Preliminary

セグメント LCD ドライバのデータシート SLCD V 1.10

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

リソース	PSoC® ブロック			API メモリ		ピン (外部 I/O あたり)
	デジタル	アナログ CT	アナログ SC	Flash*	RAM*	
CY8C24x94、CY8C28xxx						
外部 1/2 バイアス信号発生器とタイマー 16 HW を備えたアナログ MUX バスを使用する LCD ドライブ	2	0	0	747** 758*** 753****	20** 21*** 22****	x
外部 1/2 バイアス信号発生器とタイマー 16 HW/SW を備えたアナログ MUX バスを使用する LCD ドライブ	1	0	0	773** 784*** 779****	22** 23*** 24****	x
内部 Vdd/2 バイアス信号発生器とタイマー 16 HW を備えたアナログ MUX バスを使用する LCD ドライブ	2	1	0	753** 764*** 759****	20** 21*** 22****	x
内部 Vdd/2 バイアス信号発生器とタイマー 16 HW/SW を備えたアナログ MUX バスを使用する LCD ドライブ	1	1	0	779** 790*** 785****	22** 32 24	x
CY8C21x34、CY8C21345、CY8C22x45						
外部 1/2 バイアス信号発生器とタイマー 16 HW を備え、アナログ多重化バスを使用する LCD ドライブ	2	0	0	74 .75 758***	20** 21*** 22****	x

リソース	PSoC® ブロック			API メモリ		ピン (外部 I/O あたり)
	デジタル	アナログ CT	アナログ SC	Flash*	RAM*	
外部 1/2 バイアス信号発生器とタイマー 16 HW/SW を備えたアナログ MUX バスを使用する LCD ドライブ	1	0	0	770 78 773**	22** 32 24	x
CY8C20x34、CY8C20xx6、CY8C20336AN、CY8C20436AN、CY8C20636AN、CY8C21x23、CY8C21x34、CY8C21345、CY8C22x45、CY8C23x33、CY8C24x23、CY8C24x94、CY8C27x43、CY8C28xxx、CY8C29x66						
スタンドアロンの GPIO および タイマー 16 HW を使用する LCD ドライブ	2	0	0	692** 703*** 698****	19** 20*** 21****	x
スタンドアロンの GPIO および タイマー 16 HW/SW を使用する LCD ドライブ	1	0	0	718** 729*** 724****	21** 22*** 23****	x

注

* Flash と RAM のサイズは、LCD の構成、コモン、ポート、桁、ディスプレイの数によって異なります。表には、使用されているセグメントラインが 1 で、ディスプレイのない LCD での Flash と RAM が示されています。

以下の式で、Flash と RAM の合計サイズを計算します。

RAM = ベース RAM + 1 + 5*(ポート – 1)

グループサイズ = 0 :

Flash = ベース ROM + セグメント数 * 桁数 + GA*(ポート – 1) + 2*(ディスプレイ - 1) + DT

グループサイズ > 0 :

Flash = ベース ROM + 桁 + GA*(ポート – 1) + 2*(ディスプレイ - 1) + DT

ここで

ベース ROM および ベース RAM - 上の表の API メモリからの Flash および RAM のサイズ (使用されているセグメントラインが 1 で、ディスプレイのない LCD) :

セグメントの数は、7、14、または 16 :

桁 - 全桁数 :

GA = 62 (アナログ MUX バスを使用する LCD ドライブ)、GA = 55 (スタンドアロンの GPIO を使用する LCD ドライブ) :

ポート - セグメントのポート数。

ディスプレイ - ディスプレイの数。

DT は、この表から定義されます :

ディスプレイのタイプ	7 セグメント	14 セグメント	16 セグメント
グループサイズ = 0	270	520	515
グループサイズ > 0	270	526	528

** コモンの数 = 2。

*** コモンの数 = 3。

**** コモンの数 = 4。

特性および概要

- マルチプレックス - 1/2 バイアス対応
- 2、3、4 のコモン LCD ドライブ
- 30-150 Hz の更新レート
- アナログ MUX バスを使用する LCD ドライブ技法
- 外部または内部的に生成される 1/2 バイアスのオプション
- タイプ A 波形のみに対応
- コントラスト制御

セグメント LCD ドライバ (SLCD) ユーザ モジュールは、外部コンポーネントを使わずに、LCD パネルを直接駆動します。

モジュールは、ウィザードから選択できる 2 つの LCD ドライブ技法を備えています。

1. アナログ MUX バスの使用 (アナログ MUX バスを備えたデバイスのみ)

2. スタンドアロン GPIO の使用 (すべての PSoC デバイスに適用可)

アナログ MUX バスを使用する LCD ドライブ技法は、内部および外部バイアスをサポートします。内部バスは、CY8C28xxx および CY8C24x94 パーツのみでサポートされています。

Figure 1.　内部 Vdd/2 バイアス信号生成器を備えた SLCD ユーザ モジュールのブロック図

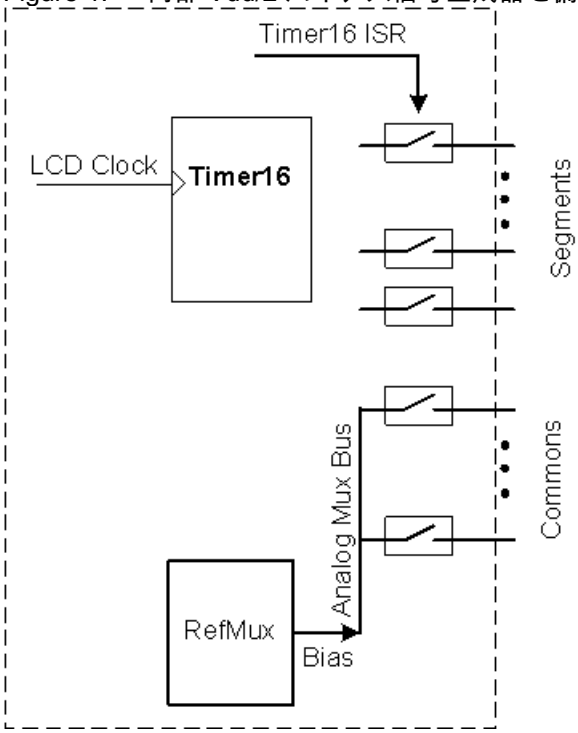
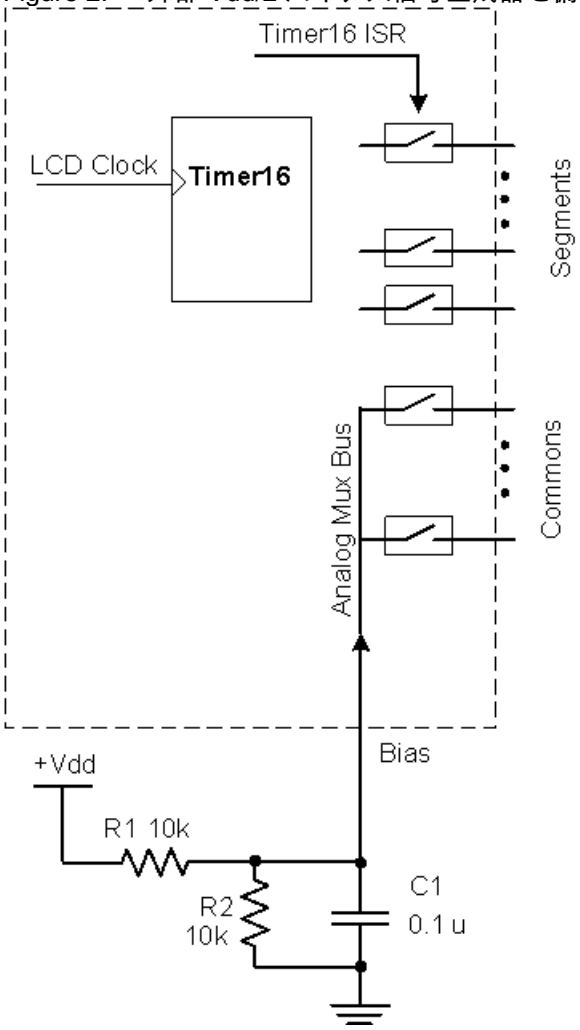


Figure 2. 外部 Vdd/2 バイアス信号生成器を備えた SLCD ユーザ モジュールのブロック図



クイック スタート

1. SLCD ユーザ モジュールを選択して配置します。
2. SLCD ユーザーモジュールのパラメータを設定します：LCD クロック、更新レート、コントラストのレベル。
3. SLCD ユーザ モジュール を右クリックし、**[SLCD Wizard] (SLCD ウィザード)** を選択します。
4. ディスプレイのコモン数に適した **[No. of Commons] (コモンの数)** を設定します。
5. LCD の **[Number of Segment Lines] (セグメントライン数)** を入力します。
6. [Add/Remove Display] (ディスプレイの追加 / 削除) の下にある **+** ボタンを押して、 数字または英数字 LCD ディスプレイを追加します。
7. [Increment/Decrement Digits] (桁の増減) の下にある **+** ボタンを押して、 ディスプレイに桁を追加します。
8. ディスプレイの桁からセグメントを **セグメント - コモン マッピング表**にドラッグドロップします。全セグメントのマッピングは、LCD データシートに対応させてください。
9. PSoC ピンにセグメントを割り当てるには、**セグメント - コモン マッピング表** から **[Pin Assignment] (ピン配置)** エリアにセグメントをドラッグドロップします。

10. PSoC ピンにコモンを割り当てるには、**セグメント - コモン マッピング表** から **[Pin Assignment] (ピン配置)** エリアにコモンをドラッグドロップします。
11. 外部バイアス信号生成器を使用する場合は、**[Bias] (バイアス)** をチップ ビューの該当するピンにドラッグドロップします。
12. **[OK] をクリック** してから、アプリケーションを生成します。
13. アプリケーション エディタに切り替えて、必要に応じてサンプルコードを使用します。
14. ターゲット ボードで PSoC をプログラムします。

機能説明

セグメント

セグメントは、数字または英数字ディスプレイの 1 本のバーです。

セグメント制御ライン

マルチプレックス LCD では、複数のセグメントが接続され、1 つの端末がピンに割り当てられます。このピンをセグメント制御ラインと呼びます。

記号

小数点、コロン、バッテリ充電インジケータ記号、ワイヤレス、アラームなどの構造を示します。

SLCD ユーザ モジュールは、1/2 バイアスで 2.8V ～ 5V のマルチプレックス LCD を直接駆動できます。このユーザ モジュールには GUI ウィザードが備わっており、LCD のタイプに応じてモジュールを簡単に構成できます。

このユーザ モジュールは、次のタイプのパネル内ディスプレイ構造をサポートします。

- 7 セグメント (数字)
- 14 セグメント (英数字)
- 16 セグメント (英数字)
- 特殊記号

UM によりサポートされる LCD を駆動する技法には、以下の 2 つがあります。

1. アナログ MUX バスを使用する LCD ドライブ

この技法では、1/2 Vdd バイアスをコモンラインに提供するためにアナログ MUX バスを使用します。この技法は、アナログ MUX バスを備えたデバイスに適用されます。

2. スタンドアロンの GPIO 使用する LCD ドライブ。

この構成では、アナログ MUX バスを備えたデバイスである必要はありません。これは、汎用入出力 (GPIO) を使用して、LCD を駆動します。この構成では、コントラスト制御も実装します。

SLCD ユーザ モジュールは、すべてのデバイスで 2 つのオプションをサポートします。

1. タイマー 16 HW

2. タイマー 16 HW/SW

タイマー 16 HW は、2 つのデジタル PSoC ブロックを消費します。タイマー 16 HW/SW は、1 つのデジタル ブロックを消費して、8 ビットの HW タイマーを実装します。残りの 8 ビットは、ファームウェアによって実装されます。タイマーの ISR は、LCD のリフレッシュタイミングとピンの LCD 波形を制御します。

