CapSense® シグマデルタプラス ADC データシート CSDADC V 1.40

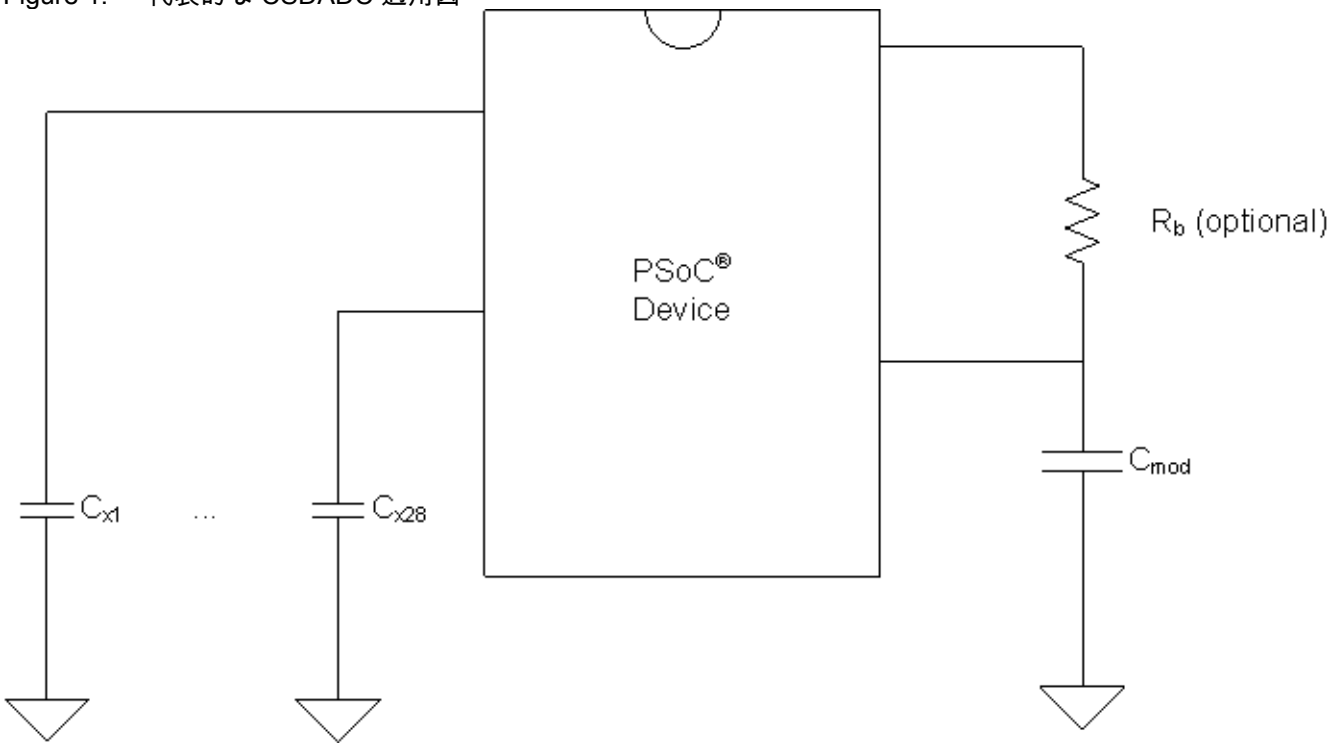
Copyright © 2011-2012 Cypress Semiconductor Corporation. All Rights Reserved.

リソース	PSoC® ブロック			API メモリ (バイト) Typical		ピン (外部入出力ごと)
	デジタル	アナログ CT	アナログ SC	フラッシュ	RAM	
CY8C21x34、CY8CLED04。Flash、RAM、ピンの使用数は、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 センサの場合)	3	2	1	1491	36	2 ー 5
RRS8 ベースのユーザ モジュール (1 センサの場合)	2	2	1	1401	36	2 ー 5
プリスケーラベースのユーザ モジュール (1 センサの場合)	2	2	1	1426	36	2 ー 5
VC2 クロックを使ったユーザモジュール (1 センサの場合)	1	2	1	1351	36	2 ー 5
各追加 CapSense® ボタン	-	-	-	10	2	1
5 つのセンサー素子を使った静電容量式スライダを使用する場合の、コード容量と RAM 使用エリアの増分	-	-	-	79	583	5
追加の各スライダ素子	-	-	-	10	2	1
スライダでダイプレックスを使用する場合の、コード容量と RAM 使用エリアの増分	-	-	-	スライダ *2	0	-
ダイ温度測定	-	-	-	380	2	-

特長および概要

- CapSense センサのスキャンと電圧の測定を、ハードウエアの、機能、設定を個別にロード (して組み立て) しなくとも、同時に使用可能。
- ADC の特長 :
  - ADC の入力モードを、固定電圧、レシオメトリックと動的に変更可能
  - 絶対電圧入力モード用のシングル スロープ ADC
  - シングル スロープ ADC 用の校正メカニズム内蔵
  - レシオメトリック入力モードに対応したインクリメンタル型 ADC
  - 粗い温度測定が可能 : 範囲 -40 ~ +125°C、精度 ± 40°C （ 分解能 ± 2°C の場合 ）
- CapSense® の特長 :
  - 実績のある CSD 方式を採用
  - 1 ~ 28 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 電源ノイズ、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てスライドセンサの物理的分解能を倍加
  - 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
  - 高抵抗伝導性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するシールド電極サポート
  - CSDADC ウィザードを使用した誘導センサとピンの割り当て
  - 温度、湿度、静電気放電 (ESD) に対応する、実績のある基底値更新アルゴリズム
  - 調整が簡単な操作パラメータ
  - Raw データ値のモニタリングとリアルタイムのパラメータ最適化のための PC GUI アプリケーションサポート

Figure 1.　代表的な CSDADC 適用図



クイック スタート

1. 専用ピンを必要とするユーザモジュールを選択・配置します（例：I2C と LCD）。ポートとピンを適宜割り当てます。
2. CSDADC ユーザ モジュールを選択、配置します。
3. CSDADC ユーザ モジュールを右クリックし、CSDADC ウィザードを開きます。
4. CapSense センサ数、コンフィグレーション、ピン割り当てを設定します。
5. ピンとグローバルパラメータを設定します。パラメータの説明を全部読み、要件とガイドラインに従います。
6. アプリケーションを生成し、Application Editor を開きます。
7. 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。
8. RS232 レベル トランスレータまたは USB-I^{2}C ブリッジをターゲット ボードに接続し、GUI を使ってパラメータを最適化します。
9. CSDADC パラメータを変更し、アプリケーションを再構築します。
10. PSoC デバイスをプログラムし、モジュールの動作を検証します。CY8C21x34/B で説明されているように、5：1 SNR の要件を達成するために CSDADC パラメータを調整してください。

機能説明

CSDADC は、機能、設定を個別にロード (して組み立て) しなくとも、静電容量式検知と ADC を使った電圧測定を行います。CSD と ADC に共通のモジュールを再利用することで、コード容量を節約します。静電容量式検知と ADC 機能の両方を必要する場合は、CSDADC を使用します。このうち片方が必要なアプリケーションの場合は、CSD ユーザ モジュールまたは ADC8 または ADC10 を使用します。

CSDADC は、スイッチト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 (CSD) を使用した手法による静電容量の検知を行います。これはレシオメトリック入力 ADC モードと絶対入力 ADC モードの両方を使用して、ランタイムにモードを切り替えられるため、異なるセンサタイプとのインターフェースが簡単にできます。例えば、サーミスタを用いて温度を測定する場合にはレシオメトリックモードを使用し、電池電圧を測定する場合には絶対電圧測定を使用することができます。

CSDADC ユーザ モジュールは、絶対入力電圧モードではシングル スロープ ADC を使用し、レシオメトリックモードではインクリメンタル ADC を使用して実装します。

このユーザーモジュールデータシートには、CSD と ADC の基本情報が記載されています。設定パラメータ、使用上のヒント、トラブルシューティングに関する理論と推奨事項の詳細については、CSD と ADC10 データシート、および関連アプリケーションノートをご参照ください。www.cypress.com から探すことができます。静電容量式検知アプリケーションの実装や ADC を今回初めて使用するユーザは、作業開始前にマニュアルをお読みください。

CSDADC ユーザーモジュールを初めてご利用になる場合は、以下の資料をお読みください。

- *CY8C21x34 PSoC ミックスシグナル　技術レファレンス マニュアル*。以下のセクションをお読みください。
  - 2 カラムのアナログ
  - デジタル クロック
  - I/O アナログ マルチプレクサ
- CSD ユーザ モジュール データシート
- ADC10 ユーザ モジュール データシート

CSD ユーザ モジュール データシートを読んだあとは、次のデザインガイドを推奨します。これらの資料は Cypress Semiconductor のウェブサイト www.cypress.com で入手可能です。:

- CapSense の導入
- CY8C20xx6A/H CapSense 設計ガイド
- CY8C21x34/B CapSense 設計ガイド
- CY8C20x34 CapSense 設計ガイド
- CY8CMBR2044 CapSense 設計ガイド

ADC 操作の説明

CSDADC は次の 2 つの ADC タイプをサポートします。

- シングル スロープ。シングル スロープ ADC は、絶対入力電圧モードをサポートします。このモードでは、ADC 読み値が入力信号と正比例し、PSoC 電源電圧に依存しません。
- インクリメンタル。インクリメンタル ADC は、レシオメトリック動作モードをサポートします。このモードでは、ADC 読み値は入力電圧と正比例し、PSoC 電源電圧と反比例します。

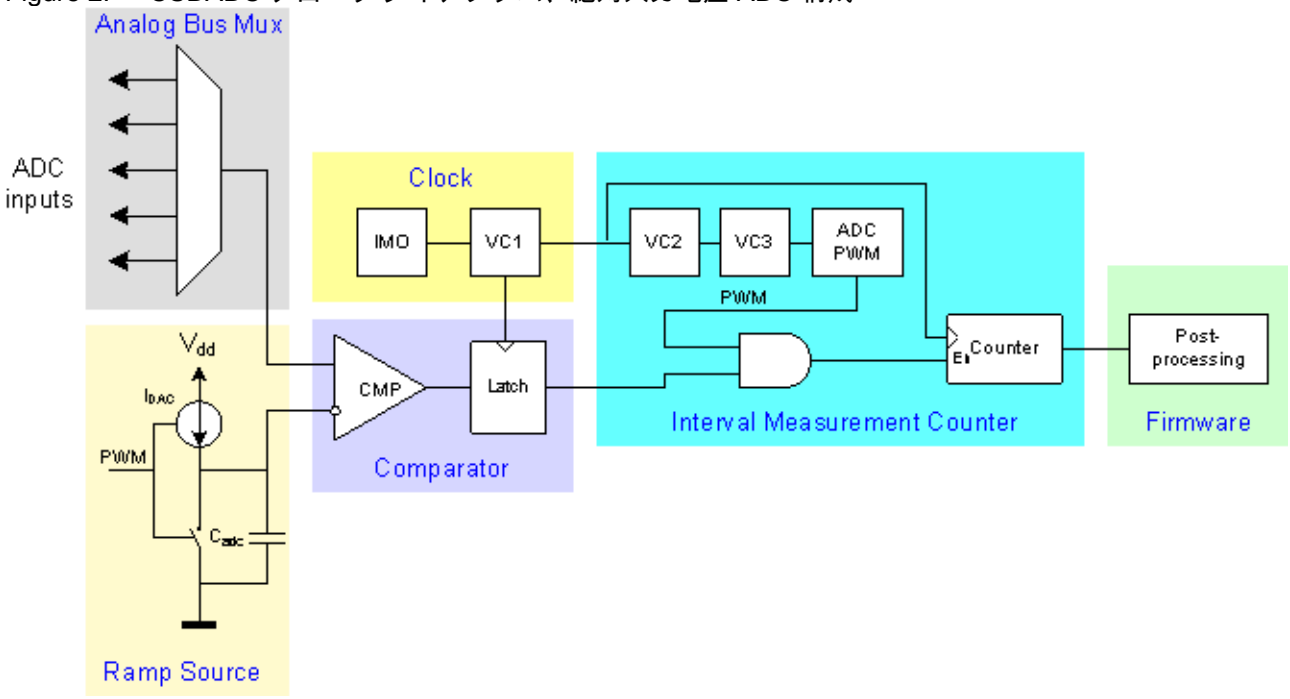
ADC サンプリングメカニズムはそれぞれで異なります。シングル スロープ ADC は、サンプリングされた瞬間の信号値を変換し、レシオメトリック ADC は、その積分の性質により変換期間全体にわたって

信号を変換します。このため、入力信号にノイズがある場合は、インクリメンタル ADC の使用をお勧めします。

シングル スロープ ADC の動作

シングル スロープ ADC の構成は、ADC10 ユーザ モジュールと同じです。

Figure 2. CSDADC ブロック ダイアグラム、絶対入力電圧 ADC 構成



シングル スロープ A/D コンバータは、最高 12-bit、すなわちフルスケール出力（0 ~ 4095 カウント）まで生成します。サンプリングレートは、ユーザのクロック源と選択されたパラメータによって決定します。

ADC は次の PSoC リソースによって構成されます。

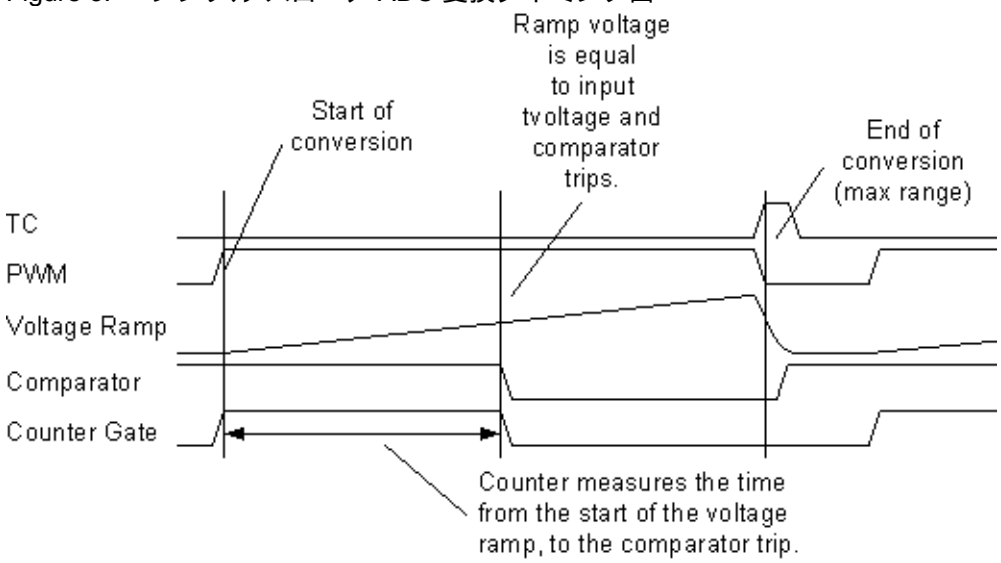
- アナログ CT ブロック。コンパレータとして構成されます。
- アナログ SC ブロック。内部コンデンサと電流源を用いてランプ発生器として構成されます。
- デジタルブロック。8 ビットカウンタとして構成されます。
- ADC のタイミング専用 PWM。

