、CSD ユーザ モジュールのポート ピン接続を確立する前に配置しなければなりません。構成の選択は、ウィザードを開いたときに反映されます。

静電容量式センサの接続を配置する場合、P1[0] と P1[1] は避けてください。これらのピンは、そのデバイスのプログラミングに使用され、センサの検出感度とノイズに影響を与える過度のルーティング静電容量を持っている可能性があります。

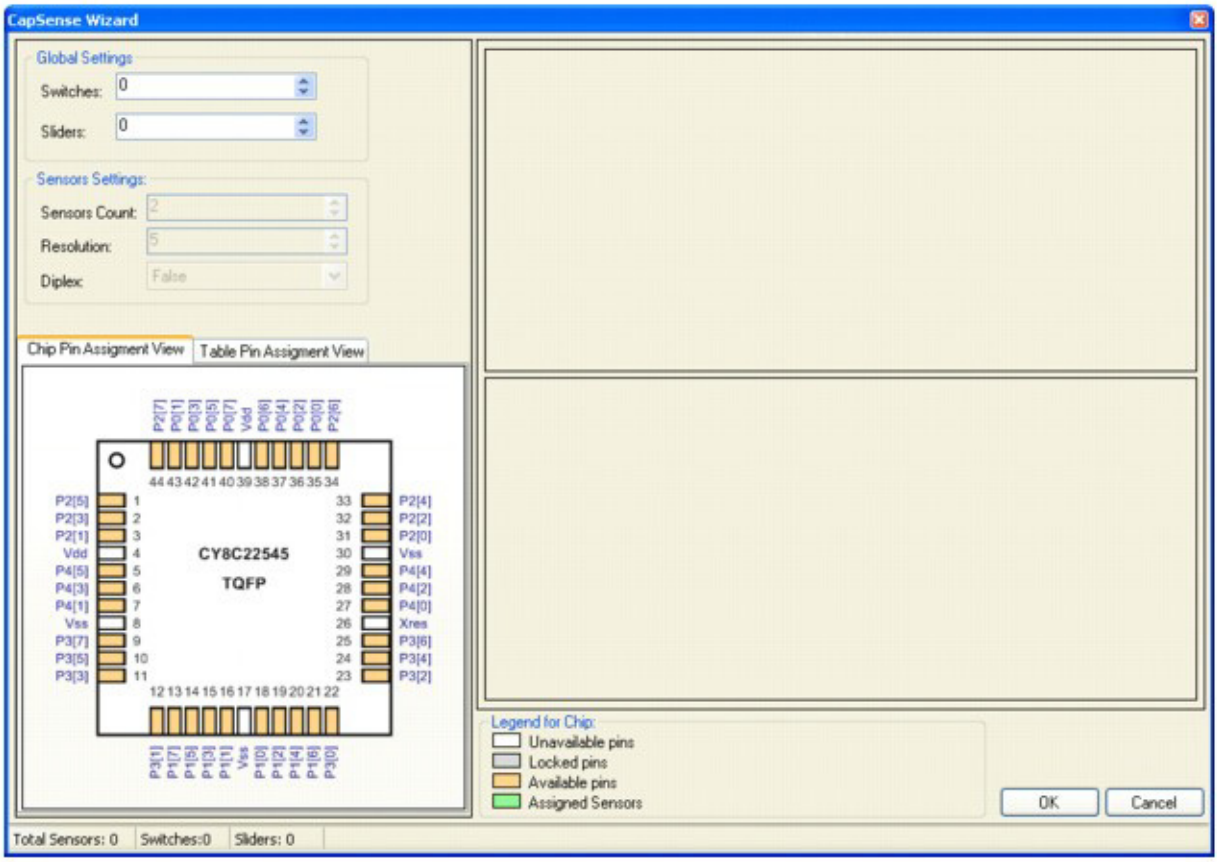
ウィザード

CSD ウィザードを使って、CapSense ボタン、スライダ、近接検知センサのピン配列を設定します。ドラッグ アンド ドロップ インタフェースを使って、構成を選択し、ボタンとセグメントを割り当てます。

1. ウィザードにアクセスするには、デバイス エディタの Interconnect View で CSD の任意のブロックを右クリックし、次に [CSD Wizard] (CSD ウィザード) を左クリックします。



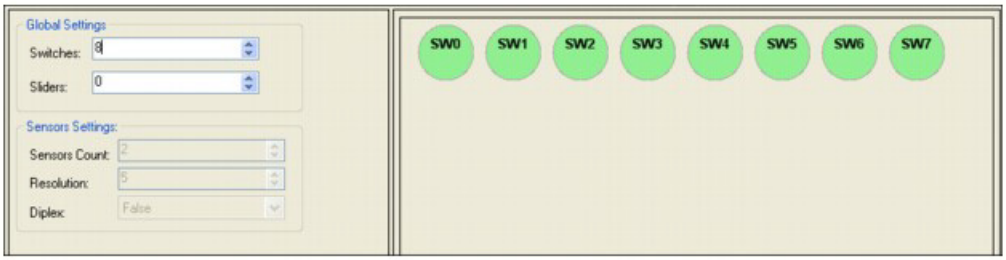
2. ウィザードが開き、センサの数とスライダセンサの数がボックスに示されます。



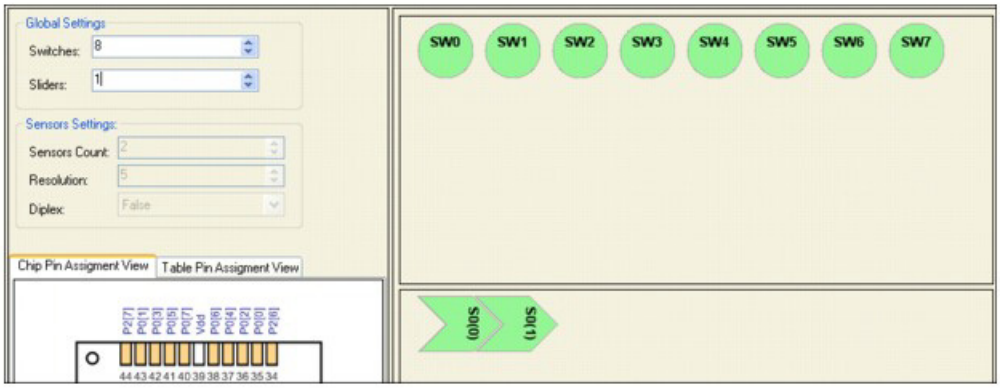
ウィザードのピン凡例

- 白 – このピンは CapSense の入力に使用できません。
- グレー – このピンはロックされています。これには 2 つの原因が考えられます。その 1 つ目は、LCD や I^2C などの別のユーザ モジュールが、そのピンを使用している場合です。2 つ目は、ピンの名前がデフォルトから変わった場合です。ピン名をデフォルトに戻すには、Pinout ビューでそのピンの表示を広げ、**Select** メニューで **Default** を選択します。これで、ピンはウィザードで割り当てられるようになりました。
- オレンジ – このピンは割り当て可能です。
- グリーン – このピンは CapSense 入力に割り当てられています。

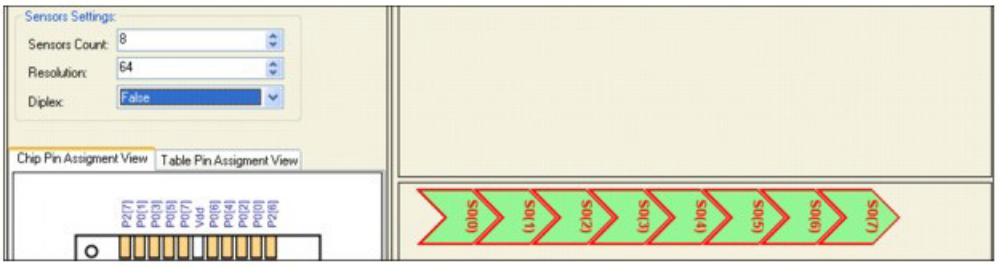
3. 独立センサ数を入力します。センサ数は、使用できるピン数に制限されています。



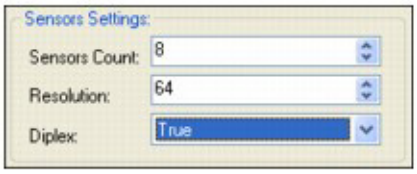
4. スライダ数を入力します。



5. 各スライダのセンサ数を入力します。スライダセンサ中の現実的な最低センサ数は 5 で、最大数はピン数によって制限されます。

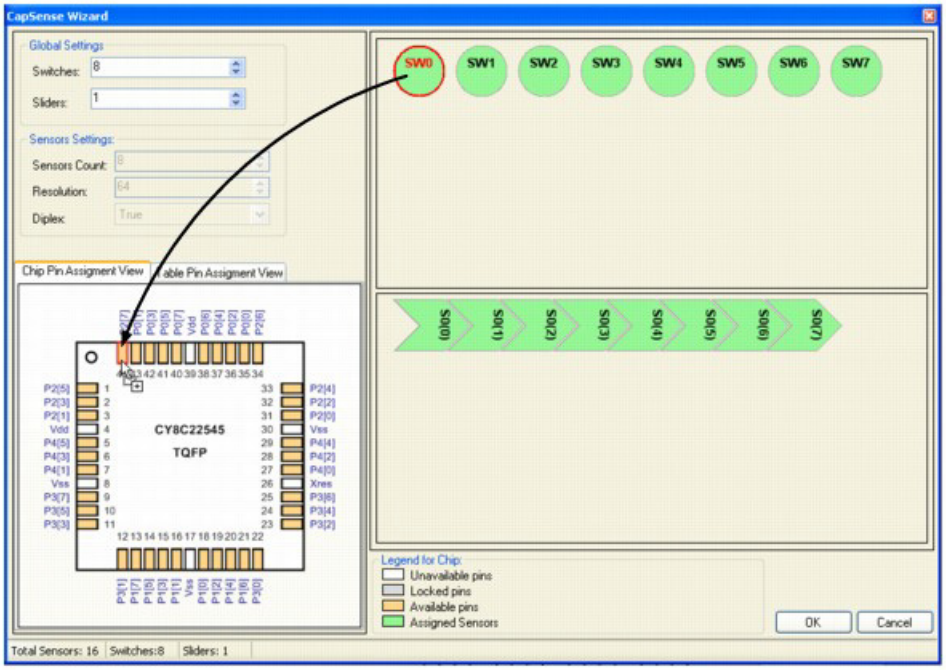


6. 出力の分解能を入力します。最低値は 5 です。最大値は、ダイプレックス スライダの場合、(センサに使用されるピン数 – 1) x 2^{16} – 1 または (2 x センサに使用されるピン数 – 1) x 2^{16} -1 です。



7. 必要に応じて、ダイプレックスを選択します。これにより、センサ用に選択されたピンを、基板上で 2 倍の数のセンサ位置にマッピングできます。上図では、ダイプレックスセンサの最初の半分だけが示されています。残りの半分は、前述の 「ダイプレックス」の項での説明の通り、自動的にマッピングされます。ピン接続に関しては、「ダイプレックス」の項のダイプレックス表を参照してください。

8. センサを左クリックし、利用可能なピンにドラッグします。ポートピンは選択するとグリーンになり、使用不可となります。ポート ピンからセンサをドラッグして外すと、センサ割り当てを変更できます。



9. 他のセンサでも同じ操作を繰り返します。
10. 個々のスライダセンサの物理ポートピンへのマッピングは、個々のセンサの場合と同じです。
11. [OK] をクリックしてデータを受け入れ、PSoC Designer に戻ります。

これでセンサの配置が完了しました。デバイス エディタ ウィンドウを右クリックし、[Refresh] (再表示) を選択すると、ピン接続が更新されます。

ユーザ モジュールのパラメータを選択し、アプリケーションを生成します。必要に応じて、サンプルプロジェクトを使用することもできます。

CSD ウィザードで数値を入力する場合は、まず古い値を削除してから、新しい値を入力してください。編集ボックスには、カーソルは表示されません。

ピン割り当てを変更するには、割り当てられているピンにカーソルを合わせてクリックし、それをスイッチボックスの外までドラッグ アンド ドロップします。これでこのピンは割り当てから外され、他に割り当てることが可能になります。

ウィザードを完了したら、[Generate Application] (アプリケーションの生成) をクリックします。入力したセンサ数、ピン割り当て、ダイプレックス、分解能に基づいて、一連の表が生成されます。これらの表は CSD_Table.asm に保存されています。

センサ表

センサ表は各センサに対して 2 バイトのエントリから構成されています。第 1 バイトはポート番号で、第 2 バイトはそのビットのビットマスクです (ビット番号ではありません)。表には、すべての独立したセンサ、次に各センサが順番に列挙されています。次に、10 個のセンサを含む表の例を示します。

```
CSD_Sensor_Table:
_CSD_Sensor_Table:
    dw    0x0001    // Port 0 Bit 0
    dw    0x0002    // Port 0 Bit 1
    dw    0x0004    // Port 0 Bit 2
    dw    0x0008    // Port 0 Bit 3
    dw    0x0010    // Port 0 Bit 4
    dw    0x0101    // Port 1 Bit 0
    dw    0x0102    // Port 1 Bit 1
    dw    0x0104    // Port 1 Bit 2
    dw    0x0108    // Port 1 Bit 3
    dw    0x0110    // Port 1 Bit 4
```

