CapSense® Successive Approximation Multifrequency のデータシート CSAMFS V 1.1

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

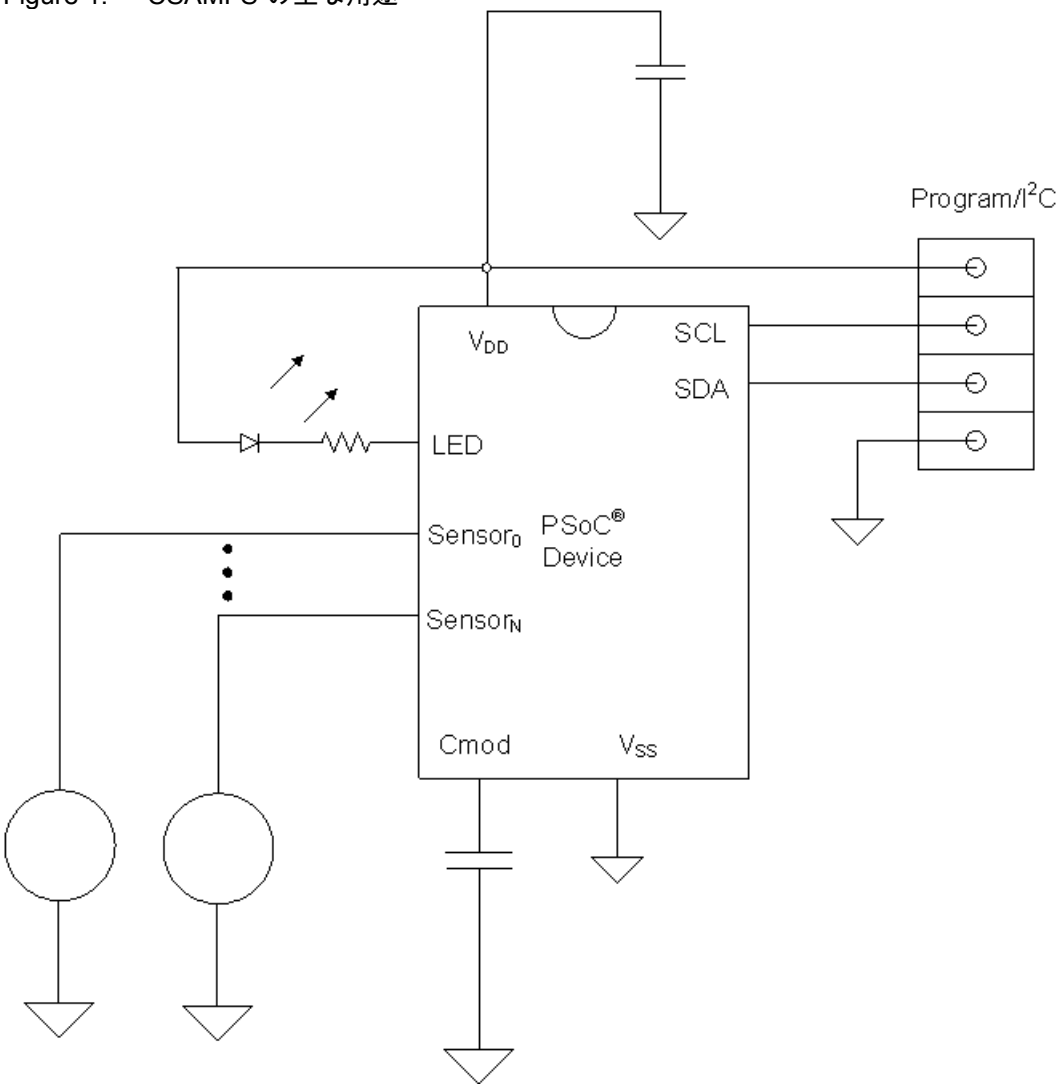
リソース	PSoC® ブロック				API メモリ (バイト) 標準値		ピン (外部入出力ごと)
	I2C/SPI	CapSense	コンパレータ	タイマ	Flash	RAM	
CY8C20x34、CY8C20x24、CY8C20x66、CY8C20x36、CY8C20336AN、CY8C20436AN、CY8C20636AN、CY8C20x46、CY8C20x96、CYONSFN2053、CYONSFN2061、CYONSFN2151、CYONSFN2161、CYONSFN2162	-	1	1	-	1930	90	1

特性および概要

- 1 ~ 28 個の静電容量式センサをスキャン
- 2 ~ 28 個の素子で静電容量式スライダをスキャン
- ダイプレックスを使って物理分解能を倍にするスライダ
- 最高 65535 分の 1 までの補間分解能を持つスライダ
- 複数のスライダセンサを使ってタッチパッドを生成
- 調整可能なセンサ感度、検出閾値、サンプリング速度
- CSAMFS ウィザードを使用したガイダンスによるセンサとピンの割り当て
- 温度変化を処理する、統合された Baseline 値更新アルゴリズム
- さまざまな環境及び物理センサに対する補正

CapSense® Successive Approximation Multifrequency Scanning (CSAMFS) ユーザ モジュールは、スイッチド キャパシタ回路、アナログ マルチプレクサ、デジタル計測機能を用いた静電容量式タッチ センサのアレイ、およびさまざまな環境や物理センサの補正を行う、ハイレベルのソフトウェア ルーチンを実装します。センサのアレイは、個別のセンサ、スライド式センサ、一対の直交するスライド式センサであるタッチパッドを組み合わせることができます。ハイレベルのソフトウェア ルーチンにより、スライダのダイプレックスが可能です。スライダのダイプレックスを使うと、2 つの異なる物理的な位置にある電気センサを、単一のピンで測定できるようになります。ダイプレックスは、I/O を追加せず、スライダの分解能を向上させます。

Figure 1. CSAMFS の主な用途



クイック スタート

1. I2C や LCD など、専用ピンを必要とするユーザ モジュールが使用されている場合、それを選択、配置し、ポートとピンを割り当てます。
2. CSAMFS ユーザ モジュールを選択、配置します。
3. CSAMFS ユーザ モジュールを右クリックし、CSAMFS ウィザードを開きます。
4. ボタンのセンサ数、スライダのコンフィグレーション、ピンの割り当て、関係する設定を行います。
5. ピン及びグローバルユーザ モジュールパラメータを設定します。
6. アプリケーションを生成し、Application Editor を開きます。
7. サンプルコードを変更して、ボタン、スライダ、タッチパッドを実装します。

機能説明

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

CSAMFS ユーザ モジュールによる測定は、グランドと PSoC の各センサーピン間の静電容量となります。導電性物体があると、センサとグランド間の静電容量が増加します。この静電容量の変化が、センサの起動を制御します。

アプリケーション情報

ここでは、CSAMFS ユーザ モジュールを使って、静電容量式検知システムを設計する方法について、手順ごとに説明します。PSoC デバイスには Vdd = 2.7V ～ 5.0V の電圧が供給され、C_{mod} は X7R タイプのコンデンサであると想定しています。

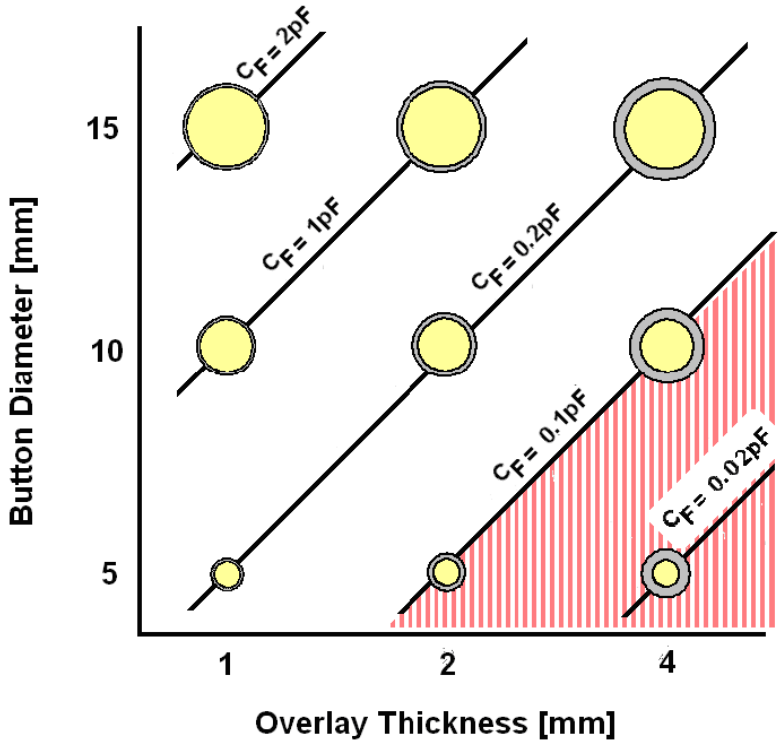
1. **ボタンのサイズ**：図 2 を使用して、特定のオーバーレイの厚さに対して正しいレベルの指の静電容量を生成するボタンのサイズを選択してください。指の最小静電容量 C_F は 0.1 pF ですが、追加設計マージンを考慮し 0.2 pF を推奨します。ボタンとグラウンド フィル間の間隔は、オーバーレイの厚さと同じです。