シングル スロープ ADC 実装のブロック ダイアグラムが図 2 に示されています。変換アルゴリズムの主要コンポーネントは、電流源、積分コンデンサ、コンパレータです。電流源がアクティブになると、コンデンサに電荷がたまり、電圧がリニアに上昇していきます。コンデンサの電圧は、アナログコンパレータ回路への入力の一つで、もう一つは変換されるアナログ入力電圧です。コンパレータは、ランプ電圧が入力電圧を超えるまで Low で、その後推移します。カウンタは、ランプ開始時とコンパレータ始動時の間の時間をトラッキングします。

基本変換波形が図 3 に示されています。12-bit ビットサンプルの場合、カウンタが 4096 に到達するように、PWM の HIGH 時間がセットされます。PWM の LOW 時間は、コンデンサが放電し、回答が処理されるように設計されています。PWM の最終カウントは、変換の結果を読むための割り込みとして使用

されます。ランプ電圧が一度も入力値を越えない場合、ADC_CR のビット 7 は HIGH にセットされます。PWM 発生器は、VC3 のみクロック源として使用し、HIGH と LOW の選択が限られています。

Figure 3.　シングル スロープ ADC 変換タイミング図



ADC 入力は、コンパレータ入力のサンプル ホールドです。サンプルおよびホールドの回路構成は、PWM によって制御されます。これによって、変換中の信号の安定が保証されます。

シングル スロープ ADC の校正

ADC は使用前に校正する必要があります。それぞれのアナログコラムには、その目的のための ADC コンデンサ トリム レジスタがあります。このレジスタは、ADC 電圧ランプのスロープを決定するコンデンサ値を制御します。ADC_TR レジスタの CAPVAL [7:0] ビットは、校正用に使用されます。この校正プロセスの目的は、フルスケール ADC 入力値の結果がフルスケール ADC コードとなるように、ランプ時間（スロープ）をチューニングすることです。これは、ランプ時間を任意のフルスケール変換期間に適合させることで、達成されます。PSoC デバイスが安定していない電源によって作動している場合、DAC 電流は電源電圧に依存しているため、キャリブレーションを定期的に行う必要があります。

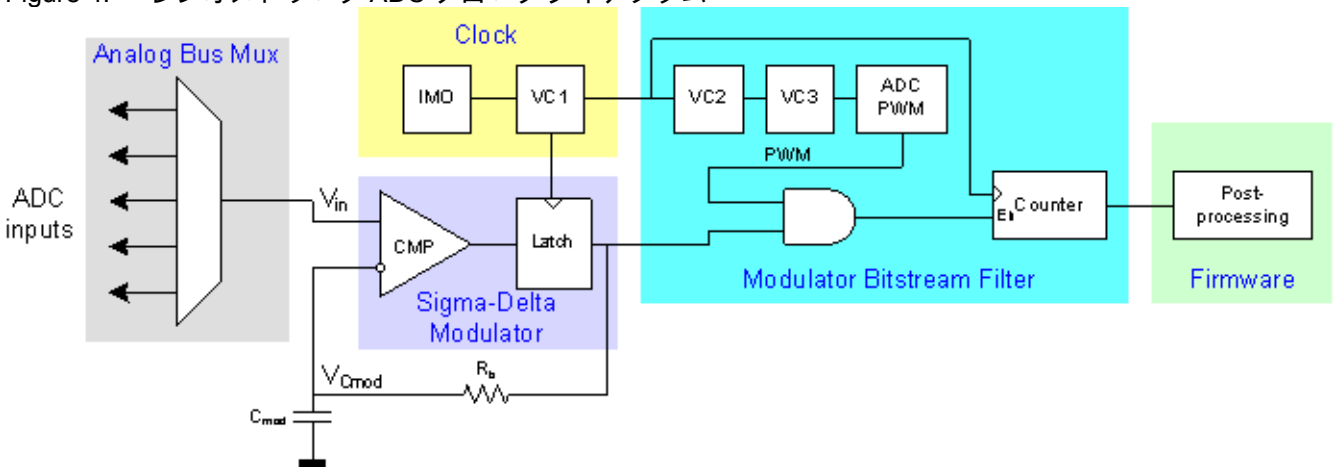
ユーザ モジュール API には、電圧のキャリブレーションとカウント値のキャリブレーションに基づいて ADC_TR レジスタをトリミングする校正アルゴリズムを実行する CSDADC_iCal というルーチンがあります。

CSDADC_iCal ルーチンの戻り値は、ADC の校正信号による実行結果です。理想的には、この値は、期待された校正数値であるべきです。それ以外の場合は、精度を上げるために、ソフトウェアメソッドを用いて精度を保証することもできます。

インクリメンタル ADC

インクリメンタル ADC 動作は、シグマデルタ一次変調器と、カウンタベースの変調器ビットストリーム デジタルフィルタの組み合わせが基になります。この ADC タイプのブロック ダイアグラムを図 4 に示します。

Figure 4. レシオメトリック ADC ブロック ダイアグラム



この ADC は次のコンポーネントによって構成されています。

- シグマデルタ変調器
- 変調器ビットストリーム フィルタ
- クロック源

この変調器はコンパレータ、コンパレータラッチ、変調コンデンサ 、C_{mod}、フィードバック　抵抗、R_b で構成されています。変調コンデンサの電圧が入力電圧 V_{in} を下回っているときは、コンパレータ出力が HIGH V_{dd} レベルで、変調コンデンサが充電されます。変調コンデンサ電圧が入力電圧を越えた場合、コンパレータ出力は低 Vss レベルに切り替わります。これにより、変調コンデンサ電圧が低下し始めます。変調器の電圧が再び入力電圧を超えると、変調コンデンサの電圧が再び上がり始め、変調コンデンサの充電・放電周期が繰り返されます。ラッチは、コンパレータ動作とクロックの VC1 信号に同期し、コンデンサの最低充電・放電間隔を制限します。シグマデルタ変調器は、変調コンデンサ C_{mod} で充電・放電を交互に繰り返し、電圧 V_{Cmod} を平均化することで電圧 V_{in} の近くに保ちます。

変調器の計算は簡単です。変調器のビットストリームのデューティ比を d_{mod} とした場合、変調器のリファレンス電圧は次のように表されます。

Equation 1

$$V_{Cmod} = V_{dd} \cdot d_{mod}$$

平均で $V_{Cmod} = V_{in}$ であることを考慮すると、変調器のデューティ比は次の式で計算できます。

Equation 2

$$d_{mod} = \frac{V_{in}}{V_{dd}}$$

ADC 変調器ビットストリームフィルタとクロック源

変調器は、入力電圧を出力ビットストリームのデューティ比に変換します。デューティ比はデジタルフィルタを用いて測定されます。CY8C21x34 電圧には専用のデシメータがないため、CY8C21x34 デバイスでの実装にはカウンタベースのフィルタを使用します。フィルタは、8-bit ハードウェアカウンタと測定間隔構成回路で構成されます。カウンタは、ハードウェアリソースを節約するために、ファームウェアで最終カウントのオーバーフローを処理することにより、16 ビットに拡張されます。カウンタは、変調器とともに標準 VC1 クロック信号を使用し、変調器出力が LOW のときはカウンタは状態を保存し、変調器出力が HIGH のときは 1 つデクリメントします。

測定間隔構成回路は、カウンタを読むために割り込みを定期的に使用し、カウンタイネーブル入力をゲートし、カウンタ値が読み取り中に変化することを防ぎます。内部メイン発振器のデューティ比測定間隔 N_m は、次の式の VC1、VC2、VC3、ADCPWM の値によって決定します。

Equation 3

$$N_m = VC1 \cdot VC2 \cdot VC3 \cdot N_{ADCPWM}$$

VC1、VC2、VC3、N_{ADCPWM} – 分周器の値。

デジタルフィルタは、各サンプルで N_m IMO サイクルを構成し、フィルタサンプル値は合計検知キャパシタンス C_s と直線的に比例します。最大サンプル値は N_{max} によって決定します。

Equation 4

$$N_{max} = VC2 \cdot VC3 \cdot N_{ADCPWM}$$

Equation 5

$$N_s = d_{mod} N_{max}$$

Equation 6

$$N = \frac{V_{in}}{V_{dd}} \cdot VC2 \cdot VC3 \cdot N_{ADCPWM}$$

ユーザ モジュールは、8-bit ハードウェアカウンタの最大カウントを拡張する 1 カウンタのオーバーフロー割り込みと、ADCPWM 割り込みという、2 つの割り込みを使用します。ADCPWM 割り込みは、コンパレータ xA0;0 バス割り込みにルーティングされます。オンにするコンパレータは 1 つのみですが、2 つのコンパレータバスを使用します。

ダイ温度測定

ADC は、オンチップダイ温度回路出力信号の測定を行います。信号は、連続時間ブロック ACE01 の PMux アナログ出力として有効です。この信号は名目上は約 1.0V、範囲は約 0.7 ~ 1.3V です。出力電圧はダイ温度の関数です。

静電容量測定の操作

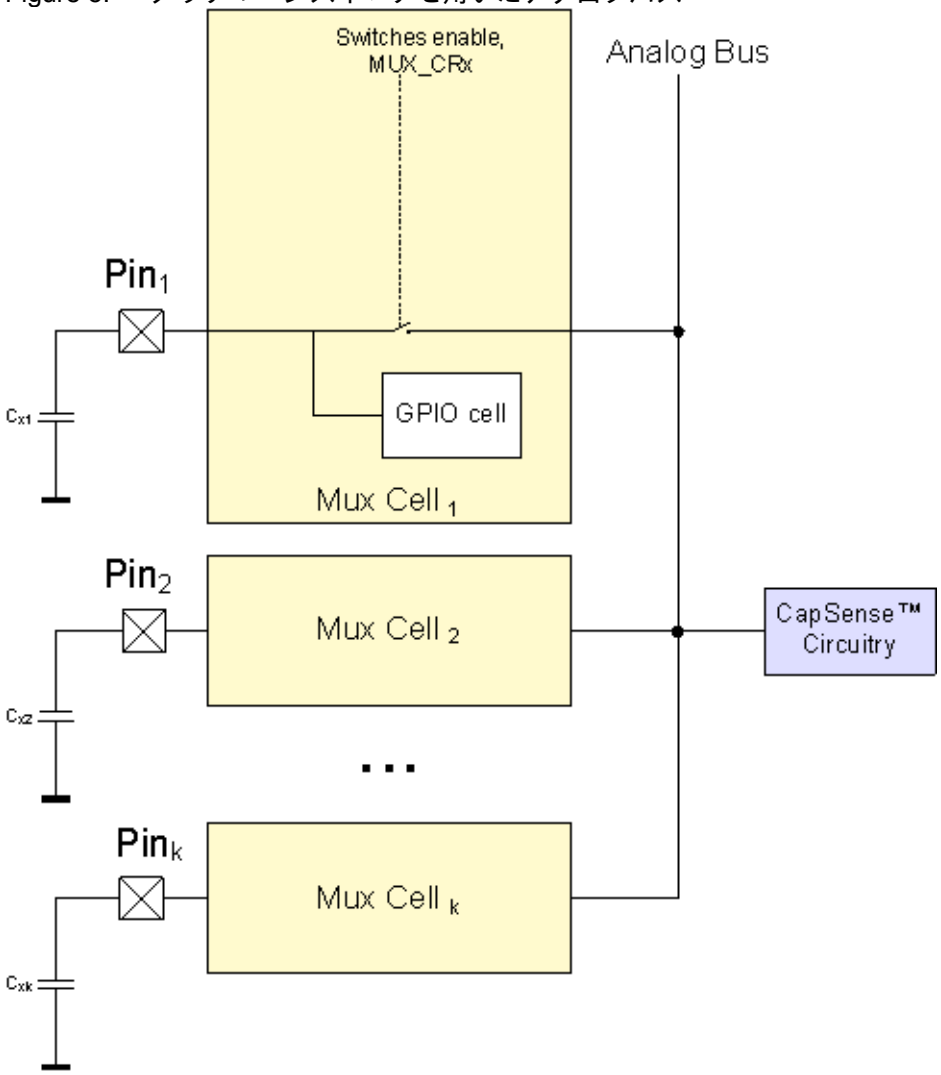
判定理論はファームウェアで実行されます。ファームウェアは、静電容量を分析し、環境要因によるゆっくりとした静電容量の変化をトラッキングし、判定理論を実行することにより、ボタンのタッチを検知し、スライダの位置を計算します。

センサのアレイをスキャンする

CY8C21x34 デバイスファミリーはアナログバスの中に構成され、任意の PSoC ピンへの静電容量式センサの接続を可能にします。CSDADC ユーザ モジュールは、内部プリチャージスイッチを使用して、クロック信号フェーズ Ph_1 で動作中のセンサを充電し、アナログバスをフェーズ Ph_2 でセンサと接続します。シグマデルタ変調器の変調コンデンサとコンパレータの入力は、アナログバスに恒久的に接続されています。