この表は CSD_wGetPortPin() ルーチンで使用されます。

グループ表

グループ表は、ボタンセンサやスライダのグループを定義します。各スライダにつきエントリが 1 つ、フリーボタンセンサにもエントリが 1 つあります。最初のエントリは必ずフリーセンサです。各エントリは 6 バイトです。第 1 バイトはセンサ表のインデックスです。第 2 バイトはグループ内のセンサ数です。第 3 バイトは、スライダがダイプレックスであるかどうかを示します (4 はダイプレックス、0 は非ダイプレックス)。第 4、第 5、第 6 バイトは固定ポイント乗数で、スライダの計算された重心が乗算されて、CSD ウィザードで望ましい分解能が達成されます。

```
CSD_1_Group_Table:
_CSD_1_Group_Table:
; Group Table:
;    Origin    Count    Diplex?    DivBtwSw(wholeMSB, wholeLSB, fractByte)
 db   0x0,      0x5,     0x00,      0x00,      0x00,      0x00 ; Buttons
 db   0x5,      0x5,     0x3,       0x0,       0x0,       0x71 ; Slider 1
```

ダイプレックス表

ダイプレックス表のスキャンオーダデータは、スライダでダイプレックスされている場合に、グループを対象として作成されます。これ以外の場合は、ラベルが作成されますが、データは存在しません。この表は、各スライダのセンサマッピングと、各スライダによるそれぞれの表のリファレンスという 2 つの部分から構成されています。ここでは、5 つのセンサ スライダを使った典型的な例を示します。

```
DiplexTable_0:
; This group is not a diplexed slider
DiplexTable_1:
    db    0,1,2,3,4,0,3,1,4,2    // 5 switch slider


 CSD_1_Diplex_Table:
_CSD_1_Diplex_Table:
    db >DiplexTable_0, <DiplexTable_0
    db >DiplexTable_1, <DiplexTable_1
```

パラメータおよびリソース

指の閾値

この閾値は、各ボタンセンサの状態を判断するために使用します。1 つでもセンサがアクティブな場合、bIsAnySensorActive() 関数は 1 を返します。すべてのセンサがオフの場合、bIsAnySensorActive() 関数は 0 を返します。

指検知閾値は、すべてのセンサとスライダに適用されます。個々のセンサ (スライダグループに属さないセンサ) では、これらの閾値は変数で、baBtnFThreshold[] アレイで提供されます。SetDefaultFingerThresholds() 関数を使用すると、閾値をデバイス エディタで設定されているデフォルト値に設定できます。個々のセンサの感度を調整するには、各センサの baBtnFThreshold[] 値を変更します。(このバイトアレイのサイズは、個々の実装センサ数と同じです。)

可能な値の範囲は 5 ～ 255 です。

ノイズ閾値

個々のセンサでは、この閾値を上回るカウント値は Baseline 値を更新しません。スライダセンサでは、この閾値を下回るカウント値は重心の計算に加えられません。可能な値は 5 ～ 255 です。

BaselineUpdate 閾値

新しい Raw カウント値が現在の Baseline 値を上回っており、その差がノイズ閾値を下回る場合 (センサの Autoreset パラメータは無効にセットされているとき)、現在の Baseline 値と Raw 値の差はバケツに集められたように加算されます。バケツが満杯になると、基準値はある値だけ増分し、バケツは空になります。このパラメータは、Baseline 値が増分するためにバケツが達成しなければならない閾値を設定します。可能な値は 0 ～ 255 です。パラメータ値が大きいときは、Baseline 値の更新速度を遅くします。より頻繁な Baseline 値の更新が必要なときは、このパラメータを小さくします。

LowBaselineReset

LowBaselineReset パラメータは、NegativeNoiseThreshold パラメータと共に作動します。サンプルカウント値が (Baseline 値 - NegativeNoiseTreshold) 以下の場合が、ある一定のサンプル数続くとき、Baseline 値は新しい Raw 値に設定されます。これは基本的に、Baseline 値のリセットが必要になる、異常に低いサンプル数を数えます。通常、起動時に finger-on-at-startup （ 立ち上げ時に指がある場合) を修正するために使用されます。

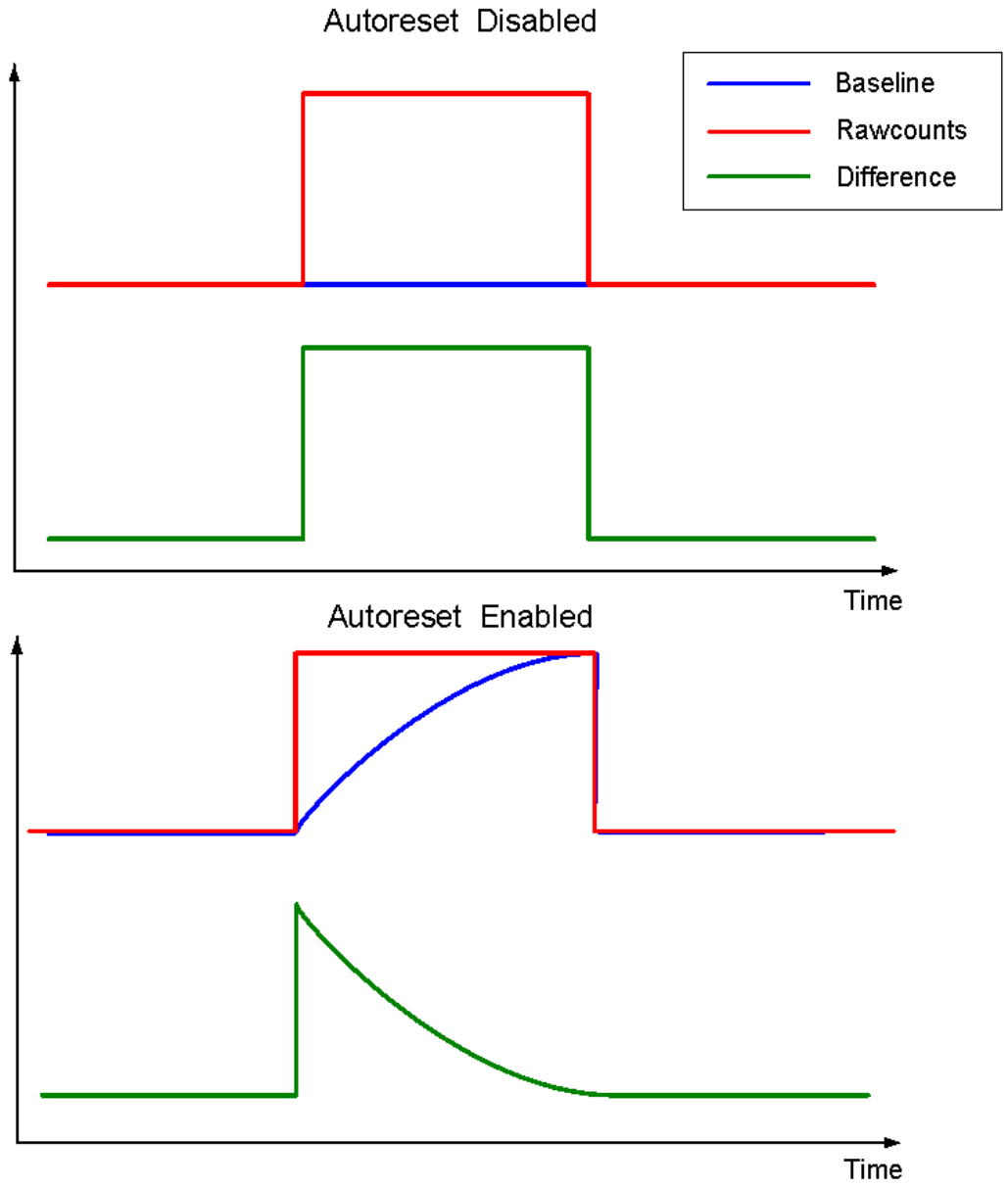
センサの Autoreset

このパラメータは、Baseline 値が常時更新されるか、信号差がノイズ閾値より低い場合のみ更新されるかを指定します。[**Enabled**] (有効) にセットされている場合、Baseline 値は常時更新されます。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生カウントが突然上がった際に、センサが永続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。

パラメータが [**Disabled**] (無効) にセットされている場合、Raw 値と Baseline 値の差がノイズ閾値パラメータを下回る場合にのみ、基準値は更新されます。長期間、センサの状態を維持する必要がない限り、このパラメータは [Disabled] にしておきます。

以下の図は、このパラメータがＢａｓｅｌｉｎｅ値更新に与える影響について示しています。

Figure 14.　センサ Autoreset パラメータ



Hysteresis (ヒステリシス)

ヒステリシスパラメータは、センサが現在動作中であるか否かに応じて、指閾値に数値を足したり、そこから引いたりします。センサがアクティブでない場合、Difference 値は指閾値 + 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、Difference 値は指閾値 - ヒステリシスを下回る必要があります。これは、デバウンス性と粘着性を指検知アルゴリズムに加えるために使用されます。ヒステリシスを伴う閾値は、bIsSensorActive() または bIsAnySensorActive() が呼び出されたときに評価されます。センサの状態は、bIsSensorActive() の戻り値または baSnsOnMask[] アレイを使用してモニタすることができます。可能な値は 0 ー 255 ですが、指閾値パラメータ設定よりも低くなければなりません。

適切なハイレベル判定論理パラメータを選択すると、環境要因 (温度や湿度の変化など) を効果的に補償し、ノイズ信号 (ESD、電源スパイク) を抑圧し、様々な条件下で信頼できるタッチ検知を実施できます。

[+] Feedback

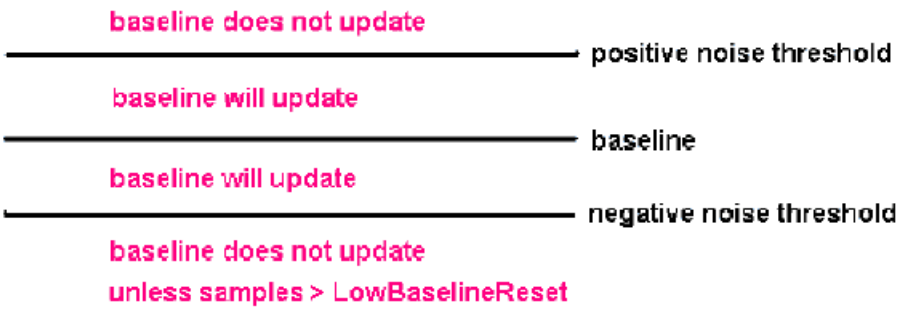
Debounce (デバンス)

デバンス パラメータは、センサの動作遷移のためのデバンスカウンタを設定します。センサがインアクティブからアクティブへ遷移するためには、Difference 値が指閾値 + ヒステリシスを上回る状態を、指定されたサンプル数以上、続けなければなりません。デバンス数は、API 関数の bIsSensorActive または bIsAnySensorActive で増分されます。

可能な値は 1 ー 255 です。1 をセットするとデバンスは起こりません。

NegativeNoiseThreshold

NegativeNoiseThreshold パラメータは、マイナスの Difference カウント閾値を追加します。現在のＲａｗカウントが Baseline 値を下回り、差分がこの閾値を上回る場合、Baseline 値は更新されません。しかし、LowBaselineReset パラメータで設定されたサンプル数の間、現在の Raw 値が低い状態で続く場合 (差分は閾値より大きい)、Baseline 値はリセットされます。



LowBaselineReset

LowBaselineReset パラメータは、NegativeNoiseThreshold パラメータと共に作動します。ある一定のサンプル数で、サンプルカウント値が (Ｂａｓｅｌｉｎｅ値 - NegativeNoiseTreshold) 以下の場合、baseline 値は新しいＲａｗ値に設定されます。これは基本的に、Baseline 値リセットが必要となる異常に低いサンプル数を数えるもので、通常、finger-on-at-startup （指置き電源投入）のために使用されます。

Scanning Speed (スキャン速度)

このパラメータは、センサのスキャン速度に影響を及ぼします。利用可能な選択肢は、**Fast** (高速)、**Normal** (標準)、**Slow** (低速) です。スキャン速度が遅いと、次のような利点があります。

- 優れた SNR
- 電源変化と温度変化に対する耐性
- システム割り込みレイテンシの低減、より長い割り込みを処理可能

割り込みレイテンシについては、「警告」セクションを参照してください。

Resolution (分解能)

このパラメータはスキャン分解能をビット数で判断します。PRS16 構成では、センサは 9 ~ 16 ビットの分解能でスキャンできます。PRS8 およびプリスケーラを持つ PRS8 構成では、センサは 8、10、12 ビットでのみスキャンできます。ビット数が N の場合、スキャン分解能の最大 Raw 値は 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 SNR が高くなります。近接検知用には高分解能を使用します。16-bit 分解能、低速スキャンモード、20 cm のワイヤによって、20 cm 以上離れた場所から手を検知できます。