SLCD ユーザ モジュールは、デバイスがスリープ モードになっている場合でも動作します。これは、LCD タイマーのクロックで CPU_32K (ILO または ECO) が選択されている場合にのみ可能です。デバイスは、定期的な間隔で起き上がり、LCD を更新します。デバイスをスリープに設定していて、LCD の動作が必要ない場合は、利用可能なクロックであればいずれでも構いません。　デバイスのスリープを設定する前に、SLCD ユーザ モジュールの Stop() API を呼び出してください。

コントラスト制御は、LCD が同じリフレッシュサイクルを保ち、リフレッシュサイクル間でデッド時間を設けることで達成されます。これによりデッド時間が増えると、アクティブなコモン信号の期間が削減されます (逆も言えます)。このデッド時間中、コモンラインとセグメントラインは、Vdd で保持されます。これにより、オンになっていたセグメントがデッド時間中一瞬だけオフになり、コントラストが調整されます。デッド時間が増えるとコントラストが下がります (逆もいえます。)。

コントラスト制御を説明する LCD 波形を 示図 4 します。コントラスト = 100% の場合、LCD 波形は 示図 3 されているような波形となります。LCD タイマー ISR 期間は、式 1 を使って計算されます。

Equation 1

$$TimerPeriod = \frac{LCDClock}{2 \times n \times Fr}$$

式 1 :

LCD_CLOCK - LCD のタイマークロック (Hz)

n - コモンの数

Fr - LCD の更新レート (Hz)。

コントラストが 100% 未満の場合に、デッド時間を導入します。デッド時間の期間は、コントラストのレベルに反比例します。

アナログＭＵＸバス LCD ドライブ技法のデッド時間 :

Equation 2

$$Pd = \left(\frac{1}{Fr}\right) - 2 \times n \times (Pc \pm \Delta)$$

スタンドアロン GPIO LCD ドライブ技法のデッド時間 :

Equation 3

$$Pd = \left(\frac{1}{Fr}\right) - 4 \times n \times (Pc \pm \Delta)$$

ここで、Δ は、ユーザが指定する増減の期間カウントです。

(Pc ± <) コントラストが 100 % の場合の、アクティブなコモン信号の新しい LCD タイマー期間

Pd - コントラストが < 100 % の場合、デッド中の LCD タイマー期間。

一般に、LCD のリフレッシュ期間は、アクティブなコモン信号期間とデッド期間から構成されます。

Equation 4

$$\text{LCD Refresh Period} = M \times n \times (Pc \pm \Delta) + Pd$$

ここで、アナログＭＵＸバス ドライブ技法の場合、M=2、標準の GPIO ドライブ技法の 場合 M=4 です。

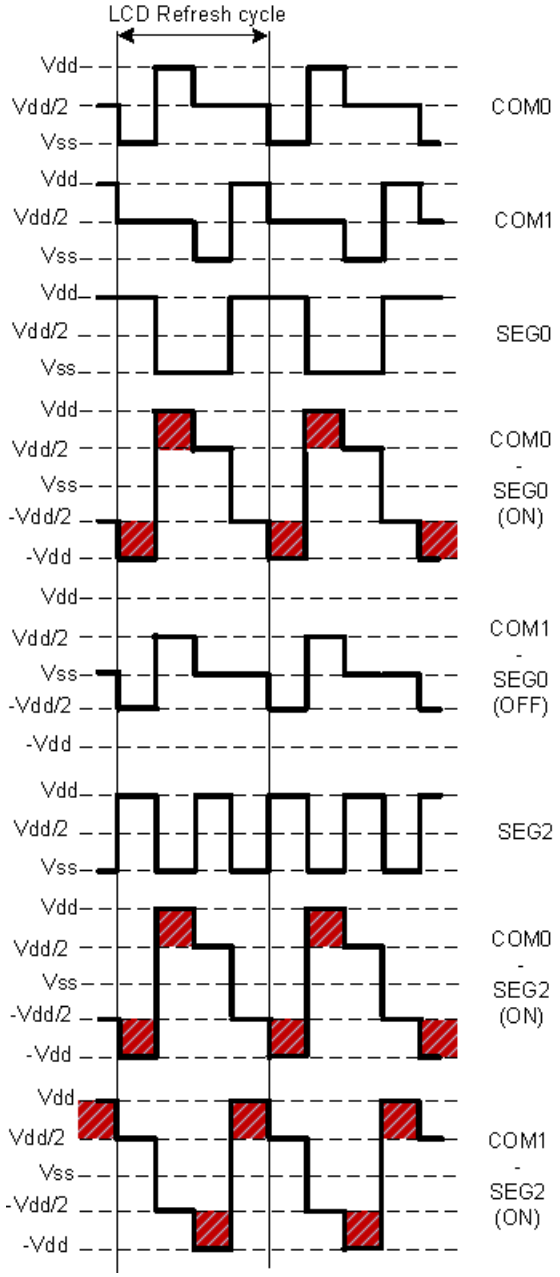
SLCD ユーザ モジュール データシートを読んだあとに、次のアプリケーションノートを推奨します。アプリケーション ノートは、サイプレス セミコンダクタのウェブサイト (www.cypress.com) からご覧いただけます。

- *PSoC を使った LCD の駆動方法* - AN2228
- *CapSense を使用した PSoC によるセグメント LCD パネルの駆動* AN56384

タイミング

図 3 に、1/2 バイアスを形成するためにアナログＭＵＸバスを使用した場合の、SLCD ユーザ モジュールのタイミングを示します。図 4 は、アナログＭＵＸバス技法を使った LCD 駆動で使用されるコントラスト制御スキームを示しています。

Figure 3.　1/2 バイアスの LCD タイプ A 波形



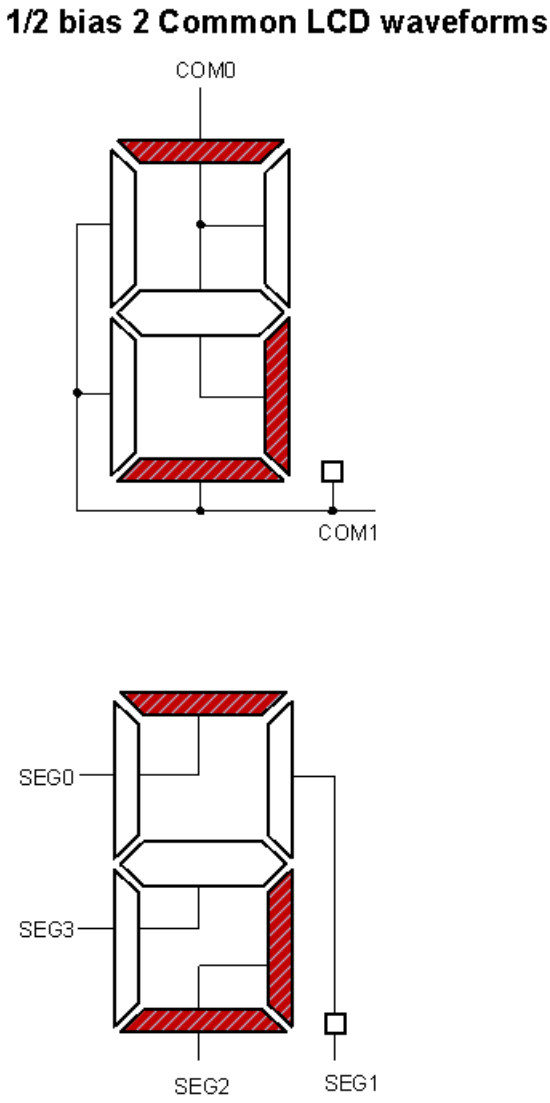


Figure 4. 1/2 バイアスおよびコントラスト = 25% の LCD 波形

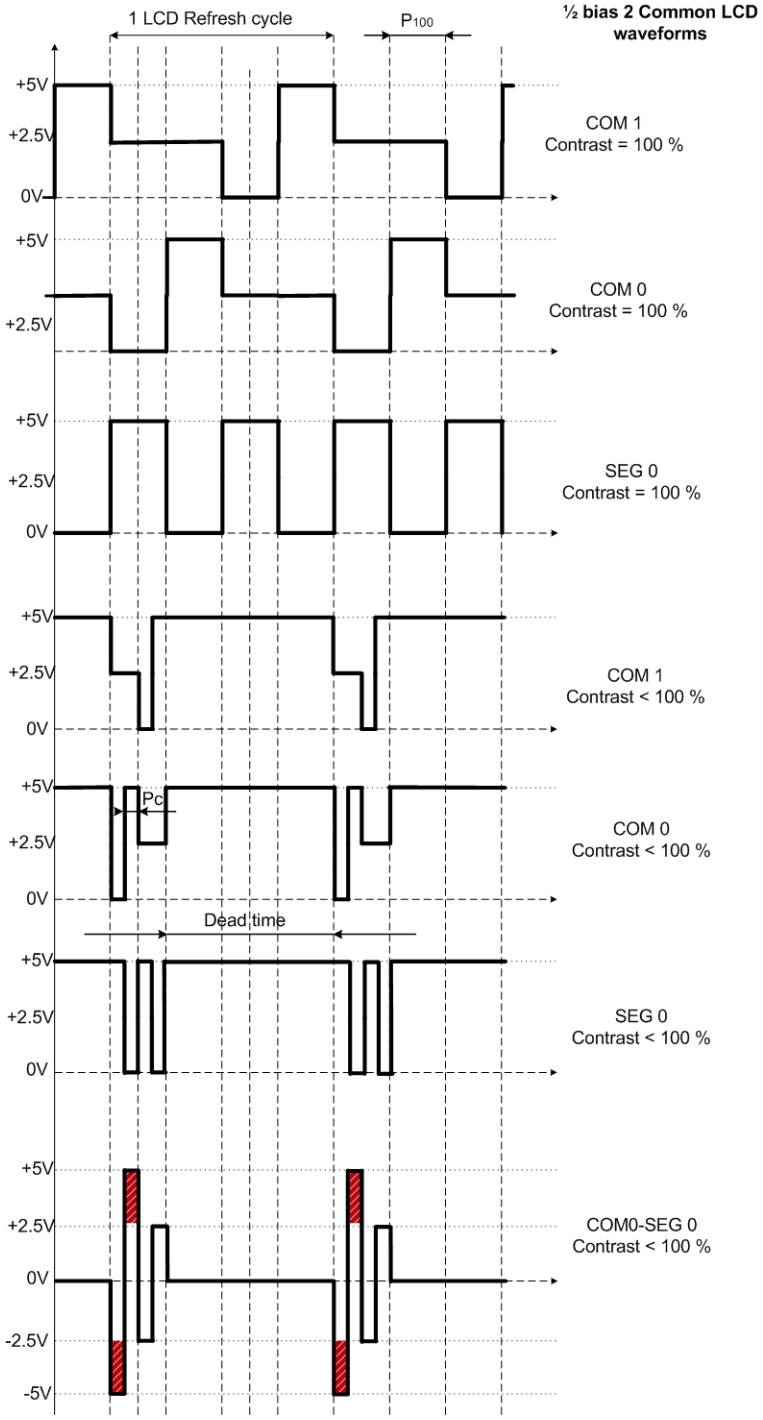
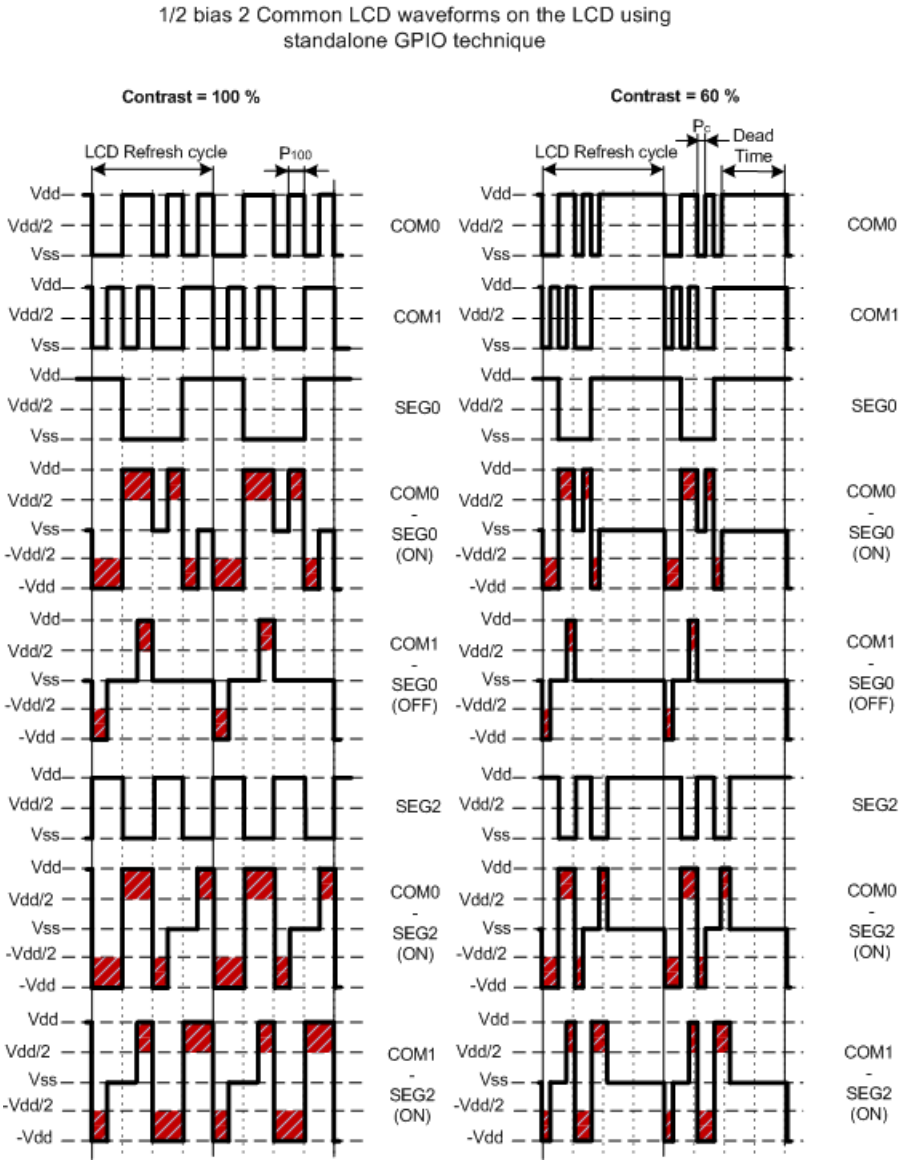