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。

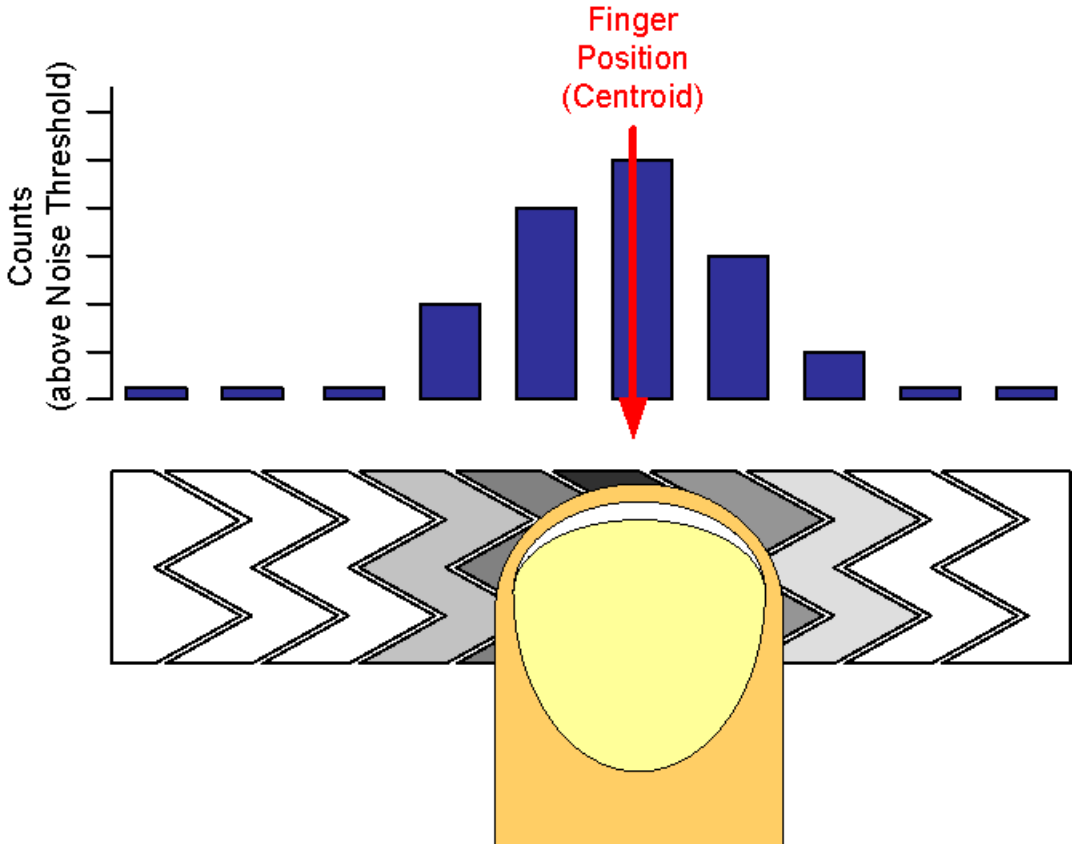
Figure 5.　プリチャージスイッチを用いたアナログバス



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。例としては、照明管理（調光装置）、音量管理、グラフィックイコライザ、速度管理などが挙げられます。これらのセンサは機械的に互いに隣接しており、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 セットのセントロイド位置を計算することによって判断できます。CSDADC 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 スライダの実践的な下限は 5 で、上限は、選択した PSoC デバイスで利用できるセンサ位置数になります。

Figure 6.　物理的センサ位置の順序



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。

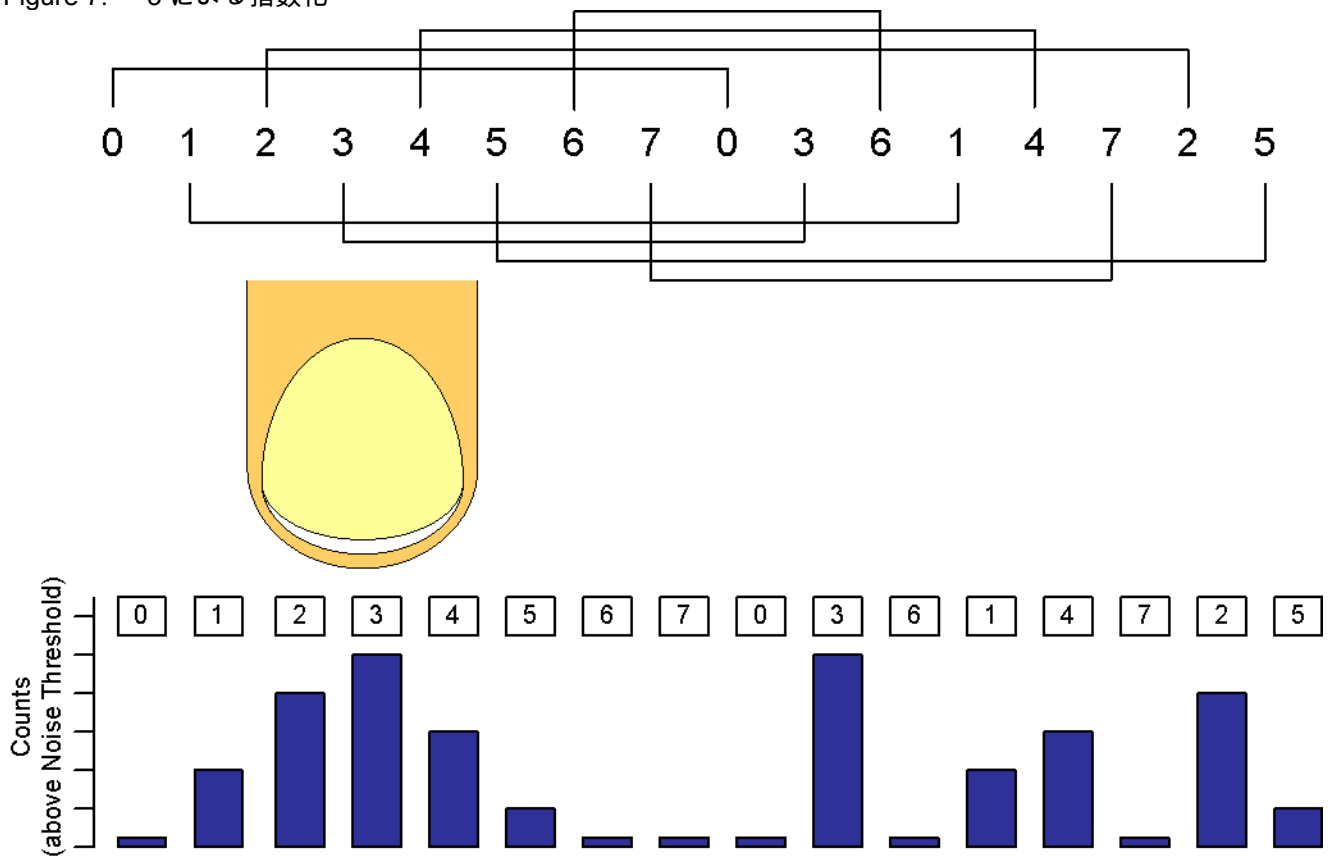
ダイプレックス

スライダセンサの各センサは、PSoC の各ピンに 2 つづつ接続されます。物理的位置の最初の（もしくは数字が小さい）半分は、CSDADC ウイ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。残りの半分（数字が大きい）のスライダセンサは、ウィザードのアルゴリズムによって自動的にマッピングされ、取り込みファイルに一覧表示されます。この順序は、半分内における近隣センサ起動が別の半分の近隣センサ起動を引き起こさないように設定されます。この順序の決定と、PCB 基板へのマッピングは慎重に行ってください。

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。最も簡単な方法は、上半分のセンサのうち、偶数センサ全部を先に決定し、次に奇数センサ全部を決定するやり方です。他の方法で

は、別の指数値を使ってセンサを配置します。。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 7.　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、一部のセンサペアではセンサ配線が長くなる可能性があります。ダイプレックス センサ指数表は、ダイプレックスを選択すると CSDADC ウィザード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。

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
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

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
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

ダイプレックススライダのスライダセグメント選択ガイドライン

スライダに必要なセグメント数の選択は、スライダの物理的長さに左右されます。ただし、ダイプレックススライダのセグメント数を決めるときは、特別に注意する必要があります。

ダイプレックススライダの設計では、スライダの長さを物理的に長くするために、1つのセンサが2つの異なる物理スライダーセグメントとして使用されます。フィンガータッチで完全にカバーされるセグメント数は、同一センサから由来する2つのセグメント間のセンサ数より少なくする必要があります。これにより、ダイプレックススライダが適切に作動することが保証されます。

たとえば、10-segment スライダが1つ（5センサ）の場合、センサ3の2つのセグメントは2個のセンサ（センサ4と0）のみで区切られます。この場合、フィンガータッチでは、スライダが正常に作動するようにするには、3つ以上のセンサセグメントでカバーしてはなりません。

12-segment スライダでは、1 フィンガータッチで 4 つ以上のセグメントをカバーしてはなりません。同様に、18-segment スライダでは、1 フィンガータッチで 5 つ以上のセグメントをカバーしてはなりません。

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リアは、しばしば 1 個のセンサより大きくなっています。

セントロイド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セントロイドを算出します。セントロイドは (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 7

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

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、セントロイド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セントロイドに掛けます。1 つの計算で補間とスケーリングを組み合わせ、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 カウントと分解能は、CSDADC 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。

セントロイドの分解能の乗数は、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。

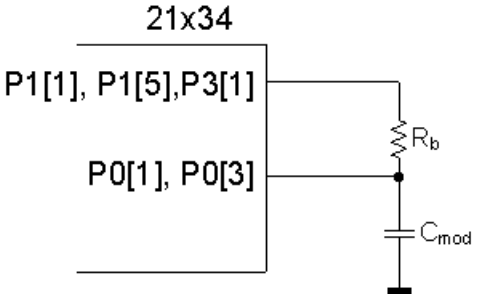
分解能 = (センサ数 – 1) x 乗数

セントロイドは 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　抵抗 R_b が必要です。コンデンサは、P0[1]、P0[3] ポート ピンと Vss グラウンドに接続できます。フィードバック　抵抗 R_b は、ポート ピン P1[1]、P1[5]、P3[1]、及びコンデンサ 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。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、CSDADC 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。

Figure 8.　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 nF です。最適な静電容量は、最大の S/N 比 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。PRS16 および PRS8 構成では殆どの場合、5.6 ~ 10 nF の値では良い結果が得られます。プリスケーラ付きの構成を選択した場合、統合コンデンサの推奨値は 22 ~ 47 nF です。フィードバック　抵抗を選択した後で、最良の S/N 比 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。抵抗値は、総センサ静電容量 C_s によって異なります。抵抗値は、次のステップで選択します。

- 異なるセンサのタッチの Raw カウントをモニターする。
- 選択されたスキャン分解能でフルスケール読み値より約 30% 小さい最大読み値を提供する抵抗値を選択する。抵抗値が減ると、Raw カウント値は増えます。

一般的な値は 500Ω ~ 10 kΩ の間で、センサキャパシタンスによって異なります。CY3213 評価ボードを使用している場合は、2kΩ 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。白物 (家電品)、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処理の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が有効です。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。CSDADC 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 9.　シールド電極 PCB レイアウト例

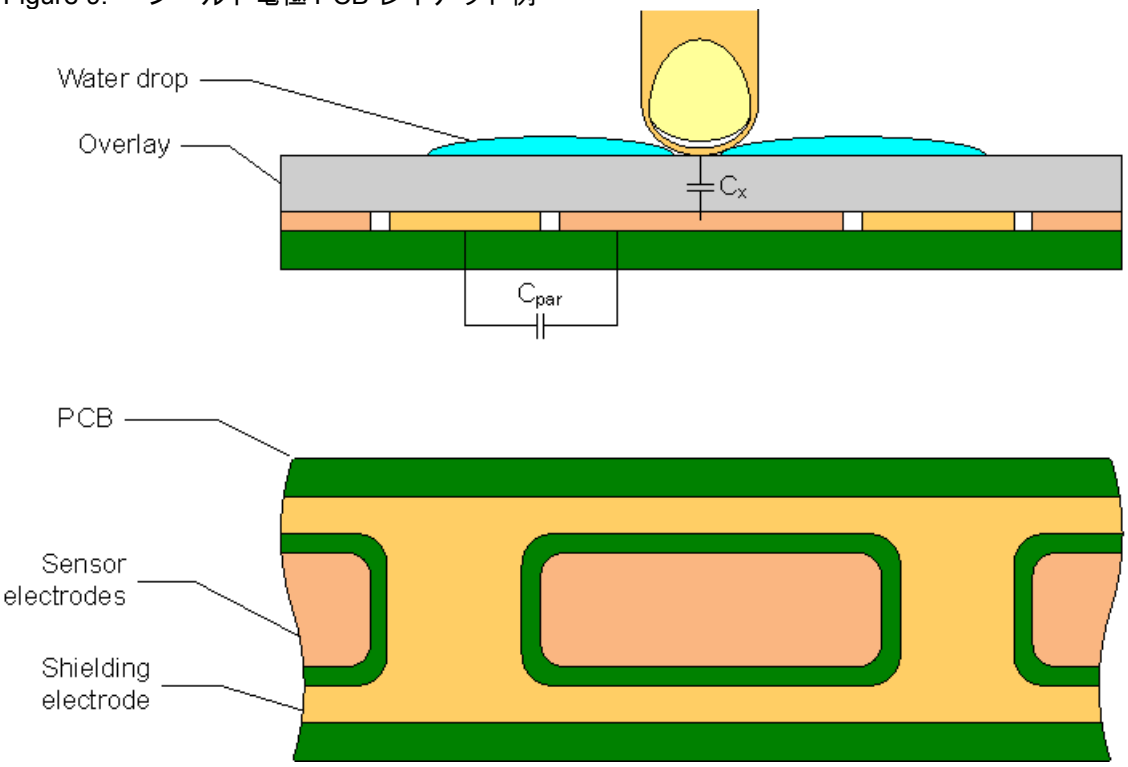


図 9 は、ボタンのシールド電極のレイアウト構成例を示しています。シールド電極は、LCD ドライブ電極のノイズの影響を阻止し、同時に浮遊静電容量を低減するため、透明な ITO タッチパッドデバイスでは特に有用です。

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。

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、グラウンドノイズと放射妨害波を軽減します。また、スルー制限　抵抗も、PSoC デバイスとシールド電極の間に接続できます。

クロック源

クロック源は、検知キャパシタ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 4 つの選択オプションをサポートしています。以下の表で、構成を比較します。

構成	動作周波数	使用するデジタル ブロック	EMC ノイズイミュニティ
PRS16	拡散スペクトル。平均は $F_{IMO}/4$、ピークは $F_{IMO}/2$。	3	高。高感度ポイントは、PRS シーケンスの繰り返し期間の倍数で、PRS 基本周波数は F_{IMO}。
PRS8	調整可能な拡散スペクトル、 $F_{IMO}/4$ ～ $F_{IMO}/512$	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検知。高い EMI 耐性が必要な場合は、PRS16 構成を使用してください。
Prescaler (プリスケーラ)	固定、IMO/(期間 + 1)	2	デバイスは、動作周波数とその調和周波数の EMC 信号に敏感です。高抵抗素材を使用した操作のみ推奨
VC_2	固定、IMO/(VC_1xVC_2)	1	デバイスは、動作周波数とその調和周波数の EMC 信号に敏感です。認定 EMC/EMI テストが予定されていない場合のみ推奨。

変調器のリファレンス源

コンパレータのリファレンスは、コンパレータの基準電圧を形成するために使用されます。基準電圧値は、検出感度を決定します。

下表は、リファレンス選択のオプションをまとめたものです。

外付けコンポーネント	UM の選択肢	使用時
なし	VBG	読み値は電源電圧に比例。電源がうまく調節されている場合にのみ使用します。
なし	ASE11	大部分のアプリケーションに推奨。このオプションから試してください。
2	AnalogColumn 入力選択	読み値はそれほど電源へ依存していません。推奨 R1 = 10k; R2 = 3.6k
2	AnalogColumn 入力選択	他のリファレンス選択でノイズが大きすぎる場合。

多くのユーザは、VBG と ASE11 リファレンス選択のみを主に使用する可能性が高いでしょう。この他の 2 つのオプションは、特殊アプリケーションの場合に役立ちます。