VCI 分周器の値は、スキャン速度によって異なります。以下の表に、スキャン速度と VC1 分周器の関係を示します。

Table 4. VC_1 分周器の値とスキャン速度

Scanning Speed (スキャン速度)	VC1
高速	2
標準	4
低速	8

Table 5. 24 MHz IMO 動作におけるスキャン時間 (単位：µs)、スキャン 速度と分解能、PRS16 構成

分解能 (単位：ビット)	Scanning Speed (スキャン速度)		
	高速	標準	低速
9	260	510	1030
10	425	850	1710
11	770	1550	3080
12	1470	2940	5840
13	2840	5680	11400
14	5560	11200	22200
15	11100	22000	44000
16	21800	44400	88000

Table 6. 24 MHz IMO 動作、PRS8 構成またはプリスケーラを持つ PRS8 構成における、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)	Scanning Speed (スキャン速度)		
	高速	標準	低速
8	85	130	260
10	130	260	510
12	260	510	1020

注 スキャン時間は、2 つ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、変調器安定化遅延、サンプル変換間隔、データ前処理時間が含まれています。

Modulator Capacitor Pin (変調コンデンサ ピン)

このパラメータは、外付け変調コンデンサ (C_{mod}) を接続するピンを設定します。利用できるピンから選択します。

Feedback Resistor Pin (フィードバック レジスタ ピン)

このパラメータは、外付けフィードバック レジスタ (R_b) を接続するピンを設定します。利用できるピンから選択します。フィードバック レジスタ接続では、ISSP プログラミングに問題が発生することがあるため、P1[1] の使用は推奨できません。ヒント：これら一部のピンが、センサ接続の割り当てなど、その他の目的で使用される場合は、UM パラメータ リストには表示されません。

Ref Value (基準値)

このパラメータは、コンパレータレファレンス値を設定します。リファレンス値は、内部の抵抗分圧器から取得されます。ゼロは、最小基準 (1/4 Vdd) に該当します。8 は、最大値 (3/4 Vdd) に該当します。パラメータが大きくなるにつれ、基準電圧は線形的に高くなります。リファレンス値が高くなると、検出感度が下がりますが、シールド電極の影響が大きくなります。

センサに静電容量の顕著な差がある設計 (異なる面積のセンサなど) では、API 関数を使い、より大きい静電キャパシタで、高いリファレンス値を設定して、Raw 値のバランスを取ることができます。

Prescaler Period (プリスケーラ期間)

このパラメータは、プリスケーラ期間レジスタを設定し、プリチャージ スイッチ出力周波数を設定します。このパラメータは、プリスケーラを持つ PRS8 構成でのみ利用できます。プリスケーラ期間値は、1 ～ 255 になります。

最大信号対雑音比 (SNR) を得るための推奨値は 2^n-1 です。

- 1
- 3
- 7
- 15
- 31
- 63
- 127
- 255

その他の値を使うと、特に低い分解能と高いスキャン速度の場合に、ノイズが増えます。

PRS Polynomial (PRS 多項)

このパラメータは、PRS8 構成とプリスケーラを持つ PRS8 構成で、PRS 多項を設定します。次の 2 つのオプションがあります。

- Short - short polynomial を使うと、SNR が改善されますが、繰り返し期間が短くなるため、デバイスが外部ノイズ源の影響を受けやすくなります。
- Long - long polynomial を使うと、SNR が悪くなりますが、デバイスがノイズ信号に影響されにくくなります。

ShieldElectrodeOut

シールド電極信号源は、空いているデジタル Row バス (Row_0_Output_1 ～ Row_0_Output_3).) のいずれかから選択できます。各 Row 出力は、3 つのピンのいずれかに出力できます。Row LUT 機能を A に設定してください。

[+] Feedback

アプリケーション プログラミング インタフェース (API)

アプリケーション プログラミング インタフェース (API) 関数は、ハイレベルでモジュールを扱うことができるように、ユーザ モジュールの一部として提供されています。このセクションでは、各関数に対するインタフェースを include ファイルによって提供される関連定数とともに示します。

注 ** ここでは、すべてのユーザ モジュール API と同様に、API 関数を呼び出すことで A と X レジスタの値は、変更される可能性があります。呼び出し前の A と X の値が必要な場合は、呼び出し元関数で A と X の値を保存してください。PSoC Designer のバージョン 1.0 以降では、効率を高めるために、この「registers are volatile (レジスタの揮発性)」ポリシーが選択され、実施されています。C コンパイラは、自動的にこの条件で考慮しています。アセンブリ言語のプログラマは、コードでこのポリシーを考慮する必要があります。一部のユーザ モジュール API 関数では A と X は変更されないこともありますが、将来も変更されないという保証はありません。

CSD を初期化し、サンプリングを開始し、CSD を終了するものは用意されています。すべてのケースで、モジュールのインスタンス名は次のエントリポイントの CSD 接頭辞と置き換えられます。誤ったインスタンス名を使用して構文エラーが発生することが頻繁にあります。

API 関数は様々なグローバルアレイに使用されます。そのため、これらのアレイを手動で変更してはなりません。ただし、デバックの目的でこれらの値を調べることが可能です。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は可能です。以下にいくつかのグローバルアレイを挙げます。

- CSD_waSnsBaseline[]
- CSD_waSnsResult[]
- CSD_waSnsDiff[]
- CSD_baSnsOnMask[]

CSD_waSnsBaseline[] – 各センサの Baseline 値データの INT 型アレイです。アレイのサイズはセンサ数と同等です。CSD_waSnsBaseline[] アレイは以下の関数によって更新されます。

- CSD_UpdateAllBaselines();
- CSD_UpdateSensorBaseline();
- CSD_InitializeBaselines().

CSD_waSnsResult[] – 各センサの Raw データの INT 型アレイです。アレイのサイズはセンサ数と同等です。CSD_waSnsResult[] データは次の関数によって更新されます。

- CSD_ScanSensor();
- CSD_ScanAllSensors().

CSD_waSnsDiff [] – 各センサの Raw データと Baseline 値データの差の INT 型アレイです。アレイのサイズはセンサ数と同等です。

CSD_baSnsOnMask[] – センサのオン・オフの状態 (ボタンまたはスライダ) を維持するバイトアレイです。CSD_baSnsOnMask[0] は、センサ 0 ~ 7 のマスクされたビットを含んでいます (センサ 0 はビット 0、センサ 1 はビット 1)。CSD_baSnsOnMask[1] はセンサ 8 ~ 15 のマスクされたビットを含んでいます。必要に応じて、同様の方法で、より多くのセンサにも対応できます。このバイトアレイには、全ての配置されているセンサの要素が含まれます。ボタンがオンの場合 1 つのビットの値は 1 で、オフの場合その値は 0 です。CSD_baSnsOnMask[] データは、CSD_bIsSensorActive(BYTE bSensor) 関数または CSD_bIsAnySensorActive() ルーチンによって更新されます。

CSD_Start

説明：

レジスタを初期化し、ユーザ モジュールを開始します。他のユーザ モジュール関数を呼び出す前に、この関数を呼び出さなければなりません。

C プロトタイプ：

```
void  CSD_Start()
```

アセンブリ：

```
lcall  CSD_Start
```

パラメータ：

なし

戻り値：

なし

特殊作用：

**

CSD_Stop

説明：

センサスキャンを停止し、内部割り込みを無効にし、CSD_ClearSensors() を呼び出してすべてのセンサをリセットして INACTIVE 状態にします。

C プロトタイプ：

```
void  CSD_Stop()
```

アセンブリ：

```
lcall  CSD_Stop
```

パラメータ：

なし

戻り値：

なし

特殊作用：

**

CSD_ScanSensor

説明：

選択されたセンサをスキャンします。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 CSD ウィザードによって順番に割り当てられます。Sw0 は センサ 0、Sw1 はセンサ 1 などのように割り当てられます。

C プロトタイプ：

```
void CSD_ScanSensor(BYTE bSensor)
```

アセンブリ：

```
mov A,  bSensor
lcall CSD_ScanSensor
```

パラメータ：

A => センサ番号

bSensor - センサ番号

戻り値：

なし

副作用

**

CSD_ScanAllSensors

説明：

各センサインデックスに対して CSD_ScanSensor() を呼び出して、構成済み全センサをスキャンします。

C プロトタイプ：

```
void CSD_ScanAllSensors()
```

アセンブリ：

```
lcall CSD_ScanAllSensors
```

パラメータ：

なし

戻り値：

なし

副作用

**

CSD_UpdateSensorBaseline

説明：

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Ｂａｓｅｌｉｎｅ値と呼ばれます。Ｂａｓｅｌｉｎｅ値はバケツメソッドを用いて更新されます。

バケツメソッドは、次のアルゴリズムを使用します。

1. CSD_UpdateSensorBaseline() が呼び出されるたびに、Raw カウント値を前回のＢａｓｅｌｉｎｅ値から引いて Difference 値を計算します。この差は CSD_waSnsDiff[] アレイに保存され、表示されます。

2. センサ Autoreset が無効な場合、CSD_UpdateSensorBaseline() が呼び出されるたびに、Difference 値をノイズ閾値と比較します。差がノイズ閾値より小さい場合、仮想バケツに入れられます。差がノイズ閾値より大きい場合、バケツは更新されません。センサ Autoreset が有効な場合、ノイズ閾値パラメータに関わらず、差分は仮想バケツに集積されます。

3. 仮想バケツで集積された Difference カウントが BaselineUpdateThreshold に達すると、Ｂａｓｅｌｉｎｅ値は 1 増分され、バケツは 0 にリセットされます。

4. Difference カウントがノイズ閾値より小さい場合、waSnsDiff[] アレイに保持されている値が 0 にリセットされます。従って、このアレイには 0 より大きく NoiseThreshold より小さい値を持つ成分は含まれません。

C プロトタイプ：

```
void CSD_UpdateSensorBaseline(BYTE bSensor)
```

アセンブリ：

```
mov   A,   bSensor
lcall  CSD_UpdateSensorBaseline
```

パラメータ：

A => センサ番号

bSensor - センサ番号

戻り値：

なし

特殊作用：

**

CSD_UpdateAllBaselines

説明：

CSD_bUpdateSensorBaseline() 関数を使用して、すべてのセンサのＢａｓｅｌｉｎｅ値を更新します。

C プロトタイプ：

```
void  CSD_UpdateAllBaselines()
```

アセンブリ：

```
call CSD_UpdateAllBaselines
```

パラメータ：

なし

戻り値：

なし

副作用：

**

CSD_bIsSensorActive

説明：

センサの Difference カウントアレイを確認し、指閾値と比較します。ここではヒステリシスを考慮します。ヒステリシス値は、センサが現在オンであるか否かに基づいて、指閾値を足したり引いたりします。アクティブ時の場合、閾値は下げられます。アクティブ時でない場合、閾値は上げられます。この関数は、CSD_baSnsOnMask[] アレイでセンサのビットを更新します。

C プロトタイプ：

```
BYTE  CSD_bIsSensorActive(BYTE bSensor)
```

アセンブリ：

```
mov  A,  bSensor
call CSD_bIsSensorActive
```

パラメータ：

bSensor - センサ番号

bSensor A => センサ番号

戻り値：

戻り値は、アクティブ時の場合 1 で、アクティブでない場合は 0 です。

A => 1 – 選択されたセンサがアクティブ。0 – 選択されたセンサがアクティブではない。

副作用：

**

CSD_bIsAnySensorActive

説明：

全センサのＤｉｆｆｅｒｅｎｃｅカウントアレイを確認し、指閾値と比較します。各センサについて CSD_bIsSensorActive() を呼び出し、CSD_baSnsOnMask[] 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。

C プロトタイプ：

```
BYTE  CSD_bIsAnySensorActive()
```

アセンブリ：

```
lcall CSD_bIsAnySensorActive
```

パラメータ：

なし

戻り値：

戻り値は、アクティブの場合 1 で、アクティブでない場合は 0 です。

A => 1 – 1 つ以上のセンサがアクティブ。0 – アクティブのセンサなし。

副作用：

**

CSD_wGetCentroidPos

説明：

重心計算のため、Difference アレイを確認。1 が存在する場合、オフセットと長さが一時変数に保存され、重心位置が CSD ウィザードで指定された分解能で計算されます。この関数は、スライダが CSD ウィザードで指定されている場合のみ利用できます。

C プロトタイプ：

```
WORD  CSD_wGetCentroidPos(BYTE bSnsGroup)
```

アセンブリ：

```
mov  A,  bSnsGroup
lcall CSD_wGetCentroidPos
```

パラメータ：

bSnsGroup - センサ グループ

個別のセンサ グループではセンサ グループ = 0

最初のスライダ グループではセンサ グループ = 1

2 つ目のスライダ グループではセンサ グループ = 2

bSnsGroup A => グループ番号

このパラメータは、スライダとして使用されるセンサグループへのレファレンスです。グループ 0 はボタン用です。スライダはグループ 1 以降に含まれています。

戻り値：

スライダの位置値は LSB は A に、MSB は X にあります。

副作用：

このルーチンは、ノイズ閾値を引くことによって Difference カウントを調整します。マイナスの Difference 値となることを避けるために、各スキャン後一度だけ呼び出します。Ｄｉｆｆｅｒｅｎｃｅカウント信号をモニターするアプリケーションの場合、Difference カウントデータ転送後にこのルーチンを呼び出します。

