Preliminary

Autotuning CapSense®Sigma-Delta（シグマデルタ）データシート CSDAUTO V 1.20

Copyright © 2009-2011 Cypress Semiconductor Corporation. All Rights Reserved.

リソース	PSoC® ブロック				API メモリ		ピン（外部 I/O あたり）
	CapSense	I²C/SPI	タイマ	コンパレータ	Flash	RAM	
CY8C20x66, CY8C20x36, CY8C20336AN, CY8C20436AN, CY8C20636AN, CY8C20x46, CY8C20x96, CY7C645xx, CY7C643/4/5xx, CY7C60424, CY7C6053x, CYONS2110, CYONS21L1T, CYONSFN2162							
	1	-	1	1	1540	35	0

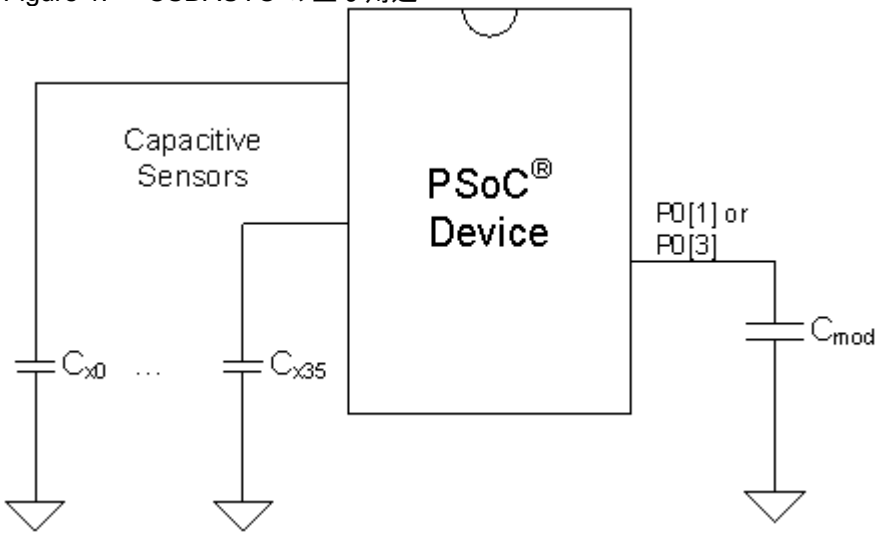
特性および概要

- AutoTuning アルゴリズムは、各センサの寄生容量に基づき、ランタイムの稼動パラメータを最適化
- 1 ~ 36 個の静電容量式センサをスキャン
- 最大 50 pF までのセンサの静電容量（Cp）を用いて、0.1 pF のタッチを検知可能。これには、アプリケーション ノート「静電容量検知 - PSoC CapSense のためのレイアウトガイドラン」を AN2292 厳守することが必要です。
- ガラスのオーバーレイで最大 15 mm まで検知可能。
- AC 電源ノイズやその他の EMC、また電源電圧の変化によるノイズ等に対する高耐性。
- 独立したボタンとして構成された静電容量センサや近接検知センサ、独立アレイとしてスライダを構成するセンサをサポート スライダと近接検知センサはこのベータ版では完全にはサポートされていません。
- ダイプレックス技法を用いて、IO ピンを増やさずに、効果的にスライダセンサー数を倍増可能
- 補間法を使用して物理ピッチより多いスライダ分解能をサポート
- タッチパッドは、網の目状になった直交スライダ ペアとして実装が可能
- シールド電極を使用することで、水の中、水膜等に対しても信頼性の高いオペレーションを提供
- CSDAUTO ウィザードを使用し、簡単にセンサーやピン配置が可能
- リアルタイムで Raw データのモニタリングを可能にする PC GUI アプリケーションサポート

CSDAUTO は、スイッチド キャパシタ電流をデジタル値に変換するデルタシグマ変調器を使用した、スイッチド キャパシタ手法による静電容量式検知を行います。

Note このユーザーモジュールは、C 言語のプロジェクトのみをサポートします。ASM(アセンブリ言語) のプロジェクトはサポートされません。

Figure 1. CSDAUTO の主な用途



機能説明

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
  - 物理センサには、通常、PCB 上で PSoC に接続された導電性のパターンを備え、その上に絶縁性のカバー（薄膜やディスプレイ上の透明なオーバーレイ）を含みます。
- 電気コンポーネント
  - センサのキャパシタンス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 コンデンサ、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。
  - 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。別個のシールド電極を使って、センサアレイをシールドして浮遊静電容量を低減します。これにより、水膜や水滴がある場合でも動作の信頼性が確保できます。
  - 高レベルのソフトウェア機能によって、スライダのダイプレックスがサポートされます。これにより、単一 I/O ピンが 2 つの物理センサにルーティングされ、スライダ エレメント数に対して半分の IO を消費するだけで済みます。

CSDAUTO ユーザモジュールを使用して CapSense デザインを実施する前に、次の文書を一読するよう推奨されています。

[+] Feedback

- *PSoC® CY8C20x66, CY8C20x66A, CY8C20x46/96, CY8C20x46A/96A, CY8C20x36, CY8C20x36A, CY8CTMG20x, CY8CTMG20xA, CY8CTST200, CY8CTST200A 技術レファレンス マニュアル* (TRM) セクション :
  - CapSense システム

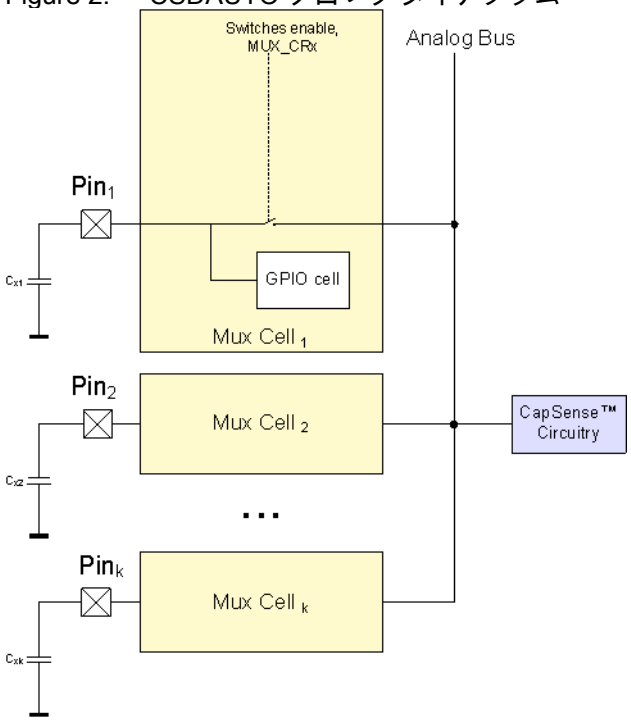
次のアプリケーションノートは、CSD ユーザ モジュールのマニュアルを読んだあとに読むよう推奨されています。アプリケーション ノートは、サイプレス セミコンダクタのウェブサイト (www.cypress.com) からご覧いただけます。

- *CapSense のベスト プラクティス* – AN2394
- *CapSense アプリケーションでの信号対雑音比要件* – AN2403
- *設計支援 - CapSense データ表示ツール* – AN2397
- *PSoC CapSense アプリケーションの EMC 設計における考慮点* – AN2318
- *電源とスリープの考慮* – 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

静電量検知操作

CY8C20x66 のデバイスファミリは、Analog Mux Bus（アナログ多重化バス）を使用しています。これにより、CapSense 回路を任意の PSoC ピンに制御できます。CSDAUTO ユーザ モジュールは、アクティブなセンサを Analog Mux Bus（アナログ多重化バス）に接続します。CapSense 回路は常に接続されているため、センサの静電量を測定し、その静電量をデジタルコードに翻訳できます。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. CSDAUTO ブロック ダイアグラム