DC 電気的特性と AC 電気的特性

Table 2. 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 3. キャパシタンス検知ノイズ

パラメータ[a]	最小値	標準値	最大値	単位	テスト条件 (Vdd = 5V, SysClk =24 MHz, CPU クロック = 12MHz, ベースライン値 >= 分解能最大カウントの 70%), PRS16
ノイズカウント		4		ピーク - ピーク	分解能 = 14
ノイズカウント		7		ピーク - ピーク	分解能 = 12
ノイズカウント		12		ピーク - ピーク	分解能 = 10

a. S/N 比 は、スキャン速度が遅く、ベースライン値が増加するにしたがってよくなります。

Table 4. キャパシタンス検知ノイズ

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 =12 MHz, CPU クロック = 6 MHz, 基準値 >= 分解能最大カウントの 70%)
ノイズカウント、 ピーク - ピーク		0.2		%, (ノイズカウント)/ (基底 - ベースライン値)	分解能 >= 14
ノイズカウント、 ピーク - ピーク		1		%, (ノイズカウント)/ (基底 - ベースライン値)	分解能 = 12
ノイズカウント、 ピーク - ピーク		10		%, (ノイズカウント)/ (基底 - ベースライン値)	分解能 = 10

a. S/N 比 は、スキャン速度が遅く、ベースライン値が増加するにしたがってよくなります。

Table 5. キャパシタンス検知ノイズ

パラメータ[a]	最小値	標準値	最大値	単位	テスト条件 (Vdd = 2.7V, SysClk =12 MHz, CPU クロック = 6 MHz, 基準値 >= 分解能最大カウントの 70%)
ノイズカウント		9		ピーク - ピーク	分解能 = 12

a. S/N 比 は、スキャン速度が遅く、ベースライン値が増加するにしたがってよくなります。

Table 6. キャパシタンススキャン中の電力消費

パラメータ	最小値	標準値	最大値	単位	テスト条件 (Vdd = 2.7V, SysClk = CPU Clock = 6 MHz)
アクティブ電流		1.43		mA	スキャン中の平均電流、8 センサ
スタンバイ電流		40		µA	スキャン速度 = 超高速、分解能 = 9、100ms レポートレート、8 センサ
		250		µA	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		5		µA	1 s レポート レート、1 センサ

Table 7. キャパシタンススキャン中の電力消費

パラメータ	最小値	標準値	最大値	単位	テスト条件 (Vdd = 3.3V, SysClk = CPU Clock = 6 MHz)
アクティブ電流		1.9		mA	スキャン中の平均電流、8 センサ
スタンバイ電流		50		µA	スキャン速度 = 超高速、分解能 = 9、100ms レポートレート、8 センサ
		300		µA	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		6		µA	1 s レポート レート、1 センサ

Table 8. キャパシタンススキャン中の電力消費

パラメータ	最小値	標準値	最大値	単位	テスト条件 (Vdd = 5.0V, SysClk = CPU クロック = 6 MHz)
アクティブ電流		3.18		mA	スキャン中の平均電流、8 センサ
スタンバイ電流		90		µA	スキャン速度 = 超高速、分解能 = 9、100ms レポートレート、8 センサ
		550		µA	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		7		µA	1 s レポート レート、1 センサ

Table 9. インクリメンタル ADC の特性

パラメータ	標準値	上限または下限	単位	条件および注意
動作電流		3.18	mA	5V 電源 CPU 6MHz
ノイズ、レシオメトリック		3	LSB	
分解能、レシオメトリック		9 ~ 16	ビット	
分解能、絶対		12	ビット	
サンプリング レート、レシオメトリック		0.04 ~ 16.6	Ksps	分解能とスキャンニング速度による
サンプリング レート、絶対		0.58 ~ 4.6	Ksps	
入力				
入力電圧範囲	-	Vss+0.3 ~ Vdd-0.3	V	
入力静電容量		3	pF	

配置

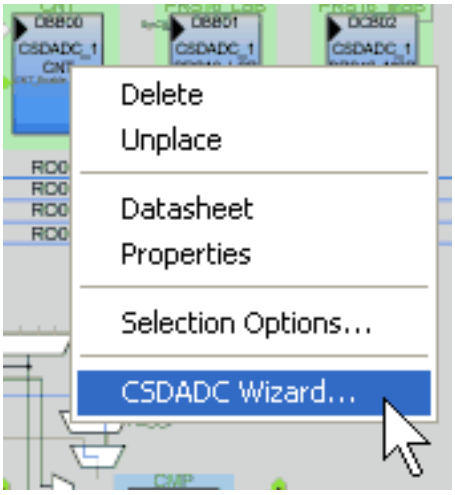
ユーザ モジュールのブロックは、ユーザ モジュールがインスタンス化されると自動的に配置されます。他の配置は利用できません。CSDADC は 1 ~ 3 のデジタルブロックを消費しますが、これは設定によって異なります。CSDADC のアナログ部は ACE00、ACE01、ASE11 に配置し、他の選択肢はありません。

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、CSDADC 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。

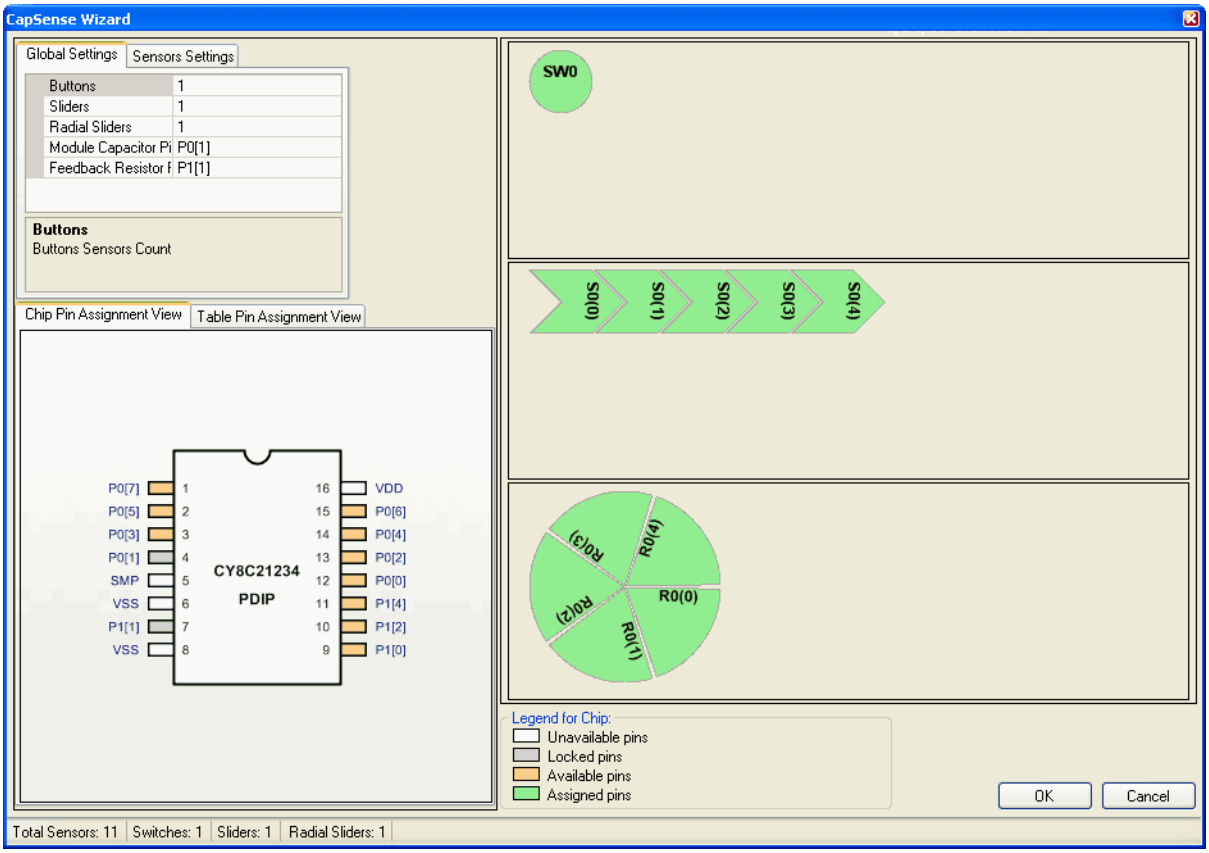
静電容量式センサの接続を配置する場合、P1[0] と P1[1] は避けてください。これらのピンは、そのデバイスのプログラミングに使用され、センサの検出感度とノイズに影響を与える過度のルーティング静電容量を持っている可能性があります。

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 CSDADC の任意のブロックを右クリックし、次に [CSDADC 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

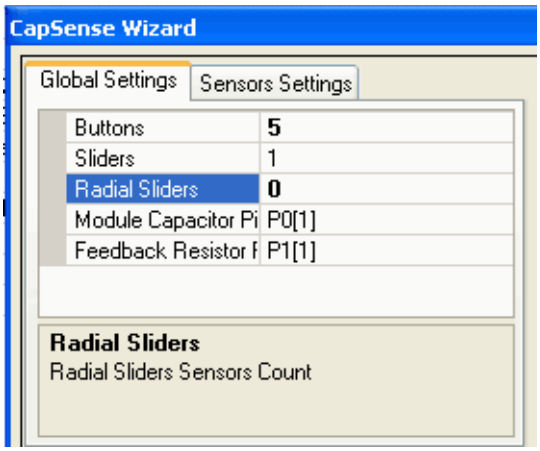
白 – このピンは CapSense の入力に使用できません。

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。

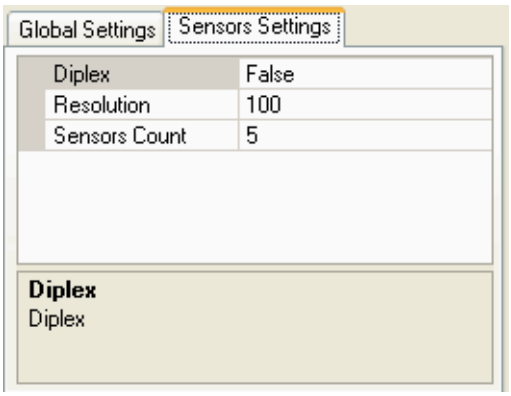
オレンジ – このピンは割り当て可能です。

グリーン – このピンは CapSense 入力に割り当てられています。

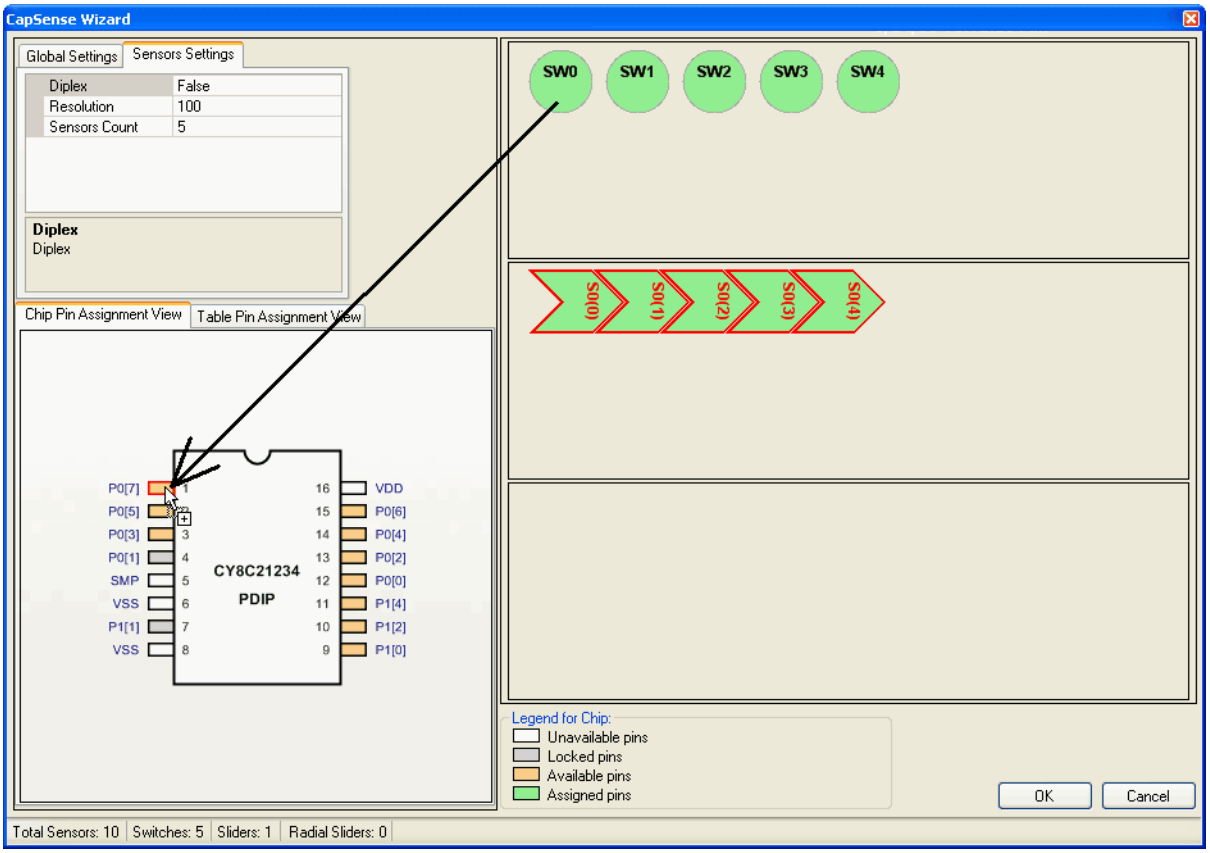
3. 独立したボタン、スライダ、ラジアル スライダの数を入力します。X-Y タッチパッドには 2 つのスライダが必要ですが、選択されるのは 1 つです。センサ数は、使用できるピン数に制限されています。