スライダセンサが 1 つでも動作中の場合、関数はゼロから CSD ウィザードで設定された分解能値までを返します。アクティブのセンサがない場合、関数は –1 (FFFFh) を返します。重心 / 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 –1 (FFFFh) を返します。必要に応じて、CSD_blsSensorActive() ルーチンを使用してタッチされたスライダセグメントを判定できます。

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重心結果を示すことがあります。ノイズで正しくない重心結果が生じないように、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)。

CSD_wGetRadialPos

説明：

Difference アレイを確認し、重心を探します。1 つ存在する場合、重心位置を CSD ウィザードに指定された分解能に計算します。この関数は、CSD ウィザードで指定されているラジアル スライダの場合のみ利用できます。

C プロトタイプ：

```
WORD  CSD_wGetRadialPos(BYTE bSnsGroup)
```

アセンブリ：

```
mov  A,  bSnsGroup
call CSD_wGetRadialPos
```

パラメータ：

bSnsGroup A => グループ番号

このパラメータは、使用中のラジアルスライダの数です。この番号は、ラジアル スライダ値の左にある CSD UM ウィザードを通して得ることができます (たとえば s2 の場合、ラジアル スライダ番号は 2)。

戻り値：

ラジアル スライダの位置の値は LSB は A、MSB は X です。

副作用：

マイナスの Difference 値と Baseline 値の更新を避けるために、各スキャン後一度だけ呼び出します。Difference カウント信号をモニターするアプリケーションの場合、Ｄｉｆｆｅｒｅｎｃｅカウントデータ転送後にこのルーチンを呼び出します。

スライダセンサが 1 つでもアクティブの場合、関数はゼロから CSD ウィザードで設定された分解能値までを返します。アクティブのセンサがない場合、関数は –1 (FFFFh) を返します。

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重心結果を示すことがあります。ノイズが正しくない重心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)。

CSD_wGetRadialInc

説明：

実際の指シフトを返します。これは現在と前回の指位置の差です。この関数は CSD_wGetRadialPos() とともに機能し、後者が生成したデータを利用します (データは内部変数に保存されます)。

C プロトタイプ：

```
WORD  CSD_wGetRadialInc(BYTE bSnsGroup)
```

アセンブリ：

```
mov  A,  bSnsGroup
call CSD_wGetRadialInc
```

パラメータ：

bSnsGroup A => グループ番号

このパラメータは、使用中のラジアル スライダの数です。この番号は、ラジアル スライダ値の左にある CSD UM ウィザードを通して得ることができます (たとえば s2 の場合、ラジアル スライダ番号は 2)。

戻り値：

指シフト値。時計回りはプラス、反時計周りはマイナス。LSB は A、MSB は X。

指シフト値は現在と前回の指位置の差です。前回スキャンの間にタッチがなかった場合 (前回 CSD_wGetRadialPos() が 1 (FFFFh) を返した)、または今回タッチがない場合 (今回 CSD_wGetRadialPos() が -1 (FFFFh) を返した)。

副作用：

このルーチンは CSD_wGetRadialPos() API の後に呼び出されなければなりません。これは、CSD_wGetRadialPos(). によってセットされる内部データ CSD_waSliderPrevPos と CSD_waSliderCurrPos を使用するためです。

CSD_InitializeSensorBaseline

説明：

選択されたセンサをスキャンして、初期値を伴う CSD_waSnsBaseline[bSensor] アレイを読み込みます。Raw カウント値は、選択されたセンサのＢａｓｅｌｉｎｅ値の配列エレメントにコピーされます。この関数は、個々のセンサの Baseline 値をリセットするために使用されます。

C プロトタイプ：

```
void CSD_InitializeSensorBaseline(BYTE bSensor)
```

アセンブリ：

```
mov A, bSensor
lcall CSD_InitializeSensorBaseline
```

パラメータ：

bSensor - センサ番号

A => センサ番号

戻り値：

なし

副作用：

**

CSD_InitializeBaselines

説明：

各センサをスキャンして、初期値を CSD_waSnsBaseline[] アレイを読み込みます。Raw カウント値は各センサの Baseline 値アレイにコピーされます。

C プロトタイプ：

```
void CSD_InitializeBaselines()
```

アセンブリ：

```
lcall CSD_InitializeBaselines
```

パラメータ：

なし

戻り値：

なし

副作用：

**

CSD_SetDefaultFingerThresholds

説明：

FingerThreshold パラメータ値を CSD_baBtnFThreshold[] アレイを読み込みます。CSD_baBtnFThreshold[] アレイがカスタム値を手動で読み込まれていない場合、この関数はスキャン前に呼び出さなければなりません。

C プロトタイプ：

```
void CSD_SetDefaultFingerThresholds()
```

アセンブリ：

```
lcall CSD_SetDefaultFingerThresholds
```

パラメータ：

なし

戻り値：

なし

副作用：

**

CSD_SetScanMode

説明：

スキャン速度と分解能を設定します。この関数を実行時に呼び出し、スキャン速度と分解能を変更することができます。関数は、ユーザ モジュールのパラメータ設定を上書きします。異なるスキャン速度と分解能でスキャンする必要のあるセンサがある場合、この関数は効果的です。例としては、通常のボタンと近接検出などが挙げられます。通常のボタンは 9-bit 分解能、300 µs スキャン速度でスキャンできます。近接検知では、長い範囲の検知を実行するために、16-bit の分解能と 12 ms 以上のスキャン時間が使用されることもあります。この関数は CSD_ScanSensor() 関数とともに使用できます。

C プロトタイプ：

```
void CSD_SetScanMode(BYTE bSpeed, BYTE bResolution)
```

アセンブリ：

```
mov     A, bSpeed
mov     X, bResolution
lcall    CSD_SetScanMode
```

パラメータ：

bSpeed - 0 ～ 3 のスキャン速度コード

bResolution - 分解能 [9..14]

戻り値：

なし

副作用：

**

CSD_SetRefValue

説明：

スキャンのリファレンス値を設定します。リファレンスがアナログ変調器 (基準パラメータの ASE11) または外部からのフィルタされた PWM/PRSPWM 信号から供給される場合にのみ有効です。許可されている値は 0..8 です。値 0 は、最大感度をもたらす最小リファレンス電圧になります。値 8 は、最大リファレンス電圧を設定し、感度が低くなります。この関数は CSD_ScanSensor(). とともに使用できます。

C プロトタイプ：

```
void CSD_SetRefValue(BYTE bRefValue)
```

アセンブリ：

```
mov     A, bRefValue
lcall    CSD_SetRefValue
```

パラメータ：

bRefValue - スキャンのリファレンス値を設定します。許可されている値は 0..8 です。

戻り値：

なし

副作用：

**

CSD_ClearSensors

説明：

各センサに対して CSD_wGetPortPin() と CSD_DisableSensor() を連続的に呼び出すことにより、すべてのセンサを非サンプリング状態にクリアします。

C プロトタイプ：

```
void  CSD_ClearSensors()
```

アセンブリ：

```
lcall  CSD_ClearSensors
```

パラメータ：

なし

戻り値：

なし

副作用：

**

CSD_wReadSensor

説明：

A (LSB) と X (MSB) で主要な Raw スキャン値を返します。

C プロトタイプ：

```
WORD  CSD_wReadSensor(BYTE bSensor)
```

アセンブリ：

```
mov A, bSensor
lcall CSD_wReadSensor
```

パラメータ：

bSensor - センサ番号

A => センサ番号

戻り値：

センサのスキャンです。A では LSB、X では MSB。

副作用：

**

CSD_wGetPortPin

説明：

特定のセンサのポート番号とピンマスクを返します。渡されたパラメータは、CSD_Sensor_Table[]. からのデータをインデックスし、選択します。戻り値は、CSD_EnableSensor()、CSD_DisableSensor(). へと渡すことができます。

C プロトタイプ：

```
WORD  CSD_wGetPortPin(BYTE bSensor)
```

アセンブリ：

```
mov  A,  bSensor
lcall     CSD_wGetPortPin
```

パラメータ：

bSensorNumber – 範囲は 0 ~ (n – 1) です。ここで n は、CSD ウィザードにセットされたセンサ数とスライダに含まれているセンサ数の合計です。センサ番号は、選択されたアクティブなセンサのポートとビットマスクを判定するために、CSD_wGetPortPin() によって使用されます。

戻り値：

A => センサのビットマスク

X => ポート番号

副作用：

**

CSD_EnableSensor

説明：

次の測定サイクルで測定するために選択されたセンサを構成します。ポートとセンサは、CSD_wGetPortPin() 関数を使用して選択できます。このとき、ポート番号とセンサビットマスクはそれぞれ X と A に読み込まれています。選択されたポートとピンを Analog High Z (アナログ高 Z) モードにし、正しい Analog Mux Bus (アナログ多重化バス) 入力を可能にするために、駆動モードを調整します。これによりコンパレータ機能も有効になります。

C プロトタイプ：

```
void  CSD_EnableSensor(BYTE bMask, BYTE bPort)
```

アセンブリ：

```
mov  X,  bPort
mov  A,  bMask
lcall  CSD_EnableSensor
```

パラメータ：

A => センサのビットマスク

X => ポート番号

bSensorMask - 任意のセンサのビットマスク

bPort - 任意のキーのポート番号

戻り値：

なし

[+] Feedback

副作用：

**

CSD_DisableSensor

説明：

CSD_wGetPortPin() 関数により選択されたセンサを無効にします。駆動モードは、Strong (001) に変更され、ゼロにセットされます。これにより、センサが効果的にグランド接続されます。ポートピンから AnalogMuxBus までの接続はオフになります。この関数パラメータは、CSD_wGetPortPin() 関数により返されます。

C プロトタイプ：

```
void  CSD_DisableSensor(BYTE bMask, BYTE bPort)
```

アセンブリ：

```
mov  X,  bPort
mov  A,  bMask
lcall  CSD_DisableSensor
```

パラメータ：

A => センサのビットマスク

X => ポート番号

bSensorMask - 任意のセンサのビットマスク

bPort - 任意のキーのポート番号

戻り値：

なし

副作用：

**

ファームウェア ソースコードの例

例 1 このコードは、ユーザ モジュールを開始し、センサを連続的にスキャンします。通信セクションは、PC チャート作成ツールに値を転送するために使用できます。

```
//-------------------------------------------------------------------------
// Sample C code for the CSD module
// Scanning all sensors continuously
//-------------------------------------------------------------------------

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

void main(void)
{
  M8C_EnableGInt;
  CSD_Start();
  CSD_InitializeBaselines() ; //scan all sensors first time, init baseline
  CSD_SetDefaultFingerThresholds() ;
  //
  // Loop Forever
  //
  while (1) {
    CSD_ScanAllSensors(); //scan all sensors in array (buttons and sliders)
    CSD_UpdateAllBaselines(); //Update all baseline levels;

//detect if any sensor is pressed
if(CSD_bIsAnySensorActive()){
// Add user code here to proceed the sensor touching
}

    // now we are ready to send all status variables to chart program
    // communication here
    //
    // OUTPUT CSD_waSnsResult[x] <- Raw Counts
    // OUTPUT CSD_waSnsDiff[x] <- Difference
    // OUTPUT CSD_waSnsBaseline[x] <- Baseline
    // OUTPUT CSD_baSnsOnMask[x] <- Sensor On/Off
  }
}
```

例 2 次のコードは、複数のセンサを並列に接続し、CSD_ScanSensor() 関数を呼び出すことによって、それらを同時にスキャンしています。このサンプルは、タッチがあったセンサを区別することなくセンサのタッチを検知する必要がある場合に有用です。これは、デバイスウェークアップ検知と、バッテリの工ネルギーを節約するためのスキャン時間の短縮に利用できます。ウェイクアップタッチを 1 件検知した後は、各センサを従来型の個別スキャンに戻すことができます。

```
//-------------------------------------------------------------------------
// Sample C code for the CSD module
// Scan several sensors in parallel
//-------------------------------------------------------------------------

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

void main(void)
{
     M8C_EnableGInt;

     CSD_Start();
     CSD_SetDefaultFingerThresholds();

     // Enable the sensor connected to P1[4]
     CSD_EnableSensor(0x10, 1);
     // Enable the sensor connected to P1[6]
     CSD_EnableSensor(0x40, 1);
     // Enable the sensor connected to P3[0]
     CSD_EnableSensor(0x01, 3);

     // Initialize baseline for sensor number "3"
     CSD_InitializeSensorBaseline(3);

     while (1) {
          // Scan continuously sensor number "3" which is connected
          //in parallel to the enabled above sensors
          CSD_ScanSensor(3);
          CSD_UpdateSensorBaseline(3);
                if(CSD_bIsSensorActive(3)){
     // Add user code here to proceed the buttons pressing
                }
     }
}
```