Figure 5. 標準の GPIO 技法を使用した場合の SLCD ユーザ モジュールのタイミング



DC 電気的特性と AC 電気的特性

Table 1. 電源電圧

パラメータ	最小値	標準値	最大値	単位	テスト条件とコメント
[1] 値 V_{DD}	2.7	5.0	5.25	V	-

[1] 電源電圧は、LCD のタイプによって決まります。

[+] Feedback

Table 2. タイミングと DC 特性

パラメータ	標準値	上限	単位	テスト条件とコメント
最大 LCD タイマー入力周波数	-	2000	kHz	Vdd = 5.0V
最小 LCD タイマー入力周波数	-	12	kHz	Vdd = 5.0V
最大 LCD タイマー期間	-	0.25	msec	Vdd = 5.0V
[2]LCD (セグメント制御ライン - コモン) オフセットエラー	-	40	mV	アナログ MUX バスを使用する LCD ドライブ Vdd = 5.0V、 CPU_Clock = SysClk/2 (12 MHz)、更新レート = 150 Hz、 コモンラインの数 = 4
	-	25	mV	アナログ MUX バスを使用する LCD ドライブ Vdd = 3.3V、 CPU_Clock = SysClk/2 (12 MHz)、更新レート = 150 Hz、 コモンラインの数 = 4
	-	11	mV	スタンドアロンの GPIO を使用する LCD ドライブ Vdd = 5.0V、 CPU_Clock = SysClk/2 (12 MHz)、更新レート = 150 Hz、 コモンラインの数 = 4
	-	8	mV	スタンドアロンの GPIO を使用する LCD ドライブ Vdd = 3.3V、 CPU_Clock = SysClk/2 (12 MHz)、更新レート = 150 Hz、 コモンラインの数 = 4
	-	30	mV	スタンドアロンの GPIO を使用する LCD ドライブ Vdd = 5.0V、 CPU_Clock = SysClk/8 (3 MHz)、更新レート = 150 Hz, コモンラインの数 = 4
	-	20	mV	スタンドアロンの GPIO を使用する LCD ドライブ Vdd = 3.3V、 CPU_Clock = SysClk/8 (3 MHz)、更新レート = 150 Hz, コモンラインの数 = 4

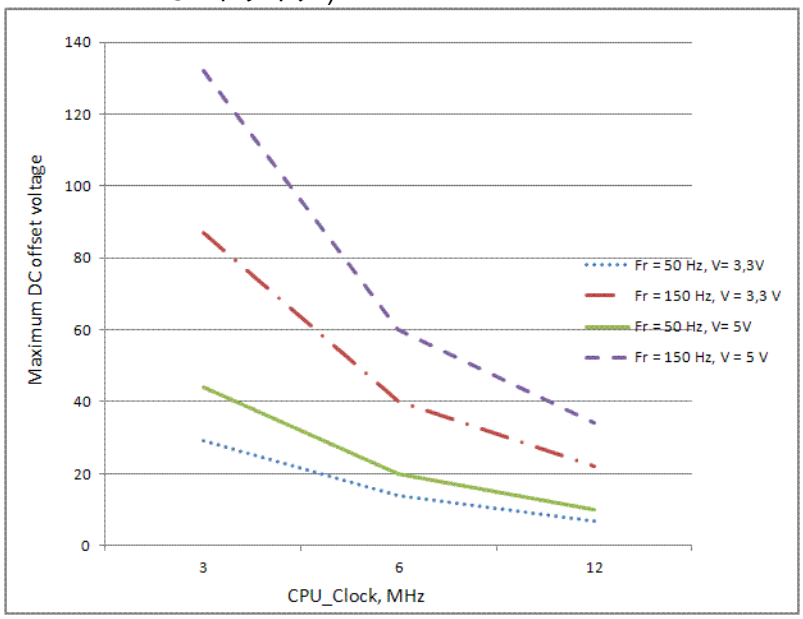
[2] LCD 波形は、SLCD UM 割り込みによって形成されます。SLCD UM 割り込みのレイテンシにより、コモンおよびセグメント信号波形に不均整が生じ、電圧のオフセットが発生します。DC オフセット電圧の値が大きくなると、液晶の流動体が劣化し、励起しなくなります。このため、LCD 電極にかかる DC 電圧により、LCD の寿命が短くなります。

ユーザのコードやその他のモジュールコードを持つ ISR をもつアプリケーションは、LCD 更新の遅延を発生させ、更なる DC オフセット電圧につながります。また、CPU_Clock が低くなると、DC のオフセット電圧がデータシートの値を超えることがあります。

オフセット電圧は、セグメントおよびコモンピン間の電圧における DC コンポーネントとして、更新期間全体で測定できます。

DC オフセット電圧に影響する割り込みレイテンシを下げるには、PSoC Designer デバイス エディタで、CPU_Clock グローバル パラメータを増やします。また、LCD リフレッシュレートが低い場合は、DC オフセット電圧は低くなります。

Figure 6. さまざまな更新レートの CPU_Clock に依存する DC オフセット電圧 (アナログ M U X バスを使用する LCD ドライブ)



CPU_Clock グローバル パラメータは、SLCD UM にとって重要なパラメータで、その構成は UM のパフォーマンスに大きく影響します。UM の構成で CPU_Clock 値が低すぎると、DC オフセットが大きくなり、時間が経つにつれて LCD に損傷を与えます。可能な場合は、より高い CPU_Clock 値を使用してください。12MHz の CPU_Clock を選択すると、ほとんどの SLCD UM 構成で、電源とパフォーマンスのバランスを良好に保つことができます。

配置

1 のプロジェクトで利用できる SLCD ユーザ モジュールのインスタンスは 1 つです。

SLCD ユーザ モジュールは、以下の構成を可能にします。

- **アナログ M U X バスと外部 1/2 バイアス信号発生器を使用するセグメント LCD ドライバ**

  電圧分配器を基にした外部抵抗を接続し、示図 2 されているような Vdd/2 バイアスを生成します。

  このバイアスは、PSoC ピンに迂回させる必要があります。これは、LCD で使用されるアナログ M U X バスに内部的に接続されます。ピンのオプションは、SLCD ユーザ モジュール ウィザードで提供されます。これについては、「ウィザード」のセクションで説明します。

  サポートされているデバイスはすべて、このオプションを持ちます。

[+] Feedback

■ **アナログＭＵＸバスと内部 Vdd/2 バイアス信号発生器を使用するセグメント LCD ドライバ**

RefMux ユーザ モジュールが配置され、Vdd/2 バイアスを生成します。さらに、LCD で使用されるアナログＭＵＸバスに渡されます。

この構成では、すべてのデバイスが 2 つのデジタル ブロックを使用します。さらに、内部バイアス生成を選択すると、CY8C28x および xxCY8C24 デバイスでアナログ CT ブロックが消費されます。

■ **スタンドアロンの GPIO を使用するセグメント LCD ドライバ：**

この構成では、アナログＭＵＸバスは使用されません。

Table 3.　ブロック リソース

ブロック	モジュール	名前	注
デジタル ブロック	Timer16/ タイマ 8	LCDTimer	これは、すべてのデバイスおよび両方のドライブ技法で使用されます。ここで、16 ビット HW タイマまたは 16 ビット HW/SW タイマのいずれかを選択できます。16 ビット HW タイマは、2 つのデジタル リソースを使用し、16 ビット HW/SW は 1 つのデジタル リソースを使用します。このタイマのクロックは、ユーザ モジュールのプロパティで選択できます。
アナログ CT ブロック	RefMux	LCDBias	CY8C28xxx および CY8C24x94 の場合にのみ、アナログＭＵＸバスを使用する LCD ドライブで内部バイアスが選択されている場合。

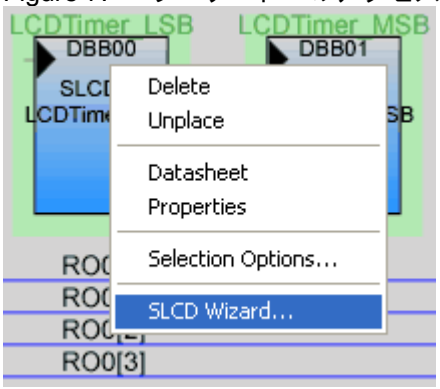
Note　CapSense ™ モジュールを設計にすでに配置している場合は、アナログＭＵＸバスを使用する LCD ドライブに適用される制限があります。