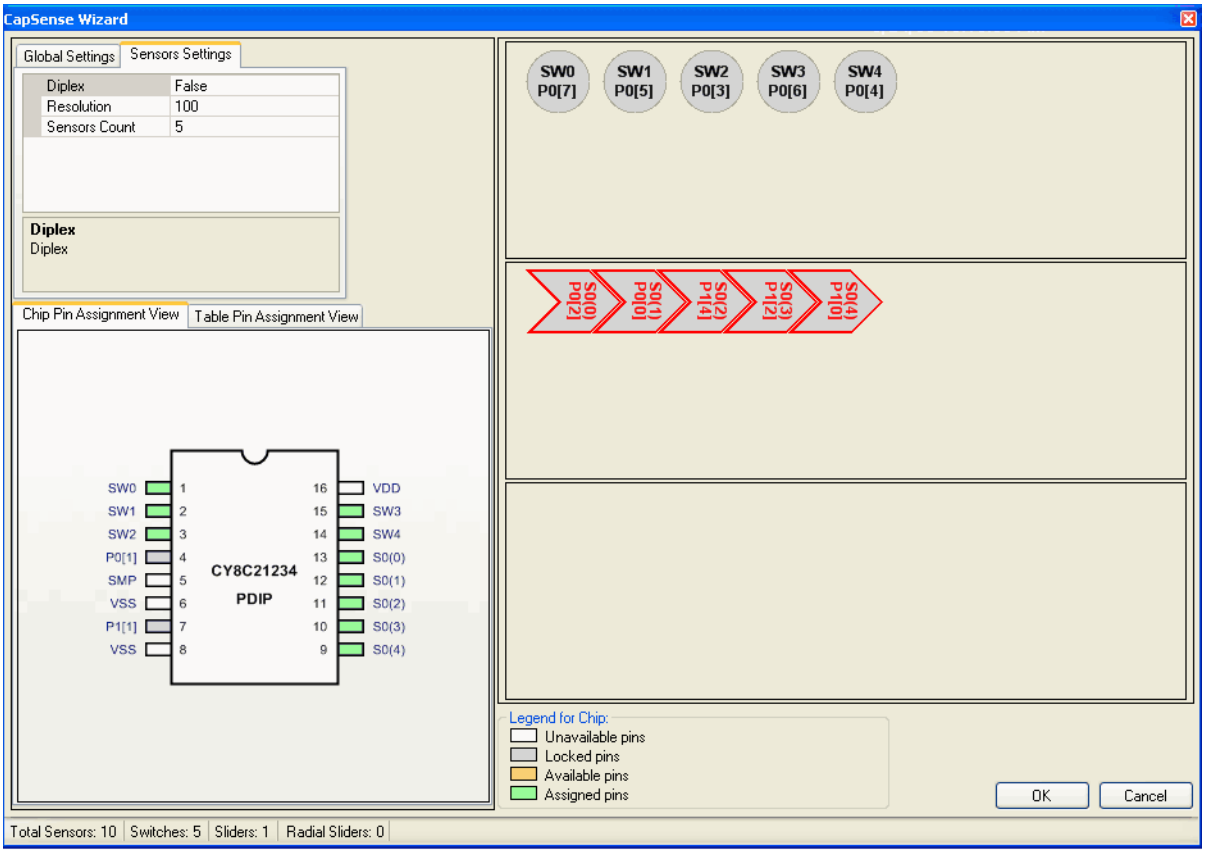
4. [センサ設定] タブを選択し、スライダとラジアルスライダの設定を指定します。各スライダのセンサ数を入力します。スライダセンサ中の現実的な最低センサ数は 5 で、最大数はピン数によって制限されます。



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 2^8 – 1 または (2 x センサに使用されるピン数 – 1) x 2^8 -1 です。
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上半分だけが示されています。下半分は、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。 「ダイプレックス」の項で、ピン説明のダイプレックス表を参照してください。
7. ボタンを左クリックし、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。



8. 他のボタンでも同じ操作を繰り返します。
9. 個々のスライダセンサを、ボタンの場合と同様に、物理ポート ピンにマッピングします。



10. [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。

CSDADC 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 CSDADC_Table.asm に保存されています。

センサ表

センサ表は各 2-byte 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。

```
CSDADC_Sensor_Table:
_CSDADC_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 CSDADC_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。2 番目の値は、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 バイトは固定小数点乗数であり、スライダの計算されたセント口イドに乗算され、CSDADC ウィザードで要求された分解能を達成します。

```
CSDADC_Group_Table:
_CSDADC_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。下記に 8 つのセンサを使用したスライダの典型例を示します。

```
DiplexTable_0:
; This group is not a diplexed slider
DiplexTable_1:
db0,1,2,3,4,5,6,7,0,3,6,1,4,7,2,5// 8 switch slider


 CSDADC_Diplex_Table:
_CSDADC_Diplex_Table:
db >DiplexTable_0, <DiplexTable_0
db >DiplexTable_1, <DiplexTable_1
```

パラメータおよびリソース

ADCEnabled

ADCEnabled パラメータは 以下の 2 つの値を持つことができます。

- Enabled (デフォルト値)
- Disabled (無効)

パラメータを Enabled に設定すると、ADC ルーチンが互換コードに含められ、ユーザのコードから呼び出すことができるようになります。このパラメータが Disabled に設定されていると、ADC ルーチンは含まれません。設計で ADC が不要になったとき、このパラメータを Disabled に設定すると便利なことがあります。このパラメータが Disabled に設定されていると、CSDADC ユーザ モジュールが CSD ユーザ モジュールのように作動します。

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、個々の実施センサ数と同等です。)

可能な値の範囲は 5 ー 255 です。デフォルト値は 40 です。

ダイ温度の測定

VTEMP 電圧を測定し、対応する温度値に変換するには、既存の ADC 機能を Enable または Disable にします。ADCEnabled パラメータが Enabled に設定される場合、パラメータは Enable のみに設定可能です。デフォルトでは、このパラメータは無効です。

ノイズ閾値

個々のセンサでは、このパラメータは閾値となるカウント値を設定し、これを上回ると基底値が更新されなくなります。スライダセンサでは、この閾値を下回るカウント値はセントロイドの計算に加えられません。可能な値は 5 から 255 です。デフォルト値は 20 です。

BaselineUpdate 閾値

新しい Raw カウント値が現在の基底値を上回っており、その差がノイズ閾値を下回る場合 (センサのオートリセットパラメータは無効にセットされている)、現在の基底値と Raw カウントの差はバケツに集められたような状態になります。バケツが満杯になると、基底値はある値分増分し、バケツは空になります。このパラメータは、基底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 から 255 です。パラメータ値が大きいときは、基底値の更新速度を遅くします。より頻繁な基底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。デフォルト値は 200 です。

LowBaselineReset

LowBaselineReset パラメータは、NegativeNoiseThreshold パラメータとともに作動します。指定されたサンプル数で、サンプルカウント値が (Baseline 値 - NegativeNoiseTreshold) 以下の場合、基底 (Baseline) 値は新しい Raw カウント値に設定されます。これは基本的に、異常に低い Raw カウントが認識されたときに、基底値をリセットする必要があるときに使われるもので、通常、指などが置かれたまま起動された時等の異常な状態をリセットする為に使用されます。。可能な値は 0 から 255 です。デフォルト値は 50 です。

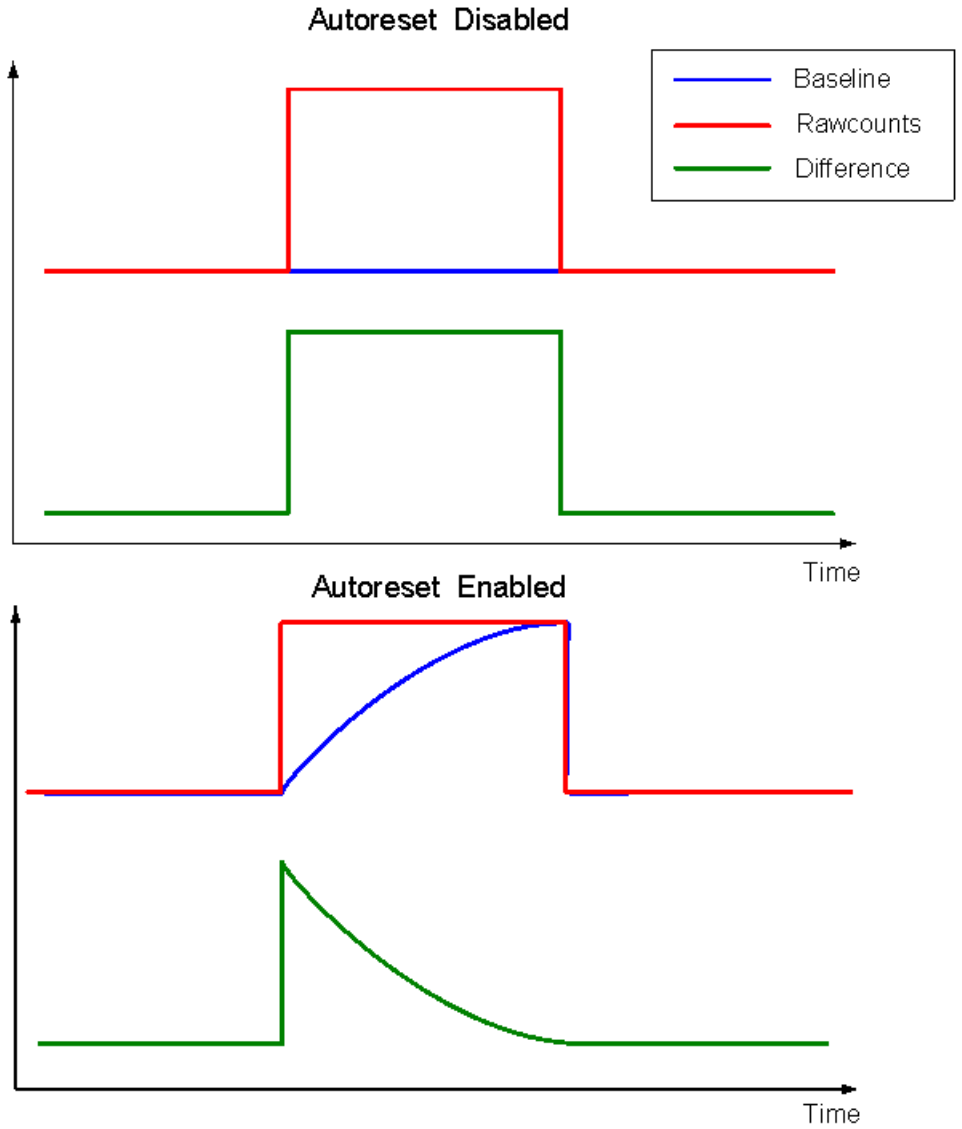
センサの Autoreset

このパラメータは、基底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、基底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力ウントが突然上がった際に、センサが恒久的にオンになるのを妨ぐことができます。この突然の上昇の原因には、大幅な電源電圧の変化、高エネルギー RF ノイズ源、非常に速い温度変化があります。このパラメータは、デフォルトでは Enabled です。

パラメータが [**Disabled**] (無効) にセットされている場合、Raw 値と基底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 Enabled にしておきます。

図 10 は、このパラメータの基底値更新への影響について図示しています。

Figure 10. センサ Autoreset パラメータ



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。センサが動作中でない場合、差の数は指閾値 + ヒステリシスを上回る必要があります。センサが動作中の場合、差の数は指閾値 - ヒステリシスを下回る必要があります。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、戻り値 bIsSensorActive() または baSnsOnMask[] アレイを使用してモニターされます。可能な値は 0 から 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。デフォルト値は 10 です。

適切な高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。

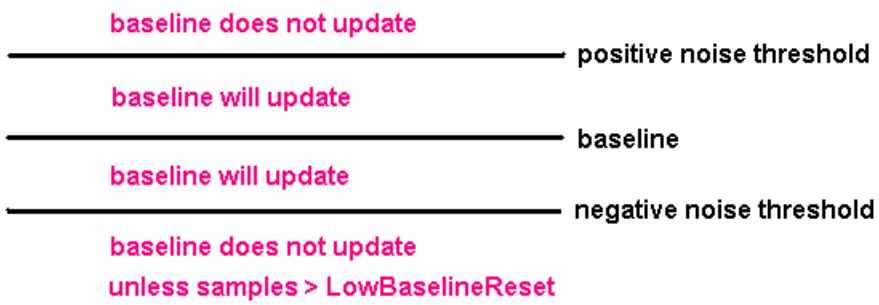
Debounce (デバウンス)

デバウンス パラメータは、センサの動作遷移のためのデバウンスカウンタを設定します。センサが非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。デバウンス数は、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ウンスは起こりません。デフォルト値は 3 です。

NegativeNoiseThreshold

NegativeNoiseThreshold パラメータは、マイナスの差カウント閾値を追加します。現在の Raw カウント値がベースライン値を下回り、これらの差 (Difference) 値がこの閾値を上回る場合、基底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 カウント値が低い状態で続く場合 差 (Difference) 値は閾値より大きい、基底値はリセットされます。可能な値は 0 から 255 です。デフォルト値は 20 です。



LowBaselineReset

LowBaselineReset パラメータは、NegativeNoiseThreshold パラメータとともに作動します。指定されたサンプル数で、サンプルカウント値が (Baseline) 値 - NegativeNoiseTreshold) 以下の場合、基底 (Baseline) 値は新しい Raw カウント値に設定されます。これは基本的に、異常に低い Raw カウントが認識されたときに、基底値をリセットする必要があるときに使われるもので、通常、指などが置かれたまま起動された時等の異常な状態をリセットする為に使用されます。可能な値は 0 から 255 です。デフォルト値は 50 です。

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。選択肢は次のとおりです。**超高速、高速、通常、低速**デフォルト値は通常です。スキャン速度が遅いと、次のような利点があります。

- 改善された S/N 比
- 電源と温度変化に対するより良いイミュニティ
- システム割り込みレイテンシの必要性が減少し、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。

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。センサは 9 ~ 16xA0;bits ビットの分解能でスキャンできます。デフォルト分解能は 12 ビットです。N ビットの分解能をスキャンする最大ローカウントは、2^N-1 です。

分解能を高くすると、検出感度とタッチ検知の 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 以上離れた手を検知できます。

スキャン速度と分解能は、VC1、VC2、VC3、ADCPWM 分周器に次のように影響を及ぼします。

スキャン速度	VC1
超高速	1
高速	2
通常	4
低速	8

分解能 (単位 : ビット)	VC2	VC3	ADCPWM
9	8	16	4
10	8	32	4
11	8	64	4
12	8	128	4
13	8	256	4
14	8	256	8
15	8	256	16
16	8	256	32

VCI 分周器は、スキャン速度によってのみ異なります。VC2、VC3、ADCPWM は分解能によってのみ異なります。

Table 10.　24xA0;MHzMHz IMO 動作におけるスキャン時間 (単位：µs)、スキャン xA0;Speed 速度と分解能、PRS16 構成

分解能 (単位：ビット)	スキャン速度			
	超高速	高速	通常	低速
9	75	110	170	300
10	110	170	300	510
11	170	300	510	1010
12	300	510	1010	2030
13	510	1010	2030	4060
14	850	1690	3380	6760
15	1520	3040	6080	12200
16	2880	5720	11500	23200

Table 11.　24 MHz IMO 動作における、スキャン時間 (単位：µs) 対スキャン速度と分解能、PRS8 構成またはプリスケーラを持つ PRS8 構成

分解能 (単位：ビット)	スキャン速度			
	超高速	高速	通常	低速
9	60	85	150	255
10	85	150	255	510
11	150	255	510	1020
12	255	510	1020	2040
13	510	1020	2040	4080
14	845	1700	3380	6760
15	1530	3060	6120	12100
16	2880	5800	11500	23000

Note　スキャン時間は、2 つのセンサスキャンの間の間隔を測定したものです。この時間には、センサのセットアップ時間、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。