例 3 次の例は、CSD_SetScanMode() 関数を使用して異なるスキャンパラメータを用いる異なるセンサをスキャンしています。ボタンタッチ検知と近接検知を実行する必要がある場合に有用です。ボタンはスキャン時間を短縮するために低分解能でスキャンし、近接検知は最大感度を実現するために高分解能でスキャンします。近接検知スキャンの頻度を下げ、ボタンタッチが検知されない場合にのみ、このコードを採用することもできます。

```
//-------------------------------------------------------------------------
// Sample C code for the CSD module
// Scanning sensors with different scanning speed and resolution
//-------------------------------------------------------------------------

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

void main(void)
{

     M8C_EnableGInt;

     CSD_Start();
     CSD_SetDefaultFingerThresholds();

     // Set Fast, 9-bit resolution mode for baseline calculations
     CSD_SetScanMode(1, 9);

     // Initialize baselines for all of the sensors which operate in
     // Fast mode and 9-bit resolution
     CSD_InitializeSensorBaseline(0);
     CSD_InitializeSensorBaseline(1);
     CSD_InitializeSensorBaseline(2);

     // Set Slow, 14-bit resolution mode for baseline calculations
     CSD_SetScanMode(3, 14);
     // Initialize baselines for all of the sensors which operate in
     // Slow mode and 14-bit resolution
     CSD_InitializeSensorBaseline(3);

     while (1) {
          // Set Fast, 9-bit resolution mode for the following buttons
          CSD_SetScanMode(1, 9);
          // Scan sensor number "0"
          CSD_ScanSensor(0);
          // Scan sensor number "1"
          CSD_ScanSensor(1);
          // Scan sensor number "2"
          CSD_ScanSensor(2);

          // Set Slow, 14-bit resolution mode for the following sensor
          CSD_SetScanMode(3, 14);
          // Scan sensor number "3"
          CSD_ScanSensor(3);

          CSD_UpdateAllBaselines();
          //detect if any sensor is pressed
          if(CSD_bIsAnySensorActive()){
```

```
                // Add user code here to proceed the buttons pressing
            }
        }
}
```

例 4 次の例は、各センサについて異なる指閾値レベルを設定する能力を示しています。異なるセンサが異なる場所に配置されており、一部のセンサが他のセンサより検出感度が高い場合に有効です。

```
//-------------------------------------------------------------------------
// Sample C code for the CSD module
// Set individual finger threshold parameter for each sensor
//-------------------------------------------------------------------------

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

void main(void)
{
M8C_EnableGInt;

CSD_Start();
CSD_InitializeBaselines();

// set finger threshold for sensor "0"
CSD_baBtnFThreshold[0] = 10;
// set finger threshold for sensor "1"
CSD_baBtnFThreshold[1] = 20;
// set finger threshold for sensor "2"
CSD_baBtnFThreshold[2] = 30;
// set finger threshold for sensor "3"
CSD_baBtnFThreshold[3] = 40;
// set finger threshold for sensor "4"
CSD_baBtnFThreshold[4] = 50;
// set finger threshold for sensor "5"
CSD_baBtnFThreshold[5] = 255;
// set finger threshold for sensor "6"
CSD_baBtnFThreshold[6] = 200;

while (1) {
// Scan continuously all sensors
CSD_ScanAllSensors();
CSD_UpdateAllBaselines();
//detect if any sensor is pressed
if(CSD_bIsAnySensorActive()){
// Add user code here to proceed the buttons pressing
}
}
}
```

設定レジスタ

Table 7. ブロック CMP、レジスタ：制御 0

ビット	7	6	5	4	3	2	1	0
値	0	0	1	1	1	0	1	0

Table 8. ブロック CMP、レジスタ：制御 1

ビット	7	6	5	4	3	2	1	0
値	0	0	1	0	0	0	0	1

レジスタ PRS8 構成

Table 9. ブロック CMP、レジスタ：制御 2

ビット	7	6	5	4	3	2	1	0
値	0	1	0	0	0	0	0	0

Table 10. ブロック CMP、レジスタ：制御 3

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 11. ブロック PRS、レジスタ：制御

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 12. ブロック PRS、レジスタ：多項

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 13. ブロック PRS、レジスタ：シード

モード / ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 14. ブロック PRS、レジスタ：関数

ビット	7	6	5	4	3	2	1	0
値	0	1	1	0	1	0	1	0

Table 15. ブロック PRS、レジスタ：Input (入力)

モード / ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 16. ブロック PRS、レジスタ：Output (出力)

モード / ビット	7	6	5	4	3	2	1	0
値	1	1	0	0	0	0	0	0

Table 17. ブロック CMP、レジスタ：制御 0

ビット	7	6	5	4	3	2	1	0
値	0	0	1	1	1	0	1	0

Table 18. ブロック CMP、レジスタ：制御 1

ビット	7	6	5	4	3	2	1	0
値	0	0	1	0	0	0	0	1

レジスタ PRS16 構成

Table 19. ブロック CMP、レジスタ：制御 2

ビット	7	6	5	4	3	2	1	0
値	0	1	0	0	0	0	0	0

Table 20. ブロック CMP、レジスタ：制御 3

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 21. ブロック CNT、レジスタ：制御

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 22. ブロック CNT、レジスタ：Period (期間)

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 23. ブロック CNT、レジスタ：比較

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 24. ブロック PRS16_LSB、レジスタ：制御

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 25. ブロック PRS16_LSB、レジスタ：多項

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 26. ブロック PRS16_LSB、レジスタ：シード

モード / ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 27. ブロック PRS16_MSB、レジスタ：制御

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 28. ブロック PRS16_MSB、レジスタ：多項

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 29. ブロック PRS16_MSB、レジスタ：シード

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 30. ブロック CNT、レジスタ：関数

ビット	7	6	5	4	3	2	1	0
値	0	0	1	0	0	0	0	1

Table 31. ブロック CNT、レジスタ：Input (入力)

モード / ビット	7	6	5	4	3	2	1	0
値	0	0	0	1	0	1	1	0

Table 32. ブロック CNT、レジスタ：Output (出力)

モード / ビット	7	6	5	4	3	2	1	0
値	0	1	0	0	0	0	0	0

Table 33. ブロック PRS16_LSB、レジスタ：関数

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	1	0	1	0

Table 34. ブロック PRS16_LSB、レジスタ：Input (入力)

モード / ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 35. ブロック PRS16_LSB、レジスタ：Output (出力)

モード / ビット	7	6	5	4	3	2	1	0
値	1	1	0	0	0	0	0	0

Table 36. ブロック PRS16_MSB、レジスタ：関数

ビット	7	6	5	4	3	2	1	0
値	0	1	1	0	1	0	1	0

Table 37. ブロック PRS16_MSB、レジスタ：Input (入力)

モード / ビット	7	6	5	4	3	2	1	0
値	0	0	1	1	0	0	0	0

Table 38. ブロック PRS16_MSB、レジスタ：Output (出力)

モード / ビット	7	6	5	4	3	2	1	0
値	1	1	0	0	0	0	0	0

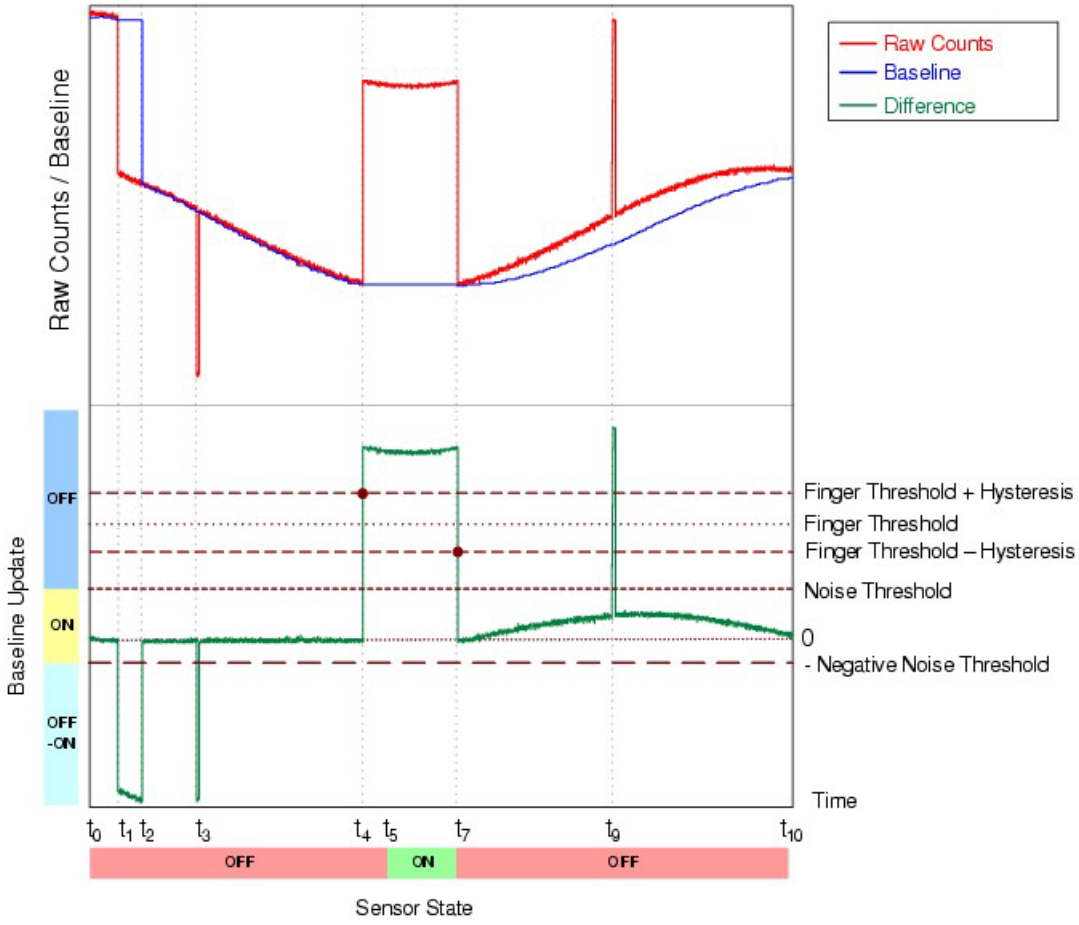
付録

ここからのセクションには、ユーザ モジュールデータシートに通常含まれている以外の情報が記載されています。これらの詳細情報は、ユーザによる CapSense アプリケーションの設計を支援するために、サイプレスのエンジニアが開発したものです。情報の一部は、将来のアプリケーションノートに組み込まれる可能性もあります。

CSD パラメータのインタラクション

以下の図は、baseline 値の更新と判定論理演算を示しており、最適なパフォーマンスを実現するための UM パラメータの設定方法を理解するうえで役立ちます。最初の図は、Sensors Autoreset パラメータが [**Disabled**] (無効) に設定されている場合のシステム動作を示しています。2 番目の図は、Sensors Autoreset パラメータが [**Enabled**] (有効) に設定されている場合を示しています。指閾値、ノイズ閾値、ヒステリシス、ネガティブノイズ閾値が、Difference 値の信号とともに示されています (Raw カウント – Baseline)。データは、システム動作を実演する人工的なテストで、低速と高速の R a w カウント変化の際に収集されました。低速の変化は温度や湿度の変化などによって、また高速の変化はセンサタッチ、ESD イベント、または強い RF フィールドの影響などによって発生します。

Figure 15. Sensors Autoreset が無効に設定されている Raw カウント、Baseline 値、Difference の信号変化の例



湿度や気温の変化により、Baseline 値レベルに近い Raw カウントは t_0 で次第に速度を落とし始めます。2 つの連続した変換の間に起きる R a w カウントの変化は NegativeNoiseThreshold パラメータを上回らないため (絶対値)、Raw カウントの最小値をトラッキングし、Raw カウント信号のうち低い値を維持することにより、Baseline 値は更新されます。

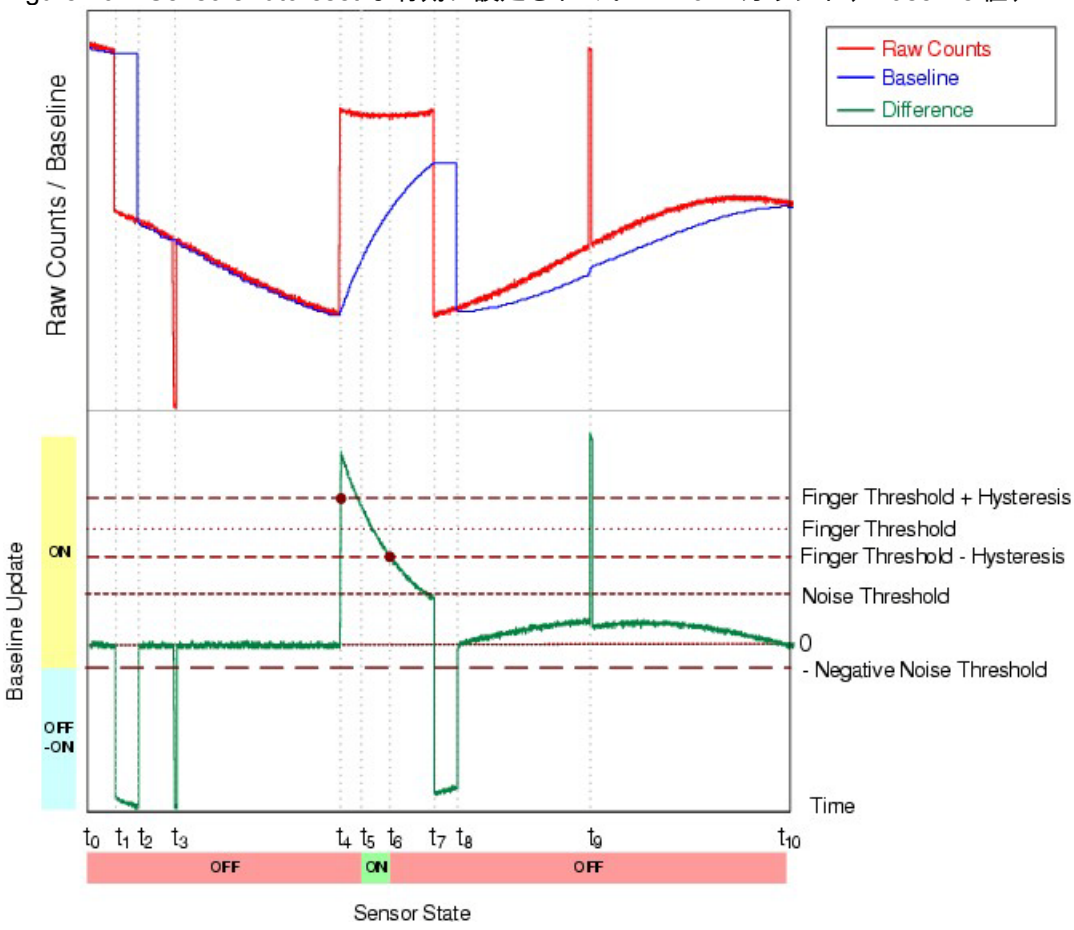
t_1 でＲａｗカウントは大幅に落下し、マイナスの差が NegativeNoiseThreshold を超えます。この状況は、指がセンサに接触しているときにデバイスの電源がオンになり、ある程度時間が経ってから指を離した場合に発生します。このとき、基準値更新のメカニズムが停止し、内部のタイムアウトカウンタが起動します。Ｂａｓｅｌｉｎｅ値は、LowBaselineReset のサンプル数の間、Difference 信号が NegativeNoiseThreshold を下回ったときにリセットされます。この状況は t_2 で起きます。