- CPU_Clock グローバル パラメータは、SLCD UM にとって重要なパラメータで、その構成は UM のパフォーマンスに大きく影響します。UM の構成で CPU_Clock 値が低すぎると、DC オフセットが大きくなり、時間が経つにつれて LCD に損傷を与えます。可能な場合は、より高い CPU_Clock 値を使用してください。12MHz の CPU_Clock を選択すると、ほとんどの SLCD UM 構成で、電源とパフォーマンスのバランスを良好に保つことができます。
- SLCD ユーザ モジュールを CapSense ボタンを使う同じ設計に配置できる唯一のデバイスは、CY8C22x45 および CY8C28xxx ファミリ デバイスです。その他の場合では、デザイン・ルール・チェックで、アナログＭＵＸバスが利用できないことを示すメッセージが表示されます。
- CY8C22x45 や CY8C28xxx ファミリでも、CapSense モジュールが両方のアナログＭＵＸバスを消費する場合は、SLCD ユーザ モジュールを配置できません。CapSense がアナログＭＵＸバス を 1 つ使用し、SLCD がその他のバスを使用する場合にのみ、CapSense と SLCD ユーザ モジュールが使用できます。

ウィザード

ウィザードにアクセスするには、Workspace Explorer で SLCD ユーザ モジュールのブロックを右クリックし、[SLCD Wizard] (SLCD ウィザード) を選択します。

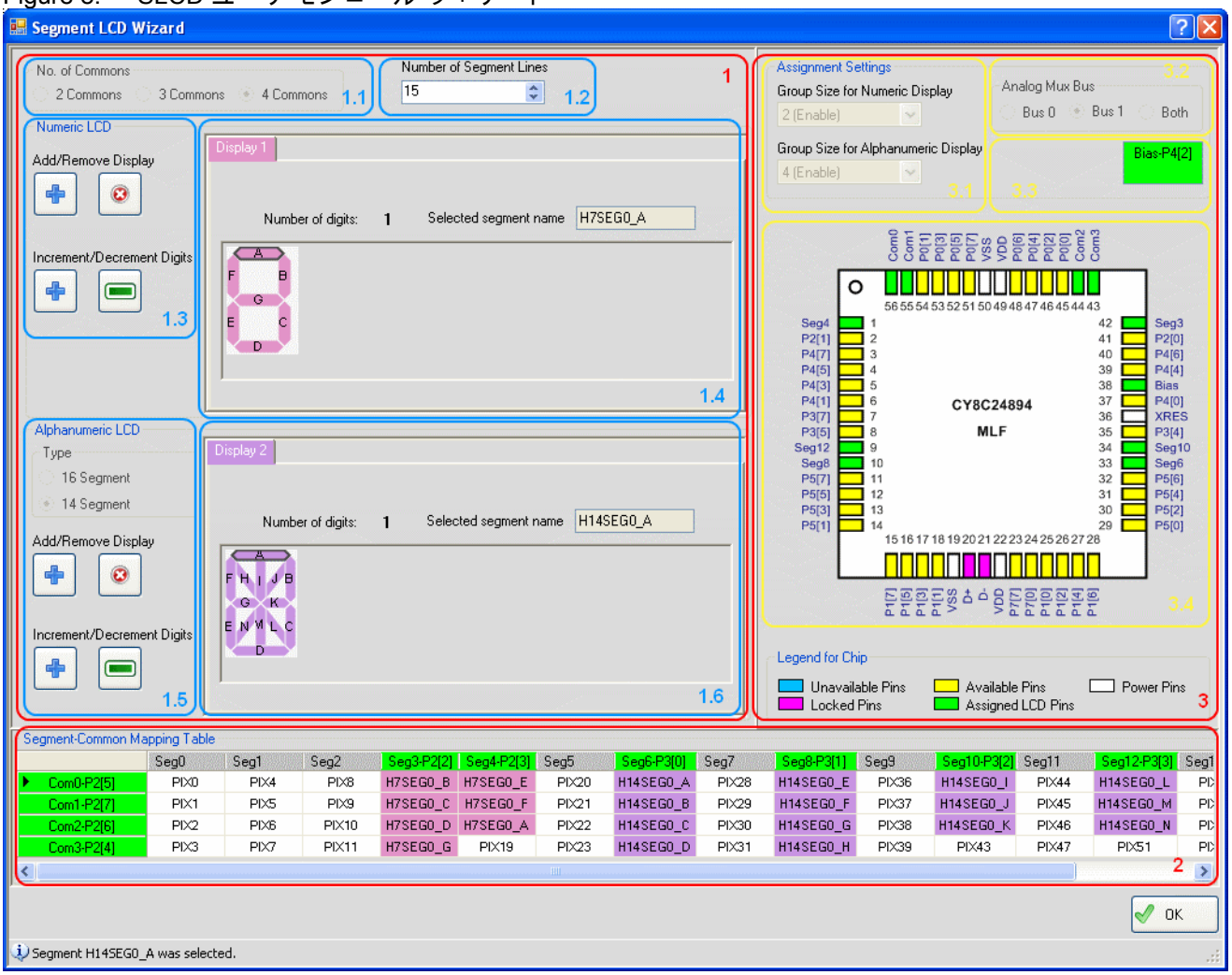
Figure 7.　ウィザードへのアクセス



SLCD ユーザ モジュール ウィザードの一般的な表示を 示図 8 します。

ウィザードを完了してプロジェクトが構築されると、該当する ID を持つすべてのディスプレイの定数が SLCD.inc および SLCD.h ファイルで利用できるようになります。特別な記号および、チップ ビューで割り当てられるセグメントラインである 「セグメント - コモン マッピング表」のフリーのセグメントの定数が生成されます。これらの定数は、SLCD ユーザ モジュール API とともに使用します。

Figure 8. SLCD ユーザ モジュール ウィザード



SLCD ウィザードには、3 つの主なセクションがあります (図では、別の色で示されています)。

1. LDC の仕様

   コモンラインおよびセグメントライン数の指定 (マルチプレックス LCD では、複数のセグメントがピンに迂回される 1 つのコラムに接続されます)、数字または英数字ディスプレイの追加または削除、ディスプレイでの桁の追加または削除を行います。

2. セグメント - コモン マッピング表

   LCD の仕様に基づき、セグメント (セグメントはディスプレイの 1 本のバー) と記号 (記号は小数点、コロン、充電器インジケータなどのアイテム) をコモンにマップします。

3. ピン配置

   バイアス ピン、コモン、セグメントラインを PSoC ピンに割り当て、ディスプレイのグループサイズを設定します。

[OK] ボタン

このボタンは、以下の 2 つの操作を実行します。

■ ウィザード パラメータの保存

[+] Feedback

■ asm および inc コード フラグメントの生成

[LCD Specification] (LDC の仕様セクション)

ここでは、コモンラインおよびセグメントライン数の設定、数字または英数字ディスプレイの追加または削除、ディスプレイでの桁の追加または削を行います。

[No. of Commons] (コモンの数)

コモンラインの数は 2、3、または 4 です。マッピング表の行数は、コモンラインの数に等しくなります。マッピング表で桁のセグメントをセグメントラインに割り当てた後で、コモンラインの数を変更することはできません。コモンラインの数を変更するには、セグメントラインからすべてのセグメントの割り当てを解除してください。

[Number of Segment Lines] (セグメントライン数)

セグメントの制御ラインの最大数は、選択した PSoC デバイスに存在するフリーの IO 数、コモンラインの数、外部バイアス ピンが使用されているかどうかによって異なります。セグメントライン数の最大値 :

Equation 5

$$SegmentLines_{MAX} \quad FreePSoCPins - CommonLines - ExternalBiasPin$$

マッピング表の列数は、セグメントラインの数に等しくなります。

Note 桁からのセグメントがマッピング表のセグメントラインに割り当てられている場合は、セグメントラインの数を減らすことはできません。

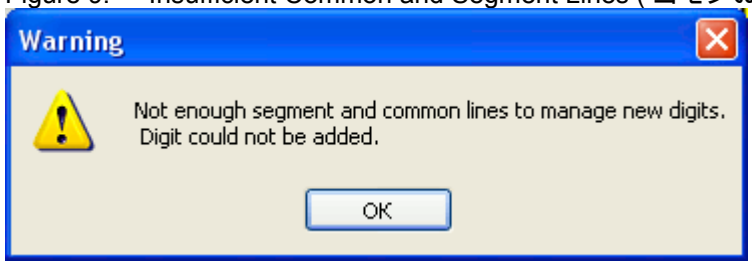
[Numeric LCD] (数字 LCD)

これは、数字ディスプレイの追加または削除、ディスプレイへでの桁の追加または削除を行います。

[Add/Remove Display] (ディスプレイの追加 / 削除) - LCD パネルで数字ディスプレイを追加または削除します。各ディスプレイは、「ディスプレイ 1」、「ディスプレイ 2」などという固有の名前を持ちます。

[Increment/Decrement Digits] (桁の増減) - 選択した数字ディスプレイで 7-segment 桁を追加または削除します。十分なセグメントおよびコモンラインが残っていない場合は、桁を追加しようとすると、次の警告が表示されます。

Figure 9. Insufficient Common and Segment Lines (コモンおよびセグメントラインの不足)



[LCD Panel Panels 1-N] (LCD パネルのパネル 1-N)

このエリアには、すべての数字ディスプレイと 7-segment 桁が含まれます。

[Number of digits] (セグメント数) は、選択したディスプレイの 7-segment 桁数を示します。

[+] Feedback

[Selected segment name] (選択したセグメント名) は、ディスプレイで選択したセグメントの名前を示します。

[Alphanumeric LCD] (英数字 LCD)

これは、英数字ディスプレイの追加または削除、ディスプレイへでの桁の追加または削除を行います。

[Type] (種類) は、桁が 14 または 16-segment 英数字ディスプレイであるかどうかを選択します。同じディスプレイで 14 と 16-segments を混同させることはできないため、ディスプレイに最初の英数字ディスプレイを追加した時点で、このコントロールは無効になります。14 と 16-segment 桁を切り替えるには、ディスプレイからすべての桁を削除します。

[Add/Remove Display] (ディスプレイの追加 / 削除) - 英数字 LCD パネルで英数字ディスプレイを追加 / 削除します。各ディスプレイは、「ディスプレイ 1」、「ディスプレイ 2」などという固有の名前を持ちます。

[Increment/Decrement Digits] (桁の増減) - 選択した英数字ディスプレイで、その種類に応じて 14-segment または 16-segment 桁を追加 / 削除します。新しい桁に対してセグメントおよびコモンラインの数が足りない場合は、に図 9 示されているような警告メッセージが表示されます。

[Alphanumeric LCD Panel] (英数字 LCD パネル)

追加された英数字ディスプレイと選択したディスプレイの 14-/16-segment 桁が含まれます。

「Number of digits」(桁数) は、選択したディスプレイの 14-segment または 16-segment 桁数を示します。

「Selected segment name」(選択したセグメント名) は、14-segment または 16-segment 桁で選択したセグメントの名前を示します。

[Segment-Common Mapping Table] (セグメント - コモン マッピング表) セクション

このセクションでは、桁と記号からコモンへのセグメント マッピングを実行できます。

行数は、コモンラインの数に等しくなります。列数は、セグメントラインの数に等しくなります。コモンおよびセグメントラインを追加または削除すると、表からローおよびコラムが増減されます。

桁からマッピング表へのセグメントの移動

桁からマッピングテーブルにセグメントをドラッグします。セグメントをドラッグすると、マッピング表でフリーのセルがすべてハイライトされます。ここでは、2 つの方法を使用できます。グループサイズ (数字または英数字ディスプレイ用) を特定の数に設定するには、**コモン インデックス法**を使用します。グループサイズを **[None] (なし)** に設定する場合、**ユニバーサル法**を使用します。