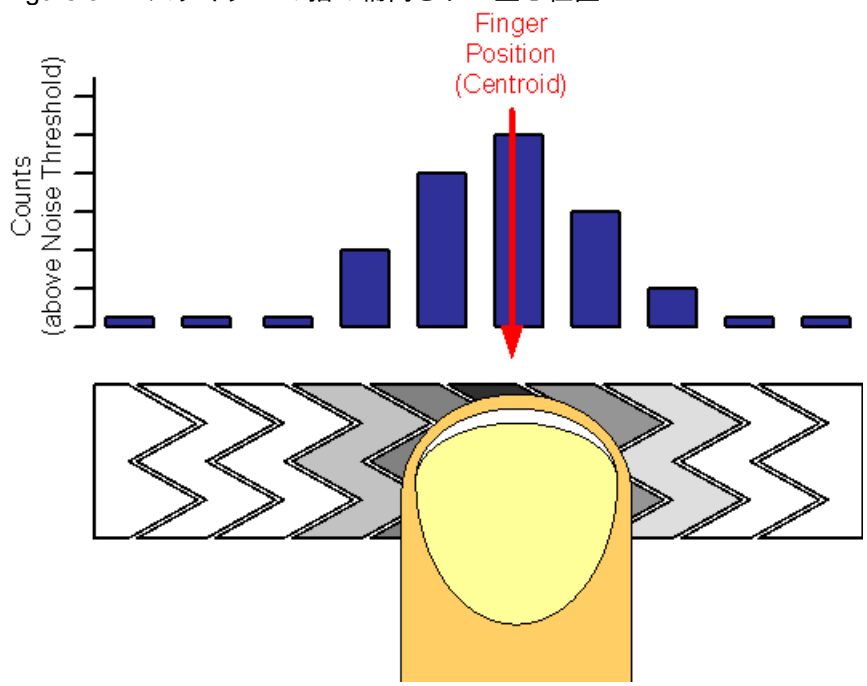


スライダ

Note スライダと近接検知センサはこのベータ版では完全サポートされてはいません。

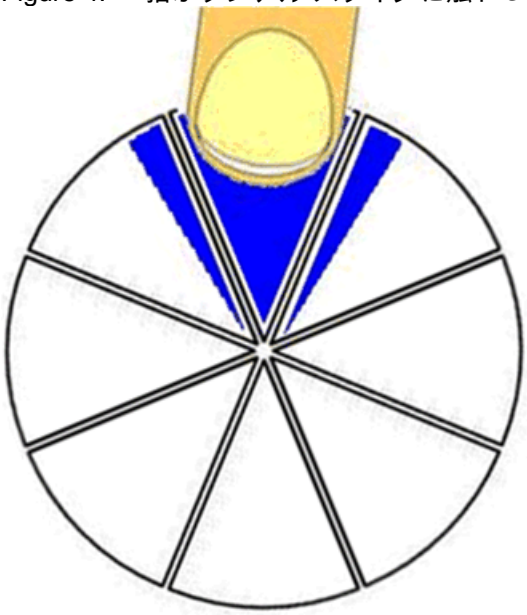
スライダは、漸次的調整を必要とする制御に使用します。たとえば、照明管理 (調光装置)、音量管理、グラフィック イコライザ、速度制御などがあります。スライダを構成するセンサは、それぞれ隣接しています。1 つのセンサの作動は、物理的に近接するセンサの部分的な作動につながります。スライダの実際の位置は、作動したセンサ セットの重心位置を計算することによって判断できます。スライダは、CSDAUTO ウィザードで、各スライダグループがそれぞれ特定の順序を持つように設定されます。センサ スライダの実質的な下限は 5 で、上限は、選択した PSoC デバイスで利用できるセンサ数になります。

Figure 3. スライダ上の指の補間された重心位置



ラジアル スライダ

Figure 4.　指がラジアル スライダに触れる



CSDAUTO 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

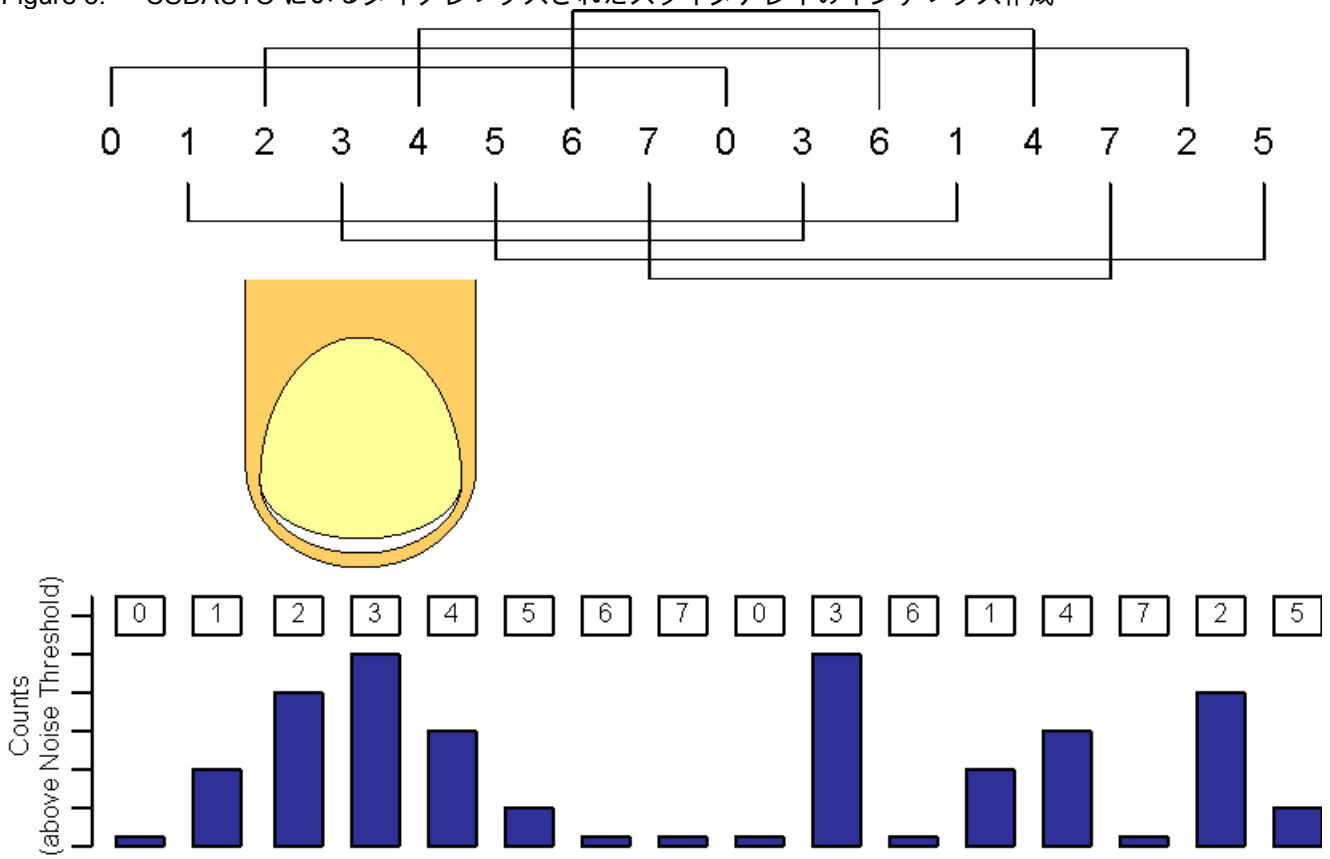
CSDAUTO UM には、ラジアル スライダをサポートする 2 つの API 関数があります。最初の関数 CSDAUTO_wGetRadiaPos() は重心位置を返し、2 つ目の関数 CSDAUTO_wGetRadialInc() が分解能単位で指のシフトを返します。指が時計回り方向に移動すると、正のオフセットとなります。

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

ダイプレックスを使用する場合、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。スライダセンサーの最初の半分 (数字が小さい) は、CSDAUTO ウイザードでデザイナが割り当てたピンを各センサーに連続的にマッピングします。物理的位置の後の (数字が大きい) 半分は、下図に示すように、ウィザードのアルゴリズムで自動的にマッピングされます。

Figure 5. CSDAUTO によるダイプレックスされたスライダアレイのインデックス作成



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。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。