2 番目に大きなマイナスの Difference 信号スパイクは、t_3 で発生します。このスパイクが発生する原因は ESD イベントなどです。サンプルカウント中のスパイクが LowBaselineReset パラメータを下回るため、Baseline 値は待機状態となり、スパイクはフィルタにかけられます。これにより、正しくない Baseline リセットと、その結果である正しくないタッチ検知を防ぐことができます。

センサは t_4 でタッチされています。Difference 信号が FingerThreshold + ヒステリシス値を超えている場合、内部デバンスカウンタが起動します。信号がこの値をデバンスサンプル以上に超えると、センサの状態がオンにセットされます。これは t_5 で起きます。Difference 信号が t_7 で FingerThreshold + ヒステリシスレベルを下回ると、センサの状態はすぐにオフに戻ります。t_9 の短く正側のスパイクは、デバンスカウンタでフィルタされます。これは、サンプルユニットにおけるスパイク期間がデバンス値を超えないためです。

生カウントは、t_7 と t_{10} の間、ゆっくりと上昇します。Ｄｉｆｆｅｒｅｎｃｅ信号が NoiseThreshold を下回っている場合、バケツアルゴリズムを用いて Baseline 値を更新します (SensorsAutoreset は Disabled (無効) に設定)。Ｄｉｆｆｅｒｅｎｃｅ信号はドリフトレートに比例します。BaselineUpdate 閾値パラメータを用いて Baseline 値更新の速度を制御することができます。パラメータ値が低いと、Baseline 値更新速度が速くなります。

Figure 16.　SensorsAutoreset が有効に設定されているＲａｗカウント、Baseline 値、Difference 信号変化の例



上の図のシステム動作は、前述のケースの動作に似ていますが、以下のような違いがあります。

- センサがタッチされている間は、動作中のＢａｓｅｌｉｎｅ更新アルゴリズムによって、タッチ時間が短縮されます (t_6)。
- 指を離すと、タッチ検知を短時間阻止している LowBaselineReset サンプル (t_8) 後に、Ｂａｓｅｌｉｎｅ値がリセットされます。これは、もう 1 つのデバンス メカニズムとして機能します。

CSD のステップ バイ ステップ調整ガイド

静電容量の検知では、検知電極で最適なパラメータを設定することが重要です。この設定に影響する変数は、以下のとおりです。

- 電極の寸法
- オーバーレイの厚さと誘電率
- PSoC デバイスへの電極接続抵抗
- 以下のような、最終的な使用条件

- 電源の存在
- 温度
- 湿度
- 水分の存在
- ESD、EMC、EMI の要件

さまざまなタスク (耐水操作、高抵抗素材を使った検知、近接検知、厚いオーバーレイを通した操作、認定テストに合格するための推奨事項など) でのベスト プラクティスは、アプリケーション ノートで別途説明します。

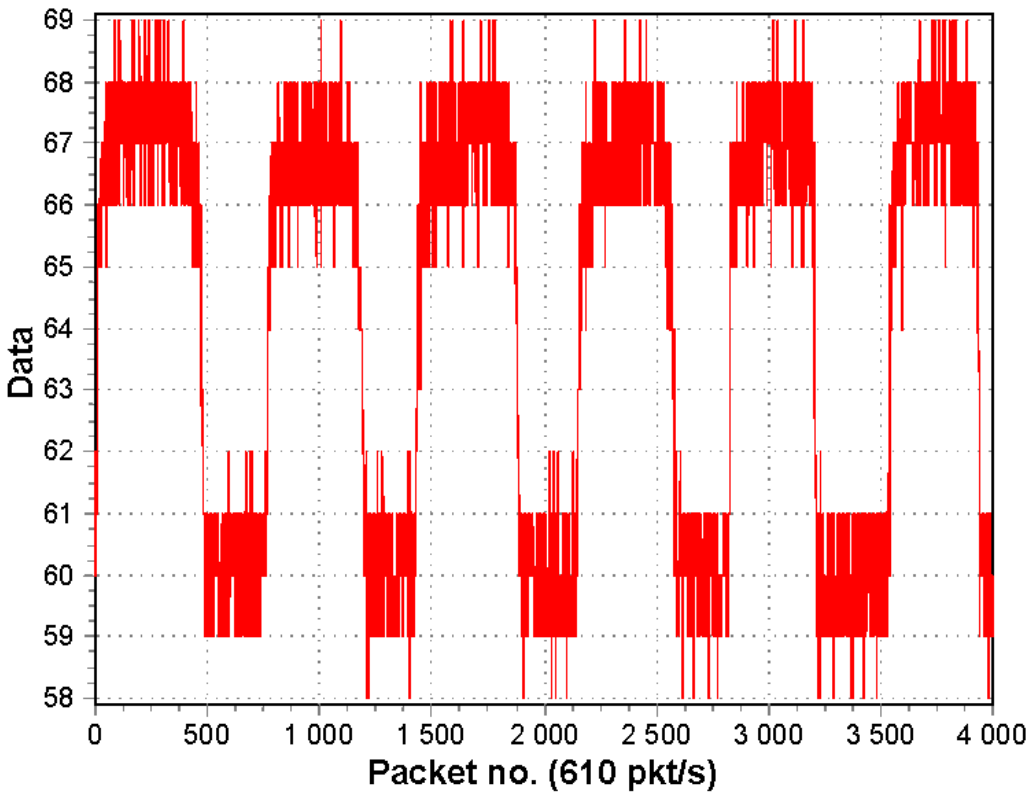
ここではテスト サンプルとして、CY3214 ボードを使った典型的な CapSense の用途で、ユーザ モジュールを設定する際の基本的なガイドラインを示します。検知ゾーンは、2 mm のプラスチック オーバーレイでカバーされています。CSD ユーザ モジュールのパラメータは、以降のステップで設定します。

1. ターゲット ボードを準備します。ターゲット アプリケーションの PCB を組み立て、オーバーレイをそれに固定します。この固定には、接着剤や特殊粘着テープを使用します。感度が著しく下がり、タッチで不明瞭なシフトが発生して誤ったボタン トリガが発生するため、PCB とオーバーレイ間に空気が入らないように注意してください。
2. データをモニタする、リアルタイムのモニタリング ツールを設定します。CSD のコンフィグレーション時には、1 つ以上のデータ シリーズをリアルタイムで監視できる PC チャート作成ツールを使用してください。ユーザ モジュールの調整時には、Raw カウント、Baseline 値、Difference 値を監視する必要があります。これには、I^2C-USB ブリッジを使用することが可能です。本テストでは、生カウント データのモニタにこれを使用しました。また、USBUART ユーザ モジュールを使用して、エミュレートしたシリアル ポートからデバッグ情報を送信する方法もあります。LCD やその他の数値ディスプレイは表示が遅く、動的なデータの変化を視覚化できないため、カウントのモニタには使用しないでください。

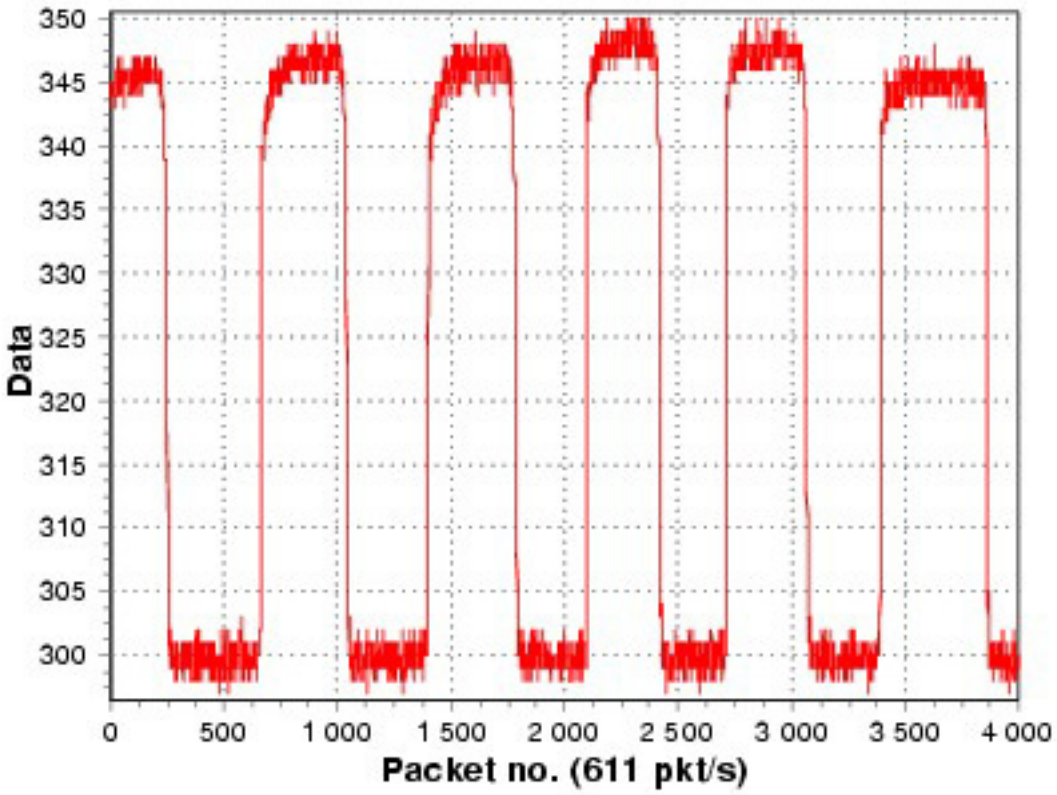
3. 初期構成を設定します。この構成では、16-bit PRS を使用します。次のパラメータが、テスト前に PSoC Designer で設定されました。

User Module Parameters	Value
FingerThreshold	40
NoiseThreshold	20
BaselineUpdateThreshold	200
Sensors Autoreset	Enabled
Hysteresis	10
Debounce	3
NegativeNoiseThreshold	20
LowBaselineReset	50
Scanning Speed	Fast
Resolution	9
Modulator Capacitor Pin	P0[5]
Feedback Resistor Pin	P3[1]
Ref Value	2
ShieldElectrodeOut	None