コモン インデックス法は、コモンピンに対して同じセグメント配列の桁を持つ LCD ディスプレイに便利です。たとえば、セグメント「a」は、すべての桁で同じコモンピンを持ちます。この方法は、多くの桁で LCD ディスプレイの Flash の使用を軽減します。また、ウィザードのコントロールにより、どのセグメントが可能かどうかが示されるため、セグメントを正しく割り当てるのにも役立ちます。

ユニバーサル法は、異なるコモンピン割り当てられた桁セグメントを持つ非標準のディスプレイに便利です。この方法を使うと、桁セグメントを表のどのセルにでも割り当てることができます。

コモン インデックス法

グループ サイズを特定の数に設定すると、ディスプレイの特定の桁からのセグメントが、グループ サイズに該当する特定のセグメントラインのみを占有します。桁から最初のセグメントをドラッグすると、他の桁で占有されていないすべてのセルがハイライトされます。

Figure 10. マッピング表でフリーのセルがすべてハイライトされる

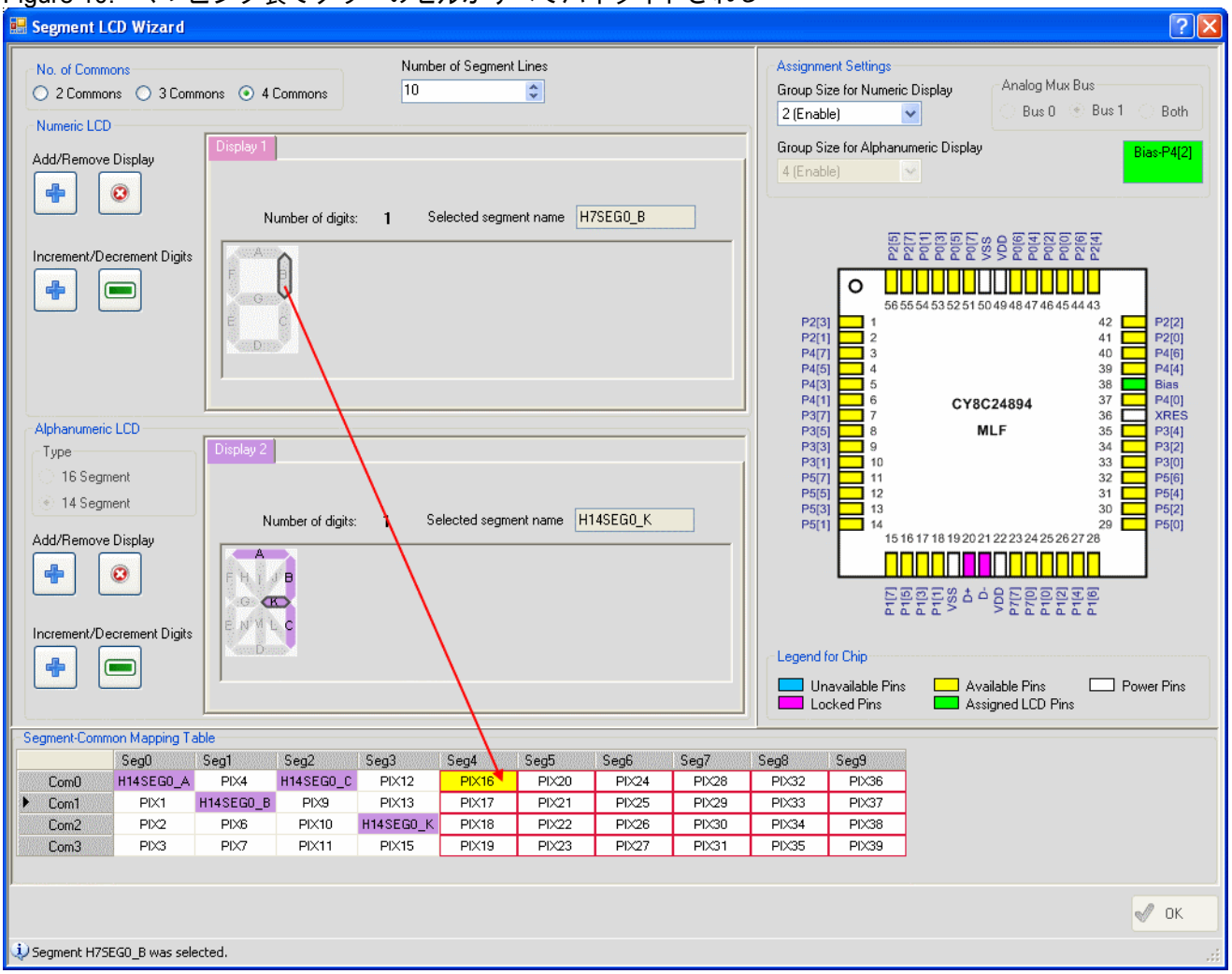
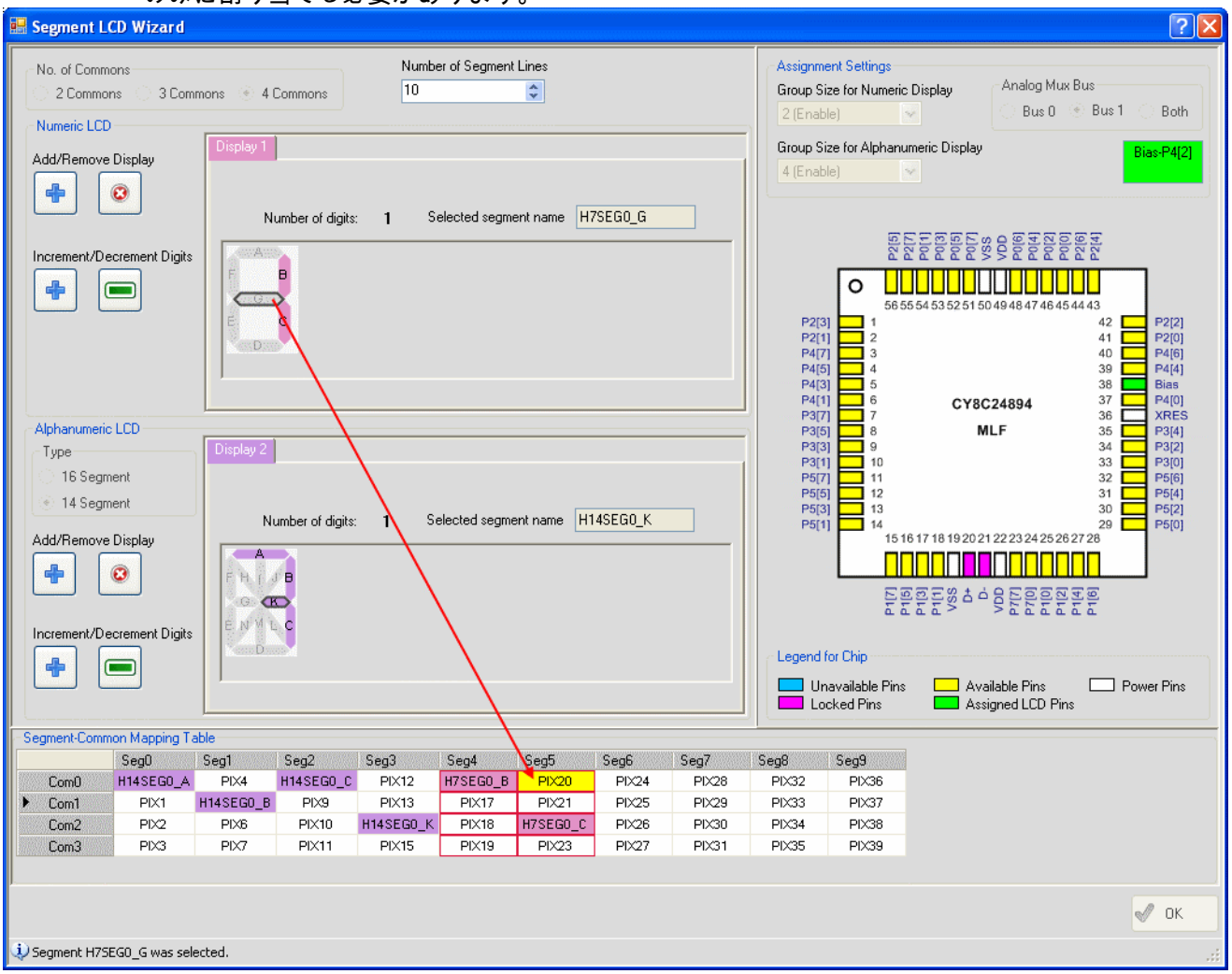


図 10 では、4 つのコモンを持つ LCD ディスプレイ は Display 2 からの桁用に取得されたセグメントライン Seg0、Seg1、Seg2、Seg3 を持っています。これらのセグメントラインは、Display 1 では利用できません。

[+] Feedback

同じ桁からの 2 つ目のセグメントを別のセグメントラインに割り当てる場合は、その他のすべてのセグメントを図 11：

Figure 11. 「マッピング表の同じグループで利用可能なセグメントセルのハイライト」で示されているセグメントのみに割り当てる必要があります。



ユニバーサル法

グループサイズを 「なし」に設定すると、すべての桁からのセグメントをセグメント - コモン マッピング表で利用可能などのセグメント セルにでも割り当てることができます。1 つのセグメント ラインは、7-segment または 14/-16-segment 桁に属するセグメントを含みます。

桁セグメントに割り当てられていない表のセルは、LCD 記号を示します。SLCD ウィザードは、SLCD_EnableSymbol API のパラメータとして使用される SLCD.inc および SLCD.h ファイルで適切な定数を生成します。

マッピング表からセグメントラインをチップに移動する

セグメントをマッピング表にマップしたら、セグメントラインをデバイスのピンにマップする必要があります。セグメントをクリックしてドラッグすると、デバイスで利用可能なピンがすべてハイライトされます。

使用する方法は、表にセグメントを割り当てるために、コモン インデックス法とユニバーサル法のどちらを使用するかによって異なります。

[+] Feedback

コモン インデックス法

同じグループのメンバであるすべてのセグメントラインがハイライトされ、チップピンにいっしょに割り当てられます。一番左にあるセグメントラインは、最下位のピン番号に割り当てられます。たとえば、Display 2 の桁からの全セグメント (参図 10 照) は、マッピング表の Seg0、Seg1、Seg2、Seg3 に割り当てられます。以下の表に、これらのセグメントラインをチップに割り当てる際のバリエーションを示します。

Table 4.　4 つのセグメントラインからチップへの割り当てのバリエーション

Seg0	Seg1	Seg2	Seg3
Px_0	Px_1	Px_2	Px_3
Px_1	Px_2	Px_3	Px_4
Px_2	Px_3	Px_4	Px_5
Px_3	Px_4	Px_5	Px_6
Px_4	Px_5	Px_6	Px_7

この表で、「x」はチップのポートを表しています。

ピン 4 がフリーでない場合は、これらのセグメントラインの割り当てには、バリエーション 1 のみが可能です。

ユニバーサル法

グループサイズを 「なし」に設定すると、すべてのラインは、チップで利用可能などのピンにでも割り当てることができます。

セグメントラインをチップに割り当てたら、割り当てられたポートとピンの名前が、Seg1-Px[2] などのように、セグメントラインの近くに表示されます。

マッピング表からコモンラインをチップに移動する

コモンラインをデバイスのピンにドラッグできます。ドラッグすると、利用可能なピンがすべてハイライトされます。CY8C28xxx および CY8C22x45 デバイスでは、アナログＭＵＸバス 1 が選択されている場合はコモンラインが偶数のピン (単一のポートで)、アナログＭＵＸバスが 0 に選択されている場合は奇数のピンに割り当てられます。その他のすべてのデバイスでは、コモンラインは、同じポートのどのピンにでも割り当てることができます。