印刷基板上のピンにセンサをマップングすると、ユーザ モジューが使用する 3 パターンのインデックスと適合します。ダイプレックスされたスライダのセンサペアの静電量は、かなりの確度で適合します (10 pF 以内)。

ダイプレックスセンサ指数表は、ダイプレックスを選択すると CSDAUTO ウィザードによって自動的に作成されます。この表は、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。

Table 1. 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

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
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
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

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 の数と分解能は、CSDAUTO 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。

外部コンポーネント選択のガイドライン

CSDAUTO は、Vss グランド端子から P0[1] または P0[3] ポートピンに接続されている外付け変調コンデンサ C_{mod} を使用できます。ピンはピン選択ウィザードの設定で選択します。選択されたピンは、他の目的で使用することはできません。

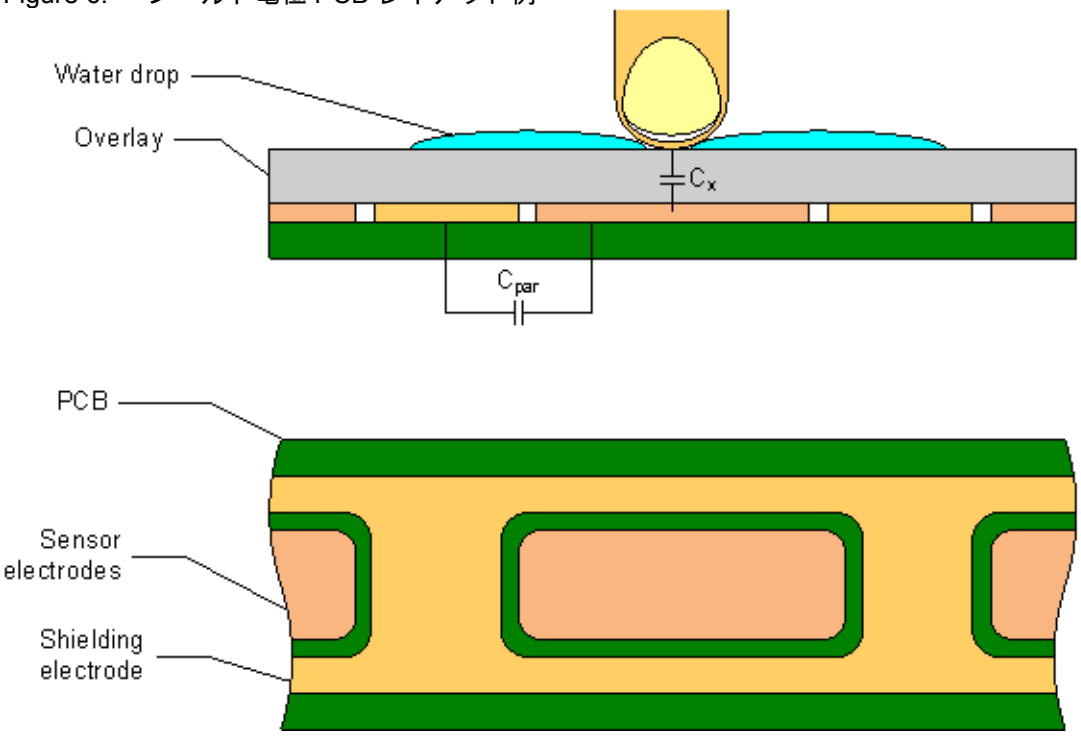
CSDAUTO が内部コンデンサとともに作動できない場合は、外付けコンデンサをお勧めします。モジュレータ用のコンデンサに推奨されている値は、2.2 nF です。最適な静電容量は、最大 SNR を得るための実験を行うことで決定することができます。ほとんどの場合、2.2 nF の値で良い結果が得られます。セラミックのコンデンサを使用するよう強く推奨します。温度静電容量係数は重要ではありません。

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。水膜が遮断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、高レベルソフトウェアの API 作業が簡単になります。CSDAUTO ユーザ モジュールは、シールド電極の個々の出力に対応します。

Figure 6.　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。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、Cpar 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。

CSDAUTO は、シールド電極を駆動するために、クロックをプリチャージするものと同じ信号を使用します。

シールド電極は、専用の PSoC ピン（ P0[7] または P1[2] ）に接続されています。選択されたピンの駆動モードは 「Strong」にセットします。放出される EMI を低減するために、スルーを制限するレジスタを、PSoC デバイスとシールド電極の間に接続することができます。

DC 電気的特性と AC 電気的特性

Table 2. 電源要件

パラメータ	最小値	典型値	最大値	単位	テスト条件とコメント
Vdd	1.7	5.0	5.25	V	適用外

Table 3. 様々な Cp における信号とノイズ

寄生容量、Cp (pF)	0.1 pF タッチの信号（カウント）	ノイズ[a]（カウント）	SNR	スキャン時間（µs）
10				
15				
20				

a. このノイズ値は、AN2292 のガイドラインに従った基準設計に基づく平均的な値です。

配置

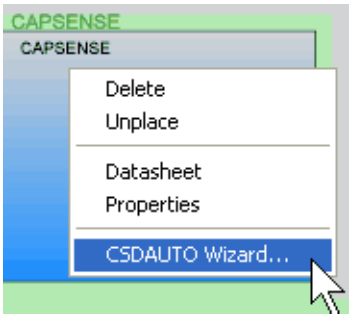
ユーザ モジュールのブロックは、ユーザ モジュールがインスタンス化されると自動的に配置されます。他の配置は利用できません。CSDAUTO ユーザ モジュールは、CapSense ブロックと 1 つのタイマ (タイマ 1) を使用します。

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、CSDAUTO ユーザ モジュールのピン接続を確立するために、CSDAUTO ウィザードを開始する前に配置しなければなりません。

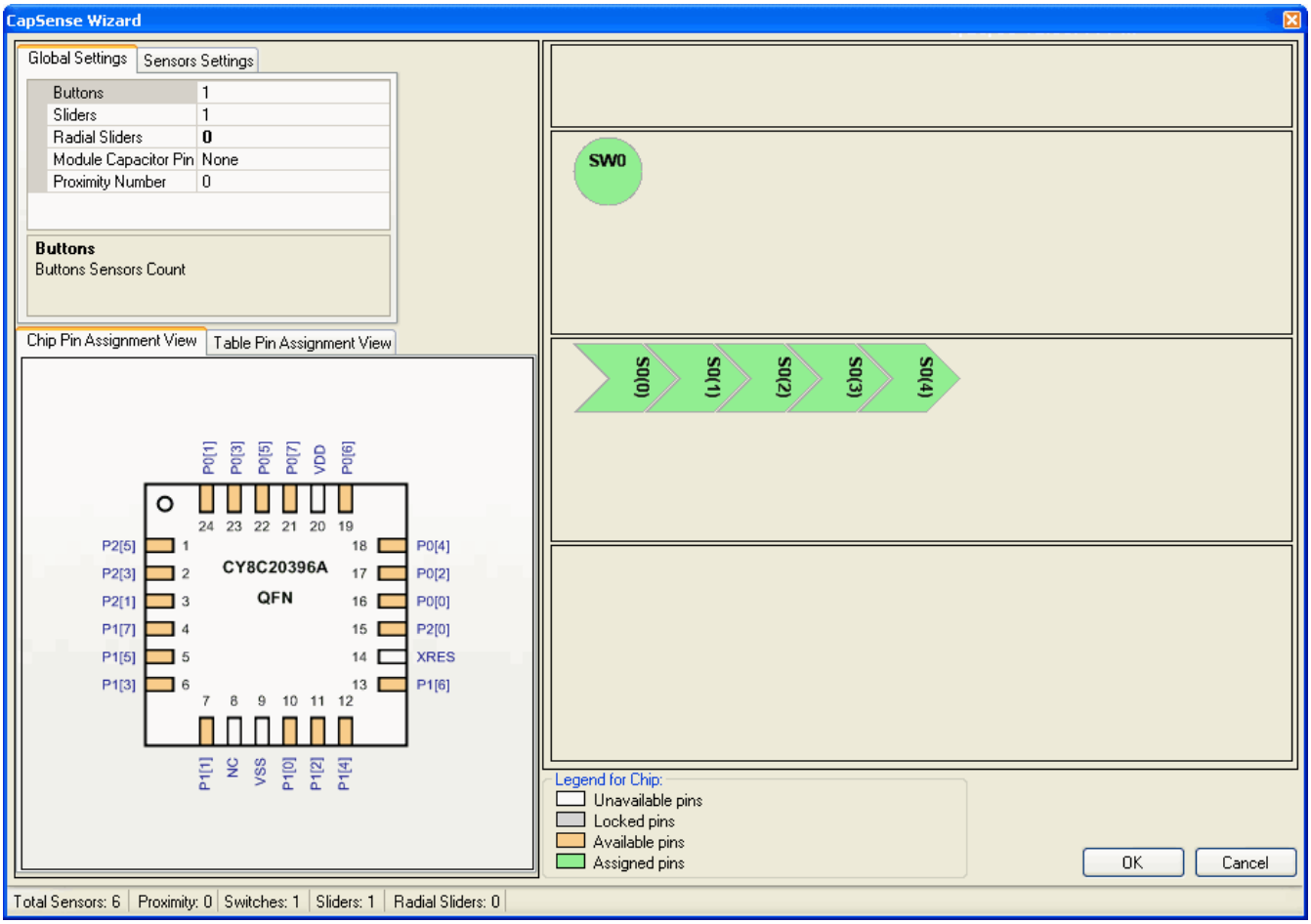
静電容量式センサの接続を配置する場合、P1[0] と P1[1] は避けてください。これらのピンは、そのデバイスのプログラミングに使用され、センサの検出感度とノイズに影響を与える過度のルーティング静電容量を持っている可能性があります。