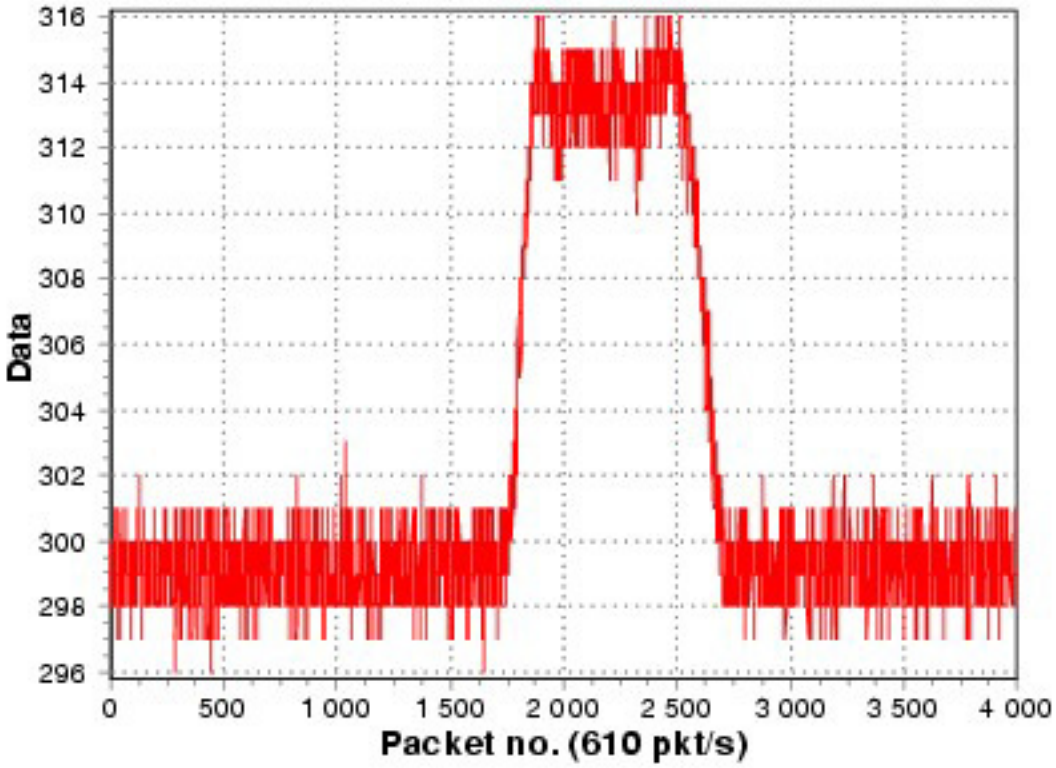
4. CSD ウィザードでセンサ ピンを割り当てます (スキャン用にセンサ P5[7]、P3[7]、P3[6] を割り当て)。
5. アプリケーションとサンプル コードを生成します。
6. チャート作成ツールを使ってセンサの Raw カウントをモニタし、ユーザ モジュール動作していることを確認します。センサにタッチすると、R a w カウント (CSD_waSnsResult 変数) が 59 ～ 68 に変化するはずです。



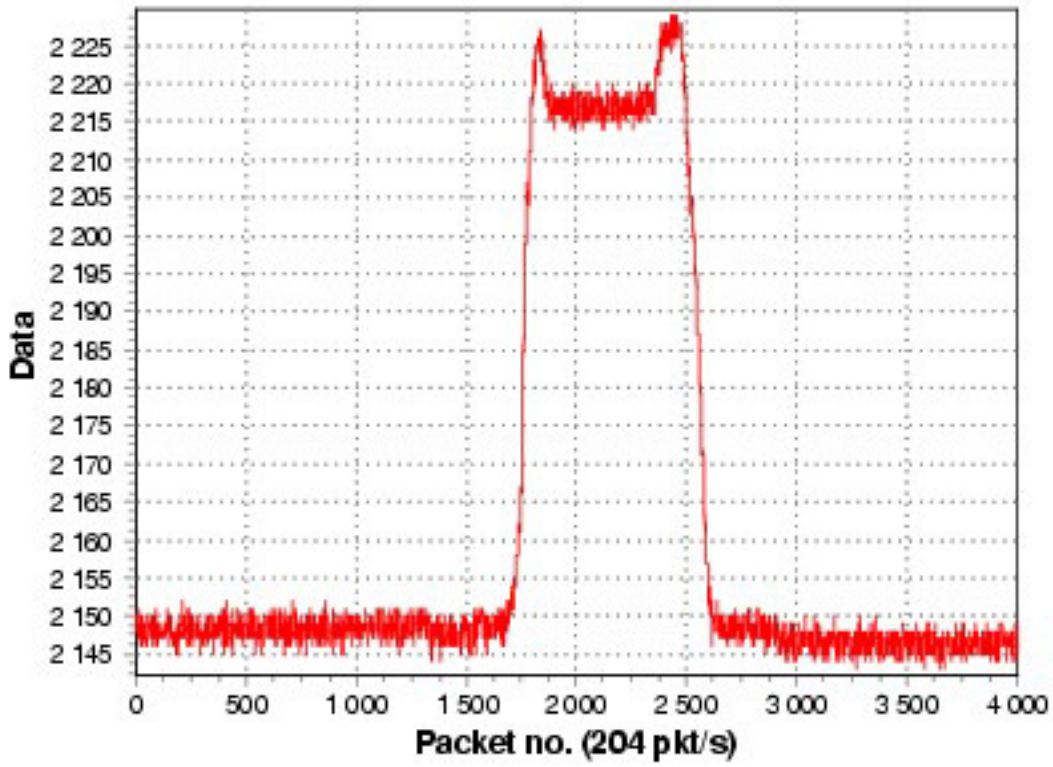
7. 外部コンポーネントを調整します。サイプレスは、5.6 nF 変調コンデンサ (C_{mod}) および 1.2 kΩ フィードバック レジスタ R_b を最初に使用しました。タッチ条件下で異なるセンサからのＲａｗカウント値を監視した後で、最大のＲａｗカウント値を生成するセンサを見つけました。このセンサからの信号は、前の図に表示されています。より低い信号値は指のタッチがないことを、高い信号値はタッチ条件を示します。このセンサからの信号値を解析すると、システムが、静電容量からコードへの変換器のダイナミック レンジをわずか 8% しか使用していないことがわかります。9-bit 分解能のフルレンジは Nm = 512 で、最大生カウントはおよそ 85 です。これは、フィードバック レジスタ値を 5.1 kΩ まで高めると、ダイナミック レンジの利用率が 60-70% 上がることを意味しています。Raw カウントの監視結果に応じて、この作業に異なるレジスタ値を使用することができます。レジスタを置き換えた後、指の応答は次のようになります。指のタッチからの応答が高くなります。



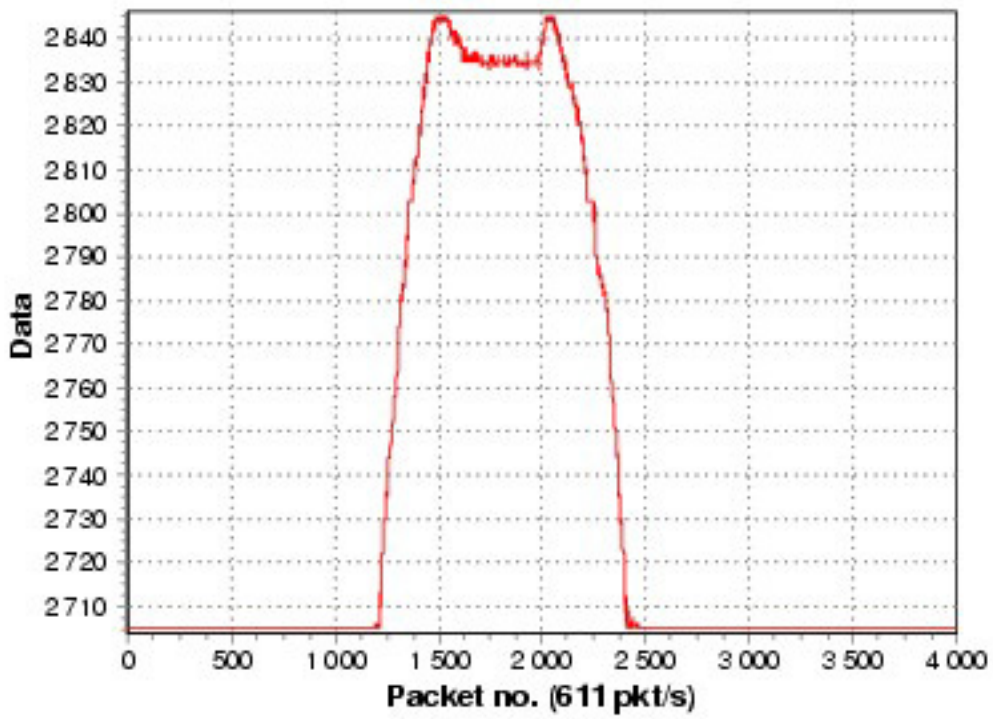
8. ワーストケースに合わせて調整します。指のシミュレータを使用して、非常に軽いタッチなど、異なる条件下でデバイスが安定して動作することを確認します。10 mm の接続していないコイルをオーバーレイに配置して、ワーストケースをシミュレートします。マッチや爪楊枝などの誘電性の物体を使って、ボタン上でコイルを移動させます。以下の図に結果を示します。ボードがセンサ周辺でグランド プレーンを使用している場合、このテストを実行できます。ボードがグランド プレーンではなくシールド電極でカバーされている場合は、指で非常に軽くタッチして、ワーストケースの応答をシミュレートできます。



9. コイルからの信号を識別できますが、安定した検出を行うには、SNR が小さすぎます。差はわずか約 9 dB です。検出感度を上げるには、より高いスキャン分解能を選択します。このテストでは、分解能を 9 ビットから 12 ビットに上げました。この設定における、コイルからの信号は次のとおりです。



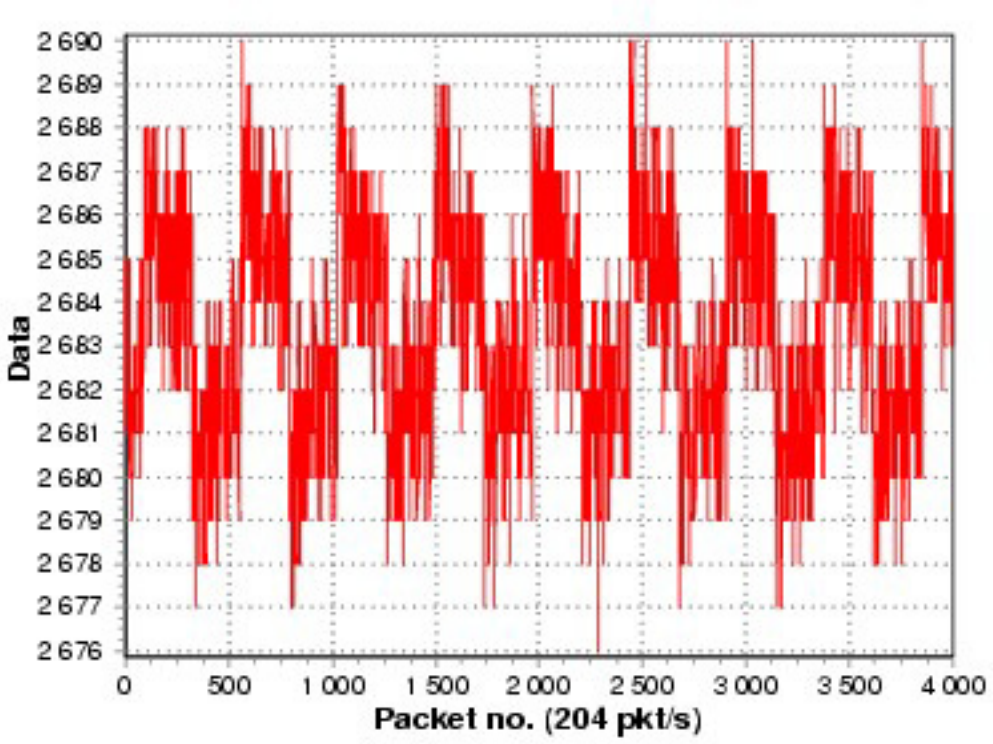
10. スキャン分解能を 9 ビットから 12 ビットに上げると、SNR が 25 dB となり、ほとんどの実用的な用途で利用できるようになります。人間の指からの信号はこれよりもはるかに大きくなります。この場合のデメリットは、スキャン時間が長くなることです。スキャン時間が重要な用途では、PRS8 構成に切り替えることができます。同じ UM パラメータ (PRS Poly は Short に設定) における PRS8 構成からのコイル応答は次のとおりです。



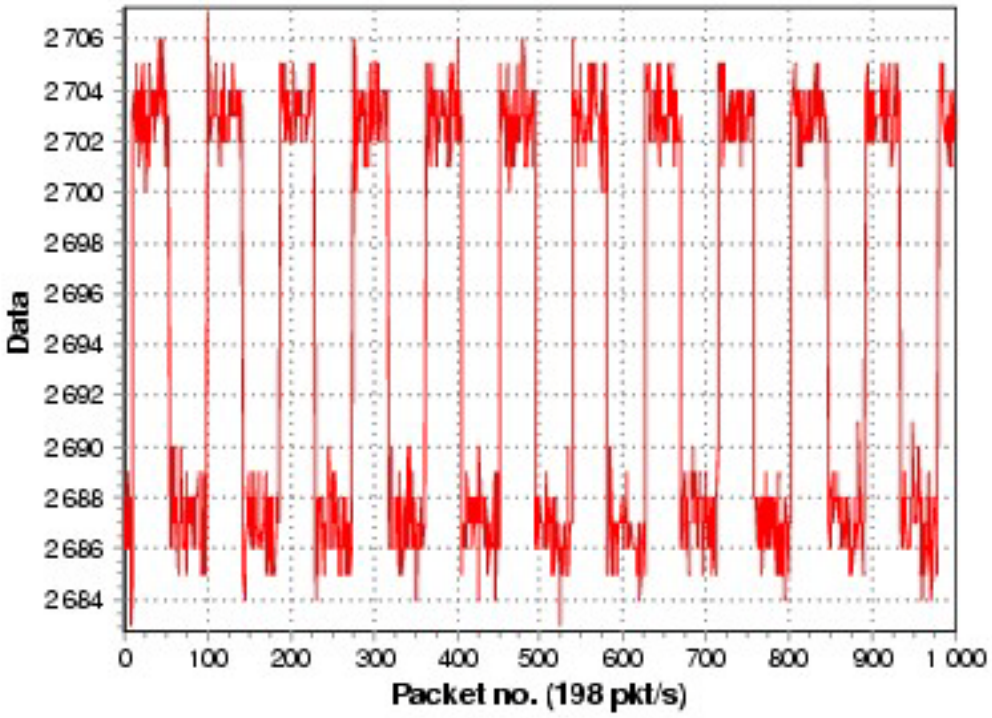
11. この構成は、PRS Poly が Short のとき、PRS8 構成より優れた SNR を提供します。しかし、より短い疑似乱数シーケンスでは、外部電気への耐ノイズ性を悪化させることがあります。
12. 閾値を設定します。ユーザ モジュールのパラメータで次の変更を行います。