割り当てたセグメントのリセット

マッピング表で割り当てたセグメントをリセットするには、マッピング表でセグメントを右クリックし、コンテキストメニューから [Reset] (リセット) を選択します。セグメントラインがチップに割り当てられていて、このラインの最後のセグメントをリセットすると、セグメントラインとチップの割り当てが解除されます。

[Pin Assignment] (ピンの割り当て) セクション

このセクションでは、バイアス ピン、コモン、セグメントラインをピンに割り当て、数字および英数字ディスプレイのグループサイズを設定します。このセクションは、LCD ドライブ技法では多少異なります。スタンドアロンの GPIO を使用するセグメント LCD ドライバのウィザードでは、[Assignment Settings] (割り当て設定) にアナログＭＵＸバス コントロールとバイアスピンがありません。

[+] Feedback

[Group Size for Numeric Display] (数字ディスプレイのグループサイズ)

[Group Size for Alphanumeric Display] (英数字ディスプレイのグループサイズ)

1 つのグループを形成するセグメントの数を指定します。グループサイズで利用できる値は、コモンとディスプレイの数、ディスプレイのタイプによって異なります。

Table 5. グループサイズ

コモンの数	ディスプレイのタイプ	
	数字	英数字
2	4	8
3	3	6
4	2	4

さらに、グループサイズの値は 「None」(なし) も選択できます。この場合、すべての桁からのセグメントをセグメント - コモン マッピング表で利用可能などのセグメント セルにでも割り当てることができます。また、すべての個別のセグメントラインを、チップで利用可能なすべてのフリーなピンに割り当てることが可能です。

桁からのセグメントが少なくとも 1 つマッピング表のセグメントラインに割り当てられている場合はこのコントロールが無効になり、すべての桁からのすべてのセグメントが割り当てられていない場合は有効になります。

Note グループサイズがゼロ以外の値に設定されると、セグメントラインのグループ全体が、チップ ビューの連続したピンに割り当てられます。

LCD が、桁のセグメントを持たないセグメントラインに属する記号を持つ場合は、空のセグメントラインをチップに移動できます。この場合、SLCD ウィザードは、これらの記号に該当する適切な定数を SLCD.inc および SLCD.h ファイルで生成します。

[Analog Mux Bus] (アナログＭＵＸバス)

アナログＭＵＸバスは、CY8C28xxx および CY8C22x45 PSoC デバイス ファミリでのみ利用できます。その他のデバイスでは、このコントロールは無効になります。このオプションは、チップへのバイアスピンとコモンラインの割り当てに影響します。

[Bias Pin] (バイアスピン)

バイアスピンは、外部バス構成を選択した場合にのみ利用できます。選択したデバイス ファミリにより、バイアスピンは、奇数または偶数のピンに割り当てる必要があります。CY8C22x45 および CY8C28xxx ファミリでは、アナログＭＵＸバス 0 が選択されている場合はバイアスピンを奇数のピンに、アナログＭＵＸバス 1 が選択されている場合は偶数のピンに割り当てることができます。その他のデバイスでは、バイアスピンは利用可能などのピンにでも割り当てることが可能です。

バイアスピンをクリックして、チップにドラッグします。ピンの名前は、割り当てを示すように更新されます。

チップ ビュー

選択した PSoC デバイスを表示します。セグメント、コモンライン、バイアスピン (該当する場合) を、チップ ビューのピンにドラッグして割り当てます。ピンを割り当てる規則は、前述の通りです。

コンテキストメニューを使用すると、1 つのピンのみ、割り当てられた複数のピン、チップで割り当てられるすべてのピンをリセットできます。このメニューは、「Chip」(チップ) コントロール

[+] Feedback

またはピンを右マウスボタンでクリックすると表示されます。「Chip」(チップ) コントロールをクリアするのに便利です。

このコンテキストメニュー ツールには、4 つのエントリがあります。

- [Reset Selected Pin] (選択したピンをリセット) - 現在選択しているピン (ピンをクリックした後で利用可能になる) をクリアします。
- [Reset All Pins of Numeric Display] (数字ディスプレイの全ピンをリセット) - 数字ディスプレイに属するすべてのピンをクリアします。
- [Reset All Pins of Alphanumeric Display] (英数字ディスプレイの全ピンをリセット) - 英数字ディスプレイに属するすべてのピンをクリアします。
- [Reset All Pins] (すべてのピンをリセット) - 割り当てられているすべてのピンをクリアします。

その他の機能

「Chip」(チップ) コントロールでピンが割り当てられていない場合は、このメニューは表示されません。

数字ディスプレイのみに属するピンが割り当てられている場合は、[Reset All Pins of Alphanumeric Display] (英数字ディスプレイの全ピンをリセット) メニュー項目は無効になります。

英数字ディスプレイのみに属するピンが割り当てられている場合は、[Reset All Pins of Numeric Display] (数字ディスプレイの全ピンをリセット) メニュー項目は無効になります。

パラメータおよびリソース

LCD Clock Source (LCD クロック源)

LCDTimer のクロック源を設定します。

Table 6. 設定

タイプ	範囲	デフォルト	コメント
-	VC1、VC2、VC3、CPU_32K、Row_x_Output_0、Row_x_Output_1、Row_x_Output_2、Row_x_Output_3、Row_x_Input_0、Row_x_Input_1、Row_x_Input_2、Row_x_Input_3、Row_x_Broadcast、Diable (無効)	VC1 (VC2*)	これは、SLCD ユーザ モジュールのクロック源を設定します。

最小 LCD タイマ クロックは、$20nF_r$ に等しくなります。

ここで、n は コモンの数、

F_r は、LCD の更新レート (Hz) です。

LCD タイマ クロックの値が下がると、全体のコントラスト レベル範囲が保証できなくなります。また、最小の LCD タイマ クロック近くで、LCD 更新レートのエラーが発生します。

最大 LCD タイマ クロックは、LCD タイマの分解能によって制限されます。

Note 設計で CapSense ユーザ モジュールを使用する場合は、CapSense ユーザ モジュールは、スキャン速度と分解能を基に、VC1、VC2、VC3 が変更される可能性があることを考慮する必要があります。これらの変更を認識し、LCD クロック値パラメータに適切な値を指定してください。スリープモード操作が必要な場合は、CPU_32K. を選択します。CPU_32K が選択されると、LCDTimer は、32K_Select パラメータ (グローバル設定) が内部の場合は ILO、外部の場合は ECO を使用します。VC1、VC2、VC3 またはクロックの Row の 1 つとともに CapSense を使用する場合、LCD クロック値パラメータにクロックの周波数値を入力します。

*16 HW/SW を持つ SLCD ユーザ モジュールは、VC1 値を持ちません。

LCD Clock Value (LCD クロック値)

LCDTimer のクロック周波数値を提供します。クロック値を kHz で入力します。デフォルト値の 0 に設定されると、ユーザ モジュールがグローバル設定で指定された分周器を基に、選択したクロック値を計算します。LCD クロック源が VC1、VC2、VC3 または CPU_32K でない場合は、LCD クロック値パラメータを手動で設定してください。

設計で CapSense ユーザ モジュールを使用する場合は、CapSense ユーザ モジュール API は、スキャン速度と分解能を基に、VC1、VC2、VC3 値を変更できることを考慮する必要があります。このため、CapSense ユーザ モジュールを使用する場合は、CPU_32K またはローなどのクロック源が推奨されます。

Table 7. 設定

タイプ	範囲	デフォルト	コメント
整数	100 ー 2000 (kHz)	0	LCDTimer のクロック値を設定します。

リフレッシュレート

LCD のリフレッシュレートを設定します。VC1、VC2、VC3 または Row がクロック源である場合は 30-150 Hz、CPU_32K が選択されている場合は 30-60 Hz が適切です。

Note タイマの期間は、LCDTimer 入力クロック、リフレッシュレート、コントラスト レベルを基にして計算されます。

Table 8. 設定

タイプ	範囲	デフォルト	コメント
小数点整数	最小 = 30、最大 = 150	50	クロック源として VC1、VC2、VC3 または Row が選択されている場合の LCDTimer のクロック値を設定します。
小数点整数	最小 = 30、最大 = 60	50	クロック源として CPU_32K が選択されている場合の LCDTimer のクロック値を設定します。

コントラスト レベル

LCD のコントラスト レベルを設定します。

Table 9. 設定

タイプ	範囲	デフォルト	コメント
テキスト	最小、最大、中央	中央	LCD のコントラスト レベルを設定します。

Note アナログ MUX バス ドライブの場合は、コントラストは、LCD フレーム間のデッド時間を設定することで調整されます。

コントラスト レベルの最大値は、以下の式で計算されます。

Equation 6

$$Max = \left(\frac{\frac{1}{Refresh\ Rate} - 500\mu s}{2 \times n} \right) \times Timer\ Input\ Frequency$$

ここで

Max - コントラスト レベルの最大値

Timer Input Frequency - LCD タイマ入力のクロック周波数値

n - コモンの数

コントラスト レベルの最小値は、以下の式で計算されます。

Equation 7

$$Min = Timer\ Input\ Frequency \times 500\mu s$$

ここで

Min - コントラスト レベルの最小値

Timer Input Frequency - LCD タイマ入力のクロック周波数値

Med - コントラスト レベルの中央値

CPU の負荷

LCD は、LCDTimer 割り込みルーチンでリフレッシュされます。LCDTimer ISR は、LCD のリフレッシュレートとコントラストを決定します。さらに、セグメントとコモンのポートのデータを更新します。LCDTimer ISR 期間は、LCD のクロック値、コモンの数、LCD の更新レート、コントラスト レベルによって異なります。

LCDTimer 期間が短すぎると、タイマのオーバーフロー率が上がります。これで、プロセッサの割り込みオーバーヘッドが高まります。

プロセッサのオーバーフローを避けるため、LCDTimer ISR 期間の最小値が 0.5 msec (または、スタンドアロンの GPIO 技法を使用している場合は 0.25 msec) に制限されます。LCDTimer が限度値に達するような値に SLCD ユーザ モジュールのパラメータを設定すると、LCD の更新レートとコントラスト レベルが、ユーザ モジュールのプロパティの値と異なります。

LCDTimer の期間を制限すると、設計規則のチェック警告が表示されます。

アプリケーション プログラミング インタフェース

アプリケーション プログラミング インタフェース (API) 関数は、高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、各機能に対するインタフェースを include ファイルによって提供される関連定数とともに示します。

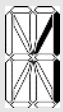
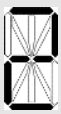
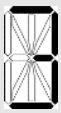
ユーザ モジュールを配置するたびに、インスタンス名が割り当てられます。デフォルトでは、PSoC Designer プロジェクトで、このユーザ モジュールの最初のインスタンスに SLCD_1 を割り当てます。これは識別子の構文ルールに従った一意の値に変更できます。割り当てたインスタンス名が、すべてのグローバル関数名、変数、および定数記号の接頭語になります。次の説明では、簡単にするために、インスタンス名は省略されて単に 「SLCD」となっています。

Note ** ここでは、すべてのユーザー 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される可能性があります。関数を呼び出す場合、呼出し後に A と X の値が必要になる場合は、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。アセンブラ言語のプログラマは、コードがこのポリシーを遵守していることを認識しなければなりません。一部のユーザ モジュール API 関数では A と X は変更されないこともありますが、将来も変更されないという保証はありません。