ウィザードへのアクセス

1. ウィザードにアクセスするには、デバイス エディタの相互接続ビューで CSDAUTO の任意のブロックを右クリックし、次に [CSDAUTO Wizard] 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

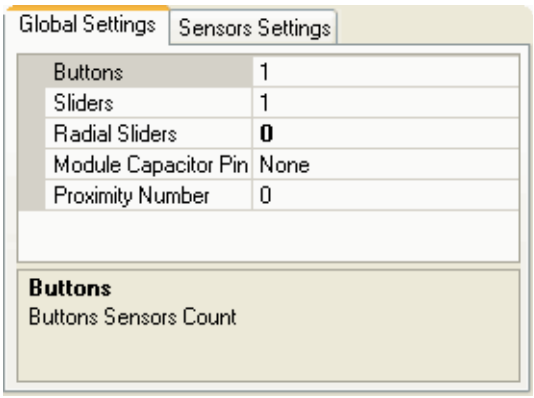
白 – このピンは CapSense の入力に使用できません。

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。

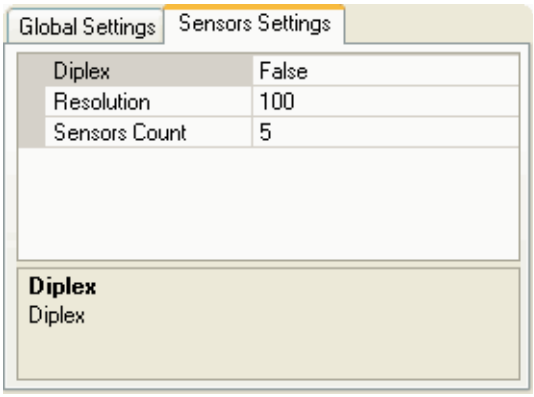
オレンジ – このピンは割り当て可能です。

グリーン – このピンは CapSense 入力に割り当てられています。

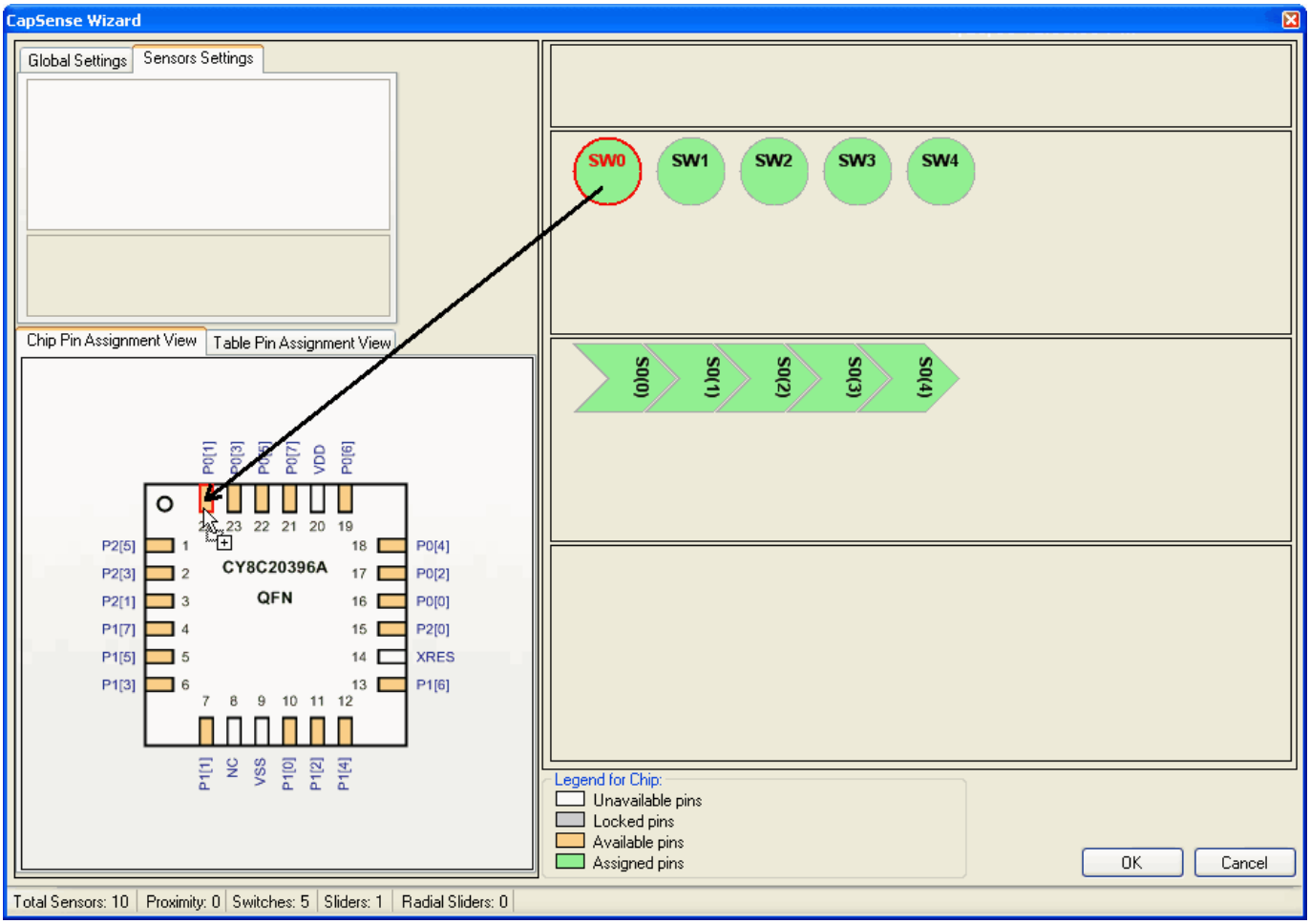
3. 独立したボタン、スライダ、ラジアル スライダの数を入力します。センサの合計数は、利用可能なピン数に制限されています（ボタンとスライダエレメント）。データを入力したら、[Enter] キーを押すと、画面が新しい値で更新されます。X-Y タッチパッドには 2 つのスライダが必要です。



4. [センサ設定] を選択し、スライダとラジアルスライダを設定をします。設定を変更するには、スライダをクリックして有効にします。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。データを入力したら、[Enter] キーを押すと、画面が更新されます。変調器コンデンサ（C_{mod}）のピンを選択します。利用可能なピン P0[1] と P0[3] から選択します。内部コンデンサが使用されている場合は、[None]（なし）を選択します。一般に、2.2 nF の外付けコンデンサを利用するほうが、より良い SNR が得られます。