変調器コンデンサ ピン

このパラメータは、外付け変調器コンデンサ（ C_{mod} ）を接続するピンをセットします。利用可能なピン P0[1]、P0[3] から選択します。デフォルトのピンは P0[1] です。

Feedback Resistor Pin (フィードバック 抵抗ピン)

このパラメータは、外付けフィードバック 抵抗 (R_b) を接続するピンを設定します。利用できるピン：P1[1]、P1[5]、P3[1] から選択します。デバイスパッケージによっては使用できないピンもあります。ヒント：これらのピンの一部が、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。今後の CSDADC ユーザ モジュールのバージ

ョンでは、フィードバック 抵抗の接続に使用できるピンが追加される可能性があります。これにより、P3 ポートを持たないパッケージで第 2 の I^2C ポートを使用できるようになります。プログラミングの問題を避けるために、P1[5] または P3[1] を使用してください。デフォルトのピンは P1[1] です。

リファレンス

このパラメータは、コンパレータのリファレンスソースを設定します。詳細については、以下の RefValue パラメータの説明を参照してください。デフォルトでは、基準はバンドギャップ (VBG) を参照します。

Ref Value (基準値)

コンパレータ基準がアナログ変調器 (ASE11) または外部フィルタリングによる PWM/PRSPWM 信号 (RC を用いて AnalogColumn_InputSelect_1 から) からのものである場合、このパラメータは、コンパレータの基準値を設定します。基準値がバンドギャップ (VBG) または外部分圧器から (抵抗分圧器を用いて AnalogColumn_InputSelect_1 から) の場合は、この値は効果がありません。デフォルト基準値は 0 です。

ここで '0' は、最小基準 (1/4 Vdd) に該当します。'8' は、最大値 (3/4Vdd) に該当します。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、より大きい静電容量を持つセンサで高い基準値を設定して、Raw カウントのバランスを取ることができます。

Note このパラメータは VC2 クロック源構成を用いた CSDADC では使用できません。

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決定します。このパラメータは、プリスケーラを持つ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。デフォルトプレスケーラ期間値は 7 です。

最大の信 S/N 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。

ShieldElectrodeOut

シールド電極信号源は、空いているデジタル行バス (Row_0_Output_0 ～ Row_0_Output_3).) のいずれかから選択できます。各行出力は、3 つのピンのいずれかに迂回できます。行 LUT 関数を A に設定してください。

Note このパラメータは、PRS8/PRS16 クロック源構成を持つ CSDADC でのみ使用できます。PWM8 クロック源構成を持つ SDADC で、シールド極性信号は恒久的に Row_0_Output_0. に接続されています。

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、インスタンス名が指定されます。デフォルトでは、PSoC Designer プロジェクトでは、CSDADC_1 をこのユーザ モジュールの最初のインスタンスとして指定します。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「CSDADC」となっています。

Note ** ここでは、全てのユーザー モジュールの API では、A と X レジスタの値は、API 関数を呼び出すことによって変更するることができます。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 が変更されないかもしれませんが、将来も変更されないという保証はありません。

大容量メモリデバイスでは、CUR_PP、IDX_PP、MVR_PP 及び MVW_PP レジスタの値を保護するのは呼び出し元の責任です。今すぐ修正されないレジスタもありますが、今後のリリースにこのケースが留まるという保証はありません。

エントリ ポイントは、CSDADC の初期化やサンプリングの開始、CSDADC の終了を行うためのものです。どの場合でも、モジュールのインスタンス名が次のエントリ ポイントの CSDADC 接頭辞を置き換えます。間違ったインスタンス名の使用は、構文エラーの一般的な原因です。

CSDADC API 関数の大部分は、CSD ユーザモジュールおよび ADC10 API からのものです。既存コードが両方のユーザ モジュールを使用した設計からのものである場合、コード変更は最小ですみます。関数の大部分は、2 つの別個のユーザ モジュールでも同じ関数名です。関数名を同じにしておくことの特殊作用として、非常によく似ている名前で CapSense モジュールと ADC モジュールで異なる機能を果たす関数があります。コーディングの際の間違いを避けるために、関数の説明をよく読んでください。ADC 機能をサポートするために、いくつかの新しい API ルーチンが追加されました。

API の機能は、様々なグローバルアレイを使用します。そのため、これらのアレイをマニュアルで変更してはいけません。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。

- CSDADC_waSnsBaseline[]
- CSDADC_waSnsResult[]
- CSDADC_waSnsDiff[]
- CSDADC_baSnsOnMask[]

CSDADC_waSnsBaseline[] – 各センサの基底値データを含む整数アレイです。アレイのサイズは静電容量式センサ数と同等です。CSDADC_waSnsBaseline[] アレイは以下の関数によって更新されます。

- CSDADC_UpdateAllBaselines();
- CSDADC_UpdateSensorBaseline();
- CSDADC_InitializeBaselines().

CSDADC_waSnsResult[] – 各静電容量式センサの Raw データを含む整数アレイです。アレイのサイズはセンサ数と同等です。CSDADC_waSnsResult[] データは次の関数によって更新されます。

- CSDADC_ScanSensor();
- CSDADC_ScanAllSensors().

CSDADC_waSnsDiff [] – 各静電容量式センサの Raw データと基底値データの差を含む整数アレイです。アレイのサイズはセンサ数と同等です。

CSDADC_baSnsOnMask[] – 静電容量式センサのオン・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。CSDADC_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)。CSDADC_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。CSDADC_baSnsOnMask[] データは、CSDADC_bIsSensorActive(BYTE bSensor) 関数または CSDADC_bIsAnySensorActive() ルーチンによって更新されます。

CSDADC_Start

説明：

レジスタを初期化し、ユーザーのモジュールを起動します。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  CSDADC_Start()
```

アセンブラ

```
lcall  CSDADC_Start
```

パラメータ：

なし

戻り値：

なし

副作用：

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_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、CSDADC_ClearSensors() を呼び出してすべてのセンサをリセットしてオフ状態にします。

C プロトタイプ

```
void  CSDADC_Stop()
```

アセンブラ

```
lcall  CSDADC_Stop
```

パラメータ：

なし

戻り値：

なし

副作用：

**

CSDADC_EnableADC

説明：

ユーザモジュールの ADC 関数を有効にします。いったん CSDADC_ABSOLUTE モードを選択すると、シングル スロープ ADC モードが開始し、ADC の校正が必要になります。詳細については、CSDADC_wCal() ルーチンを参照してください。CSDADC_RATIOMETRIC が選択されている場合、CSDADC は校正が不要です。

すべての ADC 関連 API は、CSDADC_EnableADC が呼び出されてから呼び出します。

この関数は、ADC がユーザ モジュール パラメータで有効になってからのみ使用可能です。

アセンブラ

```
mov A, Mode
lcall CSDADC_EnableADC
```

C プロトタイプ

```
void  CSDADC_EnableADC (BYTE Mode);
```

パラメータ：

モード – ADC 動作モードを指定します。このパラメータ用に、2 つの可能なオプションがサポートされます。

CSDADC_ABSOLUTE – 絶対電圧モードで ADC を構成します。

CSDADC_RATIOMETRIC – レシオメトリック測定モードで ADC を構成します。

戻り値：

なし

副作用：

**

CSDADC_EnableCapSense

説明：

このルーチンは CapSense 動作を可能にし、CapSense 関数にハードウェアの構成を設定します。

この関数は、ADC がユーザ モジュール パラメータで有効になってからのみ使用可能です。

アセンブラ

```
lcall CSDADC_EnableCapsense
```

C プロトタイプ

```
void  CSDADC_EnableCapsense(void);
```

パラメータ：

なし

戻り値：

なし

副作用：

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_EnableInput

説明：

入力ポートを ADC に接続します。接続はアナログ バスを使って行われるため、利用可能なポートはすべて ADC 入力関数に対応します。CSD モードへの切り替え前に、CSDADC_DisableInput(). を使用してピンの接続をアナログバスから外さなければなりません。同様に、CSDADC_EnableInput(). を使用して ADC 用のその他のピンを設定する前に、アナログバスからピンの接続を外さなければなりません。

この関数は、ADC がユーザ モジュール パラメータで有効になってからのみ使用可能です。

アセンブラ

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

C プロトタイプ：

```
void CSDADC_EnableInput(BYTE bMask, BYTE bPort);
```

パラメータ：

bPort - ADC の入力ポートを設定

bMask - ポートのビット

副作用：

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_DisableInput

説明：

選択したポート ピンをアナログ バスから切断することで、ADC から入力ポートを切断します。

この関数は、ADC がユーザ モジュール パラメータで有効になってからのみ使用可能です。

アセンブラ

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

C プロトタイプ：

```
void CSDADC_DisableInput(BYTE bMask, BYTE bPort);
```

パラメータ：

bPort - ADC の入力ポート

bMask - ポートのビット

副作用：

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_StartADC

説明：

ADC をオンにし、連続して実行します。

この関数は、ADC がユーザ モジュール パラメータで有効になってからのみ使用可能です。

アセンブラ

```
lcall CSDADC_StartADC
```

C プロトタイプ

```
void CSDADC_StartADC(void)
```

パラメータ：

なし

戻り値：

なし

副作用：

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_StopADC

説明：

ADC をオフにし、変換処理を直ちに停止します。

この関数は、ADC がユーザ モジュール パラメータで有効になってからのみ使用可能です。

アセンブラ

```
lcall CSDADC_StopADC
```

C プロトタイプ

```
void CSDADC_StopADC(void)
```

パラメータ：

なし

戻り値：

なし

副作用：

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_fIsDataAvailable

説明：

ADC のステータスを確認します。

この関数は、ADC がユーザ モジュール パラメータで有効になってからのみ使用可能です。

アセンブラ

```
lcall CSDADC_fIsDataAvailable
; Return value will be in A
```

C プロトタイプ

```
BYTE CSDADC_fIsDataAvailable(void)
```

パラメータ：

なし

戻り値：

データが変換され、読み取れる状態の場合は、ゼロ以外の値を返します。

副作用：

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_wGetData

説明：

変換されたデータを返します。データ サンプルが準備できたかどうかを確認するためには、CSDADC_fIsDataAvailable() を呼び出す必要があります。

この関数は、ADC がユーザ モジュール パラメータで有効になってからのみ使用可能です。

アセンブラ

```
lcall CSDADC_wGetData
;Data will be in A (LSB) and X(MSB) upon return
```

C プロトタイプ

```
WORD CSDADC_wGetData(void)
```

パラメータ：

なし

戻り値：

A および X レジスタを介した符号なし整数の形式で、変換されたデータ サンプルを返します。

副作用：

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_wGetDataClearFlag

説明：

データを取得して、データ利用可能フラグをリセットします。

この関数は、ADC がユーザ モジュール パラメータで有効になってからのみ使用可能です。

アセンブラ

```
lcall CSDADC_wGetDataClearFlag
;Data will be in A (LSB) and X(MSB) upon return
```

C プロトタイプ

```
WORD CSDADC_wGetDataClearFlag(void)
```

パラメータ：

なし

戻り値：

A および X レジスタを介した符号なし整数の形式で、変換されたデータ サンプルを返します。

副作用

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_wCal

説明：

SC ブロックでコンデンサをトリミングすることによって ADC を校正し、変換に使用する基準ランプのスロープを変化させます。ADC 入力は、ユーザが選択した校正基準である固定リファレンス電圧に設定しなければなりません。この関数は、CSDADC_Start の呼び出しと CSDADC_StartADC. の呼び出しの間に呼び出します。

この関数は、ADC がユーザ モジュール パラメータで有効になってからのみ使用可能です。

C プロトタイプ

```
int   CSDADC_wCal(WORD wVal)
```

アセンブラ

```
mov   A, [wVal+1]
mov   X, [wVal]
lcall  CSDADC_wCal
;Data will be in A (LSB) and X(MSB) upon return
```

パラメータ：

wVal: この数値は、入力が設定される基準電圧に基づいた期待値です。CSDADC_wCal() ルーチンは、結果が wVal と同じ、または wVal にできるだけ近くなるまで、SC ブロックでコンデンサをトリミングします。

戻り値：

もっとも近い結果を示す符号無しの整数を可能な wVal に返します。

副作用：

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_StartTempMeasurement

Note この API は、「ダイ温度測定」パラメータが 「Enabled」に設定されている場合のみ利用可能です。

説明：

VTemp センサ出力を PMux 入力に接続します。ADC 測定を開始します。API 使用例については、「ダイ温度測定」セクションを参照してください。

C プロトタイプ

```
void CSDADC_StartTempMeasurement (void)
```

アセンブラ

```
lcall  CSDADC_StartTempMeasurement
```

パラメータ：

なし

戻り値：

なし

副作用

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_GetTemperature

Note この API は、「ダイ温度測定」パラメータが 「Enabled」に設定されている場合のみ利用可能です。

説明：

ADC 測定を停止し、ダイ温度値を計算し、ADC 入力を回復し、データ利用可能フラグをリセットします。CSDADC_fIsDataAvailableAPI は、データサンプルが準備できたことを API が確認する前に呼び出す必要があります。API 使用例に s ついては、「ダイ温度測定」セクションを参照してください。

ダイ温度 は、次の式を使用して計算できます。

DieTemp = M * wADCCount * wCalibrationVoltage / wCalibrationCount - B

この式では、

wADCCount - ADC VTemp 電圧測定の結果

wCalibrationVoltage - API パラメータ

wCalibrationCount - CSDADC_wCal(WORD wVal) API 関数の wVal パラメータの内部に保管された値

M, B - フラッシュに保管された係数

C プロトタイプ：

```
CHAR CSDADC_GetTemperature(WORD wCalibrationVoltage)
```

アセンブラ：

```
mov A, [wCalibrationVoltage+1] ; LSB of API parameter
mov X, [wCalibrationVoltage] ; MSB of API parameter
lcall  CSDADC_GetTemperature
```

パラメータ：

WORD wCalibrationVoltage: ADC 校正に使用される校正電圧 (mV) この電圧は CSDADC_wCalAPI 機能の wVal パラメータ （カウント）に対応します。

戻り値：

'A' には、ダイ温度 （摂氏）が含まれます。

副作用

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_ScanSensor

説明：

選択された静電容量式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 CSDADC 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 CSDADC_ScanSensor(BYTE bSensor);
```

アセンブラ

```
mov A,  bSensor
lcall CSDADC_ScanSensor
```

パラメータ：

レジスタ AxA0;contains 呼び出し前のセンサ番号を含みます。

戻り値：

なし

副作用

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_ScanAllSensors

説明：

各センサインデックスに対して CSDADC_ScanSensor() を呼び出して、構成済み静電容量式センサをすべてスキャンします。

C プロトタイプ

```
void CSDADC_ScanAllSensors();
```

アセンブラ

```
lcall CSDADC_ScanAllSensors
```

パラメータ：

なし

戻り値：

なし

副作用

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_UpdateSensorBaseline

説明：

この経時的カウント値は、静電容量式センサ別に独立して計算され、センサの基底値と呼ばれます。基底 (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. CSDADC_UpdateSensorBaseline() が呼び出されるたびに、Raw カウント値を前回の基底 (Baseline) 値から引いて差 (Difference) 値を計算します。この差は CSDADC_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、CSDADC_UpdateSensorBaseline() が呼び出されるたびに、差 (Difference) 値をノイズ閾値と比較します。Diff カウント値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 BaselineUpdateThreshold 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. 差 (Difference) 値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 CSDADC_UpdateSensorBaseline(BYTE bSensor)
```

アセンブラ