   オーバーレイと組み合わせたボードの設計は、指の静電容量の最小要件を満たしていますか？

   答えが「いいえ」の場合は、より薄いオーバーレイを使用するか、センサのサイズを大きくしてください。

   答えが「はい」の場合は、手順 2 に進みます。

Figure 2. ボタンの直径とオーバーレイの厚さ



例：プラスチック オーバーレイは、2 mm の厚さを持ちます。設計の目標は、C_F = 0.2 pF です。図 2 は、センサ パッドとグラウンド フィル間の間隔が 2 mm の場合、ボタンの直径が 10 mm 必要であることを示しています。

[+] Feedback

2. **5～50 pF における CP**: CSAMFS ユーザ モジュールでモニタする C_P の範囲を確認します。
   8mil トレースを持つ厚さ 63mil の 2-layer 回路では、トレース静電容量が約 2pF/ インチとなるという経験則を使って、各 PSoC ピンの静電容量負荷を見積もります。
   5 pF ～ 50 pF の C_P 範囲に、すべてのセンサが当てはまりますか？

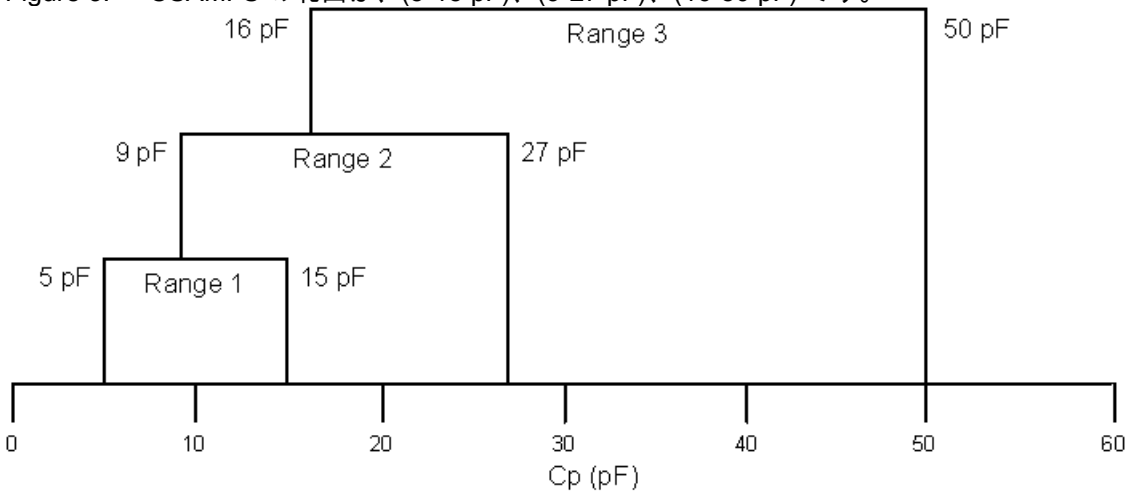
   答えが 「いいえ」の場合は、センサ パッドを PSoC デバイスに近づけるか、CSD など、別の CapSense 手法を使用してみます。24 インチを超えるトレースは、50pF の制限を超えます。

   答えが 「はい」の場合は、手順 3 に進みます。

   例：12 個のタッチ センサを持つシステムがあります。センサ パッドから PSoC ピンまでの距離は、すべてのセンサで 4 インチ未満です。これらのセンサの C_P は、露出した PCB で 9 pF ～ 18 pF の範囲になります。PSoC デバイスのパッケージング効果 (3 pF 追加) を考えると、PSoC ピンの総負荷は 12 pF ～ 21 pF になり、5-50 pF の C_P 制限に容易に収まります。

3. **CP の範囲**：指の検出感度と変換時間は、図 3 に示されているように、C_P の範囲の 1 つで指定されます。範囲は 5-15 pF、9-27 pF、16-50 pF です。
   手順 1 で求めた C_P 値に最もよく合う範囲を見つけて、手順 4 に進みます。

Figure 3. CSAMFS の範囲は、(5-15 pF)、(9-27 pF)、(16-50 pF) です。



例：PSoC ピンの総負荷範囲は、12 pF ～ 21 pF です。これらの値が最もよく合う C_P の範囲は、9 pF ～ 27 pF である範囲 2 です。

[+] Feedback

4. **Difference カウントと閾値**：Difference カウントを見積もり、指閾値とノイズ閾値のパラメータを設定します。Ｄｉｆｆｅｒｅｎｃｅカウントは、指がセンサに触れることで発生するカウントの増加分です。式 1 で、指の検出感度 S_{FINGER} と、指の静電容量 C_F を使ってＤｉｆｆｅｒｅｎｃｅカウントを計算します。指の閾値とノイズの閾値は、式 2 と式 3 を使って求めます (AN2403 を参照)。

Equation 1

$$DifferenceCounts = S_{Finger} \times C_F$$

Equation 2

$$FingerThreshold = 0.75 \times DifferenceCounts$$

Equation 3

$$NoiseThreshold = 0.5 \times DifferenceCounts$$

この 3 つの値を計算し、手順 5 に進みます。

例：システムは、以下のパラメータを持ちます。

C_P = 10 ～ 22 pF、C_F = 0.2 pF

IDACSetting = 7

SettlingTime = 245

C_{mod} = 2700 pF、X7R (+/-20%)

CSAMFS クロック = 6 MHz

IMO = 12 MHz

CPU = 6 MHz

Difference カウント、指の閾値、ノイズの閾値を求めます。

Difference カウント = 500 カウント /pF * 0.2 pF = 100 カウント

指の閾値 = 0.75 * 100 = 75 カウント

ノイズの閾値 = 0.5 * 100 = 50 カウント

5. **スキャン時間**：式 4 で、変換時間 t_C を使って、全センサをスキャンするためのスキャン時間 t_{SCAN} を見積もります。変換時間は、1 センサあたり 900 µs です。

Equation 4

$$t_{SCAN} = t_C \times (numberofsensors)$$

この値を計算し、手順 6 に進みます。

例：手順 4 で定義したシステムには、12 個のセンサがあります。

tSCAN = 900 ms/ センサ * 12 センサ = 10.8 ms

6. **平均供給電流**：式 5 で、アクティブ電流 I_{active}、スリープ電流 I_{sleep}、レポート レートを使って、平均供給電流 I_{DDCS} を見積もります。

Equation 5

$$I_{DDCS} = [t_{SCAN} \times I_{active} + (ReportRate - t_{SCAN}) \times I_{sleep}] / (ReportRate)$$

例：ステップ 5 に続き、以下のパラメータを追加します。

ReportRate = 100 ms

I_active = 1.13 mA

I_sleep = 2.6 mA

平均供給電流 IDDSC を求めます。

I_{DDCS} = [10.8 ms * 1.13 mA + 89.2 ms * 2.6 mA] / 100 ms = 124 mA

7. **直列抵抗**：RF 耐性を得るために直列抵抗器が必要かどうかを判断します。
例：PCB トレースは、銅で 25 mm 以上の長さを持ちます。こうした条件では、すべての CapSense 入力で、560Ω 直列抵抗器を推奨します。

DC 電気的特性と AC 電気的特性

別途指定されている場合を除き、Vdd = 2.7V ～ 5.0V、C_{mod} = X7R タイプのコンデンサ、許容範囲 ±20% です。

Table 1. CSAMFS ユーザ モジュールの電気的仕様

記号	説明	条件	最小値	標準値	最大値	単位
C_P	寄生容量[a]	C_P の範囲に関する注を参照してください。[b]	5		50	pF
C_F	指の静電容量[c]		0.1			pF
N	出力カウンタの分解能		16		16	ビット
S_{FINGER}	指の検出感度[d]	C_P = 5 ～ 15 pF IDAC 設定 = 5 整定時間 = 120 C_{mod} = 1200 pF CSAMFS クロック = 6 MHz IMO = 12 MHz CPU = 6 MHz	500			カウント /pF
		C_P = 9 ～ 27 pF IDAC 設定 = 7 整定時間 = 245 C_{mod} = 2700 pF CSAMFS クロック = 6 MHz、IMO = 12 MHz CPU = 6 MHz	500			カウント /pF