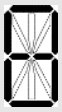
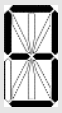
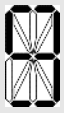
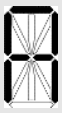
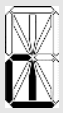
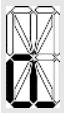
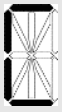
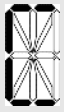
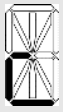
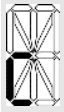
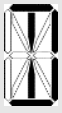
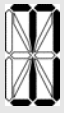
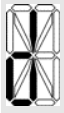
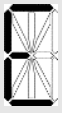
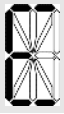
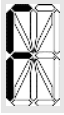
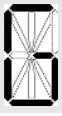
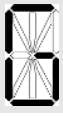
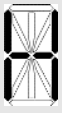
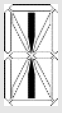
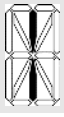
SLCD ユーザ モジュールを初期化、開始、停止するエントリ ポイントが提供されています。すべての場合で、モジュールのインスタンス名が次の入力内容に記載されて SLCD 接頭辞に置き換えられます。間違ったインスタンス名の使用は、構文エラーの一般的な原因です。

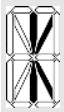
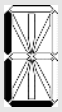
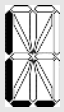
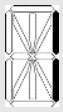
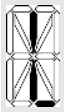
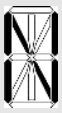
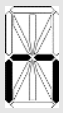
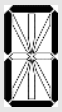
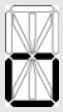
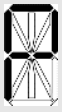
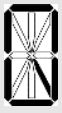
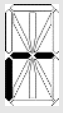
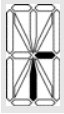
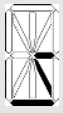
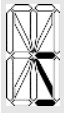
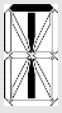
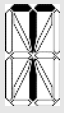
LCD への損傷を避けるため、SLCD_Start() API を呼び出す前に、グローバルな割り込みを有効にしておくことをお勧めします。さらに、グローバル割り込みを無効にする前に、SLCD_Stop() API を呼び出してください。スリープモードを使用し、LCD 操作が必要ない場合は、デバイスをスリープに設定する前に、LCD SLCD_Stop() API が必要になります。

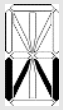
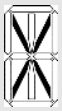
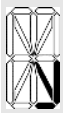
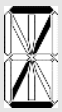
Table 10. SLCD ユーザ モジュールの文字表

文字	14-segment ディスプレイ	16-segment ディスプレイ	文字	14-segment ディスプレイ	16-segment ディスプレイ
0					
1					
2					
3					
4					
5					
6					

[+] Feedback

文字	14-segment ディスプレイ	16-segment ディスプレイ	文字	14-segment ディスプレイ	16-segment ディスプレイ
7					
8					
9					
A			a		
B			b		
C			c		
D			d		
E			e		
F			f		
G			g		
H			h		
I			i		
J			j		

文字	14-segment ディスプレイ	16-segment ディスプレイ	文字	14-segment ディスプレイ	16-segment ディスプレイ
K			k		
L			l		
M			m		
N			n		
O			o		
P			p		
Q			q		
R			r		
S			s		
T			t		
U			u		
V			v		

文字	14-segment ディスプレイ	16-segment ディスプレイ	文字	14-segment ディスプレイ	16-segment ディスプレイ
W			w		
X			x		
Y			y		
Z			z		

SLCD_Start

説明

ユーザ モジュールを有効にします。デバイス エディタで、更新レートとコントラスト レベルを設定できます。

この関数は、以下のタスクを実行します。

- すべてのピンを構成する
- 設定されているリフレッシュレートとコントラスト レベルにより、アクティブな COM 時間とデッド状態のタイマ期間を計算する
- LCDBias RefMux モジュールを開始する (該当する場合)
- LCDTimer 割り込みを有効にする

C プロトタイプ

```
void  SLCD_Start(BYTE InvertState)
```

アセンブラ

```
mov   A, InvertState
lcall  SLCD_Start
```

パラメータ

InvertState - Normal State (標準ディスプレイ) または InvertedState (反転ディスプレイ) を指定します。

記号名	値	説明
NORMAL_STATE	0	標準ディスプレイ
INVERTED_STATE	1	反転ディスプレイ

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。

LCD への損傷を避けるため、SLCD_Start() API を呼び出す前に、グローバルな割り込みを有効にしておくことをお勧めします。さらに、グローバル割り込みを無効にする前に、SLCD_Stop() API を呼び出してください。スリープモードを使用し、LCD 操作が必要ない場合は、デバイスをスリープに設定する前に、LCD SLCD_Stop() API が必要になります。

SLCD_Stop

説明

ユーザ モジュールの機能を停止します。

これは、タイマ モジュールを停止し、すべての LCD ピンを High Z Analog にします。設計に RefMux モジュールがある場合は、そのモジュールの電源をオフにします。

C プロトタイプ

```
void  SLCD_Stop(void)
```

アセンブラ

```
lcall  SLCD_Stop
```

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。

LCD への損傷を避けるため、SLCD_Start() API を呼び出す前に、グローバルな割り込みを有効にしておくことをお勧めします。さらに、グローバル割り込みを無効にする前に、SLCD_Stop() API を呼び出してください。スリープモードを使用し、LCD 操作が必要ない場合は、デバイスをスリープに設定する前に、LCD SLCD_Stop() API が必要になります。

SLCD_ChangeContrast

説明

LCD のコントラスト レベルを変更します。

C プロトタイプ

```
BYTE  SLCD_ChangeContrast(BYTE bOption, BYTE bDelta)
```

アセンブラ

```
mov   A, bOption
mov   X, bDelta
lcall SLCD_ChangeContrast
```

パラメータ

bOption：これは、以下のいずれかになります。

オプション	値	用途
INCREASE_CONTRAST	0	コントラストのレベルを上げる
DECREASE_CONTRAST	1	コントラストのレベルを下げる
SET_TO_MAX	2	コントラストを最大値に設定する
SET_TO_MIN	3	コントラストを最小値に設定する
SET_TO_MED	4	コントラストを中央値に設定する

bDelta：この値は、(アクティブＣＯＭ期間中) タイマの期間値を、どの程度上げるまたは下げる必要があるかにより決定されます。このパラメータは、INCREASE_CONTRAST または DECREASE_CONTRAST がこの API の [Option] (オプション) フィールドに渡されたときのみ有効になります。その他のオプションでは、bDelta フィールドの値は無視されます。bDelta は、API で 1 ー 20 のいずれかの値になり、結果のタイマ値は、この範囲を超えないように制限されます。

戻り値

BYTE 値：この API は、API が最大または最小値に達したかどうかを示すバイト値を返します。

戻り値	説明
0x1	タイマ期間が、最大値に達しました。*
0x2	タイマ期間が、最小値に達しました。*
0x0	タイマ期間は制限内にあります。

*API は、INCREASE_CONTRAST または DECREASE_CONTRAST オプションが渡され、タイマ期間が提供されたデルタ値で、上下限値に達したときにのみ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。

SLCD_GetInterruptStatus

説明

タイマの割り込みステータスを取得し、ユーザ アプリケーションに返します。

C プロトタイプ

```
BYTE  SLCD_GetInterruptStatus(void)
```

アセンブラ

```
lcall  SLCD_GetInterruptStatus
```

パラメータ

なし

戻り値

戻り値のタイプは BYTE です。1 つのフラグがユーザ モジュールによって維持され、LCDTimer 割り込みルーチンで 1 に設定されます。API が呼び出されると、このフラグ値を返します。このフラグは、この API の終わりで自動的にクリアされます。

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。

スリープモードを使用し、LCD 操作が必要ない場合は、デバイスをスリープに設定する前に、LCD SLCD_Stop() API を呼び出してください。

SLCD_ClearAll

説明

LCD の全セグメントをクリアします。

C プロトタイプ

```
void  SLCD_ClearAll(void)
```

アセンブラ

```
lcall  SLCD_ClearAll
```

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。

SLCD_ClearDisplay

説明

LCD の英数字または数字ディスプレイ セクションを完全にクリアします。

LCD に複数の数字または英数字ディスプレイ セクションがある場合は、それぞれのセクションが固有の DisplayID を持ちます。この関数に DisplayID が渡されたセクションのみがクリアされます。

C プロトタイプ

```
void  SLCD_ClearDisplay(BYTE DisplayID)
```

アセンブラ

```
mov   A, DisplayID
lcall  SLCD_ClearDisplay
```

パラメータ

DisplayID - どの LCD のディスプレイ セクション (英数字または数字) をクリアするかを指定します。

SLCD.inc および SLCD.h ファイルは、それぞれの DisplayID で固有の定数定義を行います。

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。

SLCD_PrintDigit

説明

数字ディスプレイまたは英数字ディスプレイの指定した桁位置にある数字を 16 進数で印刷します。

C プロトタイプ

```
void  SLCD_PrintDigit(BYTE DisplayID, BYTE DigitPosition, BYTE Number)
```

アセンブラ

```
mov   A, Number
push  A
mov   A, DigitPosition
push  A
mov   A, DisplayID
push  A
lcall  SLCD_PrintDigit
```

パラメータ

DisplayID - 英数字または数字ディスプレイの特定のセクションを指定します。

DigitPosition - 数字を表示する必要のある数字 / 英数字ディスプレイの桁位置を指定します。0 ー (n-1) の値になります。ここで、n はディスプレイの桁数です。

Number - 表示されるデータ。これは、(0-15) からの値を取ります。

SLCD.inc および SLCD.h ファイルは、それぞれの DisplayID で固有の定数定義を持ちます。

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。

SLCD_ClearDigit

説明

LCD の英数字または数字ディスプレイのセクションで指定した桁をクリアします。

LCD に複数の数字または英数字ディスプレイ セクションがある場合は、それぞれのセクションが固有の DisplayID を持ちます。DisplayID および DigitPosition がこの関数に渡された桁のみがクリアされます。

C プロトタイプ

```
void  SLCD_ClearDigit(BYTE DisplayID, BYTE DigitPosition)
```

アセンブラ

```
mov   A, DisplayID
mov   X, DigitPosition
lcall  SLCD_ClearDigit
```

パラメータ

DisplayID - 英数字または数字ディスプレイの特定のセクションを指定します。

DigitPosition - 数字をクリアする必要のある数字 / 英数字ディスプレイの桁位置を指定します。

SLCD.inc および SLCD.h ファイルは、それぞれの DisplayID で固有の定数定義を持ちます。

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。

SLCD_PrintHexInt

説明

数字ディスプレイまたは英数字ディスプレイで指定した数字を 16 進数で表示します。

C プロトタイプ

```
void  SLCD_PrintHexInt(BYTE DisplayID, int Number)
```

アセンブラ

```
mov   A, >Number
push  A
mov   A, <Number
push  A
mov   A, DisplayID
push  A
lcall  SLCD_PrintHexInt
```

パラメータ

DisplayID - 英数字または数字ディスプレイの特定のセクションを指定します。

Number - 表示されるデータ。0 ー 0xFFFF の値を取ります。指定したディスプレイのセクションは、少なくとも 4 桁を持ちます。数がディスプレイ セクションの容量を超えると、桁 (最上位) が破棄されます。ディスプレイ セクションに 4 桁以上含まれる場合は、最下位の 4 桁を残し、その他の桁は SLCD_ClearDigit API によってクリアされます。

SLCD.inc および SLCD.h ファイルは、それぞれの DisplayID で固有の定数定義を持ちます。

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。

SLCD_PrintHexByte

説明

数字ディスプレイまたは英数字ディスプレイで指定した数字を 16 進数で表示します。

C プロトタイプ

```
void  SLCD_PrintHexByte(BYTE DisplayID, BYTE Number)
```

アセンブラ

```
mov   A, DisplayID
mov   X, Number
lcall  SLCD_PrintHexByte
```

パラメータ

DisplayID - 英数字または数字ディスプレイの特定のセクションを指定します。

Number - 表示されるデータ。0 ～ 0xFF の値を取ります。指定したディスプレイのセクションは、少なくとも 2 桁を持ちます。数がディスプレイ セクションの容量を超えると、桁 (最上位から開始) が破棄されます。数字または英数字が 4 桁以上含む場合は、最下位の 2 桁を残し、その他の桁は SLCD_ClearDigit API によってクリアされます。