User Module Parameters	Value
FingerThreshold	40
NoiseThreshold	20
BaselineUpdateThreshold	200
Sensors Autoreset	Enabled
Hysteresis	10
Debounce	3
NegativeNoiseThreshold	20
LowBaselineReset	50
Scanning Speed	Fast
Resolution	12
Modulator Capacitor Pin	P0[5]
Feedback Resistor Pin	P3[1]
Ref Value	2
ShieldElectrodeOut	None

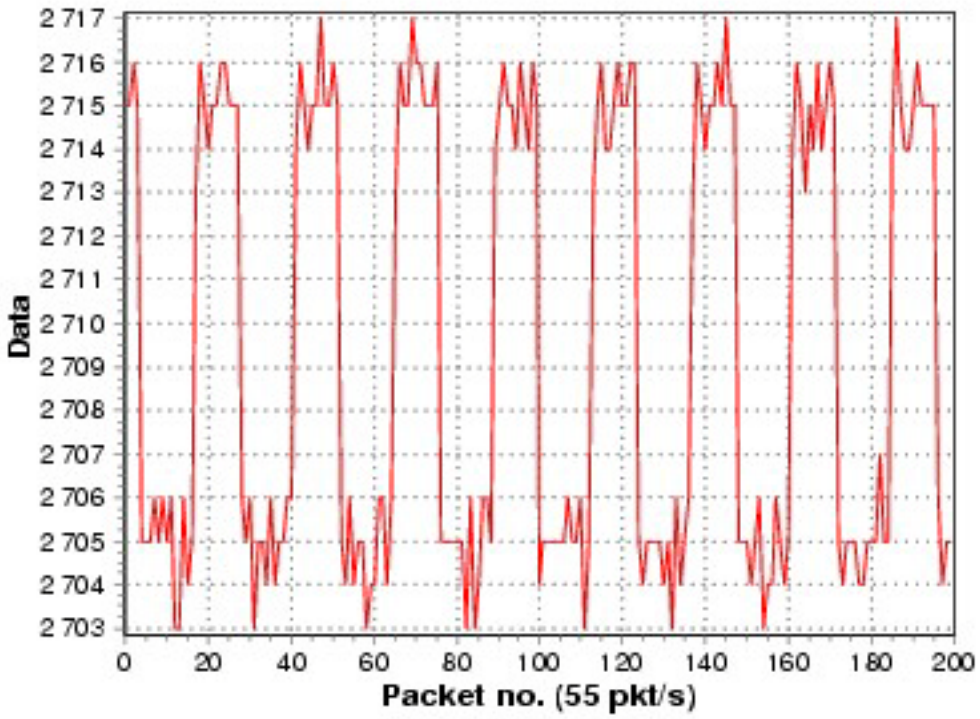
13. 最適なスキャン速度を設定します。テスト アプリケーションの電源電圧が安定していず、ターゲット デバイスの他パーツの動作により、電源が ± 5% で急激に変動すると想定します。また、PSoC デバイスが、CapSense 機能を使って 10 mA LED を複数駆動させると想定します。内部ダイ抵抗での電流低下により、内部電源電圧が変動します。CapSense システムは、この電圧過渡現象下でも動作し続けるはずです。こうした変動により、R a w カウントがどのように変化するかをテストします。LED は、同時にオンおよびオフになる必要があります。この動作では、スリープ タイマの割り込みが理想的です。また、外部のパルス源を使って、オンとオフの外部負荷をシミュレートすることもできます。以下の図は、スキャンがアクティブである間に、LED のオンとオフが切り替えられた場合の Raw カウントを示します。



14. このグラフからわかるように、スキャンがアクティブな間の LED のオンとオフは、Ｒａｗカウント値に目立った影響を与えません。急激な電源の変化に対して CapSense の安定性をテストします。非常にゆっくりとした電源の変化は、基準値更新アルゴリズムで処理され、ほとんどの場合は問題になりません。このテストでは、LM1117-ADJ 電圧レギュレータが使用されました。出力電圧は、外部信号源で駆動される MOSFET を使って、フィードバック レジスタ ネットワークの変化により変調されました。以下の図は、電源が 4.75 V と 5.25 V 間で揺らぐ場合の、センサにおけるＲａｗカウントの違いを示しています。



15. このグラフからわかるように、電源の過渡現象による生カウントの変化 (18) は閾値 (35..45) に近く、タッチを誤検出することはありません。非常に軽いタッチでは、複数のタッチ トリガが検出されることがあります。これを解決するには、ユーザ モジュールのパラメータでヒステリシスを上げます。また、スキャン速度を下げて、電源の変動による影響を軽減することも可能です。以下の図は、スキャン速度を低くして収集された Raw カウント データを示しています。



16. このグラフが示しているように、スキャン速度を下げると、電源電圧の変化が Raw カウントに与える影響を軽減することができます。過渡現象による差はおよそ 10 カウントになりました。これは閾値を大きく下回っており、CapSense モジュールの動作に悪影響を与えることはありません。この場合のデメリットはスキャン速度が 4 倍になることであり、場合によっては悪影響を引き起こすことがあります。
17. BaselineUpdateThreshold パラメータを調整します。この用途では、タッチの最長検出時間が 1sec 未満である必要があります。SensorsAutoreset パラメータを [Enabled] (有効) に設定します。BaselineUpdateThreshold が、環境の変化を補正するのに十分なＢａｓｅｌｉｎｅ値更新速度に設定されていることを確認します。たとえば、ボード上の空気が冷やされ急激に温度が変化する可能性があるキッチン家電では、この温度変化のために Raw カウントが急に下がることがあります。Ｂａｓｅｌｉｎｅ値は Raw カウント値に自動的にリセットされることにより、これを追跡します。そのため、環境要因によりＲａｗカウントが下がっても、ほとんどの場合は問題になりません。Ｒａｗカウントが温度の変化によって上がる場合は、この変化がタッチとして解釈され、偽タッチがトリガされる可能性があります。温度や他の環境要因による Raw- Baseline 値の変化が指の閾値を十分に下回るよう、Baseline 値更新速度を調整しなければなりません。このテスト時には、Ｒａｗ - Baseline 値の差の変化がモニタされました。モニタされた値は 0 であり、変化がノイズ閾値パラメータを下回っています。このパラメータは、テスト中に最小値 5 に設定されました。これは、前もってセットする BaselineUpdate 閾値パラメータが、十分な基準値追跡速度を提供し、温度の変化がこの用途では問題にならないことを意味しています。

18. 全パラメータを設定した後、ESD テストを実行することができます。ESD Debounce パラメータが Disabled (無効) に設定されている場合でも、問題なくこれらのテストをパスするはずです。ESD テストで問題がある場合は、必要に応じて、ESD Debounce パラメータを有効にしてください。このパラメータを有効にした場合のデメリットは、RAM バッファのサイズが増えることです。
19. CapSense の多くの用途では、さまざまな EMC/EMI 互換テストにパスする必要があります。EMC/EMI で問題が発生する場合は、AN2318 *「PSoC CapSense アプリケーションの EMC 設計に関する考慮点」*を参照して、問題を修正してください。また、PRS クロックを遅くしてセンサのパス放射を低減させることにより、この問題を解決することもできます。プリスケーラを使った構成を試すか、IMO モードを遅くしてみてください (24MHz ではなく 6MHz で SYSCLK を実行するなど)。PRS クロック周波数やプリスケーラ期間の設定を変更した場合、フィードバック レジスタを調整してダイナミック レンジを最大限に使用し、最大の検出感度を達成する必要があります。
20. EMC テストにパスしない場合は、スキャン速度を下げ、分解能を上げてみてください。長い PRS polynomial で、耐ノイズ性が改善されます。この場合のデメリットは、センサのスキャン時間が長くなることです。

トラブルシューティング

- UART ボーレートのクロック源として、プリチャージのプリスケーラを使用することができます。UART の推奨速度は、115,200 ボー以上です。プリスケーラ値は、24 MHz の IMO 動作で 25 に設定します。この値は 2^N の倍数ではないため、SNR を改善するため、スキャン速度を落とすことをお勧めします。実験を行ってこれをテストしてください。
- 基準設定において周期的で大きなノイズが発生する場合は、**CSD.asm** ファイルで CSD_DELAY 定数を上げてみます。この値は、測定開始前に変調器の始動時間をどれだけ遅らせるかを設定します。また、変調コンデンサの C_{mod} 低減を軽くしても問題を解決できることがあります。このノイズは、内部アナログ変調器のローパス フィルタで時間定数が低いため、変調コンデンサが前回の測定周期で異なる電圧にチャージされることが原因です。
- スキャン速度と分解能は、信号対雑音比 (SNR) に影響を与えます。スキャン速度を遅くして分解能を高めると、場合によっては SNR が改善されます。
- 電極のオーバーレイが厚い場合は、分解能を高くして、スキャン速度を遅くしなければならないことがあります。
- PRS 多項は、PRS シーケンスの繰り返し期間がサンプルの変換周期カウントに近づくよう、スキャン速度と分解能に応じて自動的に調整されます。スキャン レートを遅くして分解能を高くすると、PRS シーケンスが長くなるため、EMC テスト中の耐ノイズ性が改善されます。
- スキャン速度を遅くすると変調器動作周波数が低くなり、読み取りがコンパレータの動的特性に依存しにくくなります。電源の変動にかかわらず Raw カウントを安定させる必要がある場合、または PSoC デバイスが高い電流負荷を制御する場合には、アナログ変調器を使ってコンパレータリファレンスを内部的に生成してください。この場合の推奨スキャン速度は、Normal (標準) または Slow (低) です。
- デルタシグマ変換メソッドは、積分クラスの一部です。これは、より高い分解能において最高のパフォーマンスを示します。可能な限り最長のスキャン時間を使ってください。最高の結果を得るには、センサのスキャンを 1 ms で行います。
- 耐水を必要としない用途でも、浮遊静電容量の影響を軽減するには、シールド電極の使用が効果的です。この場合、静電容量式センサ ゾーンの下にある PCB の最下層にシールド電極を配置できます。このときハッチパターンを使用して、シールド電極の静電容量を下げることをお勧めします。

リソース利用の競合を避ける方法

このユーザ モジュールで使用されるハードウェア構成を変更しないよう注意してください。これには、以下のものが含まれます。

- GlobalOutEven_1 または GlobalOutEven_5 (変調器フィードバック レジスタのピン選択による) バスは、コンパレータの出力信号を出力バスに渡すため内部的に使用されます。これらのバスには、ソースを接続しないでください。
- コンパレータ バス 1 の LUT 関数を変更しないでください。コンパレータ Bus_1 は、**A** に設定してください。
- アナログ コラム 1 クロック源は、**VC1** に設定してください。
- VC1 は、ユーザ モジュールで内部的に設定されます。グローバル リソースで入力される値は、ランタイム時に上書きされます。
- シールド電極を使用する場合は、Row LUT 関数を **A** に設定してください。

割り込み持続時間の管理

センサ スキャンが PRS16 構成でアクティブである場合に、割り込みサービス ルーチン (ISR) 持続時間を管理します。8-bit タイマは、VC2 からクロックを得ます。タイマに関するワーストケースのオーバーフロー間隔は次のとおりです。

Equation 2

$$T_{owf} = VC_2 \cdot VC_1 \frac{256}{F_{IMO}}$$

F_{IMO} – IMO 周波数、高速、標準、低速のスキャン速度でそれぞれ VC1 = 2、4、8

VC_2 – この値は、常時 4 に設定されます。

ほとんどの場合、この間隔で問題ありません。しかし場合によっては確認が必要となります。

ISSP ピンの競合

低抵抗フィードバック レジスタを P1[1] ピンに永続的に接続すると、ISSP プログラミング エラーが発生します。これを解決するには別のピンを使用してください。

クロック速度

正しく機能するよう、CY8C24x94 デバイスの CPU 速度は、SysClk/32 以上にしてください。

改訂履歴

バージョン	作成者	説明
1.4	DHA	最大分解能は 3000 です。0.5 シフトを削除し、負の値に対して補正を追加しました。 ウィザードのピン リストが修正されました。
1.50	DHA	CY8C21x12 デバイスのサポートを追加しました。

Note PSoC Designer 5.1 から、全ユーザ モジュール データシートに改訂履歴を追加しました。このセクションは、現在および以前のユーザ モジュール バージョン間の差をハイレベルで説明するものです。

Copyright © 2007-2011 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer ™ and Programmable System-on-Chip ™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.