[+] Feedback

記号	説明	条件	最小値	標準値	最大値	単位
		C_P = 16 ～ 50 pF IDAC 設定 = 5 整定時間 = 160 C_{mod} = 5600 pF CSAMFS クロック = 3 MHz IMO = 12 MHz CPU = 3 MHz	500			カウント /pF
t_C	変換時間、1 センサあたり。[d]	C_P = 5 ～ 15 pF IDAC 設定 = 5 セトリング時間 = 120 C_{mod} = 1200 pF CSAMFS クロック = 6MHz IMO = 12 MHz CPU = 6 MHz			700	ms/ センサ
		C_P = 9 ～ 27 pF IDAC 設定 = 7 セトリング時間 = 245 C_{mod} = 2700 pF CSAMFS クロック = 6 MHz IMO = 12 MHz CPU = 6 MHz			900	ms/ センサ
		C_P = 16 ～ 50 pF IDAC 設定 = 5 セトリング時間 = 160 C_{mod} = 5600 pF CSAMFS クロック = 3 MHz IMO = 12 MHz CPU = 3 MHz			2500	ms/ センサ
I_{DDCS}	平均供給電流	Vdd = 3.3V C_P = 5 ～ 15 pF 4 ボタンのスキャン レポート レート 100 ms		35	50	mA
R_S	RF 耐性の直列抵抗器[e] [5]	25 mm 以上の高導電性パターン (銅や銀のインク)。	300		560	オーム

a. C_P には、パッケージに関連する静電容量 2 ～ 3 pF が含まれます。

b. 3 つの C_P 範囲の 1 つで指定される指の検出感度と変換時間：(5-15 pF)、(9-27 pF)、(16-50 pF)

c. 指の静電容量は、指がセンサに触れることで発生する C_P の増分です。

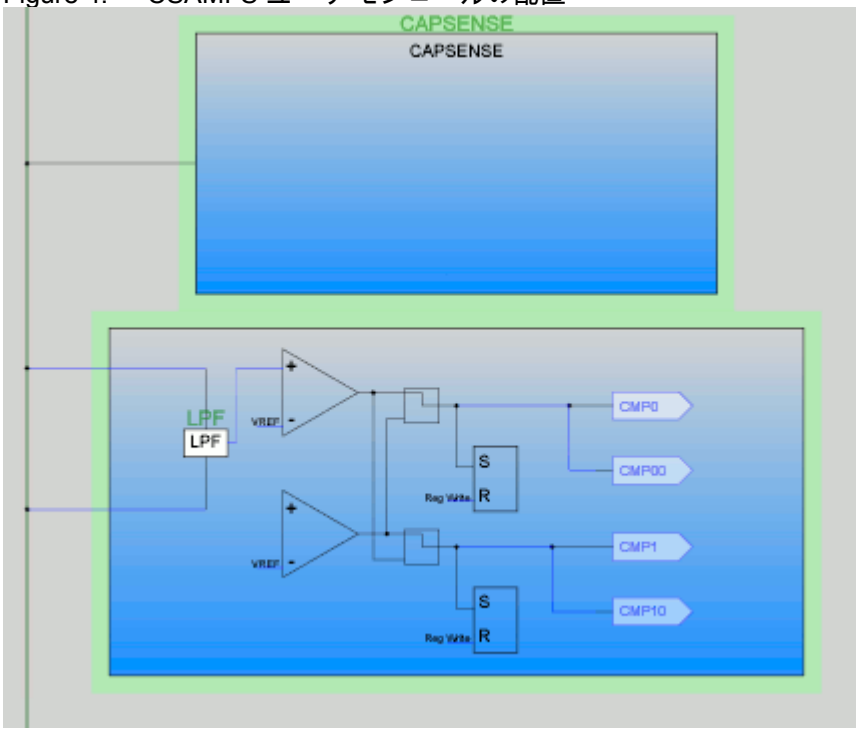
d. 指のタッチが少なくとも 50 カウントの信号を生成するよう、IDAC 設定を選択します。

e. 低導電 ITO 薄型フィルムでは、R_S は必要ありません。抵抗器は、PSoC ピンの 10 mm 以内に配置されます。

配置

ユーザ モジュールのブロックは、UM 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。特定のピンを必要とする LCD や EZI2C などのユーザ モジュールの配置は、CSAMFS ユーザ モジュールを配置する前に行ってください。これらの選択は、ウィザードを開いたときに反映されます。

Figure 4. CSAMFS ユーザ モジュールの配置

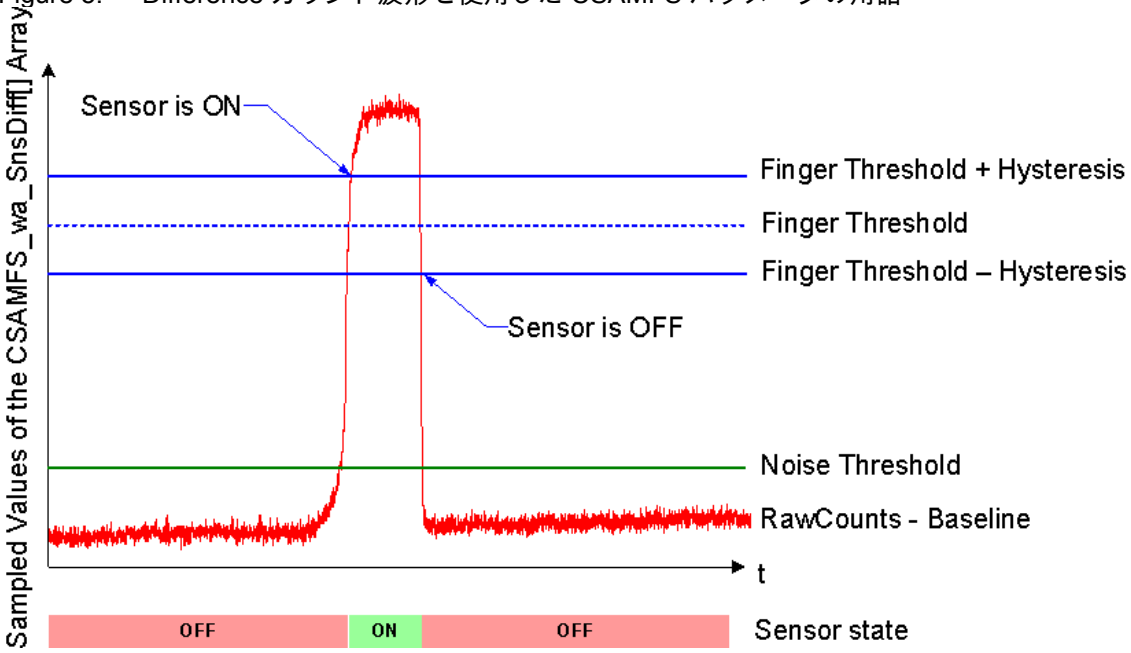


静電容量式センサの接続を配置する場合、P1[0] と P1[1] を使用しないでください。これらのピンは、センサのプログラミングに使用され、センサの検出性能を下げる過度のルーティング静電容量を持っている可能性があります。

パラメータおよびリソース

図 5 に、CSAMFS パラメータで使用される用語の一部を示します。

Figure 5.　Difference カウント波形を使用した CSAMFS パラメータの用語



指の閾値

この閾値は、Difference カウント波形を使って、各ボタン センサの状態を決定するために使用されます。wa_SnsDiff[] アレイに保存されるＤｉｆｆｅｒｅｎｃｅカウントが指の閾値以上である場合は、センサがアクティブであることを意味します。

可能な値の範囲は 3 ～ 255 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Ｂａｓｅｌｉｎｅ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重心の計算に加えられません。

可能な値は 3 ～ 255 です。

Baseline 値更新閾値

新しい Raw カウント値が現在の Baseline 値を上回っており、その差がノイズ閾値を下回る場合は、現在の Baseline 値と Raw カウントの差は「バケツ」内データに加算されます。バケツが満杯になると、Baseline 値が増分され、バケツは空になります。このパラメータは、Ｂａｓｅｌｉｎｅ値が増分するためにバケツが達成しなければならない閾値を設定します。