```
mov   A,   bSensor
lcall  CSDADC_UpdateSensorBaseline
```

パラメータ：

A => Sensor 番号

戻り値：

なし

副作用：

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_UpdateAllBaselines

説明：

CSDADC_bUpdateSensorBaseline() 関数を使用して、すべての静電容量式センサの基底値を更新します。

C プロトタイプ

```
void  CSDADC_UpdateAllBaselines()
```

アセンブラ

```
lcall CSDADC_UpdateAllBaselines
```

パラメータ：

なし

戻り値：

なし

副作用：

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_bIsSensorActive

説明：

センサの差 (Difference) 値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、CSDADC_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  CSDADC_bIsSensorActive(BYTE bSensor)
```

アセンブラ

```
mov  A,  bSensor
lcall CSDADC_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。

副作用：

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_bIsAnySensorActive

説明：

全静電容量式センサの差 (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 CSDADC_bIsSensorActive() を呼び出し、CSDADC_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  CSDADC_bIsAnySensorActive()
```

アセンブラ

```
lcall CSDADC_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：

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_wGetCentroidPos

説明：

差 (Difference) 値アレイを確認し、セントロイドを探します。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、セントロイド位置が CSDADC 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 CSDADC ウィザードで定義されている場合のみ利用できます。

C プロトタイプ

```
WORD  CSDADC_wGetCentroidPos(BYTE bSnsGroup)
```

アセンブラ

```
mov  A,  bSnsGroup
lcall CSDADC_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。

副作用：

このルーチンは、ノイズ閾値を引くことによって差 (Difference) 値カウントを調整します。マイナスの差 (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。差 (Difference) 値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、この関数はゼロから CSDADC ウィザードで設定された分解能の値までを返します。アクティブのセンサがない場合、関数は –1 (FFFFh) を返します。セントロイド / ダイプレックスアルゴリズムの実行中にエラーが発生した場合、関数は –1 (FFFFh) を返します。必要に応じて、CSDADC_bIsSensorActive() 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。

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偽のセントロイド結果を示すことがあります。ノイズが偽のセントロイド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)。

CSDADC_InitializeSensorBaseline

説明：

選択されたセンサをスキャンして、初期値を伴う CSDADC_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基底値の配列エレメントにコピーされます。この関数は、個々のセンサの基底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 CSDADC_InitializeSensorBaseline(BYTE bSensor)
```

アセンブラ

```
mov A, bSensor
lcall CSDADC_InitializeSensorBaseline
```

パラメータ：

A => センサ番号

戻り値：

なし

副作用：

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_InitializeBaselines

説明：

それぞれのセンサをスキャンして、CSDADC_waSnsBaseline[] アレイに初期値をロードします。Raw カウント値は各センサの基底値アレイにコピーされます。

C プロトタイプ

```
void CSDADC_InitializeBaselines()
```

アセンブラ

```
lcall CSDADC_InitializeBaselines
```

パラメータ：

なし

戻り値：

なし

副作用：

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_SetDefaultFingerThresholds

説明：

FingerThreshold パラメータ値を伴う CSDADC_baBtnFThreshold[] アレイを読み込みます。CSDADC_baBtnFThreshold[] アレイがカスタム値とともにマニュアル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 CSDADC_SetDefaultFingerThresholds()
```

アセンブラ

```
lcall CSDADC_SetDefaultFingerThresholds
```

パラメータ：

なし

戻り値：

なし

副作用：

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_SetScanMode

説明：

静電容量式センサのスキャン速度と分解能、およびインクリメンタル ADC の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 µ? 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でスキャンの頻度を下げます。この関数は CSDADC_ScanSensor() 関数とともに使用できます。

C プロトタイプ

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

アセンブラ

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

パラメータ：

bSpeed: スキャン速度

次の定数は bSpeed パラメータ用です。

定数	値
CSDADC_ULTRA_FAST_SPEED	0x00
CSDADC_FAST_SPEED	0x01
CSDADC_NORMAL_SPEED	0x02
CSDADC_SLOW_SPEED	0x03

bResolution: スキャン分解能 この値を必要な分解能ビット数に設定します。このパラメータ値は、9 以上 16 以下です。

bResolution パラメータには以下の定数が与えられます。

定数	値
CSDADC_9_BIT_RESOLUTION	9
CSDADC_10_BIT_RESOLUTION	10
CSDADC_11_BIT_RESOLUTION	11
CSDADC_12_BIT_RESOLUTION	12

定数	値
CSDADC_13_BIT_RESOLUTION	13
CSDADC_14_BIT_RESOLUTION	14
CSDADC_15_BIT_RESOLUTION	15
CSDADC_16_BIT_RESOLUTION	16

戻り値：

なし

副作用：

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_SetRefValue

説明：

静電容量式センサのスキャン基底値を設定します。基準がアナログ変調器 (基準パラメータの ASE11) または外部的にフィルタされる PWM/PRS 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基準電圧になります。値 8 は、最大基準電圧を設定し、感度が低くなります。この関数は CSDADC_ScanSensor(). とともに使用できます。

この関数は VC2 クロック源構成を用いた CSDADC では使用できません。

C プロトタイプ

```
void CSDADC_SetRefValue(BYTE bRefValue);
```

アセンブラ

```
mov     A, bRefValue
lcall    CSDADC_SetRefValue
```

パラメータ：

bRefValue - スキャンの基準値を設定します。許可されている値は 0..8 です。

戻り値：

なし

副作用：

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_ClearSensors

説明：

各センサに対して CSDADC_wGetPortPin() と CSDADC_DisableSensor() を続けて呼び出し、静電容量式センサをノンサンプリング状態にクリアします。

C プロトタイプ

```
void  CSDADC_ClearSensors()
```

アセンブラ

```
lcall  CSDADC_ClearSensors
```

パラメータ：

なし

戻り値：

なし

副作用：

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_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  CSDADC_wReadSensor(BYTE bSensor)
```

アセンブラ

```
mov A, bSensor
lcall CSDADC_wReadSensor
```

パラメータ：

A => Sensor 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：

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_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、CSDADC_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、CSDADC_EnableSensor()、CSDADC_DisableSensor(). へと渡すことができます。

C プロトタイプ

```
WORD  CSDADC_wGetPortPin(BYTE bSensorNum)
```

アセンブラ

```
mov  A,  bSensorNumber
lcall     CSDADC_wGetPortPin
;Data will be in A (LSB, Sensor Bitmap) and X(MSB, Port Number) upon return
```

パラメータ：

bSensorNumber – 範囲は 0 ~ (n – 1) です。ここで n は、CSDADC ウィザードで設定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、CSDADC_wGetPortPin() によって使用されます。

戻り値：

A => Sensor ビットマップ

X => Port 番号

副作用：

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_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、CSDADC_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 ハイ インピーダンス) 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  CSDADC_EnableSensor(BYTE bMask, BYTE bPort)
```

アセンブラ

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

パラメータ：

A => Sensor ビットマップ

X => Port 番号

戻り値：

なし

副作用：

API セクションの冒頭にある注意事項 ** を参照してください。

CSDADC_DisableSensor

説明：

CSDADC_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、CSDADC_wGetPortPin() 関数により返されます。

C プロトタイプ

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

アセンブラ

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

パラメータ：

A => Sensor ビットマップ

X => Port 番号

戻り値：

なし

副作用：

API セクションの冒頭にある注意事項 ** を参照してください。

ダイ温度測定

ダイ温度測定は、「ダイ温度測定」パラメータが 「Enabled」に設定されている場合のみ可能です。

コード例 :

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

void main(void)
{
    CHAR cTemp;
    WORD wVoltage;

    M8C_EnableGInt;
    CSDADC_Start(); // Start the User Module
    CSDADC_EnableADC (CSDADC_ABSOLUTE);

// Calibrate the user module using the 1.3 V internal BandGap reference source
ACE01CR1 &= ~0x04;           // Connect Vref to PMux input of ACE01 Analog Block
    CSDADC_wCal(0x428);        // Calibrate ADC
    ACE01CR1 |= 0x04;               // Restore PMux connection to Analog MUX Bus

    while (1)
    {
        CSDADC_EnableInput(0x01, 0x00); // use P0[0]
        CSDADC_StartADC();
        while (!CSDADC_fIsDataAvailable());
        wVoltage = CSDADC_wGetDataClearFlag();
        CSDADC_StopADC();
        CSDADC_DisableInput(0x01, 0x00); // required for normal CSD operation

CSDADC_StartTempMeasurement();
        while (!CSDADC_fIsDataAvailable());
        cTemp = CSDADC_GetTemperature(1300);

/* Add user code here to display the results of
        voltage and temperature measurement */
    }
}
```

Note

1. ADC は、CSDADC_StartTempMeasurementAPI を呼び出す前に校正しなければなりません。
2. CSDADC_StartTempMeasurementAPI は CSDADC_StartADCAPI を呼び出します。その結果、温度測定を開始する前に API を呼び出す必要はありません。
3. CSDADC_GetTemperatureAPI は ADC 変換を停止します。ダイ温度測定後に電圧を測定する必要がある場合、CSDADC_StartADCAPI を呼びださなければなりません。

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。

レシオメトリック入力 ADC モードと絶対入力 ADC モードの両方を使用して、ランタイムにモードを切り替えられるため、異なるセンサタイプとのインターフェースが簡単にできます。

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

//#define ACD_KIND  CSDADC_RATIOMETRIC
#define ACD_KIND  CSDADC_ABSOLUTE

WORD wCal;
WORD wResult;

void main(void)
{
  M8C_EnableGInt;
  CSDADC_Start();
  CSDADC_SetDefaultFingerThresholds();
CSDADC_InitializeBaselines();

#if (ACD_KIND==CSDADC_ABSOLUTE)
  CSDADC_EnableADC(ACD_KIND);
  wCal = CSDADC_wCal(1000);
#endif

  while (1) {
    CSDADC_EnableCapsense();
    CSDADC_ScanAllSensors();
    CSDADC_UpdateAllBaselines();

CSDADC_EnableADC(ACD_KIND);
    CSDADC_EnableInput(0x01, 0x02); // use P2[0]

CSDADC_StartADC();

while (0 == CSDADC_fIsDataAvailable());
    wResult = CSDADC_wGetDataClearFlag();

CSDADC_StopADC();

CSDADC_DisableInput(0x01, 0x02); // required for normal CSD operation
  }
}
```

アセンブリである同一プロジェクト：

```
include "m8c.inc"       ; part specific constants and macros
include "memory.inc"    ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

export _main
```

```
; ACD_KIND: EQU  CSDADC_RATIOMETRIC
  ACD_KIND: EQU  CSDADC_ABSOLUTE

_main:

    M8C_EnableGInt

lcall  CSDADC_Start
lcall  CSDADC_SetDefaultFingerThresholds
    lcall  CSDADC_InitializeBaselines

IF (ACD_KIND & CSDADC_ABSOLUTE)
    mov    A, CSDADC_ABSOLUTE
    lcall  CSDADC_EnableADC

mov    A, <1000
    mov    X, >1000
    lcall  CSDADC_wCal
    ;calibration data are located in A (LSB) and X(MSB)
ENDIF

loop:
    lcall  CSDADC_EnableCapsense
    lcall  CSDADC_ScanAllSensors
    lcall  CSDADC_UpdateAllBaselines

mov    A, ACD_KIND
    lcall  CSDADC_EnableADC

mov    A, 0x01
    mov    X, 0x02
lcall  CSDADC_EnableInput  ; use P2[0]

lcall  CSDADC_StartADC

.scan:
lcall  CSDADC_fIsDataAvailable
    cmp    A, 0
    jz     .scan
    lcall  CSDADC_wGetDataClearFlag

; The ADC rezult is stored in A (LSB) and X(MSB)

lcall  CSDADC_StopADC

    mov    A, 0x01
    mov    X, 0x02
    lcall  CSDADC_DisableInput  ; required for normal CSD operation

    jmp    loop
```

コンフィグレーション レジスタ

PRS16 クロック源構成レジスタを持つ CSDADC

Table 12. ブロック CMP、レジスタ：ACE_CONTROL1 (ACE01CR1)、バンク X

ビット	7	6	5	4	3	2	1	0
値	0	1	リファレンス			1	1	1

リファレンスは、リファレンスパラメータにより維持されます。

Table 13. ブロック CMP、レジスタ：ACE_CONTROL2 (ACE01CR2)、バンク X

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	Power	

電源は、CSDADC_Start() と CSDADC_Stop() API で制御されます。

Table 14. ブロック CMP_REF、レジスタ：ASE_CONTROL (ASE11CR0)、バンク X

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 15. ブロック CMP_REF、レジスタ：ADC_CONTROL (ADC1_CR), バンク 0

ビット	7	6	5	4	3	2	1	0
値	CMPST	1	1	0	0	オート	0	ADCEN

CMPST は読み出し専用で、ADC 関連 API によって使用されます。オートビットは ADC 関連 API を介して維持されます。ADCEN は、ADC 動作を可能にし、CSDADC API によって維持されます。

Table 16. ブロック CMP_REF、レジスタ：ADC_TRIM (ADC1_TR), バンク 1

ビット	7	6	5	4	3	2	1	0
値	ADCTrimValue							

ADCTrimValue は、CSDADC API 制御です。シングル スロープ ADC 用の ADC トリム値、および CSD 動作用のゼロ値を含みます。

Table 17. ブロック CNT、レジスタ：関数 (DxBxxFN)、バンク 1

ビット	7	6	5	4	3	2	1	0
値	0	0	1	0	0	0	0	1

Table 18. ブロック CNT、レジスタ：入力 (DxBxxIN)、バンク 1

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 19. ブロック CNT、レジスタ：出力 (DxBxxOU)、バンク 1

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 CNT、レジスタ：制御 (DxBxxCR0)、バンク 1

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	Enable

イネーブル ビットは、カウンタ動作を可能にして、CSDADC API によって維持されます。

Table 21. ブロック CNT、レジスタ：期間 (DxBxxDR0)、バンク 0

ビット	7	6	5	4	3	2	1	0
値	CounterValue 直接の読み取りや書き込みは不可							

Table 22. ブロック CNT、レジスタ：期間 (DxBxxDR1)、バンク 0

ビット	7	6	5	4	3	2	1	0
値	1	1	1	1	1	1	1	1

Table 23. ブロック CNT、レジスタ：比較 (DxBxxDR2)、バンク 0

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 PRS、レジスタ：関数 (DxBxxFN)、バンク 1