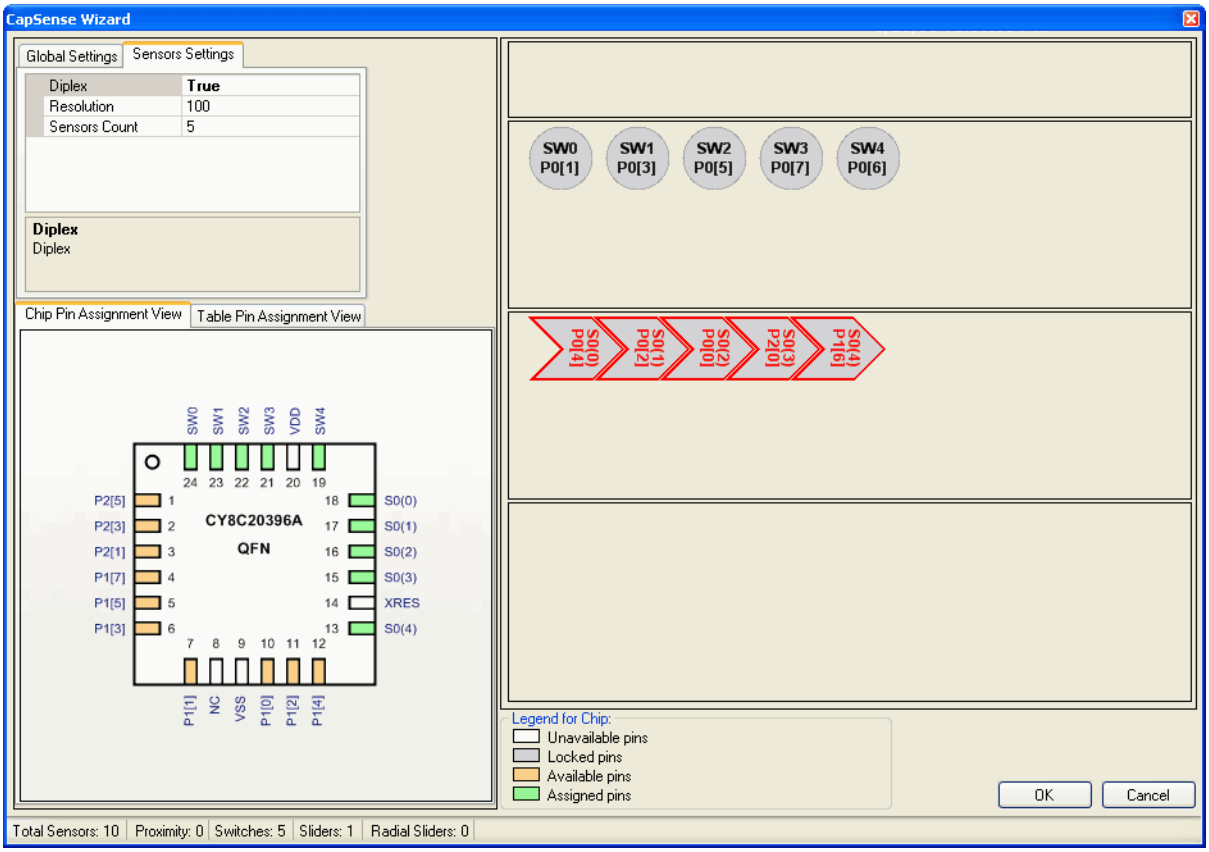


5. 出力の分解能を入力します。最低値は 5 です。最大値は、ダイプレックス スライダの場合、(センサに使用されるピン数 – 1) x 2_{16} – 1 または (2 x センサに使用されるピン数 – 1) x 2_{16} -1 です。ソフトウェアは、隣接セグメントの相対強度を使用して、タッチの結果を指定分解能に補間しようとします。ソフトウェアは、スライダのタッチの結果を、ゼロから分解能 – 1 の間の値で報告します。
6. 必要に応じて、ダイプレックスを選択します。これにより、センサ用に選択されたピンを、基板上で 2 倍の数のセンサ位置にマッピングできます。上図では、ダイプレックスセン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。
7. [Pin Assignment View] (ピン割り当てビュー) で、スイッチやセンサをピンにドラッグすると、スイッチやセンサをピンに割り当てることができます。または、[Chip Pin Assignment View] (チップピン割り当てビュー) や [Table Pin Assignment View] (表のピン割り当てビュー) で、スイッチやセンサをピンにドラッグすることもできます。ポートピンは一旦選択するとグリーンになり、使用不可となります。ポート ピンからボタンを未割当ての表へドラッグすると、センサの割り当てを変更できます。他のユーザモジュールに割り当てられているピンを選択しないよう注意してください。


CapSense Wizard
Global Settings
Sensors Settings
Chip Pin Assignment View
Table Pin Assignment View
SW0
SW1
SW2
SW3
SW4
S0(0)
S0(1)
S0(2)
S0(3)
S0(4)
CY8C20396A
QFN
Legend for Chip:
Unavailable pins
Locked pins
Available pins
Assigned pins
OK
Cancel
Total Sensors: 10
Proximity: 0
Switches: 5
Sliders: 1
Radial Sliders: 0

8. 他のセンサでも同じ操作を繰り返します。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。[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。

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 CSDAUTO_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、6 つのセンサを含む表の例を示します。

```
CSDAUTO_Sensor_Table:
_CSDAUTO_Sensor_Table:
      dw    0x0140  //  Port 1 Bit 6
      dw    0x0301  //  Port 3 Bit 0
      dw    0x0304  //  Port 3 Bit 2
      dw    0x0308  //  Port 3 Bit 3
      dw    0x0302  //  Port 3 Bit 1
      dw    0x0108  //  Port 1 Bit 3
```

この表は CSDAUTO_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、CSDAUTO ウィザードで望ましい分解能が達成されます。

```
CSDAUTO_Group_Table:
_CSDAUTO_Group_Table:
; Group Table:
;    Origin    Count    Diplex?    DivBtwSw(wholeMSB, wholeLSB, fractByte)
 db   0x0,     0x3,     0x00,      0x00,    0x00,    0x00 ; Buttons
 db   0x3,     0x8,     0x4,       0x0,     0x0,     0x44 ; 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、データは記録されません。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、8 センサ式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 0,1,2,3,4,5,6,7,0,3,6,1,4,7,2,5// 8 switch slider


 CSDAUTO_Diplex_Table:
_CSDAUTO_Diplex_Table:
 db >DiplexTable_0, <DiplexTable_0
 db >DiplexTable_1, <DiplexTable_1
```