可能な値は 0 ～ 255 です。

[+] Feedback

セトリング時間

SettlingTime パラメータは、C_{mod} コンデンサの電圧が安定するようなソフトウェアの遅延を制御します。ループは、1 反復あたり 21 の CPU サイクルを持ちます。総遅延は、式 6 で計算されます。

Equation 6

$$Delay(\mu s) = \frac{6 + 21 \cdot (Settling\ Time)}{CPU_Speed(MHz)}$$

少なくとも 5 × R × C となるような SettlingTime を選択します。ここで、R = 1 ÷ (クロック × C_P) および C = C_{mod} です。

可能な値は 2 ～ 255 です。デフォルト値は 20 で、CPU クロック 12 MHz で 35.5 us の遅延となります。

ExternalCap (外部コンデンサ)

ExternalCap パラメータは、C_{mod} コンデンサが接続する PSoC ピンを選択します。C_{mod} の推奨範囲は、1200 pF ～ 5600 pF です。C_{mod} の値は、指の検出感度と変換時間に影響を与えます。詳しくは、「DC 電気的特性と AC 電気的特性」を参照してください。

可能な値は、なし、P0[1]、P0[3] です。

Hysteresis (ヒステリシス)

Hysteresis 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、指の閾値に数値を加減します。センサがオフである場合、Ｄｉｆｆｅｒｅｎｃｅカウントは指の閾値 + ヒステリシスを上回る必要があります。センサがオンの場合、Difference カウントは指の閾値 - 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。

可能な値は 0 ～ 255 です。ただし、FingerThreshold パラメータの設定よりも低くなければなりません。

Debounce (デバウンス)

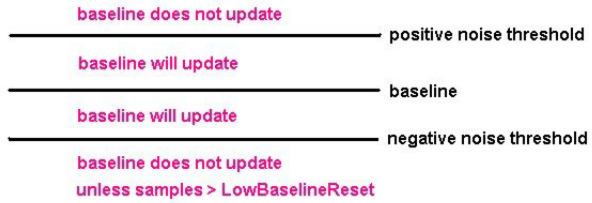
Debounce パラメータは、センサのアクティブ遷移にデバウンス カウンタを追加します。センサが非アクティブからアクティブへ遷移するためには、指定されたサンプル数以上の期間において、Difference カウント値が指の閾値 + ヒステリシスを上回る状態を維持しなければなりません。

可能な値は 1 ～ 255 です。1 に設定するとデバウンスは起こりません。

NegativeNoiseThreshold (負のノイズ閾値)

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

可能な値は 0 ～ 255 です。



LowBaselineReset (低基準値リセット)

LowBaselineReset パラメータは、NegativeNoiseThreshold パラメータと共に作動します。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、Ｂａｓｅｌｉｎｅ値は新しい Raw カウント値に設定されます。これは、Baseline 値のリセットが必要となる異常に低いサンプルの数を数えるもので、起動時に指が置かれている状態で補正を行うために使用されます。

可能な値は 0 ～ 255 です。

Sensors Autoreset (センサの自動リセット)

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) に設定されている場合、Baseline 値は常時更新されます。この設定は、センサがアクティブとなる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) に設定されている場合、Raw カウントと Baseline 値の差がノイズ閾値パラメータを下回る場合にのみ、Baseline 値は更新されます。

可能な値は、Enabled および Disabled です。

Freq Num (頻度)

このパラメータを使うと、各センサを異なるクロックで 3 回スキャンすることで、EMI の性能を高めることができます。Freq Num = 1 は標準のスキャン アルゴリズムを、Freq Num = 3 は先進的なアルゴリズムを表します。先進的なスキャン アルゴリズムを有効にすると (Freq Num = 3)、スキャン時間と RAM の消費量がほぼ 3 倍になります。

可能な値は、1 と 3 です。

Spread Spectrum (スペクトラム拡散)

このパラメータを使うと、スキャン中にクロック値をランダムに変更することで、EMI の性能を高めることができます。Freq Num = 1 の場合に、このパラメータを有効にします。

可能な値は、Disable および Enable です。

RawData メディアンフィルタ

このメディアンフィルタは、センサから最新のサンプル 3 つを調べ、中間値を採用します。これは、短いノイズスパイクを除去するために使用されます。このフィルタは、 1 サンプル分の遅延を発生させます。このフィルタは、遅延の発生および RAM 使用量により、一般的にはお勧めしません。このフィルタを有効にすると、(センサの数 × 2 × Freq Num) バイトの RAM とフラッシュ 100 バイトを消費します。これは、デフォルトでは無効になっています。

RawData IIR フィルタ

この無限インパルス応答 (IIR) フィルタは、変換結果 (生カウント) のノイズを減少させます。Raw カウントをフィルタすると XY 座標をフィルタするよりも効果的ですが、より多くの RAM が必要となります。このフィルタを有効にすると、フラッシュ 100 バイトが余分に消費されます。これは、デフォルトでは無効になっています。

デフォルトの IIR 係数は 0.5 です。

RawData IIR フィルタ係数

これは、生カウント IIR フィルタの係数です。「2」は前 ½ + 現在 ½ を意味します。「4」は前 ¾ + 現在 ¼ を意味します。許可される設定は、2 と 4 のみです。

Clock (クロック)

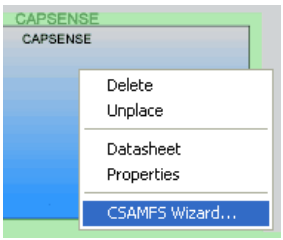
Clock パラメータは、センサの実効抵抗値を上げるために使用されます。センサのエリアが大きい場合は、実効抵抗が高くなりすぎて、スイッチド キャパシタ回路の自動校正ができなくなることがあります。タッチパッドのロー / コラムや大型の近接センサでは、検出感度が落ちる可能性があります。この場合、セトリング電圧がコンパレータの閾値をはるかに下回ります。IMO に大きな分周器を使用すると、実効抵抗が高くなり、高い静電容量が補正されます。