ビット	7	6	5	4	3	2	1	0
LSB	0	0	0	0	1	0	1	0
MSB	0	1	1	0	1	0	1	0

Table 25. ブロック PRS、レジスタ：入力 (DxBxxIN)、バンク 1

ビット	7	6	5	4	3	2	1	0
LSB	0	0	0	0	0	0	0	0
MSB	0	0	1	1	0	0	0	0

Table 26. ブロック PRS、レジスタ：出力 (DxBxxOU)、バンク 1

ビット	7	6	5	4	3	2	1	0
LSB	1	1	0	0	0	0	0	0
MSB	1	1	1	0	0	ShieldElectrodeOut		

ShieldElectrodeOut は、シールド電極からの信号を生出力に出力します。これは、同じ名前のユーザ モジュール パラメータで制御されます。

Table 27. ブロック PRS、レジスタ：制御 (DxBxxCR0)、バンク 1

ビット	7	6	5	4	3	2	1	0
LSB	0	0	0	0	0	0	0	Enable
MSB	0	0	0	0	0	0	0	0

Enable ビットは、PRS ブロックを有効にし、CSDADC API で維持されます。

Table 28. ブロック PRS、レジスタ：シフト (DxBxxDR0)、バンク 0

ビット	7	6	5	4	3	2	1	0
LSB	PRS シフト レジスタ (LSB) - 直接アクセスなし							
MSB	PRS シフト レジスタ (MSB) - 直接アクセスなし							

Table 29. ブロック PRS、レジスタ：多項 (DxBxxDR1)、バンク 0

ビット	7	6	5	4	3	2	1	0
LSB	PRS 多項 (LSB)							
MSB	PRS 多項 (MSB)							

PRS 多項は、ScanSpeed パラメータ及び Resolution パラメータに応じて、CSDADC API によって制御されます。

Table 30. ブロック PRS、レジスタ：シード (DxBxxDR2)、バンク 0

ビット	7	6	5	4	3	2	1	0
LSB	PRS シード / 比較レジスタ (LSB)							
MSB	PRS シード / 比較レジスタ (MSB)							

このレジスタの値は、ほぼすべてのパラメータ値に応じて、API によって制御されます。

PWM8 クロック源構成レジスタを持つ CSDADC

Table 31. ブロック CMP、レジスタ：ACE_CONTROL1 (ACE01CR1)、バンク X

ビット	7	6	5	4	3	2	1	0
値	0	1	リファレンス			1	1	1

リファレンスは、同じ名前のユーザ モジュール パラメータで維持されます。

Table 32. ブロック CMP、レジスタ：ACE_CONTROL2 (ACE01CR2)、バンク X

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	Power	

電源は、CSDADC_Start() と CSDADC_Stop() API で制御されます。

Table 33. ブロック CMP_REF、レジスタ：ASE_CONTROL (ASE11CR0)、バンク X

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 34. ブロック CMP_REF、レジスタ：ADC_CONTROL (ADC1_CR), バンク 0

ビット	7	6	5	4	3	2	1	0
値	CMPST	1	1	0	0	オート	0	ADCEN

CMPST は読み出し専用で、ADC 関連 API によって使用されます。オートビットは ADC 関連 API を介して維持されます。ADCEN は、ADC 動作を可能にし、CSDADC API によって維持されます。

Table 35. ブロック CMP_REF、レジスタ：ADC_TRIM (ADC1_TR), バンク 1

ビット	7	6	5	4	3	2	1	0
値	ADCTrimValue							

ADCTrimValue は、CSDADC API 制御です。シングル スロープ ADC 用の ADC トリム値、および CSD 動作用のゼロ値を含みます。

Table 36. ブロック CNT、レジスタ：関数 (DxBxxFN)、バンク 1

ビット	7	6	5	4	3	2	1	0
値	0	0	1	0	0	0	0	1

Table 37. ブロック CNT、レジスタ：入力 (DxBxxIN)、バンク 1

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 38. ブロック CNT、レジスタ：出力 (DxBxxOU)、バンク 1

ビット	7	6	5	4	3	2	1	0
値	0	1	0	0	0	0	0	0

Table 39. ブロック CNT、レジスタ：制御 (DxBxxCR0)、バンク 1

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	Enable

イネーブル ビットは、カウンタ動作を可能にして、CSDADC API によって維持されます。

Table 40. ブロック CNT、レジスタ：期間 (DxBxxDR0)、バンク 0

ビット	7	6	5	4	3	2	1	0
値	CounterValue 直接の読み取りや書き込みは不可							

Table 41. ブロック CNT、レジスタ：期間 (DxBxxDR1)、バンク 0

ビット	7	6	5	4	3	2	1	0
値	1	1	1	1	1	1	1	1

Table 42. ブロック CNT、レジスタ：比較 (DxBxxDR2)、バンク 0

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 43. ブロック PWM、レジスタ：関数 (DxBxxFN)、バンク 1

ビット	7	6	5	4	3	2	1	0
値	0	1	1	0	0	0	0	1

Table 44. ブロック PWM、レジスタ：入力 (DxBxxIN)、バンク 1

ビット	7	6	5	4	3	2	1	0
値	0	0	0	1	0	0	0	0

Table 45. ブロック PWM、レジスタ：出力 (DxBxxOU)、バンク 1

ビット	7	6	5	4	3	2	1	0
値	1	1	0	0	0	1	0	0

Table 46. ブロック PWM、レジスタ：制御 (DxBxxCR0)、バンク 1

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	Enable

Enable ビットは、PRS ブロックを有効にし、CSDADC API で制御されます。

Table 47. ブロック PWM、レジスタ：カウント (DxBxxDR0)、バンク 0

ビット	7	6	5	4	3	2	1	0
値	カウント レジスタ - 直接アクセスなし							

Table 48. ブロック PWM、レジスタ：期間 (DxBxxDR1)、バンク 0

ビット	7	6	5	4	3	2	1	0
値	PrescalerPeriod (プリスケーラ期間)							

PrescalerPeriod は、同じ名前のユーザ モジュール パラメータで制御されます。

Table 49. ブロック PWM、レジスタ：比較 (DxBxxDR2)、バンク 0

ビット	7	6	5	4	3	2	1	0
値	比較							

このレジスタの値は、リファレンス ユーザ モジュールのパラメータ値に応じて、API によって制御されます。

PRS8 クロック源構成レジスタを持つ CSDADC

Table 50. ブロック CMP、レジスタ：ACE_CONTROL1 (ACE01CR1)、バンク X

ビット	7	6	5	4	3	2	1	0
値	0	1	リファレンス			1	1	1

リファレンスは、同じ名前のユーザ モジュール パラメータで維持されます。

Table 51. ブロック CMP、レジスタ：ACE_CONTROL2 (ACE01CR2)、バンク X

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	Power	

電源は、CSDADC_Start() と CSDADC_Stop() API で制御されます。

Table 52. ブロック CMP_REF、レジスタ：ASE_CONTROL (ASE11CR0)、バンク X

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 53. ブロック CMP_REF、レジスタ：ADC_CONTROL (ADC1_CR), バンク 0

ビット	7	6	5	4	3	2	1	0
値	CMPST	1	1	0	0	オート	0	ADCEN

CMPST は読み出し専用で、ADC 関連 API によって使用されます。オートビットは ADC 関連 API を介して維持されます。ADCEN は、ADC 動作を可能にし、CSDADC API によって維持されます。

Table 54. ブロック CMP_REF、レジスタ：ADC_TRIM (ADC1_TR), バンク 1

ビット	7	6	5	4	3	2	1	0
値	ADCTrimValue							

ADCTrimValue は、CSDADC API 制御です。シングル スロープ ADC 用の ADC トリム値、および CSD 動作用のゼロ値を含みます。

Table 55. ブロック CNT、レジスタ：関数 (DxBxxFN)、バンク 1

ビット	7	6	5	4	3	2	1	0
値	0	0	1	0	0	0	0	1

Table 56. ブロック CNT、レジスタ：入力 (DxBxxIN)、バンク 1

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 57. ブロック CNT、レジスタ：出力 (DxBxxOU)、バンク 1

ビット	7	6	5	4	3	2	1	0
値	0	1	0	0	0	0	0	0

Table 58. ブロック CNT、レジスタ：制御 (DxBxxCR0)、バンク 1

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	Enable

イネーブル ビットは、カウンタ動作を可能にして、CSDADC API によって維持されます。

Table 59. ブロック CNT、レジスタ：期間 (DxBxxDR0)、バンク 0

ビット	7	6	5	4	3	2	1	0
値	CounterValue 直接の読み取りや書き込みは不可							

Table 60. ブロック CNT、レジスタ：期間 (DxBxxDR1)、バンク 0

ビット	7	6	5	4	3	2	1	0
値	1	1	1	1	1	1	1	1

Table 61. ブロック CNT、レジスタ：比較 (DxBxxDR2)、バンク 0

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 62. ブロック PRS、レジスタ：関数 (DxBxxFN)、バンク 1

ビット	7	6	5	4	3	2	1	0
値	0	1	1	0	1	0	1	0

Table 63. ブロック PRS、レジスタ：入力 (DxBxxIN)、バンク 1

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 64. ブロック PRS、レジスタ：出力 (DxBxxOU)、バンク 1

ビット	7	6	5	4	3	2	1	0
値	1	1	1	0	0	ShieldElectrodeOut		

ShieldElectrodeOut は、シールド電極からの信号を生出力に出力します。これは、同じ名前のユーザ モジュール パラメータで制御されます。

Table 65. ブロック PRS、レジスタ：制御 (DxBxxCR0)、バンク 1

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	Enable

Enable ビットは、PRS ブロックを有効にし、CSDADC API で制御されます。

Table 66. ブロック PRS、レジスタ：シフト (DxBxxDR0)、バンク 0

ビット	7	6	5	4	3	2	1	0
値	PRS シフト レジスタ - 直接アクセスなし							

Table 67. ブロック PRS、レジスタ：多項 (DxBxxDR1)、バンク 0

ビット	7	6	5	4	3	2	1	0
値	PRS Polynomial (PRS 多項)							

PRS 多項は、ScanSpeed パラメータ及び Resolution パラメータに応じて、CSDADC API によって制御されます。

Table 68. ブロック PRS、レジスタ：シード (DxBxxDR2)、バンク 0

ビット	7	6	5	4	3	2	1	0
値	PRS シード / 比較レジスタ							

このレジスタの値は、ほぼすべてのパラメータ値に応じて、API によって制御されます。

VC2 クロック源構成レジスタを持つ CSDADC

Table 69. ブロック CMP、レジスタ：ACE_CONTROL1 (ACE01CR1)、バンク X

ビット	7	6	5	4	3	2	1	0
値	0	1	リファレンス			1	1	1

リファレンスは、同じ名前のユーザ モジュール パラメータで維持されます。

Table 70. ブロック CMP、レジスタ：ACE_CONTROL2 (ACE01CR2)、バンク X

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	Power	

電源は、CSDADC_Start() と CSDADC_Stop() API で制御されます。

Table 71. ブロック CMP_REF、レジスタ：ASE_CONTROL (ASE11CR0)、バンク X

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 72. ブロック CMP_REF、レジスタ：ADC_CONTROL (ADC1_CR), バンク 0

ビット	7	6	5	4	3	2	1	0
値	CMPST	1	1	0	0	オート	0	ADCEN

CMPST は読み出し専用で、ADC 関連 API によって使用されます。オートビットは ADC 関連 API を介して維持されます。ADCEN は、ADC 動作を可能にし、CSDADC API によって維持されます。

Table 73. ブロック CMP_REF、レジスタ：ADC_TRIM (ADC1_TR), バンク 1

ビット	7	6	5	4	3	2	1	0
値	ADCTrimValue							

ADCTrimValue は、CSDADC API 制御です。シングル スロープ ADC 用の ADC トリム値、および CSD 動作用のゼロ値を含みます。

Table 74. ブロック CNT、レジスタ：関数 (DxBxxFN)、バンク 1

ビット	7	6	5	4	3	2	1	0
値	0	0	1	0	0	0	0	1

Table 75. ブロック CNT、レジスタ：入力 (DxBxxIN)、バンク 1

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 76. ブロック CNT、レジスタ：出力 (DxBxxOU)、バンク 1

ビット	7	6	5	4	3	2	1	0
値	0	1	0	0	0	0	0	0

Table 77. ブロック CNT、レジスタ：制御 (DxBxxCR0)、バンク 1

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	Enable

イネーブル ビットは、カウンタ動作を可能にして、CSDADC API によって維持されます。

Table 78. ブロック CNT、レジスタ：期間 (DxBxxDR0)、バンク 0

ビット	7	6	5	4	3	2	1	0
値	CounterValue 直接の読み取りや書き込みは不可							

Table 79. ブロック CNT、レジスタ：期間 (DxBxxDR1)、バンク 0

ビット	7	6	5	4	3	2	1	0
値	1	1	1	1	1	1	1	1

Table 80. ブロック CNT、レジスタ：比較 (DxBxxDR2)、バンク 0

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

バージョン履歴

バージョン	担当者	説明
1.1	DHA	エラーメッセージを更新しました。
1.20	DHA	1. DisableInt マクロの呼び出しをポーリング ループから ISR (PRS16 CSD モード) に移動しました。 2. AnalogComparatorColumn 及び GlobalOutput 行間の接続は、相互接続ビューに表示されます。
1.20.b	DHA	ヘルプファイルをウィザードに追加しました。
1.30	DHA	1. DiplexTable を "AREA UserModules" から "AREA Iit" に移転しました。 2. デフォルト "DiplexTable" パラメータ値を 0 に設定しました x0112。 3. "DiplexUsed" パラメータを追加してコード圧縮を改善しました。
1.40	DHA	1. Imagecraft 最適化をサポートするために、エリア宣言を更新しました。 2. このユーザモジュール データシートで分解能パラメータの記号名を追加しました。 3. ダイ温度測定機能の説明を追加し、SetScanMode() API 関数の説明を追加しました。 4. CSDADC_StartTempMeasurement および CSDADC_GetTemperature の関数を実践するために、CSDACD ユーザモジュールを更新しました。 5. ユーザモジュールウィザードで、スライダと放射スライダの分解能範囲の計算を更新しました。 6. ユーザモジュールウィザードのヘルプを更新しました。スライダ分解能パラメータの最大値 / 最小値の説明を追加しました。

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。このセクションでは、ユーザー 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ついて簡単な解説を掲載しています。

Copyright © 2011-2012 © Cypress Semiconductor Corporation. 本文書に記載される情報は、予告なく変更される場合があります。Cypress Semiconductor Corporation (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 Designer™ 及び Programmable System-on-Chip™ は、Cypress Semiconductor Corp. の商標、PSoC® は同社の登録商標です。本文書で言及するその他全ての商標又は登録商標は各社の所有物です

全てのソース コード (ソフトウェアおよび / 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。