パラメータおよびリソース

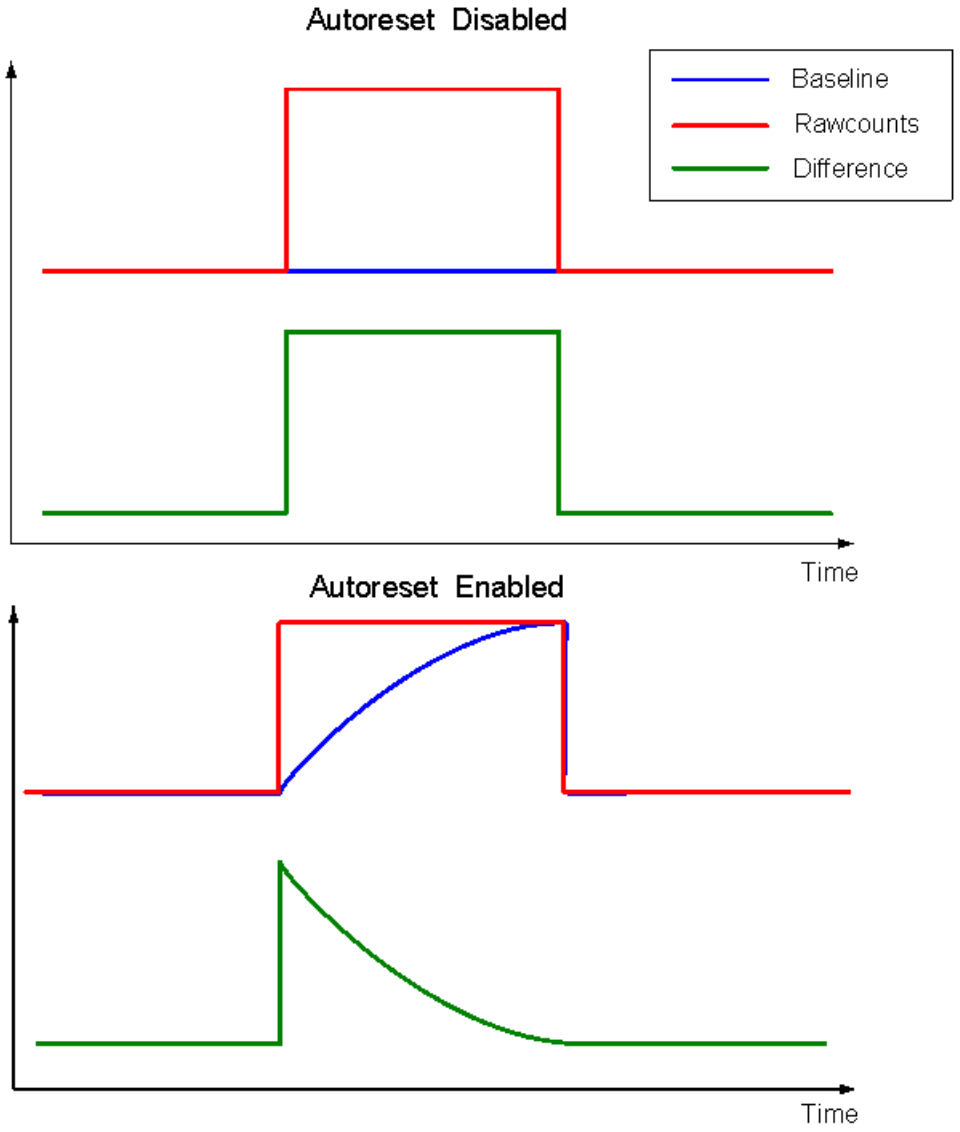
センサの Autoreset

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。何もセンサに触れずに Raw カウントが突然上がった際に、センサが永続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。

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 7.　センサ自動リセットパラメータ



[+] 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。センサが非作動中から動作中へ遷移するためには、指定されたサンプル数に対して、差分カウント値が指の閾値 + 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。

ShieldElectrodeOut

シールド電極信号源は、P0[7] または P1[2] に接続できます。

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) 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。

ユーザ モジュールを配置するたびに、インスタンス名が割り当てられます。デフォルトでは、PSoC Designer は指定されたプロジェクトで、CSDAUTO_1 をこのユーザ モジュールの最初の例として指定されています。これは識別子の構文ルールに従った一意の値に変更できます。割り当てたインスタンス名が、全てのグローバル関数名、変数、および定数記号の接頭語になります。次の説明では、簡単にするために、インスタンス名は省略されて単に 「CSDAUTO」となっています。

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。関数を呼び出す場合、呼出し後に A と X の値が必要になるならば、必ず呼び出し前に 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を処理します。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。

エントリポイントは、CSDAUTO を初期化し、サンプリングを開始し、CSDAUTO を終了するためのものです。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 CSDAUTO 接頭語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。次にいくつかのグローバルアレイを挙げます。

- CSDAUTO_waSnsBaseline[]
- CSDAUTO_waSnsResult[]
- CSDAUTO_waSnsDiff[]
- CSDAUTO_baSnsOnMask[]

CSDAUTO_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。CSDAUTO_waSnsBaseline[] アレイは以下の関数によって更新されます。

- CSDAUTO_UpdateAllBaselines();
- CSDAUTO_UpdateSensorBaseline();
- CSDAUTO_InitializeBaselines().

CSDAUTO_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。CSDAUTO_waSnsResult[] データは次の関数によって更新されます。

- CSDAUTO_ScanSensor();
- CSDAUTO_ScanAllSensors().

CSDAUTO_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。

CSDAUTO_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。CSDAUTO_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)。CSDAUTO_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。CSDAUTO_baSnsOnMask[] データは、CSDAUTO_bIsSensorActive(BYTE bSensnor) 関数または CSDAUTO_bIsAnySensorActive() ルーチンによって更新されます。

CSDAUTO_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  CSDAUTO_Start()
```

アセンブリ :

```
lcall  CSDAUTO_Start
```

パラメータ :

なし

戻り値 :

なし

特殊作用 :

**

CSDAUTO_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、CSDAUTO_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  CSDAUTO_Stop()
```

アセンブリ :

```
lcall  CSDAUTO_Stop
```

パラメータ :

なし

戻り値：

なし

特殊作用：

**

CSDAUTO_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 CSDAUTO ウィザードによって順番に割り当てられるものです。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 CSDAUTO_ScanSensor(BYTE bSensor);
```

アセンブリ：

```
mov A,  bSensor
lcall CSDAUTO_ScanSensor
```

パラメータ：

A => センサ番号

戻り値：

なし

特殊作用：

**

CSDAUTO_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. CSDAUTO_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 CSDAUTO_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、CSDAUTO_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差のカウントが BaselineUpdateTh-reshold に達すると、基準値は 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 NoiseTh-reshold 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 CSDAUTO_UpdateSensorBaseline(BYTE bSensor)
```

アセンブリ：

```
mov   A,   bSensor
lcall  CSDAUTO_UpdateSensorBaseline
```

パラメータ：

A => センサ番号

戻り値：

なし

特殊作用：

**

CSDAUTO_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、CSDAUTO_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  CSDAUTO_bIsSensorActive(BYTE bSensor)
```

アセンブリ：

```
mov  A,  bSensor
lcall CSDAUTO_bIsSensorActive
```

パラメータ：

bSensor A => センサ番号

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 – 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。

特殊作用：

**

CSDAUTO_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 CSDAUTO_bIsSensorActive() を呼び出し、CSDAUTO_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  CSDAUTO_bIsAnySensorActive()
```

アセンブリ：

```
lcall CSDAUTO_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。

特殊作用：

**

CSDAUTO_wGetCentroidPos

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、オフセットと長さが一時変数に保存され、重心位置が CSDAUTO ウィザードで指定された分解能に計算されます。この関数は、スライダが CSDAUTO 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  CSDAUTO_wGetCentroidPos(BYTE bSnsGroup)
```

アセンブリ：

```
mov  A,  bSnsGroup
lcall CSDAUTO_wGetCentroidPos
```

パラメータ：

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。

特殊作用：

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。

スライダセンサが一つでも動作中の場合、関数はゼロから CSDAUTO ウィザードで設定された分解能値を返します。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、CSDAUTO_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。

注意事項：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擬似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)。

CSDAUTO_wGetRadialPos

説明

Difference アレイを確認し、重心を探します。重心が存在する場合、重心位置を CSDAUTO 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、CSDAUTO 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  CSDAUTO_wGetRadialPos(BYTE bSnsGroup)
```

アセンブリ：

```
mov  A,  bSnsGroup
lcall CSDAUTO_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 CSDAUTO UM ウィザードを通して得ることができます (例えば 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。

特殊作用：

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)。

CSDAUTO_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 CSDAUTO_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  CSDAUTO_wGetRadialInc(BYTE bSnsGroup)
```

アセンブリ：

```
mov  A,  bSnsGroup
lcall CSDAUTO_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 CSDAUTO UM ウィザードを通して得ることができます（例えば 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 CSDAUTO_wGetRadialPos() が 1 (FFFFh) を返した)、または今回タッチがない場合 (今回 CSDAUTO_wGetRadialPos() が -1 (FFFFh) を返した)。

特殊作用：

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

CSDAUTO_InitializeSensorBaseline

説明

選択されたセンサをスキャンして、初期値を伴う CSDAUTO_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、個々のセンサのベースラインをリセットするために使用されます。