SLCD.inc および SLCD.h ファイルは、それぞれの DisplayID で固有の定数定義を持ちます。

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。

SLCD_PrintDecNumber

説明

数字ディスプレイまたは英数字ディスプレイで指定した数字を 10 進法で表示します。

C プロトタイプ

```
void  SLCD_PrintDecNumber(BYTE DisplayID, int Number)
```

アセンブラ

```
mov   A, >Number
push  A
mov   A, <Number
push  A
mov   A, DisplayID
push  A
lcall  SLCD_PrintDecNumber
```

パラメータ

DisplayID - 英数字または数字ディスプレイの特定のセクションを指定します。

Number - 表示されるデータ。0-65535 の値を取ります。指定したディスプレイのセクションは、少なくとも 5 桁を持ちます。数がディスプレイ セクションの容量を超えると、桁 (最上位から開始) が破棄されます。ディスプレイ セクションが 5 桁以上含む場合は、データの最下位 5 桁が表示されます。その他の桁は、SLCD_ClearDigit API によってクリアされます。

SLCD.inc および SLCD.h ファイルは、それぞれの DisplayID で固有の定数定義を持ちます。

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。

SLCD_PrintString

説明

「sRAMString」ポインタによって指定される文字列を、英数字ディスプレイで表示します。文字列の長さは、NumberOfChars で指定されます。この API は、英数字 LCDTimer のみで有効です。

C プロトタイプ

```
void  SLCD_PrintString(BYTE DisplayID, BYTE StartDigitPosition, Char * sRAMString, BYTE
NumberOfChars)
```

アセンブラ

```
mov   A, NumberOfChars
push  A
mov   A, >sRAMString
push  A
mov   A, <sRAMString
push  A
mov   A, StartDigitPosition
push  A
mov   A, DisplayID
push  A
lcall  SLCD_PrintString
```

パラメータ

DisplayID - 英数字ディスプレイの特定のセクションを指定します。

StartDigitPosition - 英数字ディスプレイのどの桁の位置から文字列を印刷するかを指定します。0 ー (n-1) の値になります。ここで、n はディスプレイの桁数です。

*sRAMString : RAM の文字列へのポインタ。

NumberOfChars - 文字列の文字数。

SLCD.inc および SLCD.h ファイルは、それぞれの DisplayID で固有の定数定義を持ちます。

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。

SLCD_PrintCString

説明

「sROMString」ポインタによって英数字ディスプレイで指定される文字列を表示します。文字列の長さは、NumberOfChars で指定されます。この API は、英数字 LCDTimer のみで有効です。

C プロトタイプ

```
void  SLCD_PrintCString(BYTE DisplayID, BYTE StartDigitPosition, Const Char
*sROMString, BYTE NumberOfChars)
```

アセンブラ

```
mov   A, NumberOfChars
push  A
mov   A, >sROMString
push  A
mov   A, <sROMString
push  A
mov   A, StartDigitPosition
push  A
mov   A, DisplayID
push  A
lcall  SLCD_PrintCString
```

パラメータ

DisplayID - 英数字ディスプレイの特定のセクションを指定します。

StartDigitPosition - 英数字ディスプレイのどの桁の位置から文字列を表示するかを指定します。0 ー (n-1) の値になります。ここで、n はディスプレイの桁数です。

*sROMString : ROM の文字列へのポインタ。

NumberOfChars - 文字列の文字数。

SLCD.inc および SLCD.h ファイルは、それぞれの DisplayID で固有の定数定義を持ちます。

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。

SLCD_PrintCharacter

説明

英数字ディスプレイの 「Data」(データ) に保存されている ASCII 値で指定された文字列を表示します。この API は、英数字 LCDTimer のみで有効です。

C プロトタイプ

```
void  SLCD_PrintCharacter(BYTE DisplayID, BYTE DigitPos, char Data)
```

アセンブラ

```
mov   A, Data
push  A
mov   A, DigitPos
push  A
mov   A, DisplayID
push  A
lcall  SLCD_PrintCharacter
```

パラメータ

DisplayID - 英数字ディスプレイの特定のセクションを指定します。

DigitPos - 印刷する文字があるディスプレイの位置を指定します。0 ー (n-1) の値になります。ここで、n はディスプレイの桁数です。

Data - 印刷する文字の ASCII 値を保持します。数字、大文字と小文字のアルファベットになります。

SLCD.inc および SLCD.h ファイルは、それぞれの DisplayID で固有の定数定義を持ちます。

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。

SLCD_SetInvertState

説明

LCD の全セグメントを反転します。

C プロトタイプ

```
void  SLCD_SetInvertState(BYTE InvertState)
```

アセンブラ

```
mov   A, InvertState
lcall  SLCD_SetInvertState
```

パラメータ

InvertState - Normal State (標準ディスプレイ) または InvertedState (反転ディスプレイ) を指定します。

記号名	値	説明
NORMAL_STATE	0	標準ディスプレイ
INVERTED_STATE	1	反転ディスプレイ

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。

SLCD_EnableSymbol

説明

個々のセグメントのオンとオフを切り替えます。

C プロトタイプ

```
void  SLCD_EnableSymbol(BYTE SegmentID, BYTE ON_OFF)
```

アセンブラ

```
mov   A, SegmentID
mov   X, ON_OFF
lcall  SLCD_EnableSymbol
```

パラメータ

SegmentID - 制御する必要のあるセグメントを識別します。

ON_OFF - 1 が渡されると、セグメントがオンになり、そうでない場合オフになります。

SLCD.inc および SLCD.h ファイルは、それぞれの SegmentID で固有の定数定義を持ちます。

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。

ファームウェア ソースコードの例

このアセンブリ コードは、ユーザ モジュールを起動し、LCD ディスプレイで 「1234」という数字を表示します。

```
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

_main:
    ; Enables global interrupt
    M8C_EnableGInt
    ; Strongly recommended to enable global interrupts
    ; before SLCD_Start() API to avoid the LCD damage!!!
    ; Start UM
    mov A, SLCD_NORMAL_STATE
    lcall _SLCD_Start
    ; set the contrast level to maximum
    mov A, SLCD_SET_TO_MAX
    lcall _SLCD_ChangeContrast
loop:
    mov A, 0x12
    push A
    mov A, 0x34
    push A
    mov A, SLCD_DISPLAY_ID_1
    push A
    ; prints the integer on display, specified by DisplayID
    lcall _SLCD_PrintHexInt
    jmp loop
```

C での同じコードは以下のようになります。

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

void main(void)
{
    BYTE DisplayID;
    int Number;
    DisplayID = 0; /*init Display ID */
    Number = 0x1234; /*init Number */
    M8C_EnableGInt; /*enable global interrupt*/
    // Strongly recommended to enable global interrupts
    // before SLCD_Start() API to avoid the LCD damage!!!
    SLCD_Start(SLCD_NORMAL_STATE); /*starts the module*/
    /*sets the contrast level to maximum*/
    SLCD_ChangeContrast(SLCD_SET_TO_MAX, 0);
    while(1)
    {
        /*user code*/
        SLCD_PrintHexInt(DisplayID, Number);
```

```
        /* prints the integer on display, specified by DisplayID */
    }
}
```

コンフィグレーション レジスタ

SLCD ユーザ モジュールは、タイマ デジタル PSoC ブロック 2 つ、アナログ CT ブロック 1 つ、アナログＭＵＸバス リソースを使用します。レジスタのセットを通して各ブロックがパーソナライズされ、パラメータ化されます。以下の表に、定数としての 「パーソナリティ」値と名前が付けられたビットフィールドとしてのパラメータおよび短い説明を示します。これらのレジスタの記号名は、ユーザ モジュール インスタンスの C およびアセンブリ言語インターフェイス ファイル (「.h」および 「.inc」ファイル) で定義されます。

ここでは、パラメータ化された記号のみ説明されています。

LCDTimer ブロック レジスタ

Table 11.　ブロック LCDTimer レジスタ：関数、バンク 1

ブロック / ビット	7	6	5	4	3	2	1	0
MSB	0	0	1	0	0	0	0	0
LSB	0	0	0	0	0	0	0	0

このレジスタは、デジタル ベーシック / 通信タイプ 「B」ブロックの設定を定義し、SLCD ユーザ モジュールの LCDTimer デジタル ブロックになるようにします。

Table 12.　ブロック LCDTimer レジスタ：入力、バンク 1

ブロック / ビット	7	6	5	4	3	2	1	0
MSB	0	0	1	1	LCD クロック			
LSB	0	0	0	1	LCD クロック			

このレジスタは、SLCD ユーザ モジュールの LCDTimer デジタル ブロックのクロック入力を選択するために使用されます。

Table 13.　ブロック LCDTimer レジスタ：出力、バンク 1

ブロック / ビット	7	6	5	4	3	2	1	0
MSB	0	1	0	0	0	0	0	0
LSB	0	1	0	0	0	0	0	0

このレジスタは、LCDTimer デジタル ブロック出力から利用可能な Row インターコネクトへの接続、およびクロックの再同期を制御するために使用されます。

Table 14.　ブロック LCDTimer レジスタ：Count (DR0), Bank 0

ブロック / ビット	7	6	5	4	3	2	1	0
MSB	0	0	0	0	0	0	0	0
LSB	0	0	0	0	0	0	0	0

このレジスタは、SLCD ユーザ モジュールの LCDTimer デジタル ブロックのデータ レジスタ 0 です。

Table 15. ブロック LCDTimer レジスタ：Period (DR1), Bank 0

ブロック / ビット	7	6	5	4	3	2	1	0
MSB	タイマ期間 LCB							
LSB	タイマ期間 LCB							

このレジスタは、SLCD ユーザ モジュールの LCDTimer デジタル ブロックのデータ レジスタ 1 です。

Table 16. ブロック LCDTimer レジスタ：Compare (DR2), Bank 0

ブロック / ビット	7	6	5	4	3	2	1	0
MSB	0	0	0	0	0	0	0	0
LSB	0	0	0	0	0	0	1	0

このレジスタは、SLCD ユーザ モジュールの LCDTimer デジタル ブロックのデータ レジスタ 2 です。

Table 17. ブロック LCDTimer レジスタ：Control (CR0), Bank 0

ブロック / ビット	7	6	5	4	3	2	1	0
MSB	0	1	0	0	0	0	0	0
LSB	0	0	0	0	0	0	0	Start/Stop (開始 / 停止)

このレジスタは、SLCD ユーザ モジュールの LCDTimer デジタル ブロックの制御レジスタ 0 です。

Table 18. ブロック LCDBias、レジスタ：CR0 バンク 0

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	1

Table 19. ブロック LCDBias、レジスタ：CR1 バンク 0

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	1	0	0	1

Table 20. ブロック LCDBias、レジスタ：CR2 バンク 0

ビット	7	6	5	4	3	2	1	0
値	0	0	0	1	0	1	0	0

Table 21. ANALOG_MUX_BUS、レジスタ：MUX_CRx バンク 1

ビット	7	6	5	4	3	2	1	0
値	アナログＭＵＸピン							

このレジスタは、SLCD モジュールのアナログＭＵＸバスです。このレジスタは、アナログＭＵＸバスと該当するピンの接続を制御するために使用されます。

バージョン ヒストリー

バージョン	著者	説明
1.0	DHA	初期バージョン
1.10	DHA	外部バイアス構成の MuxBusNumber ウィザード パラメータのデフォルト値が変更されました。 ASM ファイルから、.LITERAL/.ENDLITERAL ブラケットを削除しました。

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ハイレベルな解説を掲載しています。

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.