可能な値は、IMO、IMO/2、IMO/4、IMO/8 です。

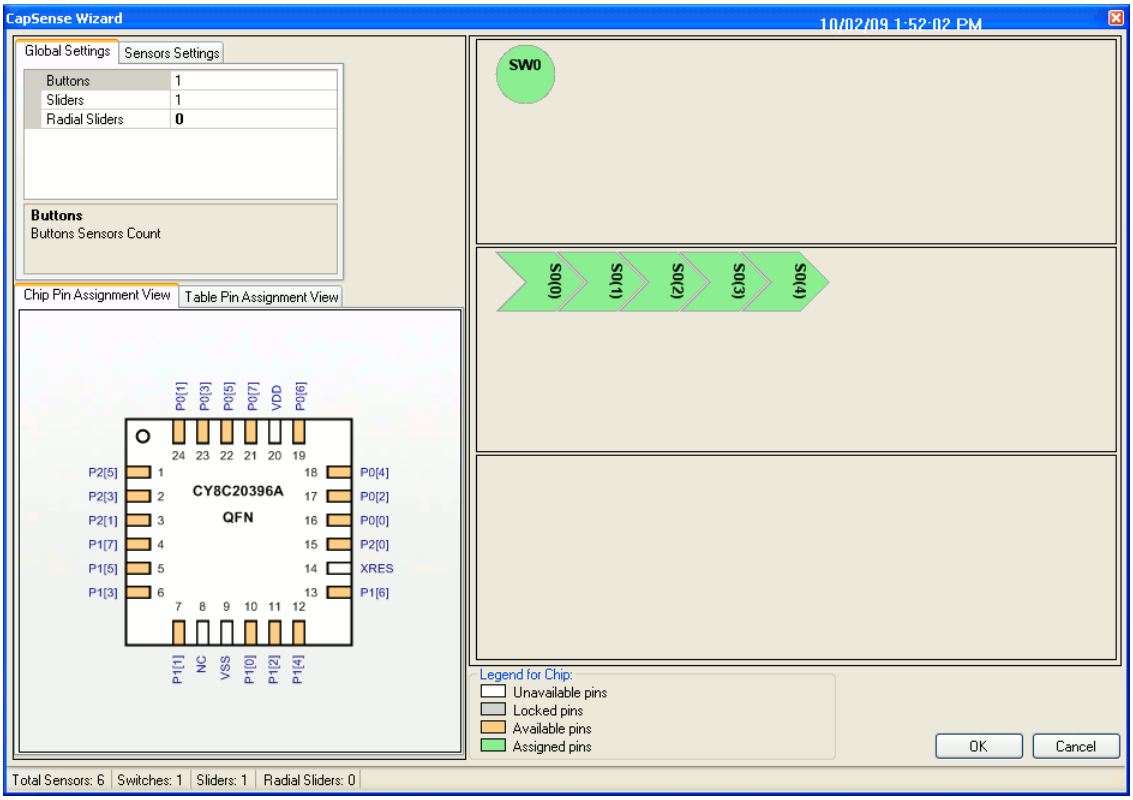
ウィザード

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

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



2. ウィザードが開き、ボタン数、リニア スライダ数、ラジアル スライダ数の入力ボックスが表示されます。



ウィザードのピン凡例

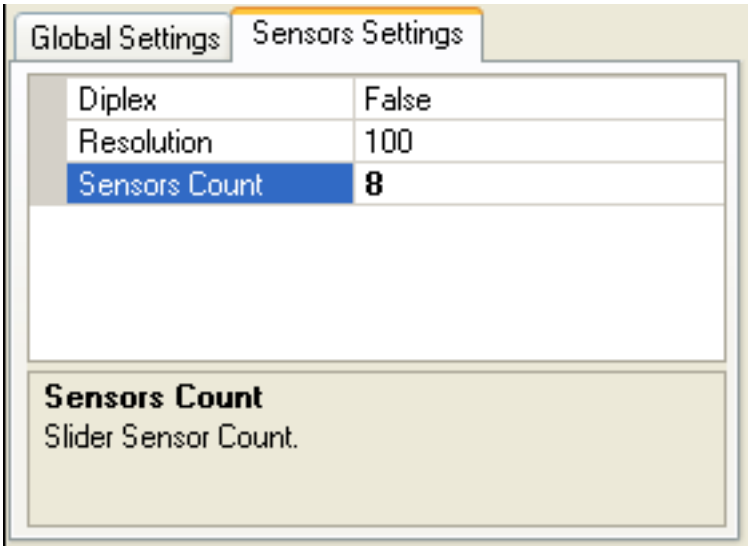
白 – このピンは CapSense の入力に使用できません。

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] (ピン配列) ビューでそのピンを表示上エクスパンドし、[**Select**] (選択) メニューで [**Default**] (デフォルト) 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。

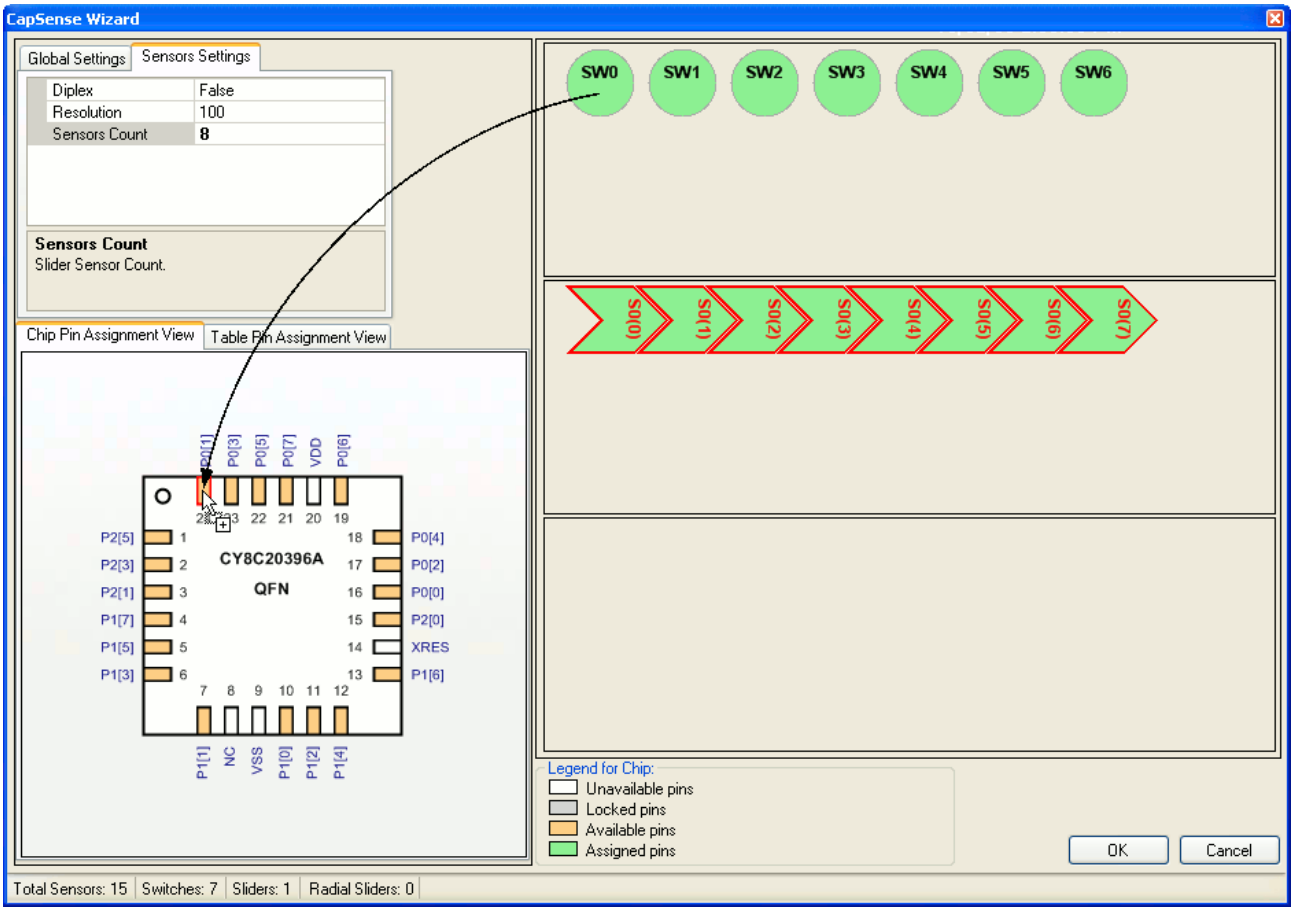
グリーン – このピンは CapSense 入力に使用されています。

3. ボタン数を入力します。センサ数は、使用できるピン数に制限されています。[Enter] キーを押して、センサ数の新しい値を入力します。
4. リニア スライダとラジアル 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。
5. スライダをクリックすると、そのスライダに対してセンサ設定が有効になります。各スライダのセンサコンポーネント数を入力します。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] キーを押して、新しい値を入力します。