C プロトタイプ：

```
void CSDAUTO_InitializeSensorBaseline(BYTE bSensor)
```

アセンブリ：

```
mov A, bSensor
lcall CSDAUTO_InitializeSensorBaseline
```

パラメータ：

A => センサ番号

戻り値：

なし

特殊作用：

**

CSDAUTO_InitializeBaselines

説明

各センサをスキャンして、初期値を CSDAUTO_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 CSDAUTO_InitializeBaselines()
```

アセンブリ：

```
lcall CSDAUTO_InitializeBaselines
```

パラメータ：

なし

戻り値：

なし

特殊作用：

**

CSDAUTO_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。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 分解能でスキャンできます。近接検出器では、長い範囲の検知を実行するために、16-bit の分解能と長いスキャン時間でスキャンの頻度を下げます。この関数は CSDAUTO_ScanSensor() 関数とともに使用できます。

C プロトタイプ：

```
void CSDAUTO_SetScanMode(BYTE bSpeed, BYTE bResolution);
```

アセンブリ：

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

パラメータ：

bSpeed, bResolution

戻り値：

なし

特殊作用：

**

CSDAUTO_SetIdacValue

説明

この関数は iDAC 電流値を設定します。異なる iDAC 設定でスキャンする必要のあるセンサがある場合に、これを使用します。この関数は CSDAUTO_ScanSensor(). とともに使用できます。

C プロトタイプ：

```
void CSDAUTO_SetIdacValue(BYTE bRefValue);
```

アセンブリ：

```
mov     A, bIdacValue
lcall    CSDAUTO_SetIdacValue
```

パラメータ：

bIdacValue - iDAC 値をセットします。許可されている値は 1.. 255 です。

戻り値：

なし

特殊作用：

**

CSDAUTO_SetPrescaler

説明

この関数はプリスケーラ値を設定します。プリスケーラ設定でスキャンする必要のあるセンサがある場合に、これを使用します。この関数は CSDAUTO_ScanSensor(). とともに使用できます。

C プロトタイプ：

```
void CSDAUTO_SetPrescaler(BYTE bPrescaler);
```

アセンブリ：

```
mov     A, bPrescaler
lcall    CSDAUTO_SetPrescaler
```

パラメータ：

bPrescaler - プリスケーラ値を設定します。次の表に、許可されている値が一覧表示されています。

名前	値	プリスケーラ
CSDAUTO_PRESCALER_1	0x00	1
CSDAUTO_PRESCALER_2	0x01	2
CSDAUTO_PRESCALER_4	0x02	4
CSDAUTO_PRESCALER_8	0x03	8
CSDAUTO_PRESCALER_16	0x04	16

名前	値	プリスケーラ
CSDAUTO_PRESCALER_32	0x05	32
CSDAUTO_PRESCALER_64	0x06	64
CSDAUTO_PRESCALER_128	0x07	128
CSDAUTO_PRESCALER_256	0x08	256

戻り値：

なし

特殊作用：

**

CSDAUTO_ClearSensors

説明

各センサに対して CSDAUTO_wGetPortPin() と CSDAUTO_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  CSDAUTO_ClearSensors()
```

アセンブリ：

```
lcall  CSDAUTO_ClearSensors
```

パラメータ：

なし

戻り値：

なし

特殊作用：

**

CSDAUTO_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  CSDAUTO_wReadSensor(BYTE bSensor)
```

アセンブリ：

```
mov A, bSensor
lcall CSDAUTO_wReadSensor
```

パラメータ：

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。

特殊作用：

**

CSDAUTO_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、CSDAUTO_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、CSDAUTO_EnableSensor()、CSDAUTO_DisableSensor(). へと渡すことができます。

C プロトタイプ：

```
WORD  CSDAUTO_wGetPortPin(BYTE bSensorNum)
```

アセンブリ：

```
mov  A,  bSensorNumber
lcall     CSDAUTO_wGetPortPin
```

パラメータ：

bSensorNumber – 範囲は 0 ~ (n – 1) です。ここで n は、CSDAUTO 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、CSDAUTO_wGetPortPin() によって使用されます。

戻り値：

A => センサのビットマップ

X => ポート番号

特殊作用：

**

CSDAUTO_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、CSDAUTO_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  CSDAUTO_EnableSensor(BYTE bMask, BYTE bPort)
```

アセンブリ :

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

パラメータ :

A => センサのビットマップ

X => ポート番号

戻り値 :

なし

特殊作用 :

**

CSDAUTO_DisableSensor

説明

CSDAUTO_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、CSDAUTO_wGetPortPin() 関数により返されます。

C プロトタイプ :

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

アセンブリ :

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

パラメータ :

A => センサのビットマップ

X => ポート番号

戻り値 :

なし

特殊作用 :

**

ファームウェア ソースコードの例

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 CSDAUTO 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)
{
  BYTE bIndex;

  M8C_EnableGInt;
  CSDAUTO_Start();
  CSDAUTO_InitializeBaselines() ; //Scan all sensors first time, init baseline

  while (1) {     //Loop forever
    for(bIndex=0; bIndex < CSDAUTO_TotalSensorCount; bIndex++) //Loop through all
sensors
  {
      CSDAUTO_ScanSensor(bIndex);      // Scan Sensors
      CSDAUTO_UpdateSensorBaseline(bIndex);  // Run baseline filter
  }