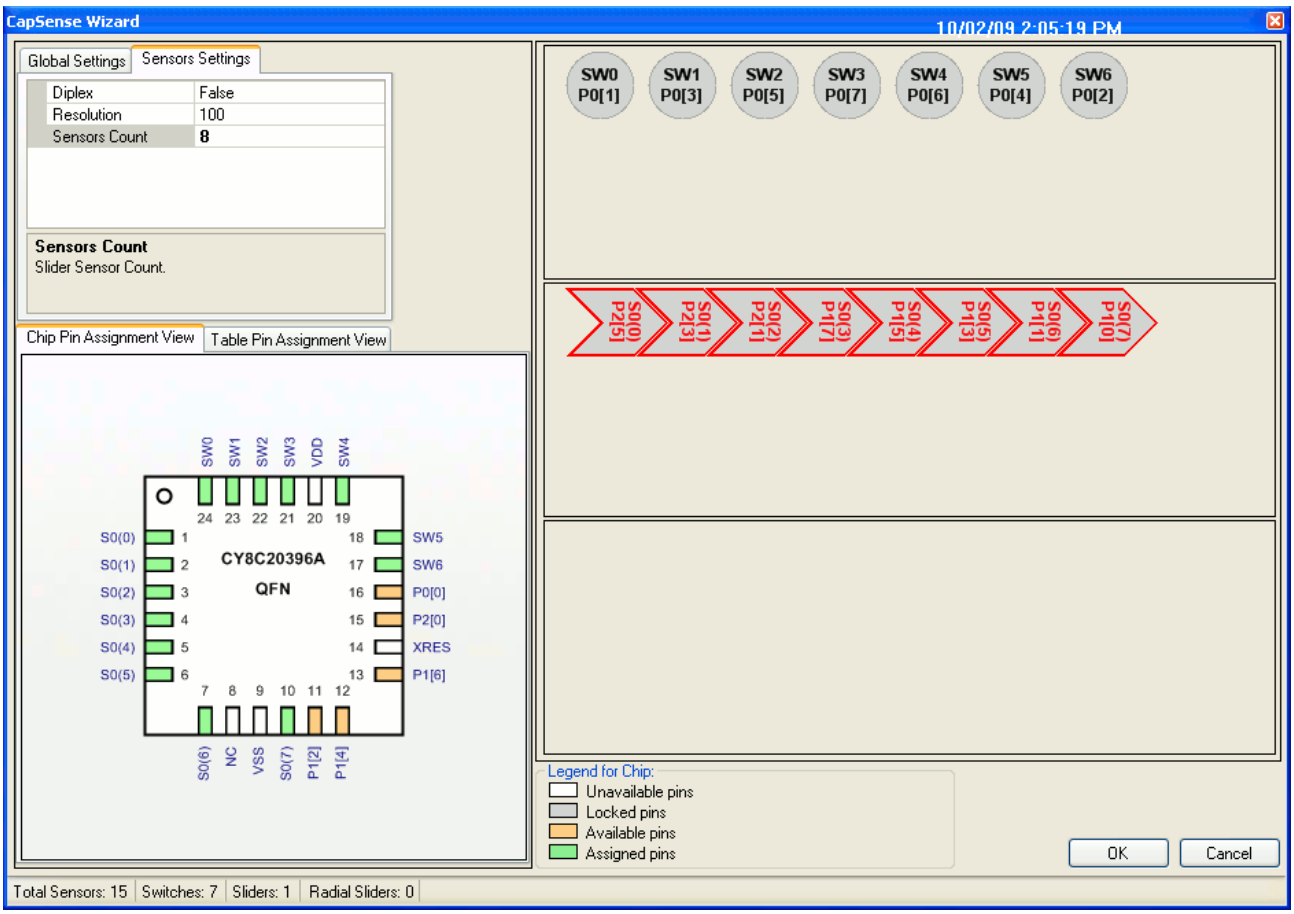


6. 出力の分解能を入力します。最低値は 5 です。最大値は、(センサに使用されるピン数 – 1) x 2^{16} – 1 、ダイプレックス スライダの場合、(2 x センサに使用されるピン数 – 1) x 2^{16} -1 です。
7. 必要に応じて、ダイプレックスを選択します。これにより、センサ用に選択されたピンを、基板上で 2 倍の数のセンサ位置にマッピングできます。ダイプレックス センサの最初の半分のみが表示されます。AN2292 で、ピン接続のダイプレックス表を参照してください。

8. [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、ピンに割り当てることができます。また、[Chip Pin Assignment View] (チップのピン割り当てビュー) や [Table Pin Assignment View] (ピン割り当て表ビュー) 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 ピンを選択するとグリーンになり、使用不可となります。ポート ピンからセンサをドラッグして外すと、センサ割り当てを変更できます。



9. 他のセンサでも同じ操作を繰り返します。



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。ここで適宜、サンプルプロジェクトを採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した。

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 CSAMFS_Table.asm に保存されています。

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) ルーチンは、実装の細部に依存することなく、ユーザのコードからユーザ モジュールを操作できるよう、ユーザ モジュールのコンポーネントとして提供されています。このセクションではインタフェースの各関数を、include ファイルに用意されている関連定数とともに示します。

Note ここではすべてのユーザ モジュール API と同様に、API 機能を呼び出して、A および X レジスタの値が変更されます。関数を呼び出し、呼び出し後に A と X の値が必要になる場合は、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、実行されています。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の存在を認識する必要があります。ユーザ モジュール API 関数の中には、A と X を変更しないものもありますが、今後変更されないという保証はありません。

CSAMFS ユーザ モジュールを初期化、開始、停止するエントリ ポイントが提供されています。いずれの場合でも、次のエントリ ポイントにおいてモジュールのインスタンス名が、CSAMFS 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。

ソフトウェアの制御パラメータ

API に渡される制御パラメータは、以下の通りです。

bSnsGroup

スライダとして使用される、あるセンサ グループへの参照です。更新するセンサ グループを選択するために CSAMFS_bGetCentroidPos が使用します。

ボタンはグループ 0 に含まれています。スライダはグループ 1 から上に含まれています。

bSensor

CSAMFS_wGetPortPin が使用するセンサ番号で、選択された動作中のセンサに対して、ポートとビットマスク (bPort および bMask) を決定します。CSAMFS_wGetPortPin は bMask と bPort を返します。この 2 つの返り値は、CSAMFS_EnableSensor と CSAMFS_DisableSensor でセンサを選択する際に使用されます。また、CSAMFS_wReadSensor でどのセンサのカウントが返されるかを設定する際にも使用します。

CSAMFS データアレイ

API 関数は、複数のグローバルアレイを使用します。これらのアレイを手動で変更しないでください。デバックの目的でこれらの値を検証することは可能です。

CSAMFS_waSnsResult

このアレイは、各センサに対する 16-bit のR a wカウント値を保持します。配列は、[Freq Num][センサの数] です。

CSAMFS_waIIR

このアレイは、IIR フィルタが有効な場合に、各センサに対する 16-bit のフィルタされた Raw カウント値を保持します。配列は、[Freq Num][センサの数] です。

CSAMFS_waSnsBaseline

このアレイは、各センサで 16-bit の Baseline 値を保持します。配列は、[頻度][センサの数] です。

CSAMFS_waSnsDiff

このアレイは、各センサで 16-bit の Difference カウント値を保持します (Raw カウント - Ｂａｓｅｌｉｎｅ値)。

CSAMFS_baSnsOnMask

この 8-bit アレイは、ボタンまたはスライダに対して、センサのオンまたはオフのデータを保持します。このアレイの各エレメントは、8 つまでのセンサの状態を保持します。CSAMFS_baSnsOnMask[0] には、センサ 0 からセンサ 7 までに対するマスクされたビットが含まれます (センサ 0 はビット 0、センサ 1 はビット 1 など)。CSAMFS_baSnsOnMask[1] には、センサ 8 からセンサ 15 までのマスクされたビットが含まれ、必要に応じてそれ以降も続きます。このバイトアレイには、配置されている全センサのエレメントが含まれます。センサがオンの場合、ビットの値は 1、センサがオフの場合は 0 です。

CSAMFS_baDACCodeScan

このアレイは、Cmod の ADC リニアランプ電圧の傾斜を設定する、8-bit IDACSetting パラメータ値を保持します。このパラメータは、CSAMFS_Start 呼び出しの直後 (Ｂａｓｅｌｉｎｅ値の初期化前またはセンサのスキャン前) にアレイがロードされた後、各センサに対して個別に設定できます。

CSAMFS_baDACCodeBaseline

このアレイは、CSAMFS_Start. で自動的に決定される、各センサの 8-bit 校正値を保持します。アレイ内の値は、各 CapSense 入力をロードする寄生容量の相対的な値を示します。

基本的な API

基本的な API は、ユーザ モジュールを開始、停止する際に使用します。

CSAMFS_Start()

説明

CSAMFS ユーザ モジュールを各センサで校正します。また、すべてのセンサ傾斜 IDAC 値の共通値を校正します。すべてのセンサ ピンをアナログ多重化バスから切り離します。すべてのセンサピンがグラウンドに短絡されます。C_{mod} コンデンサをシステムに接続します。

C プロトタイプ

```
void  CSAMFS_Start
```

アセンブリ

```
lcall  CSAMFS_Start
```

パラメータ

なし

戻り値

なし

副作用

**

CSAMFS_Stop

説明

CapSense ブロックを無効にします。すべてのセンサ ピンをアナログ多重化バスから切り離し、グラウンドに短絡させる CSAMFS_ClearSensors を呼び出します。

C プロトタイプ

```
void  CSAMFS_Stop()
```

アセンブリ

```
lcall  CSAMFS_Stop
```

パラメータ

なし

戻り値

なし

副作用

**

CSAMFS_Calibrate

説明

すべてのセンサの傾斜 IDAC 値で共通値を校正します。この関数は、CSAMFS_Start() が呼び出された場合に実行されます。センサを再校正する際、この関数をいつでも呼び出すことができます。この関数が iDAC 電流を調整し、可能な限り wLevel に近いR a wカウントになるようにします。

C プロトタイプ

```
void  CSAMFS_Calibrate(WORD wLevel)
```

アセンブリ

```
mov  A, <wLevel
mov X, >wLevel
lcall  CSAMFS_Calibrate
```

パラメータ

wlevel - 要求 Raw データ値

戻り値

なし

データ表

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、ウィザードが一連のデータ表を生成します。センサ表は CSAMFS_table.asm に保存されています。グループおよびダイプレックス表は CSAMFSHL.asm に保存されています。

CSAMFS_Sensor_Table

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

[+] Feedback

の独立したセンサ、次に各スライダ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。

```
CSAMFS_Sensor_Table:
_CSAMFS_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
```

CSAMFS_Group_Table

グループ表は、センサやスライダセンサのグループを定義します。表の各行における 1 番目の値は、グループの開始センサ番号です。2 番目の値は、グループ内のセンサ数です。ダイプレックスでない場合は、3 番目の値が 0 になります。4 番目、5 番目、6 番目の値を組み合わせて固定小数点乗数値を生成し、重心の計算用に使用します。7 つのセンサの例を以下に示します。

```
CSAMFS_Group_Table:
CSAMFS_Group_Table:
// Group Table
//        Origin Count  Diplex?  SliceMultiplier
      db    0,    0x7,    0x0,    0x00     // Buttons
```

個別のセンサやスライダセンサを使うプロジェクトでは、個別センサと個々のスライダセンサのセットが、下に表示されているように、グループ表で別々のエントリを持ちます。

```
CSAMFS_Group_Table:
CSAMFS_Group_Table:
; Group Table
;         Origin Count  Diplex?  DivBtwSns(wholeMSB, wholeLSB, fractByte)
      db    0,    0x7,    0x0,    0x00,    0x00,    0x00 ; Buttons
      db    0x6,  0xA,    0x4,    0x0,     0x7,     0xE5 ; Slider 1
```

CSAMFS_Diplex_Table

ダイプレックス表は、各スライダセンサの完全なセンサレンジのマッピングを定義します。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。10 個のセンサスライダを使った典型的な例を以下に示します。

```
DiplexTable_0:
; This group is not a diplexed slider

DiplexTable_1:
    db    0,1,2,3,4,5,6,7,8.9,0,3,6,9,1,4,7,2,5,8   // 10 switch slider

CSAMFS_Diplex_Table:
_CSAMFS_Diplex_Table:
    db >DiplexTable_0, <DiplexTable_0
    db >DiplexTable_1, <DiplexTable_1
```

ローレベル API

センサデータの取得には、ローレベル API が使用されます。

CSAMFS_ScanSensor

説明

1 個のセンサをスキャンし、静電容量を表す Raw カウント値を決定します。このルーチンは、CSAMFS_Start 関数が実行前に呼び出されたことを想定しています。ルーチンは、CapSense ブロック割り込み CSAMFS_ISR が完了するのを待つ、ブロッキング呼び出しです。次に、16-bit カウンタから CSAMFS_wADCResult グローバル変数に値を転送します。

C プロトタイプ

```
void  CSAMFS_ScanSensor(BYTE bSensor)
```

アセンブリ

```
mov  A,  bSensor
lcall CSAMFS_ScanSensor
```

パラメータ

bSensor : 範囲は 0 ~ n–1 です。ここで n は、CSAMFS 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。

戻り値

なし

副作用

**

CSAMFS_ScanAllSensors

説明

各センサインデックスに対して CSAMFS_ScanSensor を呼び出して、構成済み全センサをスキャンします。次に、Raw データと更新されたすべての Baseline 値に有効なフィルタをすべて適用します。

C プロトタイプ

```
void  CSAMFS_ScanAllSensors()
```

アセンブリ

```
lcall  CSAMFS_ScanAllSensors
```

パラメータ

なし

戻り値

なし

副作用

**

CSAMFS_ClearSensors

説明

各センサの CSAMFS_DisableSensor。センサピンをアナログ多重化バスから切り離し、グラウンドに短絡させます。

C プロトタイプ

```
void  CSAMFS_ClearSensors()
```

アセンブリ

```
lcall  CSAMFS_ClearSensors
```

パラメータ

なし

戻り値

なし

副作用

**

CSAMFS_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が、インデックスとなり、CSAMFS_Sensor_Table. からのデータを選択します。

C プロトタイプ

```
WORD  CSAMFS_wGetPortPin(BYTE bSensor)
```

アセンブリ

```
mov  A,  bSensor
lcall     CSAMFS_wGetPortPin
```

パラメータ

bSensor : 範囲は 0 ~ n–1 です。ここで n は、CSAMFS 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。

戻り値

bPort および bMask : 特定のセンサ選択を決定するために使用されるポート番号とビットマスク。

副作用

**

CSAMFS_EnableSensor

説明

次の測定サイクルでスキャンするために選択されたセンサを構成します。センサのポート番号とピンは、CSAMFS_wGetPortPin ルーチンで選択され、X と A にそれぞれロードされたポート番号とビットマスクが入っています。選択されたポートとピンを Analog High Z (アナログ High Z) モードにし、正しい Analog Mux Bus (アナログ多重化バス) 入力を可能にします。

C プロトタイプ

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

アセンブリ

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

パラメータ

bPort および bMask：特定のセンサ選択を決定するために使用されるポート番号とビットマスク。

戻り値

なし

副作用

**

CSAMFS_DisableSensor

説明

センサを切り離すため、入力が行われなくなります。典型的に、この呼び出しは、CSAMFS_wGetPortPin. と組み合わせて使用されます。ポートピンからアナログ多重バスへの接続がオフになります。駆動モードは、Strong (01) に変更され、データレジスタビットがゼロにセットされます。これにより、センサがグランド接続されます。

C プロトタイプ

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

アセンブリ

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

パラメータ

bPort および bMask：特定のセンサを選択するためのポート番号とビットマスク。

戻り値

なし

副作用

**

ハイレベル API

ハイレベル API は、ローレベル API によって取得されたセンサデータの処理に使用されます。

CSAMFS_UpdateSensorBaseline

説明

1 個のセンサのＢａｓｅｌｉｎｅ値を更新します。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. CSAMFS_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 カウントを計算します。この Difference 値は waSnsDiff アレイに保存されます。
2. CSAMFS_UpdateSensorBaseline が呼び出されるたびに、Ｄｉｆｆｅｒｅｎｃｅカウントとノイズの閾値が比較されます。Difference 値がノイズ閾値より小さい場合は、Ｄｉｆｆｅｒｅｎｃｅ値の半分が仮想バケツに追加 / 累積されます。Ｄｉｆｆｅｒｅｎｃｅ値がノイズ閾値より大きい場合、Ｂａｓｅｌｉｎｅ値は更新されません。
3. 仮想バケツで集積されたＤｉｆｆｅｒｅｎｃｅカウントが BaselineUpdateThreshold に達すると、Ｂａｓｅｌｉｎｅ値は増分され、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は、waSnsDiff アレイに保持されている値が 0 にリセットされます。

この関数は、CSAMFS_ScanAllSensors() から自動的に呼び出されます。

C プロトタイプ

```
void CSAMFS_UpdateSensorBaseline(BYTE bSensor)
```

アセンブリ

```
mov   A, bSensor
lcall  CSAMFS_UpdateSensorBaseline
```

パラメータ :

bSensor : 範囲は 0 ~ n–1 です。ここで n は、CSAMFS 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。

戻り値

なし

副作用

**

CSAMFS_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が考慮されます。ヒステリシス値は、センサの状態により、指閾値に足したり、指閾値から引いたりされます。アクティブ中の場合、閾値は下げられます。アクティブ中でない場合、閾値は上げられます。また、この関数は、CSAMFS_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  CSAMFS_bIsSensorActive(BYTE bSensor)
```

アセンブリ

```
mov  A,  bSensor
lcall CSAMFS_bIsSensorActive
```

パラメータ

bSensor : 範囲は 0 ~ n–1 です。ここで n は、CSAMFS 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。

戻り値

戻り値は、アクティブ中の場合 1 で、アクティブ中でない場合は 0 です。

副作用

**

CSAMFS_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 CSAMFS_bIsSensorActive を呼び出し、CSAMFS_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  CSAMFS_bIsAnySensorActive()
```

アセンブリ

```
lcall CSAMFS_bIsAnySensorActive
```

パラメータ

なし

戻り値

戻り値は、アクティブ中の場合 1 で、アクティブ中でない場合は 0 です。

副作用

**

CSAMFS_SetDefaultFingerThresholds

説明

CSAMFS_baBtnFThreshold アレイを FingerThreshold パラメータの値でロードします。CSAMFS_baBtnFThreshold アレイが手動でカスタム値をロードしていなければ、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

```
BYTE  CSAMFS_SetDefaultFingerThresholds()
```