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

    // now we are ready to send all status variables to chart program
    // communication here
    //
    // OUTPUT CSDAUTO_waSnsResult[x] <- Raw Counts
    // OUTPUT CSDAUTO_waSnsDiff[x] <- Difference
    // OUTPUT CSDAUTO_waSnsBaseline[x] <- Baseline
    // OUTPUT CSDAUTO_baSnsOnMask[x] <- Sensor On/Off
  }
}
```

コンフィグレーション レジスタ

Table 4. ブロック CapSense、レジスタ : CS_CR0

ビット	7	6	5	4	3	2	1	0
値	0	0	CSDAUTO_PRSCLK	0	1	0	0	EN

Table 5. ブロック CapSense、レジスタ : CS_CR1

ビット	7	6	5	4	3	2	1	0
値	1	スキャン速度		0	0	0	0	0

Power：0x01 アナログブロックの電源をオンにします。0x00 アナログブロックの電源をオフにします。

Table 6. ブロック CapSense、レジスタ : CS_CR2

ビット	7	6	5	4	3	2	1	0
値	1	0	0	0	0	1	0	0

Table 7. ブロック CapSense、レジスタ : CS_CR3

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

Table 8. ブロック CapSense、レジスタ : CS_CNTH

ビット	7	6	5	4	3	2	1	0
データ出力 MSB								

Table 9. ブロック CapSense、レジスタ : CS_CNTL

ビット	7	6	5	4	3	2	1	0
データ出力 LSB								

Table 10. ブロック CapSense、レジスタ : PRS_CR

モード / ビット	7	6	5	4	3	2	1	0
値	1	0	8/12 ビット	1	プリスケーラ			

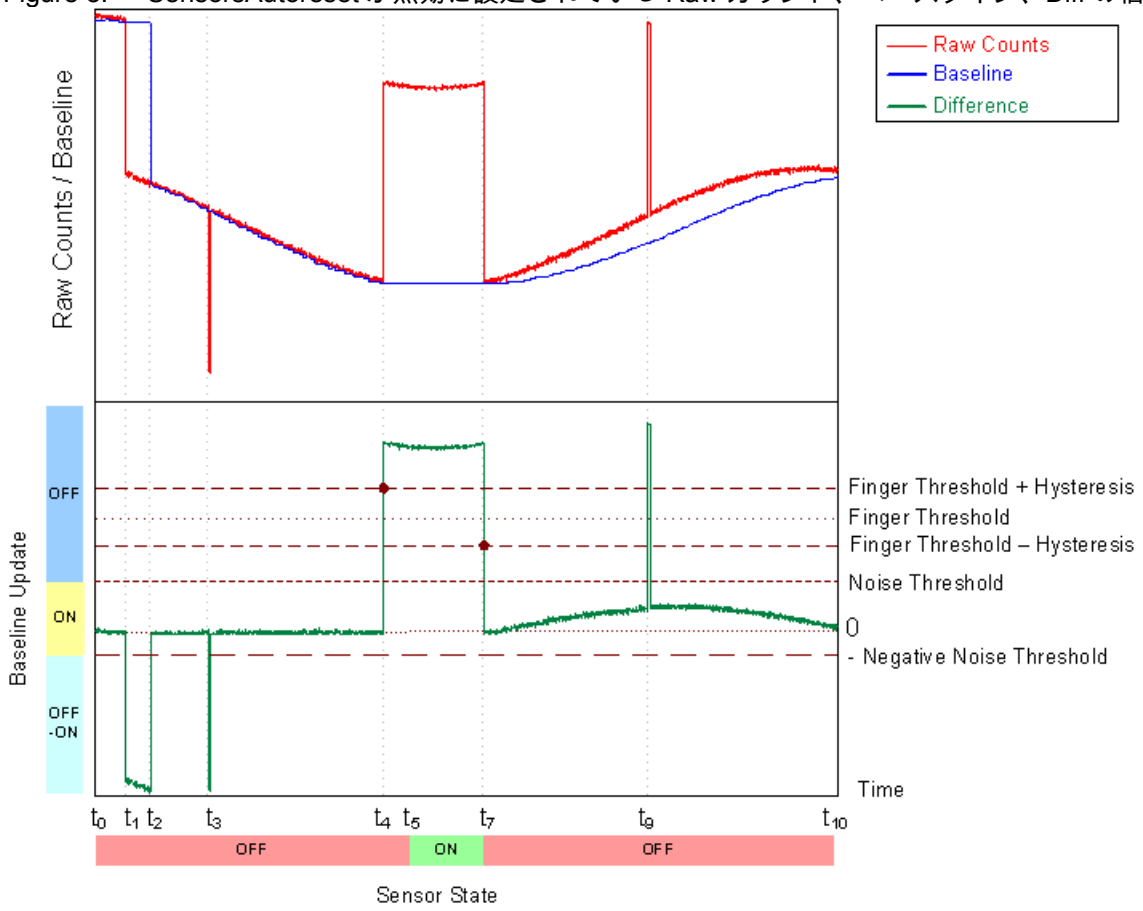
付録

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。

CSDAUTO 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 8. SensorsAutoreset が無効に設定されている Raw カウント、ベースライン、Diff 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。

[+] Feedback

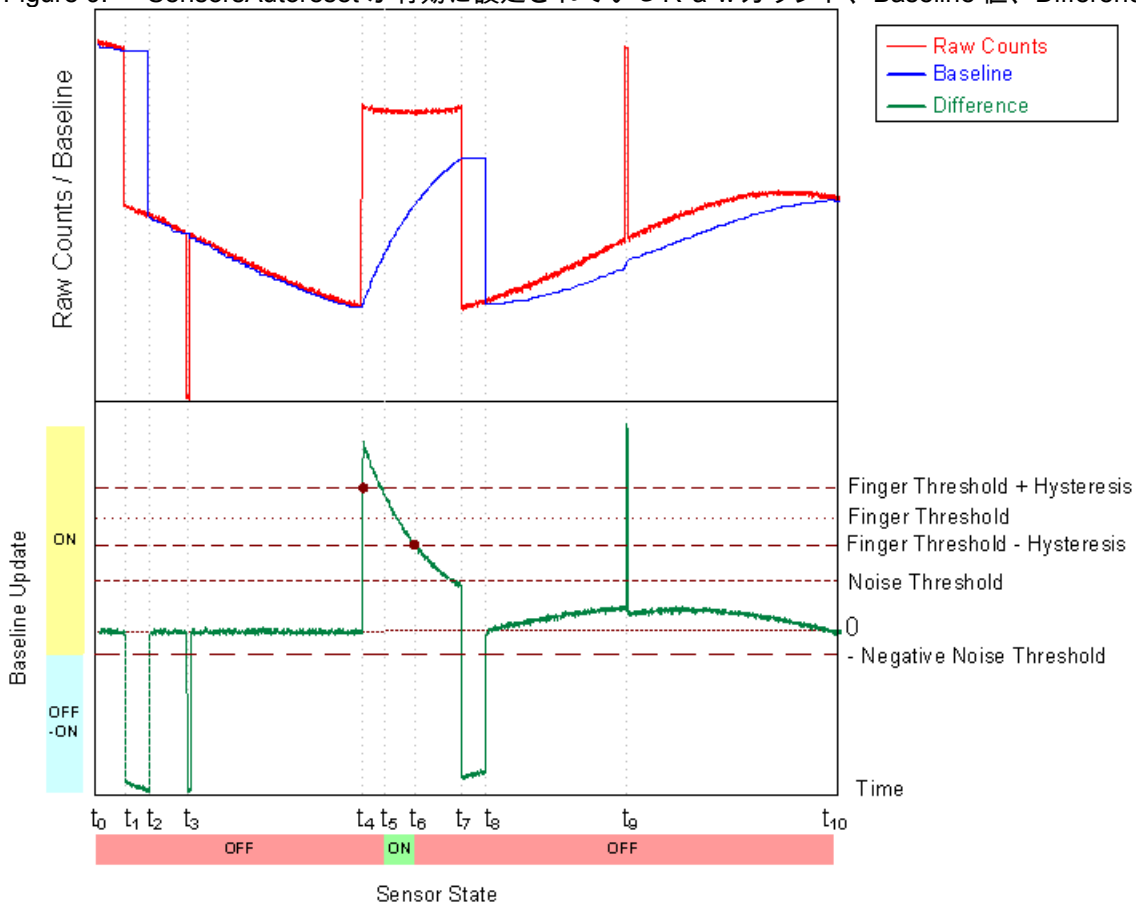
t_1 でＲａｗカウントは大幅に落下し、マイナスの差が NegativeNoiseThreshold を超えます。この状況は、指がセンサに接触しているときにデバイスの電源がオンになり、ある程度時間が経ってから指を離した場合に発生します。このとき、基準値更新のメカニズムが停止し、内部のタイムアウトカウンタが起動します。Ｂａｓｅｌｉｎｅ値は、LowBaselineReset のサンプル数の間、Difference 信号が NegativeNoiseThreshold を下回ったときにリセットされます。この状況は t_2 で起きます。

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 9.　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



[+] Feedback

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、もう一つのデバウンスメカニズムとして機能します。

バージョン ヒストリー

バージョン	著者	説明
1.1	DHA	リニア スライダがない場合のラジアルスライダの位置を更新。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。 容量を追加のスライダに追加。 ユーザ モジュールパラメータからウィザード設定セクションへの Moved ModCap 選択を移動。 新パラメータ 「センサ感度」を追加。 TMA3xx シリーズのチップをサポートするために、CSD と CSDAUTO を追加。

Note PSoC Designer 5.1 は、すべてのユーザ モジュール データシートにおいてバージョン ヒストリーを導入しています。このセクションでは、ユーザ モジュールの過去のバージョンと現在のバージョンとの違いに関して高度な解説を掲載しています。

Copyright © 2009-2011 © 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 (サイプレス セミコンダクタ社) 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。特許またはその他の権限下で、ライセンスを譲渡また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、または安全の用途のために使用することを保証するものではなく、また使用することを意図したものでもありません。さらにサイプレスは、誤動作や故障によって使用者に重大な傷害をもたらすこと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ト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® は Cypress Semiconductor Corp の登録商標であり、PSoC Creator ™ および Programmable System-on-Chip ™ は Cypress Semiconductor Corp. の商標です。本文書で言及するその他すべての商標または登録商標は、各社の所有物です。

全てのソース コード (ソフトウェアおよび / またはファームウェア) はサイプレス セミコンダクタ社 (以下「サイプレス」) が所有し、全世界の特許権保護 (米国およびその他の国)、米国の著作権法なら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り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 ソフトウェアおよび / また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コピー、使用、変更そして作成するためのライセンス、ならびにサイプレスのソース コードおよ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、また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、明示的または黙示的を問わず、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、商品性または特定目的への適合性の黙示的な保証か゛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または回路を適用また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ト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。