アセンブリ

```
lcall CSAMFS_SetDefaultFingerThresholds
```

パラメータ

なし

戻り値

なし

副作用

**

CSAMFS_InitializeBaselines

説明

それぞれのセンサをスキャンして、CSAMFS_waSnsBaseline アレイに初期値をロードします。

C プロトタイプ

```
void  CSAMFS_InitializeBaselines()
```

アセンブリ

```
lcall CSAMFS_InitializeBaselines
```

パラメータ

なし

戻り値

なし

副作用

**

CSAMFS_InitializeSensorBaseline

説明

特定のセンサに対して、スキャンし、CSAMFS_waSnsBaseline アレイに初期値をロードします。個々のセンサのＢａｓｅｌｉｎｅ値をリセットするために使用されます。

C プロトタイプ

```
void  CSAMFS_InitializeSensorBaseline(BYTE bSensor)
```

アセンブリ

```
lcall CSAMFS_InitializeSensorBaseline
```

パラメータ

bSensor : 範囲は 0 ~ n–1 です。ここで n は、CSAMFS 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。

戻り値

なし

副作用

**

CSAMFS_wGetCentroidPos

説明

重心計算のため、Ｄｉｆｆｅｒｅｎｃｅアレイを確認します。1 が存在する場合、オフセットと長さが一時変数に保存され、重心位置が CSAMFS ウィザードで指定された分解能で計算されます。

C プロトタイプ

```
WORD  CSAMFS_wGetCentroidPos(BYTE bSnsGroup)
```

アセンブリ

```
mov  A,  bSnsGroup
lcall CSAMFS_wGetCentroidPos
```

パラメータ

bSnsGroup：スライダとして使用される特定のセンサグループへのレファレンスです。

戻り値

スライダ位置の値は LSB は A に、ＭＳＢは X に置かれます。

副作用

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

注意事項：スライダセグメント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。ノイズにより正しくない重心結果を生じさせないように、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)。

ファームウェア ソースコードの例

ボタンのスキャンと LED のオンとオフの切り替え

このコードは、CSAMFS UCC ボード (CY3280-20x34) とリニアスライダ モジュールボード (CY3280-SLM) 用に記述されています。

```
//------ Sample code for CSAMFS buttons that LEDs On and Off   ----------
//------     pin assignments for Linear Slider Module plugged   ---------
//-------     into CSAMFS UCC board, CY3280-20x43+SLM            --------

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

void main(void)
{
//initialize LED states
  PRT0DR |= 0b00100010;  //turn-off LED on P0[5],P0[1]
  PRT1DR |= 0b00000100;  //turn-off LED on P1[2]
  PRT2DR |= 0b10100000;  //turn-off LED on P2[7],P2[5]

//Set port drive modes for LEDs
  PRT0DM0 |= 0b00100010;  //strong on P0[5],P0[1]
  PRT0DM1 &=~0b00100010;

  PRT1DM0 |= 0b10100100;  //strong on P1[2]
  PRT1DM1 &=~0b10100100;

  PRT2DM0 |= 0b10100000;  //strong on P2[5],P2[7]
  PRT2DM1 &=~0b10100000;

  M8C_EnableGInt;  //enable global interrupts for use with CSAMFS

  CSAMFS_Start();     //initialize the CSAMFS User Module
  CSAMFS_SetDefaultFingerThresholds();   //Load finger thresholds
  CSAMFS_InitializeBaselines();  //Set baselines to current count

  while(1)  //infinite loop scanning buttons
  {
    CSAMFS_ScanAllSensors();    //sample all buttons and compute baselines

// control the LEDs using the sensor states.
// LED ON if active, OFF if not active.
// Check buttons in sequence.
    if (CSAMFS_bIsSensorActive(0))
    {
      PRT1DR &= ~0b00000100;   //turn-on LED on P1[2]
    }
    else
    {
      PRT1DR |= 0b00000100;  //turn-off LED on P1[2]
    }
    if (CSAMFS_bIsSensorActive(1))
    {
      PRT0DR &= ~0b00100000;  //turn-on LED on P0[5]
```

```
    }
    else
    {
      PRT0DR |= 0b00100000;  //turn-off LED on P0[5]
    }
    if (CSAMFS_bIsSensorActive(2))
    {
      PRT0DR &= ~0b00000010;  //turn-on LED on P0[1]
    }
    else
    {
      PRT0DR |= 0b00000010;  //turn-off LED on P0[1]
    }
    if (CSAMFS_bIsSensorActive(3))
    {
      PRT2DR &= ~0b10000000;  //turn-on LED on P2[7]
    }
    else
    {
      PRT2DR |= 0b10000000;  //turn-off LED on P2[7]
    }
    if (CSAMFS_bIsSensorActive(4))
    {
      PRT2DR &= ~0b00100000;  //turn-on LED on P2[5]
    }
    else
    {
      PRT2DR |= 0b00100000;  //turn-off LED on P2[5]
    }
  }
}
```

リニアスライダを使って LED 強度の制御

このコードは、CSAMFS UCC ボード (CY3280-20x34) とリニアスライダ モジュールボード (CY3280-SLM) 用に記述されています。

```
//------ Sample code for CSAMFS slider controlling LED intensity -------
//------   pin assignments for Linear Slider Module plugged   ----------
//-------  into CSAMFS UCC board, CY3280-20x43+SLM               -------

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

int wCentroid = 0;  //estimated finger position; 0xffff for no finger
int wPos = 0;  //estimated finger position
int wLED_PWM; //controls LED intensity

void main(void)
{
//initialize LED states
  PRT0DR |= 0b00100010;  //turn-off LED on P0[5],P0[1]
  PRT1DR |= 0b00000100;  //turn-off LED on P1[2]
  PRT2DR |= 0b10100000;  //turn-off LED on P2[7],P2[5]
```

```
//Set port drive modes for LEDs
  PRT0DM0 |= 0b00100010;  //strong on P0[5],P0[1]
  PRT0DM1 &=~0b00100010;

  PRT1DM0 |= 0b10100100;  //strong on P1[2]
  PRT1DM1 &=~0b10100100;

  PRT2DM0 |= 0b10100000;  //strong on P2[5],P2[7]
  PRT2DM1 &=~0b10100000;

  M8C_EnableGInt;  //enable global interrupts for use with CSAMFS

  CSAMFS_Start();     //initialize the CSAMFS User Module
  CSAMFS_SetDefaultFingerThresholds();   //Load finger thresholds
  CSAMFS_InitializeBaselines();  //Set baselines to current count

  while(1)  //infinite loop scanning slider
  {
    CSAMFS_ScanAllSensors(); //sample all sensorsand and compute baselines

    wCentroid = CSAMFS_wGetCentroidPos(1); //estimated position
    if (wCentroid != 0xffff)  //0xffff means finger off slider
    {
      wPos = wCentroid;  //get position, range is 0 to 100
    }

    if (wPos > 0)  //if position>0, then pulse all LEDs ON
    {
      PRT1DR &= ~0b00000100;  //turn-on LED on P1[2]
      PRT0DR &= ~0b00100000;  //turn-on LED on P0[5]
      PRT0DR &= ~0b00000010;  //turn-on LED on P0[1]
      PRT2DR &= ~0b10000000;  //turn-on LED on P2[7]
      PRT2DR &= ~0b00100000;  //turn-on LED on P2[5]

      for (wLED_PWM = 0; wLED_PWM < wPos*wPos/100; wLED_PWM++)
      {   //control LED pulse width by position^2
      }  //this control function looks nice

    }
    // LED pulse ON is over for this period, turn all off
    PRT1DR |= 0b00000100;  //turn-off LED on P1[2]
    PRT0DR |= 0b00100000;  //turn-off LED on P0[5]
    PRT0DR |= 0b00000010;  //turn-off LED on P0[1]
    PRT2DR |= 0b10000000;  //turn-off LED on P2[7]
    PRT2DR |= 0b00100000;  //turn-off LED on P2[5]


  }  //do next scan (while loop)
}
```

その他の資料

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

バージョン ヒストリー

バージョン	著者	説明
1.1	DHA	ラジアルスライダの機能がユーザ モジュールと構成ウィザードに追加されました。
1.1.a	DHA	インライン関数がライブラリに移動されました。

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概要を掲載しています。

Copyright © 2009-2011 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

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

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

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

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